Network Flow Algorithms, Transforms and Applications

Priyatham Sai Chand Bazaru, Meet Kishorbhai Butani, Sean Butler, Shreyas Chaudhary

Abstract

Network Flows are unavoidable in our day-to-day lives, coming from all backgrounds and every walk of life. It is evident that they should be following patterns, logic and at the end algorithms that we could utilize to maximum the output through such networks be it in any field of study. This can be achieved through rigoures testing and formulation of theorems, complexity and transformations that would assist in the application of such networks in the world efficiently and aid us in the development of both this field and other fields in different aspects of science. In this presentation we aim to showcase the importance of network flow algorithms, the origins, to prove the theorems that are highly regarded such that the Ford-Fulkerson algorithm, the Hungarian method and many others. We would continue to explain the transformations that are possible on such networks, we conclude with the applications that can apply such algorithms and transformations that shaped the field of algorithms.

Index Terms

Network Flow, Ford-Fulkerson, Hungarian, transformations, applications

I. Introduction

Everywhere we look in our daily lives, we see networks. They bring us electricity, power, telephone service, transportation, and even the goods we buy. And in all of these networks, we want to move something from one place to another as efficiently as possible. This book is about how to model these applications as mathematical objects called network flow problems. We will also study different ways to solve these problems using algorithms. Network flows is a field that lies between applied mathematics, computer science, engineering, management, and operations research. It has a long history, dating back to the work of Gustav Kirchhof, who analyzed electrical circuits. This early work laid the foundation for network flow theory and showed that networks are useful mathematical objects for representing physical systems. Much of the early work in network flows was descriptive. For example, it answered questions like: If we apply a set of voltages to a given network, what will be the resulting current flow? The questions we address in this book are a bit different. We want to know: If we have different ways to use a network, which way will be most cost-effective? This is a more recent question, and it can be traced back to the late 1940s and early 1950s. This is when optimization became an independent field of study, and when computers became powerful enough to solve these problems.

We feel that a full understanding of network flow algorithms and a full appreciation for their use requires more than an in-depth knowledge of good algorithms for core models. Consequently, even though this topic is our central thrust, we also devote considerable attention to describing applications of network flow problems. Indeed, we feel that our discussion of applications throughout the text, and in a concluding chapter is one of the major distinguishing features of our coverage.

[1]

A. Flow Basics and Examples

To understand how networks work, it is preliminary to understand how a flow is defined along with the notations it takes to represent such networks. This discussion can be initiated with *Maximum Flow Problem*. The maximum flow problem is very easy to state: In a capacitated network, we wish to send as much flow as possible between two special nodes, a source node s and a sink node t, without exceeding the capacity of any arc. To solve the Maximum Flow Problem there are two major types of algorithms that can be utilized to obtain the solution. They are as follows:

1) Augmenting path algorithms that maintain mass balance constraints at every node of the network other than the source and sink nodes. These algorithms incrementally augment flow along paths from the source node to the sink node.
2) Preflow-push algorithms that flood the network so that some nodes have excesses (or buildup of flow). These algorithms incrementally relieve flow from nodes with excesses by sending flow from the node forward toward the sink node or backward toward the source node.

The flow was presented with a minimal example as illustration in 1 This helps explain the basic and picture the application that we discussed in the presentation in a much more concrete way. The flow example was presented with a change in flows and modifying the capacities of the edges to help explain how the flow that originates at the source is always flowed to the sink.

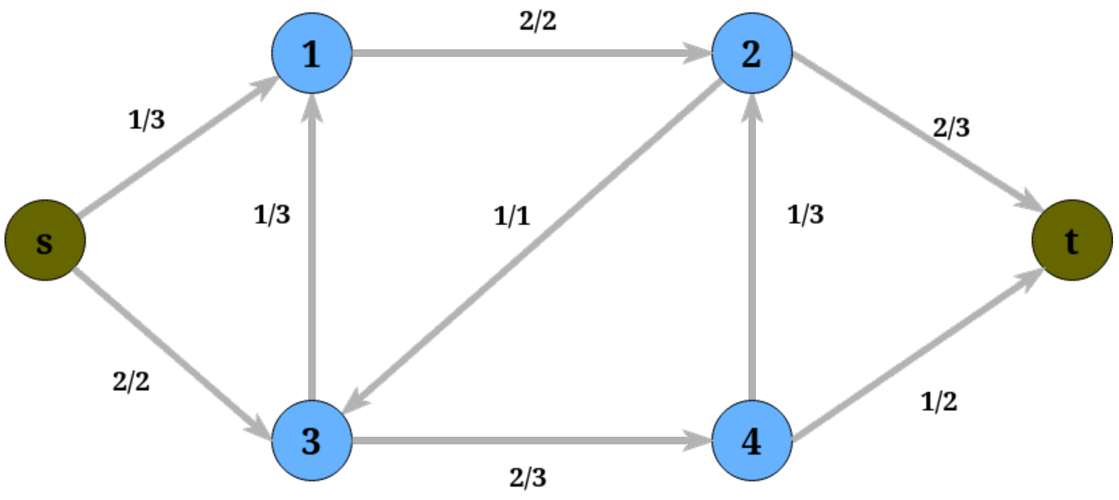


Fig. 1. Flow Example

1) Notation: We consider a capacitated network G = (N, A) with a nonnegative capacity u_{ij} associated with each arc $(i, j) \in A$. Let $V = max\{u_{ij} : (i, j) \in A\}$. As before, the arc adjacency list $A(i) = \{(i, k) : (i, k) \in A\}$ contains all the arcs emanating from node i. To define the maximum flow problem, we distinguish two special nodes in the network G: a source node s and a sink node t. We wish to find the maximum flow from the source node s to the sink node t that satisfies!The arc capacities and mass balance constraints at all nodes. We can state the problem formally as follows.

$$\text{maximise v} \tag{1}$$

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ij} = \begin{cases} v \text{ for } i = s, \\ 0 \text{ for all i} \in N - \{\text{s and t}\} \\ -v \text{ for } i = t \end{cases} \tag{2}$$

$$0 \leq x_{ij} \leq u_{ij} \text{ for each } (i, j) \in A \tag{3}$$

We refer to a vector $x = \{X_{ij}\}$ satisfying (2) and (3) as a flow and the corresponding value of the scalar variable v as the value of the flow . We consider the maximum flow problem subject to the following assumptions.

Assumption 6.1. The network is directed.

Assumption 6.2. All capacities are non-negative integers.

Assumption 6.3. The network does not contain a directed path from node s to node t composed only of infinite capacity arcs.

Assumption 6.4. Whenever an arc (i, j) belongs to A, arc (j, i) also belongs to A.

Assumption 6.5. The network does not contain parallel arcs (i.e., two or more arcs with the same tail and head nodes).

These assumptions hold for network graph that we discuss henceforth, the assumption make sure that the graphs follow the constraints that flow can be matched with the real world applications and physical complexity that would restrict the creation and formulation of theorems. With assumptions in place we can move on to the definition of other network flow types and cuts that will be followed in the next section.

