Fast Method for Maximum-Flow Problem with Minimum-Lot Sizes

Submitted by

Vithya Ganeshan

In partial fulfillment of Master of Science
in Informatics

University of Bergen,
Bergen, Norway

March 3, 2015

Supervised by

Prof. Dag Haugland

Acknowledgements

I would like to express my deepest thanks to my supervisor Professor Dag Haugland for his support and patience. His guidance helped me in many ways in shaping up the thesis work. I thank Mari Garaas Lochen, student advisor, for her help especially in getting the test system up in the critical times, I am also thankful to the system staffs who are involved in fixing the CPLEX problems in the test system. I am also indebted to the members of the Algorithms and Optimization groups with whom I have interacted during the course of my studies, particularly Professor Jan Arne Telle and Professor Fredrik Manne. Most importantly, none of this would have been possible without the love and patience of my family.

Abstract

In transportation networks, such as pipeline networks for transporting natural gas, it is often impractical to send across amounts of flow below a certain threshold. Such lower threshold is referred as the minimum-lot size. The network flow is semi-continuous when a network has minimum-lot sizes. In other terms, the flow can be either zero or within the limits of minimum-lot and maximum capacity of the network. When a network includes minimum-lot constraints, the problem of finding maximum-flow becomes complex and exact methods tend to be too time consuming. Since it is not generally required that the solution methods provide the optimal solution, this master thesis proposes a fast (inexact) method to find near-optimum solution. Also, the proposed fast method is experimentally validated and compared with the other relevant approaches available in the literature, and the results are analyzed in detail.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background

Network optimization makes a large part of combinatorial optimization, and presents a model often used for a large number of real-world applications in communications, informatics, transportation, construction projects, water resources management and supply chain management [1]. Linear Programming (LP) models exhibit a special structure that can be exploited in the construction of efficient algorithms to solve a problem. Historically, the first of these special structures to be analyzed was the transportation problem, which is a particular type of network problem. The development of an efficient solution procedure for this problem resulted in the first widespread application of LP to problems of industrial logistics [2]. A common scenario of a network-flow problem arising in industrial logistics concerns the distribution of a single homogeneous product from plant (source) to consumer market (terminal). The product need not be sent directly from source to destination, but may be routed through intermediary points reflecting warehouses or distribution centers. Further, there may be capacity restrictions that limit some of the shipping links.

1.2 Natural Gas Transportation: An Application

Norway being the largest natural gas producer in the Western Europe[1], the gas produced is taken first from the reserves using several pipelines and brought to the shore to be distributed in Norway or to be exported to other countries. Installing and maintaining pipelines involves huge amount of costs[2]. Subsequently, using those pipes incurs cost for the industry. Oil and gas industry is

[1] http://www.eia.gov/countries/country-data.cfm?fips=no

[2] http://www.ogj.com/articles/print/volume-111/issue-02/special-report–worldwide-pipeline-construction/worldwide-pipeline-construction-crude-products.html

no exception in trying to reduce the cost of production to keep up the profit margins. This involves optimizing the usage of pipes in such a way that the cost incurred is low without compromising the amount of oil transferred. In other words, the revenue generated should be maximum while having the production costs at minimum.

Optimum route calculations should obey the capacity constraints of the network, which defines the maximum amount of gas flow allowed between two points that are connected by a pipe. In gas transportation networks, it is often impractical to send across amounts of flow below a certain threshold. In other terms, the amount of gas to be transported should be between the amounts of lower and upper thresholds allowed. Some of the reasons for setting lower threshold limits are (i) operations might require lots to be large in order to be cost effective, (ii) the products appear only in batches of a minimum size, and (iii) underlying mechanical and chemical processes require a minimum level of operation. That is, either the quantity that can be shipped must be zero, or it must be between the minimum threshold and maximum capacity. In other terms, minimum threshold shows the minimum quantity that must be transported to enable a flow network to function. Otherwise the respective path has to be closed. Such a minimum threshold is referred as *minimum-lot sizes* in this paper.

Finding an optimum route to transfer the desired amount (often, as maximum as possible) of gas between source and terminal is a challenge. It is because the pipeline structure is a complex network and its constraints are dynamic in its nature, i.e., the network has different capacity constraints in different places, some parts have minimum-lot sizes and some parts do not have and it varies from time to time. Since the business requirements vary, finding such optimum routes is a frequent task in the gas transportation industry.

1.3 Maximum-Flow Problem with Minimum-lot Sizes

In the context of transportation, maximum-flow problem aims to maximize the flow of a network from a given source to a given terminal with maximum capacity constraints. Maximum-flow problem with minimum-lot sizes is a variation of maximum-flow problem, which includes minimum-lot size (i.e., minimum capacity) constraints in addition to the capacity constraints. For the sake of illustration, a directed graph is shown in Figure 1.1 with capacity and minimum-lot constraints. The sources, destinations, and intermediate points are collectively called as *nodes* of the network, and the transportation links connecting nodes are called as *arcs*.

In this work, we consider a directed graph, where a minimum lot size and a flow capacity are defined for each arc, and study the problem of maximizing the flow from a given source to a given terminal. The nodes are represented by numbered circles and the arcs by arrows. A generic arc is denoted as (i, j), in

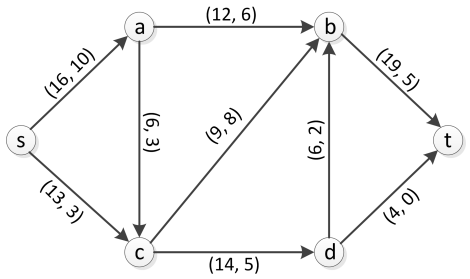

Figure 1.1: A graph showing maximum-flow problem with minimum-lot sizes

which the arc connects the nodes i and j and it has a flow from the node i to the node j. Note that some pairs of nodes, for example s and t, are not connected directly by an arc. The graph also includes some additional characteristics and these are exhibited in the figure, i.e., minimum-lot size and capacity constraints are assigned to each arc. These characteristics are shown next to each arc. Thus, the flow on arc (a, b) must be between 6 and 12 units or zero. In this case, node s is the source node and node t is the terminal node. The remaining nodes have no net supply or demand; they are intermediate points, often referred to as transshipment nodes. It is a property that the intermediate nodes must not store or leak any flow, i.e., the material that is received by an intermediate node must be sent out from that node. It is called *conservation property* of the network.

