Applications of mathematical network theory

Habilitationsschrift

zur
Erlangung des akademischen Grades
doctor rerum naturalium habilitatus (Dr. rer. nat. habil.)
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Rostock

vorgelegt von
Thomas Kalinowski, geb. am 27.11.1980 in Pasewalk,
aus Rostock

Rostock, 8. Mai 2013

Gutachter

Prof. Dr. Matthias Ehrgott
University of Auckland, Faculty of Engineering

Prof. Dr. Konrad Engel
Universität Rostock, Institut für Mathematik

Prof. Dr. Horst W. Hamacher
Universität Kaiserslautern, Fachbereich Mathematik

Datum der Verteidigung: 7. Juli 2014 for 传梅

Acknowledgements

The papers in this thesis have been written at the Institute for Mathematics of the University of Rostock from 2007 to 2010, and at the School of Mathematical & Physical Sciences of the University of Newcastle from 2010 to 2012. I thank all of my colleagues at these institutions for their support and assistance, and for making me feel at home in both places. In particular, I would like to express my gratitude to my academic teacher Konrad Engel for the continuous support and for the reliable guidance with respect to every aspect of my academic career over the last 15 years.

I thank all of my coauthors for the stimulating collaborations from which I learned a lot. I am very grateful to Natashia Boland and the OR group in Newcastle for making me feel welcome in Australia, and for making my time there highly enjoyable. In particular, I thank Hamish Waterer for teaching me many things about applied mathematics and about Newcastle's pub culture. Special thanks go to Lanbo Zheng for being really good company in the office and at other places.

A significant part of the work in this thesis was done in a project with the Hunter Valley Coal Chain Coordinator (HVCCC) P/L, and this research benefited a lot from the valuable contributions of Jonathon Vandervoort, Rob Oyston, Tracey Giles, and the Capacity Planning Team at HVCCC. I really appreciate their patience, support, and feedback.

I also thank Nina Narodytska, Uwe Leck and Ian Roberts for discussions on mathematics completely unrelated to this thesis, but essential for keeping my mind in good working condition.

Finally, I am deeply indebted to my family, and in particular to my parents, for always supporting me.

Contents

Introduction to this thesis

This thesis is about applications of the mathematical theory of networks to practical optimization problems. It is a cumulative thesis, and its main part consists of the papers [1–8] which are already published or submitted for publication. In this introductory part of the thesis it will be explained how the results in the attached papers are related. To this end we provide an overview of some important results of network theory and explain how they are used in the papers. We will also introduce the application areas that are covered by the papers: optimal treatment planning in intensity modulated radiation therapy in [5–8], and strategic and operational planning for supply chain networks in [1–4].

I would like to thank all of my coauthors for agreeing on including our joint papers. In the following list I outline my own contributions to these papers in the order in which they appear in this thesis.

- The method proposed in [8] is based on a combination of [79] and [54]. I had the idea to apply the methods from [79] to the approximation problem introduced in [54], and the details were worked out in discussions with Antje Kiesel.

- For the paper [4] I proved the NP-hardness of the problem (Proposition 1), and I formulated preliminary versions of the heuristic algorithms described in Section 3 of that paper, which were then refined as a result of discussions. For the experimental part I did the runs with the commercial solver CPLEX that were used as a benchmark for our different heuristics, and I also prepared the statistical analysis of the results, i.e. the tables and diagrams in Section 4.

- For the paper [3] I made the observation that certain logical conditions for the availability of machines can be described by a network flow model, which is essential for the mixed integer programming (MIP) approach presented in this paper. I formulated an initial MIP model which was then refined in discussions with the coauthors and industry partners. I implemented the model and analyzed the test results, where again the experimental setup was changed several times after discussions with the coauthors.

- For the paper [2] I did the analysis for the unit capacity case (Section 3) and the special case of series-parallel networks (Proposition 5). For the remaining results in Section 2 I was contributing to the discussions that led to the results and I simplified the proofs and prepared the final presentation.

- For the paper [1] I did the analysis of the two greedy heuristics (Section 6), and I developed and analyzed an initial version of the approximation algorithm in Section 7 which was later simplified significantly by the coauthors.

1 Preliminaries on network problems

In this section, we collect some important results on network optimization problems related to paths and flows and we indicate how these are used in the attached papers. Our presentation follows Part I of Schrijver's book [113]. Many more details can be found there and in other standard references on network flows such as the books of Ford and Fulkerson [68] and Ahuja, Magnanti and Orlin [10].

A *network* (or *directed graph*) $N = (V, A)$ is a pair consisting of a finite set V whose elements are called *nodes* and a finite set A whose elements are called arcs. Any arc a is associated with an ordered pair $(u, v) \in V \times V$, and the nodes u and v are also denoted by a^- and a^+, respectively. Throughout the cardinalities of V and A will be denoted by n and m, respectively. We denote the set of arcs going into a note v by $\delta^{\text{in}}(v)$ and the set of arcs leaving v by $\delta^{\text{out}}(v)$, or formally,

$$\delta^{\text{in}}(v) = \{a \in A \ :\ a^+ = v\}, \qquad \delta^{\text{out}}(v) = \{a \in A \ :\ a^- = v\}.$$

A *walk* in a network is a sequence $(v_0, a_1, v_1, a_2, v_2 \dots, a_k, v_k)$ alternating between nodes $v_0, \dots, v_k$ and arcs $a_1, \dots, a_k$ such that $a_i^- = v_{i-1}$ and $a_i^+ = v_i$ for $i = 1, \dots, k$. A walk with distinct nodes is called a *path* from v_0 to v_k (or a v_0-v_k-path), and a walk with distinct nodes v_i $(i = 0, 1, \dots, k-1)$ and $v_k = v_0$ is called a *cycle*.

1.1 The shortest path problem

Let $l : A \to \mathbb{R}$ be a given function which we call *length*. This function extends to the power set of A by $l(X) = \sum_{a \in X} l(a)$ for $X \subseteq A$. In particular, identifying the walk $P = (v_0, a_1, v_1, \dots, a_k, v_k)$ with the set of arcs $\{a_1, a_2, \dots, a_k\}$ we can speak of the length $l(P)$ of this walk. The basic version of the shortest path problem is the following: Given a network (V, A), a length function $l : A \to \mathbb{R}$ and two nodes $s, t \in V$, find a path that minimizes the length over all paths from s to t. The idea underlying the classical algorithms for this problem, the Dijkstra method and the Bellman-Ford method, is to start with a labeling of the nodes given by $d(s) = 0$ and $d(v) = \infty$ for all nodes $v \neq s$, and then iteratively pick an arc a with $a^- = u$, $a^+ = v$ and $d(v) > d(u) + l(a)$ and relabel $d(v) \leftarrow d(u) + l(a)$. Provided there are no cycles of negative length, this procedure terminates with every node v labeled with the minimum length of a path from s to v. How efficiently this general framework can be implemented depends on properties of the length function.

Nonnegative arc lengths. The original algorithm of Dijkstra [46] solves the problem in time $O(n^2)$, while in more efficient implementations using sophisticated data structures the run-time can be reduced: Using Fibonacci heaps the problem can be solved in time $O(m + n \log n)$ [69], and using scaling techniques it can be done in time $O(m + n\sqrt{\log L})$ [11] for integer arc lengths where $L = \max\{l(a) \ :\ a \in A\}$.

Networks without cycles of negative length. The Bellmann-Ford algorithm, proposed by Bellmann [21], Ford [64] and Moore [100], runs in time $O(mn)$, and using scaling techniques developed by Goldberg [71] the problem can be solved in time $O(\sqrt{n}m\log L')$ for integer arc lengths where $L' = \max\{2, \max\{-l(a) \ : \ a \in A\}\}$.

Arbitrary lengths. If cycles of negative length are allowed the problem is already NP-complete even when all arcs have length -1.

If we are not only interested in the shortest path between two special nodes s and t but in shortest paths between all pairs of nodes, we can run the Bellman-Ford algorithm n times to obtain a run-time of $O(n^2m)$. A more efficient method, at least for dense networks, is the Floyd-Warshall algorithm [63, 127] which runs in time $O(n^3)$. For sparse networks, the faster methods of Johnson [78] and Bazaraa and Langley [20] can be combined with the Fibonacci heap implementation of Dijkstra's algorithm due to Fredman and Tarjan [69] to solve the all-pairs shortest path problem in time $O(n(m + n\log n))$.

Building on the ideas of [79], shortest paths are used in the papers [5] and [8] to design efficient algorithms for the optimization of treatment plans in intensity modulated radiation therapy using multileaf collimators. This is described in more detail in Section 2.

The paper [1] deals with a variant of the shortest path problem as the first case of a new class of network optimization problems which is introduced in this work: *incremental* network problems. The basic idea is that arcs can be added to the network over time (subject to budget constraints) and in each time period some network problem, for instance a shortest path problem, has to be solved. The shortest path problem was chosen as the first example for a detailed analysis, because it is among the simplest network optimization problems. Interestingly, the incremental version of the shortest path problem is NP-complete already in very special cases, and consequently natural greedy heuristics are analyzed and an approximation algorithm is derived. In this approximation algorithm, a modified Bellman-Ford algorithm is used to determine the minimum length of a path after adding at most k arcs. More details on incremental network design problems can be found in Section 3.3.

