SCALABLE SOLUTIONS TO THE CARBON CAPTURE INFRASTRUCTURE PROBLEM

by

Caleb Whitman

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2020

ACKNOWLEDGEMENTS

I would like acknowledge my advisor Sean Yaw, for his continued help and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Algorithm Page

ABSTRACT

CO_2 capture and storage (CCS) is a climate change mitigation strategy that aims to reduce the amount of CO_2 vented into the atmosphere from industrial processes. Designing cost-effective CCS infrastructure is critical to meeting CO_2 emission reduction targets and is a computationally challenging problem. CCS infrastructure design is a generalization of the capacitated fixed charge network flow problem, CFCNF. CFCNF is NP-hard with no known approximation algorithms. In our work, we design three novel heuristics to solve CCS. We evaluate all heuristics on real life CCS infrastructure design data and find that they quickly generate solutions close to optimal. Decreasing the time it takes to determine CCS infrastructure designs will support national-level scenarios, undertaking risk and sensitivity assessments, and understanding the impact of government policies (e.g. 45Q tax credits for CCS).

1

INTRODUCTION

CO_2 capture and storage (CCS) is the process of capturing CO_2 emissions from industrial sources, notably coal-fired and natural gas power plants, and injecting it into geological reservoirs (sinks) for the purpose of combating climate change and economic benefit (e.g. enhanced oil recovery). CCS is a key technology in all climate change mitigation plans that limit global temperatures below 2 ° C of warming. To have a meaningful impact, this will involve optimizing infrastructure deployments for hundreds of sources and sinks, and thousands of kilometres of pipeline networks [38].

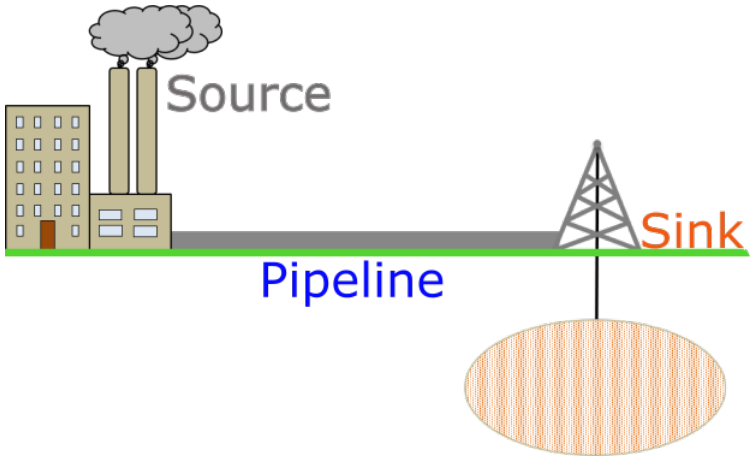

Figure 1.1: Goal: Reduce vented CO_2.

Deploying CCS infrastructure on a massive scale requires careful and comprehensive planning to ensure investments are made in a cost-effective manner [38]. *SimCCS* is an economic-engineering optimization tool for designing CCS infrastructure [29, 30, 47]. At its core, *SimCCS* formulates an optimization problem whose solution yields the most cost-effective locations and quantities of CO_2 to capture, route via pipeline, and inject for storage. *SimCCS* uses a candidate pipeline network [48] and data

about the possible source and sink locations to formulate a Mixed Integer Linear Program (MILP) that aims to minimize total cost, while capturing and injecting a target amount of CO_2. Solving this MILP yields an optimal CCS infrastructure design for the scenario under consideration. Since solving MILP problems is NP-Complete, this optimization approach does not scale as scenarios grow in size.

The Carbon Capture Infrastructure Design Problem, CIDP, aims to find a set of sources, sinks, and pipelines which capture T tons of CO_2 at minimum cost. CIDP generalizes the capacitated fixed charge network flow problem, CFNCF, which is NP-hard with no known approximation algorithms. We design three new, fast heuristics for solving CIDP. We test all approaches on real life data and show that our heuristics dramatically decrease running time with a minimal drop in solution quality. We then implement both heuristics into $SimCSS$, allowing industries to quickly and accurately estimate infrastructure design. Reducing the time it takes to determine CIDP infrastructure designs will support national-level scenarios, undertaking risk and sensitivity assessments, and understanding the impact of government policies (e.g. 45Q tax credits for CCS).

The rest of this paper is organized as follows. We formalize CIDP in Section 2 and show its relations to CFNCF. Section 3 discusses related work. Our three algorithms are presented in Section 4. Experimental results are presented in Section 4 and we conclude in Section 5.

3

PROBLEM DESCRIPTION

Given a set of CO_2 emitters (sources), geological reservoirs (sinks), internal vertices (pipeline junctions that do not occur at a source or sink), and a candidate pipeline network, the goal of the CCS Infrastructure Design Problem (CIDP) is to determine, in a cost-minimal fashion, which sources to capture from, which sinks to inject into, and which (and what diameter) pipelines to build to capture a pre-determined target quantity of CO_2. All costs and flow amounts are calculated on an annual basis (i.e. the annual cost to capture, transport, and store x tons of CO_2).

The sources and sinks are parameterized with an economic model consisting of fixed costs to open locations, and variable costs to utilize the locations (dollars per tonne of CO_2 captured/injected per year). Sinks have additional costs associated with building and using injection wells. These costs vary based on the ease of injection at that particular site. Pipelines have construction and per-tonne utilization costs dependent on their geographic location and the quantity of CO_2 they transport. Pipelines can intuitively be represented as a number of discrete pipeline sizes (i.e. diameters) and their associated cost and capacities. In a mixed integer linear program (MILP) formulation, this representation requires an integer variable for each possible pipeline edge/size pair, which results in a large number of variables and quickly leads to intractable formulations. Instead, pipelines can be represented as a smaller set of linearized functions of pipeline capacity versus cost (i.e., trends). This allows for simpler formulations compared to the discrete formulation while still ensuring that the cost model is realistic [28]. The CIDP based on linearized pipelines is formulated as an MILP in the following section.

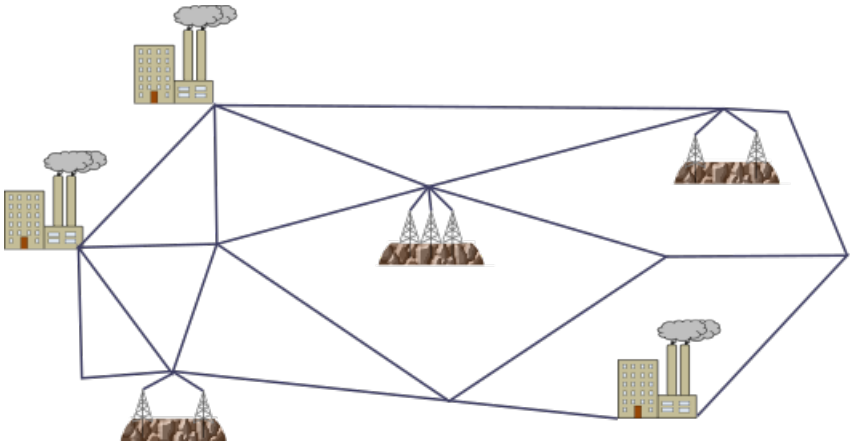

Figure 2.1: Choose sources, sinks, and edges to reduce overall cost.

MILP

The MILP contains a set of input parameters, a set of decision variables, an objective function, and a set of constraints. The goal is to minimize the objective function subject to the constraints. Simply put, we aim to send flow along from a set of sources to a set of sinks along a set of pipelines which minimize the total cost.