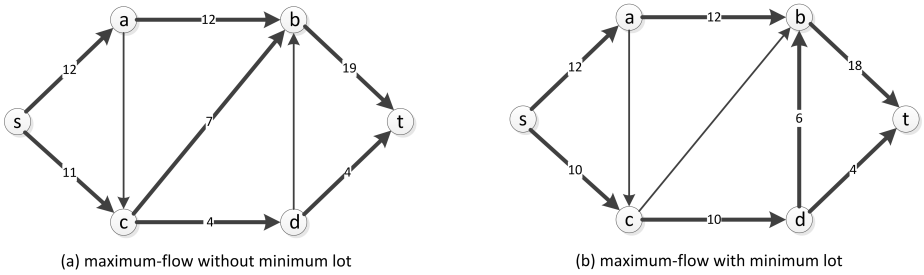

Figure 1.2: Solutions to the maximum-flow problem given in Figure 1.1. (a) Without minimum-lot, and (b) With minimum-lot

Figure 1.2(a) shows the solution to the maximum-flow problem given in Figure 1.1 without considering the minimum-lot sizes. The maximum-flow is 23 units, in this case. (s, a) transports 12 units, (a, b) transports 12 units, (b, t) transports 19 units, arc (s, c) transports 11 units, arc (c, d) transports 4 units, arc (c, b) transports 7 units, and arc (d, t) transports 4 units. The remaining arcs, i.e., (a, c) and (d, b), are not used.

Figure 1.2(b) shows the solution to the maximum-flow problem given in Figure 1.1 with the minimum-lot sizes. The maximum-flow is 22 units, in this

case. (s, a) transports 12 units, (a, b) transports 12 units, (b, t) transports 18 units, arc (s, c) transports 10 units, arc (c, d) transports 10 units, arc (d, b) transports 6 units, and arc (d, t) transports 4 units. The remaining arcs, i.e., (a, c) and (c, b), are not used. From this, it can be noticed that the minimum-lot size constraint influences the maximum-flow, in terms of total number of units that can be transferred from a given source to a given terminal and the path to be used in order to achieve it.

1.4 Organization

This thesis is organized as follows. Section 2 describes and defines maximum-flow problem with minimum-lot sizes. The literature survey of the problem is given in Section 3. The computational complexity, theoretical properties, MIP formulation, and relaxation are given in Section 4. Following that, the proposed fast method is described in Section 5. The details of the experiments conducted are reported and the results are analyzed in Section 6.6. Finally, the conclusion is drawn in Section 7.

Chapter 2

Problem

2.1 Description

In transportation networks, it is often impractical to send across amounts of flow below a lower threshold, which is referred as minimum-lot sizes. Maximum-flow problem in directed graphs with minimum lot size constraints on the arcs imposes either zero flow or flow between the lower and upper capacity. In such instances, the problem becomes much more difficult to solve, and exact methods tend to be too time consuming for practical use. In practice, it is not generally required to find the optimal solution. So, fast (inexact) methods that can provide near-optimum solutions are sufficient to the said problem. This thesis proposes an efficient computational method that can produce near optimal solutions with modest computational effort, despite the intractability of the problem.

2.2 Formal Definition

A formal definition of the maximum-flow problem with minimum-lot sizes is as follows:

Let $G = (N, A)$ be a directed graph with node set N and arc set A, and nonnegative integer vectors l and u representing lower and upper flow bounds, respectively.

Here, we assume that G contains a unique source $s \in N$ and a unique terminal $t \in N$, and that A contains a circulation arc (t, s) with $l_{ts} = 0$ and $u_{ts} = \infty$ from the terminal to the source.

Further, we consider the problem of maximizing the flow from the source through the network to the terminal, such that the flow at each arc is either zero or between the two bounds. This is equivalent to maximizing the flow recycled along arc (t, s). By defining the set of circulations in G as,

$$F(G)=\left\{ x \in \mathbb{R}_+^A: \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} \ \forall i \in \mathrm{N} \right\},$$

The problem is expressed as:

$$max \quad x_{ts}, \tag{2.1}$$

$$x \in F(G), \tag{2.2}$$

$$x_{ij} \in \{0\} \cup [l_{ij}, u_{ij}] \ \ (i,j) \in A. \tag{2.3}$$

Henceforth, we say that arc (i,j) is closed if $x_{ij} = 0$, and open otherwise. We let $X \subseteq A$ be the set of open arcs, and let $\overline{X} = A \setminus X$ be the set of closed arcs. Let G_X denote the subgraph with node set N and arc set X. We say that X is feasible if there exists some flow vector $x \in F(G)$ satisfying $l_{ij} \leq x_{ij} \leq u_{ij}$ for all $(i,j) \in X$, and $x_{ij} = 0$ for all $(i,j) \in \overline{X}$. Let $z(X)$ denote the maximum value x_{ts} can take under these conditions, and let $z(X) = -\infty$ if X is infeasible. Observe that the empty set is feasible with $z(\emptyset) = 0$.

Chapter 3

Literature Survey

Haugland et al. introduced the maximum-flow problem with minimum lot size constraints in [3]. The authors have shown how the problem can be approached using Lagrangean relaxation [4], based on Mixed Integer Programming (MIP) formulation. This approach involves a method for computing strong upper bounds on the maximum-flow, and a method for fixing binary variables based on the upper bounds. Also, the authors suggested a construction heuristic for the problem and presented results obtained from some computational experiments. In that, it is shown that the minimum-lot size constraints do not affect the maximum-flow when the percentage of arcs with nonzero minimum-lot is below 50% in a network. Maximum-flow drops below LP bound when the percentage of nonzero minimum-lot is between 50% to 70% in a graph. The suggested heuristics provides the trivial zero solution when the percentage of nonzero minimum-lot is above 70%. In other terms, the given construction heuristics struggles to provide a useful solution when the percentage of nonzero minimum-lot is above 70% in a graph. It is mentioned that it is due to the greedy nature of the heuristics that were used to start finding the path. Since the intension of developing this heuristics is close to the aim of this thesis work, we considered this as an alternative method that can be compared with the method proposed in this thesis work in addition to exact method and LP.