1.2 The maximum flow problem

Let s and t be two special nodes, called the *source* and the *sink* respectively. A *flow* is a function $f : A \to \mathbb{R}_{\geqslant 0}$ that satisfies the *flow conservation constraints*

$$\sum_{a\in\delta^{\text{in}}(v)} f(a) = \sum_{a\in\delta^{\text{out}}(v)} f(a) \qquad \text{for all } v \in V \setminus \{s,t\}.$$

Extending f to the power set of A by $f(X) = \sum_{a\in X} f(a)$ for $X \subseteq A$, the *value* of a the flow f is defined to be

$$\text{value}(f) = f(\delta^{\text{out}}(s)) - f(\delta^{\text{in}}(s))$$

which is easily seen to be equal to $f(\delta^{\text{in}}(t)) - f(\delta^{\text{out}}(t))$. Given a *capacity* function $c : A \to \mathbb{R}_{\geqslant 0}$ a flow f is called feasible if it satisfies $f(a) \leqslant c(a)$ for all $a \in A$. The maximum flow problem is to find, for a given network with capacity, a feasible flow of maximum value. An

s-t-cut in a network with source s and sink t is a partition $V = S \cup T$ of the node set into two parts such that $s \in S$ and $t \in T$. The capacity of a cut is the sum of the capacities of the arcs crossing from S to T, i.e.

$$c(S,T) = \sum_{a\in A\,:\,a^-\in S,\,a^+\in T} c(a).$$

Theorem 1 (Max-Flow Min-Cut Theorem[44, 65])**.** *The maximum value of a feasible flow in a network with respect to a capacity function c equals the minimum capacity of an s-t-cut in that network.*

The proof idea for this theorem can be turned into an algorithm for solving the maximum flow problem. The algorithm of Ford and Fulkerson [66] is based on an auxiliary network D_f, called the *residual* network, which can be associated with any feasible flow f. For an arc a with associated node pair (u,v), let the *reverse arc* $\bar{a}$ be associated with the node pair (v,u). The node set of the residual network D_f is the same as for the original network, and it has the arc set $A_f = A_f^+ \cup A_f^-$ where

$$A_f^+ = \{a:\ a\in A \text{ with } f(a) < c(a)\}, \qquad A_f^- = \{\bar{a}\ :\ a\in A \text{ with } f(a) > 0\}.$$

The algorithm starts with the feasible flow given by $f(a) = 0$ for all $a \in A$ and then iteratively finds an s-t-path P in the residual network and pushes a sufficiently small amount ε of flow along this path, which means that the flow on arcs in $A_f^+ \cap P$ is increased by ε while the flow on arcs in $A_f^- \cap P$ is reduced by ε. This algorithm also ensures that for integral capacities the resulting flow takes only integral values. The original algorithm is guaranteed to terminate only for rational capacities and has a run-time of $O(nmC)$ for integral capacities where $C = \max\{c(a)\ :\ a \in A\}$. Variants of this basic idea due to Dinic [47] and Edmonds and Karp [49] work for arbitrary capacities and have a runtime of $O(nm^2)$. The design of efficient methods for the max-flow problem has been a very active research area leading to many beautiful results on algorithms and data structures, including capacity scaling [49], dynamic trees [119] and the push-relabel method [72]. Recently, Orlin [105] solved the long-standing open problem to develop an $O(nm)$ algorithm for the max-flow problem.

Maximum flow computations are crucial for the greedy heuristic that is proposed in [4] for the annual maintenance scheduling of the Hunter Valley coal chain. This is a large supply chain in which coal is transported from the mines to the vessels in the port via a network consisting of rail track and different machines like stackers and reclaimers. Details on this maintenance scheduling problem can be found in Section 3.2.

1.3 The minimum cost flow problem

In addition to the capacity function, let there be a *cost* function $k : A \to \mathbb{R}$, and let the cost of a flow f be defined by

$$\mathrm{cost}(f) = \sum_{a\in A} k(a)f(a).$$

For a given value φ, the minimum cost flow problem is to find a flow of minimum cost subject to value$(f) = \varphi$. It is convenient to reduce this to the minimum cost circulation problem which is defined as follows. For a network with a capacity function $c : A \to \mathbb{R}_{\geqslant 0}$ and a demand function $d : A \to \mathbb{R}$, find a feasible *circulation* f of minimum cost, which is a function $f : A \to \mathbb{R}$ that satisfies $d(a) \leqslant f(a) \leqslant c(a)$ for all $a \in A$, as well as the flow conservation constraints

$$\sum_{a\in\delta^{\mathrm{in}}(v)} f(a) = \sum_{a\in\delta^{\mathrm{out}}(v)} f(a) \qquad \text{for all } v \in V.$$

The min-cost flow problem can be reduced to the min-cost circulation problem by adding an arc a with $a^- = t$, $a^+ = s$ and $d(a) = c(a) = \varphi$ and $k(a) = 0$. As for the max-flow problem the residual network with arc set A_f is useful. We can extend the cost function to the arc set A_f by putting $k(\bar{a}) = -k(a)$ for all $\bar{a} \in A_f^-$. The following observation, which is crucial for the development of algorithms solving the min-cost circulation problem, has been made several times in different contexts (cf. Chapter 12 in [113]).

Theorem 2. *A feasible circulation f has minimum cost among all feasible circulations if and only if each cycle in D_f has non-negative cost.*

This gives a method to improve a given circulation f which is not optimal: choose a negative cycle C in D_f, increase $f(a)$ by τ for arcs $a \in A_f^+ \cap C$ and decrease $f(a)$ by τ for arcs a with $\bar{a} \in A_f^- \cap C$, where τ is chosen as large as possible such that f stays a feasible circulation. Edmonds and Karp [49] use capacity-scaling to prove that for integer capacities with maximum capacity C this problem can be solved in time $O(n^4 \log C)$. The first strongly polynomial algorithm (i.e. with a run-time bound depending only on m and n) was developed by Tardos [123]. She reduces the problem to the solution of $O(m^2 \log n)$ maximum flow problems. Together with Orlin's algorithm [105] for the maximum flow problem this yields a run-time of $O(m^3 n \log n)$. The currently best run-time guarantee is due to Orlin [103, 104] who reduces it to $m \log n$ shortest path problems which implies a run-time of $O(m \log n(m + n \log n))$.

In [7], a minimum cost flow formulation is used to approximately solve a matrix decomposition problem that is again motivated by the treatment planning in intensity modulated radiation therapy as described in Section 2.

2 Intensity modulated radiation therapy using multileaf collimators

The use of high energetic radiation is an important method in cancer treatment. In order to destroy tumor cells, the patient is exposed to high energetic radiation. The linear accelerator delivering the radiation is attached to a gantry which can be rotated about the treatment couch (see Figure 1). Radiation is useful in cancer therapy because the repair mechanism of cancer cells is less efficient than that of normal cells [115]. The method of intensity modulated radiation therapy (IMRT), first proposed by Brahme [30], was developed in the

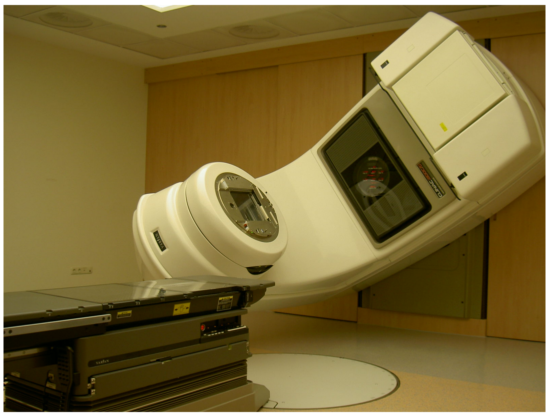

Figure 1: A linear accelerator on a gantry.

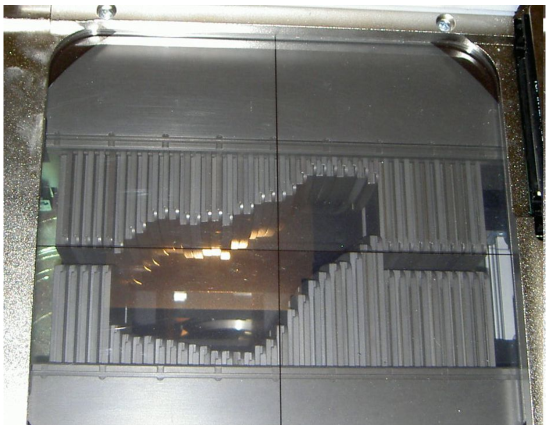

Figure 2: The leaf pairs of a multileaf collimator.

early 1990's in order to make this treatment method more efficient. Today IMRT is a widely used clinical treatment modality and a lot of research has been done on different aspects of the optimal delivery of IMRT which turned out to be a highly complex planning problem. We refer to the book of Webb [128] and the review paper of Bortfeld [28] for more details on the many facets of radiation therapy planning. On a basic level, the objectives are to ensure that the tumor receives a sufficiently high uniform dose while the damage to the normal tissue is as small as possible, and in particular the functioning of critical organs in the vicinity of the tumor is not affected. Due to the high complexity of the problem the planning process is usually divided into three phases [50]: (1) the selection of the number of beams and the directions from which radiation is delivered, (2) the selection of intensity patterns for the directions determined in the first step, and (3) the selection of an efficient way to deliver the intensity patterns. The first two steps (beam selection and intensity profile optimization) can be combined into one optimization routine, which was done, for instance, by Lee *et al.* [94], Preciado-Walters *et al.* [108] and Engel and Tabbert [55]. In *aperture-based* methods [110, 114] all three steps are done in one phase and the optimization is directly in terms of deliverable radiation fields.

Any treatment optimization starts with discretizing the patient body into so-called *voxels*. The set of voxels is then partitioned into three sets: the clinical target volume, the critical structures and the remaining tissue. There are certain dose constraints for each of these parts: the dose in the target volume has to be sufficient to kill the cancerous cells and the dose in the critical structures must not destroy the functionality of the corresponding organs. Beam directions and intensity patterns are usually determined by inverse methods based on physical models of how the radiation passes through a body [55, 76, 92, 111]. In this section, we will focus on the third step, the realization of given intensity patterns, in particular using a multileaf collimator (MLC). An MLC consists of two banks of metal leaves which block the radiation and can be shifted to form irregular shaped beams (Figure 2). With an MLC it is possible to form homogeneous fields of different shapes, and an intensity modulated field can be delivered as a superposition of homogeneous fields. After discretizing the beam into *bixels* we can assume that the required intensity pattern is given as a nonnegative integer matrix A, where each row of the matrix corresponds to a leaf pair of the MLC. There are two methods in IMRT using MLCs which differ in their technical realization, but the mathematical methods used to determine optimal treatment plans are similar. In the step–and–shoot mode the radiation is switched off whenever the leaves are moving, and the intensity modulation is the result of superimposing a finite number of homogeneous fields. In the dynamic mode [39, 85, 121, 122] the radiation is switched on during the whole treatment, and the modulation is achieved by moving the leaves with varying speed. The fluence at a particular point is proportional to the amount of time in which the point is exposed to radiation, i.e. not blocked by one of the leaves. We consider only the step–and–shoot mode, but the most common approach to the dynamic mode can be seen as an imitation of this case (see [81] and the references therein). The problem of finding optimal sequences of leaf positions was first studied by medical physicists [29, 70, 117, 131], but soon it got considerable attention in the mathematical optimization and operations research communities, which is indicated by several survey papers [51, 81, 82]. In this section some important results on this problem are presented, in particular those that are relevant to the work in the papers [5–8].