Definition of Flow A flow in a G is a function $f : V \times V \Rightarrow R$ which satisfies the following properties as we listed below these constraints and properties help define the flow completely which follow the assumptions that was listed before as well.

- **Capacity Constraint:** For all $u, v \in V, f(u, v) \leq C$
- **Flow Conservation:** For all $u \in V - \{s, t\}$,
$$\sum_{v \in V} f(u, v) = 0$$
- **Skew Symmetry:** For all $u, v \in V, f(u, v) = -f(v, u)$

The Capacity Constraint shows that any flow in the network has to be less than or equal to the capacity of the edge between two nodes which was defined. The flow conservation shows that the sum of all the flows in a network will equate to zero given that it covers all the nodes in the network except the source and the sink. Finally, skew symmetry shows that if there exists a positive flow between two nodes then the flow in the opposite way between the same number of nodes is equal in value but negative in the signage where the negative sign depicts that the flow is in the opposite direction.

B. Flow and Cuts

In this section we discuss some elementary properties of flows and cuts. We use these properties to prove the max-flow min-cut theorem to establish the correctness of the generic augmenting path algorithm. We first review some of our previous notation and introduce a few new ideas.

Residual network: The concept of residual network plays a central role in the development of all the maximum flow algorithms we consider. Earlier in Section 2.4 we defined residual networks and discussed several of its properties. Given a flow x, the residual capacity r_{ij} of any arc $(i, j) \in A$ is the maximum additional flow that can be sent from node i to node j using the arcs (i, j) and (j, 0). [Recall our assumption from Section 6.1 that whenever the network contains arc (i, j), it also contains arc (j, i).] The residual capacity r_{ij} has two components: (1) $U_{ij} - X_{ij}$, the unused capacity of arc (i, j), and (2) the current flow X_{ji} on arc (j, i), which we can cancel to increase the flow from node i to node j. Consequently, $r_{ij} = U_{ij} - X_{ij} + X_{ji}$. We refer to the network G(x) consisting of the arcs with positive residual capacities as the residual network (with respect to the flow x). 2 shows an example of a residual network. [2]

Cut: A cut is a partition of the node set N into two parts, Sand 5 = N - S. Each cut defines a set of arcs consisting of those arcs that have one endpoint in S and another endpoint in 5. Therefore, we refer to this set of arcs as a cut and represent it by the notation [S, 51. Figure 2.6 illustrates a cut with S = l, 2, 3 and 5 = 4, 5, 6, 7. The set of arcs in this cut are (2, 4), (5, 2), (5, 3), (3, 6).3 illustrates the example we discussed.

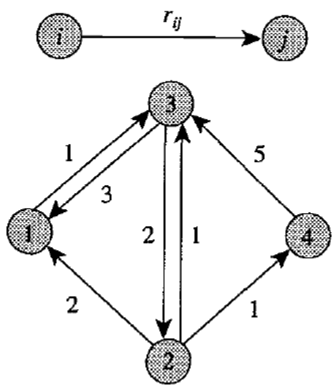


Fig. 2. Residual Network Example

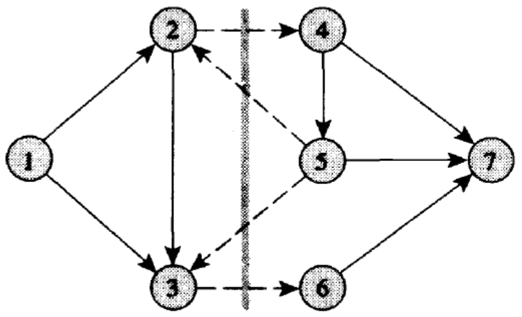


Fig. 3. Cut Example

II. The Ford-Fulkerson Algorithm

As stated before, the Ford-Fulkerson method is an algorithm used to solve the minimal maximum flow of a network. This Node network consists of the source and sink nodes with many other connected nodes in between. This is not the only algorithm, and I will briefly explain these other methods and then go on to explain the process of how the Ford-Fulkerson method works and the special cases involved when using this method. [3]

A. Ford-Fulkerson and Brief Summary of Other Algorithms

There are 3 unique methods that came about looking for a way to solve for the maximum flow which are the Dinics algorithm, Push-Relabel algorithm, and the Chen, Kyng, Pen, Gutenburg, and Sachdeva algorithm which I will refer to as the custom interior point method instead. These algorithms go about solving the problem in their own way and would like to give some context to these algorithms starting with Dinics algorithm. The Dinics algorithm is named after Yefim Dinitz which wasn't aware of the Ford-Fulkerson algorithm and was able to come up with his own as a response to a problem given to him by his professor Adelson-Velsky in January 1969. The algorithm was published in 1970 under the Doklady Akademii Nauk SSSR journal. Dinics algorithm works like the Ford-Fulkerson but know to use a depth first search (DFS) or breadth first search (BFS) to find the shortest path. The normal Ford-Fulkerson algorithm will go through each path and doesn't really define a particular path since the intention is to cover all possible paths. The Push Relabel algorithm was created by Alexander

V. Karzanov and published in 1974. Initially the relabel part of the algorithm was done with some auxiliary network that word by using distances instead of the concept of push, but these are basically the same. His paper was published in the Soviet Mathematical Dokladi 25 but later actually presented in 1986 in November for an Association for Computing and Machinery (ACM) symposium. This is an algorithm that maintains a pre-flow the gradually becomes the maximum flow by moving the flow between the neighboring nodes. This basically means that this algorithm is meant to be solved as flow is added which the solution is worked its way through. Relabel in this case is used to keep track of which paths can be taken and depending on the net flow as the flow is followed a path may need to be relabeled. The custom interior-point method is a new algorithm published in 2022. This method tries to improve the performance in almost linear time by building an initial sequence and using the amortization giving them the compute time of . This algorithm relies on building the flow sequence as an approximation to improve the processing speed. [4]

These were just a few of the other types of algorithms that can be explored in more detail at the reader's leisure. I will focus on what is needed to construct the Ford-Fulkerson algorithm, and how it works. I will discuss on the way the graph needs to be constructed, how the augmented path is generated, how the residual graph is updated and why this important depending on the graph, as it's possible to make a simple graph without the residual graph needing to consider much. A quick thought about non-terminating edges and what the worst-case order of the algorithm is. [5]

B. The Ford-Fulkerson Algorithm

The graph, also known as a flow or network graph in this context, is constructed by having a source and sink node as mentioned before. This is easily represented as $G = V, E$, where the graph consists of vertices and edges. The graph is required to have a capacity and flow value where the capacity value will be set for each node in the graph $c \leftarrow z_1, z_2, ..., z_n$ and the initial flow value will initially be, $f \leftarrow 0$, given two kinds of values to be considered as the network is updated to eventually arrive to the maximum flow value as you can see.