In [5], Eleyat et al. has shown that most of the execution time is spent on solving a series of regular maximum-flow problems and developed a parallel augmenting path algorithm to accelerate the heuristics by an average factor of 1.25. This suggested a near optimal solution and investigated a way to parallelize the method, to reduce the time spent on finding the maximum-flow in a network with capacity and minimum-lot size constraints on each pipe [6]. Both the heuristic and its parallelization were implemented and tested on the $2*6$ core AMD Opteron processor. Since this thesis work is not concerned about developing parallel algorithms, the experimental results obtained by Eleyat et al. are not comparable in this thesis work.

Thielen et al. came up with the maximum-flow problem with minimum quan-

tities in [7], and applied it for Generalized Assignment Problem (GAP) [8, 9], network flow problem [7], and packing problem [10]. Since formally both minimum quantities and minimum-lot sizes require two bounds (lower and upper) on an arc, minimum quantities are just another name for minimum-lot sizes and that is relevant in terms of the application the authors considered in the paper (i.e., in a wastewater system, a used pipe will get clogged unless at least a minimum amount of water runs through it). As expected, Thielen et al. has shown the problem is strongly NP hard to approximate on general graphs (and even bipartite graphs) within any positive factor. However, the authors have presented a pseudo-polynomial time dynamic programming algorithm for the problem, on series parallel graphs. Also the authors show that the problem is still weakly NP-complete when the minimum quantity is same for each arc in the network. $(2 - \frac{1}{\lambda})$ approximation algorithm is presented for this case, where $\lambda \geq 2$ denotes the common minimum quantity of all arcs (for $\lambda \in \{0, 1\}$, the problem can be solved optimally in strongly polynomial time as a standard maximum-flow problem without minimum quantities). Further, it is shown that the problem on series-parallel graphs can be solved in strongly polynomial time when the minimum quantity is the same for each arc in the network. Although minimum quantities and minimum-lot sizes are same in terms of maximum-flow problem, this thesis work could not consider the computational results given in [7] or [8] or [9] as a reference because these papers considered series parallel graphs and/or approximation algorithm to find solution to the problem. Whereas, this thesis work considers heuristics to provide solution to the maximum-flow problem with minimum-lot sizes, and do not consider the specific cases like series parallel graph and/or approximation techniques.

Nobibon et al. [11] considered resource loading decision problem with no upper bounds (i.e., capacity constraints) and shown that as a special variation of maximum-flow problem with minimum-lots in a bipartite networks. Since it has considered only bipartite networks and did not include capacity constraints, the results obtained in this paper could not be compared directly in this thesis.

Chapter 4

Mathematical Formulation

4.1 Computational Complexity

Minimum-lot size is a new constraint to maximum-flow problem. The constraint, minimum-lot size, imposes either zero flow or flow between the minimum and maximum capacity of a network. The distinctive nature of this constraint makes the problem NP-hard and it is proven by Haugland et al. in [3]. Further, Eleyat et al. proven the maximum-flow problem with minimum-lot sizes is strongly NP hard in [5].

4.2 Theoretical Properties

The NP-hardness shows that exact methods are unlikely to solve big instances of the problem in practical time. This means that exact solutions to the instances of realistic sizes of the problem are computationally infeasible. Hence, inexact solutions that are fast and correct but may not be optimal are the practical ones to the problem.

4.3 MIP Formulation

Mixed Integer Programming (MIP) deals with optimization techniques in which an objective function is optimized subject to both equality and inequality constraints, where two types of variables can be specified: continuous variables which can take any real value within given bounds, and binary variables which can take only 0 or 1 values. The unique feature of MIP is precisely the capability of handling the latter type of variables which in application problems will be

associated to discrete decisions in combinatorial problems. The three major features that can be accomplished with the mixed integer formulation [12] are:

(i) structural and parameter optimization can be performed simultaneously,

(ii) discrete and logical constraints can be handled explicitly with the binary variables, and

(iii) the mathematical representation provides a general systematic framework since one can formulate a variety of different synthesis problems with the same mathematical tool.

Since the maximum-flow problem with minimum-lot sizes has a semi continuous variable to represent the flow in an arc (i.e., Equation 2.3 in Section 2), we choose to go with MIP to solve the problem. MIP formulation follows.

The objective of the problem is to maximize the flow in the recycling arc from terminal to source.

$$max \quad x_{ts}, \tag{4.1}$$

The material that comes to an intermediate node must be sent out from that node, as per the conservation property. In other terms, sum of all the inflow of an intermediate node should be equal to the sum of all the outflow of that node.

$$x \in F(G), \tag{4.2}$$

The arcs have a minimum-lot l_{ij} and a maximum capacity u_{ij}, where the maximum capacity is more than or equal to minimum-lot, i.e., $u_{ij} \geq l_{ij}$. The flow in an arc x_{ij} could be either zero or between the minimum-lot and the maximum capacity. We introduce a new binary variable, y_{ij}, to indicate whether the arc, (i, j), is open or not. Hence, if the arc, (i, j), is closed then the binary variable $y_{ij} = 0$, indicating that the arc's flow $x_{ij} = 0$.

$$y \in \{0, 1\}^A, \tag{4.3}$$

Here, $y_{ij} = 1$ if and only if $x_{ij} \neq 0$. In addition to that, if the arc is open (i.e., $y_{ij} = 1$) then the value of x_{ij} should be greater than l_{ij}, i.e.,

$$x_{ij} - l_{ij} y_{ij} \geq 0, (i, j) \in A, \tag{4.4}$$

Similarly, if the arc is open then the flow x_{ij} should be less than the capacity u_{ij}.

$$x_{ij} - u_{ij} y_{ij} \leq 0, (i, j) \in A, \tag{4.5}$$

4.4 Relaxation

An essential task in designing any solution algorithm for a problem is to derive an optimal condition or a stopping criterion to terminate the algorithm, i.e., to judge if the current solution is optimal to the problem or to conclude that

there is no feasible solution to the problem. An upper bound of the problem can be used as a stopping criterion in a solution algorithm, in branch and bound algorithms[1].