Input Parameters:

$CO_2Cap \in \mathbb{R}$	Target CO_2 capture amount for project (tCO_2/yr)
S	Set of sources
F_i^S	Fixed cost of opening source i
V_i^S	Variable cost for capturing CO_2 from source i
Q_i^S	CO_2 production rate at source i $(MtCO_2/yr)$
R	Set of reservoirs
F_j^r	Fixed cost of opening reservoir j
F_j^W	Fixed cost of opening well w of reservoir j
V_j^W	Variable cost of injecting CO_2 in well at reservoir j
Q_j^W	Capacity of well at reservoir j $(MtCO_2/yr)$

Q_j^r	Capacity of reservoir j ($MtCO_2/yr$)
K	Set of candidate pipeline arcs
C	Set of pipeline capacity trends
Q_{kc}^{max}	Maximum capacity of pipeline k with trend c
Q_{kc}^{min}	Minimum capacity of pipeline k with trend c
α_{kc}	Cost to transport one tonne of CO_2 on pipeline k with trend c
β_{kc}	Cost to build pipeline k with trend c

Decision Variables:

$s_i \in \{0, 1\}$	Indicates if source at node i is opened
$a_i \in \mathbb{R}$	Amount of CO_2 captured at source i (tCO_2/yr)
$r_j \in \{0, 1\}$	Indicates if reservoir at node j is opened
$b_i \in \mathbb{R}$	Amount of CO_2 injected into reservoir j (tCO_2/yr)
$w_j \in \mathbb{N}$	Number of wells opened at reservoir j
$y_{kc} \in \{0, 1\}$	Indicates if pipeline k with trend c is opened
$p_{kc} \in \mathbb{R}$	Amount of flow in pipeline k with trend c (tCO_2/yr)

The MILP is driven by the objective function:

$$\min \left(\begin{array}{c} \overbrace{\sum_{i \in S} (F_i^{src} s_i + V_i^{src} a_i)}^{\text{capture cost}} \\ + \\ \overbrace{\sum_{k \in K} \sum_{c \in C} (\alpha_{kc} p_{kc} + \beta_{kc} y_{kc})}^{\text{transport cost}} \\ + \\ \overbrace{\sum_{j \in R} (F_j^{res} r_j + F_j^{well} w_j + V_j^{well} b_j)}^{\text{storage cost}} \end{array} \right)$$

Subject to the following constraints:

$$Q_{kc}^{min} y_{kc} \leq p_{kc} \leq Q_{kc}^{max} y_{kc}, \forall k \in K, \forall c \in C \tag{2.1}$$

$$\sum_{c \in C} y_{kc} \leq 1, \forall k \in K \tag{2.2}$$

$$\sum_{k \in K: src(k)=i} \sum_{c \in C} p_{kc} - \sum_{k \in K: dest(k)=i} \sum_{c \in C} p_{kc} = \begin{cases} a_i & \text{if i} \in \text{S} \\ -b_i & \text{if i} \in \text{R, } \forall i \in I \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$a_i \leq Q_i^S s_i, \forall i \in S \tag{2.4}$$

$$b_j \leq Q_j^w w_j, \forall j \in R \tag{2.5}$$

$$b_j \leq Q_j^r r_j, \forall j \in R \tag{2.6}$$

$$\sum_{i \in S} a_i \geq CO_2 Cap \tag{2.7}$$

Equation 2.1 ensures a pipeline is built before transporting CO_2 and flow is under required capacity. 2.2 allows at most one linear-price (i.e. trend) pipeline between any two nodes. 2.3 enforces conservation of flow at each node. 2.4 and 2.6 add capacities for sources and sinks respectively. 2.5 ensures a well must be constructed to inject CO_2 and caps injection. 2.7 ensures the total capture amount meets the target.

Fixed Charge and Concave Cost Network Flow

CIDP generalizes the capacitated fixed charge network flow problem, CFCNF. CFCNF is itself a variation of the uncapcitated version, FCNF, and the minimum concave network flow problem, MCNF. This is important as both FCNF and MCNF exist in a variety of applications and have been well studied over the years. Existing complexity results and algorithms relating to FCNF and MCNF can be applied to our problem. We explain related work in chapter 3 and here focus on defining each problem and describing their complexity.

Given a set of supply vertices (sources), demand vertices (sinks), internal vertices, and edges (pipelines), both FCNF and MCNF seek to find the minimum cost required to deliver flow from the sources to the sinks. The predominant feature of each is the associated edge cost. For MCNF, edge costs are given by any concave function, $f(x)$. FCNF is a version of MCNF where each edge cost is given by $c_f + xc_v$ where c_f is denoted as the fixed cost, c_v the variable cost, and x the amount of flow. The goal of each is to determine a minimum cost flow that satisfies all of the demand.

A wide number of variations to each problem have been examined, including capacity requirements, multi-commodity flows, and tree graphs [11]. Below, we include the MILP for CFCNF.

MILP for CFCNF

E	Edges
S	Sources
T	Sinks
d^S	Source production for source $s \in S$
d^T	Sink capacity for sink $t \in T$
$x_{i,j}$	Flow on edge (i,j)
$y_{i,j} \in \{0,1\}$	Indicates if edge is open
$f_{i,j}$	Fixed cost of edge (i,j)
$v_{i,j}$	Variable cost of edge (i,j)

The MILP is driven by the objective function:

$$min\left\langle \sum_{(i,j)\in E} v_{i,j}x_{i,j} + f_{i,j}y_{i,j} \right\rangle \tag{2.8}$$

Subject to the following constraints:

$$\sum_{j\in N} x_{(i,j)} - \sum_{j\in N} x_{(j,i)} = \begin{cases} d^S & \text{if j} \in \text{S} \\ -d^T & \text{if j} \in \text{T} \\ 0 & \text{otherwise} \end{cases} , \forall i \in N \tag{2.9}$$

$$x_{(i,j)} \leq c_{(i,j)}y_{(i,j)}, \forall (i,j) \in E \tag{2.10}$$

$$x_{(i,j)} \geq 0, \forall (i,j) \in E \tag{2.11}$$

$$y_{(i,j)} \in \{0,1\}, \forall (i,j) \in E \tag{2.12}$$

Equation 2.9 ensures conservation of flow. The capacity constraints in 2.10 prevent the flow from exceeding the edge capacity and prevent flow on closed edges. Constraint 2.11 protects against negative flow and constraint 2.12 ensures that each edge is only opened or closed.

Complexity

It is clear that CIDP generalizes CFCNF by (1) adding source and sink costs, (2) adding wells to regulate sink demand, (3) allowing parallel edges representing different pipeline sizes, (4) allowing the total source capacity to be different from the total sink capacity, and (5) allowing a subset of the capacity to be captured (i.e. the target CO_2 capture amount). This proves that CIDP is at least as hard as CFCNF. It has long been known that CFCNF is NP-Hard [11].

Chuzhoy et al. prove that there is no $c \log \log n$-approximation algorithm for the uncapacitated version of the CFCNF problem (FCNF), for some constant c, unless $NP \subseteq DTIME(n^{O(\log \log \log n)})$. They find their inapproximability results by reducing the priority Steiner tree problem to FCNF. They prove priority Steiner tree's hardness by encoding it as an instance of a set cover problem, whose hardness results follow from a reduction to MAX 3SAT(5) [5].

In the priority Steiner Tree problem, we are given a graph $G = (V, E)$. Each edge $e \in E$ has real edge costs c_e and edge priorities $p_e \in \{1, 2, 3, ..., k\}$. There are three types of nodes: a root node r, a set of terminal nodes T, and intermediate nodes

I. Each node T desires a certain priority $p_t \in \{1, 2, 3, ..., k\}$. The goal is to create minimum cost Steiner tree such that every edge e on the path from $t \in T$ to r has a priority less than the terminal priority, $p_e \leq p_t$[5].

To prove the hardness of the priority Steiner tree problem, Chuzhoy et al. reduce 3SAT(5) to priority Steiner tree using a set cover construction previously created by Lund and Yannakakis [26]. They then reduce this priority Steiner tree instance to an instance of FCNF. For each vertex with priority i, set the demand to be $n^{(k-i)}$ and for each edge of priority j, set the capacity to be $n^{5(k-j)+2}$ where k is the total number of priority levels. Given a solution to the priority Steiner tree instance, we can great an instance to FCNF with no greater cost and visa versa. This carries over the hardness result of the priority Steiner tree to FCNF [5].

Since FCNF generalizes CFCNF which in turn generalizes to CIDP, this inapproximability result also applies to CIDP. Interesting enough, outside of these approximation results, there are no known approximation algorithms for CFCNF, hinting at the difficult of the problem [11, 41].

Palekar et al. experimentally examine the computational influences of arc density and ratio between fixed cost and variable cost. They find that graphs where $\frac{c_f}{c_v} \approx 1$ are the most difficult to solve. As $\frac{c_f}{c_v}$ increases, the solution begins to behave more like the minimum spanning tree problem. As $\frac{c_f}{c_v}$ decreases, it behaves more like a minimum cost flow problem. Thus, when both fixed costs and variable costs contribute significantly to the overall cost, the problem is most difficult to solve. Finally, denser and larger networks are more difficult to solve as more edges create a larger space which must be searched [34].

