Fast methods to solve the pooling problem

Anisha Kejriwal

A thesis submitted in partial fulfillment for the degree of Master of Science (MSc)

University of Bergen
Faculty of Mathematics and Natural Sciences
Department of Informatics

May 31, 2014
Supervisor
Dag Haugland

Acknowledgements

First of all, I am very thankful to God for a great opportunity to write this thesis and for giving me the strength and courage to complete this work.

I am extremely grateful to my supervisor, Professor Dag Haugland, for his excellent guidance. I have always received valuable suggestions and encouragement from him. His support, patience and tireless proofreading throughout this work is commendable.

I would like to acknowledge the staff in the Department of Informatics for their help. I am also thankful to the students of the optimization group with whom I shared an office for maintaining a good academic environment.

I am deeply grateful to my former colleagues at Tobii Technology Norge AS for their well wishes.

Finally, I would like to express my sincere gratitude to my parents, siblings and friends for their constant support and unconditional love.

Abstract

In pipeline transportation of natural gas, simple network flow problems are replaced by hard ones when bounds on the flow quality are imposed. The sources, typically represented by gas wells, provide flow of unequal compositions. For instance, some sources may be richer in undesired contaminants, such as CO_2, than others. At the terminals, constraints on the relative content of the contaminant may be imposed. Flow streams are blended at junction points in the network, where the relative CO_2-content becomes a weighted average of the relative CO_2-content in entering streams. To account for the quality bounds at the terminals, the quality therefore must be traced from the sources via junction points to the terminals.

The problem of allocating flow at minimum cost is referred to as the pooling problem when the above-mentioned quality bounds are imposed. It is known that the pooling problem is NP-hard, which means that it is very unlikely that exact solutions can be found in instances of large scale. Some exact methods, based on strong mathematical formulations and intended for instances of small and medium size, have recently been developed. However, the literature offers few approaches to approximation algorithms and other inexact methods dedicated for large pooling problem instances.

This thesis focuses on the development of inexact or heuristic techniques for the pooling problem. The aim of these techniques is to find good feasible solutions for large pooling problem instances at a reasonable computation cost, and the methods do not guarantee global optimality. In order to achieve this, three approaches are discussed in this thesis. First, we propose an improvement heuristic which iteratively reduces the total cost. Since the quality of the solutions provided by the improvement method depends upon good initial solutions, we propose construction heuristic methods that give good feasible solutions for the pooling problem. The methods construct a sequence of sub-graphs, each of which contains a single terminal, and an associated linear program for optimizing the flow to the terminal. The optimal solution to each linear program serves as a feasible augmentation of total flow accumulated so far. Finally, we combine both the above mentioned methods, such that the solution given by the construction heuristic is used as the starting solution by the improvement method. Computational experiments indicate that all the heuristic methods proposed in this thesis are faster compared to the heuristics that were proposed earlier.

Since the exact solutions are not known in large instances, the solutions given by the heuristic methods are compared to lower bounds on the optimal objective function value. In this thesis, we also propose a constraint generation algorithm, that aims to compute lower bounds on the minimum cost fast.

May 31, 2014 – Bergen, Norway,
Anisha Kejriwal

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

The pooling problem is an important global optimization problem. One application area is pipeline transportation of natural gas. Natural gas has various uses such as heating households, air conditioning and providing fuel for cooking as it is available at affordable prices. Since natural gas burns cleaner than gasoline or diesel, natural gas-powered cars, trucks and buses are being used to reduce emissions. Therefore, efficient and economical ways of transporting natural gas to the consumers is very important. Natural gas is transported from the sources to the terminals through a pipeline network. Since pipelines are safe, efficient and, they are unseen as most of them are buried, they are the primary means of transporting natural gas to the consumer markets. The sources, typically represented by gas wells, can supply natural gas having different components like carbon dioxide (CO_2), nitrogen (N_2), ethane (C_2H_6), propane (C_3H_8) etc in varying amounts. At the terminals, constraints on the relative content of the contaminant may be imposed for the natural gas to be of use. Since the network does not have any nodes for purifying the natural gas, the only way to control the level of contaminants is by means of blending operations either at the terminals or at intermediate nodes within the network called *pools*. At the pools, the relative content of a contaminant (eg CO_2) becomes a weighted average of the relative CO_2-content in the source streams. To account for the quality bounds at the terminals, the quality therefore must be traced from the sources via pools to the terminals. The optimization problem of allocating flow at minimum cost is referred to as the *pooling problem* when the above-mentioned quality bounds are imposed.

Apart from pipeline transportation of natural gas, the pooling problem can also occur in petroleum refining and waste-water treatment. For a detailed review of industrial applications, the reader can refer to the works by Visweswaran [35] and Kallrath [21].

1.1 Background and motivation

The pooling problem is an extension of the *minimum-cost network flow problem*. The minimum-cost network flow problem consists of a flow network, that is, a directed graph $D = (N, A)$, where N is the set of nodes and A is the set of arcs. b_i denotes the net supply (arc flow out - arc flow in) at each node $i \in N$. The value of b_i is determined by the nature of node i. In particular, b_i is positive for a supply node, negative for a demand node and zero for intermediate nodes. Each arc $(i, j) \in A$ has a flow capacity u_{ij} and a unit cost c_{ij} associated with it. The flow may be routed through intermediate nodes that reflect warehouses or distribution centers instead of being sent directly from sources to terminals. The objective is to send a homogeneous flow at a minimum cost from a set of sources to a set of terminals that satisfies the demand at the terminals. The arc flows f_{ij} for each $(i, j) \in A$ must be non-negative and must be no greater than the arc capacities. Flow conservation must also be satisfied at all the intermediate nodes. An example of this problem is illustrated in Figure 1.1.

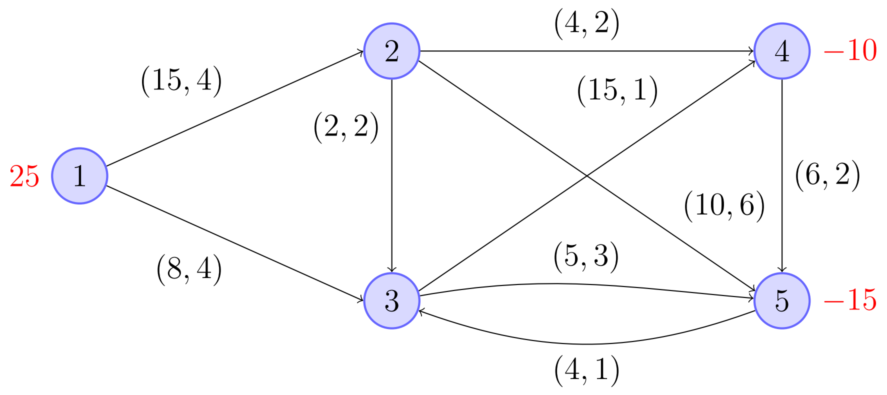

Figure 1.1: Minimum-cost network flow problem.

In Figure 1.1, node 1 is a *supply node* supplying 25 flow units, and nodes 4 and 5 are *demand nodes* requiring 10 and 15 flow units, respectively, as indicated by the negative signs. The remaining nodes have no net supply or demand and are called the *intermediate nodes*. The numbers next to the arcs specify the arc capacity and the cost of shipping 1 unit along the arc, respectively. Hence, the flow on arc 1-2 must be between 0 and 15 flow units, and each unit of flow on this arc costs 4 units. It is well known that this problem can be formulated as a linear programming problem and can hence be solved by means of efficient algorithms.

The minimum-cost network flow problem needs to be extended when the flow coming from different sources have different qualities. The *blending problem* which typically occurs in petroleum refining has only two sets of nodes: sources and terminals. The quality of the flow at different sources varies. The flow from the sources is blended at the terminals in such a way that the quality bounds at the terminals are respected. This problem can also be formulated as a linear program and can hence be solved fast. The objective in the blending problem is to minimize the total cost of producing demand at the terminals. An example of the blending problem is illustrated in Figure 1.2.

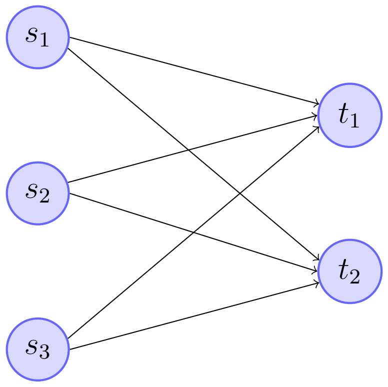

Figure 1.2: A sample blending problem.

The pooling problem combines features of both the minimum-cost network flow problem and the blending problem. The pooling problem network is tripartite and consists of sources, intermediate nodes called pools and terminals. The sources supply raw materials (eg natural gas), having different qualities. The natural gas is first blended in the pools or sent directly to the terminals in such a way that the demand at the terminals are met and the total cost of transportation is minimized. In addition, like in the blending problem, the flow must be assigned such that the resulting quality at the terminals is below the given bounds. The presence of pools introduces non-linearities and non-convexities in the model, thus making the problem difficult. In [20], Haverly defines the pooling problem by means of an instance that is studied quite frequently, referred to as Haverly1 [20]. We illustrate this instance in Figure 1.3.

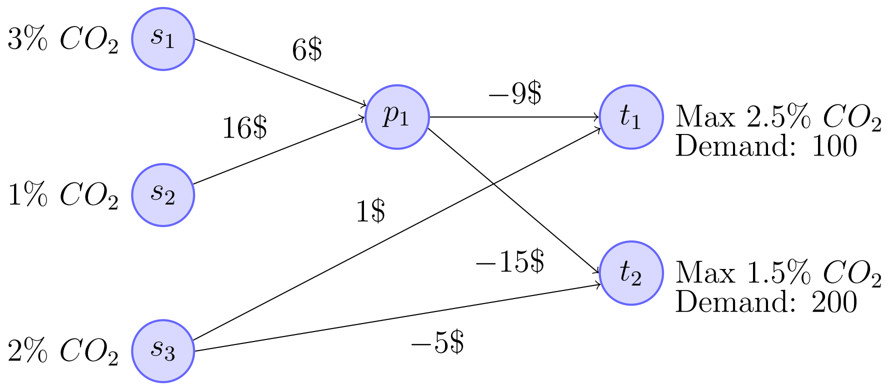

Figure 1.3: The Haverly1 pooling problem instance.

In Figure 1.3, there are three sources s_1, s_2 and s_3 that supply natural gas with different levels of contaminant CO_2. The amount of CO_2 contamination at the sources s_1, s_2 and s_3 are given as 3%, 1% and 2%, respectively. The natural gas from sources s_1 and s_2 is blended in the pool p_1 and then sent to the terminals t_1 and t_2. The natural gas from source s_3 is sent directly to the terminals t_1 and t_2. The demand at the terminals t_1 and t_2 is 100 and 200 units of natural gas, respectively, and they will pay for the natural gas only if its CO_2 content does not exceed 2.5% and 1.5%, respectively. In many pooling problem instances, for example in Haverly1 in [20], the price of the natural gas and their selling prices are specific to source and terminal nodes, respectively. These can be translated into arc costs in the network as follows: Since one unit of natural gas at sources s_1 and s_2 costs 6$ and 16$, respectively, hence, the costs of transporting one unit of natural gas from s_1 to p_1 and from s_2 to p_1 are 6$ and 16$, respectively. Terminals t_1 and t_2 pay 9$ and 15$, respectively, for one unit of natural gas. Hence, the costs of transporting one unit of natural gas along arcs (p_1, t_1) and (p_1, t_2) are -9$ and -15$, respectively. Similarly, the cost of one unit of natural gas at source s_3 is 10$. Hence, the transportation costs per unit of natural gas from $s_3 - t_1$ and $s_3 - t_2$ are 1$ (10-9) and -5$ (10-15), respectively, as shown in Figure 1.3.

The pooling problem similar to Haverly1 stated above is referred to as the *standard pooling problem* where the only links in the network are from sources to pools, pools to terminals and sources to terminals. A more complex pooling problem was introduced by Audet et al. [7] by allowing links between pools as well, referred to as the *generalized pooling problem*, an example of which is given in Figure 1.4. In this thesis, we focus only on the standard pooling problem.

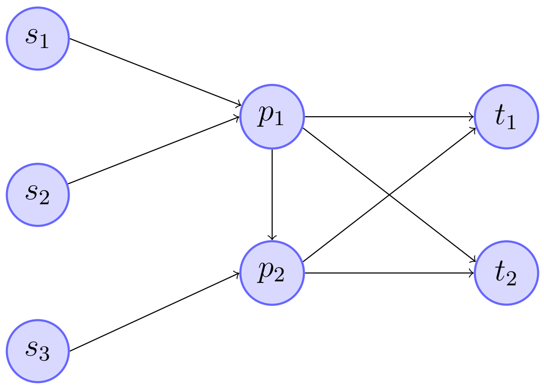

Figure 1.4: A generalized pooling problem instance.

It is known that the pooling problem is NP-hard [2], which means that it is very unlikely that exact solutions can be found in instances of large scale. Some exact methods, based on strong mathematical formulations and intended for instances of small and medium size, have recently been developed [2]. However, the literature offers few approaches to approximation algorithms and other inexact methods dedicated for large pooling problem instances. Therefore, it is important to develop and implement fast methods for the pooling problem.

1.2 Previous work (literature review)

The pooling problem has been extensively studied over the last four decades. The literature offers different solution techniques that have been proposed, and can be classified into local and global optimization techniques.

1.2.1 Local optimization techniques

The early approaches aimed to find a good local optimum. One of the approaches was proposed by Haverly [20]. It is a recursive approach in which the values of pool qualities are estimated and fixed. Then the flow is optimized by solving the resulting linear program (LP), and the new qualities are calculated by using the flow values from the solution of the LP. The method stops if the estimated values of pool qualities coincide with their new values, otherwise a new LP is constructed using the new values of the pool qualities. It was observed that the solution obtained by this recursive method depends on the initially estimated values of the pool qualities and hence it may or may not converge to a local optimum. The method was also known to be unstable and it took much computational time in large instances [24]. Another recursive approach for the pooling problem was proposed by Audet et al. [7] which was referred to as Alternate heuristic (ALT). A variable neighborhood search (VNS) heuristic based upon the ALT heuristic was also proposed by Audet et al. [7].

Successive linear programming (SLP) is another approach that has been used to solve the pooling problem in the petrochemical industries. In this method, the bi-linear terms are approximated using the first order Taylor's expansion and the resulting LP is solved. The solution from the LP is used as the new base of the Taylor's expansion, and the process is repeated until it converges to a fixed point. SLP was introduced by the name mathematical approximation program (MAP) by Griffith and Stewart [18] who tested the approach on petroleum refinery optimization in Shell oil company. SLP and the generalized reduced gradient (GRG) algorithms have been used by Lasdon et al. [22] to solve the pooling problem, and the results have shown that they have some advantages over Haverly's recursive approach. For more information about these algorithms, the reader can refer to Griffith and Stewart [18], Palacios-Gomez et al. [27], Baker and Lasdon [8] and Greenberg [17].

