**PROSIT – 3 | WHAT’S THE PROBLEM**

**KEYWORDS** A

* Simplex \* - standard technique in linear programming for solving an optimization problem, typically one involving a function and several constraints expressed as inequalities. The method assumes knowledge of an extreme point, with Phase I used to find one if necessary. An algebraic test determines the optimality of the extreme point. If not optimal, an adjacent extreme point is sought. The process is repeated until an optimal extreme point is found or an unbounded case occurs. Despite potentially exponential steps in theory, in practice, the method typically converges on the optimal solution in a small multiple of the number of extreme points.
* Operation research - Operations Research (OR) is a discipline of problem-solving and decision-making. It uses advanced analytical methods to help management run an effective organization. Problems are broken down, analyzed and solved in steps.
  + Identify a problem
  + Build a model around the real-world problem
  + Use the model and data to arrive at solutions
  + Test the solution and analyze its success
  + Implement the solution
* Sub-optimal - A suboptimal algorithm is an algorithm that produces results that are not as good as those of the best possible algorithm for the same problem. A common example of a suboptimal algorithm is the greedy algorithm, which is often used in optimization problems.
* Systems of equations / inequations \* - A system of linear equations consists of two or more linear equations made up of two or more variables such that all equations in the system are considered simultaneously.
* Metaheuristics - A metaheuristic is a higher level procedure or heuristic designed to find, generate, or select a lower level procedure or heuristic (partial search algorithm) that may provide a sufficiently good solution for an optimization problem.
* Generic problem-solving methods - broadly applicable techniques or frameworks that can be used to solve a wide range of optimization and decision problems, rather than being tailored to a specific problem. These methods are often adaptable to various types of combinatorial structures like graphs, sets, or sequences, and can be applied to many domains.
  + Greedy Algorithm
  + Dynamic Programming
  + Backtracking
  + Metaheuristics (Genetic Algorithms, Simulated Annealing, etc.)
* Dynamic programming - a method used in mathematics and computer science to solve complex problems by breaking them down into simpler subproblems. By solving each subproblem only once and storing the results, it avoids redundant computations, leading to more efficient solutions for a wide range of problems.
* Linear programming \* - a mathematical concept that is used to find the optimal solution of the linear function. This method uses simple assumptions for optimizing the given function.
* Adaptive approach - a method that can adjust its behaviour based on changing conditions or input data during the optimization process. This adaptability allows the algorithm to improve its performance or make decisions that respond to dynamic environments, incomplete information, or unknown parameters.
* Least complex approach - a method or algorithm that achieves the desired adaptability with minimal computational overhead, simplicity in design, and efficient use of resources (like time and memory).

**CONTEXT** A

Finding an efficient solution for an NP-hard problem, like the Traveling Salesman Problem (TSP), requires careful consideration of different approaches. Agathe suggests exploring Operations Research methods, including the Simplex method, dynamic programming, or metaheuristics. The goal is to balance the pursuit of exact solutions with practical ones that perform well under time constraints, ensuring computational feasibility while addressing the problem's complexity.

**PROBLEM STATEMENT** A

How can an efficient solution be found for a complex optimization problem, knowing that finding an optimal solution in polynomial time is impossible due to the problem’s computational complexity?

**HYPOTHESIS / SOLUTION APPROACHES** A

* Try using Simplex (algorithm)
* Try using Linear programming \* [[6](https://www.geeksforgeeks.org/linear-programming/)]
* Try using dynamic programming \* [[7](https://www.geeksforgeeks.org/dynamic-programming/)]
* Metaheuristics
* Systems of equations / inequations (constraints) [[8](https://math.libretexts.org/Bookshelves/Algebra/College_Algebra_1e_(OpenStax)/07%3A_Systems_of_Equations_and_Inequalities/704%3A_Systems_of_Nonlinear_Equations_and_Inequalities__Two_Variables)]

**CONSTRAINTS** A

* Computation time
* Problem size (Large)

**ACTION PLAN** A

* Study the hypothesis and resources.
* Formalize the constraints of the project using system of equations & inequations.
* Compare Simplex, Linear and Dynamic programming (Theoretically and mathematically).

**DELIVERABLE** A

After comparing the things mentioned in the Action Plan, find a solution.

