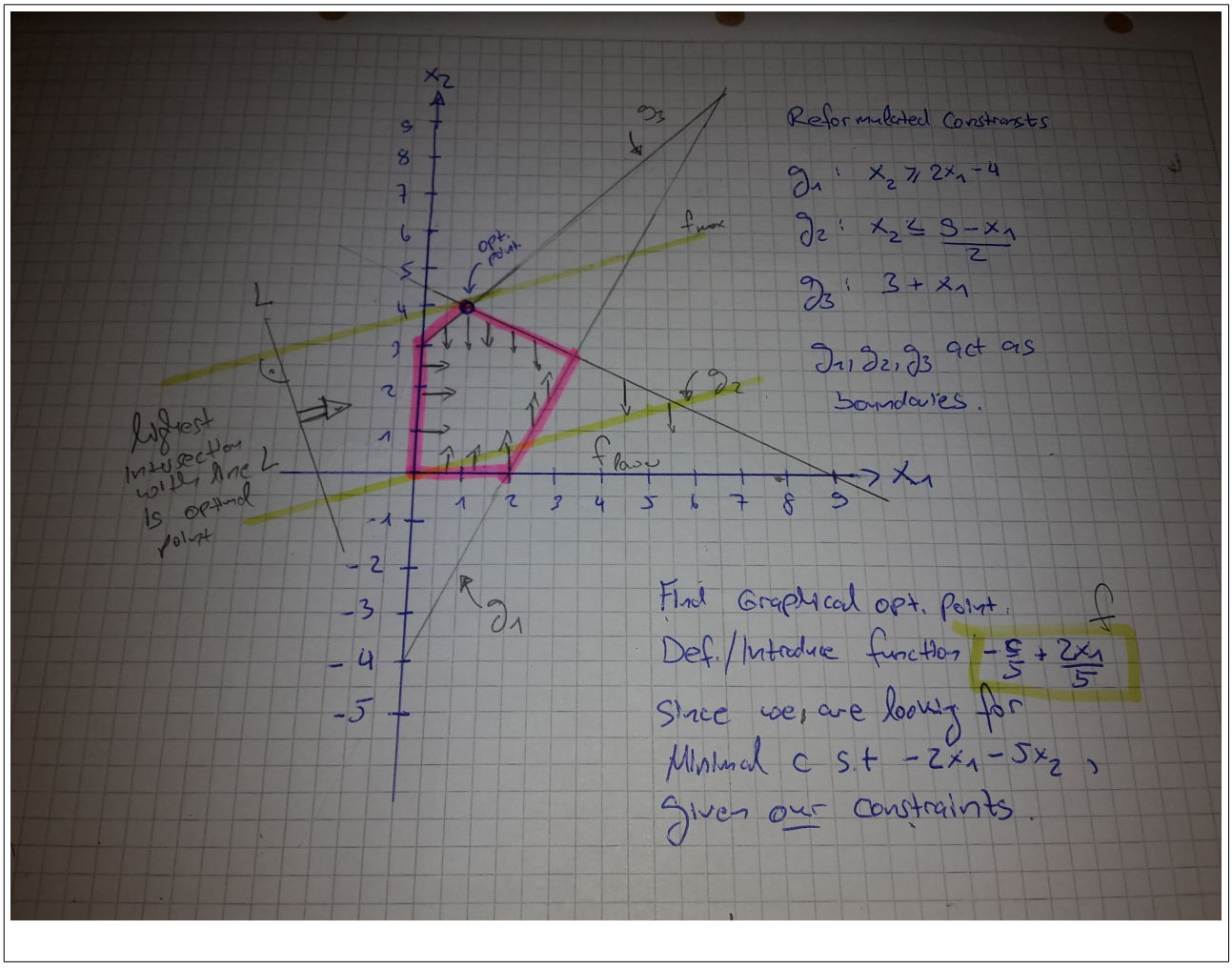
Problem-Sheet 5
My Solution

Michael Single
