MAXIMUM FLOW WITH HETEROGENOUS EDGE COST

A Project

Presented

to the Faculty of

California State University, Dominguez Hills

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Hieu Nguyen

Spring 2026

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents, Binh Nguyen and Huong Trinh, for their love, sacrifice, financial support, and constant encouragement throughout my time pursuing this degree in the United States. I am also deeply grateful to my sister, Linh Nguyen, and my brother-in-law, Vu Tran, for their support and for always being there for me during this journey.

I would like to sincerely thank Professor Bin Tang for his guidance, kindness, and wisdom throughout this project. His support, patience, and dedication to student learning made a great difference in my academic experience and helped me complete this work.

I would also like to thank California State University, Dominguez Hills, for providing a welcoming environment where I was able to grow as an international student, explore new ideas, and continue learning both academically and personally.

Finally, I am grateful to everyone who supported my personal and professional development along the way.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This project studies the problem of preserving sensor data in base station-less sensor networks when different data generators consume different amounts of edge capacity (energy) during transmission. The problem is modeled as maximum flow with heterogenous edge cost. A sensor network generator, a flow network converter, visualization tools, and several algorithm implementations were developed in Java. In addition, an integer linear programming (ILP) model was implemented in Python with Gurobi to provide an optimal benchmark for comparison, analyze and improvement. The project evaluates the paper's greedy maximum flow algorithm against the ILP solution on randomly generated network instances. The results show that the algorithm often performs close to the optimal solution, but greedy routing decisions can still block later flows and reduce total preserved data. A residual-based variation was also explored to improve rerouting behavior. The project provides both an implementation framework and an empirical analysis of the strengths and limitations of maximum flow methods under heterogenous transfer cost.

CHAPTER 1

INTRODUCTION

Wireless sensor networks are widely used in environments where direct human access is difficult, dangerous, or impractical. Examples include remote environmental monitoring, disaster areas, underwater sensing, and other harsh locations where data must be collected continuously over long periods of time. In many such settings, a permanent base station is not available to gather data immediately after it is produced. Instead, the data must be preserved within the network until it can be collected later by a mobile device such as a drone.

In a base station-less sensor network, nodes may serve different roles. Some nodes act as data generators, producing data that must be preserved. Some act as storage nodes, receiving and storing data for later collection. Other nodes serve as transshipment nodes, relaying data between generators and storage locations. Since all nodes have limited energy and storage capacity, preserving all generated data is not always possible. As a result, the network must use its available resources carefully.

This project studies the problem of data preservation in a sensor network under heterogeneous transmission cost. More specifically, it focuses on maximum flow with heterogeneous edge cost, where one unit of data generated by different source nodes may consume different amounts of edge capacity when routed through the network. This makes the problem more difficult than a standard maximum flow problem, since the remaining capacity of an edge depends not only on how much flow has been sent, but also on which source that flow originated from.

The motivation for this work is both practical and theoretical. From the practical side, sensor networks are often deployed to collect valuable information in conditions where data loss

may reduce the usefulness of the system. From the theoretical side, source-dependent flow cost extends the classical flow model and introduces new challenges in routing and capacity allocation. A greedy choice that appears efficient for one source may reduce the total amount of data preserved later in the routing process.

To study this problem, a complete experimental pipeline was developed. A sensor network generator was implemented to create random network instances with user-defined parameters. These instances were then converted into flow networks that represent node energy, data amount, and storage capacity through directed edges. A maximum flow algorithm based on the heterogeneous-cost model was implemented in Java. In addition, an integer linear programming formulation was implemented in Python with Gurobi in order to compute an exact benchmark for comparison on size-limited instances.

The main purpose of this project is to evaluate how closely the implemented maximum flow algorithm matches the optimal ILP solution. This comparison helps answer an important question: whether the algorithm provides a sufficiently accurate solution in practice, or whether its greedy decisions cause meaningful loss of preserved data. This evaluation also serves as a foundation for later work on priority-based preservation, where the objective is not only to maximize the amount of preserved data, but also to maximize the value of the preserved data.

This project makes three main contributions. First, it develops a complete software framework for studying heterogeneous-cost flow in sensor networks, including generation, conversion, visualization, and execution tools. Second, it formulates and implements an ILP model that can serve as an exact benchmark. Third, it presents an empirical comparison between the algorithmic and optimal solutions and analyzes the cases in which they differ.

The remainder of this report is organized as follows. Chapter 2 reviews related work in network flow, sensor-network data preservation, and optimization methods. Chapter 3 presents the problem formulation. Chapter 4 explains the network generation and conversion process. Chapter 5 describes the algorithms and implementation details. Chapter 6 presents the ILP benchmark formulation. Chapter 7 explains the experimental setup. Chapter 8 discusses the results and analysis. Chapter 9 concludes the report and outlines future work.

CHAPTER 2

RELATED WORK

The work presented in this project is related to three main research areas: classical network flow, data preservation in sensor networks, and optimization methods for constrained routing problems. Together, these areas provide the background needed to understand the heterogeneous-cost flow model studied in this report.

Classical Network Flow

The maximum flow problem is one of the most fundamental problems in graph algorithms and combinatorial optimization. In its standard form, a directed graph is given with a source node, a sink node, and a capacity on each edge. The goal is to maximize the amount of flow that can be sent from the source to the sink without violating edge capacities. Classical algorithms such as Ford-Fulkerson and Edmonds-Karp solve this problem by repeatedly finding augmenting paths in the residual graph until no additional feasible path can be found.

These methods provide the foundation for many routing and resource-allocation problems. However, the standard maximum flow model assumes that one unit of flow consumes one unit of edge capacity regardless of where the flow comes from. That assumption does not hold in the problem studied here. In this project, data generated by different source nodes may consume different amounts of capacity on the same edge. This requires a modified model and changes the behavior of residual capacity.

Flow Problems with Source-Dependent Cost

Many real-world networks involve flows that are not identical. In some settings, different commodities compete for the same capacity. In other cases, different flows have different values or different resource costs. These extensions show that ordinary maximum flow is not always enough when the source of the flow matters.

The heterogeneous-cost model studied in this project belongs to that broader class of source-dependent flow problems. In this model, one source may use edge capacity more efficiently than another. As a result, the order in which sources are processed can affect the final solution. A route that is acceptable for one source may leave too little capacity for a later source, even if a better overall arrangement exists.

This source-dependent structure is one of the main reasons why an exact benchmark is needed. Without an exact reference point, it is difficult to determine whether an algorithm is making good use of the network or simply following a locally convenient but globally suboptimal routing pattern.

Data Preservation in Sensor Networks

Wireless sensor networks differ from ordinary communication networks because the network is often responsible not only for transmitting data, but also for preserving it until collection becomes possible. In base station-less settings, data may remain inside the network for a significant amount of time. During this period, sensor nodes must spend energy to forward and store data, and storage nodes must reserve space for later collection.

This creates a combined routing and storage problem. A source node must decide where its data can be stored, and the network must determine how to use limited energy and limited storage capacity most effectively. In many cases, preserving one source’s data reduces the ability

of the network to preserve data from another source. This makes flow assignment an important part of data-preservation design.

The sensor-network model used in this project reflects these constraints directly. Each generator has a limited amount of data. Each storage node has limited available capacity. Each sensor node has limited energy. These constraints are then transformed into a flow-network representation so that preservation can be analyzed using graph-based optimization methods.

Greedy Algorithms and Their Limitations

A natural approach to the heterogeneous-cost problem is to process lower-cost sources first, since they consume less capacity per unit of preserved data. This idea motivates the algorithm studied in this project. The algorithm sorts source nodes by cost and processes them one at a time, finding the maximum feasible flow for each source before moving to the next one.

This strategy is simple and often effective. However, greedy algorithms may suffer from blocking effects. A low-cost source may consume a route or storage path that is more important for a later source. When that happens, the final preserved flow may be smaller than the optimal solution. Therefore, while a greedy algorithm may be attractive from an implementation point of view, it must still be validated carefully.

This issue is especially important in the present work because the project does not assume in advance that the algorithm is always optimal. Instead, the algorithm is treated as a practical method whose accuracy must be measured empirically.

Integer Linear Programming as Benchmark

Integer linear programming is a useful way to compute an exact benchmark for constrained flow problems. In the present setting, an ILP can explicitly represent source-specific

flow on each edge, enforce conservation constraints, and apply capacity constraints exactly. This allows the model to capture the heterogeneous-cost structure in a way that standard maximum flow algorithms do not.

In this project, the ILP plays a central role in evaluation. It is not used because it is faster than the algorithmic approach, but because it provides the optimal answer on instances small enough to fit the solver restriction. The comparison between the implemented algorithm and the ILP solution makes it possible to quantify both agreement and loss.

The use of an exact benchmark also improves the credibility of the evaluation. Rather than only reporting the algorithm's output, the project can show how close that output is to the optimum and can identify the conditions under which the algorithm fails to match it.