**BONUS** A

Can the simplex, linear and dynamic programming approach be applied to VRP. Justify

**ACTION PLAN NOTES** A

**SIMPLEX ALGORITHM**

Links [[1](https://optimization.cbe.cornell.edu/index.php?title=Simplex_algorithm)] [[2](https://web.stanford.edu/class/msande310/lecture09.pdf)] [[3](https://www.britannica.com/topic/simplex-method)] [[4](https://www.youtube.com/watch?v=CVA3dXkq7PU)] [[5](https://www.di.ens.fr/~vergnaud/algo0910/Simplex.pdf)] [6 – Check LP – shows how to perform the Simplex Algorithm]

* The simplex algorithm can be thought of as one of the elementary steps for solving the inequality problem, since many of those will be converted to LP and solved via Simplex algorithm. [1]
* The basic idea of the simplex method to confine the search to corner points of the feasible region (of which there are only finitely many) in a most intelligent way. [2]
* In contrast, interior-point methods will move in the interior of the feasible region, hoping to by-pass many corner points on the boundary of the region. [2]
* The key for the simplex method is to make computers see corner points; and the key for interior-point methods is to stay in the interior of the feasible region. [2]
* Theorems:
  + The feasible region for an LP problem is a convex set (Every linear equation's second derivative is 0, implying the monotonicity of the trend). Therefore, if an LP has an optimal solution, there must be an extreme point of the feasible region that is optimal. [1]

*This can also be said as*

For LP in the standard form, a Corner Point is maximal if and only if the objective vector is a conic combination of the normal direction vectors of the m hyperplanes. [2]

* + For an LP optimization problem, there is only one extreme point of the LP's feasible region regarding every basic feasible solution. Plus, there will be a minimum of one basic feasible solution corresponding to every extreme point in the feasible region. [1]

*This can also be said as*

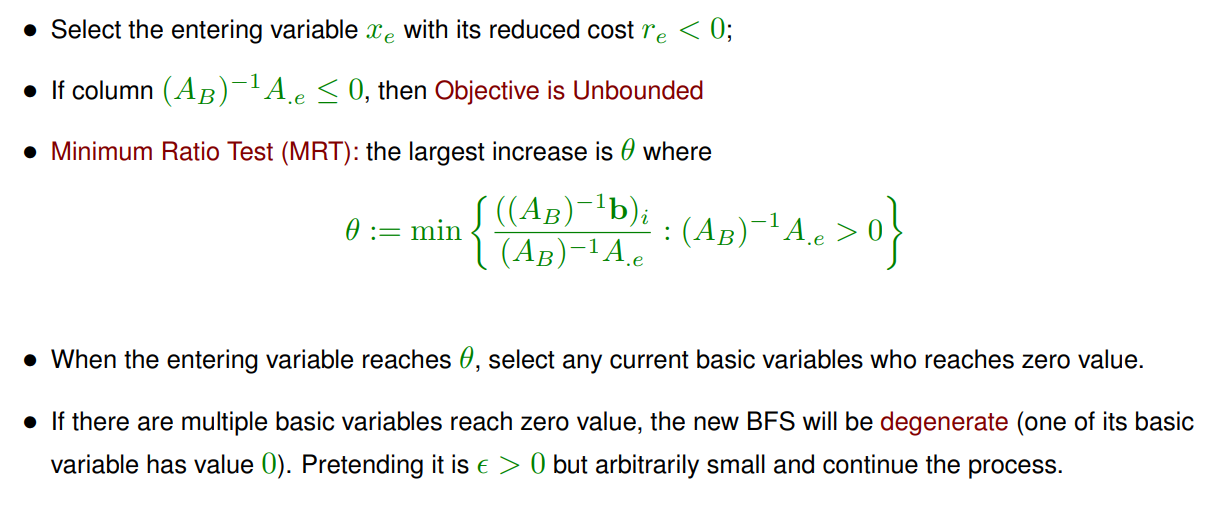
Consider the polyhedron in the standard LP form. Then a basic feasible solution and a corner point are equivalent; the former is algebraic and the latter is geometric. [2]