While we have focused primarily on CFCNF, Lamar prove that any network flow problem with non-linear costs can be converted into an equivalent MCNF in an expanded network. This includes CIDP and indicates in terms of complexity, our

problem lies somewhere in between CFCNF and the general MCNF [24].

Thus, CIDP is NP-hard with no existing approximation algorithms and no possible approximation algorithm better than $c \log \log n$. These results reinforce the need for fast and reliable heuristic approaches.

12

RELATED WORK

MCNF and CFCNF are found in many operations research application areas including transshipment problems [45], pipeline design [30], bank branch locations [31], gas and water lines [39], offshore platform drilling [15], traffic networks [35], supply chains, network communications, and a variety of other situations [11]. The network infrastructure can vary significantly between all problems, ranging from trees to multi-edged, multi-commodity designs. Due to the large number of applications, numerous exact and heuristic solutions are developed to solve these and other related problems [11].We give a brief summary of all methods in table 3.1.

While slow for large problem sets, exact solutions receive a lot of attention [11]. Most rely on various bounding and cutting approaches [12, 12, 13, 23]. Gallo et al. create one of the first branch and bound procedures [13]. Fontes et al. design a branch and bound algorithm with a sub-exponential performance time complexity by deriving fast lower bounds based on state space relaxations [12]. Adlahka et al. first relax the solution by creating linear edge costs and then use a branch and bound procedure on the relaxed problem. This improves running time but at a significant decrease in solution quality [1]. Lagrangian relaxations are commonly used for bounding. Lagrangian relaxations first relax or remove one inequality, and then introduce a *penalty* variable. The penalty variable acts in place of the inequality, and moves the solutions away from invalid results. Depending on the inequality removed, these can either provide upper or lower bounds in a fraction of the time required to solve the original solution [6, 7, 11, 13, 19, 20, 22, 23]. Kliewer and Timajev use this Lagrangian and branch and bound procedure [22]. Khang and Fujiwara iteratively find new Lagrangian relaxations for the problem until they land on the optimal solution [20]. SimCCS first formulates the problem as an MILP and then uses the well known MILP

solver CPLEX [30].

Dozens of Heuristic approaches have been tried and tested [11, 14, 32, 46]. Here we give a brief overview of the most common approaches along with few unique techniques. We will focus in depth on a local search heuristic designed by Guisewite and Pardalos [16] and a slope scaling heuristic designed by Crainic, Gendron, and Hernu [6] as these directly influence our algorithms presented in section 4.

Several heuristics first relax the problem and then either solve the relaxed version or use the relaxed version to guide solution selection [4, 6, 18, 19, 20, 22, 33, 36]. Nahapetyan and Pardalos approximate concave costs with a piece-wise linear function and then use a dynamic cost updating procedure to solve the new problem [33]. Rebennack et al. build upon this strategy and develop an exact solution based on similar ideas [36]. Burkard et al. first replace all costs with a relaxed linear representation and then use dynamic programming to solve the relaxed solution [4]. Hewitt et al. combine various bounding, linear program, and local search techniques to design a fast but reliable algorithm [18].

Slope Scaling Heuristics iteratively solve relaxed versions of the initial solution, updating costs as they go to find better approximations [6, 9, 14]. Crainic, Gendron, and Hernu develop a slope scaling heuristic which relaxes the solution by replacing the fixed and variable costs with a single variable cost. To avoid becoming stuck in local optima, they introduce a long term memory procedure, which perturbs the solution based on previous flow values and significantly improves results [6]. Gendron, Hanafib, and Todosijevićb improve this algorithm by introducing an additional iterative linear program approach[6]. We follow a similar approach in the design of the algorithm presented in section 4.

Genetic algorithms, GAs, find solutions via natural selection. A set of solutions iteratively evolve, with only the best solutions in each iteration contributing to the

new solutions in the next. These algorithms and their variations perform remarkably well on the MCNF, especially when compared to other approaches. [10, 32, 39, 43, 45, 46]. Fontes and Gonçalves design a genetic solution followed by local search to improve results [10]. Xie and Jia use a hybrid minimum cost flow and GA to solve a variant of the MCNFP [43]. Walters and Smith use an evolutionary approach based on GA to solve a tree variant of the problem [39]. Shangyao Yan et al. develop a GA and find that it performs better than several local search algorithms [45].

Simulated annealing randomly chooses new solutions based on a constantly updated probability. The probability of a new solution is based on the relative change in solution cost. At the start, solutions which decrease the solution cost are given relatively high probabilities in order to explore the solution space. After each iteration, the chance of choosing a worse solution decreases [2].Altiparmak and Karaoglan use a hybrid simulated anneal and tabu search strategy [2]. Yaghini et al. use a combination of a Simplex and simulated annealing heuristic to solve a multi-commodity flow network [44]. Dang et al. use a deterministic annealing method, a variant of simulated annealing, and focus on solving problems with high arc densities, multiple sources/sinks, and a polynomial cost function. Their contribution is unique in that many other heuristics are designed to work on networks with low arc densities and/or a single source [7].

Particle swarm heuristics create a set of particles that travel the search space. Each particles velocity and direction is constantly updated to find the best solution. Yan, Shih, and Lee create one of the first particle swarm algorithms for MCNFP combined with a genetic and thresholding heuristic [46]. A couple of heuristics described below also use particle swarm techniques [27].

A couple artificial immune algorithms, heuristics based on the mammalian immune system, are applied to the MCNF. Simply put, multiple "B-cells" repeatedly

clone themselves and attempt to find solutions. B-cells that have lower costs produce more of themselves, leading to better solutions over the long term [27, 40]. Molla-Alizadeh-Zavardehi et al. use a hybrid artificial immune and GA algorithm [40]. El-Sherbiny and Alhamali design a hybrid artificial immune and particle swarm algorithm [27].

Ant colony heuristics send "ants" along the network. Each ant leaves a pheromone trail indicating where it traveled. Ants that find good solutions leave stronger pheromones, attracting more ants to explore near the solution. Monteiro and Fontes design the first an ant colony heuristic for MCNFP [32]. Poorzahedy and Rouhani design several hybrid heuristics based on ant colony, genetic algorithm, simulated annealing, and tabu search methods [35].

Several papers invoke more unique and innovative solutions. Sadeghi-Moghaddam et al. use simulated annealing and a whale optimization algorithm to solve the MCNF with fuzzy costs. Whale optimization algorithms search the solution space in two phases. In the first phase, the solution is explored semi-randomly looking for decent solutions. In the second phase, the best regions found in the first phase are explored further. They also employ some GA and particle swarm methods [37]. Yousefi et al. use a combination of keshtel, simulated annealing, and genetic algorithms. The keshtel algorithm is based off of ducks of the same name. The ducks each go off randomly to find food. Once a large food source is found, one duck will circle the food. The circle attracts other ducks to the area. Similarly, the algorithm sends out "Keshtels" which attempt to find low cost solutions. The keshtels who find cheap solutions are joined by others and nearby solutions are explored [49]. Glover uses a ghost image processing search. Ghost image processing relaxes the problem by parameterizing the objective function and gradually updates the parameters using a sort of meta-heuristic and tabu search [15]. Silva and Loch modify the MODI method

for transportation problems to work for fixed charged [25]. Finally, Nicholson and Zang create a predictive model to determine if an arc would have a non-zero flow in the resulting solution. The model is 85% accurate [8].

Local search heuristics, by far one of the most popular approaches, greedily find solutions based on local criteria. Early heuristic attempts often completely rely on local searches, which various adding, dropping, and swapping strategies [16, 31]. Local searches are fast but often get stuck within local optimum. Tabu searches attempt to solve this problem by incorporating memory into the search procedure. As the algorithm progresses, tabu search gives less precedence to recently visited edges, prompting the algorithm to travel to unvisited parts of the graph and reduce likelihood of getting trapped within a local optimum [2, 3, 21, 35]. Many of the more recent approaches incorporate a local search algorithm to improve their existing solution [2, 9, 18, 35].

Relating to our greedy heuristic, described in section 4, Guisewite and Pardalos investigate several local search heuristics based on finding a set of shortest paths between one source and multiple sinks [16]. One of their heuristics finds all shortest paths between the source and sinks and then chooses the smallest number of cheapest paths that satisfy flow requirements as the solution. Path cost is estimated using edge costs of $c_f + xc_v$, where c_f and c_v are costs and x is the maximum flow allowed for that path [16]. As explained further in 4, our greedy heuristic is a modified version of this approach. The main difference is that our heuristic is iterative and recalculates the shortest path problem after each new path is added. Guisewite and Pardalos add all shortest paths at once, without recalculation [16].