2.1 The MLC shape matrix decomposition problem

The principle of using an MLC in the step–and–shoot mode is illustrated in Figure 3. Our aim is to determine a sequence of leaf positions and corresponding irradiation times such that the given fluence distribution is realized. Suppose the given matrix has size $m \times n$, i.e. we consider m leaf pairs, and for each leaf there are $n+1$ possible positions. Then the leaf positions can be described by certain binary matrices of size $m \times n$ called *shape matrices*, where a 0-entry means the radiation is blocked and a 1-entry means that the radiation goes through. In the basic variant of the problem, the only requirement for shape

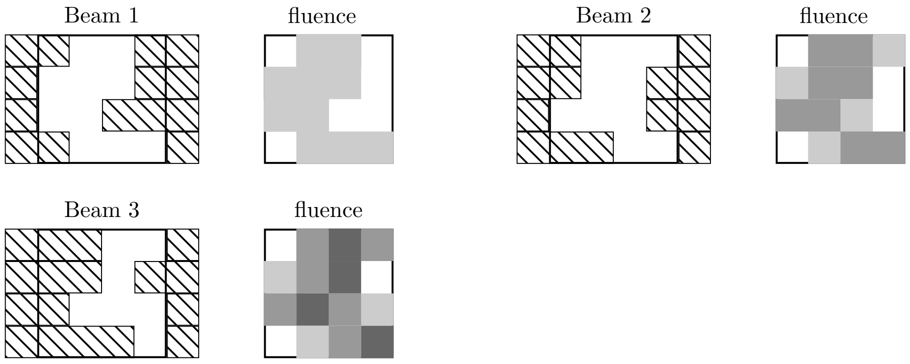

Figure 3: Intensity modulation by superimposing 3 beams of different shapes. In each step the left figure shows a leaf position and in the right figure the grey scale indicates the accumulated fluence.

matrices is that each row has the *consecutive-ones* property, i.e. there are integers l_i and r_i

for $i = 1, 2, \ldots, m$ such that

$$s_{ij} = \begin{cases} 1 & \text{if } l_i < j < r_i, \\ 0 & \text{otherwise.} \end{cases}$$

For example the first leaf position in Figure 3 corresponds to the shape matrix

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

with leaf positions $(l_1, r_1) = (1, 4)$, $(l_2, r_2) = (0, 4)$, $(l_3, r_3) = (0, 3)$ and $(l_4, r_4) = (1, 5)$. The superposition of differently shaped beams corresponds to a positive linear combination of shape matrices, where the coefficient of a shape matrix measures how long the corresponding field is applied. So any representation of the given fluence matrix A as a positive integer linear combination of shape matrices is a feasible solution for our decomposition problem. For instance:

$$A = \begin{pmatrix} 1 & 3 & 3 & 0 \\ 0 & 2 & 4 & 1 \\ 1 & 1 & 4 & 4 \\ 3 & 3 & 1 & 0 \end{pmatrix} = 2 \cdot \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}. \qquad (1)$$

We denote the set of shape matrices by $\mathcal{S}$, and consider decompositions of the form $A = \sum_{S \in \mathcal{S}} u_S S$ with $u_S \in \mathbb{N}$ for all $S \in \mathcal{S}$. There are two major objectives that have been considered in the literature: the *total beam-on time* and the *total setup time*. The former one is proportional to the sum of the coefficients in the decomposition, and the second one can be measured by the number of beams, in other words the number of shape matrices used in the decomposition. Let $\mathcal{S}_0$ denote the set of matrices with nonzero coefficient. We follow the terminology introduced by Baatar *et al.* [18] and formulate two optimization problems, the decomposition time (DT) problem and the decomposition cardinality (DC) problem

$$\textbf{(DT)} \qquad \min \left\{ \sum_{S \in \mathcal{S}} u_S \ :\ A = \sum_{S \in \mathcal{S}} u_S S,\ u_S \in \mathbb{N} \right\}, \qquad (2)$$

$$\textbf{(DC)} \qquad \min \left\{ |\mathcal{S}_0| \ :\ \mathcal{S}_0 \subseteq \mathcal{S},\ A = \sum_{S \in \mathcal{S}_0} u_S S,\ u_S \in \mathbb{N} \right\}. \qquad (3)$$

More generally, the minimization of a weighted sum of decomposition time and decomposition cardinality leads to the objective function

$$\sum_{S \in \mathcal{S}_0} u_S + \alpha |\mathcal{S}_0|, \qquad (4)$$

for a nonnegative constant α. This objective function can be interpreted as the total treatment time, where the parameter α depends on the used MLC and measures the average setup time, i.e. the time needed to move the leaves and check the setting. In a still more refined model, the varying setup time between different leaf positions can be taken into account depending on the amount of required leaf motion [42, 77]. Consequently, the order in which the beams are delivered becomes relevant and the corresponding objective function is

$$\sum_{S\in\mathcal{S}_0} u_S + \sum_{i=1}^{|\mathcal{S}_0|-1} \mu(S^{(i)}, S^{(i+1)}), \tag{5}$$

where $S^{(1)}, S^{(2)}, \ldots, S^{(|\mathcal{S}_0|)}$ is a permutation of the set of used shape matrices $\mathcal{S}_0$, and for two shape matrices S and S', $\mu(S, S')$ is proportional to the time necessary to change the setup of the MLC from the beam corresponding to S to the beam corresponding to S'. It can be shown by small examples [18, 51, 81] (matrices of size 2×3) that the objective functions (2) to (5) in general yield different optimal solutions. It has been observed that the objective values can be significantly improved if it is allowed to rotate the collimator by $90°$, thus interchanging the roles played by rows and columns [36, 48, 80].

The optimal value of (2) can be computed efficiently while the problem (3) is strongly NP-complete. A common approach is to first compute the minimal DT, and then heuristically search for a decomposition which realizes this DT and also has a small DC.

2.2 The decomposition time problem

Starting with Bortfeld *et al.* [29] and Galvin *et al.* [70], a number of algorithms for the shape matrix decomposition problem have been proposed [43, 117, 131], some of them providing the optimal *DT* while others use heuristic methods for both objectives *DT* and *DC*. A more rigorous treatment of the problem, including mathematical optimality proofs, started a few years later [9, 17, 18, 25, 52, 79, 84]. First we consider the version without additional constraints, i.e. the leaves in different rows move independently, and then we discuss in some detail two common constraints that are relevant in this thesis: the *interleaf collision constraint* (ICC) and the *tongue-and-groove constraint* (TGC).

The unconstrained problem

A fundamental insight that was already the basis of the algorithm of Bortfeld *et al.* [29] is that in the unconstrained case the decomposition problem can be solved for each row independently, and the optimal *DT* for the whole matrix is the maximum of the optimal *DT*s of the single rows. The optimality (in terms of *DT*) of both the sweep algorithm from [29] and the geometry based algorithm of Siochi [117] have been proved several times: based on network flow formulations by Ahuja and Hamacher [9], based on analyzing the leaf trajectories inspired by dynamic MLC models by Kamath *et al.* [84] and based on an explicit characterization of the minimal *DT* in terms of the matrix entries by Engel [52]. The best run-time bound for the unconstrained shape matrix decomposition problem comes from the network flow argument of Ahuja and Hamacher [9].

Theorem 3 ([9])**.** *The shape matrix decomposition problem for an $m \times n$ intensity matrix A can be solved in time $O(mn)$.*

We describe Engel's characterization [52] of the optimal *DT* in more detail because it motivates an approach to the *DC* problem described below, and in addition it can be seen as an early version of the concepts underlying the papers [5, 7, 8]. For simplicity of notation we add a 0-th and an $(n+1)$-th column to the matrix A by setting $a_{i,0} = a_{i,n+1} = 0$ for $i = 1, 2, \dots, m$. The *complexity* of the matrix A is defined by $c(A) = \max\limits_{1 \leqslant i \leqslant m} c_i(A)$ where the *i-th row complexity* $c_i(A)$ is

$$c_i(A) = \sum_{j=1}^{n} \max\{0, a_{ij} - a_{i,j-1}\}.$$

Theorem 4 ([52]). *The minimal DT for a matrix A equals $c(A)$.*

The interleaf collision constraint (ICC)

In some of the commercially available MLCs leaf overtravel is forbidden. That means the left leaf of row i and the right leaf of row $i \pm 1$ must not overlap. In this case, a shape matrix cannot contain two consecutive rows as follows:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

In terms of the left and right leaf positions this adds the constraints $l_i < r_{i+1}$ and $r_i > l_{i+1}$ for $i = 1, \dots, m-1$. Boland *et al.* [25] reduce the DT problem with ICC to a network flow problem with side constraints. The network has $O(mn^2)$ nodes arranged in m layers, and the nodes in layer i are labeled by (i, l, r) where (l, r) runs through the possible pairs of left and right leaf positions, i.e. all pairs with $0 \leqslant l < r \leqslant n+1$. The arcs are between consecutive layers and represent the combinations of leaf positions that are allowed by the ICC. In other words, the nodes (i, l, r) and $(i+1, l', r')$ are connected by an arc if and only if $l < r'$ and $r > l'$. In addition, there is a source D, which is connected to all nodes in the first layer, and a sink D' to which all nodes in the last layer are connected. Then shape matrices correspond to paths from D to D'. Figure 4 shows the node set for the case $m = 4$, $n = 2$ and two paths corresponding to the shape matrices

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

By identifying shape matrices with paths from D to D' a shape matrix decomposition can be interpreted as a D-D'-flow, where the flow value is the DT. This is the basis of the polynomial algorithm for the DT problem proposed by Boland *et al.* [25].

Theorem 5 ([25]). *The DT problem for a matrix A is equivalent to finding a D-D'-flow of minimum value subject to the constraint that for every $(i,j) \in [m] \times [n]$ the total flow through the nodes (i,l,r) with $l < j < r$ equals a_{ij}. In particular the problem can be solved in polynomial time.*

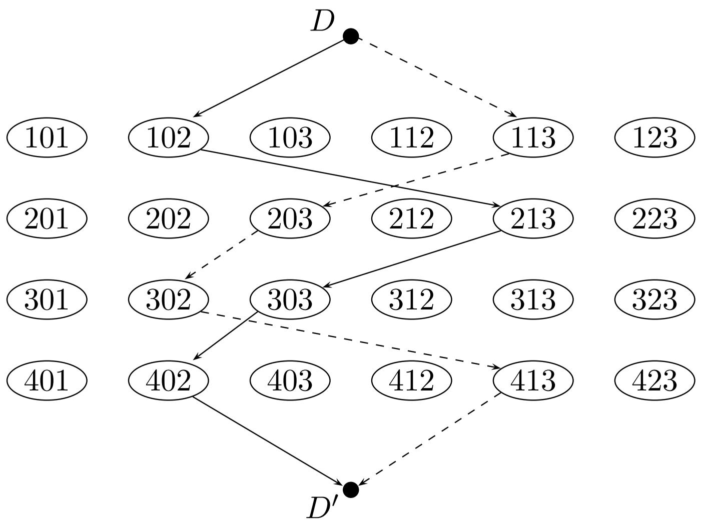

Figure 4: The shape matrix graph for $m = 4$ and $n = 2$ with two paths corresponding to shape matrices.