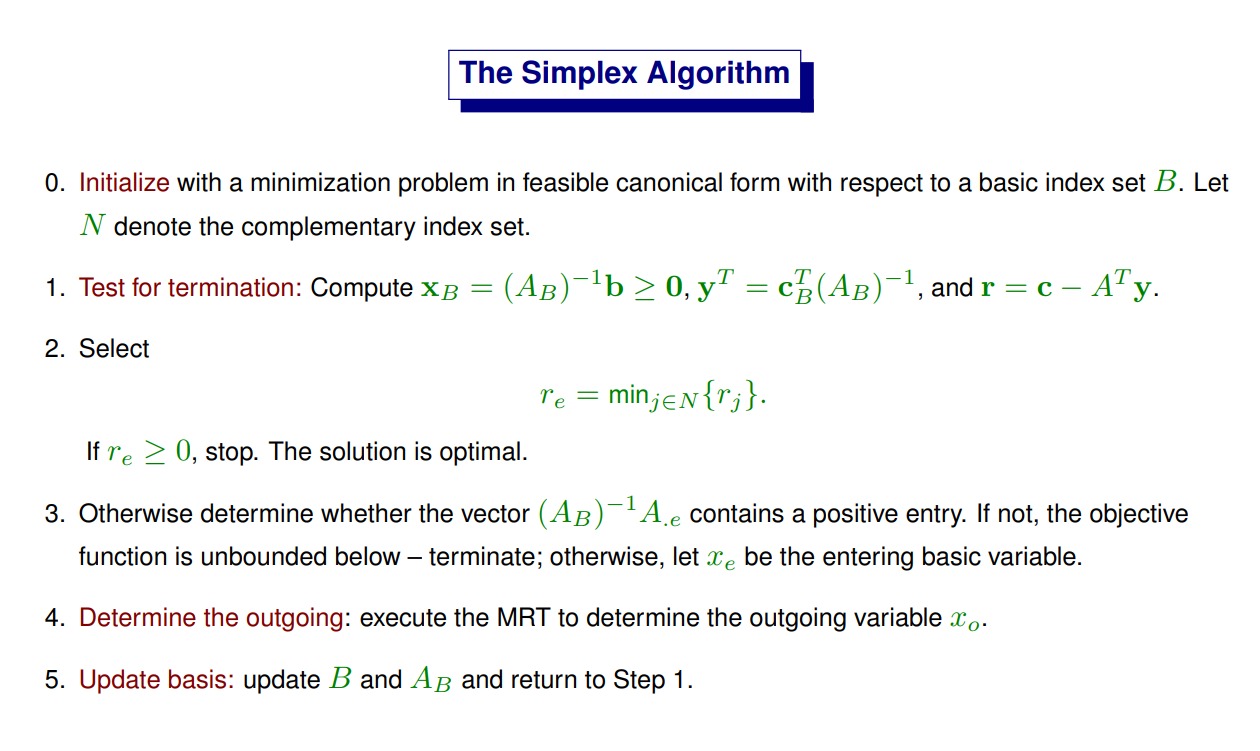
* + (The Fundamental Theorem of LP in Algebraic form) - Given (LP) and (LD) where A has full row rank m, [2]
    - if there is a feasible solution, there is a basic feasible solution (Caratheodory’s theorem);
    - if there is an optimal solution, there is an optimal basic solution.
* Optimality test – [2]
* Corner point - an intersection point of the hyperplanes of m linearly-independent inequality constraints. [2]
* These constraints are called active or binding constraints at the corner solution. [2]
* Two corner solutions are adjacent if they differ by one active constraint. [2]

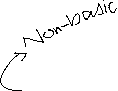
Basic Feasible Solution [2] [4] – Math behind BFS [Extra Information – can check in detail later; YouTube video – [4]]

* In the LP standard form, assume that we selected m linearly independent columns, denoted by the index set *B* from *A* and solve ***AB xB = b*** for the m-vector xB. By setting the variables xN of x corresponding to the remaining columns of A equal to zero, we obtain a solution x of Ax = b.
* Then x is said to be a basic solution to (LP) with respect to basis AB.
* The components of xB are called basic variables and those of xN are called nonbasic variables.
* Two basic solutions are adjacent if they differ by exactly one basic (or nonbasic) variable.
* If a basic solution satisfies xB ≥ 0, then x is called a basic feasible solution (BFS), and it is an extreme point of the feasible region.
* If one or more components in xB has value zero, x is said to be degenerate.

Minimum Ratio Test [2] – Math behind MRT [Extra Information – can check later; look for YouTube resources]







The simplex algorithm is like a method for moving between "corner points" (extreme points) of the feasible region (the area where all constraints are satisfied) to improve the objective function. You keep testing whether the current corner point gives you the best possible value (optimal), and if not, you choose a new variable to enter the solution, making your way towards the best solution step by step.

If you reach a point where you can't improve anymore, you've found the optimal solution. If the function can keep getting smaller without bound, the algorithm stops and says the problem is unbounded.

