Self-Study Report

The Dual Simplex Method

The standard form of a linear programming problem is as follows:

$$\begin{aligned} \text{maximize } z = \ & c^T x \\ subject\ to\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

If the problem, in standard form, is initialised with a basic feasible solution, the optimal solution can be found using the simplex method (phase II procedure). A basic feasible solution, also known as a primal feasible solution, is one where all the right-hand side values of the tableau, b, are greater than or equal to zero. The phase II procedure produces a sequence of primal feasible solutions and terminates once it reaches a solution which is primal feasible and dual feasible. A dual feasible solution is one where all the coefficients of the objective function are non-positive [1]. If the problem, in standard form, is initialised with a dual feasible solution, the optimal solution can be found using the dual simplex method. This method produces a sequence of dual feasible solutions and terminates once it reaches a solution which is dual feasible and primal feasible [1].

The dual simplex method has the following steps:

1. *Rearrange the problem such that it is in standard form (if necessary).*
2. *Add the basic variables, x_B, to the linear system of constraints to make it an equality.*
3. *Put into tableau form and ensure problem is initialised with a dual feasible solution.*
4. *Determine the "leaving" variable, x_L- basic variable with the smallest negative right-hand side value. $x_L \in x_B$*
5. *Determine the "entering" variable, x_E- non-basic variable with the smallest absolute value of the ratio between its objective function coefficient, c_n, and the coefficient of the constraint which intersects with the leaving variable, a_{Ln}.*

   $$x_E \in x_N \qquad \min \left|\frac{c_n}{a_{Ln}}\right| \text{ for } n \in N$$
6. *The "pivot" is the element, of the tableau, at the intersection between the "leaving" and "entering" variables.*
7. *Perform the Jordan exchange.*
8. *If the new tableau is a dual feasible and primal feasible solution, stop! The optimal solution has been found. Otherwise repeat steps 4-7.*

The following example is used to demonstrate the method. Consider the problem:

$$\begin{aligned} \max z = \ & -2x_1 - x_2 \\ subject\ to\ & -3x_1 - x_2 \leq -3 \\ & -4x_1 - 3x_2 \leq -6 \\ & x_1 + 2x_2 \leq 3 \\ & x_1 \geq 0, x_2 \geq 0 \end{aligned}$$

The problem is initiated with a dual solution and is already in standard form. Hence, it was easily put into tableau form as shown by T_0 . The leaving variable was x_4 since -6 was the lowest RHS value. To

determine the entering variable, each of the ratios were calculated. Since $\frac{1}{3}$ was the smallest ratio, x_2 was the entering variable. The pivot element was highlighted in red. The Jordan exchange was then performed, resulting in T_1. This tableau had a dual feasible solution but no primal feasible solution, so the algorithm returned to step 4.

T_0	x_1	x_2	RHS
$x_3 =$	-3	-1	-3
$x_4 =$	-4	-3	-6
$x_5 =$	1	2	3
$z =$	-2	-1	0
Ratio =	$\left\lvert\frac{-2}{-4}\right\rvert = \frac{1}{2}$	$\left\lvert\frac{-1}{-3}\right\rvert = \frac{1}{3}$	

The lowest RHS value was -1; since two variables had this value, the one with the lowest index was chosen as the leaving variable. The entering variable was x_1 since $\frac{2}{5}$ was the lowest ratio. The Jordan exchange was then performed resulting in T_2.

T_1	x_1	x_4	RHS
$x_3 =$	$\frac{-5}{3}$	$\frac{-1}{3}$	-1
$x_2 =$	$\frac{4}{3}$	$\frac{-1}{3}$	2
$x_5 =$	$\frac{-5}{3}$	$\frac{2}{3}$	-1
$z =$	$\frac{-2}{3}$	$\frac{-1}{3}$	2
Ratio =	$\left\lvert\frac{\frac{-2}{3}}{\frac{-5}{3}}\right\rvert = \frac{2}{5}$	$\left\lvert\frac{\frac{-1}{3}}{\frac{-1}{3}}\right\rvert = 1$	

T_2 shows a solution which is dual feasible and primal feasible, so the algorithm terminates. The optimal solution can be read from the tableau as $x_1^* = \frac{3}{5}, x_2^* = \frac{6}{5}, z^* = \frac{12}{5}$.

