Edexcel GCE Decision Mathematics (D2)

Required Knowledge Information Sheet

Daniel Hammocks

Transportation Problems

- In solving transportation problems you need to know:
  - The supply or stock
  - The demand
  - The unit cost of transporting goods
- When total supply > total demand, the problem is **unbalanced**.
- A dummy destination is needed if total supply does not equal the total demand. Transport costs to this dummy destination are zero.
- To solve the transportation problem, the **North-West corner method** is used.
- The North-West corner method:
  - Create a table, with one row for every source and one column for every destination. Each destinations demand is given at the foot of each column and each sources stock is given at the end of each row. Enter numbers in each cell to show how many units are to be sent along that route.
  - Begin with the top left hand corner. Allocate the maximum available quantity to meet the demand at this destination (but do not exceed the stock of the source).
  - As each stock is emptied, move one square down and allocate as many units as possible from the next source until the demand of the destination is met.
  - As each demand is met, move one square to the right and again allocate as many units as possible.
  - Stop when all the stock is assigned and all the demands are met.
- In a feasible solution to a transportation problem with *m rows* and *n columns*, if the number of cells used is less than $n + m - 1$ then the solution is **degenerate**.
- The algorithm requires n + m – 1 cells to be used in every solution, so a zero must be placed in an unused cell in a degenerate solution.
- To find an improved solution you need to:
  - Use the non-empty cells to find the **shadow costs**
  - Use the shadow costs and the empty cells to find the **improvement indices**
  - Use the improvement indices and the **stepping-stones method** to find an improved solution

- Transportation costs are made up of two components, one associated with the source and one with destination. The costs of using a route are called **shadow costs**. Shadow costs are worked out by:
    - Starting with the north-west corner, set the cost linked with its source to zero.
    - Move along the row to any other non-empty squares and find any other destination costs the same way.
    - When all possible destination costs for that row have been established, go to the start of the next row.
    - Move along this row to any non-empty squares and use the destination costs found earlier, to establish the source cost for the row. Once that has been done, find any further unknown destination costs.
    - Repeat steps three and four until all source and destination costs have been found.
- The **improvement index** of a route is the reduction in cost which would be made by sending one unit along that route.
    - The improvement index in sending a unit from source P to a demand point Q is found by subtracting the source cost S(P) and destination cost D(Q) from the stated cost of transporting one unit along that route C(PQ)
        - Improvement Index for Route PQ = I_{PQ} = C (PQ) – S (P) – D (Q)
    - The route with the most negative improvement index will be introduced into the solution.
    - The cell corresponding to the value with the most negative improvement index becomes the **entering cell** and the route it replaces is referred to as the **exiting cell**.
    - If there are two equal potential entering cells you may choose either. Similarly, if there are two equal exiting cells you may select either.
    - If there are no negative improvement indices the solution is **optimal**.
- The stepping-stone method is used to find out an improved solution. The stepping stone method consists of three parts:
    - Create the cycle of adjustments. The two basic rules are:
        - Within any row and any column there can only be one increasing cell and one decreasing cell
        - Apart from the entering cell, adjustments are only made to non-empty cells
    - Once the cycle of adjustments has been found you transfer the maximum number of units through this cycle. This will be equal to the smallest number in the decreasing cells (since you may not have negative units being transported).
    - You then just adjust the solution to incorporate this improvement.

Allocation (Assignment) Problems

- To reduce a cost matrix:
  - Subtract the least value in each row from each element of that row.
  - Using the new matrix, subtract the least value in each column from each element in that column.
- The Hungarian algorithm:
  - Find the reduced cost matrix
  - Find the minimum number of straight lines (horizontal or vertical) which will cover all of the zeros in the matrix
  - In an *n x n* matrix, if you cannot cover all of the zeros in fewer than n lines, you have an optimal solution and you stop.
  - In an n x n matrix, if you can cover the zeros in fewer than n lines, the solution can be improved.
  - Draw in the lines and look for the smallest uncovered element, *e*.
  - Add *e* to the elements in each covered row and each covered column, adding it twice to any element covered twice.
  - Subtract *e* from every element in the matrix.
  - Repeat procedure until an optimal solution is found.
- As a short cut, after drawing in the lines:
  - Increase each element covered by two lines by *e*
  - Leave each element covered by one line unchanged
  - Subtract *e* from each uncovered element
- If the problem is not n x n you add a dummy row or column of zeros and proceed as for an n x n matrix
- If the data is incomplete, add a large value in the matrix at an appropriate place. Use at least double the largest entry.
- To find **maximums** begin by making all the numbers in the table **negative**. Subtract the most negative number from each element then proceed as above.
- To formulate allocation problems as linear programming problems, use binary coding, with 1 representing allocating the person to a task and 0 representing not allocating the person to a task.
- Present an allocation problem as a linear programming problem by:
  - Defining decision variables
  - Writing down the objective function
  - Writing down the constraints

