LINEAR PROGRAMMING

IE-535

PROJECT REPORT

Name: Karthik Sajeev

PUID: 0030028666

Model: 22 (Lumber company)

LP Model:

The lumber company has three sources and five markets. Rail and ship are two alternative modes of transporting the wood, and we need to find the optimal shipping plan such that the overall cost is reduced, while ensuring all the demand is met and sticking to the investment budget set by the company.

The unit costs of transporting by rail are:

	1	**2**	**3**	**4**	**5**
1	61	72	45	55	66
2	69	78	60	49	56
3	59	66	63	61	47

For ships, there are two components to the cost:

Unit shipping cost:

	1	**2**	**3**	**4**	**5**
1	31	38	24		35
2	36	43	28	24	31
3		33	36	32	26

Unit investment cost:

	1	**2**	**3**	**4**	**5**
1	275	303	238		285
2	293	318	270	250	265
3		283	275	268	240

Equivalent uniform annual cost of the ships may be calculated using the formula:

Unit shipping cost + 0.1 * Unit investment cost.

The values obtained are:

	1	**2**	**3**	**4**	**5**
1	58.5	68.3	47.8		63.5
2	65.3	74.8	55	49	57.5
3		61.3	63.5	58.8	50

Now, in order to derive the cost vector, we need to examine each Source-market pair and identify whether rail or ship provides the cheaper mode of transport.

After doing so, we can arrive at the optimal cost vector which we will use in the code as:

	1	**2**	**3**	**4**	**5**
1	58.5	68.3	45	55	63.5
2	65.3	74.8	55	49	56
3	59	61.3	63	58.8	47

Now, the linear program may be written as follows:

$$\min \sum_{i=0}^{3} \quad \sum_{j=0}^{5} \quad c_{ij}.x_{ij}$$

where x_{ij} denotes the number of units of wood (in million board feet) transported from source i to market j, and c_{ij} represents the corresponding cost vector.

s.t

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} \le 10$$

$$x_{21} + x_{22} + x_{23} + x_{24} + x_{25} \le 20$$

$$x_{31} + x_{32} + x_{33} + x_{34} + x_{35} \le 15$$

$$x_{11} + x_{21} + x_{31} \ge 7$$

$$x_{12} + x_{22} + x_{32} \ge 11$$

$$x_{13} + x_{23} + x_{33} \ge 9$$

$$x_{14} + x_{24} + x_{34} \ge 10$$

$$x_{15} + x_{25} + x_{35} \ge 8$$

$$x_{ij} \ge 0, \text{ for all i and j}$$

Also, to keep within budget, we will have to include an additional constraint related to the investment costs of ships:

$$275\, x_{11} + 303\, x_{12} + 285\, x_{15} + 293\, x_{21} + 318\, x_{22} + 270\, x_{23} + 283\, x_{32} + 268\, x_{34} \le 6750$$

We can see that total supply = total demand. So we can use equalities for all the constraints except the last.

Also for convenience x_{11}, x_{12}, .., x_{35} have been taken as x_1, x_2, …, x_{15} in the code.

Using this formulation, a general LP solver has been coded in Matlab, which can be used to solve any program just by giving the coefficients and constraints. The code is shown below:

CODE:

```
%INPUTS
%Values are entered as per the formulation
%This is the ONLY section of the code that needs to be altered for
%different problems
%signs:
% -1 stands for <=
%  0 stands for =
%  1 stands for >=
A=[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0;
   0,0,0,0,0,1,1,1,1,1,0,0,0,0,0;
   0,0,0,0,0,0,0,0,0,0,1,1,1,1,1;
   1,0,0,0,0,1,0,0,0,0,1,0,0,0,0;
   0,1,0,0,0,0,1,0,0,0,0,1,0,0,0;
   0,0,1,0,0,0,0,1,0,0,0,0,1,0,0;
   0,0,0,1,0,0,0,0,1,0,0,0,0,1,0;
   0,0,0,0,1,0,0,0,0,1,0,0,0,0,1];
b=[10;20;15;7;11;9;10;8];
c=[58.5,68.3,45,55,63.5,65.3,74.8,55,49,56,59,61.3,63,58.8,47];
signs=[0,0,0,0,0,0,0,0];

%Calculation of parameters
n_c=size(A,1);                 %Number of constraints
n_v=size(A,2);                 %Number of variables

%Checking for negative b values
for i=1:n_c
    if b(i)<0                  %If negative b is found, multiply by
(-1)
        b(i)=b(i)*(-1);
        signs(i)=signs(i)*(-1);
        A(i,:)=A(i,:)*(-1);
    end
end

%Converting to standard form by adding slack variables
for i=1:n_c
    if signs(i)~=0
        n_v=n_v+1;
        c(n_v)=0;
        if signs(i)==-1
            A(i,n_v)=1;
        else
            A(i,n_v)=-1;
        end
    end
end

%Checking if A is full row rank
if rank (A) < n_c
```

```
    disp('Not full row rank');      %This is just for the purpose of
output
else                                %Redundancy is handled later on in
the code
    disp('Full row rank');
end

%Checking presence of identity in A
Atrans=A';
h=1;
%Recalculating n_c and n_v
n_c=size(A,1);
n_v=size(A,2);
B_size=n_c;                       %%Size of basis
test1=eye(B_size);                %Creating a sample identity of the
same size
B_ind=zeros(1,B_size);            %%Indices of the basis
for i=1:B_size
    test=ismember(Atrans(:,1:B_size),test1(i,:),'rows');
    sum_matches=0;                  %Checking col-by-col by first
transposing
    for j=1:n_v
        sum_matches=sum_matches+test(j);
    end
    if sum_matches>1
        disp('Redundant rows detected!');          %Identical rows
found
    elseif sum_matches==1
        for j=1:n_v
            if test(j)==1
                B_ind(i)=j;
                h=h+1;
            end
        end
    end
end

%Adding artificial variables
g=1;
A_ind=[];
art_vars=0;                 %Number of artificial variables
for i=1:B_size
    if B_ind(i)==0
        n_v=n_v+1;
        c(n_v)=0;           %Updating A,c and indices of A and B
        A(i,n_v)=1;
        B_ind(i)=n_v;
        A_ind(g)=n_v;
        g=g+1;
        art_vars=art_vars+1;
    end
end
```

```
%Computing indices of N
g=1;
N_size_cols=n_v-B_size;         %No. of columns in N
N_ind=zeros(1,(N_size_cols));
for i=1:n_v
    f=0;
    for j=1:B_size
        if i==B_ind(j)
            f=1;
        end
    end
    if f==0
        N_ind(g)=i;
        g=g+1;
    end
end
original_c=c;
B=zeros(B_size);
N=zeros(n_c,N_size_cols);
cB=zeros(1,B_size);             %Cost vectors cB and cN
cN=zeros(1,N_size_cols);
end_iter=0;
count=1;
goto_ph1=1;
direct_simplex=0;
A_rows=size(A,1);

if art_vars==0      %If no artificial variables are present, we can
directly proceed to simplex method
    disp('No artificial variables. Feasible. Proceed directly to usual
Simplex');
    goto_ph1=0;
    goto_ph2=1;
    direct_simplex=1;
else               %Otherwise, we have to perform 2-phase
    disp('Artificial variables exist. Check feasibilty. Proceed to
two-phase method');
end

if goto_ph1==1
%Phase-1 starts; calculating initial values
    disp('PHASE-1');
    z=0;
    c=zeros(1,n_v);
    red_cost=zeros(1,n_v);
    %Cost vector and initial reduced costs
    for i=1:art_vars
        c(A_ind(i))=1;
    end
    red_cost=red_cost-c;
    %Performing elementary row operations
```

```
    for i=1:B_size
        f=0;
        for j=1:art_vars
            if B_ind(i)==A_ind(j)
                f=1;
            end
        end
        if f==1
            red_cost=red_cost+A(i,:);
            z=z+b(i);
        end
    end

%Usual simplex
phase1_enter=1;
while end_iter == 0
    fprintf('Iteration %d: \n', count);
    count=count+1;
    %Calculating B,N,cB and cN based on indices
    for i=1:B_size
        B(:,i)=A(:,B_ind(i));
        cB(i)=c(B_ind(i));
    end
    for i=1:N_size_cols
        N(:,i)=A(:,N_ind(i));
        cN(i)=c(N_ind(i));
    end