Current State of the Project

This project builds on these areas by focusing on implementation and evaluation. Its main goal is not only to model the heterogeneous-cost flow problem, but also to test how well a practical algorithm performs on randomly generated sensor-network instances. To support this goal, the project combines graph generation, flow-network conversion, algorithm implementation, visualization, and exact optimization into one workflow.

The result is an implementation-based study of heterogeneous-cost data preservation in sensor networks. It provides a way to move from the theoretical problem definition to concrete experimental evidence, while also preparing the groundwork for future extensions such as value-based or priority-based preservation.

CHAPTER 3

PROBLEM FORMULATION

This chapter defines the network model and the preservation problem studied in this project. The focus is on maximum flow with heterogeneous edge cost in a base station-less sensor network. The main objective is to preserve as much data as possible while respecting the limits on node energy, storage capacity, and source-dependent transmission cost.

Sensor Network Model

The original sensor network is modeled as a graph in which each node represents a sensor. The nodes are divided into three groups:

- The first group is the set of data generator nodes. These nodes contain data that must be preserved. Each data generator has a finite amount of data available for transfer.
- The second group is the set of storage nodes. These nodes can receive and store data produced elsewhere in the network. Each storage node has a limited storage capacity.
- The third group is the set of transshipment nodes. These nodes do not generate data and do not provide storage, but they may relay data between other nodes.

A communication link exists between two nodes if they are within the selected transmission range. In the generated network, these links are based on node positions in a two-dimensional field. If two nodes are close enough to communicate, an edge is created between them.

Each node also has a limited energy level. Since data forwarding consumes resources, node energy must be included in the preservation model. In this project, node energy is represented later through the conversion to a flow network.

Data, Storage, and Energy Constraints

Three physical constraints define the preservation problem.

First, each data generator can send at most the amount of data it owns. If a data generator contains 100 units of data, then the total outgoing preserved flow associated with that source cannot exceed 100.

Second, each storage node can receive at most its available storage capacity. If a storage node can store 150 units, then the total incoming preserved data assigned to that node cannot exceed 150.

Third, each node has a limited energy level. In the converted model, this becomes an internal capacity constraint. As more data passes through the node, more of that capacity is consumed.

Taken together, these constraints determine whether a preservation plan is feasible. A solution that exceeds the source data, storage capacity, or node energy of the network is not valid.

Heterogeneous Edge Cost

The main feature that distinguishes this problem from ordinary maximum flow is heterogeneous edge cost. In a standard flow model, one unit of flow consumes one unit of edge capacity no matter which source it comes from. In the present model, this is not true.

Each data generator is assigned a source-specific cost. This cost represents how much edge capacity is consumed by one unit of data generated by that source. If source A has cost 2 and source B has cost 5, then one unit of flow from source B uses more of the network's capacity than one unit of flow from source A.

This source-dependent behavior creates a trade-off. A low-cost source may preserve more data with the same residual capacity, but assigning too much flow to low-cost sources can also block later routes that would have improved the total preserved flow.

Because of this, feasible flow is not defined only by flow amount. It is defined by flow amount together with source origin.

Objective of the Project

The primary objective in this project is to maximize the total amount of preserved data. In other words, the goal is to find the largest feasible flow from the set of data generators to the set of storage nodes under the constraints of source data, storage capacity, node energy, and heterogeneous edge usage.

This objective is appropriate for the current stage of the project because it allows the implemented algorithm to be compared directly with an exact benchmark. Later, the same framework can be extended to a weighted objective in which the network seeks to maximize preserved value or priority rather than only preserved amount.

Feasible Preservation Flow

A feasible preservation flow in this project must satisfy the following conditions.

First, the total amount of preserved data sent by each source cannot exceed that source's available data.

Second, the total amount of data arriving at each storage node cannot exceed that storage node's capacity.

Third, flow must be conserved at intermediate nodes. For any node that is not the super source or super sink in the converted network, the amount of incoming flow must equal the amount of outgoing flow.

Fourth, internal edge capacities must not be exceeded. Since these edges model node energy, any excess would represent an impossible routing plan.

Fifth, communication edges must also respect their effective capacity under the heterogeneous-cost rule. If the total weighted use of an edge exceeds its available capacity, the solution is infeasible.

These conditions define the set of valid solutions examined by both the algorithmic and ILP methods.

Why a Flow-Network Conversion is Needed

The original sensor network is not directly in a form that standard flow algorithms can use. Node energy, source data, and storage capacity appear naturally as node-level properties rather than edge capacities. To apply graph-based optimization methods, the network must first be transformed into a directed flow network.

This conversion is one of the key design steps in the project. It makes it possible to express the preservation problem in a unified way and allows the same converted instance to be used by both the maximum flow algorithm and the ILP benchmark.

Summary

The problem studied in this project is a constrained preservation problem in a base station-less sensor network. The network includes data generators, storage nodes, and transshipment nodes. Each node has limited resources, and each source has its own transmission cost. The goal is to preserve as much data as possible without violating data, storage, or energy limits.

This formulation provides the basis for the network conversion described in the next chapter and for the algorithmic and optimization methods presented later in the report.

CHAPTER 4

SENSOR NETWORK GENERATION AND FLOW NETWORK CONVERSION

This chapter explains how sensor-network instances are created and how they are converted into flow networks. The generation step provides the experimental input for the project. The conversion step turns each generated sensor network into a form that can be used by the implemented algorithm and the ILP benchmark.

Sensor Network Generation

A random sensor-network generator was implemented in Java to create input instances for testing. The generator accepts user-defined parameters and produces a network in a two-dimensional rectangular field.

The main input parameters are the width and length of the field, the total number of nodes, the number of data generator nodes, the number of storage nodes, the range of data amount, the range of storage capacity, the range of node energy, the range of source cost, the range of priority values, and the communication range. The remaining nodes are assigned as transshipment nodes.

Each generated node is placed at a random coordinate within the field. A subset of nodes is then selected as data generators, another subset is selected as storage nodes, and the rest become transshipment nodes. Every node is given an energy value. Each data generator is assigned a data amount, a source cost, and a priority value. Each storage node is assigned a storage capacity.

After node creation, communication links are generated based on distance. If two nodes are within the selected communication range, they are treated as connected. This produces the original sensor-network graph.

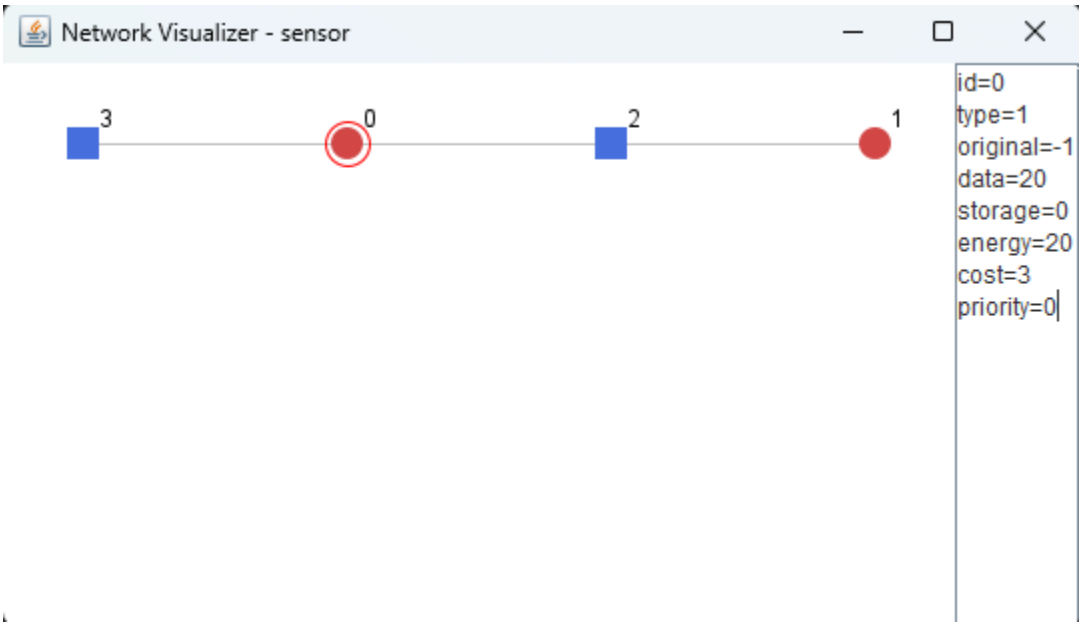

Figure 1. Sensor network

The generator exports the instance to a text file so it can be reused by later programs. It also supports visualization so that the generated network can be inspected manually.

Purpose of the Generator

The purpose of the generator is to create controlled experimental cases. Because the project focuses on algorithm evaluation rather than on a fixed public data set, synthetic generation makes it possible to test many parameter combinations systematically.

This approach also allows repeated testing under the same settings with different random seeds. That is important for evaluation, since one parameter setting may lead to several different network structures depending on how nodes are placed.