Finally, we note that the MCNF lacks an approximation algorithm for all but the most trivial of cases [11, 32]. A few simplified instances of the problem, for example, ones where the number of concave edges has been reduced to one, are solvable in

polynomial time. Generally speaking, the lower the number of edges with concave costs, the easier the problem is to solve. However, the general algorithms lacks any approximation [41]. Given the amount of work in the field, we can theorize finding any approximation is difficult.

All related work is summarized below in Table 3.1.

Table 3.1, summary of related work:

Author	**Year**	**Application(s)**	**Approach(es)**
Gallo et al. [13]	1980	Minimum Concave Cost Network Flow	Branch and Bound
Guisewite and Pardalos [16]	1991	Uncapacited Minimum Concave Cost Network Flow	Local Search
Khang and Fujiwara [20]	1991	Capacitated Fixed Charge Network Flow	Branch and Bound with Lagrangian Relaxation
Hochbaum and Segev [19]	1989	Fixed Charge Network Flow	Lagrangian Bounds;Local Search
Ekşioğlu et al. [9]	1970	Fixed Charge Network Flow	Dynamic Slope Scaling; Local Search
Bazlamacci and Hindi [3]	1996	Concave-cost Uncapacitated Transshipment Problem	Extreme-point Search; Tabu Search
Smith and Walters [39]	2000	Water and gas lines	GA

Burkard et al. [4]	2001	Single Source Minimum Concave Cost Network Flow	Dynamic Programming; Linear Approximations
Crainic et al. [6]	2005	Multicommodity Capacitated Fixed-Charge Network Design	Slope Scaling Relaxation; Lagrangean Bounds
Glover [15]	2005	Fixed Charge Network Flow	Ghost Image
Kliewer and Timajev [22]	2005	Various Capacitated Network Design Problems	Lagrangean based branch and bound
Monteiro and Fontes [31]	2005	Bank Branch Locations	Local Search
Yan et al. [45]	2005	Concave Cost Transshipment Problems	GA; Tabu Search; Local Search
Fontes et al. [12]	2006	Minimum Concave Cost Network Flow	Dynammic Programming; Branch and Bound
Kim et al. [21]	2006	Fixed Charge Network	Dynamic Slope Scaling; Tabu Search
Fontes and Gonçalves [10]	2007	Minimum Concave Cost Network Flow	GA; Local Search
Poorzahedy and M.Rouhani [35]	2007	Road Transportation Network	Ant Colony; GA; Simulated Annealing; Tabu Search

Altiparmak and Karaoglan [2]	2008	Concave Cost Transportation Problems	Tabu Search; Simulated Annealing
Altiparmak and Karaoglan [2]	2008	Capacitated Fixed Charge Multicommodity Network Flow	Simplex Method; Simulated Annealing
Nahapetyan and Pardalos [33]	2008	Fixed Charge Network Flow	Dynamic Cost Updating Procedure; Relaxation Techniques
Rebennack et al. [36]	2009	Fixed Charge Network Flow	Bilinear Modeling
Adlakha et al. [1]	2010	Fixed Charge Transportation Problem	Branch and Bound
Hewitt et al. [18]	2010	Fixed Charge Network Flow	Cutting procedure; Relaxations; Local Search
Dang et al. [7]	2011	Minimum Concave Cost Network Flow	Deterministic Annealing; Lagrangian Relaxation
S.Molla-Alizadeh-Zavardehi et al. [40]	2011	Capacitated Fixed Charge Transportation Problem	Artificial Immune Algorithm; GA
Yan et al. [46]	2011	Minimum Concave Cost Network Flow	GA; Particle Swarm
Xie and Jia [43]	2012	Nonlinear Fixed Charge Transportation Problem	GA

M.El-Sherbiny and M.Alhamali [27]	2013	Fixed Charge Transportation Problem	Artificial Immune Algorithm; Particle Swarm
Monteiro et al. [32]	2013	Single Source Uncapacited Minimum Concave Cost Network Flow	GA; Ant Colony
Loch and da Silva [25]	2014	Fixed Charge Transportation Problem	MODI Algorithm
D.Nicholson and Zhang [8]	2016	Fixed Charge Network Flow	Predictive Analysis
Gendron et al. [14]	2018	Multicommodity Capacitated Fixed-Charge Network Flow	Slope Scaling Relaxation; Iterative Linear Programming
Hashmi et al. [17]	2019	Fuzzy Two Stage Fixed Charge Network Flow	Fuzzy Goal Programming
Sadeghi-Moghaddam et al. [37]	2019	Fuzzy Fixed Charge Network Flow	Whale Optimization Algorithm; Particle Swarm; GA
Yousefi et al. [49]	2019	Fixed Charge Transportation Problem	Simulated Annealing; Keshtel; GA

Table 3.1: Summary of Related Work

21

HEURISTICS

We introduce two new heuristics and one hybrid heuristic to solve CIDP. As all existing solutions for CIDP rely on exact approaches, these heuristics provide a much needed speedup. Each heuristic is empirically tested in chapter 4.

Greedy Heuristic

As mentioned in chapter 3, Guisewite and Pardalos create a heuristic which chooses the set of least cost shortest paths that capture the target flow [16]. This approach often fails as each shortest path is calculated independently from the others. In Figures 4.1 to 4.3 we give an example of a problematic network.

Here, we see that the heuristic chooses a set of disjoint paths as the final solution. While each individual path is relatively cheap, the sum of all paths together leads to an expensive solution. The optimal solution has all source to sink paths run through the central, middle edge. Due to its high fixed costs, this path is ignored in the heuristic.

Our algorithm adds in an iterative procedure to better capture the single purchase of fixed costs. Instead of choosing all shortest paths at once, we instead choose the least cost shortest path, add it to the solution, and then recalculate all possible shortest paths. During recalculation, all edges with existing flow have a fixed cost of zero. This allows for greater shortest path sharing, leading to far better solutions than before, as seen in section 4. We demonstrate the improvement in Figure 4.4.

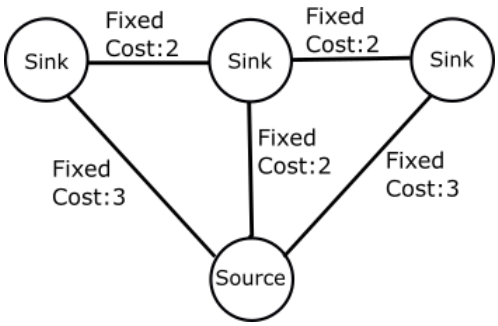

Figure 4.1: Problematic Network

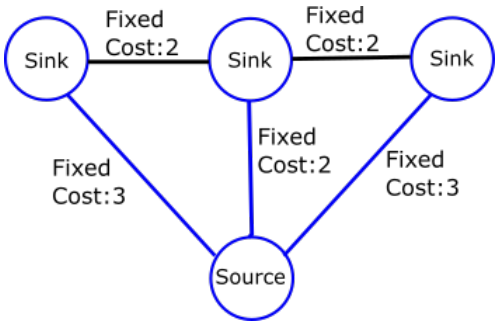

Figure 4.2: Guisewite and Pardalos solution. Total cost of 8.

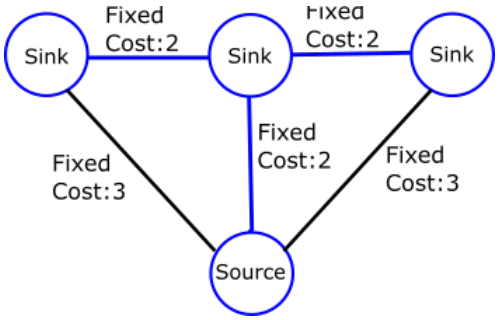

Figure 4.3: Optimal Solution. Total cost of 6.