T_2	x_3	x_4	RHS
$x_1 =$	$\frac{-3}{5}$	$\frac{1}{5}$	$\frac{3}{5}$
$x_2 =$	$\frac{4}{5}$	$\frac{-3}{5}$	$\frac{6}{5}$
$x_5 =$	-1	1	0
$z =$	$\frac{-2}{5}$	$\frac{-1}{5}$	$\frac{12}{5}$

Integer Linear Programming (ILP) Problem

This is a linear programming problem with the added constraint that all variables must be integers [2]. The general form of the problem is laid out as follows:

$$\begin{aligned} \text{maximize } z = \ & c^T x \\ subject\ to\ & Ax \leq b \\ & x \geq 0 \\ & x_i \in \mathbb{Z} \end{aligned}$$

There are many real-world problems that could be formulated as integer linear programs, these include but are not restricted to: scheduling problems, modelling fixed costs and modelling non-linear terms [2]. To motivate the need for integer linear programming, the travelling salesman problem is considered.

The salesman wants to plan a tour of n cities, {0, 1, …, n-1}, starting and ending in his home city, 0. The distance between each city is known and he wants to find the optimal tour which minimises the total distance he travels. To construct the problem, a tour is defined by a list of cities in the order they were visited; $s^k = 0, s_1, s_2, \dots, s_{n-1}$where s^k represents the k^{th} possible tour and s_i represents the i^{th} city visited. The distance between city i and city j is denoted by c_{ij}. A decision variable, x_{ij}, is introduced for defining the objective function.

$$x_{ij} = \begin{cases} 1, & if\ tour\ visits\ city\ j\ immediately\ after\ city\ i \\ 0, & otherwise \end{cases}$$

Hence, the decision variable can only take an integer value. The objective function is defined as:

$$\min \sum_{i,j} c_{ij} x_{ij}$$

To ensure a set of decision variables represent a possible tour, strict constraints must be added. For example, the salesman can only travel to one city at a time and only leave one city at a time. These conditions can be modelled, respectively, by the following equality constraints:

$$\sum_j x_{ij} = 1 \ \ and \ \sum_i x_{ij} = 1$$

Another constraint must be applied to ensure sub-tours (tours not including all n cities) are excluded. This can be done by introducing another parameter, t_{s_i}, which defines the position in the tour a city was visited; $t_{s_i} = i\ for\ i = 0, 1, \dots, n-1$. The following condition ensures no sub tours are possible [2]:

$$t_j \geq t_i + 1 - n\left(1 - x_{ij}\right) for\ i \geq 0, j \geq 1, i \neq j$$

Grouping all these constraints together results in the travelling salesman problem formulated as the integer linear program below. This can be moulded into the standard form using the rules for inequalities, etc.

$$\min \sum_{i,j} c_{ij} x_{ij}$$

$$Subject\ to \sum_{j} x_{ij} = 1$$

$$\sum_{i} x_{ij} = 1$$

$$t_j \geq t_i + 1 - n(1 - x_{ij})\ for\ i \geq 0, j \geq 1, i \neq j$$

$$t_0 = 0, x_{ij} \in \{0,1\}, t_i \in \{0, 1, 2, \dots\}$$

Solving an integer linear program requires finding integer feasible solutions; hence the simplex method cannot be used. Techniques such as cutting planes and branch-and-bound are commonly used for solving these problems.

Branch-and-Bound algorithm

Given an Integer Linear Programming (ILP) problem, of the form discussed in the previous section, the branch-and-bound algorithm is used to find the optimal solution. The algorithm is initiated by ignoring the integer constraint and optimizing the remaining problem known as the LP-relaxation. This is the first LP problem, evaluated by the algorithm, so it is denoted by P_0. The feasible points of the ILP problem fall within the feasible region of the LP-relaxation; hence, the optimal solution of the LP-relaxation provides an upper bound for the ILP problem. The solution of P_0 can be found using the simplex method. This and its corresponding objective function value should be stored.

Given the optimal solution of P_0, the elements of the solution vector must be evaluated. Variables with integer values will not be considered as branching variables. Out of the remaining variables, one must be used as the branching variable. A simple method to choose is selecting the one with the lowest index. The branching variable is denoted by x_k. This variable is used to divide P_0 into two sub-problems- P_1 and P_2. These sub-problems inherit the problem P_0, but each add an additional (different) constraint. In the optimal solution to the ILP problem, x_k will take an integer value which satisfies one of the conditions: $x_k \leq \alpha$ or $x_k \geq \beta$ where α is the integer value immediately below x_k and β is the integer value immediately above x_k. One of these constraints will be added to P_1 and the other to P_2.