Relaxation extends the feasible region of the original problem. In the extended feasible region, the objective function has a value that is at least as big as the objective function of the original problem. So, the optimal answer of the relaxed problem is an upper bound of the original problem. For example, consider the following optimization problem[2] where $f : \mathbb{R}^n \to \mathbb{R}$ and $S \subseteq \mathbb{R}^n$:

$$\begin{gathered}\text{Maximize } f(x)\\ \text{subject to } x \in S\end{gathered}$$

A relaxation of the above problem has the following form:

$$\begin{gathered}\text{Maximize } f_R(x)\\ \text{subject to } x \in S_R\end{gathered}$$

where $f_R : \mathbb{R}^n \to \mathbb{R}$ is such that $f_R(x) \geq f(x)$ for any $x \in S$ and $S \subseteq S_R$. It is clear that the optimal solution f_R^* of the relaxation is an upper bound of the optimal solution of the initial problem.

4.4.1 Continuous Relaxation

Continuous relaxation allows variables that are constrained to take only integer values to take non-integer values, in an arbitrary problem, . For example, consider the model given in Section 4.3. In Equation 4.3, the variable y is constrained to take discrete values, i.e., either 0 or 1, which is formally represented as $y \in \{0, 1\}$. Applying continuous relaxation to the variable y, the variable y can take any value between 0 and 1, which is formally represented as $y \in [0, 1]$. The continuous relaxation increases the feasible region of a variable, without affecting the objective function of the problem.

The following is the continuous relaxed form of the model given in Section 4.3.

$$max \quad x_{ts}, \tag{4.6}$$

$$x \in F(G), \tag{4.7}$$

$$y \in [0, 1]^A, \tag{4.8}$$

$$x_{ij} - l_{ij}\, y_{ij} \geq 0, (i, j) \in A, \tag{4.9}$$

$$x_{ij} - u_{ij}\, y_{ij} \leq 0, (i, j) \in A, \tag{4.10}$$

From the continuously relaxed model, it can be observed that the objective function is not changed. However, the problem now becomes a polynomially solvable linear program. By solving this, we can arrive at an upper bound for the model explained in the Section 4.3.

[1] http://support.sas.com/documentation/cdl/en/ormpug/63352/HTML/default/viewer.htm#ormpug_optmilp_sect010.htm

[2] http://www.ens-lyon.fr/DI/wp-content/uploads/2012/01/LagrangianRelax.pdf

4.4.2 Lagrangian Relaxation

Lagrangian relaxation approximates a difficult problem of constrained optimization by a simpler problem. A solution to the relaxed problem is an approximate solution to the original problem, and provides useful information. Lagrangian relaxation involves removing one or more constraints from a problem. Such removal is penalized with a penalty variable, i.e., each constraint is dualized and then added to the objective function. The penalty variable is referred as the dual variable or Lagrange multiplier. Since one or more constraints may be dualized it is possible to generate as many as $2^k - 1$ subproblems, where k is the number of constraints. The subprograms differ, depending on the constraints that got dualized. In practice, this relaxed problem can often be solved more easily than the original problem.

Lagrangian relaxation for the maximum-flow problem with minimum-lot sizes is follows.

First, the minimum-lot size constraints should be removed from MIP formulation (Section 4.3) of the problem,

$$x_{ij} - l_{ij} y_{ij} \geq 0, (i,j) \in A, \tag{4.11}$$

Next, penalty, Lagrangian multiplier ($\lambda_{ij} \in \mathbb{R}_+^A$), should be added to the objective function of the problem,

$$max \quad x_{ts} + \lambda_{ij}(x_{ij} - l_{ij} y_{ij}), \tag{4.12}$$

$$x \in F(G), \tag{4.13}$$

$$y \in \{0,1\}^A, \tag{4.14}$$

$$x_{ij} - u_{ij} y_{ij} \leq 0, (i,j) \in A, \tag{4.15}$$

Similar to minimum-lot size constraints, if we remove capacity constraints and add penalty, Lagrangian multiplier ($\mu_{ij} \in \mathbb{R}_+^A$), then the objective function becomes,

$$max \quad x_{ts} + \mu_{ij}(u_{ij} y_{ij} - x_{ij}), \tag{4.16}$$

$$x \in F(G), \tag{4.17}$$

$$y \in \{0,1\}^A, \tag{4.18}$$

$$x_{ij} - l_{ij} y_{ij} \geq 0, (i,j) \in A, \tag{4.19}$$

We reach the following by making a formulation without the capacity and the minimum-lot size constraints,

$$max \quad x_{ts} + \mu_{ij}(u_{ij} y_{ij} - x_{ij}) + \lambda_{ij}(x_{ij} - l_{ij} y_{ij}), \tag{4.20}$$

By assumption, we can efficiently compute the optimal value for the relaxed problem with a fixed vector λ.

Chapter 5

Proposed Fast Method

5.1 Construction Method: Constrained Edmonds-Karp Algorithm

The Edmonds-Karp Algorithm [13] finds the maximum flow of a network when the capacities are given, using Breadth First Search (BFS) [13] to order the paths from a given source to a given terminal. To find the maximum flow with minimum-lot sizes, we have added minimum-lot constraints to the Edmonds-Karp Algorithm, and it is given in the Algorithm 1.

The algorithm functions as follows. It uses BFS to find path $p \in G$ from the source s to the terminal t (`line 1`). The possible maximum-flow of the path $c_f(p)$ is calculated by finding the minimum of the (remaining) capacities of each arc, i.e., $c_f(u, v) : (u, v)$ is in p (`line 2`). For each path, the maximum of all the minimum-lots of arcs $(m_f(u, v) : (u, v)$ is in $p)$ is calculated and it is set as the minimum-lot of the path $m_f(p)$ (`line 3`). The feasible flow $f_f(p)$ is found from $c_f(p) - m_f(p)$ (`line 4`). If any of the arcs do not satisfy the minimum-lot constraints of the path, the flag *sendFlow* is set to FALSE (`lines 5-7`). If the path has all edges satisfying the minimum-lot constraints, the flag *sendFlow* is set to TRUE (`line 8`). After that, the flow is sent through that path and the feasible flow $f_f(p)$ is applied to each arc in the same path, and the minimum flow of each arc in the path is reduced by the feasible flow (`lines 9-12`). And then, this path is added to G_r (`line 14`). The same procedure is followed for each of the path returned by the BFS. The resulting graph is G_r returned after processing all the paths .