In [5], a variant of the graph in Figure 4 is used in a heuristic approach to minimize the DC in a shape matrix decomposition, taking into account both the ICC and the tongue-and-groove constraint described below. The algorithms proposed by Baatar *et al.* [18] and Kamath *et al.* [84] are based on the observation that $\max\{0, a_{i,j+1} - a_{ij}\}$ is a lower bound for the time for which the left leaf has to be in position j, i.e. for the sum of the coefficients of shape matrices with $s_{ij} = 0$ and $s_{i,j+1} = 1$, and similarly for right leaf positions. Introducing variables for the amounts by which these lower bounds are exceeded, Baatar *et al.* [18] formulate a linear program and develop a combinatorial algorithm which implies that the ICC increase the run-time by a factor of m.

Theorem 6 ([18])**.** *The DT problem with ICC can be solved in time $O(m^2n)$.*

Kamath *et al.* [84] describe a similar algorithm but with an optimality proof that works only under the additional assumption that the MLC leaves move only from left to right. An alternative approach to the decomposition time problem with ICC can be found in [79]. Generalizing Theorem 4, the minimal *DT* of a shape matrix decomposition with ICC can be characterized as the maximal length of an s-t-path in the directed graph $G = (V, A)$ with node set V and arc set A defined as follows:

$$\begin{aligned}
V &= \{s,t\} \cup [m] \times [0, n+1], \\
A &= \{(s,(i,0)) \ : \ i \in [m]\} \cup \{((i,n+1),t) \ : \ i \in [m]\} \\
&\quad \cup \{((i,j),(i,j+1)) \ : \ i \in [m],\ j \in [0,n]\} \\
&\quad \cup \{((i,j),(i+1,j)) \ : \ i \in [m-1],\ j \in [n]\} \\
&\quad \cup \{((i,j),(i-1,j)) \ : \ i \in [2,m],\ j \in [n]\}\,.
\end{aligned}$$

The length function $l : A \to \mathbb{N}$ is defined by

$$\begin{aligned}
l(s,(i,0)) = l((i,n+1),t) &= 0 && (i \in [m]),\\
l((i,j-1),(i,j)) &= \max\{0, a_{ij} - a_{i,j-1}\} && (i \in [m],\ j \in [n+1]),\\
l((i,j),(i+1,j)) &= -a_{ij} && (i \in [m-1],\ j \in [n]),\\
l((i,j),(i-1,j)) &= -a_{ij} && (i \in [2,m],\ j \in [n]).
\end{aligned}$$

This graph is called the *DT-ICC-graph* for the matrix A. Figure 5 shows the DT-ICC-graph

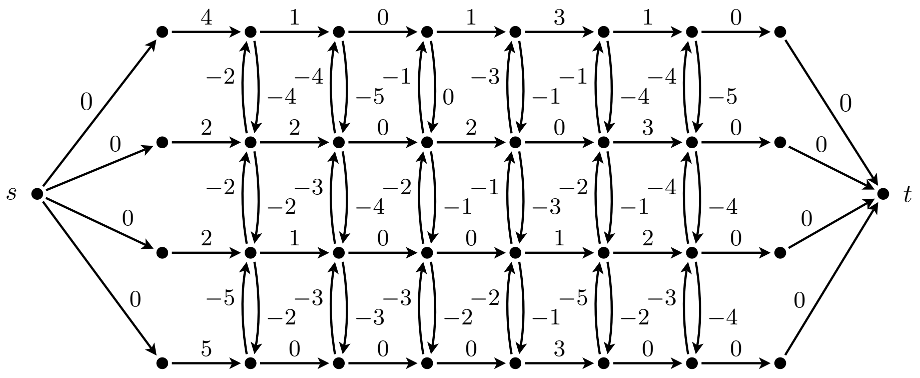

Figure 5: A DT-ICC-Graph.

for the matrix

$$A = \begin{pmatrix} 4 & 5 & 0 & 1 & 4 & 5\\ 2 & 4 & 1 & 3 & 1 & 4\\ 2 & 3 & 2 & 1 & 2 & 4\\ 5 & 3 & 3 & 2 & 5 & 3 \end{pmatrix}.$$

Using duality the following min-max-characterization can be proved.

Theorem 7 ([79]). *The minimal DT of a shape matrix decomposition of the matrix A equals the maximum length of an s-t-path in the corresponding ICC shape matrix decomposition graph.*

This characterization of the minimal *DT* as a maximal path length is the basis of the approach to the approximated shape matrix decomposition problem proposed in [8].

The tongue-and-groove constraint (TGC)

Another feature of many MLCs is the tongue–and–groove design [45, 129]. To prevent leaking radiation through the gap between two adjacent leaves a tongue-and-groove design is used which creates a small overlap of the regions covered by adjacent leaves. This overlap can cause significant under-dosage effects, and many researchers have proposed mechanisms to control these under-dosage effects [86, 97, 109, 117]. A natural strategy is to require that

$a_{ij} \leqslant a_{i\pm1,j}$ implies that in each of the used beams bixel $(i \pm 1, j)$ is open whenever bixel (i, j) is open, or in terms of the shape matrices:

$$\begin{aligned} a_{ij} \leqslant a_{i-1,j} \wedge s_{ij} = 1 &\quad \Longrightarrow \quad s_{i-1,j} = 1, \\ a_{ij} \geqslant a_{i-1,j} \wedge s_{i-1,j} = 1 &\quad \Longrightarrow \quad s_{ij} = 1 \end{aligned}$$

for $i = 2, \dots, m$ and $j = 1, \dots, n$. These conditions will be called *tongue-and-groove constraints* (TGC) and they ensure that the overlap region between two adjacent bixels receives the smaller of the two relevant doses. The method of Kamath *et al.* [84] can be modified to solve the DT problem with TGC. Imitating the synchronization approach proposed by van Santvoort and Heijmen [112] for MLCs in dynamic mode, this was done by Kamath *et al.* [86] under the additional assumption of unidirectional leaf movement.

Theorem 8 ([86]). *Assuming unidirectional leaf movement, shape matrix decompositions with minimum DT can be determined in time $O(m^2 n)$ for the problem with TGC as well as for the problem with both constraints, ICC and TGC.*

This theorem can be proved in the same way as Theorem 6. For the problem with ICC and TGC the assumption of unidirectional leaf movement can be dropped: using the duality based algorithm from [79] with a modified length function yields a decomposition with unidirectional leaf movement which has minimum *DT* among all shape matrix decompositions with ICC and TGC [5]. The complexity of the DT problem with TGC (but without ICC) is still open.

Problem. Determine the computational complexity of the DT problem with TGC.

Note that Engelbeen and Kiesel [57] showed that for a binary input matrix A the DT problem with TGC can be solved in time $O(m^2 n^2)$.

2.3 The decomposition cardinality problem

In contrast to the DT problem, the DC problem is already computationally hard for very special cases as shown by Baatar *et al.* [18] and Collins *et al.* [38].

Theorem 9 ([18]). *The DC problem (3) is strongly NP-hard for single row matrices.*

Theorem 10 ([38]). *The DC problem (3) is strongly NP-hard for single column matrices.*

The proofs of these two hardness results proceed by reduction from 3-Partition and Positive NotAllEqual-3Sat, respectively. Using the reduction from 3-Partition, Bansal *et al.* [19] prove that the DC problem cannot be well approximated. More precisely, it is APX-hard, which means that there exists a positive ε such that, unless $P = NP$, there is no polynomial time algorithm with an approximation ratio $1 + \varepsilon$.

Theorem 11 ([19]). *The DC problem is APX-hard even for a single row with entries polynomially bounded in n.*

The approaches that have been proposed to overcome this hardness can be roughly classified into three types.

Heuristics. Most of the early algorithms heuristically search for a decomposition with a small DC among all decompositions with the minimum DT. This can be seen as lexicographic minimization for the bi-objective problem to minimize the pair (DT, DC). For instance, a greedy strategy derived from the characterization of the minimal DT in Theorem 4 is used by Engel [52]: While the intensity matrix A is nonzero, find the maximal integer u such that there exists a shape matrix S with $c(A - uS) = c(A) - u$ and $A - uS$ still nonnegative, and continue with $A - uS$. The same strategy is applied in [79] for the decomposition with ICC, and in [5] for ICC and TGC. A similar greedy idea is used by Baatar *et al.* [18] for the decomposition with ICC. The algorithm of Baatar *et al.* [16] takes this approach a step further: instead of extracting a single shape matrix, each iteration step consists of the extraction of a set of shape matrices with the maximum coefficient. Gunawardena *et al.* [73] describe a multiple start local search heuristic using results of Engel [52]. An approach using ideas from computational geometry is developed by Chen *et al.* [33–35]. This method builds on the geometric interpretation of intensity maps as a 3-D "mountain" made of unit cubes which goes back to Siochi [117]: The base of the mountain representing an $m \times n$-matrix A is an $m \times n$-rectangle and on the square (i, j) there is a pile of a_{ij} unit cubes. Then the DC problem corresponds to the decomposition of such a mountain into polytopes build from unit cubes which represent the fields that can be delivered by the MLC.

Approximation algorithms. There have been some attempts to obtain algorithms with a guaranteed approximation ratio. An algorithm of Bansal *et al.* [19] is shown to have approximation ratio 24/13 for the single row case. Building on this, approximation algorithms for general matrices were developed by Luan *et al.* [96] and extended by Biedl *et al.* [23]. The approximation guarantee obtained is in terms of the maximum value L and the maximum row difference D which are defined as (recall $a_{i0} = a_{i,n+1} = 0$ for all i)

$$\begin{aligned} L &= \max\{a_{ij} \ : \ (i,j) \in [m] \times [n]\}, \\ D &= \max\{|a_{ij} - a_{i,j-1}| \ : \ i \in [m],\, j \in [n+1]\}. \end{aligned}$$

One of the algorithms in [23] has run-time $O(mn^2 h \log D)$ and approximation ratio $\frac{24}{13}(\log D + 1)$.