In [3], Alfaki and Haugland proposed a construction heuristic for the pooling problem. This method constructs a sequence of sub-graphs, each consisting of a single terminal, and a bi-linear program associated with it. The bi-linear program was solved for optimizing the flow to the terminal, and the corresponding flow augmentation was made for each sub-problem. This method was designed to give good feasible solutions, especially in large-scale instances, and it did not guarantee to find the optimal solution. Experimental results showed that this heuristic outperformed multi-start local optimization techniques provided by commercially available software in large-scale instances.

Another approach to solve the pooling problem is decomposition. The idea of this method is to decompose the problem into two linear sub-problems by fixing a variable in the bi-linear terms. These sub-problems are solved for their respective global optimums at each iteration, and the process continues until the stopping criteria is satisfied. A well-known decomposition method was proposed by Benders [9] for solving nonlinear optimization problems. A generalization of Benders decomposition was proposed by Geoffrion [15]. Floudas and Aggarwal [12] proposed a method that searches for a global solution based on the generalized Benders decomposition. It was observed that this method could not guarantee convergence to a global solution.

The literature mentions some discretization approaches for the pooling problem. In one of these approaches used by Tomasgard et al. [33] and Rømo et al. [30], the quality variables were discretized to approximate the bi-linear constraints. This resulted in the approximation of the pooling problem by a mixed integer linear programming (MILP) problem. A similar approach was used by Faria and Bagajewicz [11], Pham [28] and Pham et al. [29]. A generalization of the discretization approach by Pham et al. [29], was proposed by Alfaki and Haugland [4]. In this method, the bi-linear terms were linearized by discretizing the domain of the proportion variables into a fixed number of points. Computational experiments performed in [4], showed that this method outperformed traditional solution methods where continuous models were used. Recently, different discretization methods have been proposed to approximate a bi-linear program as a MILP by Gupte et al. [19]. These ideas were tested on random instances of the pooling problem. The experiments suggested that discretization is a promising approach especially for large-scale and generalized pooling problems.

An analysis on the sensitivity of local optimal solutions with respect to problem parameters was performed by Greenberg [17] and Frimannslund et al. [14].

1.2.2 Global optimization techniques

During the last two decades, many global optimization techniques have been proposed for the pooling problem. An approach based on duality theory and Lagrangian relaxation was first proposed by Floudas and Aggarwal [12] and Visweswaran and Floudas [34]. The method was called Global Optimization Algorithm (GOP) which guaranteed convergence to a global solution. Another duality related approach was proposed by Ben-Tal et al. [10]. Some Lagrangian-based approaches for the pooling problem can be found in [1] and [6]. McCormick [25] used convex and concave envelopes to construct a relaxation which was applied in a branch-and-bound algorithm. This approach was first applied to the pooling problem by Foulds et al. [13]. Recently, different relaxation techniques have been integrated into the branch-and-bound algorithm. Some of these relaxation techniques include the reduced reformulation linearization technique by Liberti and Pantelides [23] and the piecewise linear relaxation technique by Gounaris [16].

Global optimization algorithms are quite effective for instances of modest size. In larger instances, however, global optimizers fail to converge in reasonable time, while existing local optimizers depend largely on good initial guesses [3]. For recent research on the pooling problem, the reader can refer to the works by Alfaki [5] and Misener [26].

This research aims to develop fast methods to obtain optimal or near-optimal solutions for the pooling problem. We propose improvement and construction heuristic methods to achieve this. Our goal is to reduce the computation time needed by these methods, especially for large-scale instances. We follow the approach of linearizing the bi-linear problem by fixing the values of the variables that participate in the bi-linear terms. The next chapters of this thesis reviews some of the formulations for the pooling problem existing in the literature, and then we describe various approaches we use in order to get good feasible solutions.

1.3 Problem definition

We are given a directed graph $D = (S, P, T, A)$, where the node set $N = (S, P, T)$ consists of the sources S, pools P and terminals T, and the arc set $A \subseteq (S \times P) \cup (P \times T) \cup (S \times T)$ links sources with pools, pools with terminals and sources with terminals. We let K be a finite set, and refer to its elements as quality attributes. Define the vectors of unit cost $c \in \Re^A$, and let b_i denote the flow capacity of node $i \in N$. For all $k \in K$, we are given an input quality q_s^k corresponding to attribute $k \in K$ at each source $s \in S$, and a quality bound q_t^k corresponding to attribute $k \in K$ at each terminal $t \in T$. For each pool $p \in P$, we denote the sets of neighbor sources and terminals by $S_p = \{s \in S : (s, p) \in A\}$ and $T_p = \{t \in T : (p, t) \in A\}$, respectively. For each source $s \in S$ and each terminal $t \in T$, respectively, we denote the sets of neighbor pools by $P_s = \{p \in P : (s, i) \in A\}$, and $P_t = \{p \in P : (i, t) \in A\}$. We also define $S_t = \{s \in S : (s, t) \in A\}$ and $T_s = \{t \in T : (s, t) \in A\}$. The set of arcs incident to some source is denoted $A_S = A \cap (S \times (P \cup T))$, and the set of arcs incident to some terminal is denoted $A_T = A \cap ((S \cup P) \times T)$.

Let f_{ij} denote the flow on arc $(i, j) \in A$. Then the *pool qualities* are defined as $w_p^k = \frac{\sum_{s \in S_p} q_s^k f_{sp}}{\sum_{s \in S_p} f_{sp}}$ $(p \in P,\ k \in K)$. That is, the quality at pool p is a weighted average of the input qualities at the sources from which the pool receives flow, and the weights are identical to the flow values. Analogously, the *terminal qualities* are given as $w_t^k = \frac{\sum_{s \in S_t} q_s^k f_{st} + \sum_{p \in P_t} w_p^k f_{pt}}{\sum_{s \in S_t} f_{st} + \sum_{p \in P_t} f_{pt}}$.

1.4 Structure of the thesis

This thesis is composed of six chapters, including this introductory chapter. The remaining chapters are organized as follows: Chapter 2 describes various model formulations for the pooling problem. Chapter 3 explains the heuristic methods, mainly the improvement and the construction heuristics, which aim to find good feasible solutions for large pooling problem instances. Chapter 4 explains a constraint generation algorithm for fast computation of lower bounds on the minimum cost. Chapter 5 presents the results of various numerical experiments performed on the methods described in Chapters 3 and 4. Finally, Chapter 6 gives the conclusion and possible directions for future work.

Chapter 2

Model formulations for the pooling problem

The pooling problem can be formulated in different ways. These formulations can lie into two major categories. The first one takes into account flow and quality variables, whereas the second uses flow proportions instead of quality variables. These formulations are discussed more in the following sections.

2.1 The quality formulation

Combining the flow and quality variables in a model, and utilizing $\sum_{s\in S_p} f_{sp} = \sum_{t\in T_p} f_{pt}$ $(p \in P)$, leads to the so-called P-formulation of the problem which was introduced by Haverly [20]:

$$\min_{f,w} \sum_{(i,j)\in A} c_{ij} f_{ij} \tag{2.1}$$

$$\text{s.t.} \sum_{p\in P_s} f_{sp} + \sum_{t\in T_s} f_{st} \le b_s \qquad s \in S \tag{2.2}$$

$$\sum_{s\in S_p} f_{sp} \le b_p \qquad p \in P \tag{2.3}$$

$$\sum_{s\in S_t} f_{st} + \sum_{p\in P_t} f_{pt} \le b_t \qquad t \in t \tag{2.4}$$

$$\sum_{s\in S_p} f_{sp} - \sum_{t\in T_p} f_{pt} = 0 \qquad p \in P \tag{2.5}$$

$$\sum_{s\in S_p} q_s^k f_{sp} - w_p^k \sum_{t\in T_p} f_{pt} = 0 \qquad p \in P, k \in K \tag{2.6}$$

$$\sum_{s\in S_t} q_s^k f_{st} + \sum_{p\in P_t} w_p^k f_{pt} - q_t^k \left(\sum_{s\in S_t} f_{st} + \sum_{p\in P_t} f_{pt} \right) \le 0 \qquad t \in T, k \in K \tag{2.7}$$

$$f_{ij} \ge 0 \qquad (i,j) \in A \tag{2.8}$$

Constraints (2.2)–(2.4) reflect the flow capacity bound at all sources, pools and the terminals. Constraint (2.5) expresses conservation of flow around the pools. Constraint (2.6) results from the definition of pool qualities given in Section 1.3. Constraint (2.7) results from the definition of terminal qualities given in Section 1.3 and the quality bound constraint, $w_t^k \le q_t^k$ for all $k \in K$. Constraint (2.8) reflects that the flow in the network must be non-negative. We note that (2.1)–(2.8) is not a linear program, as constraints (2.6)–(2.7) contain products of variables.

2.2 Proportion formulations

The pooling problem can be reformulated so that the quality variables can be replaced by variables that represent proportions of flow. Two types of proportion variables can be used, namely source and terminal proportions. On that basis, the formulations can further be divided into two parts:

2.2.1 Formulations with source proportions

2.2.1.1 Q-formulation

We introduce *source proportions* as variables associated with each pair consisting of a pool and a source: For each $p \in P$ and $s \in S_p$, let y_p^s denote the proportion of the flow entering pool p that originates from source s, i.e. $y_p^s = \frac{f_{sp}}{\sum_{t \in T_p} f_{pt}}$ if the flow through p is non-zero. Then the flow variable f_{sp} can be replaced by $y_p^s \sum_{t \in T_p} f_{pt}$, and the pool quality w_p^k can be replaced by $\sum_{s \in S_p} q_s^k y_p^s$. We arrive at the Q-formulation which was first proposed by Ben-Tal et al. [10]:

$$\begin{aligned}
\min_{f,y} \; & \sum_{s \in S} \sum_{t \in T_s} c_{st} f_{st} + \sum_{s \in S} \sum_{p \in P_s} c_{sp} \sum_{t \in T_p} y_p^s f_{pt} \\
& + \sum_{p \in P} \sum_{t \in T_p} c_{pt} f_{pt} \\
\text{s.t.} \; & \sum_{p \in P_s} \sum_{t \in T_p} y_p^s f_{pt} + \sum_{t \in T_s} f_{st} \le b_s && s \in S && (2.9) \\
& \sum_{t \in T_p} f_{pt} \le b_p && p \in P && (2.10) \\
& \sum_{s \in S_t} f_{st} + \sum_{p \in P_t} f_{pt} \le b_t && t \in t && \\
& \sum_{s \in S_t} q_s^k f_{st} + \sum_{p \in P_t} \sum_{s \in S_p} q_s^k y_p^s f_{pt} && && \\
& - q_t^k \left(\sum_{s \in S_t} f_{st} + \sum_{p \in P_t} f_{pt} \right) \le 0 && t \in T, k \in K && (2.11) \\
& \sum_{s \in S_p} y_p^s = 1 && p \in P && (2.12) \\
& 0 \le y_p^s \le 1 && p \in P, s \in S_p && (2.13) \\
& f_{st} \ge 0 && s \in S, t \in T_s && (2.14) \\
& f_{pt} \ge 0 && p \in P, t \in T_p && (2.15)
\end{aligned}$$

We arrive at constraint (2.9) by replacing the value of f_{sp} in constraint (2.2), at (2.10) by utilizing the flow conservation in constraint (2.3) and at

(2.11) by replacing the value of w_p^k in (2.7). The source proportions must lie between 0 and 1 and they must sum to 1 which is reflected by constraints (2.13) and (2.12), respectively. In this formulation, the flow variables correspond to the arcs that are entering terminals and they must be non-negative as shown by the constraints (2.14)-(2.15).

2.2.1.2 PQ-formulation

The Q-formulation can be extended to the PQ-formulation, which was introduced by Tawarmalani and Sahinidis [32] by adding some constraints to it. The new constraints are:

$$\sum_{t \in T_p} y_p^s f_{pt} - b_p y_p^s \leq 0 \qquad p \in P, s \in S_p \tag{2.16}$$

$$f_{pt} = \sum_{s \in S_p} y_p^s f_{pt} \qquad p \in P, t \in T_p \tag{2.17}$$

They are formed by applying the reformulation linearization technique (RLT) [31] to constraints (2.10) and (2.12). Hence, constraint (2.16) is obtained by multiplying (2.10) by y_p^s, and (2.17) by multiplying (2.12) by f_{pt}.

Sometimes, it is convenient to introduce the variable $x_{spt} = y_p^s f_{pt}$, denoting the flow along path (s, p, t), where $(s, p), (p, t) \in A$. Then, we also have $f_{pt} = \sum_{s \in S_p} x_{spt}$, obtained by replacing $y_p^s f_{pt}$ by x_{spt} in constraint (2.17), and the PQ-formulation becomes:

$$
\begin{aligned}
\min_{f,y,x} \; & \sum_{s\in S}\sum_{t\in T_s} c_{st} f_{st} + \sum_{s\in S}\sum_{p\in P_s} c_{sp} \sum_{t\in T_p} x_{spt} \\
& + \sum_{p\in P}\sum_{t\in T_p} c_{pt} f_{pt} \\
\text{s.t.} \; & \sum_{p\in P_s}\sum_{t\in T_p} x_{spt} + \sum_{t\in T_s} f_{st} \le b_s && s \in S && (2.18) \\
& \sum_{t\in T_p} f_{pt} \le b_p && p \in P \\
& \sum_{s\in S_t} f_{st} + \sum_{p\in P_t} f_{pt} \le b_t && t \in t \\
& \sum_{s\in S_t} q_s^k f_{st} + \sum_{p\in P_t}\sum_{s\in S_p} q_s^k x_{spt} \\
& - q_t^k \left(\sum_{s\in S_t} f_{st} + \sum_{p\in P_t} f_{pt} \right) \le 0 && t \in T, k \in K && (2.19) \\
& \sum_{t\in T_p} x_{spt} - b_p y_p^s \le 0 && p \in P, s \in S_p && (2.20) \\
& f_{pt} = \sum_{s\in S_p} x_{spt} && p \in P, t \in T_p && (2.21) \\
& x_{spt} = y_p^s f_{pt} && s \in S, p \in P_s, t \in T_p && (2.22) \\
& \sum_{s\in S_p} y_p^s = 1 && p \in P \\
& 0 \le y_p^s \le 1 && p \in P, s \in S_p \\
& f_{st} \ge 0 && s \in S, t \in T_s \\
& f_{pt} \ge 0 && p \in P, t \in T_p
\end{aligned}
$$

Constraints (2.18), (2.19) and (2.20) result from replacing the bi-linear term $y_p^s f_{pt}$ by x_{spt} in constraints (2.9), (2.11) and (2.16), respectively. Constraint (2.21) ensures that the flow on the arc (p, t) equals the total flow on paths intersecting the arc. Constraint (2.22) reflects that the flow along path (s, p, t) equals the flow on arc (p, t) times the proportion of flow at pool p coming from source s.

