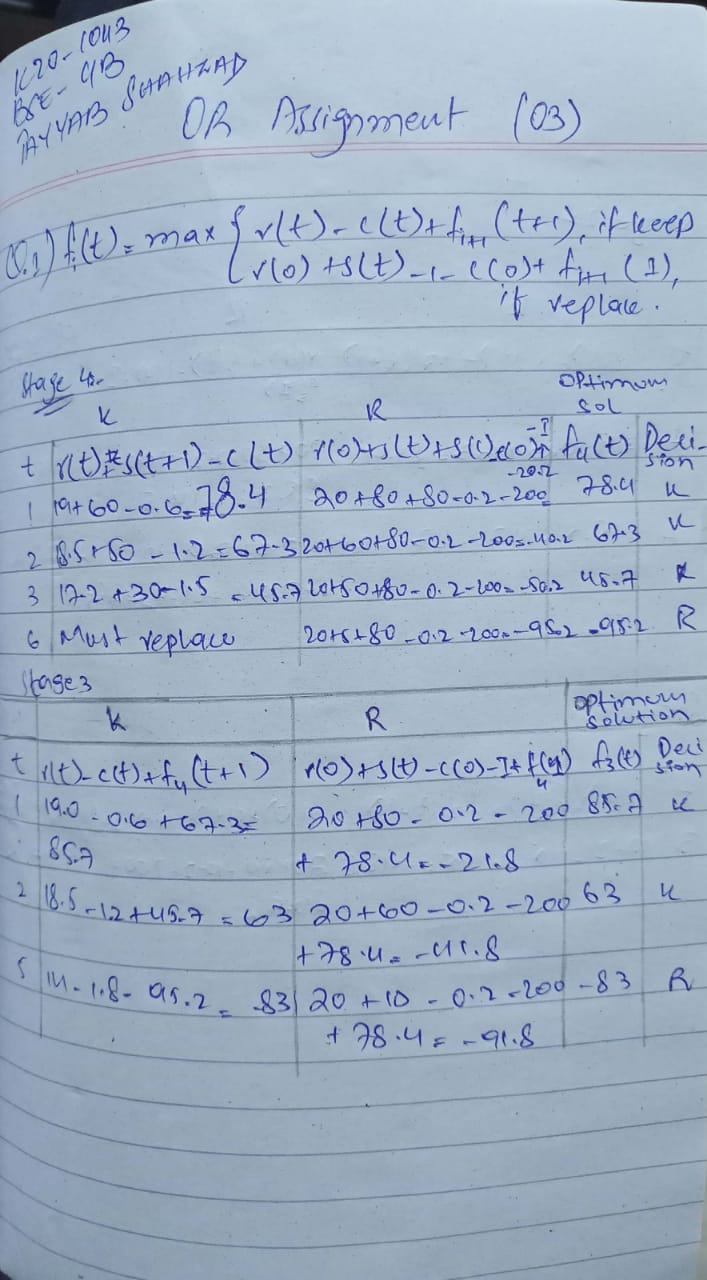