The P_1 problem is evaluated next. Its optimal solution and its corresponding objective function value are found and stored. If the solution is a feasible point, the objective function value is identified as the "best so far". There is no dividing this problem into sub-problems since the solutions found, by adding more constraints to the problem, won't improve. If the solution is not a feasible point, the same branching process as above is repeated. A tree of LP sub-problems is starting to grow. This tree is referred to as the enumerate tree. An example of a two variable problem is depicted as an enumerate tree below. In figure 1, the second box around P_1 indicates the optimal solution along that branch has been found. Thus, in the context of trees, that node is referred to as a leaf.

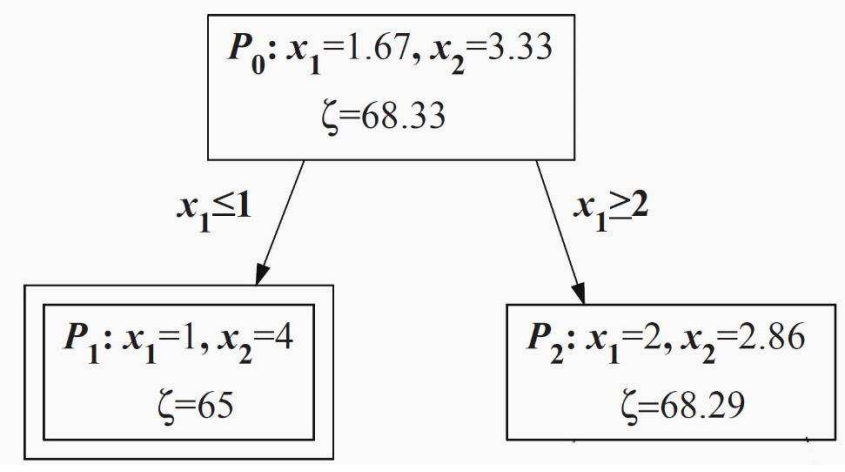

Figure 1: Example enumerate tree after solving P_1 and P_2 [2]

Each problem will be divided into a sub-problem unless a feasible solution is found. If the feasible solution is an improvement on the "best so far", this new solution will be referred to as the "best so far". Once the algorithm arrives at a leaf of the tree, it must move back up the structure until it hits a solved problem with an unsolved problem below. There are many situations when the algorithm can

choose between many sub-problems to solve next. In order to methodically navigate the tree, the depth-first search is used. This was used over the breadth-first search as it had been observed that integer solutions commonly lie deep in the tree. By finding feasible points earlier in the search, computation can be saved by reducing the depth of search along a branch due to the current "best so far" value being better than that found higher up the branch. The algorithm, finally, terminates when each branch of the tree ends on a leaf. The optimal solution to the ILP problem is the "best so far" solution at the end. Figure 2 illustrates the full enumerate tree of the example shown in figure 1. The order the tree was traversed is indicated by the ascending P_i index. Each branch is terminated at a leaf. The optimal solution is the leaf with the highest objective function value. In this example, it would be 68 from P_8.

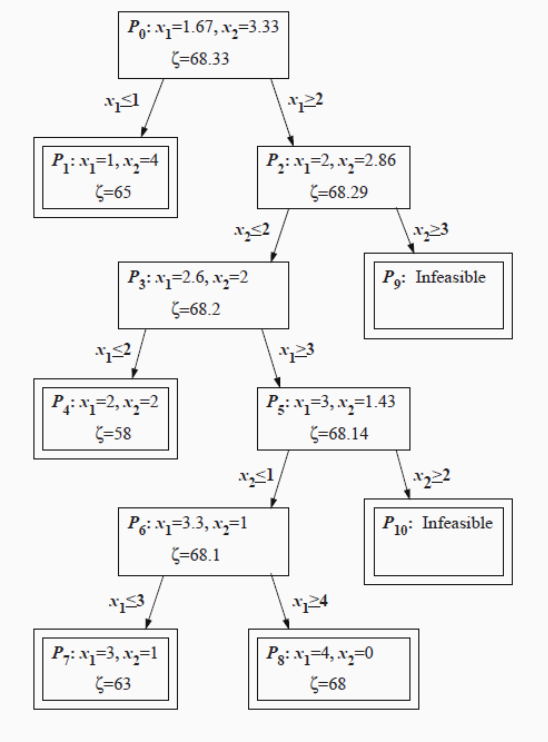

Figure 2: Full enumerate tree [2]