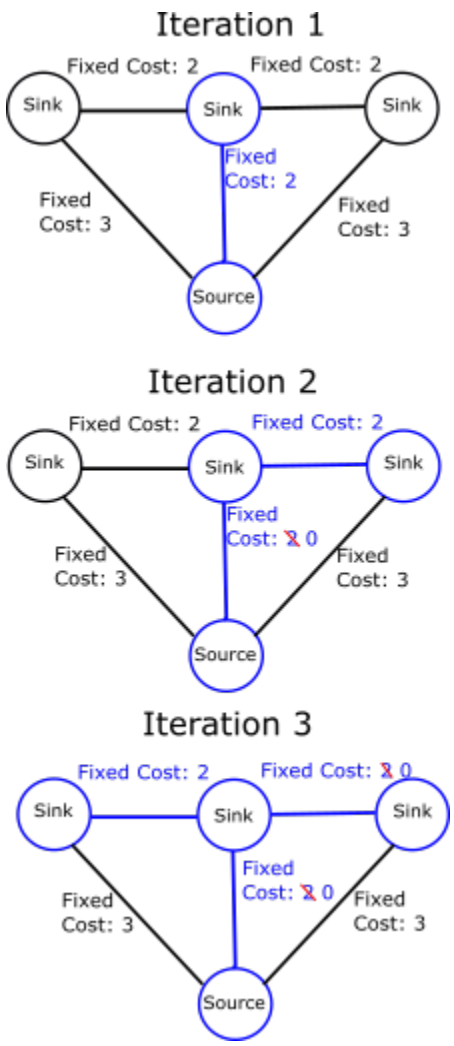

Figure 4.4: Heuristic Solution. Achieves optimal cost of 6.

Algorithm

Algorithm 4.1 considers the least cost path for each $(source, sink)$ pair and greedily selects the path that is able to capture CO_2 for the lowest cost per ton. Pipeline costs are factored into this calculation by determining the additional pipeline capacity and infrastructure needed to support the new $(source, sink)$ path. When a path is added to the solution, all associated costs and flows are appropriately adjusted for future paths. This process then repeats until the target quantity of CO_2 is captured. A preliminary version of this algorithm appeared in a poster at ACM's e-Energy conference in 2019 [42]. This algorithm is presented in Algorithm 4.1.

Complexity

The heuristic runs in $O(in^2)$ where n is the number of nodes and i is the number of iterations. The number of iterations is heavily dependent on the size of the problem and the capacity of each each. In chapter 4, we find that for real life data, i can be large and significantly contributes to overall running time.

Pathological Case

Our heuristic performs significantly worse than optimal if fixed costs are dispersed pathologically. One construction leading to an arbitrarily bad solution is shown in Figure 4.5. Assume that half of all sources must be included in the solution to reach target capacity and that all costs, except those displayed, are zero.

The heuristic chooses the least cost path at every iteration. In Figure 4.5, this is always one of the paths in the top sub-graph, with a fixed cost of 1. This will lead to a total cost of n, where n is the number of source nodes in either the top or bottom subgraph. However, the optimal solution, the bottom sub-graph, only has a total cost of 2. Thus, by adding an arbitrary number of source nodes, we can increase the cost

Algorithm 4.1: Greedy CCS Infrastructure Design

Input: Sources S, Sinks K, Pipelines P, Capture Target T

Output: $S' \subseteq S, K' \subseteq K, P' \subseteq P$

```
1: S', K', P' = ∅; cap = 0
2: while cap < T do
3:         cheapestPair = ∅; minCost = ∞
4:         for (s, k) ∈ S × K do
5:                 c = min(s.capacityLeft, k.capacityLeft)
6:                 cost = s.captureCost(c) + k.injectionCost(c)
7:                 path = cheapestPath(s, k, c) {considers P'}
8:                 cost = cost + path.cost
9:                 cost = cost / c
10:                if cost < minCost then
11:                        minCost = cost
12:                        minPair = (s, k, c, path)
13:                end if
14:        end for
15:        S'.add(s), K'.add(k), P'.add(path)
16:        cap += c
17: end while
```

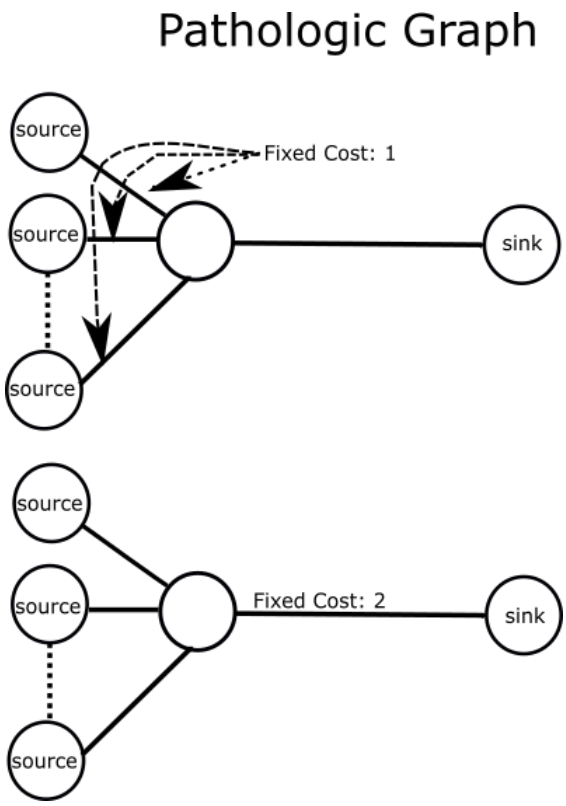

Figure 4.5: Pathological graph with two subgraphs. The heuristic chooses the top subgraph while the optimal solution is the bottom subgraph.

of the heuristic by n and end up with a cost that is arbitrarily worse than optimal.

Flow Infrastructure Design Heuristic

We motivate our approach by again highlighting the differences between CIDP and CFCNF. There are four main differences: multiple edge costs, wells, the number of sources/sinks, and the required captured flow. However, out of all of these, the only important difference is the multiple edge costs. In polynomial time, CIDP can be converted into a multiple edge version of CFNCF. The conversion process, represented in Figure 4.6, is as follows:

1. Create new source S_n and new sink T_n. Set capacity of S_n and T_n to target capture amount. Set S_n and T_n costs to 0.

2. For all sources s, create a new pipeline p from S_n to s with the same capacity, fixed costs, and variable costs as s.

3. For all sinks t, create a new pipeline p from t to T_n. p will have a set of pipeline capacities and costs (trends) relating to the well costs and capacities for t.

4. Remove all costs and capacities from all s and t, transforming them into intermediate nodes.

This change highlights the fact that any algorithm designed for CFCNF can applied to CIDP as long as we account for multiple edge costs (pipeline trends).

We design a slope scaling heuristic based off of the idea of approximating all edge costs with a single, variable cost. Removing all fixed costs from the problem transforms it from an NP hard problem to a simple minimum cost flow problem in P.

Each pipeline trend cost is approximated by $x_i(c_f/x_{i-1} + c_v)$ where c_f is the original fixed cost, c_v is the original variable cost, and x_i is the amount of flow in

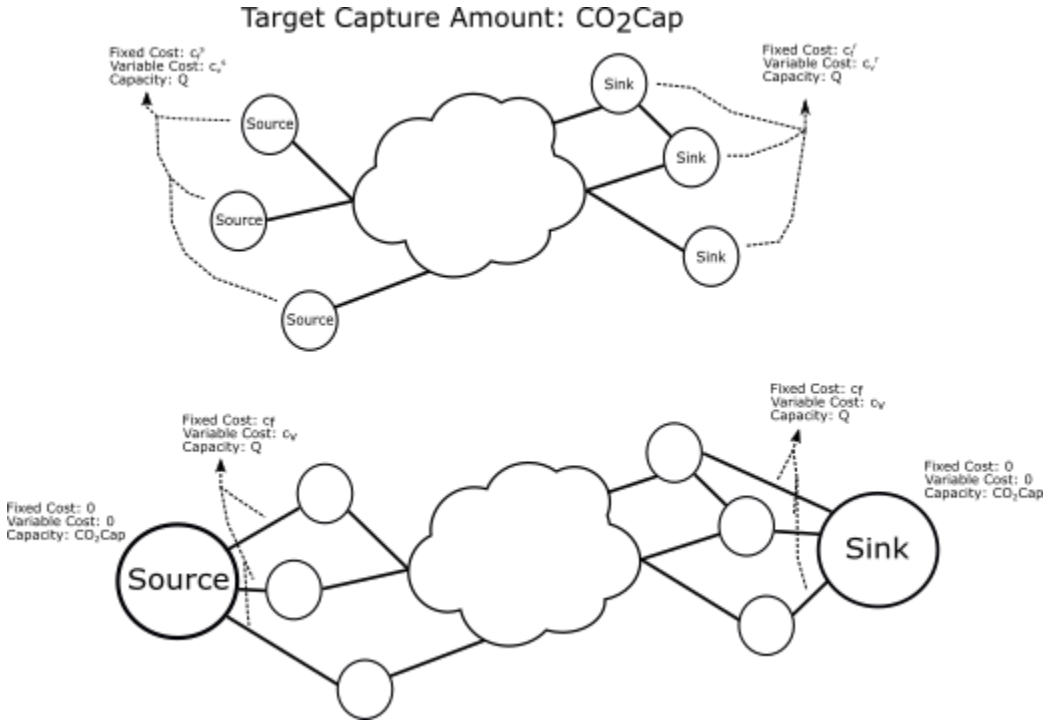

Figure 4.6: CIDP To CFCNF with multiple edge costs and capacities.