2.2.2 Formulation with terminal proportions

A sibling formulation to the PQ-formulation was developed in [2] by introducing the terminal proportions y_p^t $(p \in P,\ t \in T_p)$. For each pool $p \in P$, y_p^t is defined as the proportion of flow at p that goes to the terminal $t \in T$, i.e. $y_p^t = \frac{f_{pt}}{\sum_{s \in S_p} f_{sp}}$ if the flow through p is non-zero. The TP-formulation reads:

$$\begin{aligned}
\min_{f,y,x} & \sum_{s \in S} \sum_{t \in T_s} c_{st} f_{st} + \sum_{s \in S} \sum_{p \in P_s} c_{sp} f_{sp} \\
& + \sum_{p \in P} \sum_{t \in T_p} c_{pt} \sum_{s \in S_p} x_{spt} \\
\text{s.t.} & \sum_{p \in P_s} f_{sp} + \sum_{t \in T_s} f_{st} \le b_s && s \in S \\
& \sum_{s \in S_p} f_{sp} \le b_p && p \in P \\
& \sum_{s \in S_t} f_{st} + \sum_{p \in P_t} \sum_{s \in S_p} x_{spt} \le b_t && t \in t \\
& \sum_{s \in S_t} q_s^k f_{st} + \sum_{p \in P_t} \sum_{s \in S_p} q_s^k x_{spt} \\
& - q_t^k \left(\sum_{s \in S_t} f_{st} + \sum_{p \in P_t} \sum_{s \in S_p} x_{spt} \right) \le 0 && t \in T, k \in K \\
& \sum_{s \in S_p} x_{spt} - b_p y_p^t \le 0 && p \in P, t \in T_p && (2.23) \\
& f_{sp} = \sum_{t \in T_p} x_{spt} && p \in P, s \in S_p && (2.24) \\
& x_{spt} = y_p^t f_{sp} && s \in S, p \in P_s, t \in T_p && (2.25) \\
& \sum_{t \in T_p} y_p^t = 1 && p \in P && (2.26) \\
& 0 \le y_p^t \le 1 && p \in P, t \in T_p && (2.27) \\
& f_{sp} \ge 0 && s \in S, p \in P_s && (2.28) \\
& f_{st} \ge 0 && s \in S, t \in T_s && (2.29)
\end{aligned}$$

The constraints can be interpreted as TP-variants of the constraints in the PQ-formulation. Hence, the flow variable f_{pt} equals $y_p^t \sum_{s \in S_p} f_{sp}$, and the path flow $x_{spt} = y_p^t f_{sp}$ as shown by the constraint (2.25). Then, we also have $f_{sp} = \sum_{t \in T_p} x_{spt}$ as given by the constraint (2.24). The terminal proportions must lie between 0 and 1 and they must sum to 1 which is reflected by constraints (2.27) and (2.26), respectively. In this formulation, the flow variables correspond to the arcs that are leaving sources and they must be non-negative as shown by the constraints (2.28)-(2.29).

2.2.3 Formulation with both source and terminal proportions

In [2], Alfaki and Haugland derived the STP-formulation by combining both the source and terminal proportions in one model. Therefore, all the variables and the constraints from both the PQ and TP-formulations are merged. The linear relaxations of the PQ, TP and STP-formulations defined in [2] are referred to as PQ, TP and STP-relaxations, respectively. The lower bound on the optimal objective function value provided by the STP-relaxation, is at least as tight as those provided by the PQ and TP-relaxations. Experiments performed in [2], show that, the STP-relaxation gave a bound identical to the best of those given by the PQ and TP-relaxations in all the small-scale instances, whereas, it gives tighter bounds than its two competitors in most of the large-scale instances. The STP-formulation can be stated as follows:

$$\min_{f,y,x} \sum_{(i,j)\in A} c_{ij} f_{ij} \tag{2.30}$$

$$\text{s.t.} \sum_{p\in P_s} f_{sp} + \sum_{t\in T_s} f_{st} \le b_s \qquad s \in S \tag{2.31}$$

$$\sum_{s\in S_p} f_{sp} \le b_p \qquad p \in P \tag{2.32}$$

$$\sum_{s\in S_t} f_{st} + \sum_{p\in P_t} f_{pt} \le b_t \qquad t \in t \tag{2.33}$$

$$\sum_{s\in S_t} q_s^k f_{st} + \sum_{p\in P_t} \sum_{s\in S_p} q_s^k x_{spt} - q_t^k \left(\sum_{s\in S_t} f_{st} + \sum_{p\in P_t} f_{pt} \right) \le 0 \qquad t \in T, k \in K \tag{2.34}$$

$$f_{sp} = \sum_{t\in T_p} x_{spt} \qquad p \in P, s \in S_p \tag{2.35}$$

$$f_{pt} = \sum_{s\in S_p} x_{spt} \qquad p \in P, t \in T_p \tag{2.36}$$

$$f_{ij} \ge 0 \qquad (i,j) \in A \tag{2.37}$$

$$x_{spt} = y_p^s f_{pt} \qquad s \in S, p \in P_s, t \in T_p \tag{2.38}$$

$$x_{spt} = y_p^t f_{sp} \qquad s \in S, p \in P_s, t \in T_p \tag{2.39}$$

$$\sum_{s\in S_p} y_p^s = 1 \qquad p \in P \tag{2.40}$$

$$\sum_{t\in T_p} y_p^t = 1 \qquad p \in P \tag{2.41}$$

$$\sum_{t\in T_p} x_{spt} - b_p y_p^s \le 0 \qquad p \in P, s \in S_p \tag{2.42}$$

$$\sum_{s\in S_p} x_{spt} - b_p y_p^t \le 0 \qquad p \in P, t \in T_p \tag{2.43}$$

$$0 \le y_p^s, y_p^t \le 1 \qquad p \in P, s \in S_p, t \in T_p \tag{2.44}$$

Chapter 3

Heuristic methods

Since the pooling problem is NP-hard [2], it is essential to develop fast heuristic methods that in many instances can produce optimal or near-optimal solutions.

3.1 Improvement heuristic

A popular idea is to utilize the bi-linear structure of the problem, which means that we get a linear program if e.g. the y-variables are fixed. We consider the problem where all constraints involving y-variables are removed from the STP-formulation, i.e. the problem defined by (2.30)–(2.37). We refer to the reduced problem, which we observe is a linear program, as the *independent flow relaxation*. It corresponds to neglect of the fact that flow streams are blended at the pools. Another interpretation is that we allow that, for instance, 80% of the flow from source s_1 to pool p is sent to terminal t, whereas, for instance, 30% of the flow from source s_2 to pool p is sent to terminal t. In practice, these proportions have to be identical, which is ensured by constraints (2.38)–(2.44).

Once we have solved the independent flow relaxation (let $(\hat{f}, \hat{x})$ denote an optimal solution to it), we make an estimate of either all the source proportions or all the terminal proportions. For the sake of reasoning, we choose the source proportions. The proportion of the flow through pool p entering from source s, is naturally estimated as $\hat{y}_p^s = \frac{\hat{f}_{sp}}{\sum_{t \in T_p} \hat{f}_{pt}}$. Again, since (2.38) is not imposed, we do not necessarily have that $\hat{y}_p^s = \frac{\hat{x}_{spt}}{\hat{f}_{pt}}$ for $t \in T_p$.

Next, we solve the independent flow relaxation with the additional constraints $x_{spt} - \hat{y}^s_p f_{pt} = 0$ for all $s \in S$, $p \in P_s$, and $t \in T_p$. This is identical to (2.38) with *fixed* source proportions ($\hat{y}^s_p$ is a constant, not a variable), and the constraints become linear. Based on the new optimal solution $(\hat{f}, \hat{x})$, we estimate the terminal proportions $\hat{y}^t_p = \frac{\hat{f}_{pt}}{\sum_{s \in S_p} \hat{f}_{sp}}$, and we solve the independent flow relaxation with side constraints $x_{spt} - \hat{y}^t_p f_{sp} = 0$ for all $s \in S$, $p \in P_s$, and $t \in T_p$. As expressed in Algorithm 1, the procedure continues by alternations between fixed source and terminal proportions until no significant flow changes are observed.

Algorithm 1 Alternating proportions.

```
 1: Let $\left(\hat{f}, \hat{x}\right)$ be an optimal solution to (2.30)–(2.37)
 2: Let mode equal either $S$ or $T$
 3: **repeat**
 4:     **if** mode=$S$ **then**
 5:         **for** $p \in P$, $s \in S_p$ **do**
 6:             $\hat{y}^s_p \leftarrow \frac{\hat{f}_{sp}}{\sum_{t \in T_p} \hat{f}_{pt}}$
 7:         **end for**
 8:         $\left(\hat{f}, \hat{x}\right) \leftarrow$ optimal solution to (2.30)–(2.37) with additional constraints
 9:             $x_{spt} - \hat{y}^s_p f_{pt} = 0$ for all $s \in S$, $p \in P_s$, and $t \in T_p$.
10:         mode $\leftarrow T$
11:     **else**
12:         **for** $p \in P$, $t \in T_p$ **do**
13:             $\hat{y}^t_p \leftarrow \frac{\hat{f}_{pt}}{\sum_{s \in S_p} \hat{f}_{sp}}$
14:         **end for**
15:         $\left(\hat{f}, \hat{x}\right) \leftarrow$ optimal solution to (2.30)–(2.37) with additional constraints
16:             $x_{spt} - \hat{y}^t_p f_{sp} = 0$ for all $s \in S$, $p \in P_s$, and $t \in T_p$.
17:         mode $\leftarrow S$
18:     **end if**
19: **until** no change in $\hat{f}$
```

3.2 Basic construction heuristic

The quality of the solutions provided by the improvement heuristic described in the previous section depends upon good initial solutions. Therefore, we develop a *basic construction heuristic* for the pooling problem in this section. This method is designed to give good feasible solutions. The idea of this method is to construct a flow by sending to only one terminal at a time and accumulating the corresponding flow. The selection of terminal is based on a ranking system. Since a linear program (LP) is used to do this, it makes the method fast.

The pooling problem with a single terminal denoted by $t_n \in T$ where $n \in \{1, 2...|T|\}$, can be expressed as a LP given by:

$$\min_f \sum_{(i,j)\in A} c_{ij} f_{ij} \tag{3.1}$$

$$\text{s.t.} \sum_{p\in P_s} f_{sp} + f_{st_n} \leq b_s - \sum_{i\in P_s\cup T_s} F_{si} \qquad s \in S \tag{3.2}$$

$$\sum_{s\in S_p} f_{sp} \leq b_p \cdot open_p \qquad p \in P_{t_n} \tag{3.3}$$

$$\sum_{s\in S_{t_n}} f_{st_n} + \sum_{p\in P_{t_n}} f_{pt_n} \leq b_{t_n} \tag{3.4}$$

$$\sum_{s\in S_p} f_{sp} - f_{pt_n} = 0 \qquad p \in P_{t_n} \tag{3.5}$$

$$\sum_{s\in S_{t_n}} (q_s^k - q_{t_n}^k) f_{st_n} + \sum_{p\in P_{t_n}} \sum_{s\in S_p} (q_s^k - q_{t_n}^k) f_{sp} \leq 0 \qquad k \in K \tag{3.6}$$

$$f_{ij} \geq 0 \qquad (i,j) \in A \tag{3.7}$$

It is linear as we have eliminated all the terminals except one. We define $open_p$ for $p \in P_{t_n}$ which equals 1 if pool p is open and 0 otherwise. We set $open_p$ to 1 initially. In the above LP, constraints (3.2)–(3.4) reflect the flow capacity bound at all sources, pools and the terminal t_n, respectively. Constraint (3.5) is for conservation of flow at the pools. Constraint (3.7) reflects that the flow in the network must be non-negative. Constraint (3.6) reflects the quality bounds at the terminal t_n, and can be explained as follows:

Let us assume that the sources supply natural gas with only one contaminant which is CO_2. Then, q_s^k denotes the relative CO_2 content that enters the network at the sources. The natural gas containing CO_2 flows to the terminal t_n only. The total flow at t_n is written as: $\sum_{s\in S_{t_n}} f_{st_n} + \sum_{p\in P_{t_n}} f_{pt_n}$ which equals $\sum_{s\in S_{t_n}} f_{st_n} + \sum_{p\in P_{t_n}} \sum_{s\in S_p} f_{sp}$. The total amount of CO_2 sent to t_n is $\sum_{s\in S_{t_n}} q^k_s f_{st_n} + \sum_{p\in P_{t_n}} \sum_{s\in S_p} q^k_s f_{sp}$. The relative CO_2 content at t_n, $\frac{\sum_{s\in S_{t_n}} q^k_s f_{st_n}+\sum_{p\in P_{t_n}} \sum_{s\in S_p} q^k_s f_{sp}}{\sum_{s\in S_{t_n}} f_{st_n}+\sum_{p\in P_{t_n}} \sum_{s\in S_p} f_{sp}}$, must be below the quality bound at that terminal, $q^k_{t_n}$.

Multiplying $q^k_{t_n}$ with the denominator of the fraction stated above and moving it to the the left hand side of the inequality, we get:

$\sum_{s\in S_{t_n}} q^k_s f_{st_n}+\sum_{p\in P_{t_n}} \sum_{s\in S_p} q^k_s f_{sp}-\sum_{s\in S_{t_n}} q^k_{t_n} f_{st_n}-\sum_{p\in P_{t_n}} \sum_{s\in S_p} q^k_{t_n} f_{sp} \leq 0$. This can be expressed as constraint (3.6) which is linear. We note that, when all the flow in the network is directed to a single terminal t_n, the absolute amount of CO_2 after natural gas is blended in the pools is $\sum_{p\in P_{t_n}} \sum_{s\in S_p} q^k_s f_{sp}$ from the pools $p \in P_{t_n}$ to t_n. Whereas, when the network has multiple terminals to which the flow is directed to, the absolute amount of CO_2 from the pools $p \in P_t$ to a terminal $t \in T$ is $\sum_{p\in P_t} w^k_p f_{pt}$ which is a bi-linear function. Therefore, only when the pooling problem network has a single terminal, it can be expressed as a linear program.

The initial step of our basic construction method constitutes of ranking the terminals. We let F_{ij} denote the flow accumulation on arc $(i, j) \in A$ and set it to 0 initially. In order to rank the terminals, the LP given by (3.1)–(3.7) is solved for each terminal. As a result of solving the LP for all terminals, the terminal which receives flow at minimum cost is ranked one and so on. In ranked order, let the terminals be denoted by $t_1, t_2...t_{|T|}$.

In the next step, we start with the terminal which is ranked first, i.e. t_1. Optimal flow is given to this terminal by solving the LP (3.1)–(3.7). In order to protect the higher ranked terminals from quality deterioration, we block the pools assigning flow to the current terminal, i.e. no more flow can be received by these pools and no more flow can be sent from them to any other terminal. Therefore, we set the value of $open_p$ to 0 for the pools p that we block. The value of F_{ij} is updated if there is some flow in the network. This constitutes one iteration in the basic construction heuristic algorithm. After that, the terminal ranked second is selected, i.e. t_2. We have to deduct the flow already assigned by the sources in the previous iteration from the source flow capacity. Hence, in constraint (3.2), the sum of all the flow assigned by source $s \in S$, i.e. $\sum_{i\in P_s\cup T_s} F_{si}$ is subtracted from b_s. Even when all pools are closed, the remaining terminals can still receive flow directly from the sources if it is profitable. This procedure continues until all terminals are processed, as stated in Algorithm 2.