Exact algorithms. More recently, there was some more activity in the area of exact algorithms for the problem. The integer programming approach of Langer *et al.* [93] is computationally feasible only for very small instances. The combinatorial algorithm in [83] solves the problem in time $O(mn^{2L+2})$ where L is the maximal entry of the input matrix A. This is polynomial for fixed L, but computational tests show that the method is not efficient enough to tackle problems of practical relevance. The same is true for the combined integer and constraint programming model of Baatar *et al.* [15], the constraint programming approach of Ernst *et al.* [58] and the mixed integer programming model of Wake, Boland and Jennings [126]. Another integer programming model was developed by Taşkın *et al.* [124], and their algorithm, combining integer programming decomposition and combinatorial search techniques, was shown to be capable of solving real-world problems within practicable computational limits. Moreover, the model is flexible in that additional constraints like ICC and TGC can be easily included. There has also been some work on worst case run-time bounds for exact algorithms. Cambazard *et al.* [31] show that the problem for a single row is fixed parameter tractable for the parameter L. In particular, there is a shortest-path based algorithm with run-time $O(p(L)^2n)$, where $p(L)$ is the number of unordered partitions of L as a sum of positive integers. Based on this they propose two hybrid methods [32] using Lagrangian relaxation and column generation, respectively. Finally, Biedl *et al.* [22] present an algorithm with an improved asymptotic run-time analysis. In particular, they show that the DC problem can be solved in time $O(mn^L/2^{(1/2-\varepsilon)(H-1)})$ for every $\varepsilon > 0$.

2.4 Approximative shape matrix decomposition

The input data for the shape matrix decomposition problem is a matrix A which is the result of optimizing beam directions and dose distributions. The algorithms producing the matrix A are based on simplified models of the radiation physics which do not capture all the dosimetric details. Thus it makes sense to think of the entries of A as having an error bar attached. In addition, the exact shape matrix decomposition of A might imply a prohibitively large DT. These considerations motivate the question for the minimum DT in a segmentation of a matrix that approximates the matrix A in some sense. The investigation of this problem was proposed first by Engel and Kiesel [54]. More precisely, the problem introduced in this paper is looking for a matrix B that lexicographically minimizes the following two objectives:

1. the minimum DT of a shape matrix decomposition of the matrix B, and
2. the *total change*, defined as $\sum\limits_{(i,j)\in[m]\times[n]} |a_{ij} - b_{ij}|$.

The condition that B approximates A is reflected by the constraints that $|a_{ij} - b_{ij}| \leqslant \delta$ for all $(i,j) \in [m] \times [n]$ where δ is an input parameter. Engel and Kiesel [54] solved this for the unconstrained case, and their algorithm has a run-time bound of $O(\delta^3mn^2)$. In [8] the min-max-characterization of the optimal DT for the problem with ICC from Theorem 7 is used to derive an algorithm for determining the minimum DT of an approximation matrix in time $O(\delta^2mn^2)$. For the total change minimization this paper suggests a heuristic approach. The complete bi-objective problem is considered in [7]. This paper presents a minimum cost network flow formulation for the dual of the problem to find a shape matrix decomposition with minimum total change subject to the condition that the DT equals a prescribed value, for instance the smallest possible one which can be determined according to [8]. From this formulation an algorithm is derived which solves the problem in time $O(m^2n^2\log^2(mn))$. Another approximation problem, motivated by known dosimetric shortcomings of certain radiation field forms, is treated by Kiesel and Gauer [88], where the set of allowed shape matrices is restricted to matrices that correspond to field shapes with good dosimetric properties. In many cases this restriction makes an exact shape matrix decomposition impossible, hence an approximation is necessary. A related but more general problem is investigated by Engelbeen *et al.* [56] where a number of results on hardness of approximation are derived. The more general problem is to find, for an input set $\mathcal{S}$ of binary $m \times n$-matrices (for instance the set of dosimetrically acceptable shape matrices in [88]), the best approximation of a given matrix A as a positive linear combination of elements of $\mathcal{S}$.

2.5 Decomposition into rectangle matrices

It is an interesting problem to study the shape matrix decomposition problem for smaller classes of binary matrices. In the IMRT application, intensity modulation without an MLC leads to the problem of decomposing the intensity matrix A into binary matrices whose 1-entries form a rectangle [41, 130]. For binary input matrices this is a well studied problem in computational geometry [59, 95, 102, 120]. For matrices with arbitrary nonnegative integer entries the problem was investigated by Engel [53]. For the special case of matrices with two rows his maximum flow formulation leads to a combinatorial algorithm which runs in time $O(n^2)$. For more than two rows an integer programming formulation is presented and an efficient algorithm based on the revised simplex method is described. The paper [53] ends with a list of open problems concerning the integrality gap of rectangle decomposition problems. The second of these problems asks for a proof of the conjecture that for binary input matrices this integrality gap vanishes, and this problem is settled in [6].

3 Dynamic network problems

In this section, we collect results about network problems with an additional time aspect. In Section 3.1 we describe some of the main results in the area of *flows over time*, and in Section 3.2 we provide some background on the scheduling problem studied in [2], [3] and [4] where the objective is to maximize the total flow through a network over a given time horizon. Finally, Section 3.3 contains the description of a new class of network design problems, introduced in [1], where a network is built over a number of time periods, and the aim is to arrange the construction process in such a way, that not only the ultimate network is optimal, but also the intermediate networks that are available in each time period.

3.1 Flows over time

Many real-world systems that are modeled by a mathematical network, such as transportation or telecommunication networks, have a time component which is not reflected in the basic models presented in Section 1. There are several ways to introduce a temporal dimension into the standard flow problems: the arcs can have associated transit times, or the network specifications like capacities and costs might change over time. Such temporal aspects in flow problems can be captured using the concept of *flow over time* (also called *dynamic flow*) which was introduced by Ford and Fulkerson [67, 68] in the following form: given a network with transit times on the arcs, determine the maximum flow that can be sent from a source s to a sink t in T time units. In [67] this problem is reduced to the computation of a static minimum cost circulation in the extended network obtained by adding an arc from t to s, and interpreting the transit times as costs. The correctness of the algorithm follows from a generalization of the Max-Flow Min-Cut Theorem (Theorem 1) using the appropriate notion of an *s-t-cut over time.* In contrast, the dynamic version of the minimum cost flow problem is NP-complete which was shown by Klinz and Woeginger [90], but it can be solved in pseudo-polynomial time using *time-expanded networks.* A time expanded network corresponding to the network (V, A) contains copies of every node $v \in V$ for different points in time, and an arc $a = (v, w) \in A$ with transit time τ_a is replaced by arcs joining the copy of v for time i with the copy of w for time $i + \tau_a$. The minimum cost flow problem over time has a fully polynomial-time approximation scheme based on condensed time-expanded networks introduced by Fleischer and Skutella [62]. For a comprehensive overview of the large area of flows over time we refer to the the survey papers of Aronson [14], Powell *et al.* [107] and Kotnyek [91], as well as the expository book chapter of Skutella [118].

In the problems relevant to this thesis the transit times are neglected, but the concept of a flow over time makes sense also for zero transit times. Suppose the goal is to send certain amounts of flow between different source-sink pairs and the capacity of the network is not sufficient to satisfy all demands. Then looking for a collection of flows such that their sum satisfies all demands is a flow over time problem which was studied by Hajek and Ogier [74] and Fleischer [60]. Another situation where flow over time with zero transit times naturally occurs is when the input data (capacities and costs) changes over time [12, 13, 75, 99, 106]. In the problem studied in the papers [2], [3] and [4], the capacities are piecewise constant as a function of time. Variants of the maximum flow problem with piecewise constant capacities were investigated by Ogier [101] and Fleischer [61]. In contrast to their work, in the problem described in Section 3.2, the piecewise constant capacities are not given as input, but depend on the scheduling decisions that are made while solving the problem.

3.2 Maintenance scheduling for the Hunter Valley Coal Chain

The port of Newcastle, NSW, Australia, is the world's largest coal exporting facility, used for shipping around 80 different brands of coal from about 40 mines owned by 11 producers. Four train haulage operators bring the coal from the mines to three coal loading terminals. The inland portion of the coal export chain, following the path of the Hunter river, is the Hunter Valley Coal Chain (HVCC, see Figure 6). In 2009, the Hunter Valley Coal Chain Coordinator Limited (HVCCC, `http://www.hvccc.com.au`) was founded as an independent legal entity to plan and coordinate the daily operations as well as the long term strategic planning of coal producers and service providers. The book chapter of Boland and Savelsbergh [26] provides an excellent overview on the history of the HVCCC, as well as on a variety of challenging optimization problems that have been identified and that are in the focus of highly active research. These range from detailed models for the daily operational planning [24, 37], over medium term planning problems as the annual maintenance scheduling [3, 4], to decision support tools for the long term strategic planning of infrastructure expansion to satisfy increasing demand [27, 116].

The necessary steps to bring the coal from the mines to the ships in the port can be coarsely described as follows. From load points associated with one or more mines, the

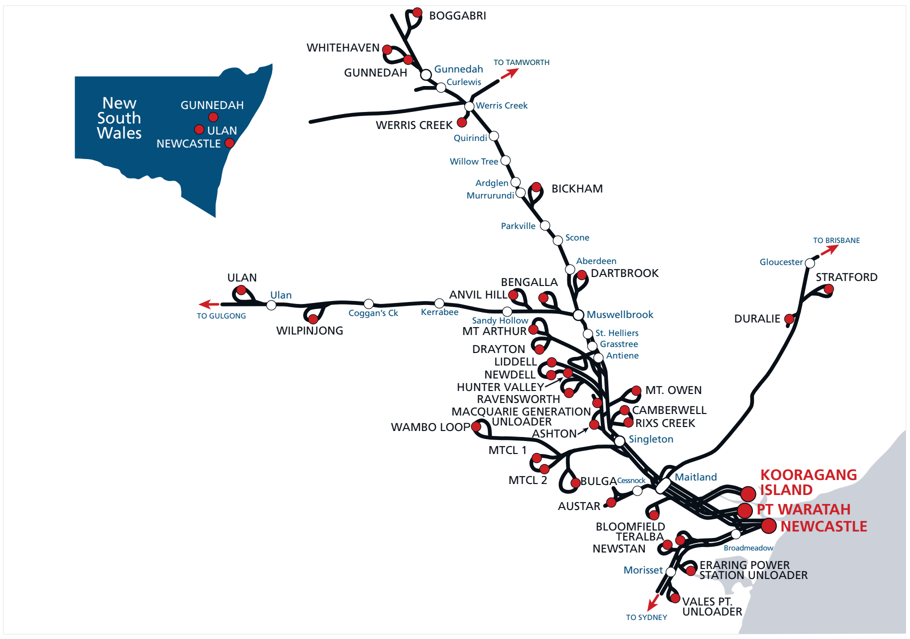

Figure 6: The Hunter Valley Coal Chain ([26], see also `http://www.hvccc.com.au`).

coal is transported to the terminal by train. Upon arrival, a train dumps its contents at a dump station, and it is then transported to a pad where it is added to a stockpile by a stacker. It usually must dwell on the pad for up to ten days before the ship is available for loading. Reclaimers are used to reclaim the coal which is then transferred to a berth on a conveyor. Finally, it is loaded onto a ship with a ship loader. The structure of the coal chain is schematically shown in Figure 7.