The Travelling Salesman Problem

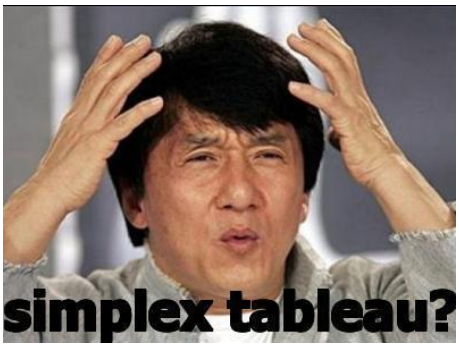
- A walk in a network is a finite sequence of edges such that the end vertex of one edge is the start vertex of the next.
- A walk which visits every vertex, returning to its starting vertex, is called a tour.
- The **optimal solution** is found between the **best upper bound** and **best lower bound**.
- In the **classical travelling salesman problem** you must visit each vertex **only once** before returning to the start
- In the **practical travelling salesman problem** you must visit each vertex **at least once** before returning to the start.
- If you convert a network into a complete network of least distances, the practical and classical travelling salesman problems are the same.
- To create a complete network of least distances, you ensure that the **triangle inequality** holds for all triangles in the network:
    - The longest side ≤ the sum of the two shorter sides
- Use the minimum spanning tree method to find an upper bound:
    - Find the minimum spanning tree to guarantee that each vertex is included
    - Double this minimum connector so that completing the cycle is guaranteed
    - Seek shortcuts (use some of the non-included arcs to enable you to bypass a repeat of some of the minimum spanning tree)
- Aim to make the upper bound as low as possible to reduce the interval in which the optimal solution is contained.
- Use the minimum spanning tree method to find a lower bound:
    - Remove each vertex in turn, together with its arcs
    - Find the residual minimum spanning tree (RMST) and its length
    - Add to the RMST the 'cost' of reconnecting the deleted vertex by the two shortest distinct, arcs and note the totals
    - The greater of these totals is used for the lower bound
- The nearest neighbour algorithm can also be used to find an upper bound:
    - Select each vertex in turn as a starting point
    - Go to the nearest vertex which has not yet been visited
    - Repeat until all vertices have been visited and then return to the start vertex using the shortest route
    - Once all the vertices have been used as the starting vertex, select the tour with the smallest length as the upper bound.

Further Linear Programming

- To formulate a linear programming problem:
  - Define your decision variables
  - Write down the objective function
  - Write down the constraints
- Inequalities can be transformed into equations using **slack variables**.
- The **simplex algorithm** allows you to:
  - Determine if a particular vertex, on the edge of the feasible region, is optimal
  - Decide which adjacent vertex you should move to in order to increase the value of the objective function
- To use a simplex tableau to solve a maximising linear programming problem, where the constraints are given as inequalities:
  - Create the initial tableau
  - Look along the objective row for the most negative entry, this indicates the pivot column
  - Calculate the Θ values for each of the constraint rows, where
    - Θ = (the term in the value column) / (the term in the pivot column)
  - Select the row with the smallest, positive, Θ value to become the pivot row
  - The element in the pivot column and the pivot row is the pivot
  - Divide the pivot row by the pivot, and change the basic variable at the start of the row to the variable at the top of the pivot column, this is now the pivot row
  - Use the pivot row to eliminate the pivots variable from the other rows (the pivot column now has 1 and zeros)
  - Repeat until there are no more negative numbers in the objective row
  - The tableau is optimal and the non-zero values can be read off using the basic variable column and the value column
- When integer solutions are needed, test points around the optimal solution to find which fit the constraints and give a maximum for the objective function

Basic variable	x	y	z	r	s	t	Value	Row operations
x	1	0	$\frac{16}{7}$	$\frac{2}{7}$	$\frac{3}{7}$	0	$\frac{32}{7}$	R1 ÷ $\frac{7}{2}$
y	0	1	$\frac{6}{7}$	$-\frac{1}{7}$	$\frac{2}{7}$	0	$\frac{12}{7}$	R2 − $\frac{1}{2}$R1
t	0	0	$-\frac{13}{7}$	$\frac{1}{7}$	$-\frac{2}{7}$	1	$\frac{30}{7}$	R3 + $\frac{1}{2}$R1
P	0	0	$\frac{107}{7}$	$\frac{2}{7}$	$\frac{17}{7}$	0	$\frac{144}{7}$	R4 + R1