Since the Edmonds-Karp algorithm finds the maximum-flow of a network in $\mathcal{O}(VE^2)$, the maximum-flow with minimum-lot sizes can also be found in $\mathcal{O}(VE^2)$. However, optimality of the solution returned by this algorithm may

Algorithm 1: Constrained-Edmonds-Karp (G,s,t)

Input: Graph G, Source s, and Terminal t
Output: Graph containing only feasible paths G_r

while *there exists a path p from s to t from BFS (G,s,t))* **do**
  $c_f(p) = min\{c_f(u,v) : (u,v) \text{ is in } p)\}$
  $m_f(p) = max\{m_f(u,v) : (u,v) \text{ is in } p)\}$
  $f_f(p) = (c_f(p) - m_f(p))$
  if $f_f(p) < 0$ **then**
    $sendFlow =$ FALSE
  if $sendFlow =$ TRUE **then**
    for *each arc* (u,v)*in p* **do**
      $(u,v).f = (u,v).f + f_f(p)$
      $m_f(u,v) = m_f(u,v) - f_f(p)$
      $c_f(u,v) = c_f(u,v) - (u,v).f$
  add p to G_r
return G_r

vary depending on the order in which paths are picked and it might not be optimal.

Proposition 5.1.1. *This algorithm provides a feasible flow.*

Proof. In first step, the flow sent throw each arc is at least as much as the minimum-lot of that arc, and in each subsequent iterations adds flow. Hence, $(u,v).f \geq m_f(u,v)\ \forall\ (u,v) \in A$ □

5.2 Improvement Method

The construction method given in the Section 5.1 could not add a path in the final solution when the path do not fulfill the minimum-lot constraint. However, it might as well be part of the optimal solution based on accumulation of flow passing through from other paths and by that fulfilling the minimum lot constraints. This is particularly the case when a path includes a very high minimum-lot constraint while all or most of its neighboring inflow and outflow arcs have smaller capacity. Considering this, we improve the construction method by adding all the flow that could not be added to construction method into a reservoir network and then we run the construction method (`lines 7 and 17` in Algorithm 2). If all the arcs of a path fulfill the minimum-lot constraints then that path is added to the result.

Since the improved method uses the construction algorithm, the running time of this algorithm changes by a linear factor. So, the complexity still remains at $\mathcal{O}(VE^2)$.

Algorithm 2: Improved-and-Constrained-Edmonds-Karp (G,s,t)

Input: Graph G, Source s, and Terminal t
Output: Graph containing only feasible paths G_r

while *there exists a path p from s to t from BFS (G,s,t))* **do**
  $c_f(p) = min\{c_f(u,v) : (u,v)$ is in $p)\}$
  $m_f(p) = max\{m_f(u,v) : (u,v)$ is in $p)\}$
  $f_f(p) = c_f(p) - m_f(p)$
  if $f_f(p) < 0$ **then**
    $sendFlow =$ FALSE
    add p to $G_{reserve}$
  if $sendFlow =$ TRUE **then**
    for *each arc* (u,v)*in* p **do**
      $(u,v).f = (u,v).f + f_f(p)$
      $m_f(u,v) = m_f(u,v) - f_f(p)$
      $c_f(u,v) = c_f(u,v) - (u,v).f$
  add p to G_r
$G_r = G_r \cup$ Constrained-Edmonds-Karp($G_{reserve}$,s,t)
return G_r

Chapter 6

Experimentation

6.1 Purpose

The purpose of this experimentation was to compare the fast method proposed in this thesis (Section 5) with the methods given by Haugland et al. in [3] and to analyze its performance including the execution stability.

6.2 Input

Table 6.1: Base Graphs

Graph	Parameters					
	a	**b**	c_1	c_2	$\|N\|$	$\|A\|$
$\mathcal{GL}$	15	5	2000	10,000	1,125	5,100
$\mathcal{GL}_1$	6	31	1	10,000	1,116	4,800
$\mathcal{GL}_2$	9	100	1	10,000	8,100	36,819
$\mathcal{GW}$	5	15	2000	10,000	112	432
$\mathcal{GW}_1$	16	4	1	10,000	1,024	4,608
$\mathcal{GW}_2$	37	6	1	10,000	8,214	38,813

Two graphs, namely $\mathcal{GW}$ and $\mathcal{GL}$, used to conduct experiments in [3], used to evaluate the proposed fast method given in Section 5 in comparison with the methods proposed by Haugland et al. Four graphs, namely $\mathcal{GL}_1$, $\mathcal{GL}_2$, $\mathcal{GW}_1$, and $\mathcal{GW}_2$, used to test the proposed method further. Those graphs were generated using RMFGEN-generator of Goldfarb and Grigoriadis [14], by supplying four parameter values, namely a, b, c_1, and c_2, as the input. The parameter details and the sizes of the graphs are shown in Table 6.2. To generate the capacities

of in-frame arcs randomly, the generator was modified to produce capacities in the range of $[c_2, c_2a^2]$. Eight instances were generated for two graphs and four instances were generated for other six graphs, by changing the percentage of arcs which has minimum-lot sizes. The percentage range varies from is 30% to 100%. For example, the graph $\mathcal{GL}$-30 has 30% arcs with minimum-lot size constraints. These arcs were selected randomly by drawing from the entire arc set A, and for each of them, l_{ij} was randomly generated in the range $[u_{ij}/4, u_{ij}]$

6.3 Software

- ILOG CPLEX[1] (Version 12.51) is used as the solver to solve LP problem.
- Java[2] Version 1.7 is used with an Application Programming Interface (API) provided by ILOG CPLEX to call CPLEX directly via the Java Native Interface[3] (JNI). The Java interface is built on top of ILOG Concert Technology[4].
- Eclipse[5] is used as an Integrated Development Environment (IDE) for developing the proposed fast method as a Java application.
- Ubuntu release 12.04 is used as the operating system.

6.4 Hardware

A standalone computer was used for all the development and experimentation purposes. It had two 64-bit cores (Intel x86_64 architecture), each core's speed is 2GHz, and 8GB RAM.

6.5 Implementation

The following are implemented for this experimentation purposes.

(i) The fast method proposed in Section 5,

(i) The optimal flow was produced by supplying MIP formulation given in the Section 4.3, and

(iii) Upper bound was computed by using the LP-relaxation of the model given in the Section 4.4.1.