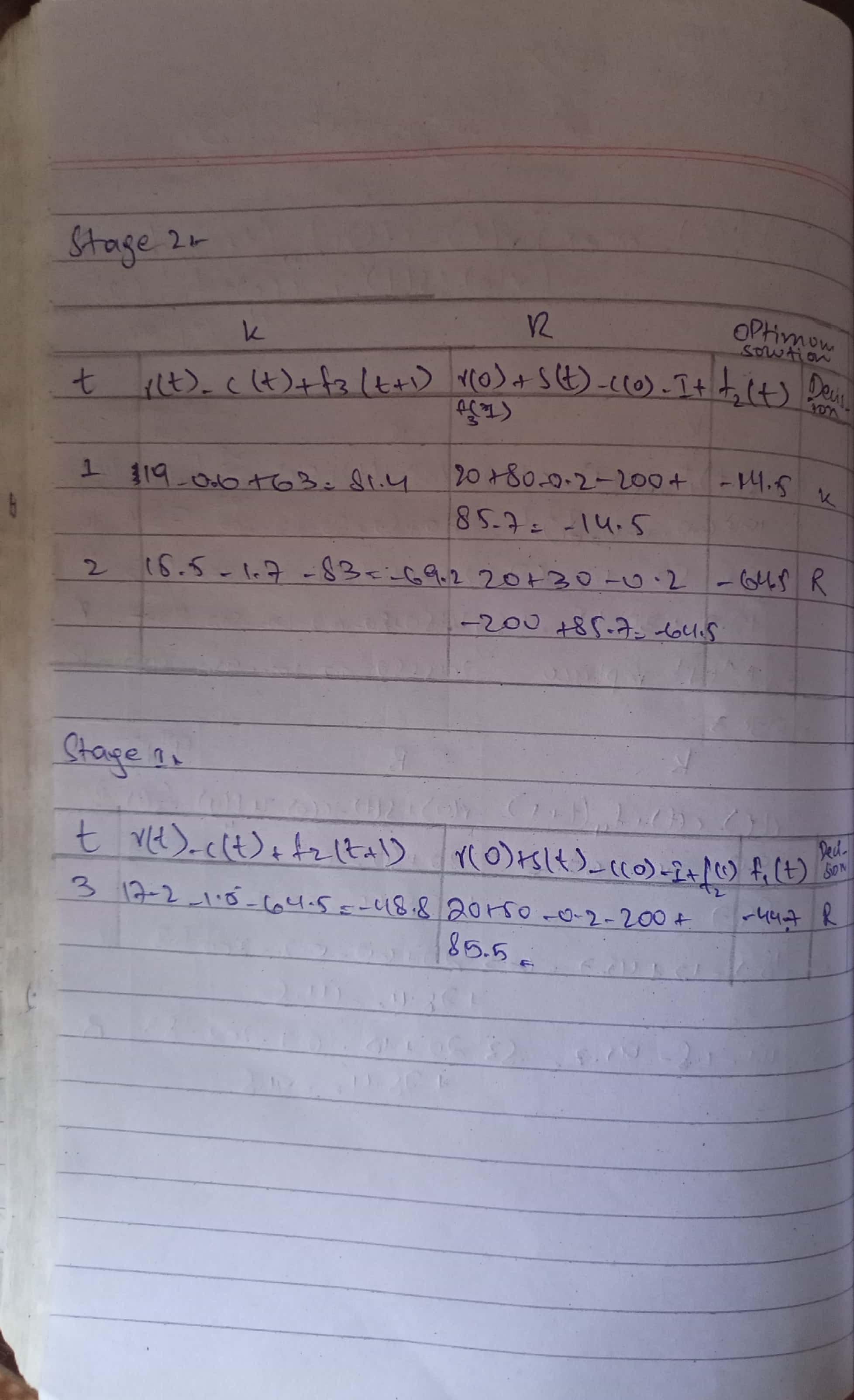
**Question01:**