08-917-445
msingle@students.unibe.ch

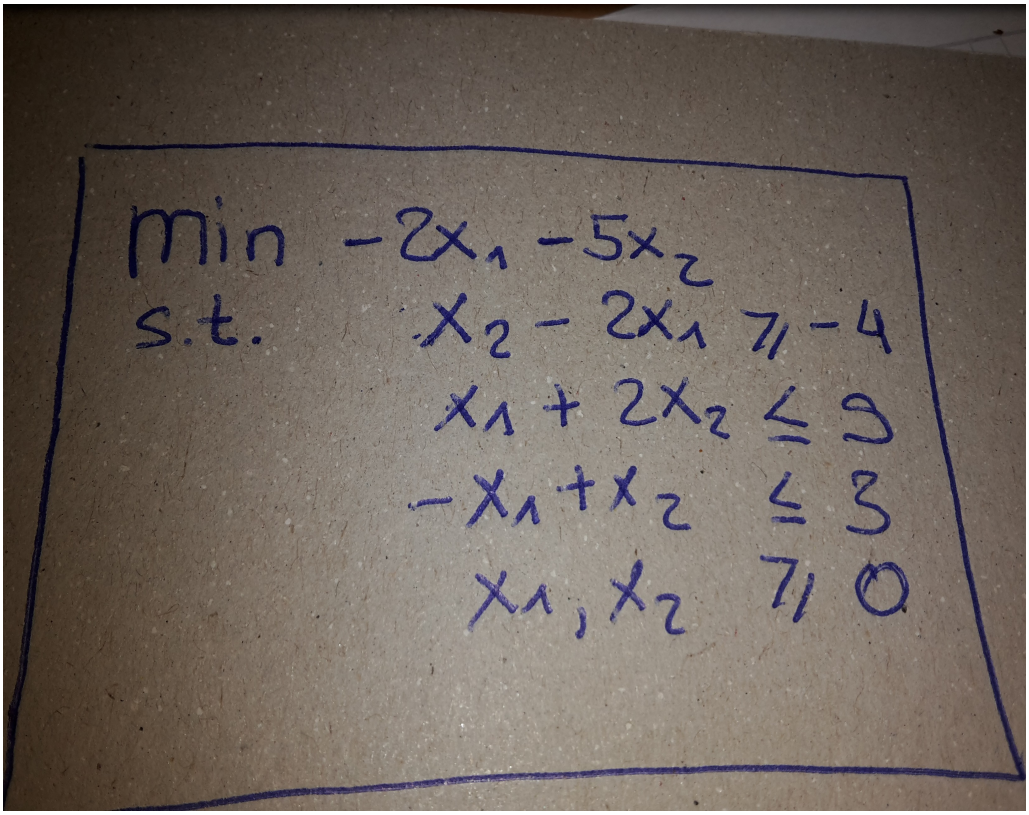
Task1:

a)