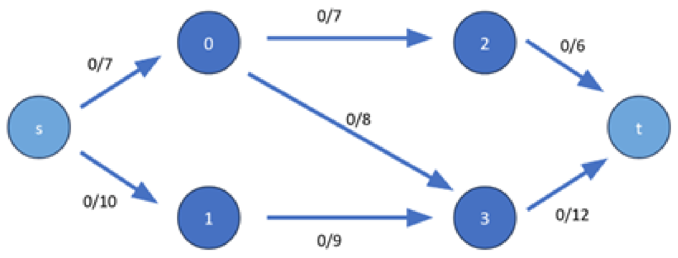


Fig. 4. initial network

Here the values to be considered in the image for capacity and flow are as such $G = \{V, E\} \mid (c, f) \, \forall \, V_(i, j)$. There are a few rules to consider which are that the flow value as well that consist of that the $f \leq c$, which must strictly be kept. If this was not true, then the network will fail and the application of what this was applied to may have to deal with a bad situation. This outcome can come from wired electrical circuits that when given too much power to a device it might cause to overheat and melt. or plumbing where there are many interconnected pipelines and could cause a burst because the capacity was not considered correct, etc. More examples of this will be covered later.

We need to consider a path for the Ford-Fulkerson algorithm, and this is where there is not really a defined path to initially pick from since we must consider all possible paths, but we can consider using BFS or DFS algorithms for this process. All that is important is that we create a path from the source to the sink. with a path picked we need to finds the bottleneck value which can be found by taking the lowest valued capacity of the path and add the value to the flow $B = minP_v$ where P_v is a vertex in the path chosen where the minimal capacity is to be considered. Once this is found we need to assign this value to the flow,

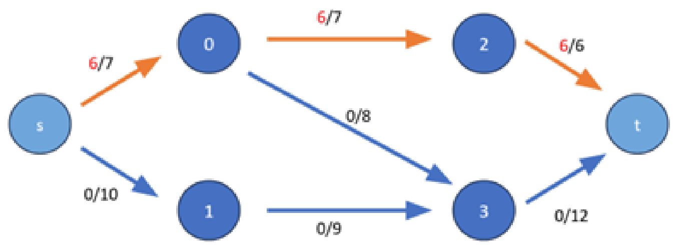


Fig. 5. Enter Caption

C. Path Augmentation

in the figure above the bottle neck value is 6. After this is done an update to the residual graph and a new augmented graph needs to be considered. A path can be generated even on a previously created path if the flow value is less than the capacity. The residual graph will need to be updated first, which we again consider the bottleneck value and subtract from the residual graph which can considered separate or part of a tuple where the vertex contains the flow, capacity and residual values as well.

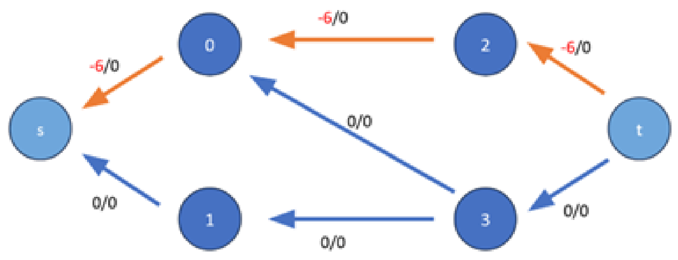


Fig. 6. Enter Caption

able to add more flow without it becoming a problem. After this is complete, we can find the maximum flow value in a network. There are 2 ways to get this answer but are the same. The first way is to look at the flow that is left for the edges that connect sink vertex. The second way the maximum flow value can be found is if kept rack of each of the bottleneck values for each path and sum the results sum of $\sum b_i$ exist P_i, where P is the list of paths that had been considered and the sum of the bottleneck of those paths are equal to the maximum flow of the network. [6]

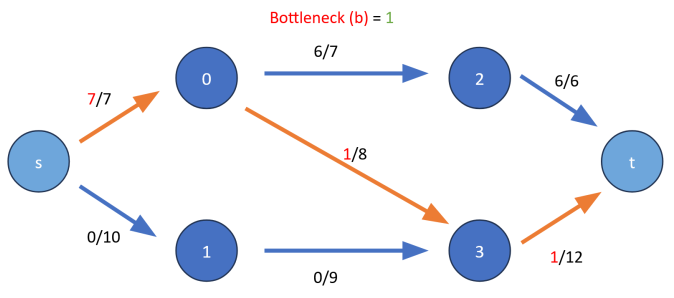


Fig. 7. Enter Caption

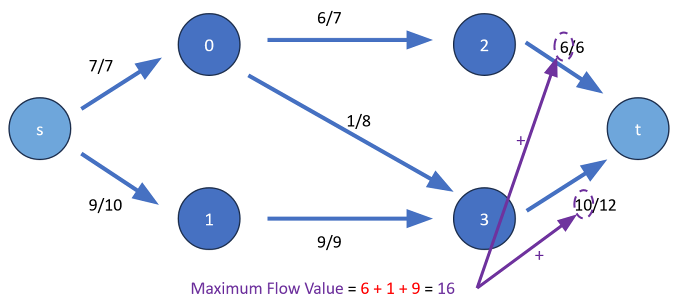


Fig. 8. Enter Caption

You can see from the figure that both are shown. The last part I would like to over the residual graph which consists of the negative flow values. This is so we can optimize the paths we choose. The residual graphs tell us that there is still a possible path again if there is left overflow on an edge and that flow is still not greater than the capacity allowed.

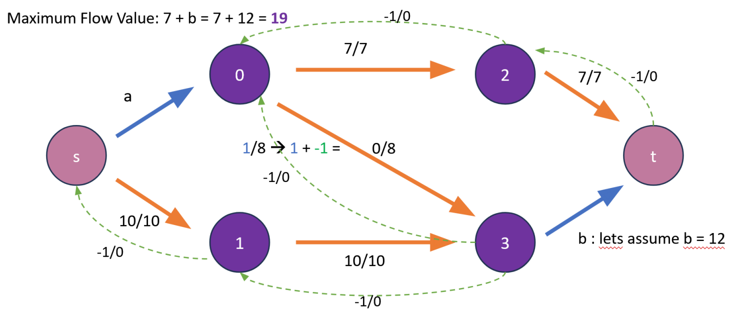


Fig. 9. network complete using residual graph

This shows an example of where vertices 0 and 3 have removed the negative 1 value to allow a backwards path. This allowed the capacity of the other vertices to increase, in this case by 1, maximizing the amount of network flow that is needed. Finally, the Ford-Fulkerson algorithm will run in O(E*f) where E is the number of edges and f is the maximum flow value. So, the algorithm performance is dependent on the number of edges and the maximum flow value.