**Question 02:**

Part 1)

Dynamic programming:

Dynamic programming is both a mathematical optimization method and a computer programming method. The method was developed by Richard Bellman in the 1950s and has found applications in numerous fields, from aerospace engineering to economics.

Characteristics of Dynamic programming:

1.subproblems overlap:

Subproblems are smaller variations of an original, larger problem. For example, in the Fibonacci sequence, each number in the series is the sum of its two preceding numbers (0, 1, 1, 2, 3, 5, 8,...). If you want to calculate the nth Fibonacci value in the sequence, you can break down the entire problem into smaller subproblems. These subproblems then overlap with one another as you find solutions by solving the same subproblem repeatedly. The overlap in subproblems occurs with any problem, which allows you to apply dynamic programming to break down complex programming tasks into smaller parts.

2. Substructure has optimal property:

Optimal substructure property materializes when you can arrive at an optimal solution after constructing all the other solutions that occurred from every subproblem you solved. The [solution you calculate](https://www.indeed.com/career-advice/career-development/how-to-calculate-average) from each overlap applies to the overall problem in order to function and optimize recursion. In the example of the Fibonacci sequence, each subproblem contains a solution that you can apply to each successive subproblem to find the next number in the series, making the entire problem display optimal substructure property.

Deterministic Dynamic Programming:

Deterministic Dynamic Programming Chapter Guide. Dynamic programming (DP) determines the optimum solution of a multivariable problem by decomposing it into stages, each stage comprising a single-variable subproblem.

TOP DOWN EXAMPLE:

Apply the Fibonacci sequence, where each number in the series represents the sum of the first two preceding numbers:

*{0, 1, 1, 2, 3, 5, 8,...}*

Understanding that the sum of the first two preceding values is the next number in the series can help you solve the entire problem when you want to calculate the nth Fibonacci number. Because you know the pattern for computing the future values in the series, you can apply a formula to determine the optimal solution without breaking down the problem into smaller subproblems. Using the appropriate formula when n > 1, you can compute an optimal solution with the top-down method when solving for an nth value of 13:

*Fib(n) = Fib(n - 1) + Fib(n - 2) =*

*Fib(13) = Fib(13 - 1) + Fib(13 - 2) =*

*Fib(n) = Fib(12) + Fib(11) = 23*

Using the top-down approach and applying memorization, you can cache the result of 23 in the database to input and call on your [code string](https://www.indeed.com/career-advice/career-development/how-to-code) for additional tasks.

Part2)

Integer programming with proto type example:

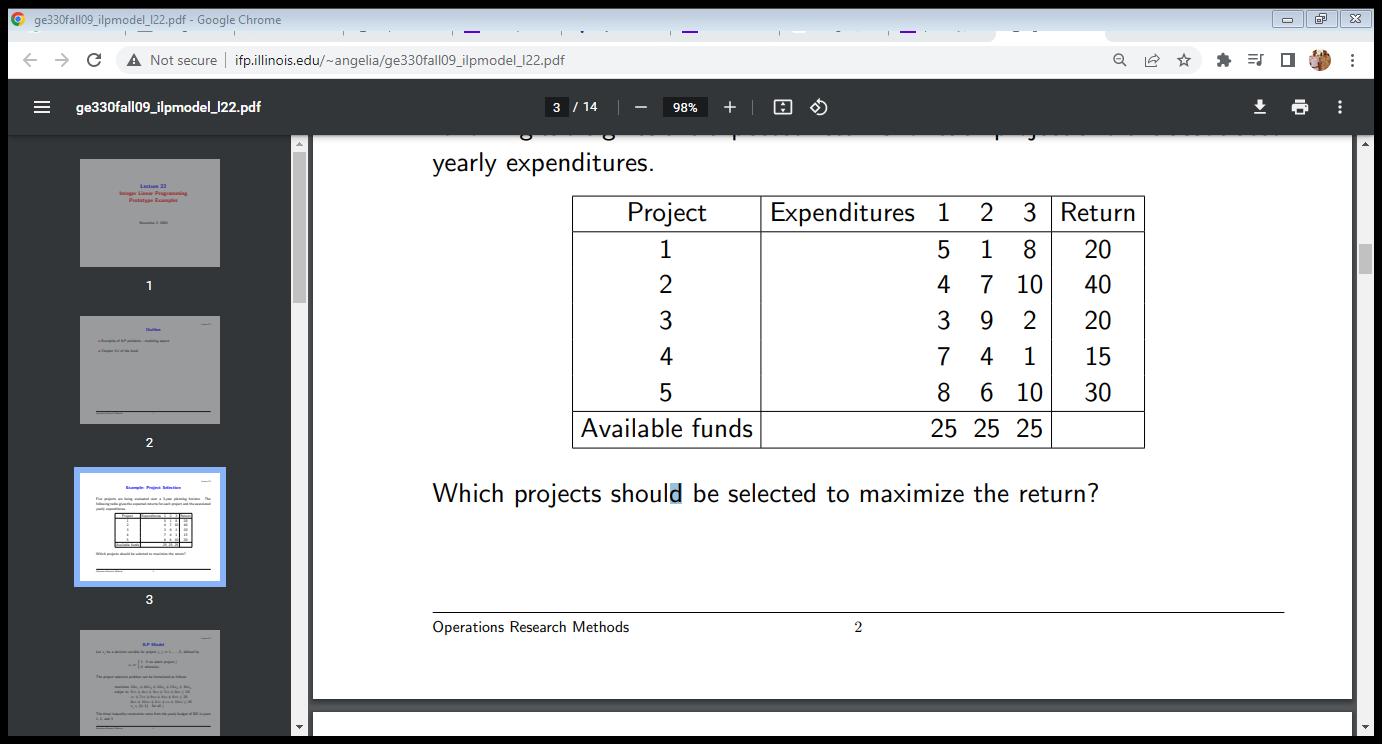
An integer programming problem is a [mathematical optimization](https://en.wikipedia.org/wiki/Mathematical_optimization) or [feasibility](https://en.wikipedia.org/wiki/Constraint_satisfaction_problem) program in which some or all of the variables are restricted to be [integers](https://en.wikipedia.org/wiki/Integer). In many settings the term refers to integer [linear programming](https://en.wikipedia.org/wiki/Linear_programming) (ILP), in which the objective function and the constraints (other than the integer constraints) are [linear](https://en.wikipedia.org/wiki/Linear_function_(calculus)).