Staff Scheduling Problem

Formulation of model

Two different model formulations were considered and compared to find the optimal solution- one which minimised the labour cost, met the employees target to ensure good customer service and didn't overwork the staff. Each model considered different shift patterns; hence the general form of each model was the same, along with the meaning of the parameters and variables. The format of an Integer Linear Programming (ILP) problem was used for each:

$$\begin{aligned} \text{maximize } z = \ & c^T x \\ subject\ to\ & Ax \leq b \\ & x \geq 0 \\ & x_i \in \mathbb{Z} \end{aligned}$$

The parameters of the two models were as follows:
c_j = *cost of employee on shift j,* $c_j \in \mathbb{R}$
b_i = *number of employees needed for time slot i,* $b_i \in \mathbb{Z}$
a_{ij} = *is slot i part of shift j?* $a_{ij} \in \{0, 1\}$ *where 0 represents no and 1 represents yes*

The variables of the two models were:
x_j = *number of employees on shift j,* $x_j \in \mathbb{Z}$
z = *daily labour cost,* $z \in \mathbb{R}$

Model 1- Lunch hour taken in middle of shift

Full-time employees were paid for 7-hour shifts, which included a 1-hour lunch break. Thus, the daily cost of a full-time employee was £84. In this model, the lunch break was taken exactly in the middle of their shift. Therefore, possible shift patterns for full-time employees were:

$$j = \begin{cases} 1, & 9-16\ with\ lunch\ taken\ at\ 12-13 \\ 2, & 10-17\ with\ lunch\ taken\ at\ 13-14 \end{cases}$$

Part-time employees were paid for 4-hour shifts, with no lunch break. Hence, the daily cost of a part-time employee was £30. Possible shift patterns for part-time employees were:

$$j = \begin{cases} 3, & 9-13 \\ 4, & 10-14 \\ 5, & 11-15 \\ 6, & 12-16 \\ 7, & 13-17 \end{cases}$$

Model 2- Maximum of 5 consecutive working hours

Again, full time employees were paid for 7-hour shifts, which included a 1-hour lunch break. Therefore, the daily cost of a full-time employee was £84. In this model, the lunch break was taken anytime during the day providing the employee didn't work more than 5 consecutive hours. This gave greater flexibility to the shift pattern of a full-time employee, meaning there was more possible shift patterns:

$$j = \begin{cases} 1, & 9-16\ with\ lunch\ taken\ at\ 10-11 \\ 2, & 9-16\ with\ lunch\ taken\ at\ 11-12 \\ 3, & 9-16\ with\ lunch\ taken\ at\ 12-13 \\ 4, & 9-16\ with\ lunch\ taken\ at\ 13-14 \\ 5, & 9-16\ with\ lunch\ taken\ at\ 14-15 \\ 6, & 10-17\ with\ lunch\ taken\ at\ 11-12 \\ 7, & 10-17\ with\ lunch\ taken\ at\ 12-13 \\ 8, & 10-17\ with\ lunch\ taken\ at\ 13-14 \\ 9, & 10-17\ with\ lunch\ taken\ at\ 14-15 \\ 10, & 10-17\ with\ lunch\ taken\ at\ 15-16 \end{cases}$$

Part-time employees were paid for 4-hour shifts, with no lunch break. Hence, the daily cost of a part-time employee was £30. Possible shift patterns for part-time employees were:

$$j = \begin{cases} 3, & 9-13 \\ 4, & 10-14 \\ 5, & 11-15 \\ 6, & 12-16 \\ 7, & 13-17 \end{cases}$$

Implementation

The programming language chosen to model and solve the problem was Matlab. The *intlinprog* function from the *optimization* package was used. This required 8 parameters to be defined. With respect to the standard ILP format above, the parameters included *A*, *b* and *c*. Separate parameters were included for equality conditions which saved converting them into the standard form. In addition, the index of the variables, which the integer constraint applied to, needed to be defined- in this case, it was all of them. Finally, upper and lower bounds of the solution space were required- the lower bound was zero, to ensure only positive integer solutions; the upper bound was infinity, although this could have been used to set a threshold for the maximum number of staff allowed. Running this function returned the solution (optimal number of employees, on each shift) and its associated minimal labour cost. In addition, the total number of hours worked was calculated. This was used to analyse overstaffing issues.

Results

Neither model was able to find a solution when equality constraints were used for the number of employees required for each time slot. To make these constraints less rigorous, they were changed to inequalities which ensured the stated number of employees per time slot was a minimum. The results of each model are presented in table 1.