    %xB
    xB=inv(B)*b;

    if phase1_enter==0
        %Optimal value
        z=cB * xB;
        %w
        w=cB * inv(B);
        %(zj-cj) values
        red_cost=zeros(1,n_v);
        for i=1:n_v
            f=0;
            for j=1:N_size_cols
                if i==N_ind(j)
                    f=1;
                end
            end
            if f==1
                red_cost(i)=w*A(:,i)-c(i);
            end
        end
    end

    phase1_enter=0;
    %Finding maximum reduced cost and hence the entering variable
```

```
    if max(red_cost)>0
        [M,I]=max(red_cost);
        k=I;
        fprintf('x%d enters\n', k);

    % Computing yk values
    yk=inv(B)*A(:,k);
    ratios=ones(n_c,1);
    ratios=ratios*Inf;
    if max(yk)>0                        %if any yk is non-negative
        for i=1:B_size
            if yk(i)>0
                ratios(i)=xB(i)/yk(i);            %fill ratios matrix
            end
        end
        %Procedure for implementing Bland's rule to avoid cycling in
case
        %of multiple candidates in the min. ratio test
        temp=find(ratios==min(ratios));
        lowest_x_ind=[];
        for i=1:length(temp)
            lowest_x_ind(i)=B_ind(temp(i));
        end
        [M,I]=min(lowest_x_ind);
        I=temp(I);
        r=B_ind(I);                         %index corresponding to lowest
ratio
        fprintf('x%d leaves \n', r);
        % Computing new basic and non-basic indices
        B_ind(find(B_ind == r))=k;
        N_ind(find(N_ind == k))=r;
    else
        end_iter=1;
        g=1;
        %If all yk are non-positive
        disp('Unbounded!');
        disp('Ray with vertex:');
        for i=1:n_v
            f=0;
            for j=1:N_size_cols
                if i==N_ind(j)
                    f=1;
                end
            end
            if f==1
                fprintf('x%d = 0\n', i);
            else
                fprintf('x%d = %f \n', B_ind(g), xB(g));
                g=g+1;
            end
        end
        %Printing out the direction in case of unboundedness
```

```
        disp('and direction:');
        g=1;
        for i=1:n_v
            f=0;
            for j=1:N_size_cols
                if i==N_ind(j)
                    f=1;
                end
            end
            if f==1
                if i==k
                    fprintf('d%d = 1\n', i);
                else
                    fprintf('d%d = 0\n', i);
                end
            else
                fprintf('d%d = %f \n', B_ind(g), -yk(g));
                g=g+1;
            end
        end
    end

    else
        %If all reduced costs are non-positive
        end_iter=1;
        g=1;
        disp('Stop, Optimality reached');
        fprintf('Optimal value of Phase 1 is %f \n', z);
    end
end

if z==0
    %Check for presence of art vars. in the basis
    art_in_basis=0;
    for i=1:art_vars
        for j=1:B_size
            temp=find(A_ind(i)==B_ind(j));
            if temp>0
                art_in_basis=art_in_basis+1;
            end
        end
    end

    %Cheching for redundancy
    %If there is even one artificial variable in the basis..
    while art_in_basis>0
        %Finding the row with artificial var.
        for i=1:B_size
            temp=find(A_ind==B_ind(i));
            if temp>0
                art_row=i;
                break
```

```
            end
        end

        %Checking row corresponding to the artificial variable
        temp=inv(B)*N;
        g=1;
        temp1=[];
        for j=1:length(N_ind)
            f=0;
            if temp(art_row,j)~=0    %If there is a non-zero element
corresponding to a nonbasic var,pivot and update
                for i=1:art_vars
                    if A_ind(i)==N_ind(j)
                        f=1;
                    end
                end
                if f==0
                    temp1(g)=N_ind(j);
                    g=g+1;
                end
            end
        end

        %If a column to pivot is found
        if length(temp1)>0
            k=min(temp1);
            fprintf('x%d enters\n', k);
            r=B_ind(art_row);            %Update the tableau
            fprintf('x%d leaves\n', r);
            B_ind(find(B_ind == r))=k;
            N_ind(find(N_ind == k))=r;

            for i=1:B_size