[1] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[2] https://www.java.com/en/
[3] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/
[4] http://www-01.ibm.com/software/commerce/optimization/interfaces/
[5] https://eclipse.org/

6.6 Results and Analysis

6.6.1 Proposed and Existing Methods

Table 6.2 shows the instances of the graphs used as the input in this test, the optimum flow, LP-bound, the flow obtained by Haugland et al. in [3], and the flow generated by the method proposed in this thesis, i.e., the Algorithm 2 given in Section 5.2 (hereinafter referred to as *proposed method*).

Table 6.2: Instances, and its optimal flow, LP-bound, and flow produced by existing and proposed fast method

Instance	**Optimal**	**LP-bound**	**Fast Methods**			
			Haugland et al.			**Proposed**
			H_1	H_2	H_3	V_1
$\mathcal{GW}$-30	130,587	130,587	130,587	130,587	130,587	55,837
$\mathcal{GW}$-40	130,587	130,587	130,587	130,587	130,587	48,775
$\mathcal{GW}$-50	130,587	130,587	34,714	34,714	20,563	37,205
$\mathcal{GW}$-60	96,971	130,587	12,255	17,394	0	27,903
$\mathcal{GW}$-70	77,587	130,587	0	0	0	19,892
$\mathcal{GW}$-80	47,639	130,587	0	0	0	15,603
$\mathcal{GW}$-90	47,639	130,587	0	0	0	11,735
$\mathcal{GW}$-100	47,639	130,587	0	0	0	8,474
$\mathcal{GL}$-30	1,338,057	1,338,057	1,338,057	1,338,057	1,338,057	608,761
$\mathcal{GL}$-40	1,338,057	1,338,057	1,338,057	1,338,057	1,338,057	465,149
$\mathcal{GL}$-50	1,338,057	1,338,057	1,046,717	3111	123,895	378,240
$\mathcal{GL}$-60	1,335,045	1,338,057	0	0	0	283,816
$\mathcal{GL}$-70	1,335,045	1,338,057	0	0	0	187,869
$\mathcal{GL}$-80	1,335,045	1,338,057	0	0	0	142,759
$\mathcal{GL}$-90	1,335,045	1,338,057	0	0	0	74,462
$\mathcal{GL}$-100	1,335,045	1,338,057	0	0	0	44,145

In the practical situations, it is expected that almost all the arcs has minimum-lot size constraints in a network. In such situation, the methods (i.e., H_1, H_2, and H_3) proposed by Haugland et al. could not generate a flow when the percentage of minimum-lot size constraints are more than 60% in the case of $\mathcal{GL}$ graph and more than 70% in the case of $\mathcal{GW}$ graph, which can be observed from the Table 6.2. The method proposed in this thesis performs better and could produce flow even when a network has 100% minimum-lot size constraints. For example, our method (V_1) produced the flow of $283,816$ units in the graph $\mathcal{GL}$-60, whereas the Haugland et al. methods can produce only 0 as the possible flow. Promisingly, the proposed method functioned in all the graphs that are taken in this experimentation. When there were 100% minimum-lot size constraints, the method proposed produced the flow of $44,145$ units in the graph

$\mathcal{GL}$-100, and 8,474 units in the graph $\mathcal{GW}$-100.

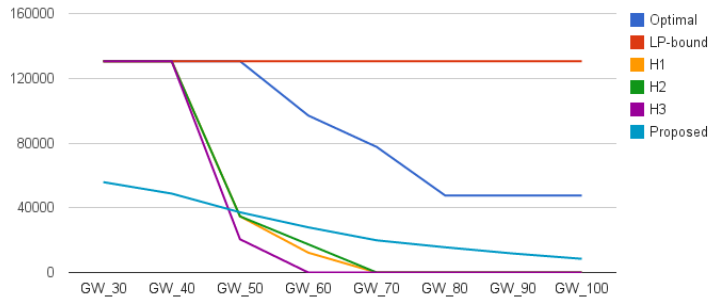

Figure 6.1: Performance trend of the fast methods available and proposed in comparison with the optimal flow and LP-bound, in the case of $\mathcal{GW}$ graph.

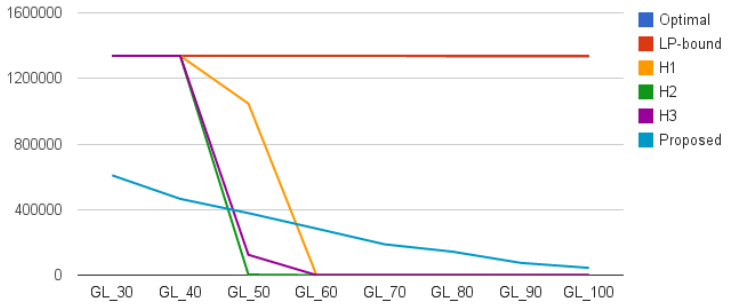

Figure 6.2: Performance trend of the fast methods available and proposed in comparison with the optimal flow and LP-bound, in the case of $\mathcal{GL}$ graph.

Figures 6.1 and 6.2 show the performance trend of the fast methods given by Haugland et al. and the method proposed in this thesis, in a graphical manner. In that, Figure 6.1 shows the case of $\mathcal{GW}$ graph, and Figure 6.2 shows the case of $\mathcal{GL}$ graph. From these figures, it can be observed that the proposed method linearly drops the maximum-flow when the percentages of minimum-lot sizes increases. Also, the flow never touches zero. Whereas, Haugland et al's methods drops suddenly when the percentage of minimum-lot sizes touches 40% ($\mathcal{GL}$ graph) to 50% ($\mathcal{GW}$ graph) and reaches zero after 60% ($\mathcal{GL}$ graph) to 70% ($\mathcal{GW}$ graph). However for 30% to 50% of constraints, Haugland et al's methods

perform better than the method proposed in this thesis. It shows that Haugland et al's methods are better suited when a network's minimum-lot constraints are up to 50% or 60%, and the proposed method is well suited when the constraints percentage goes more beyond 50 to 60%.