An Example of a Simplex Tableau



How I Feel Completing Simplex Tableaux

Daniel Hammocks

Game Theory

- A two person game is one in which only two parties can play
- When playing safe each player looks for the worst that could happen if she/he makes each choice in turn. She/he then picks the choice that results in the least worst option
- A zero-sum game is one in which the two entries in each cell all add up to zero.
- A pay-off matrix is always written from the row player's point of view unless you are told otherwise.
- The play safe strategies are:
    - The row maximin for the row player
    - The column minimax for the column player
- In a zero-sum game there will be a stable solution if and only if
    - *the row maximin = the column minimax*
- If one named row (or column) is *always* a better option than another named row (or column) then the better option **dominates**.
- The pay-off matrix can be reduced by deleting a dominated row or column
- To determine an optimal strategy, form probability equations and graph them.
- The intersection point gives you the information you need to work out the probabilities and the value of the game
- If there is more than one intersection point the probability giving the highest minimum point gives the solution
- For a player with three choices:
    - Use dominance to reduce the pay-off matrix
    - If you cannot do this, use linear programming
- To use simplex:
    - Transform the game by adding n to each element to make all values > 0
    - Define the decision variables
    - Write down the objective function
    - Write down the constraints

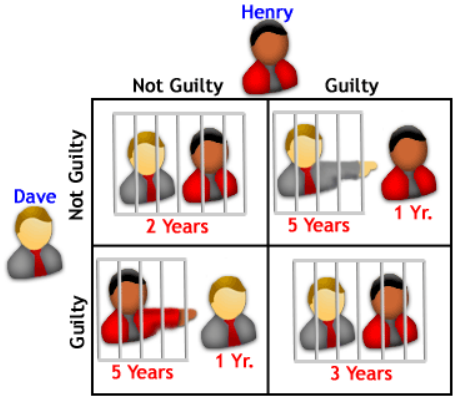
An Example Where Game Theory Can Be Used

Prisoners Dave and Henry are held in separate rooms and cannot communicate

They are both suspected of a crime

They can either plead guilty or not guilty

Payoffs are years in prison



Daniel Hammocks

Network Flows

- In **capacitated directed networks**, (**capacitated directed graphs** or **capacitated digraphs**) each are has a weight which represents the **capacity** of that arc.
- The **feasibility condition** says that the flow along an arc must not exceed the capacity of that arc.
- The **conservation condition** (on all but source and sink vertices) says that:
    - Total flow into a vertex = Total flow out of the vertex
- If an arc contains a flow equal to its capacity it is **saturated**
- A **cut** is a set of arcs whose removal separates the network into two parts X and Y, where X contains at least the source and Y contains at least the sink.
- The capacity (value) of a cut is the sum of the capacities of those arcs in the cut which are directed from X to Y
- When evaluating the capacity of a cut, only include the capacities of the arcs flowing **into** the cut
- The value of a cut is calculated using **capacities**
- In the labelling procedure, two arrows are drawn on each arc:
    - The forward arrow indicated any spare capacity
    - The backward arrow identifies by how much the flow in the arc could be reduced
- Each flow-augmenting route starts at S and finishes at T
- Forward and backward arrows can both be used as long as there is capacity in the direction in which you wish to move
- When sending an extra flow of value f along a flow-augmenting route, you:
    - Subtract f from each 'forward' arrow on the route
    - Add f to each 'backward' arrow on the route
- Sometimes it is necessary to make use of a backflow when finding a flow-augmenting route
- The maximum flow-minimum cut theorem states:
    - In a network the value of the maximum flow = the value of the minimum cut
- If you can find a cut, with capacity equal to the flow, then the flow is maximal
- The minimum cut passes through:
    - Saturated arcs if directed into the cut
    - Empty arcs if directed out of the cut
- In networks with more than one source and/or sink, you create a supersource and/or supersink together with arcs of appropriate capacity and then proceed as normal
- The arcs leading from the supersource to each source must have total capacity equal to the capacity leaving each source.
- The arcs leading from each of the sinks to the supersink must have total capacity and total flow equal to that arriving at each sink.

Dynamic Programming

- A **stage** is a move along the path from the initial state to the final state
- A **state** is the vertex being considered
- An **action** is a directed arc from one state to the next. In selecting an arc you consider what happens if you do that action.
- A **destination** is the vertex you arrive at having taken an action
- A **value** is the sum of the weights on the arcs used in a sequence of actions
- Bellman's principle of optimality states:
    - Any part of an optimal path is itself optimal
- In **dynamic programming** you work backwards from the final destination, T, in a series of stages, to the initial vertex, S
- At each stage you choose the optimal route. The current optimal paths are developed using the paths found in the stage before
- You can use dynamic programming to solve maximum and minimum problems presented in network form
- A **minimax** problem is one in which the longest leg is as short as possible
- A **maximin** problem is one in which the shortest leg is as long as possible
- You can use dynamic programming to solve minimax and maximin problems presented in network and table form.