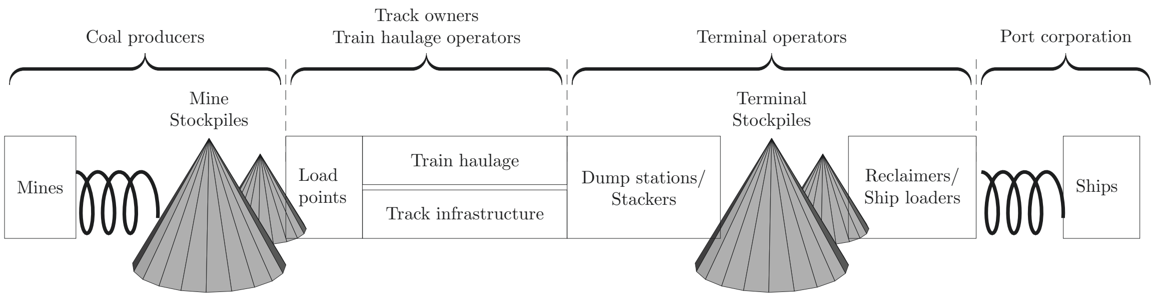

Figure 7: Schematic view of the Hunter Valley Coal Chain (adapted from [40]).

The problem discussed in the papers [3] and [4] is motivated by the annual maintenance planning process carried out by the HVCCC. Supply chain components such as railway track sections, terminal equipment and load points have to undergo regular preventive and corrective maintenance, causing a significant loss in system capacity (up to 15%). The

HVCCC had observed that careful scheduling of the maintenance jobs – good alignment of them – could reduce the impact of maintenance on the network capacity, and established a regular planning activity to carry it out, called "capacity alignment".

The starting point of the work in [4] is the observation that the movement of the coal through the system can be viewed as a flow in a network, where the nodes are load-points at the mines and junctions in the railway network, and arcs are railway track sections or certain pieces of equipment at the terminals. In this model any maintenance job causes a temporary capacity reduction on the corresponding arc, so the evaluation of a fixed maintenance schedule with respect to the total annual throughput is a maximum flow over time problem with piecewise constant capacities, and this reduces to a sequence of independent maximum flow problems, one for each time interval of constant capacities. More formally, an optimization problem called *maximum total flow with flexible arc outages* (**MaxTFFAO**) is introduced. An instance of this problem is given by

- a network (V, A) with a source $s \in V$ and a sink $t \in V$,
- a time horizon T, and
- a list J of jobs, where a *job* j is specified by its associated arc a_j, its processing time p_j, its release date $r_j \in [T - p_j + 1]$, and its deadline $d_j \in [r_j + p_j - 1, T]$.

A solution of the problem is specified by start times $S_j \in [r_j, d_j - p_j + 1]$ for the jobs $j \in J$. The arc a_j is not available for carrying flow in the p_j time periods $S_j, S_j + 1, \dots, S_j + p_j - 1$. In order to evaluate a solution, we denote the set of available arcs at time i by A_i, i.e.

$$A_i = A \setminus \{a \ : \ a = a_j \text{ for some } j \in J \text{ with } S_j \leqslant i \leqslant S_j + p_j - 1\}.$$

The value of the solution $(S_j)_{j \in J}$ is $\sum_{i=1}^{T} F_i$ where F_i is the maximum value of an s-t-flow in the network (V, A_i). By reduction from 3-Partition, it can be shown that this problem is hard even when the network contains only one node except s and t, a so-called *transshipment node.*

Theorem 12 ([4]). *The problem **MaxTFFAO** is strongly NP-complete even for a network with a single transshipment node.*

In [4], several improvement heuristics are presented. They are based on a local search where the neighbourhood of a solution consists of all solutions that can be obtained by changing the start time of at most one job. Dual information from the maximum flow problems for evaluating the current solution is used to identify jobs for which changing the start time can improve the objective.

A special case of the problem **MaxTFFAO** is studied [2]: an arc can be associated with at most one job, all processing times and release dates are equal to 1, and all deadlines are equal to T. Thus the release dates and deadlines are no constraints, and the job set can be identified with a subset of the arc set. Consequently, the problem can be stated as follows: given a network and a subset $J \subseteq A$ of arcs that have to undergo maintenance for exactly one time period in the time horizon, what is the maximum flow that can be pushed through the network? The proof of Theorem 12 in [4] shows that this special case is strongly NP-complete. In [2] several problem characteristics are identified that influence the complexity.

Theorem 13 ([2]). *If all arcs have capacity 1 then the problem can be solved in polynomial time.*

Theorem 14 ([2]). *Already for a network with a single transshipment node and a time horizon of two periods it is NP-hard to decide if scheduling all jobs at the same time is optimal.*

Theorem 15 ([2]). *For series parallel networks and fixed time horizon, there is a pseudo-polynomial algorithm.*

In [3] the HVCC is considered in much more operational detail. A large-scale mixed integer programming model for the maintenance scheduling problem is presented which takes into account the following aspects.

1. The coal is stored at the terminal, typically between three and ten days.
2. In reality, not every maintenance job makes the corresponding arc completely unavailable. In some cases, the capacity is reduced only by a certain amount.
3. There are constraints on which jobs can be done simultaneously, for instance due to availability of equipment or workforce, or due to safety reasons.
4. The capacities of the machines at the terminal are not independent. For instance, there are stackers whose stacking capacity depends on the dump station from which it gets the coal.
5. In practice, the maintenance scheduling does not start from scratch, but from preliminary schedules prepared by the individual service providers like railway companies and terminal operators. In addition to maximizing the total throughput, it is desirable that the optimized schedule deviates as little as possible from the initial schedule.

The first point is captured by introducing special nodes representing the pads where the coal is stored at the terminals. In these nodes flow can be stored which is a well studied concept in research on flows over time (cf. [91]). This can be interpreted as introducing a time expanded network which has a copy of the original network for each time period and the networks for consecutive time periods are linked by arcs connecting the copies of the storage nodes. The second point makes it necessary to introduce binary variables for pairs (a, ρ) of an arc a and a ratio ρ, indicating if the arc a is affected by a job which reduces its capacity by a factor of $\rho \in [0, 1]$. This is possible because there are only very few possible values for ρ. The third point can be modelled by introducing clique constraints that ensure that from a set of jobs at most one job is processed at any given time. For the fourth point, it is established that for every terminal in the HVCCC instance the logic of the operations can be captured by a network model, although the one-to-one correspondence between arcs

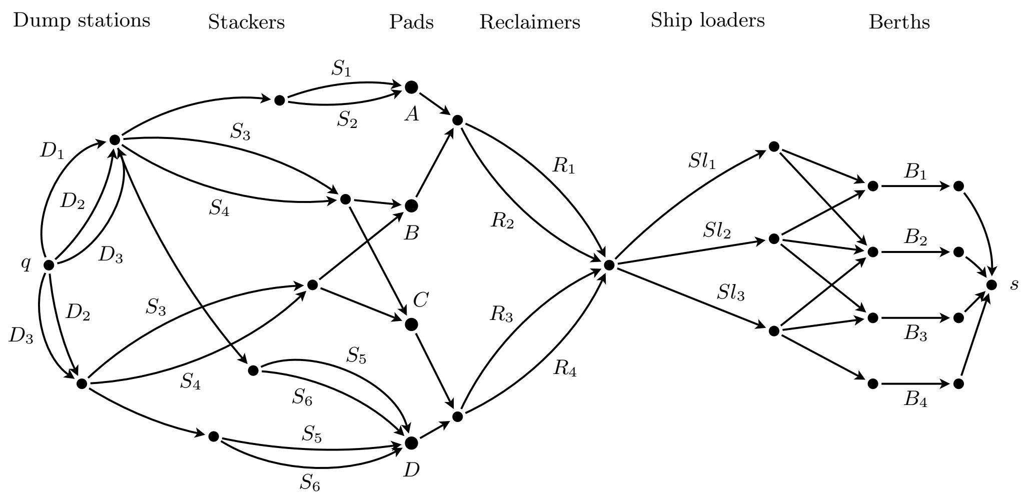

Figure 8: A terminal network. Arcs with the same label represent the same piece of equipment.

and machines has to be given up. Figure 8 shows the most complicated terminal. For the fifth point, a two-stage approach is proposed. In the first phase, the total throughput is maximized without taking into account the initial schedule. In the second phase, the MIP is changed in two ways. (i) The objective is changed from maximizing the throughput to minimizing the number of jobs whose start time differs from the one in the initial schedule. (ii) A lower bound for the total throughput is added to ensure that the total throughput for the optimized schedule is sufficiently close to the value found in the first phase.

Because the planning horizon in the practical problem is a whole year and the maintenance jobs are scheduled at a time scale of a half hour, the MIP is too large to be solved directly. A method based on solving smaller subproblems is proposed to deal with this difficulty. Starting from any feasible solution, the values of all variables which model events outside a certain time window are fixed, and the resulting smaller model is solved. After updating the values for the variables that are not fixed, the time window is moved, and the procedure is iterated. The paper reports on extensive computational tests of this strategy.

3.3 Incremental network design problems

Another area where a temporal aspect naturally occurs in a network optimization context is network design. This fundamental class of optimization problems has a rich research tradition, and it is a very general framework, containing a variety of important problems as special cases. Magnanti and Wong [98] give an extensive overview of models and algorithms with a focus on applications in transportation planning, and Kerivin and Mahjoub [87] survey design problems for telecommunication networks. Recently, in the area of transportation network planning there has been a growing interest in network design in multiple stages [89, 125]. This is very natural in a context where for instance a road network is expanded over several years, and properties of the intermediate steps in the expansion process have to be taken into account in addition to the final result. A class of problems that captures this basic idea is introduced in [1]. An *incremental network design problem* can be associated with any network optimization problem like shortest path, maximum flow, etc. In the basic version, an instance of an incremental network design problem is given by a network (V, A), a time horizon $T \in \mathbb{N}$, and a partition $A = A_e \cup A_p$ of the arc set into the set A_e of *existing arcs* and the set A_p of *potential arcs*. For a complete instance specification, there must be given some additional information like costs, capacities and budgets. The interpretation is that in the beginning of the planning horizon the network is (V, A_e) and the potential arcs in A_p represent the expansion options, i.e. they can be added to the network (possibly at a cost). In addition to limiting the expansion per time period by the cost structure, there can also be explicit budgets. The task is to decide, for each time period $i = 1, 2, \dots, T$, which arcs are added. The objective is to optimize the cumulative value which is obtained as the sum of the values for the associated network problem when it is solved on the network that has been build up to time i for $i = 1, 2, \dots, T$.