iteration i. When x_{i-1} is equal to the actual amount of flow, the total cost becomes the true edge cost, $c_f + x_i c_v$. While this strategy does gradually improve the solution, experimentation finds that it becomes easily trapped in local optima

To avoid getting suck, a memory procedure stores how heavily each pipeline was used in prior iterations. The algorithm cycles through diversification and intensification phases. Diversification prioritizes rarely used pipelines, allowing the algorithm to explore new edges. Intensification prioritizes heavily used pipelines, improving already discovered solutions. Pipeline usage is calculated using the number of times the pipeline has flow above zero and the average, maximum, and standard deviation of flow. This idea was first discovered and implemented by Crainic, Grendron, and Hernu [6]. We expand their idea by implemented a similar memory procedure for each pipeline trend, instead of the entire pipeline as a whole.

The overall procedure is outlined in algorithm 4.2.

Hybrid Infrastructure Design Heuristic

The Greedy-Flow Hybrid merges algorithms 4.1 and 4.2. First, algorithm 4.2 runs to completion. Next, we iteratively remove and add flow. Flow is removed by randomly choosing a source and then using a random breadth first search. Flow is added back in using algorithm 4.1. This gradually improves the solution by replacing potentially expensive paths with cheaper ones. Pseudocode is presented in algorithm 4.3.

Algorithm 4.2: Flow Infrastructure Design

Input: Sources S, Sinks K, Pipelines P, Capture Target T

Output: $S' \subseteq S, K' \subseteq K, P' \subseteq P$

$S', K', P' = \emptyset$; $cap0$;P_t =pipeline trends

$numIter_{pt}, avgFlow_{pt}, stdFlow_{pt}, maxFlow_{pt} = 0, \ \ \forall pt \in P_t$

ω^+, ω^- = positive constant

for $i < maxIterations$ **do**

    $v_{pt} = \frac{avgFlow_{pt}}{maxFlow_{pt}} \ \ \forall pt \in P_t$

    $\delta^+_{pt} = avgFlow_{pt} + \omega^+ stdFlow_{pt}$

    $\delta^-_{pt} = avgFlow_{pt} + \omega^- stdFlow_{pt}$

    if i =Intensify Iteration **then**

        if $numIter_{pt} \geq \delta^+_{pt}$ **then**

            $cost_{pt,i} = cost_{pt,i-1}(1 - v_{pt})$

        else if $numIter_{pt} \leq \delta^-_{pt}$ **then**

            $cost_{pt,i} = cost_{pt,i-1}(2 - v_{pt})$

        end if

        Solve modified LP

    end if

    if i =Diversify Iteration **then**

        if $numIter_{pt} \geq \delta^+_{pt}$ **then**

            $cost_{pt,i} = cost_{pt,i-1}(1 + v_{pt})$

        else if $numIter_{pt} \leq \delta^-_{pt}$ **then**

            $cost_{pt,i} = cost_{pt,i-1}(v_{pt})$

        end if

        Solve modified LP

    end if

end for

Algorithm 4.3: Greedy-Flow Hybrid

Input: Sources S, Sinks K, Pipelines P, Capture Target T

Output: $S' \subseteq S, K' \subseteq K, P' \subseteq P$

1: $S^0, K^0, P^0 = Iterative\ Flow(S, K, P, T)$

2: **for** $i < max\_iterations$ **do**

3: $s \in S^{i-1}$ = random source

4: Remove flow from s via breath first search

5: $S^i, K^i, P^i = Greedy(S^{i-1}, K^{i-1}, P^{i-1}, T)$

6: **end for**

7: **return** S^i, K^i, P^i

Evaluation

We implement all heuristics into SimCCS and test on real life data. Source and sink data were provided by the Great Plains Institute in support of the 2019 National Petroleum Council carbon capture, use, and storage study. Due to the proprietary nature of the data sources and non-disclosure agreements, the raw source and sink data cannot be distributed with this thesis. The data included sources and sinks across most of the contiguous U.S.A, with a total of 150 potential sources and 270 potential sinks. Pipelines and costs were generated via the SimCCS geologic and economic model.

First, we run five premilarily tests to determine the influence of source/sink ratio on computational difficulty. We randomly choose x sources and y sinks, varying the x/y ratio from 1/79 to 79/1. We find that tests with a large difference between the number of sinks and sources ($x/y \neq 1$) are unable to capture a significant amount of CO_2 due to the source/sink capacities. For example, if we have 79 sources and 1 sink, then our capture amount is limited by the sink capacity. This reduces the overall complexity of the problem. Thus, problem sets with a near equal number of sources and sinks are the most difficult to solve. For this reason, we use test sets with an equal number of sources and sinks for our main results.

Five test cases are generated by randomly choosing sources and sinks. We create 10 scenarios, each with x random sources and x random sinks. Results are averaged across these 10 scenarios to obtain the final cost and time for that test case. x is gradually increased from 10 to 120 to highlight the performance of each algorithm across different infrastructure sizes. Time is limited to three days. Any attempt exceeding this limit is excluded from the results.

All algorithms are benchmarked on a machine with an Intel Core i7-2450

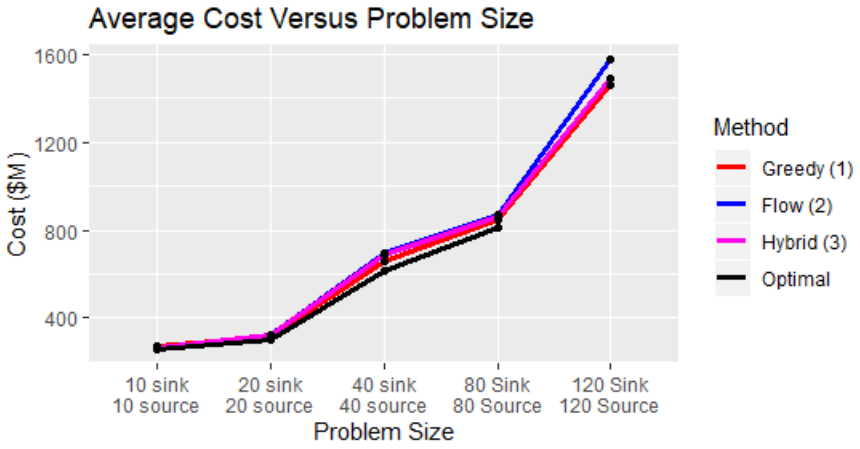

Figure 4.7: Cost Versus Size.

processor running at 2.1 GHz using 32 GB of RAM, running Fedora 30. The optimal MILPs are solved on this machine using IBM's optimization tool CPLEX, version 12.10.

Results

Figure 4.7 presents the cost comparison between all algorithms and the optimal solution found via CPLEX. In terms of absolute value, all algorithms perform close to optimal, with no difference exceeding 80 million dollars. In terms of percent difference, we find that Algorithm 4.1 consistently stays within 7% of optimal. Algorithm 4.3 always improves the initial solution provided by Algorithm 4.2. However, Algorithm 4.2 varies considerably, ranging from a 4 to 13 percent difference. We highlight percent differences in Figure 4.8.