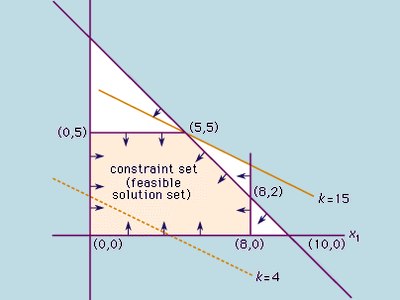


Figure 1: [3]

Simplex algorithm usually takes polynomial time [5].

**LINEAR PROGRAMMING**

Links [[6](https://www.geeksforgeeks.org/linear-programming/)] [[9](https://www.cs.cmu.edu/~15451-f17/lectures/lec18-lp2.pdf)]

* Linear programming or Linear optimization is a technique that helps us to find the optimum solution for a given problem, an optimum solution is a solution that is the best possible outcome of a given particular problem. [6]
* Components: [6]
  + Decision Variables: Variables you want to determine to achieve the optimal solution.
  + Objective Function: Mathematical equation that represents the goal you want to achieve
  + Constraints: Limitations or restrictions that your decision variables must follow.
  + Non-Negativity Restrictions: In some real-world scenarios, decision variables cannot be negative
* Characteristics: [6]
  + Finiteness: The number of decision variables and constraints in an LP problem are finite.
  + Linearity: The objective function and all constraints must be linear functions of the decision variables. It means the degree of variables should be one.
* General formula of a linear programming problem (LPP) is: [6]
  + Objective Function: Z = ax + by
  + Constraints: cx + dy ≥ e, px + qy ≤ r
  + Non-Negative restrictions: x ≥ 0, y ≥ 0
  + In the above condition x, and y are the decision variables.
* To solve a LPP: [6]
  + Step 1: Mark the decision variables in the problem.
  + Step 2: Build the objective function of the problem and check if the function needs to be minimized or maximized.
  + Step 3: Write down all the constraints of the linear problems.
  + Step 4: Ensure non-negative restrictions of the decision variables.
  + Step 5: Now solve the linear programming problem using any method generally we use either the simplex or graphical method.
* Simplex algorithm is a part of Linear Programming. [9]
  + The running time is still linear in the number of constraints, but blows up exponentially in the dimension.
  + It’s not guaranteed to run in polynomial time, and you can come up with bad examples for it, but in general the algorithm runs pretty fast.
* Linear programs could always be solved in polynomial time by something called the Ellipsoid Algorithm (but it tends to be slow in practice). [9]
* A faster polynomial-time algorithm called Karmarkar’s Algorithm was developed, which is competitive with Simplex. [9]

**DYNAMIC PROGRAMMING**

Links [[7](https://www.geeksforgeeks.org/dynamic-programming/)]

* Dynamic Programming (DP) is a method used in mathematics and computer science to solve complex problems by breaking them down into simpler subproblems. By solving each subproblem only once and storing the results, it avoids redundant computations, leading to more efficient solutions for a wide range of problems.
* Working:
  + Identify Subproblems: Divide the main problem into smaller, independent subproblems.
  + Store Solutions: Solve each subproblem and store the solution in a table or array.
  + Build Up Solutions: Use the stored solutions to build up the solution to the main problem.
  + Avoid Redundancy: By storing solutions, DP ensures that each subproblem is solved only once, reducing computation time.
* When to use DP?
  + Optimal Substructure: Optimal substructure means that we combine the optimal results of subproblems to achieve the optimal result of the bigger problem.
  + Overlapping Subproblems: The same subproblems are solved repeatedly in different parts of the problem.
* Approaches:
  + Top-Down Approach [Memoization – Prosit 1]
    - In the top-down approach, also known as memoization, we start with the final solution and recursively break it down into smaller subproblems. To avoid redundant calculations, we store the results of solved subproblems in a memoization table.
    - Top-down approach:
      * Starts with the final solution and recursively breaks it down into smaller subproblems.
      * Stores the solutions to subproblems in a table to avoid redundant calculations.
      * Suitable when the number of subproblems is large and many of them are reused.
  + Bottom-Up Approach [Tabulation]
    - In the bottom-up approach, also known as tabulation, we start with the smallest subproblems and gradually build up to the final solution. We store the results of solved subproblems in a table to avoid redundant calculations.
    - Bottom-up approach:
      * Starts with the smallest subproblems and gradually builds up to the final solution.
      * Fills a table with solutions to subproblems in a bottom-up manner.
      * Suitable when the number of subproblems is small and the optimal solution can be directly computed from the solutions to smaller subproblems.
* Common algorithms that use DP:
  + Longest Common Subsequence (LCS): Finds the longest common subsequence between two strings.
  + Shortest Path in a Graph: Finds the shortest path between two nodes in a graph.
  + Knapsack Problem: Determines the maximum value of items that can be placed in a knapsack with a given capacity.
  + Matrix Chain Multiplication: Optimizes the order of matrix multiplication to minimize the number of operations.
  + Fibonacci Sequence: Calculates the nth Fibonacci number.