In the final evaluation, groups of random instances with the same parameters can be used to compute average performance and confidence intervals.

Sensor Network File Contents

Each generated sensor-network file contains the information needed for later stages of the project. In general, the file records:

- the number of nodes and edges,
- the number of data generator and storage nodes,
- the role of each node,
- the coordinates of each node,
- the data amount at each data generator,
- the storage capacity at each storage node,
- the energy value of each node,
- the source cost of each data generator,
- the adjacency information of the network.

The file format was designed to remain numeric and simple so that it can be used directly by the conversion program and by later evaluation scripts.

Need for Flow-Network Conversion

Although the generated sensor network describes the real problem structure, it is not directly suitable for standard flow algorithms. In particular, node energy is a node property, not an edge capacity. Storage is also associated with specific nodes. To solve the preservation

problem with flow methods, these constraints must be embedded into a directed network with explicit capacities.

For this reason, a conversion program was implemented to transform the original sensor network into a flow network. This converted network is the common input used by both the heterogeneous-cost algorithm and the ILP benchmark.

Node Splitting

The first step of the conversion is node splitting. Each original sensor node i is replaced by two nodes in the flow network, denoted i-prime and i-double-prime.

An internal directed edge is added from i-prime to i-double-prime. The capacity of this edge is set equal to the energy value of the original node.

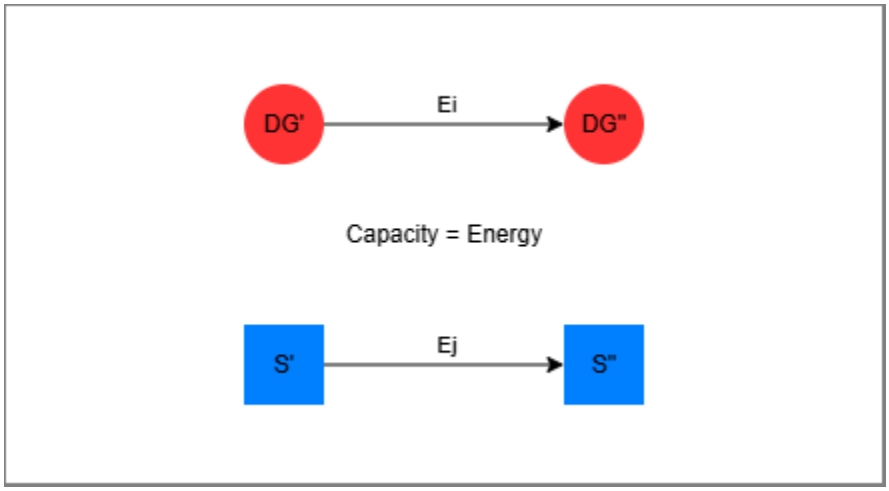

Figure 2. Node Splitting

This transformation is important because it converts node energy into edge capacity. As data passes through the split representation of a node, it consumes the capacity of the internal edge. In this way, the energy limit of the original sensor is enforced by the flow network.

Source and Sink Construction

After node splitting, a super source and a super sink are added to the flow network.

For each data generator node, an edge is added from the super source to the generator's prime node. The capacity of this edge is set equal to the amount of data held by that data generator. This ensures that the source cannot send more data than it owns.

For each storage node, an edge is added from the storage node's double-prime representation to the super sink. The capacity of this edge is set equal to the amount of storage available at that node. This ensures that the storage node cannot receive more data than it can hold.

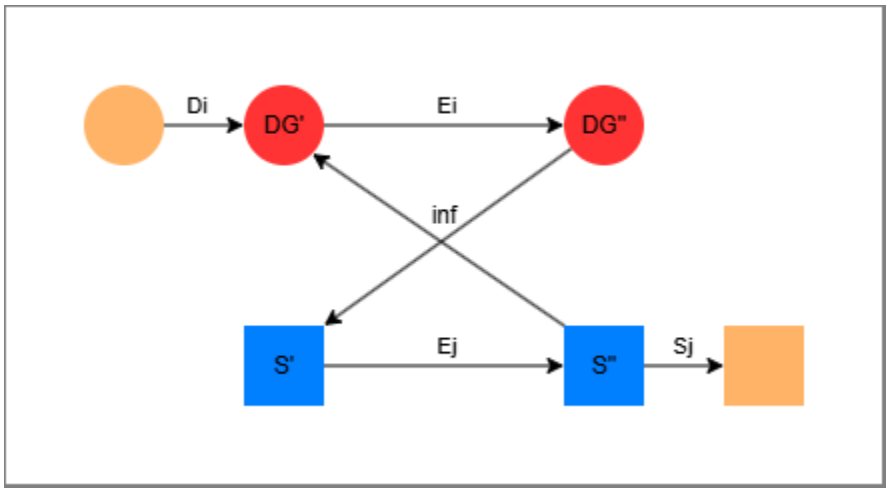

Figure 3. Super source and sink construction

Together, these edges represent the source and storage constraints in the converted network.

Conversion of Communication Links

The next step is to convert the communication structure of the original sensor network.

If two original nodes u and v are connected in the sensor network, then the converted flow network includes directed routing edges between their split-node representations. In this project, an original communication link becomes two directed routing edges: one from u-double-prime to v-prime and one from v-double-prime to u-prime.

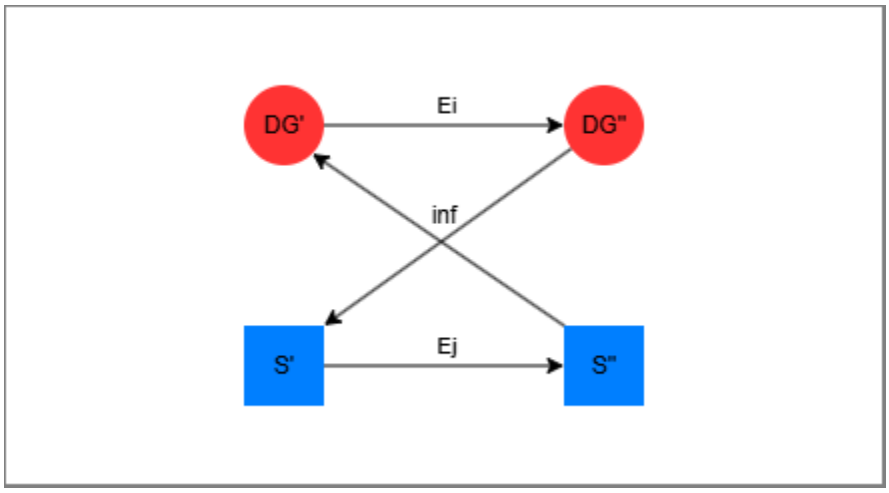

Figure 4. Communication link

These edges represent the possibility of forwarding data between the two original nodes. Their capacities are set to infinite (or a very large value in implementation) so that the actual limiting factors remain the node-energy edges, the source-data edges, and the storage-capacity edges.

Resulting Flow Network

After conversion, the flow network contains the following components:

- a super source,
- a super sink,
- a pair of split nodes for every original node,
- internal energy edges from prime to double-prime,

- source edges from the super source to generator prime nodes,
- storage edges from storage double-prime nodes to the super sink,
- and routing edges derived from the original communication links.

This structure allows preservation to be analyzed using flow methods. Source data, node energy, and storage capacity are all represented as edge capacities in one directed network.