Figure 4.9 presents the running time comparison between all algorithms and the optimal CPLEX approach for all sizes. All algorithms run considerably faster than CPLEX, with CPLEX unable to find an optimal solution to the 120 sink, 120 source data set within three days. For problem sizes of 40 and under, all algorithms complete within a minute. Due to its $O(in^2)$ time complexity, Algorithm 4.1 takes considerably

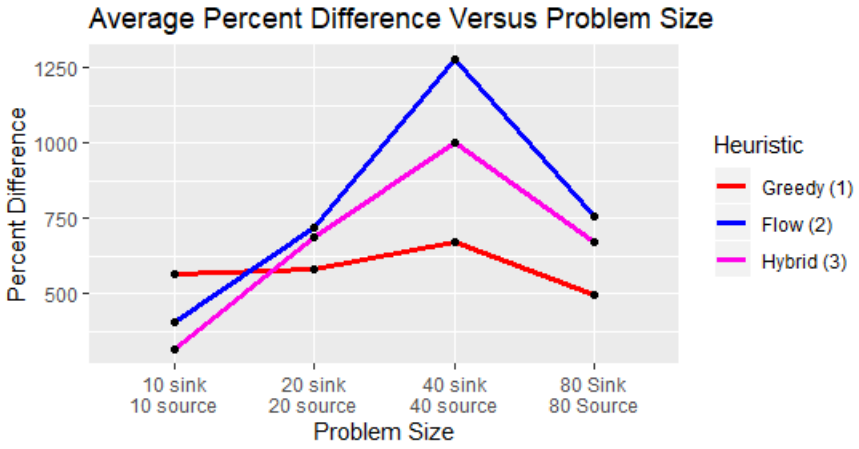

Figure 4.8: Percent Difference to Optimal Versus Size.

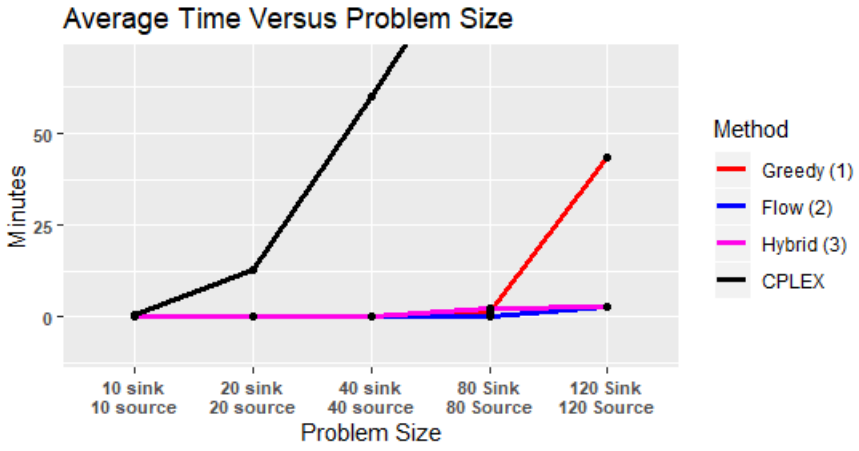

Figure 4.9: Time Versus Size.

longer to complete for larger datasets. We find that the number of iterations, i, tends to be a strong driving factor in average completion time, with all 120 Sink, 120 Source scenarios taking over 100 iterations.

While Algorithm 4.3 also runs in $O(in^2)$ time, the number of iterations, i, remains significantly low enough that we see no dramatic increase in time from Algorithm 4.2. Algorithm 4.2's running time increases slightly as we increase problem size, but remains faster than a minute for all tested data.

We see a trade off between precise costs, accurate costs, and time for all algorithms. This presents a different use case for each algorithm. Algorithm 4.1 remains the most precise, consistently staying within 7% of optimal. This suggests that the algorithm will work well for small to mid-ranged scenarios, consistently providing reliably good solutions. Algorithm 4.2 is the fastest and also achieves high accuracy for some datasets. Algorithm 4.3 consistently improves Algorithm 4.2 with a minimal increase in time. For most larger scenarios, Algorithm 4.3 will work best as it quickly finds good solutions.

CONCLUSION

We create three heuristics for CIDP and experimentally show that they are fast with minimal loss in solution quality for real life data. All algorithms represent viable approaches for approximating CCS infrastructure design for large scenarios. This allows organizations to better explore scenarios that were previously hindered by the exponential running time of solving CIDP optimally.

Future practical work should focus on more testing and stabilizing/lowering heuristic costs. Testing should take place on real life scenarios from other organizations. Scenarios produced by other groups may see differences in network structure, requiring modifications of each algorithm. In addition, further improvements should look at reducing the cost for all algorithms, improving usage feasibility. In particular, algorithms 4.1 and 4.3 may benefit from a long term memory procedure to better explore the solution space. Algorithm 4.2 should see more work to lower the high variability in solution quality.

Future theoretical work should look at how well each heuristic performs on CFCNF and what sorts of quality differences arise. Benchmarking against other CFCNF heuristics would allow us to better quantify the theoretical performance of our heuristics. Finally, creating any sort of approximation algorithm for CFCNF would represent a significant contribution to the field and propel our understanding of the problem.

Bibliography

[1] Veena Adlakha, Krzysztof Kowalski, and Benjamin Lev. A branching method for the fixed charge transportation problem. *Omega*, 8:393 – 397, 2010. doi: doi.org/10.1016/j.omega.2009.10.005.

[2] F Altiparmak and I Karaoglan. An adaptive tabu-simulated annealing for concave cost transportation problems. *Journal of the Operational Research Society*, 59:331 – 341, 2008. doi: 10.1057/palgrave.jors.2602301.

[3] Cüneyt F. Bazlamacci and Khalil S. Hindi. Enhanced adjacent extreme-point search and tabu search for the minimum concave-cost uncapacitated transshipment problem. *Journal of the Operational Research Society*, 47:1150 – 1165, 1996.

[4] Rainer E. Burkard, Helidon Dollani, and Phan Thien Thach. Linear approximations in a dynamic programming approach for the uncapacitated single-source minimum concave cost network flow problem in acyclic networks. *Journal of Global Optimization*, 19:121 – 139, Sep 2001. doi: 10.1023/A:1008379621400.

[5] Julia Chuzhoy, Anupam Gupta, Joseph (Seffi) Naor, and Amitabh Bhuvangyan Sinha. On the approximability of some network design problems. *ACM Transactions on Algorithms*, 23(3), May 2008. doi: doi.org/10.1145/1361192.1361200.

[6] Teodor Gabriel Crainic, Bernard Gendron, and Geneviève Hernu. A slope scaling/lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design. *Journal of Heuristics*, 10: 525–545, Sep 2005. URL `https://link.springer.com/article/10.1023/B:HEUR.0000045323.83583.bd`.

[7] Chuangyin Dang, Yabin Sun, Yuping Wang, and Yang Yang. A deterministic annealing algorithm for the minimum concave cost network flow problem. *Neural networks : the official journal of the International Neural Network Society*, 24: 699–708, Apr 2011. doi: 10.1016/j.neunet.2011.03.018.

[8] Charles D.Nicholson and Weili Zhang. Optimal network flow: A predictive analytics perspective on the fixed-charge network flow problem. *Computers Industrial Engineering*, 99:260 – 268, 2016. doi: doi.org/10.1016/j.cie.2016.07.030.

[9] Burak Ekşioğlu, Sandra Duni Ekşioğlu, and Panos M. Pardalos. Solving large scale fixed charge network flow problems. *Equilibrium Problems and Variational Models*, pages 163–183, Jan 1970. URL `https://link.springer.com/chapter/10.1007/978-1-4613-0239-1_9#citeas`.

[10] Dalila B.M.M. Fontes and José Fernando Gonçalves. Heuristic solutions for general concave minimum cost network flow problems. *Networks*, 50:67–76, Apr 2007. URL https://onlinelibrary.wiley.com/doi/epdf/10.1002/net.20167.

[11] Dalila B.M.M. Fontes and José Fernando Gonçalves. Solving concave network flow problems. *FEP Working Papers*, 475, Dec 2012. URL http://wps.fep.up.pt/wps/wp475.pdf.

[12] Dalila B.M.M. Fontes, Eleni Hadjiconstantinou, and Nicos Christofides. A branch-and-bound algorithm for concave network flow problems. *Journal of Global Optimization*, 34:127 – 155, 2006.

[13] Giorgio Gallo, Claudio Sandi, and Claudio Sodini. An algorithm for the min concave cost flow problem. *European Journal of Operational Research*, 4:248 – 255, 1980.

[14] Bernard Gendron, Saïd Hanafib, and Raca Todosijevićb. Matheuristics based on iterative linear programming and slope scaling for multicommodity capacitated fixed charge network design. *European Journal of Operational Research*, 268:78–81, Jul 2018. URL https://www.sciencedirect.com/science/article/pii/S0377221718300432.

[15] Fred Glover. Parametric ghost image processes for fixed-charge problems: A study of transportation networks. *Journal of Heuristics*, 11:307 – 336, 2005. doi: doi.org/10.1007/s10732-005-2135-x.