Model	Minimal labour cost (£)	Total hours worked	Hours worked per hours required
Model 1	636.00	64	1.23
Model 2	576.00	56	1.08

Table 1: Comparison of each model's optimal solution

The table shows that the optimal solution of model 2 saved £60 per day on labour costs compared to model 1. This was achieved by allowing employees to take lunch breaks at various time slots

throughout the day. Model 2 had a total of 4 full-time employees and 8 part-time employees; whereas model 1 had a total of 4 full-time employees and 10 part-time employees. The flexibility of lunch break times reduced the number of part-time staff by 2 and the total number of daily hours worked by 8. The required total hours worked in a day was 52. Thus, model 2 only exceeded this by 8% (4 hours); in comparison, model 1 surpassed this by 23% (12 hours). Since the minimal number of full-time staff was used in both solutions, these extra hours were made up by part-time staff. This was equivalent to 3 shifts in model 1and 1 shift in model 2. Therefore, overstaffing was more of an issue for model 1.

Conclusion

Out of the models considered, model 2 should be implemented since it produced the lowest labour cost. Shifts were designed with lunch breaks taken at different points throughout the day, ensuring full-time employees didn't work more than 5 consecutive hours. The optimal solution of this model had only 4 more hours worked than the total required to ensure good customer service. If this level of overstaffing was unsatisfactory, other models could be investigated. One such model would be to increase the hours worked by full-time staff from 6 to 7. The full-time staff would then be at the cafeteria for the full working day, with one hour for lunch included. This could reduce the minimum number of full-time staff required for each day from one to two, whilst maintaining at least one member of full-time staff is working each time slot.

References

[1] S. J. W. Jorge Nocedal, *Numerical Optimization*. New York: Springer, 2006.
[2] R. J. Vanderbei, *Linear programming*. Springer, 2015.

Appendix- Matlab Code

```
%using equalities
% f = [84 84 30 30 30 30 30]';
% intcon = 1:7;
% A = [-1 -1 0 0 0 0 0];
% b = -4;
% Aeq = [1 0 1 0 0 0 0
%        1 1 1 1 0 0 0
%        1 1 1 1 1 0 0
%        0 1 1 1 1 1 0
%        1 0 0 1 1 1 1
%        1 1 0 0 1 1 1
%        1 1 0 0 0 1 1
%        0 1 0 0 0 0 1];
%
%    beq = [6 5 7 8 8 7 5 6]';
% lb = zeros(7,1);
% ub = inf(7,1);

%replacing equalities with inequalities
f = [84 84 30 30 30 30 30]';
intcon = 1:7;
A = [-1 0 -1 0 0 0 0
     -1 -1 -1 -1 0 0 0
     -1 -1 -1 -1 -1 0 0
      0 -1 -1 -1 -1 -1 0
     -1 0 0 -1 -1 -1 -1
     -1 -1 0 0 -1 -1 -1
     -1 -1 0 0 0 -1 -1
      0 -1 0 0 0 0 -1
     -1 -1 0 0 0 0 0];

b = [-6 -5 -7 -8 -8 -7 -5 -6 -4]';
Aeq = zeros(1,7);
beq = 0;
lb = zeros(7,1);
ub = inf(7,1);

x = intlinprog(f, intcon, A, b, Aeq, beq, lb, ub)

total_hours_worked = transpose(x)*[6 6 4 4 4 4 4]'
```

throughout the day. Model 2 had a total of 4 full-time employees and 8 part-time employees; whereas model 1 had a total of 4 full-time employees and 10 part-time employees. The flexibility of lunch break times reduced the number of part-time staff by 2 and the total number of daily hours worked by 8. The required total hours worked in a day was 52. Thus, model 2 only exceeded this by 8% (4 hours); in comparison, model 1 surpassed this by 23% (12 hours). Since the minimal number of full-time staff was used in both solutions, these extra hours were made up by part-time staff. This was equivalent to 3 shifts in model 1and 1 shift in model 2. Therefore, overstaffing was more of an issue for model 1.

Conclusion

Out of the models considered, model 2 should be implemented since it produced the lowest labour cost. Shifts were designed with lunch breaks taken at different points throughout the day, ensuring full-time employees didn't work more than 5 consecutive hours. The optimal solution of this model had only 4 more hours worked than the total required to ensure good customer service. If this level of overstaffing was unsatisfactory, other models could be investigated. One such model would be to increase the hours worked by full-time staff from 6 to 7. The full-time staff would then be at the cafeteria for the full working day, with one hour for lunch included. This could reduce the minimum number of full-time staff required for each day from one to two, whilst maintaining at least one member of full-time staff is working each time slot.

References

[1] S. J. W. Jorge Nocedal, *Numerical Optimization*. New York: Springer, 2006.
[2] R. J. Vanderbei, *Linear programming*. Springer, 2015.