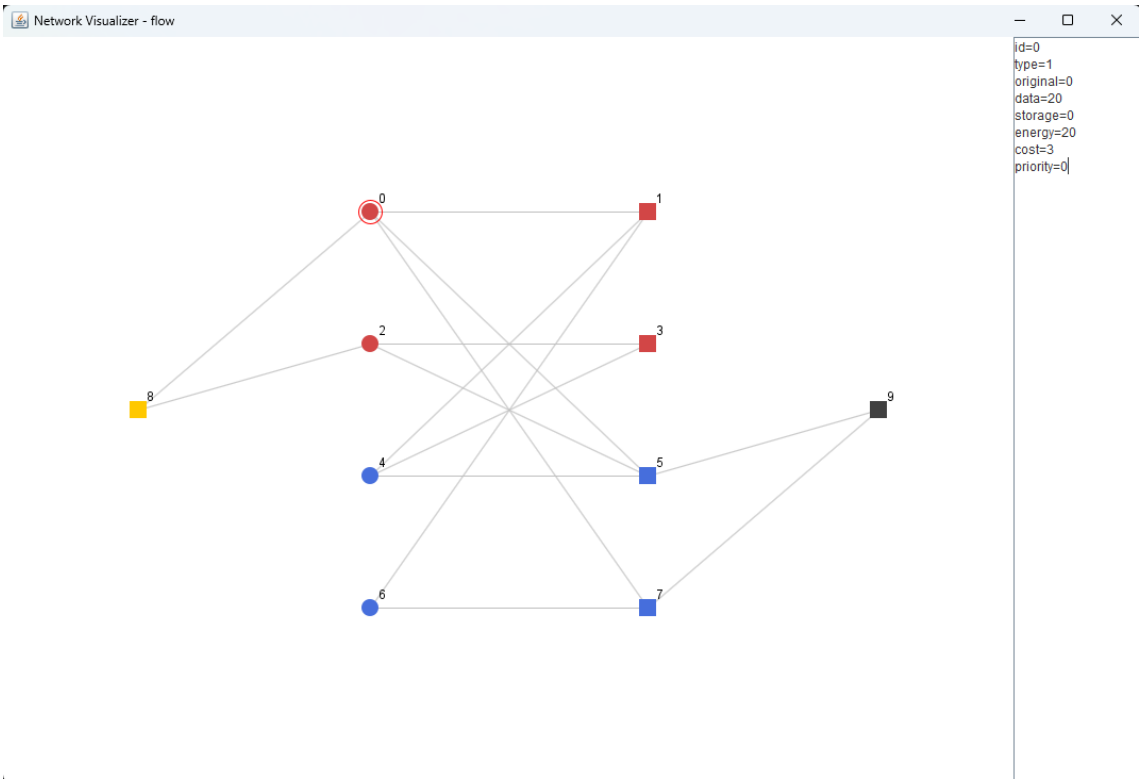

Figure 5. Converted flow network

Use in the Project Pipeline

The converted flow network is the central input to the later stages of the project.

The maximum flow algorithm reads the converted network and computes a preservation solution under heterogeneous source cost. The ILP model reads the same converted network and computes an exact benchmark solution. Since both methods use the same converted instance, their outputs can be compared directly.

A separate visualization tool was also developed so that saved sensor-network and flow-network files can be displayed without rerunning the generator. This was useful during debugging and when examining cases where the algorithm and ILP did not agree.

Summary

The generation and conversion stages provide the foundation for the entire project. The generator creates reproducible random sensor-network instances under user-controlled parameters. The conversion program then transforms each instance into a directed flow network in which node energy, source data, and storage capacity are represented by edge capacities.

This converted network makes it possible to apply both the heterogeneous-cost maximum flow algorithm and the exact ILP benchmark to the same problem instance. The next chapter describes the algorithmic methods and implementation details built on top of this converted model.

CHAPTER 5

ALGORITHMS AND IMPLEMENTATION

This chapter describes the algorithms implemented in this project and explains how the software was organized. The main focus is the heterogeneous-cost maximum flow algorithm and the integer linear programming benchmark used for comparison. In addition, this chapter summarizes the generator, converter, visualizer, and experiment scripts that make up the full project pipeline.

Overview of the Implementation Pipeline

The project was implemented as a sequence of separate programs so that each stage could be tested independently. The overall pipeline is as follows.

First, a sensor network is generated with a selected set of parameters. Second, the generated sensor network is converted into a flow network. Third, the flow network is visualized if needed for manual inspection. Fourth, the heterogeneous-cost maximum flow algorithm is run on the converted network. Fifth, the ILP benchmark is run on the same instance when the problem size is within the solver limit. Finally, the outputs are compared and summarized.

This structure was useful for two reasons. It simplified debugging because each stage could be checked on its own. It also made it easier to rerun experiments with different parameters without rewriting the full workflow each time.

Software Components

The implementation includes several main components.

The first component is the sensor network generator. This program creates random networks under user-defined parameters and exports the result to a text file.

The second component is the sensor-to-flow converter. This program reads the generated sensor network and produces the converted flow-network representation.

The third component is the visualizer. Separate visualizations were supported for the original sensor network and the converted flow network. This was especially useful for confirming whether a mismatch case was caused by a structural bottleneck or by an implementation issue.

The fourth component is the maximum flow algorithm for the heterogeneous-cost setting. This algorithm was implemented in Java.

The fifth component is the ILP benchmark, implemented in Python using Gurobi.

The final component is the comparison script, which automates repeated testing, runs both methods, and writes summary results to output files.

File-Based Design

All stages of the project communicate through text files. The sensor generator writes a sensor-network file. The converter reads that file and writes a flow-network file. The maximum flow algorithm and the ILP benchmark both read the converted flow file and each produces its own output file. The comparison script then reads those outputs and generates a summary table.

This file-based design keeps the programs loosely coupled. If one component is changed, the others do not need to be rewritten as long as the file format remains consistent. It also makes it easier to inspect sample inputs and outputs manually.

Heterogeneous-Cost MF Algorithm

The main algorithm implemented in this project follows the source-by-source idea described in the paper. The algorithm first sorts the data generators in nondecreasing order of source cost. It then processes them one at a time.

For one source, the algorithm attempts to find the maximum amount of feasible flow from the super source to the super sink in the converted network. Since the source has a specific cost, the effective residual capacity of a scalable edge depends on how much capacity remains and how much one unit of this source's flow will consume.

After the maximum feasible flow for that source is found, the corresponding edge usage is deducted from the network capacity. The algorithm then moves to the next source and repeats the process.

The total preserved flow is the sum of the flow values assigned to all processed sources.

Reason for Cost Ordering

The source-ordering rule is based on the idea that low-cost sources use edge capacity more efficiently. If one source consumes only 2 units of edge capacity per unit of preserved data while another consumes 8, then processing the lower-cost source first may allow more total data to be preserved.

This ordering rule is simple and intuitive, but it is also one of the algorithm's main limitations. A low-cost source may occupy a bottleneck edge or a storage path that a later source needs more urgently. In that situation, the greedy choice may reduce the final preserved flow even though each local step looked reasonable at the time.

Worth mentioning that the greedy idea also works in assumption that if an edge is used for a lower cost DG, then a higher cost DG later being process can not use that edge since its cost

is higher than the original node using that edge so undo and rerouting is not required. But later when analyzing the test results a situation emerges where if the lower cost DG uses a specific edge and pushes multiple flow to it then when the higher cost DG but lower than the total unit of flow the original DG has pushed through that edge then rerouting is required to archive optimal solution. The example will be presented in later chapters.

Residual Graph and Augmenting Paths

The implemented algorithm uses a residual-graph idea similar to ordinary maximum flow. For the current source, the algorithm searches for feasible augmenting paths from the super source to the super sink. A path is feasible only if every edge on that path still has enough residual capacity for one more unit of this source's flow times the cost of the source.

If an augmenting path is found, the algorithm increases the source's assigned flow and updates residual capacity accordingly. This process repeats until no additional feasible path remains for the current source.

Different variants were explored during the project in order to study how rerouting affects the result. In particular, a global-residual version was tested to see whether keeping a shared residual structure across source iterations could improve the match with the ILP benchmark. This version is not identical to the original source-by-source method, but it was useful for understanding how blocking and rerouting influence performance. On the other hand, this version introduced another aspect to the problem: is data splitable?

Limitations of the Greedy Method

The main limitation of the algorithm is that it is greedy with respect to source order and local residual feasibility. Once an early source uses a critical route, a later source may lose a feasible opportunity even if the network still contains a better overall arrangement.

This issue appeared clearly in several experimental cases:

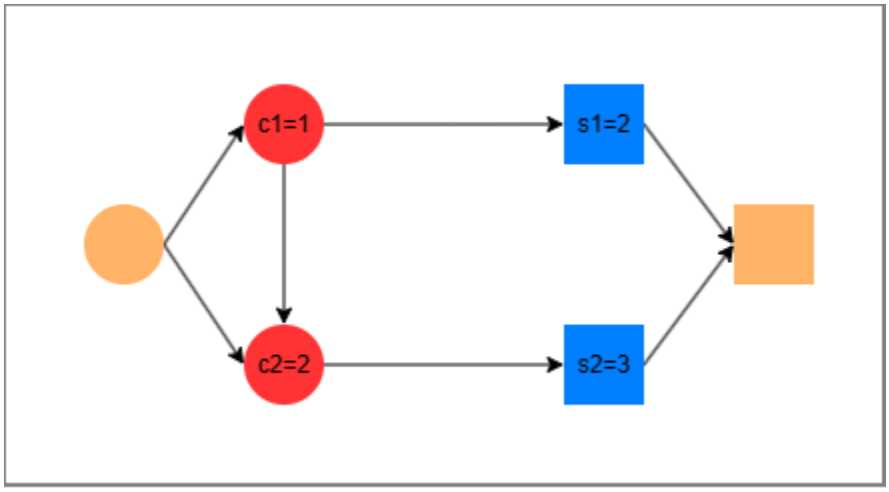

Figure 6. Example flow blockage 1

The above example is one cases that greedy will fail to archive the optimal solution. According to the paper-based implementation, it will choose DG 1 to process first since it has lowest cost and try to push maximum number of flow from DG 1 and the flow will go to storage 2 since it has the most storage capacity.