Integer programming is [NP-complete](https://en.wikipedia.org/wiki/NP-complete). In particular, the special case of 0-1 integer linear programming, in which unknowns are binary, and only the restrictions must be satisfied, is one of [Karp's 21 NP-complete problems](https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems).

If some decision variables are not discrete the problem is known as a mixed-integer programming problem.[[](https://en.wikipedia.org/wiki/Integer_programming#cite_note-1)

Example:

Project Selection Five projects are being evaluated over a 3-year planning horizon. The following table gives the expected returns for each project and the associated yearly expenditures.



ILP Model

Let xj be a decision variable for project j, j = 1, . . . , 5, defined by xj = ( 1 if we select project j 0 otherwise The project selection problem can be formulated as follows: maximize 20x1 + 40x2 + 20x3 + 15x4 + 30x5 subjec to 5x1 + 4x2 + 3x3 + 7x4 + 8x5 ≤ 25 x1 + 7x2 + 9x3 + 4x4 + 6x5 ≤ 25 8x1 + 10x2 + 2x3 + x4 + 10x5 ≤ 25 xj ∈ {0, 1} for all j The three inequality constraints come from the yearly budget of $25 in years 1, 2, and 3.

Complete example link:

<http://www.ifp.illinois.edu/~angelia/ge330fall09_ilpmodel_l22.pdf>

part3)

Binary Integer programming (BIP) and its application and formulation with example:

Binary integer programming and it’s applications:

Binary integer programming is the problem of finding a binary vector *x* that minimizes a linear function *fTx* subject to linear constraints:

https://lost-contact.mit.edu/afs/inf.ed.ac.uk/group/teaching/matlab-help/Yesterday/R2014a/optim/ug/eqn1190748587.png

such that *A·x* ≤ *b*, *Aeq·x* = *beq*, *x* binary.

Integer linear program (ILP) applications generally fall into two categories: direct and transformed. In the direct category, the nature of the situation precludes assigning fractional values to the variables of the model. For example, the problem may involve determining whether or not a project is undertaken (binary variable) or finding the optimal number of machines needed to perform a task (general integer variable). In the transformed category, auxiliary integer variables are used to convert analytically intractable situations into models that can be solved by available optimization algorithms. For example, in sequencing two jobs, A and B, on a single machine, job A may precede job B or vice versa. The or-constraints make the problem analytically intractable because all mathematical programming algorithms deal with and-constraints only. Section 9.1.4 shows how auxiliary binary variables are used to transform the or constraints into and-constraints without altering the nature of the model.

Problem:

There are two types of balls, big (B) and small (S), which need to packed into boxes. One box can contain either:

nothing, or

1 S, or

1 B, or

2 S, or

2 B, or

1 B and 2 S

We are given the weight (float not integer) of each ball (in array ww). There are some constraints on the weights of the boxes and balls:

The total weight of a box ≤T≤T

In configurations 4 and 6 above, the difference of S should be ≤D≤D

In configuration 6 above, the weight of 2 S should be ≥≥ weight of 1 B

Now I want to minimise the number of boxes used.