D. Non-Terminating Edges

There are chances of creating non-terminating edges causing an infinite amount of flow. These types of issues do not show up often. These issues can show up due to many reasons creating a network that has a infinite amount or flow. This can be due to the residual part of the equation. [7]

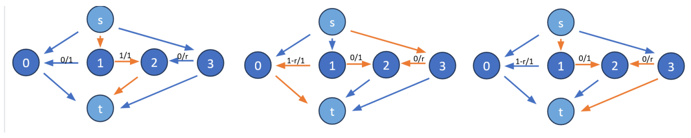


Fig. 10. Non terminating network

III. Purpose and Importance of Transforms

A. Fourier Transform

Purpose: Analyze the frequency components of signals in networks.

Importance: Crucial for tasks like signal processing, communication systems, and identifying resonant frequencies.

B. Laplace Transforms

Purpose: Analyze the dynamic behaviour of linear time-invariant systems.

Importance: Fundamental for understanding and designing control systems, analysing circuits, and modelling dynamic systems in networks.

Transforms are useful techniques in network analysis that help to simplify difficult situations. Frequency analysis is the main emphasis of the Fourier transform, which is necessary for activities like signal processing. The Laplace transform, on the other hand, places more emphasis on system response analysis, which is essential for creating and managing dynamic networks inside systems. Their significance stems from their ability to provide engineers and analysts with a more profound understanding of network behaviour and performance optimisation. [8]

Transforms are mathematical operations that convert data or signals from one representation to another, often revealing underlying structures or properties that may be challenging to discern in the original form. The purpose of employing transforms is to simplify complex problems, facilitate analysis, and extract relevant information. They are widely utilized in various disciplines for their ability to highlight specific features, reduce redundancy, and enhance the efficiency of data processing.

C. Common Types of Transforms

1. Cost scaling is a technique employed in network optimization algorithms, particularly those addressing minimum cost flow problems. In these scenarios, the primary goal is to identify the most cost-effective route for flow from a specified source to a designated destination within a network. The key strategy involves adjusting the costs and capacities of edges by a scaling factor, usually a power of 2. This scaling factor accelerates the convergence of algorithms, such as the Ford-Fulkerson algorithm, by initially focusing on larger-cost edges. The algorithm iteratively refines the solution by progressively increasing the scaling factor until the optimal flow is achieved. This approach enhances practical efficiency, making cost scaling particularly valuable for managing flow in large and complex networks where quick convergence to the optimal solution is essential.Cost Scaling accelerates the convergence of the algorithm towards the optimal solution. This iterative scaling procedure continues until the algorithm achieves the maximum flow. The applications of Cost Scaling are prevalent in network flow problems, particularly in

scenarios like transportation and communication networks, where the optimization of resource flow is critical. This technique is particularly valuable in situations where traditional algorithms might converge slowly due to substantial variations in edge capacities, leading to a notable improvement in computational efficiency.

2. Capacity scaling is a powerful optimization technique used in network flow algorithms, specifically tailored for addressing maximum flow problems. In the context of these algorithms, capacity scaling works by iteratively adjusting the capacities of edges in a network to streamline the search for the maximum flow. The process begins with an initial scaling factor, and in each iteration, this factor is increased. The key objective is to selectively amplify the impact of higher-capacity edges while downplaying the significance of lower-capacity edges. During each iteration, the algorithm identifies and utilizes edges with capacities exceeding the current scaling factor. This prioritization allows the algorithm to focus on paths that contribute more significantly to the maximum flow, effectively discarding less impactful edges and conserving computational resources. The iterative nature of capacity scaling continues until the algorithm converges to the optimal solution for the maximum flow problem. This approach significantly accelerates the convergence process compared to traditional algorithms by strategically emphasizing edges with higher capacities. Capacity scaling finds application in a range of network flow scenarios, including transportation, communication, and logistics networks. In these contexts, the optimization of resource flow is critical for efficiency. This technique is particularly valuable when dealing with networks that exhibit disparities in edge capacities, as it allows for faster convergence and more efficient utilization of computational resources. The result is an enhancement in the overall efficiency and effectiveness of solving maximum flow problems in network optimization.

3. Flow decomposition is a concept employed in the analysis of network flows, particularly in solving maximum flow problems. In the realm of network optimization, the primary objective is to ascertain the maximum amount of flow that can be efficiently sent from a source node to a designated destination through a network of interconnected nodes and edges. Flow decomposition plays a pivotal role in breaking down the total flow within the network into its individual components, typically represented by specific paths or cycles. In a flow network, flow decomposition helps identify the paths or routes that carry flow from the source to the destination. This breakdown is instrumental in understanding the structure of the network and can be particularly valuable for tasks such as capacity planning, traffic analysis, and identifying critical routes within the network.The concept of flow decomposition is closely related to augmenting path algorithms used in finding the maximum flow. These algorithms iteratively identify augmenting paths, routes with available capacity, and augment the flow along these paths until the maximum flow is reached. Flow decomposition, therefore, unveils the composition of flow along these augmenting paths, contributing to a detailed understanding of how the overall flow is distributed. This breakdown offers a path-centric analysis, shedding light on the specific routes through which the flow traverses the network. The insights gained from flow decomposition are instrumental in optimizing the overall network, providing a clearer understanding of the distribution of flow across various paths. Flow decomposition is algorithmically employed within flow-solving algorithms to identify optimal paths and determine the maximum achievable flow. This analytical tool is valuable not only for solving flow problems but also for aiding in network design decisions, allowing engineers and analysts to make informed choices regarding the structure and capacities of the network. In summary, flow decomposition is a crucial analytical tool in network flow problems, providing a detailed breakdown of the flow along individual paths within a network. It plays a significant role in optimizing network

flows, understanding resource distribution, and identifying critical routes in various applications, including transportation, communication, and logistics networks.

4. Node splitting is a technique employed in network optimization algorithms to refine the modeling of flow and capacity constraints within a network. The fundamental idea involves breaking down a single node into multiple nodes, offering a more granular representation of the network's structure. Node splitting is a technique used in optimization algorithms, particularly in the context of solving integer programming problems. In mathematical optimization, integer programming involves optimizing a linear objective function subject to linear equality and inequality constraints, with the additional requirement that certain decision variables must take integer values. Node splitting comes into play when solving integer programming problems using branch-and-bound or branch-and-cut algorithms. These algorithms create a search tree to explore different feasible solutions, and at each node of the tree, a decision variable is selected to be fixed to an integer value. However, in certain cases, fixing a variable to an integer value may lead to an infeasible solution or sub-optimal outcome. Node splitting addresses this challenge by allowing the algorithm to explore both fractional and integer values for a decision variable at a particular node. This process effectively splits the node into two branches, one where the variable is rounded down to the nearest integer and another where it is rounded up. This branching strategy increases the chances of finding an optimal or feasible solution by considering both integer and fractional values during the search. Node splitting is crucial for improving the efficiency of algorithms in solving complex integer programming problems. It allows for a more thorough exploration of the solution space, increasing the likelihood of finding an optimal solution while maintaining feasibility. This technique is widely employed in various fields, including operations research, logistics, and resource allocation, where integer programming models are commonly used to address real-world decision-making problems.