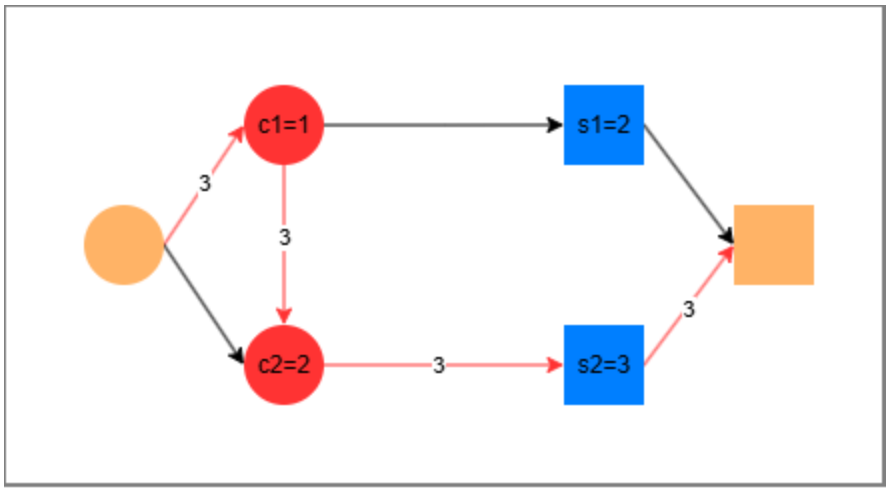

Figure 6.1

From this state the original implementation will deduct the edge capacity from DG 2 to Storage 2 and make it impossible for DG 2 to push any of the flow to any storage. But if the algorithm still counts the flow from edge DG 1 to DG 2 as a regular flow instead of deducing the capacity after first iteration. Then it enables DG 2 to use a reversed edge from DG 2 to DG 1 and rerouting 2 flows from DG 1 to Storage 1, creating room for DG 2 to push 1 package/2 flow to Storage 2. As a result, the total flow of the graph increases by 66%, achieving optimal outcome.

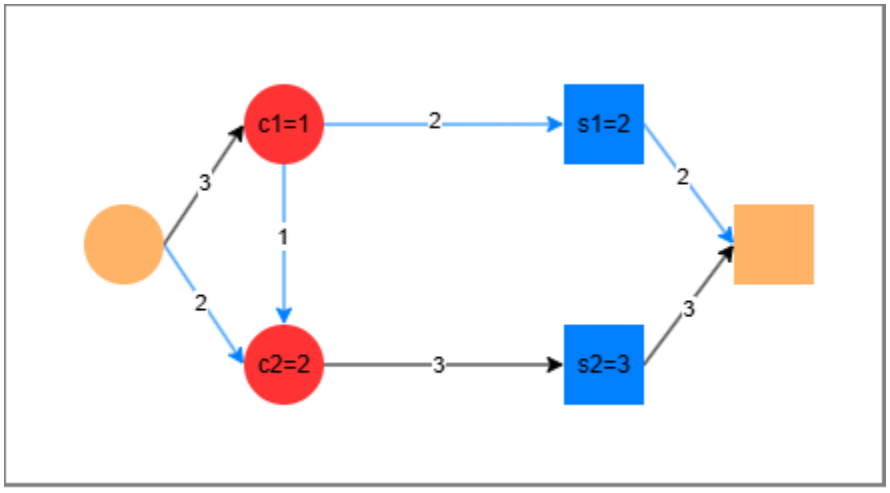

Figure 6.2

Without rerouting, the ILP benchmark will be able to find a higher total preserved flow by balancing source assignments globally, while the algorithm committed too much capacity to an earlier route.

Because of this, the original algorithm should be viewed as a practical heuristic rather than as a guaranteed exact solver for all instances.

ILP as an Exact Benchmark

To evaluate the algorithm properly, an exact benchmark was needed. This benchmark was implemented as an integer linear program. The ILP does not process sources one by one. Instead, it optimizes all source-specific flow variables together in one model.

The ILP uses a source-indexed edge-flow variable. Each variable represents the amount of flow on an edge that originated from a particular source. This makes it possible to model the heterogeneous-cost rule exactly, since the capacity consumed on an edge depends on both the amount of flow and the identity of the source.

The objective is to maximize total injected source flow. The constraint set enforces source limits, flow conservation, edge-capacity limits, and integrality.

Since the ILP solves the problem globally rather than greedily, its solution is treated as the optimal answer for instances small enough to fit the Gurobi solver restriction.

Programming Languages and Tools

Java was used for the generator, converter, visualizer, and algorithm implementation. Java was selected because it made it easy to manage multiple standalone programs and file-based workflows.

Python was used for the ILP benchmark and the comparison scripts. This choice made it easier to use the Gurobi optimization library and to automate repeated experiments.

The project also used spreadsheet tools such as Excel for computing summary statistics and confidence intervals for report figures.

Output and Debugging Support

Several forms of output were used to support debugging and evaluation.

The generator exports a text description of the sensor network. The converter exports the flow network. The maximum flow algorithm writes total flow, per-source flow, and edge-flow summaries. The ILP program writes corresponding optimal values. The comparison script combines the results into summary files that can be analyzed later.

This output design was useful because errors often became visible only when the intermediate network files were inspected directly. For example, examining the flow network made it possible to understand whether a mismatch was caused by a structural bottleneck, a greedy routing decision, or an implementation bug.

Summary

The implementation in this project was designed and developed as a modular experimental framework rather than as a single monolithic program. This made it easier to generate random instances, inspect converted networks, run multiple algorithms, and compare their outputs.

The main algorithm implemented here is a greedy heterogeneous-cost maximum flow method. It is simple and often effective, but it can lose optimality when early source assignments

block better later routes. For that reason, an ILP benchmark was also implemented to provide an exact reference point. The next chapter presents the ILP formulation in more detail.

CHAPTER 6

ILP BENCHMARK FORMULATION

This chapter presents the integer linear programming model used as the exact benchmark in this project. The purpose of the ILP is to compute the optimal preservation result for the converted flow network and to provide a reference against which the implemented algorithm can be evaluated.

Purpose of the ILP Model

The heterogeneous-cost maximum flow algorithm studied in this project is greedy. It processes sources in a fixed order and commits edge usage after each source. Although this method is efficient and often close to optimal, it does not guarantee the best global solution in every case.

To measure the gap between the algorithm and the optimum, a mathematical optimization model was needed. Integer linear programming was chosen because it can represent source-specific flow, flow conservation, source and sink limits, and the heterogeneous edge-capacity rule in a precise and exact way.

For each instance that fits within the solver's restricted license limit, the ILP computes the optimal maximum preserved flow. That result is then compared directly with the algorithmic output.

Flow-Network Basis of the ILP

The ILP is built on the converted flow network rather than on the original sensor network. This is important because the converted flow network already represents all of the physical constraints in directed edge form.

In the converted network, each original node is split into a prime node and a double-prime node. The internal edge between them represents node energy. Edges from the super source to data generator prime nodes represent source data availability. Edges from storage double-prime nodes to the super sink represent storage capacity. Routing edges represent communication links between sensors.

Because of this construction, the ILP can be written as a flow problem on a directed graph with capacities.

Decision Variables

A standard max-flow ILP would normally use one variable per directed edge. That is not sufficient in the present problem because different sources consume different amounts of edge capacity on the same edge.

To address this, the ILP uses a source-specific flow variable. Let $x_{k,u,v}$ denote the amount of flow on directed edge (u, v) that originated from source k.

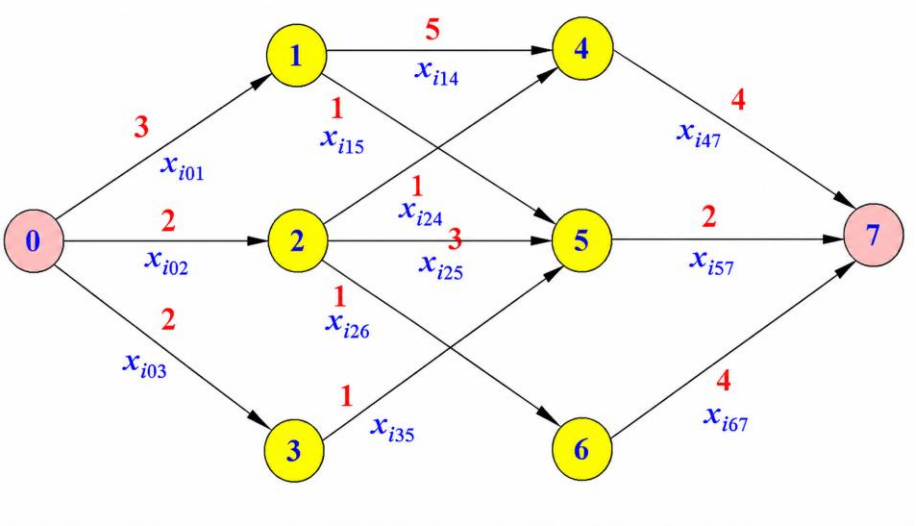

Figure 7. ILP variables graph for maximum flow problem

This variable plays two roles. First, it records how much preserved data from a given source travels along each edge. Second, it allows the model to account for source-dependent edge usage in the capacity constraints.