[16] G.M. Guisewite and P.M. Pardalos. Algorithms for the single-source uncapacitated minimum concave-cost network flow problem. *Journal of Global Optimization*, 1:245–265, 1991. URL https://link.springer.com/article/10.1007/BF00119934.

[17] Nausheen Hashmi, Syed Aqib Jalil, and Shakeel Javaid. A model for two-stage fixed charge transportation problem with multiple objectives and fuzzy linguistic preferences. *Soft Computing*, 23:12401 – 12415, 2019. doi: doi.org/10.1007/s00500-019-03782-1.

[18] Mike Hewitt, George L. Nemhauser, and Martin W.P. Savelsbergh. Combining exact and heuristic approaches for thecapacitated fixed charge network flow problem. *Informs Journal on Computing*, 22:179 – 337, 2010. doi: doi.org/10.1016/j.omega.2009.10.005.

[19] Dorit S. Hochbaum and Arie Segev. Analysis of a flow problem with fixed charges. *Networks*, 19, May 1989. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230190304.

[20] Do Ba Khang and Okitsugu Fujiwara. Approximate solutions of capacitated fixed-charge minimum cost network flow problems. *Networks*, 21:47 – 58, Oct 1991.

[21] Dukwon Kim, Xinyan Pan, and Panos M. Pardalos. Enhanced adjacent extreme-point search and tabu search for the minimum concave-cost uncapacitated transshipment problem. *Computational Economics*, 27:273 – 293, 2006.

[22] Georg Kliewer and Larissa Timajev. Relax-and-cut for capacitated network design. *European Symposium on Algorithms*, 3669:47 – 58, 2005.

[23] Krzysztof Kowalski, Benjamin Lev, Wenjing Shen, and Yan Tu. A fast and simple branching algorithm for solving small scale fixed-charge transportation problem. *Operations Research Perspectives*, 1:1 – 5, 2014.

[24] Bruce W. Lamar. Network optimization problems: Algorithms, applications and complexity. *World Scientific*, pages 147 – 167, 1993.

[25] Gustavo Valentim Loch and Arinei Carlos Lindbeck da Silva. A computational experiment in a heuristic for the fixed charge transportation problem. *International Refereed Journal of Engineering and Science*, 3:1 – 7, 2014.

[26] Carsten Lund and Mihalis Yannakaki. On the hardness of approximating minimization problems. *J. Assoc. Comput.Mach*, 41:960–981, 1994.

[27] Mahmoud M.El-Sherbiny and Rashid M.Alhamali. A hybrid particle swarm algorithm with artificial immune learning for solving the fixed charge transportation problem. *Computers Industrial Engineering*, 64:610 – 620, 2013. doi: doi.org/10.1016/j.cie.2012.12.001.

[28] Richard S. Middleton. A new optimization approach to energy network modeling: anthropogenic CO_2 capture coupled with enhanced oil recovery. *International Journal of Energy Research*, 37(14):1794–1810, 2013.

[29] Richard S. Middleton and Jeffrey M. Bielicki. A scalable infrastructure model for carbon capture and storage: SimCCS. *Energy Policy*, 37(3):1052 – 1060, 2009. ISSN 0301-4215.

[30] Richard S. Middleton, Sean Yaw, Brendan Hoover, and Kevin Ellett. An open-source tool for optimizing co_2 capture, transport, and storage infrastructure. *Environmental Modelling and Software*, 2020. doi: 10.1016/j.envsoft.2019.104560.

[31] Marta S. Rodrigues Monteiro and Dalila B. M. M. Fontes. Locating and sizing bank-branches by opening, closing or maintaining facilities. *Operations Research Proceedings*, 2005:303 – 308, 2005.

[32] Marta S.R. Monteiro, Dalila B.M.M. Fontes, and Fernando A.C.C. Fontes. Concave minimum cost network flow problems solved with a colony of ants. *Journal of Heuristics*, 19:1–33, Feb 2013. URL `https://link.springer.com/article/10.1007/s10732-012-9214-6`.

[33] Artyom Nahapetyan and Panos Pardalos. Adaptive dynamic cost updating procedure for solving fixed charge network flow problems. *Computational Optimization and Applications*, 39:37 – 50, 2008. doi: doi.org/10.1007/s10589-007-9060-x.

[34] Udatta S. Palekar, Mark H. Karwan, and Stanley Zionts. A branch-and-bound method for the fixed charge transportation problem. *Management Science*, 36, Sep 1990. doi: 10.1287/mnsc.36.9.1092.

[35] Hossain Poorzahedy and Omid M.Rouhani. Hybrid meta-heuristic algorithms for solving network design problem. *European Journal of Operational Research*, 182:578 – 596, 2007. doi: 10.1016/j.ejor.2006.07.038.

[36] Steffen Rebennack, Artyom Nahapetyan, and Panos M. Pardalos. Bilinear modeling solution approach for fixed charge network flow problems. *Optimization Letters*, 3:347 – 355, 2009. doi: doi.org/10.1007/s10589-007-9060-x.

[37] Samira Sadeghi-Moghaddam, Mostafa Hajiaghaei-Keshteli, and Mehdi Mahmoodjanloo. New approaches in metaheuristics to solve the fixed charge transportation problem in a fuzzy environment. *Neural Computing and Applications*, 31:477 – 497, 2019. doi: doi.org/10.1007/s00521-017-3027-3.

[38] Berend Smit. Computational carbon capture, 2014. ACM e-Energy 2014 Keynote.

[39] David K Smith and Godfrey A Walters. An evolutionary approach for finding optimal trees in undirected networks. *European Journal of Operational Research*, 102:593 – 602, 2000. doi: 10.1016/S0377-2217(98)00385-3.

[40] S.Molla-Alizadeh-Zavardehi, M.Hajiaghaei-Keshteli, and R.Tavakkoli-Moghaddam. Solving a capacitated fixed-charge transportation problem by artificial immune and genetic algorithms with a prüfer number representation. *Expert Systems with Applications*, 38:10462 – 10474, 2011. doi: doi.org/10.1016/j.eswa.2011.02.093.

[41] Hoang Tuy. The mccnf problem with a fixed number of nonlinear arc costs: Complexity and approximation. In *Approximation and Complexity in Numerical Optimization*, volume 42, pages 525 – 544. 2000. doi: doi.org/10.1007/978-1-4757-3145-3_30.

[42] Caleb Whitman, Sean Yaw, Richard S. Middleton, Brendan Hoover, and Kevin Ellett. Efficient design of CO_2 capture and storage infrastructure. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, e-Energy '19, page 383–384, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366717.

[43] Fanrong Xie and Renan Jia. Nonlinear fixed charge transportation problem by minimum cost flow-based genetic algorithm. *Computers Industrial Engineering*, 63:763 – 778, 2012. doi: doi.org/10.1016/j.cie.2012.04.016.

[44] Masoud Yaghini, Mohsen Momeni, and Mohammadreza Sarmadi. A simplex-based simulated annealing algorithm for node-arc capacitated multicommodity network design. *Applied Soft Computing*, 12:2997 – 3003, 2012. doi: doi.org/10.1016/j.asoc.2012.04.022.

[45] Shangyao Yan, Der-shin Juang, Chien-rong Chen, and Wei-shen Lai. Global and local search algorithms for concave cost transshipment problems. *Journal of Global Optimization*, 33:123 – 156, 2005.

[46] Shangyao Yan, Yu-Lin Shih, and Wang-Tsang Lee. A particle swarm optimization-based hybrid algorithm for minimum concave cost network flow problems. *Journal of Global Optimization*, 49:539–559, 2011.

[47] Sean Yaw, Richard S. Middleton, and Yinzhi Wang. Simccs. `https://github.com/simccs/SimCCS`, 2018.

[48] Sean Yaw, Richard S. Middleton, and Brendan Hoover. Graph simplification for infrastructure network design. *Combinatorial Optimization and Applications*, pages 576–589, 2019. doi: https://doi.org/10.1007/978-3-030-36412-0_47.

[49] Komeil Yousefi, Ahmad J. Afshari, and Mostafa Hajiaghaei-Keshteli. Solving the fixed charge transportation problem by new heuristic approach. *Journal of Optimization in Industrial Engineering*, 12(1):41–52, 2019. ISSN 2251-9904. doi: 10.22094/joie.2017.738.1469. URL `http://www.qjie.ir/article_538020.html`.