Table 6.3: CPU time taken by the exact and the proposed method

Instance	**CPU Time (Seconds)**	
	Exact	**Proposed**
$\mathcal{GW}$_30	1.16	0.05
$\mathcal{GW}$_40	1.27	0.051
$\mathcal{GW}$_50	1.41	0.076
$\mathcal{GW}$_60	2.0	0.076
$\mathcal{GW}$_70	3.5	0.076
$\mathcal{GW}$_80	39.52	0.075
$\mathcal{GW}$_90	7870	0.076
$\mathcal{GW}$_100	5623	0.076

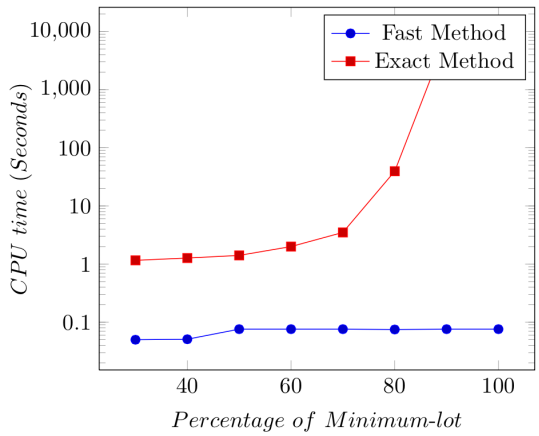

Figure 6.3: Percentage of Minimum-lot sizes Vs. CPU time in seconds, in the case of $\mathcal{GW}$ graph

The CPU time required to find a solution to different instances of $\mathcal{GW}$ graph is shown in the Table 6.3 and it is plotted in Figure 6.3. From that, it can be observed that the proposed method consumed much lesser time than the exact method. For example in the case of $\mathcal{GW}$_90, the proposed method took only 0.076 seconds, whereas the exact method took 7870 seconds. Interestingly, the proposed method did not vary the computation time in terms of the percentage of minimum-lot size constraints in the graph. Whereas, the exact method exponentially increased the time required to process the graph when the percentage

of minimum-lot size constraints increased, especially when it goes above 70%.

6.6.2 Performance and Execution Stability

To make sure that the method proposed can perform and produce a flow even when there are 100% minimum-lot constraints in a network, we tested the method with some more graphs and those results are reported in Table 6.4 and 6.5. Since none other than us have tested these graphs before, we compared the flow that was generated by the proposed method with the optimal flow and LP-bound. From the Table 6.4, it can be observed that the proposed method generated the flow of $8,127$ units in the graph $\mathcal{GL}_1$-100 and $78,480$ units in the graph $\mathcal{GW}_1$-100.

Table 6.4: Instances, and its optimal flow, LP-bound, and flow produced by the proposed fast method

Instance	**Optimal**	**LP-bound**	**Proposed Method**
$\mathcal{GL}_1$-25	135,671	135,671	74,425
$\mathcal{GL}_1$-50	135,671	135,671	43,931
$\mathcal{GL}_1$-75	135,671	135,671	26,738
$\mathcal{GL}_1$-100	135,671	135,671	8,127
$\mathcal{GW}_1$-25	1,249,160	1,249,160	629,191
$\mathcal{GW}_1$-50	1,249,160	1,249,160	350,611
$\mathcal{GW}_1$-75	1,249,160	1,249,160	162,680
$\mathcal{GW}_1$-100	1,249,160	1,249,160	78,480

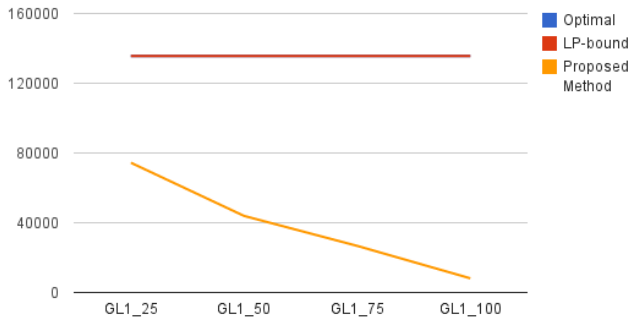

Figure 6.4: Performance trend of the fast methods available and proposed in comparison with the optimal flow and LP-bound, in the case of $\mathcal{GL}_1$ graph.

Performance trends observed from the Figures 6.4 and 6.5 continued to show the similar trend that we observed in the Figures 6.2 and 6.1, i.e., the flow

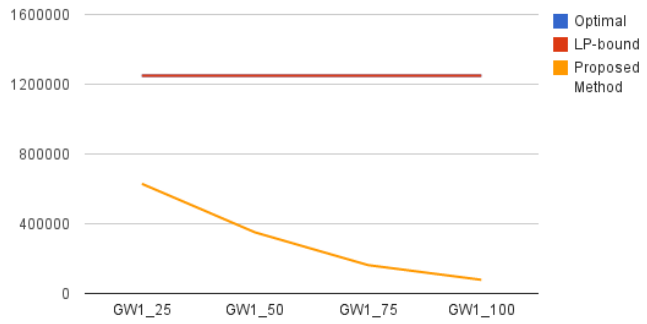

Figure 6.5: Performance trend of the fast methods available and proposed in comparison with the optimal flow and LP-bound, in the case of $\mathcal{GW}_1$ graph.

drops linearly when the minimum-lot constraints increases in a network and the proposed method can generate flow up to 100%.

Table 6.5: Instances, and its maximum-flow, and flow produced by the proposed fast method. Here, the optimal bound computation resulted in "out of memory" exception.

Instance	**maximum-flow**	**Proposed Method**
$\mathcal{GL}_2$-25	334,028	201,736
$\mathcal{GL}_2$-50	334,028	121950
$\mathcal{GL}_2$-75	334,028	70,366
$\mathcal{GL}_2$-100	334,028	35,296
$\mathcal{GW}_2$-25	6,785,136	3,532,798
$\mathcal{GW}_2$-50	6,785,136	1,670,138
$\mathcal{GW}_2$-75	6,785,136	893,033
$\mathcal{GW}_2$-100	6,785,136	0

The last test results, i.e., Table 6.5, 35, 296 units are generated as the flow for the graph $\mathcal{GL}_2$-100, and the performance trend continued (in Figure 6.6) showing the similar trend that we observed in the Figures 6.2, 6.1, 6.4, and 6.5. Due to the "out of memory" error, neither the optimum value nor the LP-bound were calculated using CPLEX. Instead, we used maximum-flow algorithms to find the bound. So were are not able to show the optimum value for this graph. However, an interesting point here is that the proposed method requires much lesser amount of system memory than the method meant for finding optimum value. The resulting zero flow of the graph $\mathcal{GW}_2$-100 raises an interesting question whether the proposed method fails to find the flow or the optimal flow itself is

zero for this graph. Hence, the result of $\mathcal{GW}_2$-100 is inconclusive in this regard.