All decision variables are constrained to be nonnegative integers, since the project models preserved data as discrete flow units.

Objective Function

The objective of the ILP is to maximize the total amount of preserved data.

In the implemented model, this is expressed as the total amount of source flow injected into the network from the super source. Since flow is conserved throughout the converted network, maximizing injected source flow is equivalent to maximizing preserved flow.

$$max \sum_{i=0}^{k} x_{i,(s,s_i)}$$

This objective matches the main goal of the current stage of the project. Later, the same framework can be extended to a weighted objective in which preserved value or priority is maximized instead of raw flow amount.

Source Constraints

Each data generator has a finite amount of data. Therefore, the total amount of flow originating from source i cannot exceed the data available at that source.

$$x_{i,(s,s_i)} \leq d_i$$

In the converted network, this limit is enforced through the super-source edge connected to that source's prime node. The ILP includes a source-capacity constraint so that the injected flow for each source does not exceed its available data.

This guarantees that the model does not create artificial flow beyond what the original sensor network provides.

Flow Conservation Constraints

The ILP enforces flow conservation at every nonterminal node. For each source-specific commodity, the total inflow at an intermediate node must equal the total outflow.

$$\sum_{(u,v)\in \mathrm{E}} x_{i,(u,v)} = \sum_{(v,w)\in \mathrm{E}} x_{i,(v,w)} \qquad \forall i, \forall v \in V\{s,t\}$$

This means that data cannot appear or disappear in the network except at the super source and the super sink. The conservation constraints are written separately for each source because the model distinguishes sources through the source-indexed decision variable.

These constraints are essential for correctness. Without them, the model could assign disconnected or physically impossible flows.

Edge-Capacity Constraints

The most important part of the ILP is the heterogeneous edge-capacity rule.

In a standard max-flow model, the total flow on an edge cannot exceed its capacity. In the present model, one unit of flow from different sources may consume different amounts of capacity. Therefore, the capacity constraint must be written using source-specific cost.

$$\sum_{i=1}^{k} c_i \, x_{i,(u,v)} \leq ca(u,v) \qquad \forall (u,v) \in E_{intenal}$$

For an internal scalable edge, the sum over all sources of source-cost multiplied by source-specific flow must not exceed the edge capacity. In other words, the model does not count only how much flow uses the edge; it also counts how expensive that flow is with respect to its source.

This is the defining feature of the heterogeneous-cost model and is the main reason why the ILP requires source-indexed variables.

Source-Side and Sink-Side Super Edges

The converted network also contains super-source and super-sink side edges. These edges represent source supply and storage capacity rather than ordinary routing energy.

$$\sum_{i=1}^{k} x_{i,(s,s_r)} \leq ca(s,s_r) \qquad \forall s_r \in S$$

$$\sum_{i=1}^{k} x_{i,(t_j,t)} \leq ca(t_j,t) \qquad \forall t_j \in S$$

For this reason, the implemented ILP does not apply the heterogeneous-cost scaling to those super edges. Instead, they are treated as ordinary capacity limits in the model.

The source-side edges limit how much data a generator may inject, and the sink-side edges limit how much data a storage node may receive.

This treatment is consistent with the interpretation of the converted network used throughout the project.

Integrality Constraints

All flow variables are restricted to integer values. This reflects the project's modeling choice that preserved data is counted in discrete units rather than as continuous fractional flow.

$$x_{i,(u,v)} \in \mathrm{Z}_{\geq 0}$$

The integrality requirement also makes the ILP a true benchmark for the implemented algorithm, which also treats data as unit flow rather than as a continuous amount.

Gurobi library license restrictions

Although the ILP provides the exact solution, it is much more expensive computationally than the greedy algorithm. In addition, the free Gurobi license used in this project imposes a model-size limit.

Because the number of variables grows with both the number of sources and the number of edges, larger instances can easily exceed the size allowed by the restricted license. For that reason, the exact benchmark is only used on networks small enough to fit within the license limit.

This restriction does not reduce the usefulness of the ILP for the project. Even on smaller instances, it provides a valuable ground-truth comparison that makes it possible to evaluate the algorithm's accuracy and identify mismatch patterns.

Role of the ILP in the Evaluation

The ILP is not intended to replace the algorithmic method in practice. Instead, it serves as the standard against which the algorithm is judged.

For each tested instance that fits the solver limit, the ILP output provides the optimal preserved flow. The algorithmic result can then be compared with that value. If the two match, the algorithm is exact on that case. If they differ, the difference shows how much preserved flow was lost by the greedy method.

This comparison is central to the evaluation chapter, where match rate, relative loss, and failure cases are analyzed.

Summary

The ILP benchmark provides an exact formulation of the heterogeneous-cost preservation problem on the converted flow network. It uses source-specific flow variables, maximizes total preserved flow, and enforces source, sink, conservation, capacity, and integrality constraints.

Although it is limited to smaller instances because of solver size restrictions, it plays a crucial role in the project by making it possible to evaluate the implemented algorithm against the true optimum.

CHAPTER 7

EXPERIMENTAL SETUP

This chapter describes the parameters used for the testing process of the implementation in this projects.

Generated Test Instances

The experiments in this project were based on randomly generated sensor-network instances. A custom sensor network generator was implemented to produce these instances under user-defined parameters. This approach made it possible to test the algorithm and benchmark on many different network structures while keeping the parameter settings controlled.

Each generated instance is a two-dimensional field defined by a selected width and length. A specified number of nodes is placed randomly within that field. A subset of those nodes is designated as data generators, another subset is designated as storage nodes, and the remaining nodes are treated as transshipment nodes. Each data generator is assigned a data amount, a source cost, and a priority value. Each storage node is assigned a storage capacity. Every node is assigned an energy value.

Communication links are then created based on transmission range. If two nodes are within the range, they are connected in the generated sensor network. The resulting network is exported to a text file and later converted into the flow-network representation used by the algorithm and the ILP benchmark.

Parameter Settings

The generator supports a predefined range of input parameters. These include field dimensions, number of nodes, number of data generators, number of storage nodes, communication range, source data range, storage-capacity range, energy range, cost range, and priority range.

For the experiments reported in this project, the most important parameters were the number of nodes, the number of data generators, the number of storage nodes, the communication range, and the cost range. These parameters have the strongest effect on network density, available routing choices, and the size of the corresponding ILP model. Other parameter like priority is expect to use in future part of the project where the algorithm incorporates values into the results.

Because the free Gurobi license imposes a size limit on the number of decision variables and constraints (maximum of 2000 variables and 2000 constraints), the exact ILP benchmark could only be used on small and medium-sized instances. For that reason, one representative evaluation setting used in this report consists of 40 to 50 original nodes, 10 to 15 data generators, and 10 to 15 storage nodes. In this configuration, each data generator contains 50 to 150 units of data, each node energy is from 30 to 50 unit, and source cost is selected from the range 1 to 10. This setting is small enough to fit the solver restriction while still producing a meaningful variety of network structures.

Additional larger instances were also generated for broader testing, but the ILP comparison is only reported for those cases that fit the solver limit.

Experiment Workflow

Each experiment follows the same basic workflow.

First, a sensor-network instance is generated. Second, that instance is converted into a flow network. Third, the heterogeneous-cost maximum flow algorithm is run on the converted network. Fourth, the ILP benchmark is run on the same converted network whenever the instance is within the solver limit. Fifth, the outputs are compared and written to summary files.

This workflow ensures that both methods operate on exactly the same network instance. As a result, any difference in the final preserved flow comes from the solution method rather than from a difference in input.

The comparison scripts were designed to automate this process across many random seeds. This made it possible to generate previous test cases for the same parameter setting and later compute average performance and confidence intervals.

Evaluation Metrics

Several evaluation measures were used in this project.

The first metric is total preserved flow. This is the total amount of data successfully preserved by the algorithm or the ILP benchmark.

The second metric is per-source preserved flow. This shows how much data from each data generator was preserved. In mismatch cases, this metric helps identify whether the loss is spread across sources or concentrated on one or two particular generators.

The third metric is match rate. For one tested instance, the algorithm is considered a match if its total preserved flow is equal to the ILP optimum. Over multiple runs, the match rate is the percentage of instances for which the algorithm matches the ILP result exactly.

The fourth metric is relative flow loss. This is used only in instances where the algorithm does not match the ILP. It is computed as the difference between the ILP result and the algorithm

result, divided by the ILP result. This measures how much preserved flow was lost relative to the optimum.

Summary

The experiments in this project are based on randomly generated sensor-network instances that are converted into flow networks and evaluated by both the implemented algorithm and the ILP benchmark. The evaluation focuses on total preserved flow, per-source flow, match rate, and relative loss. Confidence intervals will be implemented for future insight of the experiments.

This experimental design provides a fair and reproducible basis for comparison. The next chapter presents the results and explains what they show about the strengths and limitations of the heterogeneous-cost maximum flow approach.