My approach

I have modelled this as a linear programming problem with binary variables.

Let there be MM boxes, NSNS small balls and NBNB big balls.

Decision variables:

b1jb1j is a binary variable which is 11 if box jj is in configuration 11, else 00. Similarly for b2j,b3j,b4j,b5j,b6jb2j,b3j,b4j,b5j,b6j for each box jj.

xijxij is a binary variable which is 11 if ith big ball is in box jj, else 00.

yijyij is a binary variable which is 11 if ith small ball is in box jj, else 00.

Constraints:

Total weight of any box ≤T≤T

∑ixijwi+∑iyijwi≤T∀j∑ixijwi+∑iyijwi≤T∀j

Weight difference constraint:

yijwi−ykjwk≤D∀i,j,kyijwi−ykjwk≤D∀i,j,k

Each ball used exactly once:

∑jxij=1∀i∑jxij=1∀i

∑jyij=1∀i∑jyij=1∀i

Each box is exactly one configuration:

b1j+b2j+b3j+b4j+b5j+b6j=1

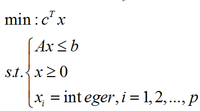
part 4)

One approach to solving integer programming problems is to ignore the integrality conditions and solve the problem with continuous decision variables. This is referred to as: quickest solution method.

Part5)

Branch and cut method is a very successful algorithm for solving a variety of integer programming problems, and it also can provide a guarantee of optimality. Many problems involve variables which are not continuous but instead have integer values, and they can be solved by branch-and cut method. This method are exact algorithm consisting of a combination of a cutting plane method and a branch-and-bound algorithm

Problem Statement

[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-2.png)

the above is a standard mixed-integer linear problem. the x and c are n-vector; b is m-vector; A is a m\*n matrix. if p=n, then the problem will become a pure integer linear problem. moreover, we can set the whole variables is equal to 0 or 1, then these variables will be a binary variables. Pure branch-and-bound can be speed up by combining cutting planes at the top of a branch-and-bound tree or at every node of the tree, because cutting planes considerably reduce the size of the tree.

Algorithm Process

1. Initialzation: Denote the initial LP problem by ILP^0 and set the active nodes to be L={ILP^0}. set the upper bound to be z^h=+∞. select one problem l∈L and set its lower-bound ont he optimal value is z^l=-∞.

2. Termination:if L=∅,then the solution x*which yielded the incumbent objetive value z is optimal. if the x* not exist, then the ILP is infeasible.

3. Problem selection: select and delete a problem ILP^l from L.

4. Relaxation: solve the LPR of ILP^l. If the ralaxation is infeasible, set z^h=+∞ and go to step 6.

5. Add Cutting Planes: Search for cutting planes, and if any are found, add them to the relazation and return to step 4.

6. Fathoming & Pruning:

(a)If z^l≥z, then go to step 2

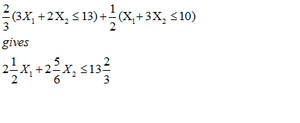
(b)If z^l<z and x^IR is integral feasible, then update z=z^l, delete all problem with z^l≥z, and go to step 2.

7. Partitioning: let [Wiki-3.png](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-3.png) be a partition of the constraint set S^l of problem ILP^l. Add problems [WIKI-4.png](https://optimization.mccormick.northwestern.edu/index.php/File:WIKI-4.png) to L, where ILP^lj is ILP^l with feasible region restricted to S^lj and [Wiki-5.png](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-5.png) for j=1,...k is set to the value of z^l for thr parent problem l. Go to step 2.