                B(:,i)=A(:,B_ind(i));
                cB(i)=c(B_ind(i));
            end
            for i=1:N_size_cols
                N(:,i)=A(:,N_ind(i));
                cN(i)=c(N_ind(i));
            end
            %xB values
            xB=inv(B)*b;
            %Optimal value
            z=cB * xB

            art_in_basis=art_in_basis-1;

        else
            %All zeros in the corresponding row. Eliminate artificial
variable row directly
            r=B_ind(art_row);
            fprintf('x%d is eliminated\n', r);
```

```
            N_size_cols=N_size_cols+1;              %Updating tableau
            N_ind(N_size_cols)=B_ind(art_row);
            B_ind(art_row)=[];
            B_size=length(B_ind);
            A_rows=B_size;
            n_c=B_size;
            A(art_row,:)=[];
            b(art_row)=[];
            B=zeros(B_size);
            N=zeros(n_c,N_size_cols);

            for i=1:B_size
                B(:,i)=A(:,B_ind(i));
                cB(i)=c(B_ind(i));
            end

            for i=1:N_size_cols
                N(:,i)=A(:,N_ind(i));
                cN(i)=c(N_ind(i));
            end

            %xB values
            xB=inv(B)*b;

            art_in_basis=art_in_basis-1;
        end
    end

    disp('Proceed to Phase 2');
    goto_ph2=1;
else
    %If at the end of phase-1, z* is not zero, then we have
infeasibility
    disp('Original LP is not feasible');
    goto_ph2=0;
end
end

if goto_ph2==1
    if direct_simplex==0
        %Phase-2 starts
        disp('PHASE 2');
        c=original_c;
        A=zeros(A_rows,size(A,2));
        g=1;
        %Finding columns corresponding to B of A
        for i=1:B_size
            A(g,B_ind(i))=1;
            g=g+1;
        end
        %Finding columns corresponding to N of A
```

```
    temp=inv(B)*N;
    for i=1:N_size_cols
        A(:,N_ind(i))=temp(:,i);
    end
    %Removing columns corresponding to artificial variables
    dec_A_ind=sort(A_ind,'descend');
    for i=1:art_vars
        A(:,dec_A_ind(i))=[];
        c(dec_A_ind(i))=[];
        temp=find(N_ind==dec_A_ind(i));
        if temp>0
            N_ind(temp)=[];
        end
        n_v=n_v-1;
    end
    %New reduced cost
    red_cost=zeros(1,length(c));
    red_cost=red_cost-c;

    %New parameters
    B_size=length(B_ind);
    N_size_cols=length(N_ind);
    B=zeros(B_size);
    N=zeros(n_c,N_size_cols);
    cB=zeros(1,B_size);
    cN=zeros(1,N_size_cols);
    b=xB;

    for i=1:B_size
        B(:,i)=A(:,B_ind(i));
        cB(i)=c(B_ind(i));
    end
    for i=1:N_size_cols
        N(:,i)=A(:,N_ind(i));
        cN(i)=c(N_ind(i));
    end
    %Calculating new reduced cost using elementary operations
    for i=1:B_size
        red_cost=red_cost+(cB(i)*A(i,:));
        z=z+(cB(i)*xB(i));
    end
end

%Usual simplex
end_iter=0;
count=1;
while end_iter == 0
    if direct_simplex==1
        fprintf('Iteration %d: \n', count);
        count=count+1;
        for i=1:B_size
            B(:,i)=A(:,B_ind(i));
```

```
            cB(i)=c(B_ind(i));
        end
        for i=1:N_size_cols
            N(:,i)=A(:,N_ind(i));
            cN(i)=c(N_ind(i));
        end
        %xB values
        xB=inv(B)*b;

        %Optimal value
        z=cB * xB;
        %w
        w=cB * inv(B);
        %(zj-cj) values
        red_cost=zeros(1,n_v);
        for i=1:n_v
            f=0;
            for j=1:N_size_cols
                if i==N_ind(j)
                    f=1;
                end
            end
            if f==1
                red_cost(i)=w*A(:,i)-c(i);
            end
        end
    end

    direct_simplex=1;
    %Finding maximum reduced cost and hence the entering variable
    if max(red_cost)>0
        [M,I]=max(red_cost);
        k=I;
        fprintf('x%d enters\n', k);

        % Computing yk values
        yk=inv(B)*A(:,k);
        ratios=ones(n_c,1);
        ratios=ratios*Inf;
        if max(yk)>0                        %if any yk is non-negative
            for i=1:B_size
                if yk(i)>0
                    ratios(i)=xB(i)/yk(i);
                end
            end
            %Again, using Bland's rule
            temp=find(ratios==min(ratios));
            lowest_x_ind=[];
            for i=1:length(temp)
                lowest_x_ind(i)=B_ind(temp(i));
            end
            [M,I]=min(lowest_x_ind);
```