Since our basic construction heuristic solves only a linear program in each iteration, it is expected to be faster than construction methods based on bi-linear sub-problems [3].

Algorithm 2 Basic construction method.

```
1: $F \leftarrow 0$
2: $open_p \leftarrow 1$ for all $p \in P_{t_n}$
3: Solve an LP given by (3.1)–(3.7) for each $t \in T$.
4: Rank the terminals $t_1, t_2 ... t_{|T|}$ by the criterion of minimum cost of sending
   flow to it.
5: $n \leftarrow 1$
6: repeat
7:     $t \leftarrow t_n$
8:     $\hat{f} \leftarrow$ optimal solution to (3.1)–(3.7)
9:     for $p \in P$ do
10:        if $\hat{f}_{pt_n} > 0$ then
11:            $open_p \leftarrow 0$
12:        end if
13:    end for
14:    $F \leftarrow F + \hat{f}$
15:    $n \leftarrow n + 1$
16: until $n > |T|$
```

3.2.1 Construction heuristic

The basic construction heuristic stated above can be improved so that we get a better feasible solution, and thereby an upper bound on the optimal objective function value that is closer to the global solution. Blocking pools that send flow to the current terminal is unnecessarily strict. Therefore, the idea of improving it is to reuse the pools. In addition, preliminary experiments suggest that the terminals that received zero flow during the ranking process, do not contribute to the profit. Hence, we eliminate such terminals and as a result of it, the method becomes fast as well. Henceforth, we refer to this method when applying the term *construction heuristic*.

For simplicity, we divide the pools into 2 sets. Let U denote the set of pools that are used and the set U' denote the pools that are unused. For each terminal $t \in T$, we denote the sets of used and unused neighbor pools by $U_t = \{u \in U : (u,t) \in A\}$ and $U'_t = \{u' \in U' : (u',t) \in A\}$, respectively.

The LP that is solved to rank the terminals and that assigns optimal flow to the selected terminal t_n is stated as follows:

$$\min_f \sum_{(i,j)\in A} c_{ij} f_{ij} \tag{3.8}$$

$$\text{s.t.} \sum_{p\in P_s} f_{sp} + f_{st_n} \le b_s - \sum_{i\in P_s\cup T_s} F_{si} \qquad s \in S \tag{3.9}$$

$$\sum_{s\in S_p} f_{sp} \le b_p - F_{pt_n} \qquad p \in U_{t_n} \tag{3.10}$$

$$\sum_{s\in S_{t_n}} f_{st_n} + \sum_{p\in P_{t_n}} f_{pt_n} \le b_{t_n} \tag{3.11}$$

$$\sum_{s\in S_p} f_{sp} - f_{pt_n} = 0 \qquad p \in U_{t_n} \tag{3.12}$$

$$\sum_{p\in U_{t_n}} (q_p^k - q_{t_n}^k) f_{pt_n} + \sum_{p\in U'_{t_n}} \sum_{s\in S_p} (q_s^k - q_{t_n}^k) f_{sp} + \sum_{s\in S_{t_n}} (q_s^k - q_{t_n}^k) f_{st_n} \le 0 \qquad k \in K \tag{3.13}$$

$$f_{ij} \ge 0 \qquad (i,j) \in A \tag{3.14}$$

$$\sum_{s\in S_p} (q_p^k - q_s^k) f_{sp} \ge 0 \qquad p \in U_{t_n} \tag{3.15}$$

Constraints (3.9), (3.11), (3.12) and (3.14) are the same as constraints (3.2), (3.4), (3.5) and (3.7), respectively. As the pools are not blocked any more, we deduct the flow already assigned by the used pools from the pool flow capacity. Hence, in constraint (3.10), the flow assigned by $p \in U_{t_n}$, i.e. F_{pt_n} is subtracted from b_p.

Flow to terminal t_{n+1} may intersect the pools supporting $t_1, t_2..., t_n$. Hence, we have to make sure that the quality at the terminals does not deteriorate as a result of more flow assigned to the pools. Since in iterations $n+1, n+2..., |T|$, we do not send flow to terminal t_n, quality deterioration at this terminal is avoided if we add the constraints that the quality at the pools sending flow to t_n does not deteriorate. The *pool qualities* are defined as $w_p^k = \frac{\sum_{s\in S_p} q_s^k f_{sp}}{\sum_{s\in S_p} f_{sp}}$ $(p \in P,\ k \in K)$. That is, the quality at pool p is a weighted average of the input qualities at the sources from which the pool receives flow, and the weights are identical to the flow values. We define the required pool quality q_p^k for $p \in U_{t_n}$ corresponding to attribute $k \in K$, to be identical to the current pool quality $\frac{\sum_{s\in S_p} q_s^k F_{sp}}{\sum_{s\in S_p} F_{sp}}$. The pool quality after flow is assigned to t_{n+1} should be at least as good as the quality before the flow assignment. This gives the inequality $q_p^k \geq w_p^k$. Thus, $q_p^k \geq \frac{\sum_{s\in S_p} q_s^k f_{sp}}{\sum_{s\in S_p} f_{sp}}$. Multiplying q_p^k with $\sum_{s\in S_p} f_{sp}$ and moving $\sum_{s\in S_p} q_s^k f_{sp}$ to the left hand side of the inequality, we get: $q_p^k \sum_{s\in S_p} f_{sp} - \sum_{s\in S_p} q_s^k f_{sp} \geq 0$ which can be expressed as the linear constraint (3.15).

The quality bounds at terminal t_n are given by constraint (3.13). The current terminal t_n receives flow from U, U' and S. The total flow at t_n is: $\sum_{p\in U_{t_n}} f_{pt_n} + \sum_{p\in U'_{t_n}} \sum_{s\in S_p} f_{sp} + \sum_{s\in S_{t_n}} f_{st_n}$. Assuming again that the quality attribute k corresponds to relative CO_2-content, the total amount of the contaminant CO_2 sent to t_n is: $\sum_{p\in U_{t_n}} w_p^k f_{pt_n} + \sum_{p\in U'_{t_n}} \sum_{s\in S_p} q_s^k f_{sp} + \sum_{s\in S_{t_n}} q_s^k f_{st_n}$ which is a bi-linear expression. Hence, the relative CO_2 content at t_n becomes:
$\frac{\sum_{p\in U_{t_n}} w_p^k f_{ptn} + \sum_{p\in U'_{tn}} \sum_{s\in S_p} q_s^k f_{sp} + \sum_{s\in S_{tn}} q_s^k f_{stn}}{\sum_{p\in U_{tn}} f_{ptn} + \sum_{p\in U'_{tn}} \sum_{s\in S_p} f_{sp} + \sum_{s\in S_{tn}} f_{stn}} \leq q_{tn}^k$. Multiplying q_{tn}^k with the denominator of the fraction, moving it to the left hand side of the inequality, rearranging it, and replacing w_p^k with q_p^k, we get constraint (3.13). This constraint is overly strict $(q_p^k \geq w_p^k)$ which is a deliberate choice we made as q_p^k is a constant, thus keeping the sub-problem linear.

We now propose three versions of the construction heuristic named *no* *ranking*, *one time ranking* and *re-ranking*. The one time ranking method proceeds in the same way as the basic construction heuristic except that in this method, the pools are not blocked and the terminals that received zero flow during the ranking process are eliminated. We denote the set of all the terminals that receive non-zero flow during the ranking process by T'. The method proceeds as shown in Algorithm 3. In the no ranking method, the terminals are not ranked at all. In this method, optimal flow is sent to each terminal by selecting them in the order in which they are input. Apart from the ranking strategy, the rest of the algorithm proceeds in the same way as Algorithm 3. In the re-ranking method, all the terminals which are ranked after the current terminal, are re-ranked by the end of each iteration. Here again, the terminals that receive zero flow during the ranking process are eliminated and the set T' is updated. In the next iteration, the terminal that is ranked first according to the new ranking system, is selected. The re-ranking process is reflected in Algorithm 3 by replacing steps 16 and 17 with steps 18 to 23.

Algorithm 3 The construction method.

```
 1: $F \leftarrow 0$
 2: $U \leftarrow \{\ \}$ (the set of used pools)
 3: $T' \leftarrow T$ (the set of all terminals that receive non-zero flow)
 4: *Solve an LP given by (3.8)–(3.14) for each $t \in T'$.
 5: $T' \leftarrow T' - \{t \in T' : \sum_{i \in S_t \cup P_t} f_{it} = 0\}$
 6: Rank the terminals $t_1, t_2 ... t_{|T'|}$ by the criterion of minimum cost of sending
    flow to it.
 7: $n \leftarrow 1$
 8: repeat
 9:     $t \leftarrow t_n$
10:     for $p \in U_{t_n}$ , $k \in K$ do
11:         $q_p^k \leftarrow \frac{\sum_{s \in S_p} q_s^k F_{sp}}{\sum_{s \in S_p} F_{sp}}$
12:     end for
13:     $\hat{f} \leftarrow$ optimal solution to (3.8)–(3.15)
14:     $U \leftarrow U \cup \{p \in P : \sum_{p \in P_{t_n}} \hat{f}_{pt_n} > 0\}$
15:     $F \leftarrow F + \hat{f}$
16:     $n \leftarrow n + 1$
17: until $n > |T'|$
```

*For the no ranking method, steps 4 to 6 are omitted and $t_1, t_2 ... t_{|T|}$ denote the terminals in the order they are input to the algorithm.

Additional steps for the re-ranking method.

```
18: $T' \leftarrow T' - t_n$
19: Solve an LP given by (3.8)–(3.15) for each $t \in T'$.
20: $T' \leftarrow T' - \{t \in T' : \sum_{i \in S_t \cup P_t} \hat{f}_{it} = 0\}$
21: New rank of the terminals is $t_1, t_2 ... t_{|T'|}$ by the criterion of minimum cost
    of sending flow to it.
22: $n \leftarrow 1$
23: **until** $T' \neq \{\ \}$
```

Chapter 4

Relaxations

4.1 Fast computation of lower bounds on the minimum cost

Since the independent flow relaxation (2.30)–(2.37) is defined by removing constraints from the STP-formulation of the pooling problem, its optimal objective function value, z_0, is a lower bound on the optimal objective function value, z^*, of the pooling problem. That is, $z_0 \leq z^*$, and by solving (2.30)–(2.37), we easily get a bound on how good any solution possibly can be. Computing z^* is not realistic in large instances, and therefore, we have to compare the solutions from our heuristics to bounds like e.g. z_0.

Unfortunately, the bound z_0 may be very weak in the sense that it is much smaller than z^*. In such cases, it tells little about how good our heuristic solutions are. A better bound can be obtained by adding more valid constraints.

Since pool p receives only a fraction y_p^s of its flow from source $s \in S_p$, the flow f_{sp} can at most spend the same proportion of the pool capacity b_p. Hence, $f_{sp} \leq b_p y_p^s$ is a valid constraint for all $s \in S$, $p \in P_s$ and $t \in T_p$. Analogously, we get $f_{pt} \leq b_p y_p^t$. Further, let $b_{sp} = \min\{b_s, b_p\}$ and $b_{pt} = \min\{b_p, b_t\}$ be the flow capacities of arcs (s,p) and (p,t), respectively. Since the flow from source s occupies at most a proportion y_p^s of the capacity of arc (p,t), we have the constraint $x_{spt} \leq b_{pt} y_p^s$. Likewise, we get $x_{spt} \leq b_{sp} y_p^t$.

In Algorithm 4, we demonstrate how the suggested constraints can be added gradually in order to improve the lower bound. We start out by solving the independent flow relaxation, which contains no new constraints and no proportion variables. Once the relaxation is solved, we estimate the excluded proportions as suggested in Algorithm 1. All violated constraints of the types discussed above are added to the relaxation, which is solved again. This procedure is repeated until no violations are found. Note that we gradually extend two initially empty sets of pools, denoted PS and PT. The former contains all pools $p \in P$ for which the source proportions y_p^s have been added, and the latter holds the same information related to terminal proportions. We note that the constraints in the independent flow relaxation are a proper subset of the constraints in the STP-relaxation proposed in [2]. Constraints added in Algorithm 4 are violated STP-constraints. Hence, the algorithm converges to the STP-relaxation.

Algorithm 4 Lower bound on optimal cost.

```
 1: $PS \leftarrow \emptyset$, $PT \leftarrow \emptyset$
 2: Let LP be the LP-problem (2.30)–(2.37)
 3: Let $\left(\hat{f}, \hat{x}\right)$ be an optimal solution to LP
 4: **repeat**
 5:   **for** $s \in S$, $p \in P_s$, $t \in T_p$ **do**
 6:     **if** $p \notin PS$ **then**
 7:       $\hat{y}_p^s \leftarrow \frac{\hat{f}_{sp}}{\sum_{t \in T_p} \hat{f}_{pt}}$
 8:     **end if**
 9:     **if** $p \notin PT$ **then**
10:       $\hat{y}_p^t \leftarrow \frac{\hat{f}_{pt}}{\sum_{s \in S_p} \hat{f}_{sp}}$
11:     **end if**
12:     **if** $\hat{f}_{sp} > b_p \hat{y}_p^s$ **then**
13:       Extend LP by the constraint $f_{sp} \leq b_p y_p^s$
14:       **if** $p \notin PS$ **then**
15:         Extend LP by the constraints $\sum_{s \in S_p} y_p^s = 1$ and $y_p^s \geq 0$
16:         $PS \leftarrow PS \cup \{p\}$
17:       **end if**
18:     **end if**
```

```
19:         if $\hat{f}_{pt} > b_p \hat{y}_p^t$ then
20:             Extend LP by the constraint $f_{pt} \leq b_p y_p^t$
21:             if $p \notin PT$ then
22:                 Extend LP by the constraints $\sum_{t \in T_p} y_p^t = 1$ and $y_p^t \geq 0$
23:                 $PT \leftarrow PT \cup \{p\}$
24:             end if
25:         end if
26:         if $\hat{x}_{spt} > b_{pt} \hat{y}_p^s$ then
27:             Extend LP by the constraint $x_{spt} \leq b_{pt} y_p^s$
28:             if $p \notin PS$ then
29:                 Extend LP by the constraints $\sum_{s \in S_p} y_p^s = 1$ and $y_p^s \geq 0$
30:                 $PS \leftarrow PS \cup \{p\}$
31:             end if
32:         end if
33:         if $\hat{x}_{spt} > b_{sp} \hat{y}_p^t$ then
34:             Extend LP by the constraint $x_{spt} \leq b_{sp} y_p^t$
35:             if $p \notin PT$ then
36:                 Extend LP by the constraints $\sum_{t \in T_p} y_p^t = 1$ and $y_p^t \geq 0$
37:                 $PT \leftarrow PT \cup \{p\}$
38:             end if
39:         end if
40:     end for
41:     $\left(\hat{f}, \hat{x}, \hat{y}\right) \leftarrow$ optimal solution to LP
42: until no violations were found
43: return the optimal objective function value of LP
```

Chapter 5

Numerical experiments

All experiments were run on a 64 bit Ubuntu (Release 12.04) computer equipped with 3.00 GHz Intel(R) quad-core processor and 8GB RAM. All models and algorithms have been written in GAMS modeling language.