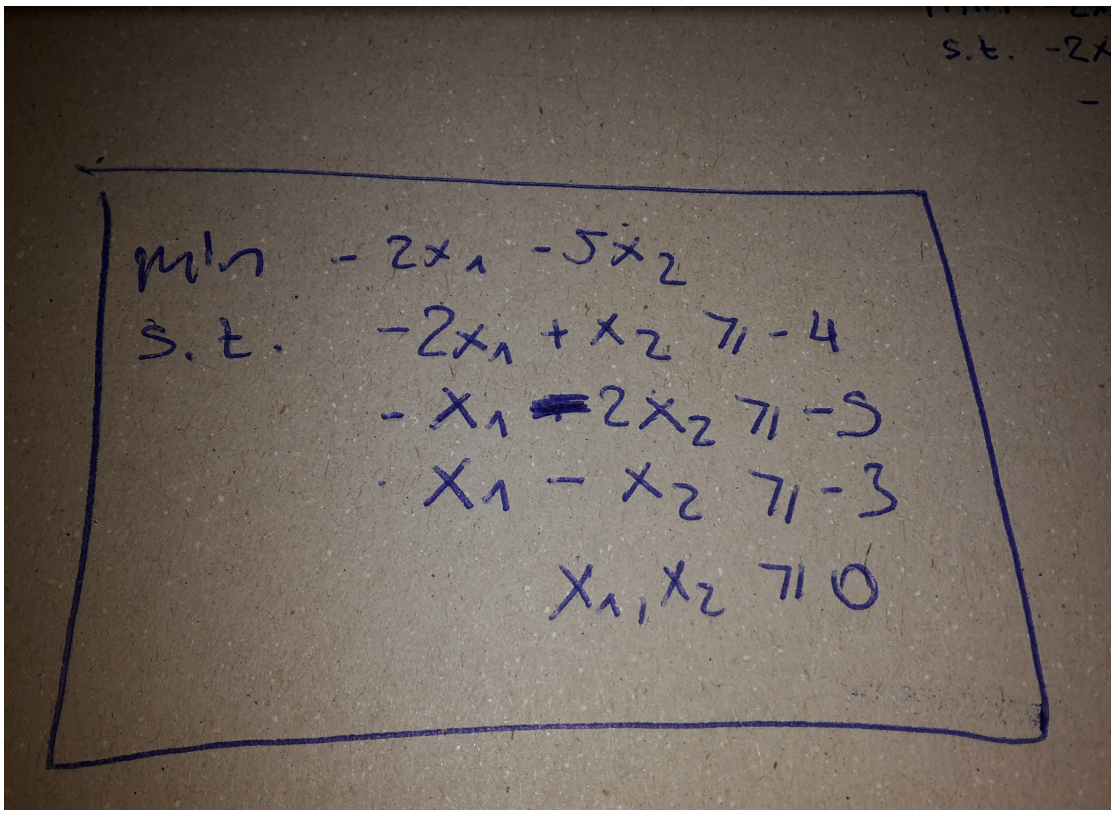


b)

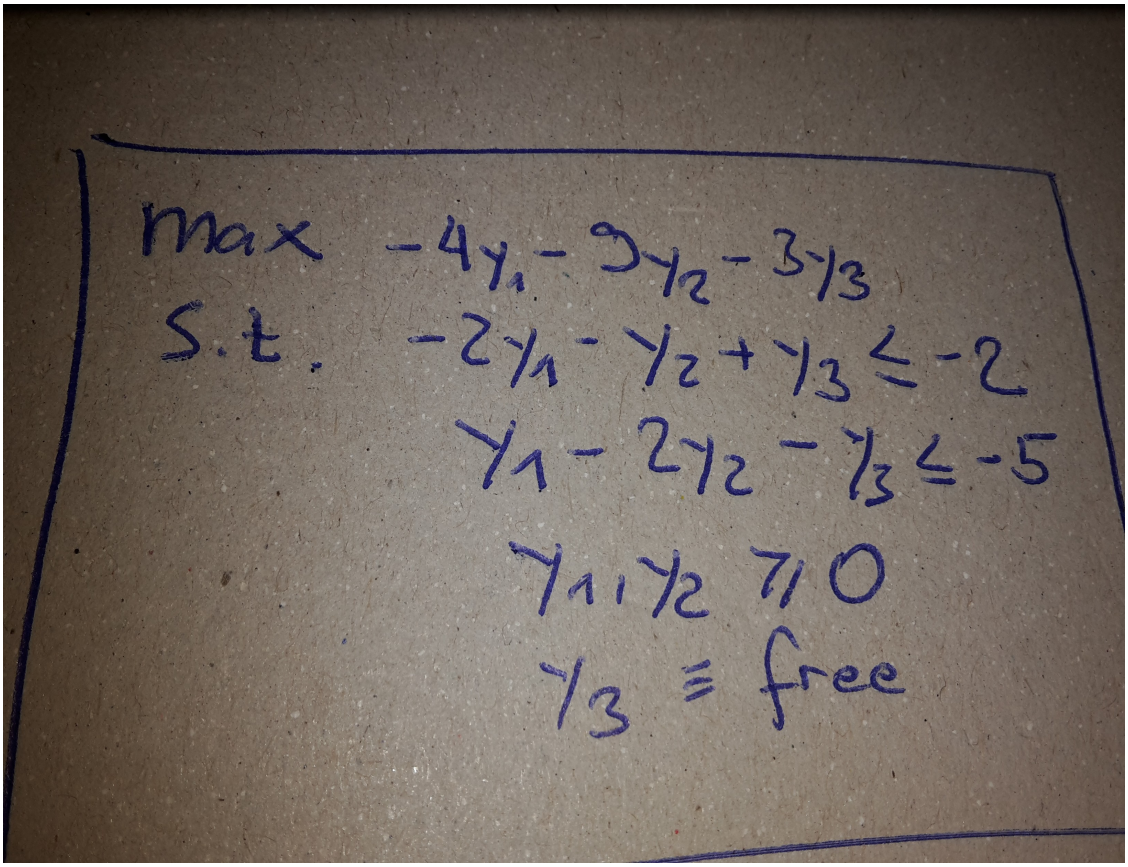
From the initially given primal LP



To an normalized formalization (all inequalities >=-fied)