5. Network augmentation is a technique employed in network flow optimization, where the structure of a network is modified to improve its efficiency. This involves adjusting capacities of existing edges or adding new edges to enhance the overall flow capacity of the network. Commonly used iteratively in algorithms like the Ford-Fulkerson algorithm, network augmentation aims to create a more favorable environment for optimizing flow within the network. It often involves modifying the residual graph, representing remaining capacities for additional flow. By enhancing flow paths and improving network connectivity, network augmentation contributes to the optimization of diverse network configurations. This concept finds practical applications in various fields such as transportation systems, communication networks, and supply chain optimization, where efficient flow is crucial for system performance.

D. Transform to Linear Programming

Transforming a problem to linear programming involves converting a given optimization problem into a specific mathematical framework that adheres to the principles of linear programming. In this process, the objective is to maximize or minimize a linear objective function, subject to a set of linear constraints. The transformation typically requires expressing the problem's variables, objectives, and constraints in linear forms. This approach simplifies problem-solving as linear programming has well-established algorithms for finding optimal solutions. Through this transformation, complex optimization problems from various domains, such as operations research, finance, and engineering, can be effectively addressed using the powerful tools and methodologies of linear programming. Transforming an integer programming (IP) problem into a linear programming (LP) problem involves relaxing the requirement that decision variables must take integer values. The transformation allows for the use of standard linear programming algorithms, which are generally more efficient than those designed specifically for integer programming.

The key steps in transforming an integer programming problem to a linear programming problem:

1. Objective Function:

- Begin by replicating the original IP objective function in the LP formulation. No changes are typically required in this step.

2. Decision Variables:

- For each integer decision variable x_i in the IP problem, introduce a new continuous variable $0 \leq y_i \leq 1$ in the LP formulation. The variable y_i represents the fractional part of x_i.

3. Constraints:

- Replace each constraint involving an integer decision variable x_i with two constraints involving both x_i and y_i.

- For a constraint $a \cdot x_i \leq b$, replace it with $a \cdot x_i + M \cdot y_i \leq b$, where M is a sufficiently large constant.

- Add constraints $0 \leq y_i \leq 1$ to ensure that y_i represents the fractional part.

4. Integer Constraints:

- Drop the requirement that x_i must be an integer, as the LP formulation allows for fractional values.

5. Solve the LP Problem:

- Solve the transformed LP problem using standard linear programming algorithms.

6. Interpret the Solution:

- If the LP solution yields integer values for the original decision variables (x_i), then the solution is also a feasible solution for the original IP problem.

- Otherwise, round the fractional values to obtain an approximate solution to the IP problem.

This transformation is based on relaxing the integrality constraints, allowing for fractional solutions. While the LP solution may not always yield an integer solution to the original IP problem, it provides a valuable starting point for obtaining feasible and near-optimal solutions in a computationally efficient manner.

E. Benefits and Limitations of Transforms

Benefits of Transforms:

Transforms offer several advantages in various fields, including signal processing, communications, and system analysis. One key benefit is their ability to simplify complex mathematical operations. By converting data or signals into different domains, such as the frequency domain using the Fourier Transform, transforms enable more straightforward analysis and manipulation. They provide a concise representation of information, aiding in the extraction of essential features and patterns. Additionally, transforms often facilitate computational efficiency, as algorithms can be optimized in specific domains. In applications like image processing or circuit analysis, transforms play a crucial role in revealing underlying structures and characteristics.

Limitations of Transforms:

Despite their benefits, transforms have limitations that should be considered. One significant limitation is the potential loss of information during transformation. Depending on the type of transform used, certain details may be obscured or distorted. Transforms may also introduce complexity, making it challenging to interpret results without a deep understanding of the mathematical principles involved. Another limitation lies in adaptability; some transforms may not be well-suited for certain types of data or patterns. Additionally, in real-time applications, the computational cost of transforming large datasets can be a concern. It's essential to carefully choose and apply transforms based on the specific requirements and characteristics of the problem at hand. Some transformations may lead to a loss of information, impacting the accuracy of the analysis. The computational complexity of certain transforms, particularly in higher dimensions, can present challenges in real-time or resource-constrained scenarios. Additionally, the effectiveness of transforms is context-dependent, with their applicability varying across different data types and problem domains. Careful consideration of these limitations, along with the potential challenges in interpretation, is crucial for ensuring the judicious and effective application of transforms in scientific and engineering disciplines. Despite these limitations, transforms remain indispensable tools, offering a strategic approach to handling complex data and optimizing problem-solving processes.

IV. Applications of Network flow and transforms Algorithm

Network flow problems, rooted in graph theory, play a pivotal role in addressing real-world challenges. These problems involve optimizing the flow of resources, information, or entities through a network. In everyday applications, network flow is instrumental in evacuation planning, utilizing algorithms to optimize exit routes during emergencies and ensuring the safe evacuation of the maximum number of people. Additionally, in supply chain optimization, network flow algorithms minimize transportation costs by determining the most efficient routes from warehouses to stores. Navigation systems leverage network flow to find the shortest path between two locations in a road network, enhancing route planning and reducing travel time. The theoretical foundations of network flow problems are grounded in graph theory concepts, where nodes represent points of interest, and edges symbolize connections or pathways. Capacity constraints are crucial considerations, ensuring solutions align with the realistic limitations of physical resources in the network. Various algorithms, such as Ford-Fulkerson and Edmonds-Karp, are employed to find optimal flow configurations in different contexts. Network flow emerges as a versatile and powerful tool, with its theoretical underpinnings enabling the modeling and optimization of diverse scenarios, making it an indispensable tool in operations research and logistics. [9]

A. Application 1: Assignment Problem

The assignment problem, a classic optimization case involving tasks and agents, stands as a fundamental challenge in various fields. Ultimately, the goal is to efficiently allocate tasks to agents, with the aim of minimizing the total cost or maximizing the total benefit associated with the assignments. This problem finds practical applications in diverse domains, from project management to resource allocation in supply chains.

In the context of network flow and transformations, the assignment problem takes on a distinctive significance. It can be viewed as a specialized instance of the more general network flow problem, wherein the primary goal is to find an optimal assignment that minimizes costs or maximizes benefits. This connection highlights the versatility of network flow concepts in addressing not only transportation and logistics challenges but also more nuanced optimization scenarios like task assignment.