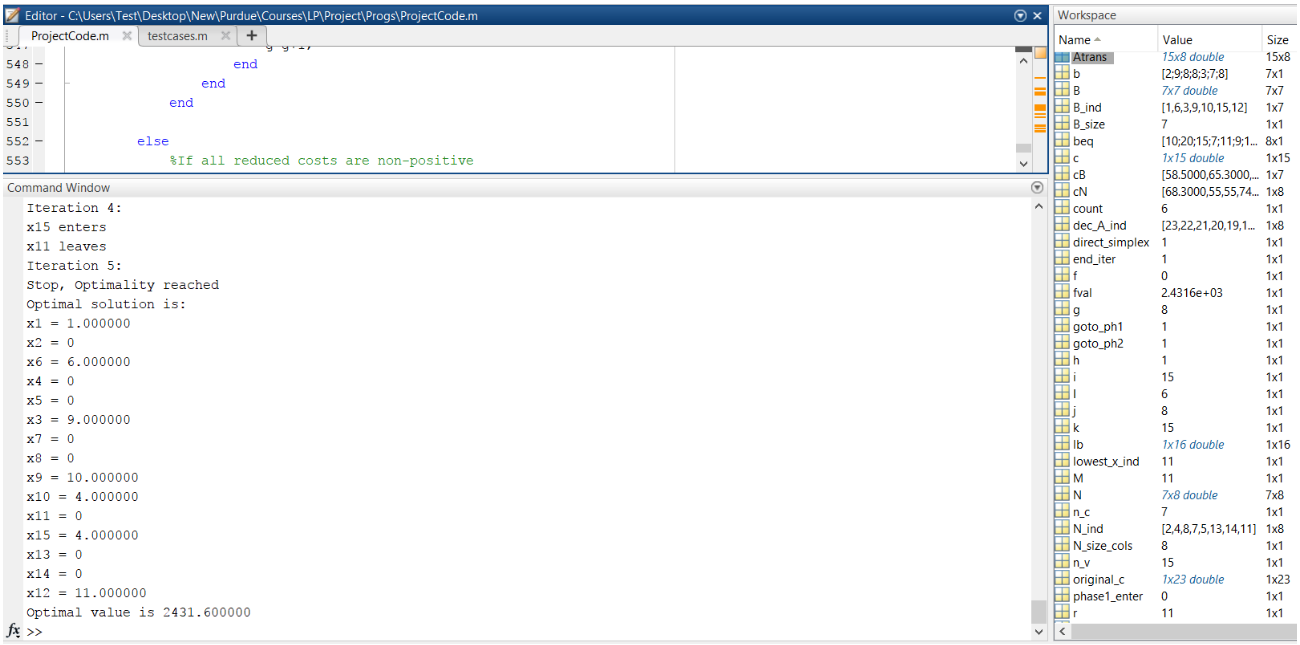
```
                I=temp(I);
                r=B_ind(I);                                %index corresponding
to lowest ratio
                fprintf('x%d leaves \n', r);
                % Computing new basic and non-basic indices
                B_ind(find(B_ind == r))=k;
                N_ind(find(N_ind == k))=r;
            else
                %In case all yk are non-positive
                end_iter=1;
                g=1;
                disp('Unbounded!');
                disp('Ray with vertex:');
                for i=1:n_v
                    f=0;
                    for j=1:N_size_cols
                        if i==N_ind(j)
                            f=1;
                        end
                    end
                    if f==1
                        fprintf('x%d = 0\n', i);
                    else
                        fprintf('x%d = %f \n', B_ind(g), xB(g));
                        g=g+1;
                    end
                end
                disp('and direction:');
                g=1;                               %Printing out the directions
                for i=1:n_v
                    f=0;
                    for j=1:N_size_cols
                        if i==N_ind(j)
                            f=1;
                        end
                    end
                    if f==1
                        if i==k
                            fprintf('d%d = 1\n', i);
                        else
                            fprintf('d%d = 0\n', i);
                        end
                    else
                        fprintf('d%d = %f \n', B_ind(g), -yk(g));
                        g=g+1;
                    end
                end
            end

        else
            %If all reduced costs are non-positive
            end_iter=1;
```

```
            g=1;
            disp('Stop, Optimality reached');
            disp('Optimal solution is: ');
            for i=1:n_v          %Printing out the optimal solution
                f=0;
                for j=1:N_size_cols
                    if i==N_ind(j)
                        f=1;
                    end
                end
                if f==1
                    fprintf('x%d = 0\n', i);
                else
                    fprintf('x%d = %f \n', B_ind(g), xB(g));
                    g=g+1;
                end
            end
            fprintf('Optimal value is %f \n', z);
        end
    end
end
```