To the dual problem:



c)

solution to the dual problem:

y = [y1, y2, y3] = [0, 7/3, 1/3]

with maximum value eqaul to -22

solution to the primal problem

x = [x1, x2] = [1, 4]

with minimum value eqaul to -22

since the primal and the dual problem both have the solution -22, we have found an optimal solution.

Note that a solution can be either found by :

- having a closer look to the plot from task 1, a) for the primal probel (min. Formulation) we see, when having a closer look at the boundaries that there are two possible options, either the point (1,4) in the x1-x2 coordinate system or the point close to the point (3,3). since we are looking for the minimum, and everything from the left and from the right side to this point (along the boundary) basically just gives lower values, and are descreasing lin-functions, (1,4) is the only viable choice.
- or by using a Lagrangien (of the convex formulation of this optimization problem), taking its derivatives and then finding its zeros (i.e. The unknowns – either the y or x vector),
- or by applying a numerical solver such as e.g. Matlab offers.

d)

from task c) we know the following facts

y = [y1, y2, y3] = [0, 7/3, 1/3]

x = [x1, x2] = [1, 4]

furthermore

b = [b1, b2] = [-2, -5] since we used b' * x

c = [c1, c2, c3] = [-4, -9, -3] since we used c' * y

Check CSC

-2x1 – x1 -+ x1 = -2*1 = -2 = b1 ok

a21*1+ a22*4 = -5 ok

a11y1 + a12y2 + a13y3 = c1 ok

a21y1 + a22y2 + a23y3 = c2ok

a31y1 + a32y2 + a33y3 = c3 ok

Task 2:

Given a directed Graph G=(E,V) as shown in Figure 2.1 and Figure 2.2,

we want to fnd the shortest paths between vertex A and all other nodes using the Bellman-Ford algorithm.

We use the following implementation of the Bellman-Ford Algorithm:

```
function BellmanFord(list vertices, list edges, vertex source)
   ::weight[],predecessor[]

   // This implementation takes in a graph, represented as
   // lists of vertices and edges, and fills two arrays
   // (weight and predecessor) with shortest-path
   // (less cost/weight/metric) information

   // Step 1: initialize graph
   for each vertex v in vertices:
       if v is source then weight[v] := 0
       else weight[v] := infinity
       predecessor[v] := null

   // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge (u, v) with weight w in edges:
           if weight[u] + w < weight[v]:
               weight[v] := weight[u] + w
               predecessor[v] := u

   // Step 3: check for negative-weight cycles
   for each edge (u, v) with weight w in edges:
       if weight[u] + w < weight[v]:
           error "Graph contains a negative-weight cycle"
   return weight[], predecessor[]
```

Algorithm: Bellman-Ford from Wikipedia

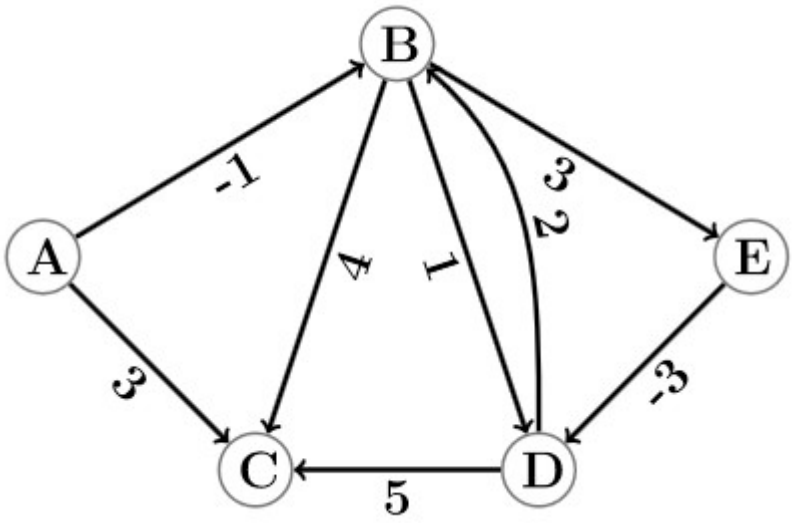


Figure 2.1: Problem 1

Edge-List with their weights	
Edge (u,v): Notation u => v edge from u to v	**Weight of edge (u,v)**
A => B	-1
A => C	3
B => C	4
B => D	1
B => E	3
D => B	2
D => C	5
E => D	-3