5.1 Test instances

The experiments were carried out on 13 small-scale and 20 large-scale standard pooling problem instances, the size of which are shown in Table 5.1 and 5.2, respectively. All instances can be downloaded in GAMS-format from `http://www.ii.uib.no/~mohammeda/spooling/`. In Table 5.1, the instance identifier is given in the first column. Columns 2-5 give the number of sources, pools, terminals and quality attributes for each small-scale instance. In Table 5.2, columns 1-5 give the same type of information as given in Table 5.1 and column 6 gives the number of arcs for each large-scale instance.

Table 5.1: Size of small-scale instances.

Instance	$\|S\|$	$\|P\|$	$\|T\|$	$\|K\|$
Adhya1	5	2	4	4
Adhya2	5	2	4	6
Adhya3	8	3	4	6
Adhya4	8	2	5	4
Bental4	4	1	2	1
Bental5	5	3	5	2
Foulds2	6	2	4	1
Foulds3	11	8	16	1
Foulds4	11	8	16	1
Foulds5	11	4	16	1
Haverly1	3	1	2	1
Haverly2	3	1	2	1
Haverly3	3	1	2	1

Table 5.2: Size of large-scale instances.

Instance	$\|S\|$	$\|P\|$	$\|T\|$	$\|K\|$	$\|A\|$
A0	20	10	15	24	171
A1	20	10	15	24	179
A2	20	10	15	24	192
A3	20	10	15	24	218
A4	20	10	15	24	248
A5	20	10	15	24	277
A6	20	10	15	24	281
A7	20	10	15	24	325
A8	20	10	15	24	365
A9	20	10	15	24	407
B0	35	17	21	34	384
B1	35	17	21	34	515
B2	35	17	21	34	646
B3	35	17	21	34	790
B4	35	17	21	34	943
B5	35	17	21	34	1044
C0	60	15	50	40	811
C1	60	15	50	40	1070
C2	60	15	50	40	1248
C3	60	15	50	40	1451

5.2 Experiments with heuristic methods

In this section, we present the results of numerical experiments performed in order to analyze the efficiency of different heuristic methods that were proposed in Chapter 3.

In the first experiment, we have implemented the improvement heuristic described in Section 3.1. In this method, the initial solution is obtained by solving the independent flow relaxation (IFR) defined by (2.30)–(2.37). It is a sub-problem of the pooling problem obtained by eliminating all the constraints involving the proportion variables from the STP-formulation. Consequently, the relaxation is a linear program. There are two starting modes in which the proportion values are estimated: S and T. If the starting mode is S, the source proportions are estimated first and if the starting mode is T, then the terminal proportions are estimated first. Table 5.3 shows the results of the experiment for small-scale instances. The first column gives the instance identifier. Columns 2-4 report the elapsed time (in seconds), the number of iterations and the objective function value of the best solution found using the improvement heuristic, respectively, when the heuristic starts from S-mode. Columns 5-7 report the same type of information when the heuristic starts from T-mode. The elapsed times reported in columns 2 and 5, include the time spent in finding the solution to the independent flow relaxation. Column 8 states the optimal solution. In each instance, the best solution is given in bold. Henceforth, we use the same strategy to bold values in all the tables.

Table 5.3: Comparison between the improvement heuristic and the optimal solution for small-scale instances.

Instance	Start from S mode			Start from T mode			
	time(sec)	#It.	Obj. value	time(sec)	#It.	Obj. value	Opt. solution
Adhya1	0.33	3	-68.74	0.18	1	0	**-549.80**
Adhya2	0.20	1	0	0.35	3	-549.33	**-549.80**
Adhya3	0.21	1	0	0.35	3	-549.33	**-561.05**
Adhya4	0.33	3	-372.54	0.21	1	0	**-877.65**
Bental4	0.20	1	0	0.38	4	**-450.00**	**-450.00**
Bental5	0.23	2	**-3500.00**	0.27	2	**-3500.00**	**-3500.00**
Foulds2	0.31	2	**-1100.00**	0.26	2	**-1100.00**	**-1100.00**
Foulds3	0.30	2	**-8.00**	0.27	1	0	**-8.00**
Foulds4	0.31	2	-7.50	0.38	3	**-8.00**	**-8.00**
Foulds5	0.29	2	**-8.00**	0.35	2	-0.5	**-8.00**
Haverly1	0.21	1	0	0.32	3	**-400.00**	**-400.00**
Haverly2	0.32	3	**-600.00**	0.21	1	0	**-600.00**
Haverly3	0.21	1	0	0.30	3	**-750.00**	**-750.00**

Table 5.4 gives the results of the same experiment described above for large-scale instances. Columns 1-7 give the same type of information as given in Table 5.3 for the respective columns. Column 8 reports the upper bound

on the global minimum cost, obtained by applying the STP-formulation on BARON as global solver. These results were found when BARON was interrupted after one CPU-hour of computations, if it could not solve the problem. We name this as the *interrupted BARON* solution denoted as *Int. BARON* in the table.

Table 5.4: Comparison between the improvement heuristic and the bound from a global solver for large-scale instances.

Instance	Start from S mode			Start from T mode			Int. BARON
	time(sec)	#It.	ub	time(sec)	#It.	ub	ub
A0	1.33	11	-24613.94	0.75	4	-13642.09	**-35812.33**
A1	1.22	9	-14486.35	0.66	4	-11778.18	**-29276.56**
A2	0.78	5	-13889.79	0.70	2	-4703.16	**-23042.04**
A3	2.58	19	-36037.89	0.67	4	-29675.13	**-39446.54**
A4	1.61	9	-33565.23	1.40	9	**-36534.11**	-33687.13
A5	1.68	8	-20850.75	2.40	13	-20567.13	**-24015.54**
A6	3.61	21	**-41312.16**	5.18	30	-41004.36	-37074.67
A7	5.41	27	-39665.69	5.20	24	**-42034.55**	-38074.67
A8	1.99	9	-29202.94	2.66	12	**-30075.17**	-28795.26
A9	1.89	7	-18763.35	3.61	16	-21750.30	**-21912.35**
B0	10.91	51	-31145.11	5.89	26	**-41481.88**	-20802.12
B1	2.47	7	-55690.60	4.48	14	**-59568.55**	-50055.21
B2	14.83	35	-46046.19	10.48	23	**-52504.93**	-18567.64
B3	30.67	50	-63843.69	16.23	26	**-73469.18**	-18327.40
B4	56.05	65	-56762.53	31.90	36	**-58929.99**	-3711.84
B5	15.42	13	-56252.02	20.52	18	**-59382.28**	-10653.72
C0	18.02	35	**-69821.59**	20.81	35	-28379.66	-15197.45
C1	26.90	33	**-72821.42**	13.95	13	-33152.34	-25196.93
C2	49.63	40	**-109122.14**	12.31	6	-40873.07	-7497.52
C3	37.19	21	-95101.15	30.24	15	**-112167.75**	-7163.54

The results from the above table show that the improvement heuristic gave better upper bounds than the global solver in 14 (A4, A6-A8, B0-B5, C0-C3) out of 20 large-scale instances. It can also be observed from Tables 5.3 and 5.4, that in most of the instances, the modes from which the improvement heuristic starts with, i.e. S and T-modes can make a big difference on how good the objective function values are. We can observe from Table 5.4 that, in most large-scale instances, starting from the T-mode gives better results than starting from the S-mode and the T-mode is faster, hence needing fewer iterations. We also note that the improvement heuristic is very fast compared to the global solver. The results obtained from the global solver required one

CPU-hour of computations while those from the improvement heuristic take less than a minute in all the instances, and the longest time recorded is 56.05 seconds in instance B4 when the method started from the S-mode.

In the second experiment, we have implemented the basic construction heuristic and the one time ranking version of the construction heuristic described in Sections 3.2 and 3.2.1, respectively. We have then made a comparison between the two methods and the global solution. Table 5.5 shows the results of the experiment for small-scale instances. The instance identifiers are given in the first column. Columns 2-3 report the elapsed time (in seconds) and the objective function value of the best solution found by using the basic construction heuristic, respectively. Columns 4-5 report the same type of information when we use the one time ranking version of the construction heuristic. Column 6 states the optimal solution.

Table 5.5: Comparison between the basic construction heuristic, the construction heuristic and the optimal solution for small-scale instances.

Instance	Basic construction heuristic		Construction heuristic		
	time(sec)	Obj. value	time(sec)	Obj. value	Opt. solution
Adhya1	0.79	-509.78	0.43	-509.78	**-549.80**
Adhya2	0.80	-509.78	0.43	-509.78	**-549.80**
Adhya3	0.96	-518.98	0.40	-552.85	**-561.05**
Adhya4	0.88	-505.61	0.50	**-877.65**	**-877.65**
Bental4	0.34	**-450.00**	0.39	**-450.00**	**-450.00**
Bental5	0.72	-3100.00	0.64	**-3500.00**	**-3500.00**
Foulds2	0.59	-1000.00	0.51	**-1100.00**	**-1100.00**
Foulds3	2.40	-3.00	1.99	-6.00	**-8.00**
Foulds4	3.97	-3.00	2.10	-7.50	**-8.00**
Foulds5	3.46	-1.50	2.02	**-8.00**	**-8.00**
Haverly1	0.97	**-400.00**	0.28	**-400.00**	**-400.00**
Haverly2	0.29	**-600.00**	0.28	**-600.00**	**-600.00**
Haverly3	0.80	**-750.00**	0.29	**-750.00**	**-750.00**

The results in the above table show that in terms of solution quality, the construction heuristic performed as well as or better than the basic construction heuristic in all the instances. The construction heuristic is also observed to be faster than the other method in all the instances except Bental4.

Table 5.6 gives the results of the same experiment described above for large-scale instances. Columns 1-5 give the same type of information as given in Table 5.5 for the respective columns. Column 6 states the interrupted BARON solution.

Table 5.6: Comparison between the basic construction heuristic, the construction heuristic and the bound from a global solver for large-scale instances.

Instance	Basic construction heuristic		Construction heuristic		Int. BARON
	time(sec)	ub	time(sec)	ub	ub
A0	2.63	-22574.21	1.92	-23620.99	**-35812.33**
A1	2.60	-20981.72	1.71	-23068.09	**-29276.56**
A2	2.63	-14107.38	1.76	-14134.63	**-23042.04**
A3	2.87	-26388.90	2.00	-29311.53	**-39446.54**
A4	4.02	-29385.01	2.06	-28173.99	**-33687.13**
A5	2.68	-16339.52	1.73	-18698.44	**-24015.54**
A6	2.27	-26905.93	3.29	-28087.85	**-37074.67**
A7	2.97	-29889.10	2.20	-31807.11	**-38074.67**
A8	3.50	-24213.43	2.45	-24092.10	**-28795.26**
A9	2.38	-14342.84	1.71	-14342.84	**-21912.35**
B0	4.69	-27764.53	3.63	**-29099.91**	-20802.12
B1	6.04	-44195.44	3.81	-48000.75	**-50055.21**
B2	4.78	-37618.34	4.32	**-38592.58**	-18567.64
B3	5.43	-49463.28	4.30	**-52800.38**	-18327.40
B4	5.38	-44648.39	4.44	**-47552.56**	-3711.84
B5	6.32	-40743.71	5.25	**-41808.46**	-10653.72
C0	12.79	-51200.65	13.90	**-54381.41**	-15197.45
C1	10.27	-63415.24	14.80	**-67526.37**	-25196.93
C2	11.50	-68514.59	16.64	**-75472.87**	-7497.52
C3	12.74	-73872.22	15.60	**-85085.64**	-7163.54

The results from the above table show that in terms of solution quality, the construction heuristic performed as well as or better than the basic construction heuristic in all the instances except A4 and A8. Except 5 instances (A6 and C0-C3), the construction heuristic is observed to be faster than the other method.

We note that both the heuristics in this experiment are faster than the global solver. We observe that in 11 instances (A0-A9 and B1), the global solver provides better results than both the heuristics after one CPU-hour. However, in the remaining instances (B0, B2-B5 and C0-C3), the construction heuristic found better upper bounds than the global solver in less than twenty seconds. The longest time recorded is 16.64 seconds in instance C2.

In the third experiment, we have implemented three versions of the construction heuristic described in Section 3.2.1. Tables 5.7 and 5.8 show the results of the experiment for small-scale and large-scale instances, respectively. In these tables, the instance identifiers are given in the first column. Columns 2 and 3 report the elapsed time (in seconds) and the objective function value, respectively, obtained by using the construction heuristic with the no ranking method. Columns 4-5 and 6-7 give the same type of information as given in columns 2 and 3 by using the one time ranking and the re-ranking methods, respectively. In Table 5.7, column 8 states the optimal solution, while in Table 5.8, it states the interrupted BARON solution.

Table 5.7: Comparison between the three versions of the construction heuristic and the optimal solution for small-scale instances.

Instance	No ranking		One time ranking		Re-ranking		
	time(sec)	Obj. value	time(sec)	Obj. value	time(sec)	Obj. value	Opt. solution
Adhya1	0.25	-509.78	0.43	-509.78	0.51	-509.78	**-549.80**
Adhya2	0.23	-509.78	0.43	-509.78	0.44	-509.78	**-549.80**
Adhya3	0.24	-552.85	0.40	-552.85	0.56	-552.85	**-561.05**
Adhya4	0.30	-470.83	0.50	**-877.65**	0.91	**-877.65**	**-877.65**
Bental4	0.15	-100.00	0.39	**-450.00**	0.31	**-450.00**	**-450.00**
Bental5	0.29	**-3500.00**	0.64	**-3500.00**	1.37	**-3500.00**	**-3500.00**
Foulds2	0.28	**-1100.00**	0.51	**-1100.00**	0.93	**-1100.00**	**-1100.00**
Foulds3	1.08	**-8.00**	1.99	-6.00	7.06	-6.50	**-8.00**
Foulds4	1.09	**-8.00**	2.10	-7.50	10.01	-7.50	**-8.00**
Foulds5	0.99	**-8.00**	2.02	**-8.00**	13.19	**-8.00**	**-8.00**
Haverly1	0.14	-100.00	0.28	**-400.00**	0.32	**-400.00**	**-400.00**
Haverly2	0.15	**-600.00**	0.28	**-600.00**	0.34	**-600.00**	**-600.00**
Haverly3	0.16	-125.00	0.29	**-750.00**	0.33	**-750.00**	**-750.00**

From Table 5.7, we observe that in instances Foulds3 and Foulds4, the no ranking method gave the optimal solution while the other two methods did not. In instances Adhya4, Bental4, Haverly1 and Haverly3, the no ranking method could not give the optimal solution unlike the other two methods. In the rest of the instances, all three methods gave the same solution.