After defining this general problem class, the paper [1] deals with the special case where the underlying problem is the shortest path problem, the time horizon equals the number of potential arcs, and the budget constraint is that at most one arc can be added per time period. This problem is called Incremental network design problem with shortest paths (IND-SP) [1]:

Instance. A network (V, A), a source node $s \in V$ and a sink node $t \in V$, a partition $A = A_e \cup A_p$, a length function $l : A \to \mathbb{R}_{\geqslant 0}$, and the time horizon $T = |A_p|$.

Solution. A permutation $(a_1, \dots, a_T)$ of A_p.

Value. $\sum\limits_{i=1}^{T} L_i$ where L_i is the length of a shortest s-t path in the network $(V, A \cup \{a_1, \dots, a_i\})$.

We call a shortest s-t-path in (V, A_e) an *initial shortest path* and denote its length by L_0. Similarly, a shortest s-t-path in $(V, A_e \cup A_p)$ is called *ultimate shortest path* and has length L_T which is independent of the solution $(a_1, \dots, a_T)$.

Theorem 16 ([1])**.** *The problem IND-SP is NP-hard.*

Two natural greedy strategies for the problem IND-SP are to

1. build an ultimate shortest path as quickly as possible, and
2. always build arcs that lead to a shorter path as quickly as possible.

In [1], it is shown that there are instances where both strategies, thus also the combined strategy that takes the better of the two solutions, are performing arbitrary bad in comparison to the optimal solution. On the other hand, a 4-approximation algorithm is derived which is based on scaling the gap between the lengths of the initial shortest path and the ultimate shortest path. Let $\Delta = L_0 - L_T$ denote this gap. Then $O(\log \Delta)$ paths are constructed whose lengths are bounded by $L_T + \Delta/2^i$ for $i = 1, 2, \dots, \lfloor \log_2 \Delta \rfloor$. In an initial step this algorithm requires the computation of shortest s-t-paths containing at most k potential arcs for each $k = 1, 2, \dots, T$. The run-time of this step can be bounded by $O(Tn^3)$ which dominates the run-time of the complete algorithm, thus giving the following result.

Theorem 17 ([1])**.** *A 4-approximation for the problem IND-SP can be computed in time* $O(Tn^3)$*.*

Papers in this thesis

[1] Matthew Baxter, Tarek Elgindy, Andreas T Ernst, Thomas Kalinowski and Martin WP Savelsbergh. “Incremental network design with shortest paths”. In: *European Journal of Operational Research* 238.3 (2014), pp. 675–684. DOI: `10.1016/j.ejor.2014.04.018`.

[2] N. Boland, T. Kalinowski, R. Kapoor and S. Kaur. “Scheduling unit processing time arc shutdown jobs to maximize network flow over time: complexity results”. In: *Networks* 63.2 (2014), pp. 196–202. DOI: `10.1002/net.21536`.

[3] N. Boland, T. Kalinowski, H. Waterer and L. Zheng. “Mixed integer programming based maintenance scheduling for the Hunter Valley Coal Chain”. In: *Journal of Scheduling* 16.6 (2013), pp. 649–659. DOI: `10.1007/s10951-012-0284-y`.

[4] N. Boland, T. Kalinowski, H. Waterer and L. Zheng. “Scheduling arc maintenance jobs in a network to maximize total flow over time”. In: *Discr. Appl. Math.* 163 (2014), pp. 34–52. DOI: `10.1016/j.dam.2012.05.027`.

[5] T. Kalinowski. “Reducing the tongue–and–groove underdosage in MLC shape matrix decomposition”. In: *Algorithmic Operations Research* 3.2 (2008), pp. 165–174.

[6] T. Kalinowski. “A dual of the rectangle-segmentation problem for binary matrices”. In: *The Electronic Journal of Combinatorics* 16 (2009), R89.

[7] T. Kalinowski. “A minimum cost flow formulation for approximated MLC segmentation”. In: *Networks* 57.2 (2011), pp. 135–140. DOI: `10.1002/net.20394`.

[8] T. Kalinowski and A. Kiesel. “Approximated MLC shape matrix decomposition with interleaf collision constraint”. In: *Algorithmic Operations Research* 4.1 (2009), pp. 49–57.

Other references

[9] R.K. Ahuja and H.W. Hamacher. “A Network Flow Algorithm to Minimize Beam-On Time for Unconstrained Multileaf Collimator Problems in Cancer Radiation Therapy”. In: *Networks* 45.1 (2005), pp. 36–41. DOI: `10.1002/net.20047`.

[10] R.K. Ahuja, T.L. Magnanti and J.B. Orlin. *Network flows.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[11] R.K. Ahuja, K. Mehlhorn, J.B. Orlin and R.E. Tarjan. “Faster Algorithms for the Shortest Path Problem”. In: *Journal of the ACM* 37.2 (1990), pp. 213–223.

[12] E.J. Anderson, P. Nash and A.B. Philpott. "A class of continuous network flow problems". In: *Mathematics of Operations Research* 7.4 (1982), pp. 501–514. DOI: `10.1287/moor.7.4.501`.

[13] E.J. Anderson and A.B. Philpott. "Optimisation of flows in networks over time". In: *Probability, statistics and optimisation.* Ed. by F.P. Kelly. Wiley Ser. Prob. Math. Statist. Wiley, 1994, pp. 369–382.

[14] J.E. Aronson. "A survey of dynamic network flows". In: *Annals of Operations Research* 20.1 (1989), pp. 1–66. DOI: `10.1007/BF02216922`.

[15] D. Baatar, N. Boland, S. Brand and P. Stuckey. "Minimum cardinality matrix decomposition into consecutive-ones matrices: CP and IP approaches". In: *Proc. 4th CPAIOR 2007.* Ed. by P. Van Hentenryck and L. Wolsey. Vol. 4510. LNCS. Springer, 2007, pp. 1–15. DOI: `10.1007/978-3-540-72397-4`.

[16] D. Baatar, N. Boland, R. Johnston and H.W. Hamacher. "A new sequential extraction heuristic for optimizing the delivery of cancer radiation treatment using multileaf collimators". In: *INFORMS Journal on Computing* 21.2 (2009), pp. 224–241. DOI: `10.1287/ijoc.1080.0288_1`.

[17] D. Baatar and H.W. Hamacher. "New LP Model for Multileaf Collimators in Radiation Therapy". Contribution to the conference ORP3, University of Kaiserslautern. 2003.

[18] D. Baatar, H.W. Hamacher, M. Ehrgott and G.J. Woeginger. "Decomposition of integer matrices and multileaf collimator sequencing". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 6–34. DOI: `10.1016/j.dam.2005.04.008`.

[19] N. Bansal, D. Coppersmith and B. Schieber. "Minimizing Setup and Beam-On Times in Radiation Therapy". In: *Proc. 9th WS on Approximation Algorithms for Combinatorial Optimization Problems APPROX'06.* Ed. by J. Díaz, K. Jansen, J.D.P. Rolim and U. Zwick. Vol. 4110. LNCS. Springer, 2006, pp. 27–38. DOI: `10.1007/11830924_5`.

[20] M.S. Bazaraa and R.W. Langley. "A dual shortest path algorithm". In: *SIAM Journal on Applied Mathematics* 26.3 (1974), pp. 496–501. DOI: `10.1137/0126047`.

[21] R. Bellman. "On a Routing Problem". In: *Quarterly of Applied Mathematics* 16 (1958), pp. 87–90.

[22] T. Biedl, S. Durocher, C. Engelbeen, S. Fiorini and M. Young. "Faster optimal algorithms for segment minimization with small maximal value". In: *Discr. Appl. Math.* 161.3 (2013), pp. 317–329. DOI: `10.1016/j.dam.2012.09.011`.

[23] T. Biedl, S. Durocher, H.H. Hoos, S. Luan, J. Saia and M. Young. "A note on improving the performance of approximation algorithms for radiation therapy". In: *Information Processing Letters* 111.7 (2011), pp. 326–333. DOI: `10.1016/j.ipl.2010.12.011`.

[24] N. Boland, D. Gulczynski and M. Savelsbergh. "A stockyard planning problem". In: *EURO Journal of Transportation and Logistics* (2013), pp. 1–40. DOI: `10.1007/s13676-012-0011-z`.

[25] N. Boland, H. W. Hamacher and F. Lenzen. "Minimizing beam-on time in cancer radiation treatment using multileaf collimators". In: *Networks* 43.4 (2004), pp. 226–240. ISSN: 0028-3045. DOI: `10.1002/net.20007`.

[26] N. Boland and M. Savelsbergh. "Optimizing the Hunter Valley coal chain". In: *Supply Chain Disruptions: Theory and Practice of Managing Risk*. Ed. by H. Gurnani, A. Mehrotra and S. Ray. Springer-Verlag London Ltd., 2011, pp. 275–302.

[27] N. Boland, M. Savelsbergh and H. Waterer. "Shipping Data Generation for the Hunter Valley Coal Chain". submitted. 2013.

[28] T.R. Bortfeld. "IMRT: a review and preview". In: *Phys. Med. Biol.* 51 (2006), R363–R379. DOI: `10.1088/0031-9155/51/13/R21`.

[29] T.R. Bortfeld, D.L. Kahler, T.J. Waldron and A.L. Boyer. "X–ray field compensation with multileaf collimators". In: *Int. J. Radiat. Oncol. Biol. Phys.* 28 (1994), pp. 723–730. DOI: `10.1016/0360-3016(94)90200-3`.

[30] A. Brahme. "Optimization of stationary and moving beam radiation therapy techniques". In: *Radiother. Oncol.* 12 (1988), pp. 129–140. DOI: `10.1016/0167-8140(88)90167-3`.

[31] H. Cambazard, E. O'Mahony and B. O'Sullivan. "A shortest path-based approach to the multileaf collimator sequencing problem". In: *Proc. 6th CPAIOR 2009*. Ed. by W.-J. van Hoeve and J.N. Hooker. Vol. 5574. LNCS. Springer, 2009, pp. 41–55. DOI: `10.1007/978-3-642-01929-6_5`.

[32] H. Cambazard, E. O'Mahony and B. O'Sullivan. "Hybrid methods for the multileaf collimator sequencing problem". In: *Proc. 7th CPAIOR 2010*. Ed. by A. Lodi, M. Milano and P. Toth. Vol. 6140. LNCS. Springer, 2010, pp. 56–70. DOI: `10.1007/978-3-642-13520-0_9`.