Distances d					
Iteration (after)	**A**	**B**	**C**	**D**	**E**
Initialization	0	Inf	Inf	Inf	inf
1	0	-1	3	-1	2
2	0	-1	3	-1	2
3	0	-1	3	-1	2
4	0	-1	3	-1	2

Predecessors p					
Iteration	**A**	**B**	**C**	**D**	**E**
Initial	-	-	-	-	-
1	-	A	A	E	B
2	-	A	A	E	B
3	-	A	A	E	B
4	-	A	A	E	B

There is no negative weight cycle contained in this Graph from figure 2.1

using the predecessor list we can back-track a shortest path from any Vertex to the starting vertex A.

Its path cost can the be calculated by looking up the edge costs and then simply summing the weights of its corresponding shortest path.

Shortest Pat from A

- to B: A=>B has a costs of -1
- to C: A=>C has a costs of 3
- to D: A=>B=>E=>D has a costs of -1+3-3 = -1
- to E: A=>B=>E has a costs of -1+3 = 2

Note that we already have found the shortes path after the first Iteration.

Therfore I propose the following optimization: If after the k-th iteration no values in neither p or d arrays have changes, compared to the values they had in the (k-1)-th iteration, we can end the loop after the k-th iteration.

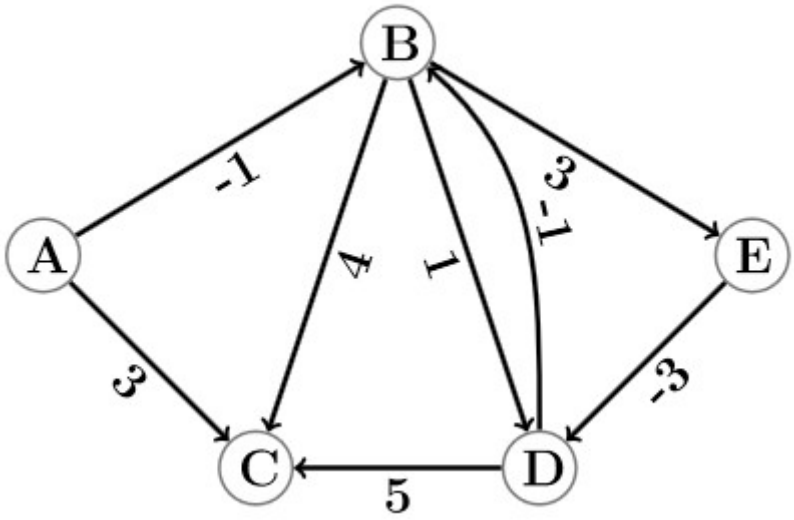


Figure 2.2: Problem 2

Edge-List with their weights	
Edge (u,v): Notation u => v edge from u to v	**Weight of edge (u,v)**
A => B	-1
A => C	3
B => C	4
B => D	1
B => E	3
D => B	-1
D => C	5
E => D	-3

Distances d					
Iteration (after)	**A**	**B**	**C**	**D**	**E**
Initialization	0	Inf	Inf	Inf	inf
1	0	-1	3	-1	2
2	0	-2	3	-1	2
3	0	-2	2	-2	1
4	0	-3	2	-2	1

Predecessors p					
Iteration	**A**	**B**	**C**	**D**	**E**
Initial	-	-	-	-	-
1	-	A	A	E	B
2	-	D	A	E	B
3	-	D	B	E	B
4	-	D	B	E	B

The graph from figure 2.2 contains a negative-weight cycle since e.g. The inequality $d(B) + w(B=>E) < d(E)$ holds true which is in numbers equal to $-3 + 3 < 1$