Table 5.8: Comparison between the three versions of the construction heuristic and the bound from a global solver for large-scale instances.

Instance	No ranking		One time ranking		Re-ranking		Int. BARON
	time(sec)	ub	time(sec)	ub	time(sec)	ub	ub
A0	1.12	-22763.91	1.92	-23620.99	3.85	-24320.97	**-35812.33**
A1	1.16	-12027.73	1.71	-23068.09	3.72	-23068.09	**-29276.56**
A2	1.15	-7407.32	1.76	-14134.63	2.40	-14134.63	**-23042.04**
A3	1.13	-23580.98	2.00	-29311.53	4.73	-27056.05	**-39446.54**
A4	1.23	-24000.23	2.06	-28173.99	4.33	-29462.14	**-33687.13**
A5	1.15	-18058.07	1.73	-18698.44	3.29	-18364.42	**-24015.54**
A6	1.34	-30202.84	3.29	-28087.85	6.42	-28400.16	**-37074.67**
A7	1.24	-32612.05	2.20	-31807.11	4.07	-31807.11	**-38074.67**
A8	1.18	-20689.08	2.45	-24092.10	6.16	-24092.10	**-28795.26**
A9	1.16	-15462.14	1.71	-14342.84	2.93	-14342.84	**-21912.35**
B0	2.51	-26075.28	3.63	**-29099.91**	7.08	**-29099.91**	-20802.12
B1	2.56	-41381.16	3.81	-48000.75	9.90	-45818.85	**-50055.21**
B2	2.90	-27296.44	4.32	**-38592.58**	10.44	-35835.71	-18567.64
B3	2.94	**-55667.34**	4.30	-52800.38	12.24	-49880.27	-18327.40
B4	2.99	-44137.71	4.44	**-47552.56**	13.27	-47000.63	-3711.84
B5	3.21	-41396.91	5.25	-41808.46	14.32	**-43048.22**	-10653.72
C0	11.98	-50757.30	13.90	-54381.41	31.78	**-56092.77**	-15197.45
C1	12.72	-49163.00	14.80	**-67526.37**	32.91	-65713.76	-25196.93
C2	13.39	-55591.08	16.64	**-75472.87**	46.28	-74354.66	-7497.52
C3	13.59	-72177.07	15.60	**-85085.64**	41.88	-80173.56	-7163.54

From Table 5.8, we observe that the no ranking method gives better upper bounds than the other two methods in 4 instances (A6, A7, A9 and B3). The re-ranking method is also better than the other two methods in 4 different instances (A0, A4, B5 and C0), and the one time ranking method gives better bounds than the other two methods in 8 instances (A3, A5, B1, B2, B4 and C1-C3). In the rest of the instances, the one time ranking and the re-ranking methods give the same upper bounds, which are better than the bounds given by the no ranking method. We also observe that in 9 instances (B0, B2-B5 and C0-C3), the construction heuristic gives better upper bounds than the global solver. From the above two tables, we note that the algorithm without ranking is the fastest, and the re-ranking algorithm is the slowest. This is obvious, because many linear programs are solved for ranking the terminals.

In order to further improve the upper bounds on the optimal objective function value, we performed a fourth experiment. In this experiment, we used the solution that we obtained from the different versions of the construction heuristic as the starting solution for the improvement heuristic. The results of the experiment are recorded in Tables 5.9 to 5.14. The formats of all these tables are the same. The first column gives the instance identifier. Columns 2 and 3 report the elapsed time (in seconds) and the objective function value, respectively, found by using one of the three versions of the construction heuristic. These values are taken from Tables 5.7 and 5.8, and are only included to verify if the improvement method improved the solution of the different versions of the construction heuristic. Columns 4-5 and 6-7 report the same type of information as reported in columns 2-3 when the improvement method is used starting from S-mode and T-mode, respectively. The elapsed times reported in columns 4 and 6 include the time needed to construct the solution, i.e. the time reported in column 2. In Tables 5.9, 5.11 and 5.13, column 8 states the optimal solution, while in Tables 5.10, 5.12 and 5.14, it states the interrupted BARON solution.

Table 5.9: Results from the improvement heuristic with initial solution from the construction heuristic using no-ranking method for small-scale instances.

Instance	No ranking		Impr. method S mode		Impr. method T mode		Opt. solution
	time(sec)	Obj. value	time(sec)	Obj. value	time(sec)	Obj. value	
Adhya1	0.25	-509.78	0.42	-509.78	0.48	-509.78	**-549.80**
Adhya2	0.23	-509.78	0.51	-509.78	0.46	-509.78	**-549.80**
Adhya3	0.24	-552.85	0.51	-552.85	0.48	-552.85	**-561.05**
Adhya4	0.30	-470.83	0.48	-470.83	0.50	-470.83	**-877.65**
Bental4	0.15	-100.00	0.32	-100.00	0.30	-100.00	**-450.00**
Bental5	0.29	**-3500.00**	0.49	**-3500.00**	0.47	**-3500.00**	**-3500.00**
Foulds2	0.28	**-1100.00**	0.48	**-1100.00**	0.41	**-1100.00**	**-1100.00**
Foulds3	1.08	**-8.00**	1.18	**-8.00**	1.19	**-8.00**	**-8.00**
Foulds4	1.09	**-8.00**	1.21	**-8.00**	1.33	**-8.00**	**-8.00**
Foulds5	0.99	**-8.00**	1.46	**-8.00**	1.47	**-8.00**	**-8.00**
Haverly1	0.14	-100.00	0.32	-100.00	0.32	-100.00	**-400.00**
Haverly2	0.15	**-600.00**	0.31	**-600.00**	0.32	**-600.00**	**-600.00**
Haverly3	0.16	-125.00	0.30	-125.00	0.30	-125.00	**-750.00**

Table 5.10: Results from the improvement heuristic with initial solution from the construction heuristic using no-ranking method for large-scale instances.

Instance	No ranking		Impr. method S mode		Impr. method T mode		Int. BARON
	time(sec)	ub	time(sec)	ub	time(sec)	ub	ub
A0	1.12	-22763.91	6.67	-29780.50	2.21	-24429.90	**-35812.33**
A1	1.16	-12027.73	2.15	-21710.13	2.41	-21002.94	**-29276.56**
A2	1.15	-7407.32	1.55	-7546.06	1.87	-7546.06	**-23042.04**
A3	1.13	-23580.98	3.53	-29818.48	4.19	-34468.35	**-39446.54**
A4	1.23	-24000.23	3.62	-31645.54	5.55	**-36930.53**	-33687.13
A5	1.15	-18058.07	4.36	**-24888.63**	4.25	-24838.55	-24015.54
A6	1.34	-30202.84	5.46	**-41670.94**	6.45	-41646.83	-37074.67
A7	1.24	-32612.05	5.46	**-39961.53**	5.72	-39877.52	-38074.67
A8	1.18	-20689.08	4.87	-24467.09	5.39	-22091.16	**-28795.26**
A9	1.16	-15462.14	4.61	-21062.36	4.73	-21256.51	**-21912.35**
B0	2.51	-26075.28	9.62	-37451.60	9.03	**-39761.39**	-20802.12
B1	2.56	-41381.16	11.24	-56375.22	7.53	**-58189.39**	-50055.21
B2	2.90	-27296.44	11.24	-51079.37	16.96	**-51526.15**	-18567.64
B3	2.94	-55667.34	26.29	**-68823.92**	13.05	-59995.96	-18327.40
B4	2.99	-44137.71	28.35	**-58924.51**	38.68	-56239.56	-3711.84
B5	3.21	-41396.91	35.94	-59725.70	26.67	**-59733.81**	-10653.72
C0	11.98	-50757.30	29.97	-72105.91	34.49	**-73286.24**	-15197.45
C1	12.72	-49163.00	45.49	**-77600.57**	64.59	-72423.04	-25196.93
C2	13.39	-55591.08	54.52	**-104662.46**	76.44	-101158.79	-7497.52
C3	13.59	-72177.07	60.92	**-105069.65**	90.87	-102406.46	-7163.54

Table 5.11: Results from the improvement heuristic with initial solution from the construction heuristic using one-time ranking method for small-scale instances.

Instance	One time ranking		Impr. method S mode		Impr. method T mode		
	time(sec)	Obj. value	time(sec)	Obj. value	time(sec)	Obj. value	Opt. solution
Adhya1	0.43	-509.78	0.82	-509.78	1.46	-509.78	**-549.80**
Adhya2	0.43	-509.78	0.96	-509.78	1.12	-509.78	**-549.80**
Adhya3	0.40	-552.85	1.47	-552.85	0.87	-552.85	**-561.05**
Adhya4	0.50	**-877.65**	0.81	**-877.65**	1.12	**-877.65**	**-877.65**
Bental4	0.39	**-450.00**	0.78	**-450.00**	0.75	**-450.00**	**-450.00**
Bental5	0.64	**-3500.00**	1.76	**-3500.00**	2.07	**-3500.00**	**-3500.00**
Foulds2	0.51	**-1100.00**	1.02	**-1100.00**	1.51	**-1100.00**	**-1100.00**
Foulds3	1.99	-6.00	3.12	**-8.00**	4.51	-6.00	**-8.00**
Foulds4	2.10	-7.50	3.02	-7.50	4.20	-7.50	**-8.00**
Foulds5	2.02	**-8.00**	3.99	**-8.00**	3.87	**-8.00**	**-8.00**
Haverly1	0.28	**-400.00**	1.48	**-400.00**	0.64	**-400.00**	**-400.00**
Haverly2	0.28	**-600.00**	1.22	**-600.00**	0.70	**-600.00**	**-600.00**
Haverly3	0.29	**-750.00**	0.86	**-750.00**	1.76	**-750.00**	**-750.00**

Table 5.12: Results from the improvement heuristic with initial solution from the construction heuristic using one-time ranking method for large-scale instances.

Instance	One time ranking		Impr. method S mode		Impr. method T mode		Int. BARON
	time(sec)	ub	time(sec)	ub	time(sec)	ub	ub
A0	1.92	-23620.99	5.60	-26591.74	4.04	-30337.70	**-35812.33**
A1	1.71	-23068.09	4.43	-23539.35	3.13	-23524.47	**-29276.56**
A2	1.76	-14134.63	6.71	-18917.70	3.24	-18817.43	**-23042.04**
A3	2.00	-29311.53	10.13	-31004.79	6.01	-31679.76	**-39446.54**
A4	2.06	-28173.99	7.48	**-37723.94**	6.68	-37567.94	-33687.13
A5	1.73	-18698.44	8.90	-23386.91	7.40	-22415.50	**-24015.54**
A6	3.29	-28087.85	6.06	-40482.00	4.70	**-41341.00**	-37074.67
A7	2.20	-31807.11	10.08	**-40769.62**	4.88	-40405.92	-38074.67
A8	2.45	-24092.10	12.79	-26626.57	10.57	**-29368.42**	-28795.26
A9	1.71	-14342.84	19.98	-21604.48	8.38	-21017.71	**-21912.35**
B0	3.63	-29099.91	16.47	-34910.51	8.18	**-35317.30**	-20802.12
B1	3.81	-48000.75	15.90	**-55929.08**	9.72	-54290.30	-50055.21
B2	4.32	-38592.58	21.22	-48820.71	22.41	**-49946.69**	-18567.64
B3	4.30	-52800.38	52.00	-69089.90	68.44	**-69856.72**	-18327.40
B4	4.44	-47552.56	38.60	-58886.14	50.91	**-58979.56**	-3711.84
B5	5.25	-41808.46	30.57	**-56538.58**	53.90	-55865.29	-10653.72
C0	13.90	-54381.41	48.15	**-69276.53**	29.74	-68978.86	-15197.45
C1	14.80	-67526.37	43.46	**-79698.41**	43.73	-76039.49	-25196.93
C2	16.64	-75472.87	80.14	-111453.91	71.44	**-117736.60**	-7497.52
C3	15.60	-85085.64	49.93	**-107587.08**	110.70	-103627.95	-7163.54

Table 5.13: Results from the improvement heuristic with initial solution from the construction heuristic using re-ranking method for small-scale instances.

Instance	Re-ranking		Impr. method S mode		Impr. method T mode		
	time(sec)	Obj. value	time(sec)	Obj. value	time(sec)	Obj. value	Opt. solution
Adhya1	0.51	-509.78	0.68	-509.78	0.62	-509.78	**-549.80**
Adhya2	0.44	-509.78	0.65	-509.78	0.59	-509.78	**-549.80**
Adhya3	0.56	-552.85	0.71	-552.85	0.90	-552.85	**-561.05**
Adhya4	0.91	**-877.65**	0.99	**-877.65**	0.92	**-877.65**	**-877.65**
Bental4	0.31	**-450.00**	0.67	**-450.00**	0.45	**-450.00**	**-450.00**
Bental5	1.37	**-3500.00**	1.78	**-3500.00**	1.64	**-3500.00**	**-3500.00**
Foulds2	0.93	**-1100.00**	1.14	**-1100.00**	1.02	**-1100.00**	**-1100.00**
Foulds3	7.06	-6.50	7.88	-6.50	7.04	-6.50	**-8.00**
Foulds4	8.83	-7.50	8.90	-7.50	9.21	-7.50	**-8.00**
Foulds5	9.10	**-8.00**	9.50	**-8.00**	9.26	**-8.00**	**-8.00**
Haverly1	0.32	**-400.00**	0.67	**-400.00**	0.45	**-400.00**	**-400.00**
Haverly2	0.34	**-600.00**	0.44	**-600.00**	0.44	**-600.00**	**-600.00**
Haverly3	0.33	**-750.00**	0.79	**-750.00**	0.51	**-750.00**	**-750.00**

Table 5.14: Results from the improvement heuristic with initial solution from the construction heuristic using re-ranking method for large-scale instances.