OUTPUT:

The screenshot is attached below:



For the sake of clarity, the whole output copied from Matlab is pasted below:

>> ProjectCode

Not full row rank

Artificial variables exist. Check feasibilty. Proceed to two-phase method

PHASE-1

Iteration 1:

x1 enters

x19 leaves

Iteration 2:

x2 enters

x16 leaves

Iteration 3:

x6 enters

x1 leaves

Iteration 4:

x7 enters

x20 leaves

Iteration 5:

x3 enters

x21 leaves

Iteration 6:

x4 enters

x2 leaves

Iteration 7:

x8 enters

x17 leaves

Iteration 8:

x11 enters

x3 leaves

Iteration 9:

x9 enters

x6 leaves

Iteration 10:

x12 enters

x22 leaves

Iteration 11:

x5 enters

x18 leaves

Iteration 12:

Stop, Optimality reached

Optimal value of Phase 1 is 0.000000

x23 is eliminated

Proceed to Phase 2

PHASE 2

x1 enters

x4 leaves

Iteration 1:

x10 enters

x7 leaves

Iteration 2:

x3 enters

x5 leaves

Iteration 3:

x6 enters

x8 leaves

Iteration 4:

x15 enters

x11 leaves

Iteration 5:

Stop, Optimality reached

Optimal solution is:

x1 = 1.000000

x2 = 0

x6 = 6.000000

x4 = 0

x5 = 0

x3 = 9.000000

x7 = 0

x8 = 0

x9 = 10.000000

x10 = 4.000000

x11 = 0

x15 = 4.000000

x13 = 0

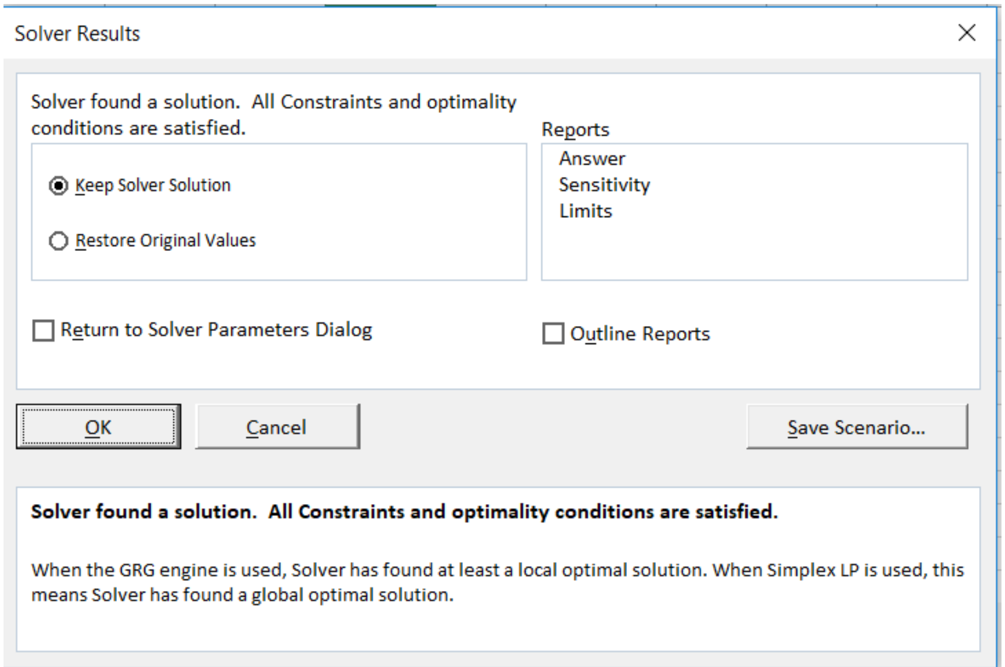
x14 = 0

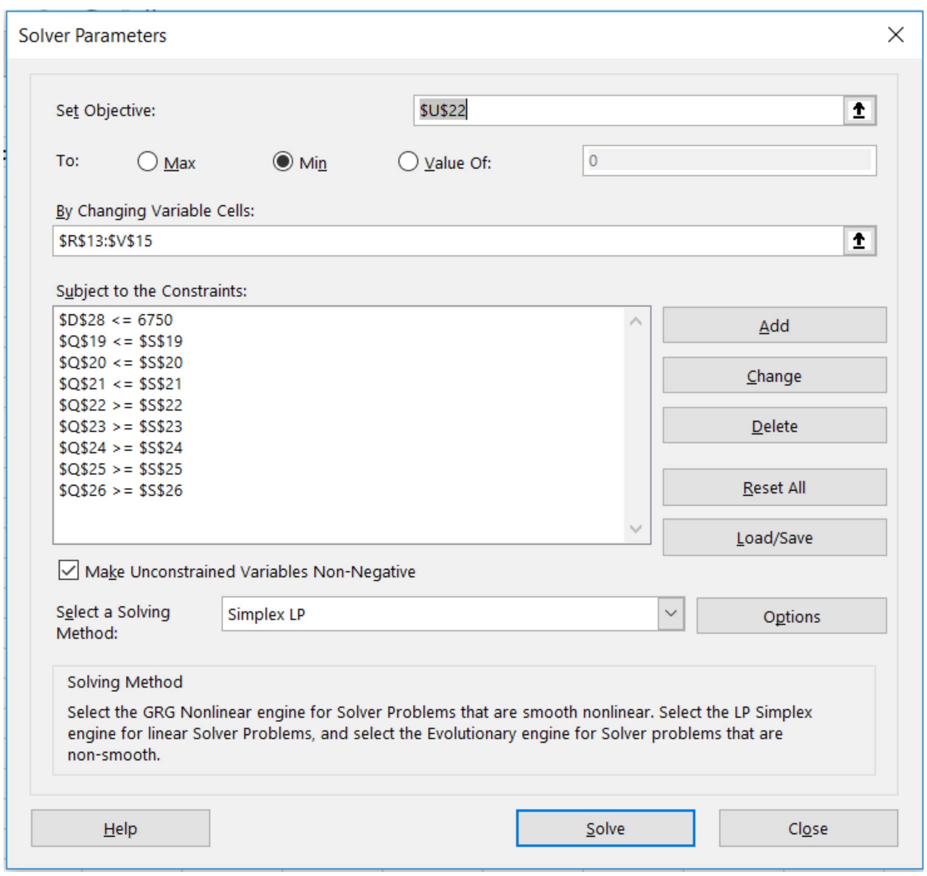
x12 = 11.000000

Optimal value is 2431.600000

COMMERCIAL SOLVER:

The same LP has been solved using MS Excel Solver , the results of which are sttached below:





A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	Unit cost by rail market								**Unit cost by ship market**								**Investment for ship market**				
	1	**2**	**3**	**4**	**5**				**1**	**2**	**3**	**4**	**5**				**1**	**2**	**3**	**4**	**5**
1	61	72	45	55	66			**1**	31	38	24		35			**1**	275	303	238		285
2	69	78	60	49	56			**2**	36	43	28	24	31			**2**	293	318	270	250	265
3	59	66	63	61	47			**3**		33	36	32	26			**3**		283	275	268	240
	Total cost for ship market								**Final cost matrix (c)**								**Solution matrix**				
	1	**2**	**3**	**4**	**5**				**1**	**2**	**3**	**4**	**5**				**1**	**2**	**3**	**4**	**5**
1	58.5	68.3	47.8		63.5			**1**	58.5	68.3	45	55	63.5			**1**	1	0	9	0	0
2	65.3	74.8	55	49	57.5			**2**	65.3	74.8	55	49	56			**2**	6	0	0	10	4
3		61.3	63.5	58.8	50			**3**	59	61.3	63	58.8	47			**3**	0	11	0	0	4
							Constraint matrix									**Sum**		**b**			
	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	10		10			
	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	20		20		**z**	
	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	15		15			
	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	7		7		2431.6	
	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	11		11			
	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	9		9			
	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	10		10			
	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	8		8			
	Total Investment		5146																		

Microsoft Excel 16.0 Answer Report

Worksheet: [Book1.xlsx]Original problem

Report Created: 12/13/2017 10:27:16 PM

Result: Solver found a solution. All Constraints and optimality conditions are satisfied.

Solver Engine

Engine: Simplex LP

Solution Time: 0.047 Seconds.

Iterations: 20 Subproblems: 0

Solver Options

Max Time Unlimited, Iterations Unlimited, Precision 0.000001, Use Automatic Scaling

Max Subproblems Unlimited, Max Integer Sols Unlimited, Integer Tolerance 1%, Assume NonNegative

Objective Cell (Min)

Cell	Name	Original Value	Final Value
$U$22	z	2431.6	2431.6

Variable Cells

Cell	Name	Original Value	Final Value	Integer
$R$13		1	1	Contin
$S$13	Solution matrix	0	0	Contin
$T$13	Investment for ship market	9	9	Contin
$U$13		0	0	Contin
$V$13		0	0	Contin
$R$14		6	6	Contin
$S$14	Solution matrix	0	0	Contin
$T$14	Investment for ship market	0	0	Contin
$U$14		10	10	Contin
$V$14		4	4	Contin
$R$15		0	0	Contin
$S$15	Solution matrix	11	11	Contin
$T$15	Investment for ship market	0	0	Contin
$U$15		0	0	Contin
$V$15		4	4	Contin

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$D$28	Total Investment	5146	$D$28<=6750	Not Binding	1604
$Q$19	Sum	10	$Q$19<=$S$19	Binding	0
$Q$20	Sum	20	$Q$20<=$S$20	Binding	0
$Q$21	Sum	15	$Q$21<=$S$21	Binding	0
$Q$22	Sum	7	$Q$22>=$S$22	Binding	0
$Q$23	Sum	11	$Q$23>=$S$23	Binding	0
$Q$24	Sum	9	$Q$24>=$S$24	Binding	0

CONCLUSION:

As can be seen, the optimal objective function value obtained from both is the same:

$z^* = 2431.6$

$x_{11} = 1$

$x_{13} = 9$

$x_{21} = 6$

$x_{24} = 10$

$x_{25} = 4$

$x_{32} = 11$

$x_{35} = 4$

Remaining x_{ij} are zeros.