**METAHEURISTICS [NOT IN DETAIL – MORE IN PROSIT 4]**

Links [[10](https://iao.hfuu.edu.cn/images/teaching/lectures/metaheuristic_optimization/01_introduction.pdf)] [[11](https://katalon.com/resources-center/blog/black-box-testing)] [[12](https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf)]

* A metaheuristic is a method for solving very general classes of problems. It combines objective functions or heuristics in an abstract and hopefully efficient way, usually by treating them as black box-procedures. [10]
* Metaheuristics are applied to “I know it when I see it problems”. [12]
* They’re algorithms used to find answers to problems when you have very little to help you: you don’t know beforehand what the optimal solution looks like, you don’t know how to go about finding it in a principled way, you have very little heuristic information to go on, and brute-force search is out of the question because the space is too large. But if you’re given a candidate solution to your problem, you can test it and assess how good it is. [12]
* Algorithm performance has two dimensions: solution quality and required runtime. [10]
* Anytime Algorithms are optimization methods which maintain an approximate solution at any time during their run and iteratively improve this guess. [10]
* All metaheuristics are Anytime Algorithms. [10]
  + Most optimization algorithms produce approximate solutions of different qualities at different points during their process.
* Optimization problems and algorithms can be divided into online and offline processes.
  + Online optimization problems are tasks that need to be solved quickly in a time span usually ranging between ten milliseconds to a few minutes.
  + In offline optimization problems, time is not so important and a user is willing to wait maybe up to days or weeks for better results.
* Types of Algorithms:
  + In each execution step of a deterministic algorithm, there exists at most one way to proceed. If no way to proceed exists, the algorithm has terminated.
  + A randomized or probabilistic algorithm includes at least one instruction that acts on the basis of random numbers. In other words, a probabilistic algorithm violates the constraint of determinism.

**SYSTEM OF EQUALITIES/INEQUALITIES [SHOULD BE STUDIED FIRST – HIGH SCHOOL MATHEMATICS – WORKSHOP]**

Links [[8](https://math.libretexts.org/Bookshelves/Algebra/College_Algebra_1e_(OpenStax)/07%3A_Systems_of_Equations_and_Inequalities/704%3A_Systems_of_Nonlinear_Equations_and_Inequalities__Two_Variables)]

**COMPARISON OF LINEAR PROGRAMMING, DYNAMIC PROGRAMMING, AND SIMPLEX ALGORITHM [WRT TO PROJECT]**

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Linear Programming | Dynamic Programming | Simplex Algorithm |
| Overview | Optimizes a linear objective function with linear constraints. | Breaks down problems into simpler sub-problems solved recursively. | An iterative method to solve LP problems by moving along feasible edges. |
| Relevance to Project | Suitable if travel times are static and linear. | Suitable for dynamic, time-dependent traffic flows. | Suitable for static linear relationships, not for time-varying conditions. |
| Handling Dynamic Traffic Flow | Struggles with non-linear, time-varying traffic flows. | Can effectively model and handle time-dependent traffic changes. | Cannot handle dynamic traffic flows due to linear assumptions. |
| Modelling Complexity | Simple for linear, static problems. | Capable of handling complex, non-linear, and dynamic problems. | Simple for static, linear problems; fails with complex dynamics. |
| Computation Time | Efficient for small to medium problems but not dynamic ones. | Computationally expensive, especially for large-scale problems. | Efficient for large LP problems but not adaptive to real-time changes. |
| Scalability | Scales well for linear and static cases. | Can be computationally expensive with larger problem sizes. | Scales efficiently for large linear optimization problems. |
| Flexibility | Limited flexibility with non-linear constraints like traffic. | Highly flexible for time-dependent and complex conditions. | Limited to linear optimization; lacks flexibility for dynamic constraints. |
| Best Use Case | Static travel time, linear cost minimization. | Dynamic, time-dependent travel optimization, like in your project. | Large-scale static optimization with linear relationships. |

Conclusion: Dynamic Programming (DP) is the most suitable for your project, which involves dynamic traffic flows and time-dependent changes.