Instance	Re-ranking		Impr. method S mode		Impr. method T mode		Int. BARON
	time(sec)	ub	time(sec)	ub	time(sec)	ub	ub
A0	3.85	-24320.97	4.21	-27447.51	5.02	-27625.79	**-35812.33**
A1	3.72	-23068.09	4.36	-23539.35	5.05	-23524.47	**-29276.56**
A2	2.40	-14134.63	4.31	-18917.70	4.60	-18817.43	**-23042.04**
A3	4.73	-27056.05	6.31	-30893.75	8.43	-33468.45	**-39446.54**
A4	4.33	-29462.14	7.23	**-36271.35**	8.61	-36252.81	-33687.13
A5	3.29	-18364.42	6.04	-21562.94	7.03	-21628.55	**-24015.54**
A6	6.42	-28400.16	6.56	-40527.00	7.77	**-41341.00**	-37074.67
A7	4.07	-31807.11	7.02	**-40769.62**	7.14	-40405.92	-38074.67
A8	6.16	-24092.10	9.45	-26626.57	14.85	**-29368.42**	-28795.26
A9	2.93	-14342.84	15.16	-21604.48	9.26	-21017.71	**-21912.35**
B0	7.08	-29099.91	10.26	-34910.51	21.92	**-40421.55**	-20802.12
B1	9.90	-45818.85	16.10	**-57747.66**	18.20	-55808.31	-50055.21
B2	10.44	-35835.71	21.65	-46372.38	38.61	**-47921.77**	-18567.64
B3	12.24	-49880.27	41.36	**-71448.88**	30.57	-70086.77	-18327.40
B4	13.27	-47000.63	32.39	-56023.98	29.16	**-56670.11**	-3711.84
B5	14.32	-43048.22	36.76	**-57824.22**	67.22	-56557.78	-10653.72
C0	31.78	-56092.77	49.74	**-68031.43**	59.55	-67114.21	-15197.45
C1	32.91	-65713.76	47.26	-83248.11	58.81	**-83980.63**	-25196.93
C2	46.28	-74354.66	58.15	-95739.27	168.88	**-108483.44**	-7497.52
C3	41.88	-80173.56	108.68	**-107581.79**	61.09	-95373.62	-7163.54

From Tables 5.10, 5.12 and 5.14, we observe that for all large-scale instances, the solution produced by the construction heuristic has been improved by the improvement heuristic. We also observe from these tables that the modes from which the improvement heuristic starts with, i.e. S and T modes can make a difference on how good the objective function values are. However, based on the results obtained, we cannot say that one of the modes is always better than the other. Tables 5.9 and 5.13 show that in the small-scale instances, the improvement heuristic did not improve the solution obtained from the construction heuristic. We can observe the same thing from Table 5.11 with one exception, Foulds3, whose solution is improved to achieve the optimal solution.

In order to analyze the efficiency of the improvement method combined with the three versions of the construction heuristic, we report experiments comparing them in the next section.

5.2.1 Comparisons between heuristic methods

We now compare the results of the improvement heuristic starting with the solutions obtained from each of the three versions of the construction heuristic. Only the results for large-scale instances are compared. For the ease of comparison, we have collected the best values obtained by all these methods in Table 5.15. We also compare these solutions with the interrupted BARON solution. Column 1 of this table gives the instance identifier. Column 2 gives the elapsed time (in seconds), obtained by adding the time needed by the improvement method in the S and T-modes, and using the solution given by the no ranking method as initial solution. Column 3 reports the minimum of the objective function values in the S and T-modes obtained by the same method as mentioned above. Columns 4-5 and 6-7 report the same type of information as reported in columns 2-3, obtained by using the improvement method in combination with the one time ranking and the re-ranking methods, respectively. Column 8 gives the interrupted BARON solution.

Table 5.15: Comparing output from the improvement method in combination with the three versions of the construction heuristic for large-scale instances.

Instance	No ranking impr.		One time ranking impr.		Re-ranking impr.		Int. BARON
	time(sec)	ub	time(sec)	ub	time(sec)	ub	ub
A0	8.88	-29780.50	9.64	-30337.70	9.23	-27625.79	**-35812.33**
A1	4.56	-21710.13	7.56	-23539.35	9.41	-23539.35	**-29276.56**
A2	3.42	-7546.06	9.95	-18917.70	8.91	-18917.70	**-23042.04**
A3	7.72	-34468.35	16.14	-31679.76	14.74	-33468.45	**-39446.54**
A4	9.17	-36930.53	14.16	**-37723.94**	15.84	-36271.35	-33687.13
A5	8.61	**-24888.63**	16.30	-23386.91	13.07	-21628.55	-24015.54
A6	11.91	**-41670.94**	10.76	-41341.00	14.33	-41341.00	-37074.67
A7	11.18	-39961.53	14.96	**-40769.62**	14.16	**-40769.62**	-38074.67
A8	15.13	-24467.09	23.36	**-29368.42**	24.30	**-29368.42**	-28795.26
A9	9.34	-21256.51	28.36	-21604.48	24.42	-21604.48	**-21912.35**
B0	18.65	-39761.39	24.65	-35317.30	32.18	**-40421.55**	-20802.12
B1	18.77	**-58189.39**	25.62	-55929.08	34.30	-57747.66	-50055.21
B2	28.20	**-51526.15**	43.63	-49946.69	60.26	-47921.77	-18567.64
B3	39.34	-68823.92	120.44	-69856.72	71.93	**-71448.88**	-18327.40
B4	67.03	-58924.51	89.51	**-58979.56**	61.55	-56670.11	-3711.84
B5	62.61	**-59733.81**	84.47	-56538.58	103.98	-57824.22	-10653.72
C0	64.46	**-73286.24**	77.89	-69276.53	109.29	-68031.43	-15197.45
C1	110.08	-77600.57	87.19	-79698.41	106.07	**-83980.63**	-25196.93
C2	130.96	-104662.46	151.58	**-117736.60**	227.03	-108483.44	-7497.52
C3	151.79	-105069.65	160.63	**-107587.08**	169.77	-107581.79	-7163.54

From the above table, we observe that the no ranking method gives better upper bounds than the other two methods in 7 instances (A3, A5, A6, B1, B2, B5 and C0). The one time ranking method gives better bounds than the other two methods in 5 instances (A0, A4, B4, C2 and C3), and the re-ranking method is better than the other two methods in 3 instances (B0, B3 and C1). In the rest of the instances, the one time ranking and the re-ranking methods give the same upper bounds, which are better than the bounds given by the no ranking method. We note that the no ranking method is faster than the other two methods in all instances except 3 (A6, B4 and C1). Among the one time ranking and the re-ranking methods, the former method is faster than the latter in 12 instances (A1, A4, A6, A8, B0-B2, B5 and C0-C3). We also observe that for 15 instances (A4-A8, B0-B5 and C0-C3), we got better upper bounds than the interrupted version of global solver.

The results from Table 5.15 also indicate that we can not choose any one method which will always succeed in giving better feasible solutions than the remaining two methods. Since we do not know which of the three methods would perform best in terms of solution quality, we propose an approach in which all the three methods are run sequentially. Henceforth, this approach is referred to as *Impr2*, and it reports the best solution given among the three versions of the construction heuristic. The time required by Impr2 is the sum of the times required by each of the three improvement methods individually. The results obtained by applying Impr2 is reported in the following comparison.

Next, we compare Impr2 with the improvement method starting with the solution from the independent flow relaxation (named *Impr1*). Note that the experiments on Impr1 have already been performed at the beginning of Section 5.2. In Table 5.16, column 1 gives the instance identifier. Column 2 reports the elapsed time (in seconds), obtained by adding the time needed by Impr1 in the S and T-modes. Column 3 reports the minimum of the objective function values in the S and T-modes obtained by the same method. Column 4 states the elapsed time (in seconds), required by Impr2 and column 5 states the objective function value given by it. Column 6 reports the interrupted BARON solution.

Table 5.16: Comparison between Impr1, Impr2 and the bound from a global solver for large-scale instances.

Instance	Impr1		Impr2		Int. BARON
	time(sec)	ub	time(sec)	ub	ub
A0	2.08	-24613.94	27.75	-30337.70	**-35812.33**
A1	1.88	-14486.35	21.53	-23539.35	**-29276.56**
A2	1.48	-13889.79	22.28	-18917.70	**-23042.04**
A3	3.25	-36037.89	38.60	-34468.35	**-39446.54**
A4	3.01	-36534.11	39.17	**-37723.94**	-33687.13
A5	4.08	-20850.75	37.98	**-24888.63**	-24015.54
A6	8.79	-41312.16	37.00	**-41670.94**	-37074.67
A7	10.61	**-42034.55**	40.30	-40769.62	-38074.67
A8	4.65	**-30075.17**	62.79	-29368.42	-28795.26
A9	5.50	-21750.30	62.12	-21604.48	**-21912.35**
B0	16.80	**-41481.88**	75.48	-40421.55	-20802.12
B1	6.95	**-59568.55**	78.69	-58189.39	-50055.21
B2	25.31	**-52504.93**	132.09	-51526.15	-18567.64
B3	46.90	**-73469.18**	231.71	-71448.88	-18327.40
B4	87.95	-58929.99	218.09	**-58979.56**	-3711.84
B5	35.94	-59382.28	251.06	**-59733.81**	-10653.72
C0	38.83	-69821.59	251.64	**-73286.24**	-15197.45
C1	40.85	-72821.42	303.34	**-83980.63**	-25196.93
C2	61.94	-109122.14	509.57	**-117736.60**	-7497.52
C3	67.43	**-112167.75**	482.19	-107587.08	-7163.54

From Table 5.16, we observe that Impr1 is faster compared to Impr2. The longest time recorded for Impr1 is 87.95 seconds in instance B4, whereas, for Impr2 is 509.57 seconds in instance C2. We also note that in terms of solution quality, Impr2 performs better than Impr1 in 11 instances (A0-A2, A4-A6, B4-B5, C0-C2), whereas, Impr1 is better than Impr2 in the remaining 9 out of 20 large-scale instances. We observe that our heuristics have provided better solutions than the global solver in 15 (A4-A6, A7-A8, B0-B5, C0-C3) out of 20 large-scale instances.

5.3 Experiments with relaxations

In this section, we present the results of a numerical experiment performed in order to analyze the efficiency of the algorithm proposed in Chapter 4 for fast computation of lower bounds.

In this experiment, we have implemented the algorithm described in Section 4.1. The solution with which the algorithm starts, is obtained by solving the independent flow relaxation (IFR) defined by (2.30)–(2.37). As mentioned earlier, it is a sub-problem of the pooling problem obtained by eliminating all the constraints involving the proportion variables from the STP-formulation. Consequently, the relaxation is a linear program. After solving the independent flow relaxation, violated constraints are added to it and it is solved again, in order to improve the lower bound, until no violations are found.

Table 5.17 shows the results of the experiment for small-scale instances. Column 1 gives the instance identifier. Columns 2-3 report the elapsed time (in seconds) and the objective function value obtained by solving the independent flow relaxation, respectively. Columns 4-6 report the elapsed time (in seconds), the number of iterations needed, and the number of constraints added to the independent flow relaxation, respectively, by our algorithm. The elapsed time reported in column 4 includes the time required to solve the independent flow relaxation, i.e. the time reported in column 2. Columns 7-8 state the elapsed time (in seconds) and the lower bound on the global minimum cost, obtained by solving the STP-relaxation [2].

Table 5.17: Comparison between the time required by our algorithm and the STP-relaxation for small-scale instances.

Instance	time(sec)	IFR	time(sec)	#It.	#Const.	time(sec)	lb
Adhya1	0.16	-856.25	0.23	2	5	1.00	-840.27
Adhya2	0.16	-574.78	0.31	3	6	0.53	-574.78
Adhya3	0.19	-574.78	0.37	3	6	0.38	-574.78
Adhya4	0.15	-976.44	0.34	3	6	0.37	-961.93
Bental4	0.18	-550.00	0.38	3	9	1.06	-541.67
Bental5	0.24	-3500.00	0.26	2	2	0.40	-3500.00
Foulds2	0.17	-1100.00	0.22	2	2	0.42	-1100.00
Foulds3	0.20	-8.00	0.55	5	103	1.19	-8.00
Foulds4	0.21	-8.00	1.31	7	184	0.62	-8.00
Foulds5	0.25	-8.00	1.31	7	337	0.57	-8.00
Haverly1	0.17	-500.00	0.34	2	1	0.96	-500.00
Haverly2	0.18	-1000.00	0.32	2	1	0.85	-1000.00
Haverly3	0.14	-875.00	0.31	2	2	0.97	-800.00

From the above table we observe that our algorithm is faster than the

STP-relaxation in all instances except Foulds4-Foulds5. We also note that the solution to the independent flow relaxation, is as good as that obtained from the STP-relaxation in 9 instances (Adhya2-Adhya3, Bental5, Foulds2-Foulds5, Haverly1-Haverly2).

Table 5.18 shows the results of the same experiment for large-scale instances. Columns 1-8 give the same type of information as given in Table 5.17.

Table 5.18: Comparison between the time required by our algorithm and the STP-relaxation for large-scale instances.

Instance	time(sec)	IFR	time(sec)	#It.	#Const.	time(sec)	lb
A0	0.25	-37819.45	1.14	7	261	0.57	-37412.23
A1	0.22	-31689.93	0.90	8	312	1.06	-31438.51
A2	0.23	-23902.41	1.28	8	450	0.54	-23743.36
A3	0.27	-42253.73	1.79	8	187	0.52	-42032.79
A4	0.27	-43475.58	1.92	10	651	0.95	-43396.84
A5	0.32	-28257.75	2.56	11	1398	0.76	-28257.75
A6	0.31	-42463.05	3.01	11	1637	1.50	-42463.05
A7	0.40	-44682.25	3.41	13	1943	2.17	-44682.25
A8	0.40	-30666.87	3.94	13	2115	3.55	-30666.87
A9	0.31	-21933.99	3.49	12	2185	1.62	-21933.99
B0	0.44	-45466.54	2.78	7	942	1.37	-45465.92
B1	0.72	-65528.17	5.85	13	1647	2.04	-65523.34
B2	0.93	-56537.36	10.93	16	3739	6.75	-56438.06
B3	1.36	-74050.47	13.36	16	5210	11.80	-74050.47
B4	2.31	-59469.66	16.56	17	7993	26.27	-59469.66
B5	4.66	-60696.36	28.86	20	12370	41.19	-60696.36
C0	1.10	-100125.94	22.48	14	2665	11.97	-98218.60
C1	1.75	-120669.99	33.35	12	6223	40.24	-118673.48
C2	3.48	-136398.61	75.53	21	8617	44.74	-135740.45
C3	4.32	-130315.02	65.65	18	12134	681.31	-130315.02

From the above two tables, we observe that in all the instances, the solution to the independent flow relaxation can be found in much less time than the time required by the STP-relaxation.

From Table 5.18 we observe that the solution to the independent flow relaxation, is as good as that obtained from the STP-relaxation in 9 instances (A5-A9, B3-B5, C3). We also note that our algorithm is faster than the STP-relaxation in 5 instances (A1, B4, B5, C1, C3). In the rest of the instances the STP-relaxation is faster than our algorithm. The largest time difference between the two methods is observed in the instance C3, in which our method is around ten times faster than the other method.

5.4 Observations

Experiments with heuristic methods indicate that in terms of speed, the improvement method with the initial solution obtained from the independent flow relaxation, wins over the improvement method with the initial solution obtained from the construction heuristic. In terms of the quality of solutions, the results indicate that the winner is the other way round.

In order to determine the efficiency of our heuristic methods with respect to other heuristics existing in the literature, we compare the best results obtained from our heuristics with the construction heuristic given by Alfaki and Haugland [3]. The heuristic method described in [3] was coded in GAMS modelling language, and the experiments for this method were run on a computer equipped with 3.00 GHz Intel(R) quad-core processor and 8GB RAM. The comparison between the two methods is shown in Table 5.19.