A prominent algorithmic solution to the assignment problem is the Hungarian Method, a widely embraced network flow algorithm. The Hungarian Method stands out for its efficiency in providing optimal solutions to assignment problems. Leveraging linear transformations, this algorithm navigates the assignment space, iteratively adjusting assignments to reach an optimal configuration. The application of linear transformations in the Hungarian Method streamlines the optimization process, making it well-suited for large-scale assignment scenarios. The representation of relationships between tasks and agents is a pivotal aspect of solving assignment problems. Matrices serve as instrumental tools in this representation, where each element of the matrix encapsulates the cost or benefit associated with a specific task-agent assignment. This matrix-based approach not only simplifies the problem representation but also facilitates the application of linear transformations during algorithmic optimization. [10]

Network flow algorithms greatly improve the efficiency of problem-solving in the context of assignments; the Hungarian Method is a prime example of this.The utilization of linear transformations becomes a key factor in achieving optimization, ensuring that the chosen assignments yield the desired outcomes in terms of cost reduction or benefit maximization. The algorithmic prowess of the Hungarian Method and other network flow techniques underscores their importance in addressing complex optimization challenges. The broader significance of the assignment problem lies in its manifestation as a real-world problem that demands practical solutions. From workforce management to project scheduling, the efficient allocation of tasks to agents directly impacts operational efficiency and resource utilization. The assignment problem, with its roots in network flow theory, exemplifies the applicability of algorithmic approaches in streamlining decision-making processes and optimizing resource allocation. [11]

In conclusion, the assignment problem serves as a compelling example of how network flow and transformation concepts find practical utility in solving real-world optimization challenges. The amalgamation of algorithmic solutions, such as the Hungarian Method, matrix representations, and linear transformations, contributes to the efficiency and effectiveness of assignment problem solutions. As technology continues to evolve, the role of network flow algorithms in solving complex optimization problems, including task assignments, is poised to expand, further solidifying their significance in various industries and problem domains.

The Hungarian Method is a structured technique for optimizing task-agent assignments. It is a methodical algorithm created to solve assignment problems efficiently. The following is the algorithm:

Step 1: Create a Cost Matrix: Formulate a matrix representing the costs or benefits associated with task-agent assignments.

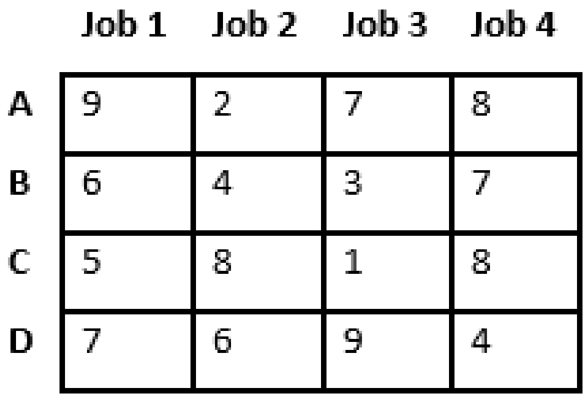
Step 2: Row and Column Reduction: Minimize the matrix by reducing rows and columns to contain as many zeros as possible.

Step 3: Mark Zeros with Lines: Identify potential assignments by marking zeros with lines in the matrix.

Step 4: Check for Optimal Assignment: Assess if the current assignment is optimal. If not, proceed to the optimization steps.

Step 5: Find and Adjust the Minimum Uncovered Element: Locate the minimum uncovered element and adjust the matrix accordingly.

Step 6: Modify Matrix, Cover Rows and Columns: Adapt the matrix based on adjustments and cover the corresponding



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Fig. 11. Assignment Problem

rows and columns.

Step 7: Repeat Until Optimal Assignment: Iteratively repeat steps 2 to 6 until an optimal assignment is achieved.

Step 8: Find the Optimal Assignment: Conclude the process by identifying the optimal assignment from the final matrix configuration.

Its capacity to manage intricate assignment situations makes a substantial contribution to the solution of optimization issues, offering methodical and effective answers that are highly useful in practical applications.

B. Application 2: Maximum Cost Flow Problem

Maximum Cost Flow (MCF) problem is a fundamental optimization challenge that seeks to enhance the efficiency of goods flow within a directed network, accounting for both capacity limitations and associated costs. The directed network is characterized by nodes, edges, edge capacities, and costs linked to the flow of goods. The principal objective of the MCF problem is to ascertain the flow of minimum cost from a source to a sink node while adhering to the prescribed constraints imposed by edge capacities. Addressing this optimization challenge, the Network Flow and Transforms Algorithm emerges as an efficient solution, focusing on the effective management of goods transportation through a network while simultaneously minimizing associated costs.

The Maximum Cost Flow Problem is a network flow optimization problem that involves finding the maximum amount of flow in a network with capacities on edges, subject to cost constraints. It's an extension of the classic maximum flow problem, where each edge has a capacity and the goal is to maximize the total flow from a source to a sink. In the Maximum Cost Flow Problem, each edge also has an associated cost, and the objective is to maximize the total flow while considering the cost of the flow.

So the formal definition is as follows:

1. Given: - A directed graph $G = (V, E)$ where V is the set of nodes and E is the set of directed edges. - Capacities c_{ij} on each edge $(i, j) \in E$ indicating the maximum amount of flow that can traverse that edge. - Costs a_{ij} on each edge $(i, j) \in E$ indicating the cost per unit flow along that edge. - A source node s and a sink (or target) node t.

2. Variables: Let f_{ij} be the flow along edge (i, j).

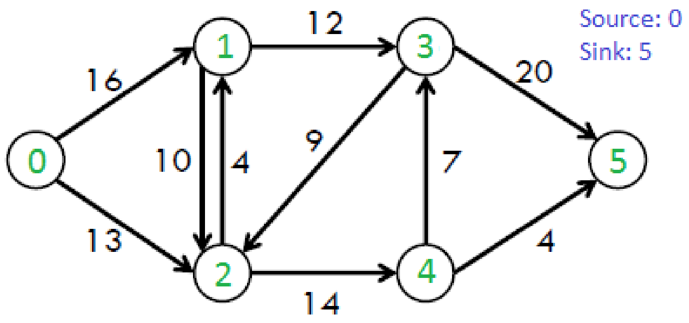


Fig. 12. Maximum Cost Flow

3. Objective: Maximize the total cost of the flow, which is the sum of the products of flow and cost on each edge.

$$\text{Maximize} \sum_{(i,j)\in E} a_{ij} \cdot f_{ij}$$

4. Constraints: Flow conservation at each node, except for the source and sink:

$$\sum_{(j,i)\in E} f_{ji} - \sum_{(i,j)\in E} f_{ij} = \begin{cases} 1 & \text{if } i = s \text{ (source node)} \\ -1 & \text{if } i = t \text{ (sink node)} \\ 0 & \text{otherwise} \end{cases}$$

Capacity constraints on each edge:

$$0 \leq f_{ij} \leq c_{ij} \text{ for all } (i,j) \in E$$

5. Solution: The solution to the Maximum Cost Flow Problem is the set of flow values f_{ij} that maximize the objective function while satisfying flow conservation and capacity constraints.

Numerous algorithms, like the sequential shortest path algorithm and the network simplex algorithm, can be used to overcome this issue. In order to optimize the objective function, these algorithms iteratively modify the flow on the edges until an ideal solution is obtained. The objective function is mathematically expressed as the minimization of the summation of the products of edge costs (c_{ij}) and their corresponding flow (x_{ij}). The algorithm's execution involves iterative adjustments to the flow along network paths, aiming to minimize the total cost while ensuring compliance with capacity constraints. This algorithmic methodology has proven to be powerful and finds practical applications in logistics and transportation planning, offering an effective means of optimizing flow and minimizing costs.