CHAPTER 8

RESULTS AND ANALYSIS

This chapter presents the main results of the project and analyzes the differences between the heterogeneous-cost maximum flow algorithm and its variation (splitable flow and reroutable path) with the ILP benchmark. The purpose of the analysis is to determine how closely the algorithm matches the optimal solution and to understand the situations in which it does not.

General Performance

The overall results show that the implemented algorithm often performs close to the ILP optimum on the tested instances. In a representative experiment using 500 randomly generated

networks with 40 nodes, 10 data generators, and 10 storage nodes, the algorithm matched the ILP result in 88 percent of the runs. In the remaining cases, the average relative flow loss was approximately 9.73 percent.

This is a strong result for a greedy method. It suggests that the heterogeneous-cost maximum flow algorithm is usually able to preserve nearly the same amount of data as the exact optimization model under this size of network. At the same time, the mismatch cases are important because they show that the algorithm is not always exact.

Interpretation of the Match Rate

A high match rate means that the algorithm often makes the same effective preservation decisions as the ILP. In those cases, the source order and local augmenting-path choices do not cause a global loss of preserved flow. The network either contains enough capacity or enough connections for the greedy decisions to remain safe, or the optimal structure happens to agree with the algorithm's routing order.

However, the match rate should not be interpreted as proof that the algorithm is exact in general since the problem is assume to be NP-hard. It only shows that under the tested parameter setting, most random instances do not trigger the algorithm's main weakness. The mismatch cases are therefore especially valuable because they reveal the situations where the greedy logic breaks down.

Relative Flow Loss

The average relative flow loss in the non-matching cases was about 9.73 percent in the representative experiment. This means that even when the algorithm fails to match the ILP, the amount of lost preserved data is often small compared with the optimal value.

From an academic point of view, the relative loss is acceptable and provides a promising direction to continue to explore the design of the algorithm. From a practical point of view, this loss may cause a problem since data is extremely valuable, even a slight loss in crucial numbers can drastically change the outcome of the prediction (weather forecast, earthquake prediction). To make an improvement the cause of the mismatch and loss need to be found and addressed.

Typical Cause of Mismatch

As mentioned in figure 1, one of the main causes of mismatch observed in the experiments is greedy blocking. Since the algorithm processes lower-cost sources first, it may assign flow to a route that looks efficient for the current source but consumes a bottleneck that is more valuable to a later source.

In such a case, the earlier source is not necessarily using a bad route in isolation. The problem is that the route choice is made without jointly optimizing the remaining sources. The ILP, by contrast, considers all source assignments at once and can avoid using a critical edge or storage path too early.

This pattern appeared in several manual inspection cases. In those cases, the ILP was able to preserve additional flow by rerouting an earlier source through a different feasible path, while the greedy algorithm had already committed the earlier source to a locally attractive but globally restrictive route.

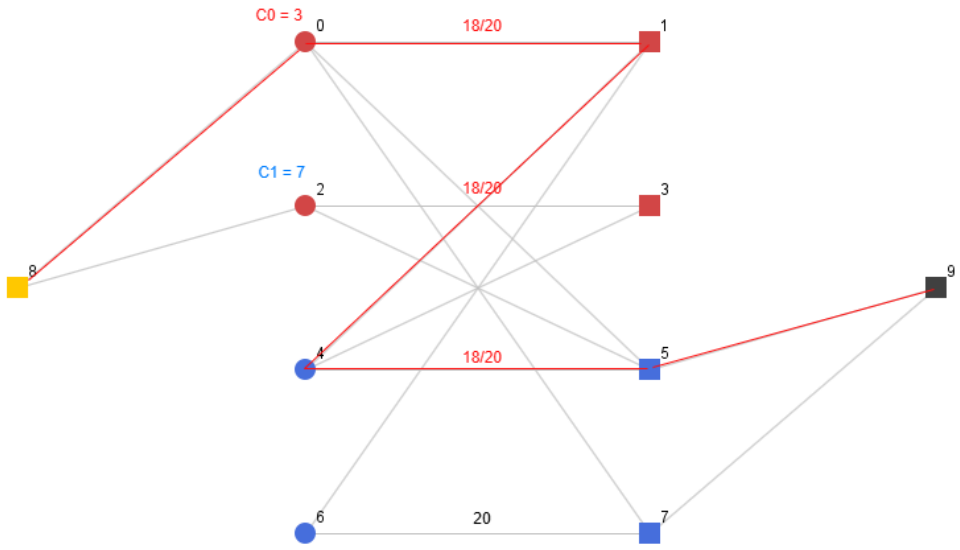

Figure 8. Bottleneck example

In the example above the algorithm will choose the path 8-0-1-4-5-9 during the first iteration to push flow from DG 0 to sink because DG 0 has the lowest cost of 3. So the total flow will be 3x6=18 unit of edge capacity. After that edge (4, 5) will be deducted according to the unit of flow and remain 2 after the first iteration. So when DG 2 try to push its data to sink it meet with the (4, 5) bottleneck as data from DG 2 cost 7 unit > 2 remaining unit and the algorithm stop here. The total flow is 3 from DG 1.

Structural Bottlenecks

Mismatch cases revealed that certain network structures are more likely to expose the weakness of the greedy algorithm. One common pattern is the presence of a small number of storage access paths that are reachable by multiple data generators. If an early source consumes

one of those paths, a later source may lose its only feasible route even though another route existed for the earlier source.

Another common pattern is a narrow relay structure in which several source-to-storage routes share one or two internal edges. In such cases, source order becomes especially important because early use of those shared edges can reduce flexibility for the remaining sources.

These patterns show that mismatch is not random. It is tied to how the available capacity is distributed in the network and how many alternatives exist for rerouting.

Per-Source Flow Behavior

The per-source flow output helps explain mismatch cases more clearly than total flow alone. In many non-matching instances, the loss is concentrated on one data generator rather than distributed evenly across all of them. A later source often receives less preserved flow because an earlier source used a route that reduced the later source's residual options.

This observation supports the conclusion that the problem is not merely a small arithmetic difference in total flow. It is often the result of one source being effectively blocked by earlier assignments. The ILP avoids this by balancing the source flows globally rather than sequentially.

Residual-Based Variant

During the project, an additional residual-based variation was developed to test whether rerouting could improve performance. This variation maintains a shared residual structure across iterations instead of fully fixing the edges' capacity before the next one begins.

The purpose of this experiment was to determine whether allowing previous flow decisions to be undone and rerouted would reduce the gap to the ILP benchmark. In some tested

cases, this idea did improve the result because it allowed the network to recover capacity that had been committed too early.

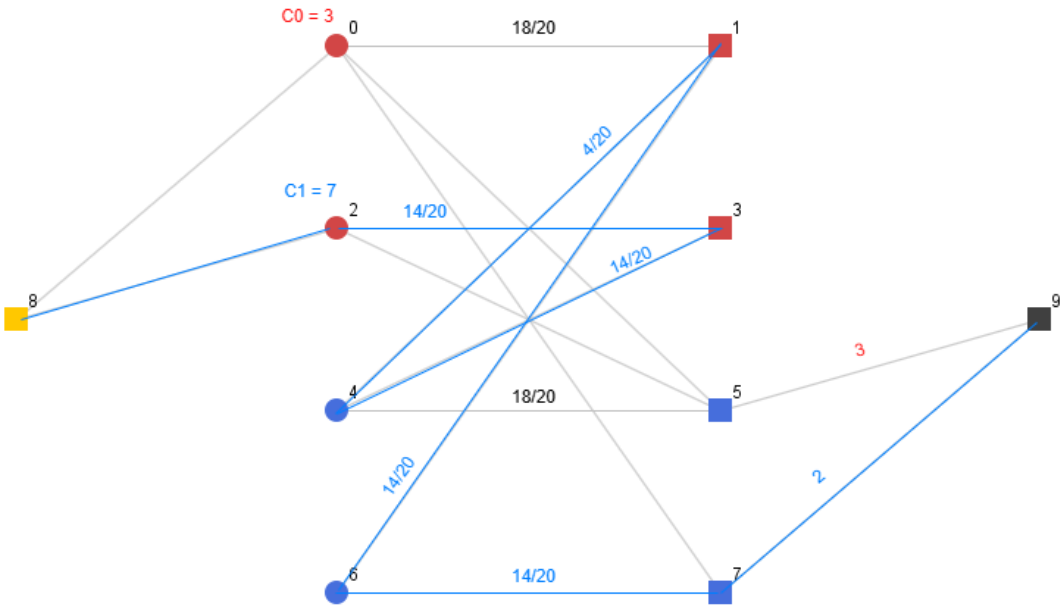

Figure 8.1

Continuing of the previous example, the variation algorithm allows the flow to use a previously occupied edge (4, 1) with is originally 18 flows from 1 to 4 as a reversed edge to push 14 units of flow from DG 2 to the sink. Resulting in extra 2 final flow in the solution. But because of rerouting happen, some data is split during this process. Let examine the flow from DG 1. First it push 3 flow (18 unit) from 0 to 1. Then 4 unit of flow (equivalent to 4/3 package of data) is pushed from 1 to 4 while the other 14 unit of flow (equivalent to 14/3 package of data) is pushed from 1 to 6. This variant ensures that the final solution is optimal but with the assumption that data package can be split, which might not be the case in real world situations. It is better viewed as a modified heuristic rather than as a direct implementation of the original method. For

that reason, it is useful as an exploratory comparison. In an extensive experiment using 1000 randomly generated networks with 50 nodes, 15 data generators, and 15 storage nodes, the algorithm matched the ILP result with an increasing rate of 93.4 percent of the runs. Moreover, the average loss is also significantly reduced to 1.85 percent in cases of mismatch. In addition, in rare cases the splitable variant of the algorithm even out perform the ILP optimal solution. One reason for this is that ILP algorithms implemented has an integral constraint whereas the splitable variant algorithm do not.