Column 1 of this table gives the instance identifier. Column 3 gives the minimum of the objective function values among Impr1 and Impr2, that were mentioned in Table 5.16 of Section 5.2.1, and column 2 gives a sum of the elapsed time (in seconds) corresponding to both the methods stated above. Column 5 gives a sum of the CPU-time (in seconds) needed by the heuristic stated in [3], when it uses each of the P and PQ-formulations. Column 6 states the minimum of the objective function values found by the same heuristic using either the P or PQ-formulation. The values in the above two columns are collected from [3]. Columns 4 and 7 state the optimality gap in percentage for our heuristic and the heuristic in [3], respectively, calculated by the formula: $|\frac{ub-lb}{lb}|.100$ %, where lb refers to the lower bound on the global minimum cost, collected from [2], and is reported in column 8. Column 9 states the interrupted BARON solution.

Table 5.19: Comparison between our heuristics, the construction heuristic in [3] and the bound from a global solver for large-scale instances.

Instance	Our heuristics			Heuristic in [3]			Int. BARON	
	time(sec)	ub	gap(%)	time(sec)	ub	gap(%)	lb	ub
A0	29.83	-30337.70	16.68	967.91	-32289.61	11.31	-36411.10	**-35812.33**
A1	23.41	-23539.35	20.94	4796.53	-23398.40	21.42	-29775.79	**-29276.56**
A2	23.76	-18917.70	17.90	529.41	-16828.37	26.97	-23042.04	**-23042.04**
A3	41.85	-36037.89	10.25	971.02	-32766.89	18.40	-40155.50	**-39446.54**
A4	42.18	**-37723.94**	10.98	1073.48	-35084.35	17.20	-42374.92	-33687.13
A5	42.06	**-24888.63**	11.90	2661.33	-20758.56	26.52	-28251.33	-24015.54
A6	45.79	**-41670.94**	1.86	33.01	-32002.94	24.63	-42460.51	-37074.67
A7	50.91	**-42034.55**	5.93	1182.29	-30958.66	30.71	-44682.25	-38074.67
A8	67.44	**-30075.17**	1.93	1643.33	-22965.36	25.11	-30666.87	-28795.26
A9	67.62	-21750.30	0.74	1726.28	-15650.97	28.57	-21912.35	**-21912.35**
B0	92.28	**-41481.88**	8.19	4432.81	-32158.57	28.82	-45179.93	-20802.12
B1	85.64	**-59568.55**	8.42	4006.95	-48944.04	24.76	-65048.03	-50055.21
B2	157.40	**-52504.93**	6.38	1851.10	-33388.34	40.47	-56083.32	-18567.64
B3	278.61	**-73469.18**	0.78	1768.17	-55964.56	24.42	-74050.47	-18327.40
B4	306.04	**-58979.56**	0.82	1611.84	-37413.97	37.09	-59469.66	-3711.84
B5	287.00	**-59733.81**	1.59	3430.97	-36333.67	40.14	-60696.36	-10653.72
C0	287.47	**-73286.24**	23.86	3862.77	-66213.11	31.21	-96256.99	-15197.45
C1	344.19	**-83980.63**	28.34	3868.22	-80219.43	31.54	-117185.23	-25196.93
C2	571.51	**-117736.60**	12.93	1800.58	-68571.44	49.29	-135228.17	-7497.52
C3	549.62	**-112167.75**	13.93	2610.69	-79446.89	39.03	-130315.02	-7163.54

The results in the above table, are considerably in the favor of our heuristics. We observe that in all the instances except A0, our heuristics give better upper bounds on the optimal solution than the construction heuristic in [3]. Observing the time, our heuristics is much faster than theirs in all the instances except A6. Our heuristics are faster because they solve only linear programs, whereas the other construction heuristic solves bi-linear programs.

We also note that in all the instances except A0, our heuristics give the smallest optimality gap. In 6 instances (A6, A8, A9 and B3-B5), our heuristics reduced the optimality gap to less than 2 %, while the gap given by the other heuristic method is above 24 % in the same instances. Also, we have been able to give better solutions than the global solver in 15 (A4-A8, B0-B5 and C0-C3) out of 20 large-scale instances and those solutions are, to the best of our knowledge, the best solutions known to date.

Experiments with relaxations indicate that our algorithm can compute lower bounds faster than the STP-relaxation, in most of the small scale instances. In large-scale instances, our algorithm is faster than the STP-

relaxation in 5 instances. It is observed from Table 5.18, that in all the instances except C3, the times required by the two methods are in the same range. In the instance C3, our algorithm is around ten times faster than the STP-relaxation. Our experiments also show that in 9 out of 20 large-scale instances, the optimal solution to the independent flow relaxation coincides with the solution to the STP-relaxation. We observe that the independent flow relaxation has the same optimal solution when it is re-solved after adding violated constraints to it. This indicates that the relaxation has multiple optima. Hence, the time spent by algorithm in iterative re-solves is not productive.

Chapter 6

Conclusion and future work

6.1 Conclusion

In the past, various inexact and exact solution techniques have been proposed for the pooling problem. Solving large pooling problem instances at a reasonable computation cost has been a challenge, because of the bi-linear structure of the problem. Most of the inexact solution methods that focus on large pooling problem instances, depend upon good starting solutions, and therefore they can be trapped easily at weak solutions. In this thesis, our goal was to develop solution techniques that can compute optimal or near optimal solutions fast for large pooling problem instances. We have implemented heuristic methods to achieve this.

First, we proposed an improvement heuristic which starts with the solution to the independent flow relaxation. The relaxation is obtained by eliminating all the constraints involving the bi-linear terms. Then, either the source or the terminal proportions are estimated and fixed, and the sub-problem is solved again with the additional constraints. The procedure is repeated by alternations between fixed source and terminal proportions until there is no significant changes in the flow. Experimental results indicate that even in large instances, this heuristic is very fast and it gives fairly good upper bounds on the global minimum cost.

Next, in order to provide good initial solutions for the improvement heuristic, we developed a basic construction heuristic. This heuristic considers a sequence of sub-graphs, each of which consists of a single terminal, and an associated linear program for optimizing the flow to the terminal. The terminal is selected based on a ranking system. The optimal solution to each linear program serves as a feasible augmentation of total flow accumulated so far. In the basic construction method, we blocked the pools that send flow to a terminal, in order to avoid deterioration of quality at the remaining terminals. As this technique is unnecessarily strict, we proposed an improved version of it called the construction heuristic in which the pools are reused, and we add a constraint which ensures that the quality at the terminals does not deteriorate as a result of it. Also, the terminals that received zero flow during the ranking process are eliminated since they do not contribute to the profit. Experimental results suggest that the latter method improves the solution provided by the former one.

In order to further improve the solutions, we merged the improvement and the construction heuristic. We used the solution provided by the construction heuristic as the starting solution for the improvement heuristic. Experimental results on 20 large-scale instances indicate that, this method outperformed other methods with respect to the quality of the solutions. As it needed additional time to construct the solutions, this method was observed to be the slowest.

Overall, experimental results on 20 large scale pooling problem instances indicate that, our heuristic methods outperformed other heuristic methods that have been proposed earlier.

Finally, we have proposed an algorithm to compute the lower bound for the pooling problem fast. This algorithm starts with the solution to the independent flow relaxation. This starting solution is a lower bound on the optimal objective function value and it may be weak. In order to improve this bound, we add valid constraints to the relaxation and solve it again. The procedure is repeated until no constraints are violated. The speed of this algorithm is compared to the speed of a relaxation method from the literature. The two methods converge to the same solution as they involve the same constraints. Experimental results indicate that our algorithm is fast for most of the small pooling problem instances. In large-scale instances, it was observed to be faster than the other method in five instances. Except one large instance, the time required by the two methods was in the same range. It was also observed that the independent flow relaxation has multiple optima. Therefore, in some instances, the algorithm re-solves the relaxation iteratively until none of the constraints are violated, even though there is no change in the cost.

6.2 Future work

Possible directions for future work would be to investigate other heuristics similar to the ones proposed by us. Some modifications can be done to the improvement heuristic. In this method, the proportion variables are fixed and the algorithm can possibly be trapped in a solution close to the initial one. As an alternative to fixing the proportions, deviations from the last observed proportions can be penalized and the penalties can be increased gradually. This would offer more freedom to search for good solutions in early iterations than what is the case when proportions are fixed.

Another direction for future work would be to further improve the construction heuristic proposed by us, such that it can find even better feasible solutions.

In Algorithm 4 for fast computation of lower bounds, we observed that the algorithm re-solves the independent flow relaxation iteratively after adding violated constraints to it, even when there is no change in the cost. An improvement to this algorithm would be to modify the stopping criteria such that it terminates if the cost is unchanged after a fixed number of iterations.

A possible extension of this work can also be the implementation of the methods proposed in this thesis for the generalized version of the pooling problem in which the network consists of links between pools as well, and has applications in refineries, chemical plants and water treatment facilities.

It would also be interesting to note the performance of the algorithms in terms of speed if they are implemented in a platform other than GAMS such as C++ or Java.

Bibliography

[1] Adhya, N., Tawarmalani, M., and Sahinidis, N.V. (1999). A Lagrangian approach to the pooling problem. *Industrial & Engineering Chemistry Research*, **38** (5), 1956-1972.

[2] Alfaki, M. and Haugland, D. (2013). Strong formulations for the pooling problem. *Journal of Global Optimization*, **56** (3), 897-916.

[3] Alfaki, M. and Haugland, D. (2013). A cost minimization heuristic for the pooling problem. *Annals of Operations Research*, 1-15.

[4] Alfaki, M., and Haugland, D. (2011). Comparison of discrete and continuous models for the pooling problem. In A. Caprara and S. Kontogiannis (Eds.), *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems* (Vol. 20, pp. 112-121). OASICS. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Germany.

[5] Alfaki, M. (2012). Models and Solution Methods for the Pooling Problem (Doctoral dissertation, The University of Bergen).

[6] Almutairi, H., and Elhedhli, S. (2009). A new Lagrangian approach to the pooling problem. *Journal of Global Optimization*, **45** (2), 237-257.

[7] Audet, C., Brimberg, J., Hansen, P., Le Digabel, S., and Mladenovic, N. (2004). Pooling problem: Alternate formulations and solution methods. *Management Science*, **50** (6), 761-776.

[8] Baker, T.E., and Lasdon, L.S. (1985). Successive linear programming at Exxon. *Management Science*, **31** (3), 264-274.

[9] Benders, J.F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, **4** (1), 238-252.

[10] Ben-Tal, A., Eiger, G., and Gershovitz, V. (1994). Global minimization by reducing the duality gap. *Mathematical Programming*, **63** (2), 193-212.

[11] Faria, D.C., and Bagajewicz, M.J. (2008). A new approach for the design of multi-component water/wastewater networks. *Computer Aided Chemical Engineering*, **25**, 43-48.

[12] Floudas, C.A., and Aggarwal, A. (1990). A decomposition strategy for global optimization search in the pooling problem. *Operations Research Journal On Computing*, **2** (3), 225-235.

[13] Foulds, L.R., Haugland, D., and Jörnsten, K. (1992). A bilinear approach to the pooling problem. *Optimization*, **24** (1), 165-180.

[14] Frimannslund, L., Gundersen, G., and Haugland, D. (2008). Sensitivity analysis applied to the pooling problem. Technical Report 380, University of Bergen.

[15] Geoffrion, A.M. (1972). Generalized benders decomposition. *Journal of Optimization Theory and Applications*, **10** (4), 237-260.

[16] Gounaris, C.E., Misener, R., and Floudas, C.A. (2009). Computational comparison of piecewise-linear relaxation for pooling problems. *Industrial & Engineering Chemistry Research*, **48** (12), 5742-5766.

[17] Greenberg H.J. (1995). Analysing the Pooling Problem. *ORSA Journal of Computing*, **7** (2), 205-217.

[18] Griffith, R.E., and Stewart, R.A. (1961). A nonlinear programming technique for the optimization of continuous processing systems. *Management Science*, **7** (4), 379-392.

[19] Gupte, A., Ahmed, S., Dey, S.S., and Cheon, M.S. (2013). Pooling problems: relaxations and discretizations. School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA. and ExxonMobil Research and Engineering Company, Annandale, NJ.

[20] Haverly, C.A. (1978). Studies of the behavior of recursion for the pooling problem. *ACM SIGMAP Bulletin*, **25**, 19-28.

[21] Kallrath, J. (2000). Mixed integer optimization in the chemical process industry: Experience, potential and future perspectives. *Chemical Engineering Research and Design*, **78** (6), 809-822.

[22] Lasdon, L.S., Waren, A.D., Sarkar, S., and Palacios, F. (1979). Solving the pooling problem using generalized reduced gradient and successive linear programming algorithms. *ACM Sigmap Bulletin*, **27**, 9-15.

[23] Liberti, L., and Pantelides, C.C. (2006). An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *Journal of Global Optimization,* **36** (2), 161-189.

[24] Main, R.A. (1993). Large recursion models: Practical aspects of recursion techniques. In T. Ciriani and R. Leachman (Eds.), *Optimization in Industry: Mathematical Programming and Modeling Techniques in Practice* (pp. 241-249). New York, USA: John Wiley & Sons Ltd.

[25] McCormick, G.P. (1976). Computability of global solutions to factorable nonconvex programs: part I - convex underestimating problems. *Mathematical Programming*, **10** (1), 147-175.

[26] Misener, R. (2013). Novel global optimization methods: Theoretical and computational studies on pooling problems with environmental constraints. (Doctoral dissertation, Princeton University).

[27] Palacios-Gomez, F., Lasdon, L.S., and Engquist, M. (1982). Nonlinear optimization by successive linear programming. *Management Science*, **28** (10), 1106-1120.

[28] Pham, V. (2007). A Global Optimization Approach to Pooling Problems in Refineries. (Masters thesis, Department of Chemical Engineering, Texas A&M University, Texas, USA).

[29] Pham, V., Laird, C., and El-Halwagi, M. (2009). Convex hull discretization approach to the global optimization of pooling problems. *Industrial & Engineering Chemistry Research*, **48** (4), 1973-1979.

[30] Rømo, F., Tomasgard, A., Hellemo, L., Fodstad, M., Eidesen, B.H., and Pedersen, B. (2009). Optimizing the Norwegian natural gas production and transport. *Interfaces*, **39** (1), 46-56.

[31] Sherali, H.D., and Adams, W.P. (1998). *A reformulation-linearization technique for solving discrete and continuous non-convex problems. Nonconvex Optimization and its Applications.* **31**, Kluwer Academic Publishers.

[32] Tawarmalani, M., and Sahinidis, N.V. (2002). *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

[33] Tomasgard, A., Rømo, F., Fodstad, M., and Midthun, K. (2007). Optimization models for the natural gas value chain. In G. Hasle, K. Lie and E. Quak (Eds.), *Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF* (pp. 521-558). Springer.

[34] Visweswaran, V., and Floudas, C.A. (1990). A global optimization algorithm (GOP) for certain classes of nonconvex NLPs-II. Application of theory and test problems. *Computers & chemical engineering*, **14** (12), 1419-1434.

[35] Visweswaran, V. (2009). MINLP: Applications in blending and pooling problems. *Encyclopedia of Optimization*, Springer US, 2114-2121.