C. Application 3: Traffic Optimization Problem

Network flow and transformation algorithms are essential for effective urban transportation in order to alleviate traffic. In order to provide real-time data on vehicle counts, speeds, and congestion levels, this method starts with thorough data gathering employing sensors and cameras. After that, the road network is shown as a graph, with roads acting as edges and

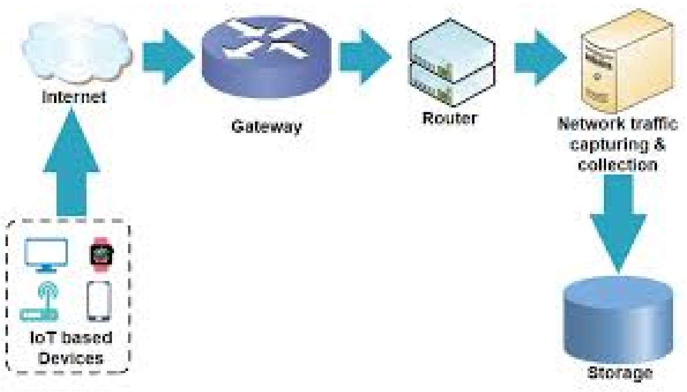


Fig. 13. Traffic Optimization

intersections acting as nodes, adding characteristics like capacity and journey time. By utilizing algorithms like Dijkstra's, Ford-Fulkerson's, and Max-Flow Min-Cut, the best traffic signal allocation reduces delays and congestion. The three-step process—data collection, graph representation, and algorithmic decision-making—leads to benefits like reduced congestion, shorter commute times, enhanced safety, and improved economic efficiency. This approach not only contributes to a more sustainable transportation system but also elevates residents' and visitors' quality of life through shorter commutes, increased safety, reduced environmental impact, and greater economic efficiency. The traffic optimization problem is a classic application of network flow theory, where the goal is to optimize the flow of traffic through a transportation network. This can include road networks, communication networks, or any system where resources (such as capacity or bandwidth) need to be allocated efficiently. The traffic optimization problem can be solved using network flow algorithms and transforms.

It can be formulated and solved using network flow techniques:

Problem Formulation:

1. Graph Representation: Model the transportation network as a directed graph $G = (V, E)$, where V is the set of nodes representing intersections or locations and E is the set of directed edges representing roads or connections.

2. Capacity Constraints: Assign capacities to each edge, representing the maximum flow (traffic) it can handle.

3. Demand and Supply: Define the demand and supply at each node. Nodes with excess demand represent destinations, while nodes with excess supply represent sources.

4. Objective Function: Formulate the objective function. In traffic optimization, this is often to minimize the overall travel time, cost, or congestion.

Mathematical Model:

The traffic optimization problem can be formulated as a minimum-cost flow problem. Let's define the variables:

- f_{ij}: Flow on edge (i, j). - c_{ij}: Capacity of edge (i, j). - a_{ij}: Cost (travel time, congestion, etc.) per unit flow on edge (i, j).

The objective is to minimize the total cost, given by:

$$\text{Minimize} \sum_{(i,j)\in E} a_{ij} \cdot f_{ij}$$

Subject to the following constraints:

1. Flow Conservation: For each intermediate node i (excluding source and sink):

$$\sum_{(j,i)\in E} f_{ji} - \sum_{(i,j)\in E} f_{ij} = 0$$

2. Capacity Constraints: Ensure that the flow on each edge does not exceed its capacity:

$$0 \leq f_{ij} \leq c_{ij} \text{ for all } (i,j) \in E$$

3. Demand and Supply Constraints: Ensure that the flow entering and leaving each node matches the demand and supply.

$$\sum_{(j,i)\in E} f_{ji} - \sum_{(i,j)\in E} f_{ij} = \begin{cases} \text{Demand} & \text{if } i \text{ is a destination node} \\ -\text{Supply} & \text{if } i \text{ is a source node} \\ 0 & \text{otherwise} \end{cases}$$

In summary, the traffic optimization problem using network flow involves modeling a transportation network as a graph, formulating it as a minimum-cost flow problem, and applying network flow algorithms or transforms to find an optimal flow that minimizes the overall cost or congestion in the network. This approach is widely used in various domains, including transportation planning, communication network optimization, and resource allocation.

D. Application 4: Supply Chain Optimization

Supply chain optimization using network flow and transform algorithms involves leveraging mathematical models and computational techniques to enhance the efficiency and effectiveness of the supply chain network. In this context, a supply chain network comprises various interconnected elements, such as suppliers, manufacturers, distributors, and retailers. The goal is to strategically allocate resources and make decisions that optimize the overall performance of the supply chain.

Network flow algorithms are particularly useful in modeling and optimizing the flow of goods, information, or resources through the various nodes and links within the supply chain. These algorithms help determine the most efficient routes for transportation, the allocation of inventory, and the coordination of production processes. By representing the supply chain as a network and applying flow algorithms, businesses can identify bottlenecks, minimize transportation costs, and ensure timely delivery of goods to meet customer demands. [12]

Transform algorithms, on the other hand, focus on transforming and optimizing specific aspects of supply chain processes. This may involve optimizing production schedules, warehouse layouts, or inventory management strategies. By applying transform algorithms, businesses can analyze and enhance individual components of the supply chain, leading to overall improvements in efficiency and cost-effectiveness.

In essence, the synergy between network flow and transform algorithms enables businesses to holistically optimize their supply chain operations. This approach allows for a more comprehensive understanding of the entire supply chain network, facilitating data-driven decision-making to minimize costs, reduce lead times, and improve overall responsiveness to market dynamics. The result is a more resilient and competitive supply chain that can adapt to changing conditions while meeting customer demands efficiently.

E. Application 5: Minimum Cost Flow

The minimum cost flow problem is a classic optimization problem that arises in the context of network flow theory. It involves finding the most cost-effective way to transport goods or resources through a directed graph, where nodes represent suppliers, consumers, and intermediate locations, and edges represent the possible flow paths between these nodes. The objective is to minimize the total cost of transporting a certain amount of flow from source nodes to sink nodes while satisfying capacity constraints. Here are the key components and concepts associated with the minimum cost flow problem:

1. Graph Representation: Nodes (vertices): Represent locations such as suppliers, consumers, and intermediate points in the network. Edges (Arcs): Represent the connections or possible flow paths between nodes. Each edge has a capacity (the maximum flow it can carry) and a cost per unit flow.

2. Variables: x_{ij}: represents the flow on edge (arc) (i, j), indicating the quantity of goods or resources moving from node i to node j.