Practical Meaning of the Results

Taken together, the results suggest that the implemented algorithms is a strong practical method for the tested small and medium-sized cases. It often matches the optimum exactly, and when it does not, the average loss is small (even smaller on splitable cases). This makes it a useful approach when exact optimization is either too slow or unavailable.

At the same time, the mismatch analysis shows that the greedy method is not exact in general. It can fail when the network contains shared bottlenecks, asymmetric routing alternatives, or source-ordering conflicts. Therefore, the algorithm should be understood as an efficient heuristic with good empirical performance rather than as a universally optimal solver.

On the other hand, when data is assumed to be splitable, the variant algorithm perform extremely good and even better than the optimal integral ILP benchmark. This can open up to another direction for this project by addressing the splitable factor of data.

CHAPTER 9

CONCLUSION AND FUTURE WORK

This project studied data preservation in base station-less sensor networks under heterogeneous transmission cost. A complete software framework was developed to support this study, including a random sensor-network generator, a sensor-to-flow-network converter, visualization tools, a heterogeneous-cost maximum flow algorithm, a splitable flow variant of MF algorithm, an ILP benchmark, and comparison scripts.

The project showed that the flow-network conversion provides a useful way to represent source data, storage capacity, and node energy within a single directed graph model. This made it possible to apply both a practical flow algorithm and an exact optimization model to the same preservation problem.

The experimental results showed that the implemented algorithm often performs close to the ILP optimum on the tested instances. In the representative case discussed in this report, the match rate was high and the average relative flow loss in non-matching instances was small. These results indicate that the algorithm is effective in many practical cases, even though it is greedy.

At the same time, the project also identified the main limitation of the method. Because the algorithm processes sources sequentially in cost order, early low-cost assignments may consume bottlenecks that are more important for later sources. This can reduce the final preserved flow even when a better global arrangement exists. The ILP benchmark was especially valuable in revealing these cases and in showing that the mismatch comes from the structure of the greedy method rather than from the basic flow model alone. As a result the splitable variant

of the algorithm is explored to result in an even higher match rate and lower loss rate at the assumption of data can be split.

Another useful outcome of the project was the development of tools for analyzing mismatch cases. The visualizer, intermediate file outputs, and per-source flow summaries made it possible to inspect individual networks and understand why the algorithm and ILP differed. This was important not only for debugging, but also for explaining the algorithm's strengths and limitations in a concrete way.

Future work

There are several directions for future work. The first is to extend the evaluation to larger networks. At present, the exact ILP comparison is limited by the size restriction of the available solver license. A larger license or a different exact optimization tool would make it possible to study more realistic network sizes.

The second direction is to design improved routing methods that reduce the blocking effect of the greedy source order. This may include better tie-breaking rules, bottleneck-aware source selection, or controlled rerouting strategies.

The third direction is to study preservation under nonsplittable data assumptions. Some of the exploratory rerouting ideas in this project implicitly treat flow in a way that is more flexible than strictly indivisible data packets. A more realistic model may require preserving this indivisibility while still improving over the original greedy approach.

The fourth direction is to incorporate data priority directly into the optimization model. The current project focuses on maximizing preserved flow amount. A natural extension is to maximize preserved value, so that the network preserves the most important data when resources

are limited. The current implementation framework provides a useful starting point for that extension.

In conclusion, this project demonstrates that heterogeneous-cost maximum flow is a useful model for data preservation in sensor networks and that a practical greedy algorithm can achieve near-optimal performance on many instances. It also shows that exact benchmarking is necessary to understand the limits of that algorithm and to guide future improvements.

REFERENCES

Rivera, G. and Tang, B. Priority-Based Data Preservation in Challenging Environments: a Maximum Weighted Flow Approach.

Jason, R. and Tang, B. (2026). Maximum Weighted Flow With Storage Constraint

Yu, Y., Hsu, S., Chen, A., Chen, Y., & Tang, B. (2023). Truthful and optimal data preservation in base station-less sensor networks: An integrated game theory and network flow approach. *ACM Transactions on Sensor Networks*, *20*(1), 1-40.

Chen, Y. and Tang, B. "Data Preservation in Base Station-less Sensor Networks: A Game Theoretic Approach", Proceedings of the 6th EAI International Conference on Game Theory for Networks, Kelowna, BC, Canada, 2016.

Tang, B., Jaggi, N., Wu, H., & Kurkal, R. (2013). Energy-efficient data redistribution in sensor networks. ACM Transactions on Sensor Networks, 9(3), Article 33.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). Network flows: Theory, algorithms, and applications. Prentice Hall.

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM, 19(2), 248–264.

Ford, L. R., Jr., & Fulkerson, D. R. (1962). Flows in networks. Princeton University Press.

Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum-flow problem. Journal of the ACM, 35(4), 921–940.

Heinzelman, W. R., Chandrakasan, A., & Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. In Proceedings of the 33rd Hawaii International Conference on System Sciences.

Hou, X., Shi, J., Wang, H., & Sherali, H. D. (2012). Maximizing data preservation in intermittently connected sensor networks. In Proceedings of the IEEE International Conference on Mobile Ad Hoc and Sensor Systems.

Gurobi Optimization, LLC. (2025). Gurobi Optimizer Reference Manual. https://www.gurobi.com

APPENDIX A: GITHUB REPOSITORY

The complete source code for this project is available in the following GitHub repository: https://github.com/nguyentrunghieu2909/max-flow-heterogenous-edge-cost

The repository includes the sensor network generator, the flow-network converter, the visualization tools, the implemented flow algorithms, the modified splitable maximum flow algorithm, the ILP benchmark, the experiment scripts, and the README file with execution instructions.

APPENDIX B: RESULT FILES

Following is the snapshot of the data sheet for the result of the experiments. The full result is in the following Google Sheet:

https://docs.google.com/spreadsheets/d/1_iBGT0qCgag972HvCN0K9xc_vpBAYIzUfyJL3rbi3jY/edit?usp=sharing

Table 1. Result test on unsplittable data with arguments: 1000x1000, range 125, 40 nodes, 10 DGs, 10 storage, data [50-100], storage [50-150], energy [30-50], cost [1-10], priority [1-100], seed 1.

Seed	Algorithm 2 Total flow	ILP Total Flow	Total Flow Match	Per-Source Match
1	73	73	True	True
2	22	22	True	False
3	24	24	True	True
4	46	46	True	True
5	58	58	True	True
6	10	10	True	True
7	47	50	False	False
8	29	29	True	True
9	31	31	True	True
10	16	16	True	True
11	68	68	True	True
12	73	73	True	True
13	70	70	True	True
14	15	15	True	True
15	73	76	False	False
16	56	56	True	False
17	22	22	True	True
18	16	16	True	True
19	86	86	True	True
20	56	56	True	True

Table 2. Result test on splittable data with arguments: 1000x1000, range 125, 50 nodes, 15 DGs, 15 storage, data [50-100], storage [50-150], energy [30-50], cost [1-10], priority [1-100], seed 1.

Seed	Algorithm 2 Total Flow	ILP Total Flow	Total Flow Match	Per-Source Match
1	67	67	True	True
2	67	67	True	True
3	169	169	True	True
4	34	34	True	True
5	18	18	True	False
6	72	72	True	True
7	31	31	True	True
8	34	35	False	False
9	47	46	False	False
10	103	103	True	False
11	51	51	True	False
12	16	16	True	True
13	21	21	True	False
14	64	64	True	True
15	26	26	True	True
16	68	68	True	True
17	72	72	True	True
18	99	99	True	True
19	126	127	False	False
20	30	31	False	False

APPENDIX C: LIVE DEMO RECORDING

A several minutes screen recording of the project is available at the following link: https://drive.google.com/file/d/1itancNGmpfg-ynFOsTxaQ4E3VI8dRz4Q/view?usp=sharing

The recording demonstrates the overall workflow of the project, including network generation, conversion, algorithm execution, and result comparison.