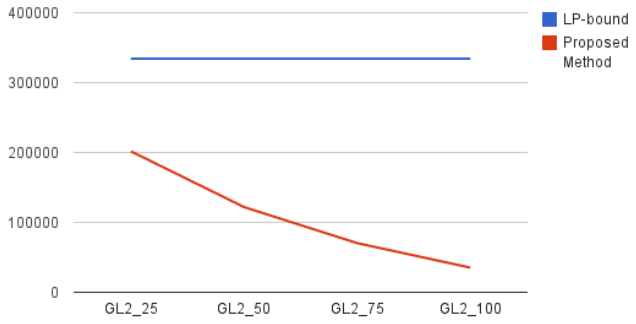

Figure 6.6: Performance trend of the fast methods available and proposed in comparison with the optimal flow and LP-bound, in the case of $\mathcal{GL}_2$ graph.

Chapter 7

Conclusion

It is observed that it is often impractical to send across amount of flow below a certain threshold, such as in pipeline networks for transporting natural gas. Some of the reasons identified are operations require lots to be large in order to be cost effective, products appear only in batches of a minimum size, and underlying mechanical and chemical processes require a minimum level of operation. Thus, either the quantity that can be shipped must be zero or it must be between the minimum threshold (lot sizes) and maximum capacity. In such instances, the problem of finding maximum-flow becomes much more difficult to solve, and the exact methods tend to be too time consuming for practical use. The level of difficultly is strongly NP-hard and it is proven in the literature. Since in practice it is not generally required to use the optimal solution, few fast (inexact) methods are suggested in the literature to produce near-optimum solutions. In this thesis, we have proposed a fast method by introducing minimum-lot constraints to Edmonds-Karp Algorithm, and tested it with six different graphs generated using RMFGEN-generator of Goldfarb and Grigoriadis. Eight instances were created from two graphs ($\mathcal{GL}$ and $\mathcal{GW}$) by differing the percentage of minimum-lot constraints, ranging from 30% to 100%. Similarly, four instances are created from the remaining six graphs ranging from 25% to 100%. Unfortunately, there was only one paper (Haugland et al's) in the literature that is close to the aim of this thesis work and had results that can be comparable with the input samples that we had for testing.

The results comparison show that the Haugland et al's methods could not generate a flow when the percentage of minimum-lot constraints are more than 60% in the case of $\mathcal{GL}$ graph and more than 70% in the case of $\mathcal{GW}$ graph. The method proposed in this thesis performs and could produce flow even in such cases, i.e., even when a network has 100% minimum-lot size constraints. For example, our method produced the flow of $283,816$ units in the graph $\mathcal{GL}$-60, whereas the Haugland et al. methods can produce only 0 as the possible flow. Promisingly, the proposed method functioned in all the graphs that are taken in this experimentation. When there were 100% minimum-lot size constraints, the method proposed produced the flow of $44,145$ units in the graph $\mathcal{GL}$-100,

and $8,474$ units in the graph $\mathcal{GW}$-100. It is observed that the proposed method linearly drops the maximum-flow when the percentages of minimum-lot sizes increases, whereas Haugland et al's methods drops suddenly when the percentage of minimum-lot sizes touches 40% ($\mathcal{GL}$ graph) to 50% ($\mathcal{GW}$ graph) and reaches zero after 60% ($\mathcal{GL}$ graph) to 70% ($\mathcal{GW}$ graph). However for 30% to 50% of constraints, Haugland et al's methods perform better than the method proposed in this thesis. It shows that Haugland et al's methods are better suitable when a network's minimum-lot constraints are up to 50% or 60%, and the proposed method is well suitable when the constraints percentage goes more beyond 50 to 60%.

Although the fast method proposed in this thesis can produce a flow even when the percentage of minimum-lot constraints is 100% in a graph, the difference between the flow produced by the proposed method and the optimum flow is considerable. Improving the proposed fast method to give results more close to the optimum value is a potential direction to go further and that could be considered as a future work.

Bibliography

[1] A. Sifaleras. Minimum cost network flows: Problems, algorithms, and software. *Operations Research*, 23(1):3–17, 2013.

[2] A. Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.

[3] D. Haugland, M. Eleyat, and M. L. Hetland. The maximum flow problem with minimum lot sizes. In *International Conference on Computational Logistics*, volume 6971, pages 170–82, Hamburg, Germany, 2011.

[4] J. E. Beasley. Modern heuristic techniques for combinatorial problems. chapter Lagrangian Relaxation, pages 243–303. John Wiley & Sons, Inc., 1993.

[5] M. Eleyat, D. Haugland, M. L. Hetland, and L. Natvig. Parallel algorithms for the maximum flow problem with minimum lot sizes. In *Operations Research Proceedings*, volume 6971, pages 183–88, Zurich, Switzerland, 2011. Springer.

[6] M. Eleyat. *Accelerating the Regina Network Flow Simulator on Multi-core Systems*. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2014.

[7] C. Thielen and S. Westphal. Complexity and approximability of the maximum flow problem with minimum quantities. *Networks: An International Journal*, 62(2):125–131, 2013.

[8] S. O. Krumke and C. Thielen. The generalized assignment problem with minimum quantities. *European Journal of Operations Research*, 228(1):46–55, 2013.

[9] M. Bender, C. Thielen, and S. Westphal. Mathematical foundations of computer science. volume 8087, chapter A Constant Factor Approximation for the Generalized Assignment Problem with Minimum Quantities and Unit Size Items, pages 135–145. Springer, 2013.

[10] C. Thielen. Using minimum quantities to guarantee resource efficiency in combinatorial optimization problems. In *International Conference on*

Resource Efficiency in Interorganizational Networks, pages 233–235, Gottingen, Germany, 2013.

[11] F. T. Nobibon, R. Leus, K. Nip, and Z. Wang. Resource loading with time windows. *European Journal of Operations Research*, 13(1):1–13, 2015 (In-Press).

[12] I. E. Grossmann. Mixed-integer programming approach for the synthesis of integrated process flowsheets. *Papers from 25th CONICET International Conference*, 9:463–482, 1985.

[13] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[14] D. Goldfarb and M. D. Grigoriadis. A computational comparison of the dinic and network simplex methods for maximum flow. *Annals of Operations Research*, 13(1):81–123, 1988.