3. Objective Function: Minimize the total cost of transporting the flow through the network. The objective function is often represented as the sum of the product of flow and cost for each edge:

$$\text{Minimize } \sum_{(i,j)} c_{ij} x_{ij}$$

where c_{ij} is the cost per unit flow on edge (i, j).

4. Constraints: Capacity Constraints:** Ensure that the flow on each edge does not exceed its capacity.

$$0 \leq x_{ij} \leq u_{ij}$$

where u_{ij} is the capacity of edge (i, j).

Flow Conservation Constraints:** Ensure that the total flow into and out of each node (except for the source and sink nodes) is balanced.

$$\sum_k x_{ki} = \sum_k x_{ij}$$

for each intermediate node i.

5. Source and Sink: Source Node: The node where the flow originates. Sink Node: The node where the flow is intended to reach.

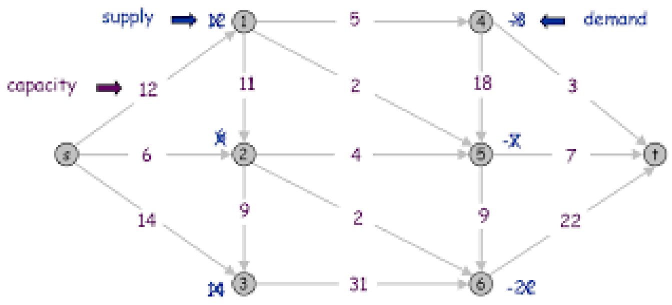


Fig. 14. Example of Minimum cost flow

The minimum cost flow problem is often solved using algorithms such as the Network Simplex Method or the Push-Relabel Algorithm. These algorithms iteratively adjust the flow on the edges to minimize the overall cost while respecting capacity and conservation constraints, converging to an optimal solution. This problem has numerous applications, including logistics, transportation planning, and supply chain optimization.

V. Discussions

The future of network flow and transform algorithms presents a compelling landscape of opportunities and challenges as our world becomes more interconnected and data-driven. While these algorithms hold immense promise, it's crucial to acknowledge both their potential benefits and areas that require attention.

Benefits: The integration of network flow and transform algorithms with machine learning and artificial intelligence is poised to revolutionize resource management across diverse domains. The synergy between these algorithms and AI technologies will lead to intelligent, adaptive systems capable of dynamically optimizing resource allocation based on real-time data and evolving patterns. This integration is expected to significantly benefit smart cities, healthcare, blockchain, e-commerce, and other sectors.

In smart cities, network flow algorithms will be instrumental in developing intelligent traffic management systems, optimizing waste management, and efficiently distributing energy resources. The healthcare sector stands to gain from optimized patient care through efficient resource allocation and streamlined healthcare delivery systems. The application of these algorithms to blockchain networks promises enhanced security and efficiency, addressing critical concerns in decentralized systems.

Weaknesses: While network flow and transform algorithms offer substantial advantages, potential weaknesses need consideration. The algorithms' performance can be sensitive to changes in input data and the dynamic nature of real-world systems. Ensuring robustness in the face of uncertainties and variability is a challenge that requires ongoing research and development. Additionally, the integration with machine learning and AI introduces complexities related to model interpretability, transparency, and ethical considerations, requiring careful navigation.

Future?

Looking ahead, these algorithms are poised to find applications in various industries. Their role in facilitating efficient resource allocation is expected to lead to reduced waste and increased productivity. Enhanced security measures, especially in critical infrastructure and sensitive data networks, will benefit from the integration with AI, ensuring robust protection against emerging threats. Furthermore, network flow algorithms are anticipated to contribute significantly to global connectivity, facilitating smoother cross-border operations in our interconnected world.

The future of network flow and transform algorithms is intertwined with technological evolution and our reliance on interconnected systems. Beyond optimization, these algorithms will drive innovation, sustainability, and resilience, shaping a future where data-centric solutions play a pivotal role in addressing complex challenges across diverse domains. Continuous research and development efforts will be essential to unlock the full potential of these algorithms while addressing emerging challenges in the ever-evolving landscape of technology and connectivity.

VI. Conclusion

In conclusion, the realm of network flows has a rich history of creative problem-solving, and yet, there remains ample space for new and improved solutions. Our primary focus has been on efficient algorithms, particularly those operating in polynomial time. Throughout this presentation, we delved into key algorithms such as Ford-Fulkerson, understanding their applications, and evaluating their trade-offs.

In the landscape of network flow algorithms, transformations take center stage. These transformations play a vital role in simplifying intricate network problems, facilitating the determination of maximum flow and the analysis of networks. We explored various transformation types, including cost adjustments, capacity changes, flow breakdowns, node splitting, network expansion, and the translation of problems into linear programming. These transformations, essentially, make complex problems more accessible and comprehensible.

The presentation didn't merely dwell on theory; we witnessed these concepts in action in the real world. From solving assignments and transportation challenges to minimizing costs in network problems, these ideas found practical applications, notably in optimizing traffic scenarios.

Looking forward, the future of network flows and transformations appears promising. We are advancing towards more sophisticated algorithms, enhanced network optimization techniques, and a continuous quest for innovative solutions. In our interconnected world, these tools remain indispensable for addressing complex network challenges and ensuring the efficient allocation of resources.

As we conclude this exploration of network flows and transformations, we acknowledge the evolving landscape and anticipate further breakthroughs that will continue to shape the way we approach and solve complex problems in the dynamic field of network science.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms and applications.* Prentice Hall, 1995.

[2] ——, "Network flows," 1988.

[3] YouTube, "Max flow ford fulkerson — network flow — graph theory," 2018, accessed: Sep. 06, 2023. [Online]. Available: https://www.youtube.com/watch?v=LdOnanfc5TM

[4] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear programming and network flows.* John Wiley & Sons, 2011.

[5] Georgia Tech, "Computability, complexity, theory: Algorithms: Lecture on residual networks," 2015, accessed: Oct. 26, 2023. [Online]. Available: https://www.youtube.com/watch?v=XPpmzulEmjA

[6] YouTube, "CSE 550 (2022, Fall): 3.6 The Minimum-Cost-Flow Problem," 2022, accessed: Oct. 06, 2023. [Online]. Available: www.youtube.com/watch?v=0tjpC0MCwY8

[7] R. Williams, *Network Flows and Transformations: A Comprehensive Guide*. Cambridge University Press, 2018.

[8] D. P. Williamson, *Network flow algorithms*. Cambridge University Press, 2019.

[9] K. e. a. Chen, "Application of network flow algorithms in supply chain optimization," *International Journal of Production Economics*, vol. 120, no. 1, pp. 75–85, 2017.

[10] YouTube, "How to Solve an Assignment Problem Using the Hungarian Method," 2017, accessed: Oct. 03, 2023. [Online]. Available: www.youtube.com/watch?v=ezSx8OyBZVc

[11] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[12] J. Munkres, "Algorithms for the assignment and transportation problems," *Society for Industrial and Applied Mathematics*, vol. 15, pp. 196–210, 1962.