Generating Cutting Planes

Try to get better bounds so that more nodes of the branch and bound tree can be fathomed as early as possible. the idea is adding constraints that eliminate fractional solutions to the LP without eliminating any integer solutions. If we add exactly the right inequalities, the every extreme point of the LP will be integer If we add the right inequalities and the IP can be solved by solving the LP. The cutting planes take a weighted combination of the inequalities from the current LPR, and exploiting the fact that variables must be integral. So we also called this method "Chvatal-Gomory Cutting Planes"

example

[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-6.png)

LHS of the inequality is round down, which gives

[Wiki-7.png](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-7.png)

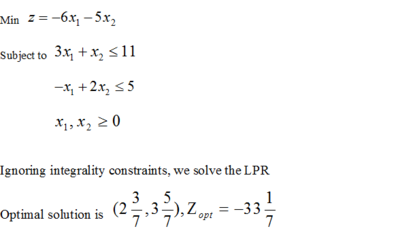
In any feasible solution to an IP, the LHS must take an integer vaalue, so the Rhs is round down.

finally we have the valid inequality:

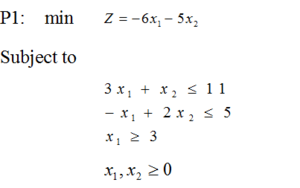
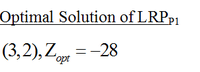
[WIKI-1.png](https://optimization.mccormick.northwestern.edu/index.php/File:WIKI-1.png)

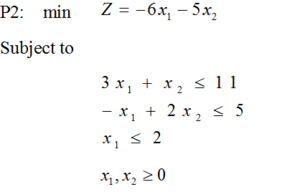
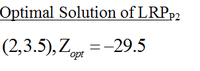
Example

The integer programming problem

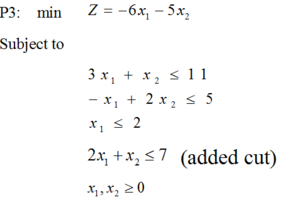
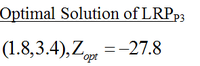
[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-9_2.png)

Sub-problems generated by branching on X1

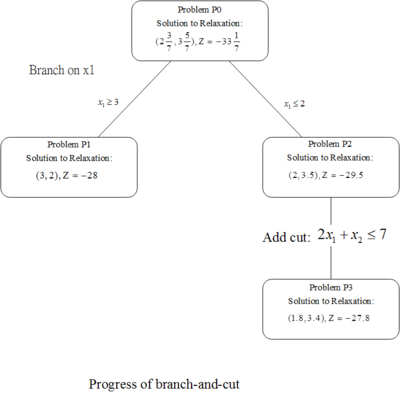
[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-10_2.png) [](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-11.png)

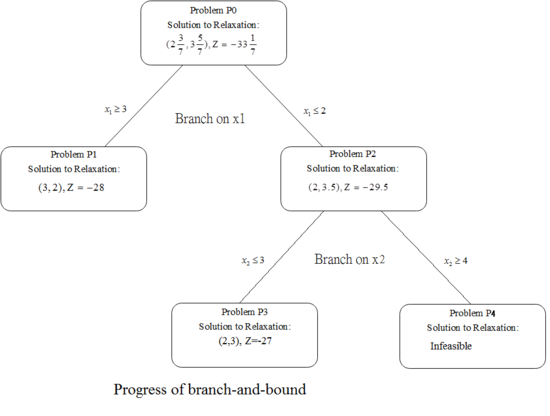
[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-12_2.png) [](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-13.png)

Add a cut to P2

[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-14_2.png) [](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-15.png)

Comparison

[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-16.png)

[](https://optimization.mccormick.northwestern.edu/index.php/File:Wiki-30_4.png)

We can found that there are only three steps for branch-and-cut method; however, the branch-and-bound method uses 4 steps to solve the same problem. We can prove that using the branch-and-cut method is faster than using the branch-and-bound method

Conclusion

There are many methods to solve the mixed-integer linear programming. Gomory Cutting Planes is fast, but unreliable. Branch and Bound is reliable but slow. The Branch and cut combine the advantages from these two methods and improve the defects. It has proven to be a very successful approach for solving a wide variety of integer programming problems. We can solve the MILP by taking some cutting planes before apply whole system to the branch and bound, Branch and cut is not only reliable, but faster than branch and bound alone. Finally, we understand that using branch and cut is more efficient than using branch and bound.

Part 6)

Constraint programming is an embedding of constraints in a host language. The first host languages used were logic programming languages, so the field was initially called constraint logic programming. The two paradigms share many important features, like logical variables and backtracking.