[33] D. Chen, X. Hu, S. Luan, C. Wang and X. Wu. "Geometric algorithms for static leaf sequencing problems in radiation therapy". In: *International Journal of Computational Geometry & Applications* 14 (2004), pp. 311–339. DOI: `10.1142/S0218195904001494`.

[34] D. Chen, X. Hu, S. Luan, X. Wu and C. Yu. "Optimal terrain construction problems and applications in intensity-modulated radiation therapy". In: *Algorithmica* 42.3 (2005), pp. 265–288. DOI: `10.1007/s00453-005-1169-7`.

[35] D. Chen, X.S. Hu, S. Luan, S.A. Naqvi, C. Wang and C.X. Yu. "Generalized geometric approaches for leaf sequencing problems in radiation therapy". In: *International Journal of Computational Geometry & Applications* 16 (2006), pp. 175–204. DOI: `10.1142/S0218195906001999`.

[36] Y. Chen, Q. Hou and J.M. Galvin. "A graph-searching method for MLC leaf sequencing under constraints". In: *Medical physics* 31 (2004), p. 1504. DOI: `10.1118/1.1737512`.

[37] R. Clement. “MIP models for scheduling the operations for a coal loading facility”. In: *Proc. 44th ORSNZ conference*. 2009, pp. 139–148.

[38] M.J. Collins, D. Kempe, J. Saia and M. Young. “Nonnegative integral subset representations of integer sets”. In: *Information Processing Letters* 101.3 (2007), pp. 129–133. DOI: 10.1016/j.ipl.2006.08.007.

[39] D.J. Convery and M.E. Rosenbloom. “The generation of intensity-modulated fields for conformal radiotherapy by dynamic collimation”. In: *Physics in Medicine and Biology* 37.6 (1992), pp. 1359–1374. DOI: 10.1088/0031-9155/37/6/012.

[40] Hunter Valley Coal Chain Coordinator. *Overview Presentation.* http://www.hvccc.com.au/Communications/Miscellaneous%20Presentations/HVCCC%20Overview%202012.pdf (13th March 2013). 2012.

[41] J. Dai and Y. Hu. “Intensity-modulation radiotherapy using independent collimators: an algorithm study”. In: *Medical physics* 26 (1999), pp. 2562–2570. DOI: 10.1118/1.598794.

[42] J. Dai and W. Que. “Simultaneous minimization of leaf travel distance and tongue-and-groove effect for segmental intensity-modulated radiation therapy”. In: *Phys. Med. Biol.* 49.23 (2004), pp. 5319–5331. DOI: 10.1088/0031-9155/49/23/009.

[43] J. Dai and Y. Zhu. “Minimizing the number of segments in a delivery sequence for intensity-modulated radiation therapy with a multileaf collimator”. In: *Medical physics* 28 (2001), pp. 2113–2120. DOI: 10.1118/1.1406518.

[44] G.B. Dantzig and D.R. Fulkerson. “On the max-flow min-cut theorem of networks”. In: *Linear inequalities and related systems*. Ed. by H.W. Kuhn and A.W. Tucker. Vol. 38. Annals of Mathematics Studies. Princeton University Press, 1956, pp. 215–221.

[45] J. Deng, T. Pawlicki, Y. Chen, J. Li, S.B. Jiang and C.M. Ma. “The MLC tongue-and-groove effect on IMRT dose distributions”. In: *Phys. Med. Biol.* 46.4 (2001), pp. 1039–1060. DOI: 10.1088/0031-9155/46/4/310.

[46] E.W. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[47] E.A. Dinic. “Algorithm for solution of a problem of maximum flow in networks with power estimation”. In: *Soviet Math. Dokl.* 11.5 (1970), pp. 1277–1280.

[48] X. Dou, X. Wu, J. Bayouth and J. Buatti. “The matrix orthogonal decomposition problem in intensity-modulated radiation therapy”. In: *Proc. 12th annual international conference on Computing and Combinatorics COCOON 2006*. Ed. by D.Z. Chen and D.T Lee. Vol. 4112. LNCS. Springer, 2006, pp. 156–165. DOI: 10.1007/11809678.

[49] J. Edmonds and R.M. Karp. “Theoretical improvements in algorithmic efficiency for network flow problems”. In: *Journal of the ACM* 19.2 (1972), pp. 248–264.

[50] M. Ehrgott, Ç. Güler, H.W. Hamacher and L. Shao. "Mathematical optimization in intensity modulated radiation therapy". In: *Annals of Operations Research* 175.1 (2010), pp. 309–365. DOI: 10.1007/s10479-009-0659-4.

[51] M. Ehrgott, H.W. Hamacher and M. Nußbaum. "Decomposition of matrices and static multileaf collimators: A survey". In: *Optimization in medicine.* Ed. by C.J.S. Alves, P.M. Pardalos and L.N. Vicente. Springer, 2008, pp. 25–46.

[52] K. Engel. "A new algorithm for optimal multileaf collimator field segmentation". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 35–51. DOI: 10.1016/j.dam.2004.10.007.

[53] K. Engel. "Optimal matrix-segmentation by rectangles". In: *Discr. Appl. Math.* 157.9 (2009), pp. 2015–2030. DOI: 10.1016/j.dam.2008.12.008.

[54] K. Engel and A. Kiesel. "Approximated matrix decomposition for IMRT planning with multileaf collimators". In: *OR Spectrum* 33.1 (2011), pp. 149–172. DOI: 10.1007/s00291-009-0168-5.

[55] K. Engel and E. Tabbert. "Fast Simultaneous Angle, Wedge, and Beam Intensity Optimization in Inverse Radiotherapy Planning". In: *Optimization and Engineering* 6.4 (2005), pp. 393–419. DOI: 10.1007/s11081-005-2065-3.

[56] C. Engelbeen, S. Fiorini and A. Kiesel. "A closest vector problem arising in radiation therapy planning". In: *Journal of combinatorial optimization* 22.4 (2011), pp. 609–629. DOI: 10.1007/s10878-010-9308-8.

[57] C. Engelbeen and A. Kiesel. "Binary matrix decompositions without tongue-and-groove underdosage for radiation therapy planning". In: *Algorithmic Operations Research* 5.2 (2010), pp. 119–132.

[58] A.T. Ernst, V.H. Mak and L.R. Mason. "An exact method for the minimum cardinality problem in the treatment planning of intensity-modulated radiotherapy". In: *INFORMS Journal on computing* 21.4 (2009), pp. 562–574. DOI: 10.1287/ijoc.1080.0308.

[59] L. Ferrari, P.V. Sankar and J. Sklansky. "Minimal Rectangular Partitions of Digitized Blobs". In: *Computer Vision, Graphics and Image Processing* 28 (1984), pp. 58–71. DOI: 10.1016/0734-189X(84)90139-7.

[60] L. Fleischer. "Faster algorithms for the quickest transshipment problem". In: *SIAM journal on Optimization* 12.1 (2001), pp. 18–35. DOI: 10.1137/S1052623497327295.

[61] L. Fleischer. "Universally maximum flow with piecewise-constant capacities". In: *Networks* 38.3 (2001), pp. 115–125. DOI: 10.1002/net.1030.

[62] L. Fleischer and M. Skutella. "Quickest flows over time". In: *SIAM Journal on Computing* 36.6 (2007), pp. 1600–1630. DOI: 10.1137/S0097539703427215.

[63] R.W. Floyd. "Algorithm 97: shortest path". In: *Communications of the ACM* 5.6 (1962), p. 345.

[64] L.R. Ford. *Network Flow Theory.* Tech. rep. Paper P-923. Santa Monica, California: RAND Corporation, 1956.

[65] L.R. Ford and D.R. Fulkerson. "Maximal flow through a network". In: *Canadian Journal of Mathematics* 8.3 (1956), pp. 399–404.

[66] L.R. Ford and D.R. Fulkerson. "A simple algorithm for finding maximal network flows and an application to the Hitchcock problem". In: *Canadian Journal of Mathematics* 9 (1957), pp. 210–218.

[67] L.R. Ford and D.R. Fulkerson. "Constructing maximal dynamic flows from static flows". In: *Operations Research* 6.3 (1958), pp. 419–433. DOI: 10.1287/opre.6.3.419.

[68] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton, N.J.: Princeton Univ. Press, 1962.

[69] M.L. Fredman and R.E. Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *Journal of the ACM* 34.3 (1987), pp. 596–615. DOI: 10.1145/28869.28874.

[70] J.M. Galvin, X.G. Chen and R.M. Smith. "Combining multileaf fields to modulate fluence distributions". In: *Int. J. Radiat. Oncol. Biol. Phys.* 27 (1993), pp. 697–705. DOI: 10.1016/0360-3016(93)90399-G.

[71] A.V. Goldberg. "Scaling algorithms for the shortest paths problem". In: *SIAM Journal on Computing* 24.3 (1995), pp. 494–504. DOI: 10.1137/S0097539792231179.

[72] A.V. Goldberg and R.E. Tarjan. "A new approach to the maximum-flow problem". In: *Journal of the ACM* 35.4 (1988), pp. 921–940. DOI: 10.1145/48014.61051.

[73] A.D.A. Gunawardena, W. D'Souza, L.D. Goadrich, R.R. Meyer, K.J. Sorensen, S.A. Naqvi and L. Shi. "A difference-matrix metaheuristic for intensity map segmentation in step-and-shoot IMRT delivery". In: *Phys. Med. Biol.* 51.10 (2006), pp. 2517–2536. DOI: 10.1088/0031-9155/51/10/011.

[74] B. Hajek and R.G. Ogier. "Optimal dynamic routing in communication networks with continuous traffic". In: *Networks* 14.3 (1984), pp. 457–487. DOI: 10.1002/net.3230140308.

[75] J. Halpern. "A generalized dynamic flows problem". In: *Networks* 9.2 (1979), pp. 133–167. DOI: 10.1002/net.3230090204.

[76] H.W. Hamacher and K.-H. Küfer. "Inverse radiation therapy planning – a multiple objective optimization approach". In: *Discr. Appl. Math.* 118 (2002), pp. 145–161. DOI: 10.1016/S0166-218X(01)00261-X.

[77] J. Jing, R. Cao, Y. Wu, G. Li, H. Lin, M. Cheng, X. Pei, W. Kong and G. Li. "Improved Model on Minimizing Static Intensity Modulation Delivery Time". In: *2nd International Conference on Biomedical Engineering and Informatics*. IEEE. 2009, pp. 1–4. DOI: 10.1109/BMEI.2009.5305406.

[78] D.B. Johnson. "Efficient algorithms for shortest paths in sparse networks". In: *Journal of the ACM* 24.1 (1977), pp. 1–13.

[79] T. Kalinowski. "A duality based algorithm for multileaf collimator field segmentation with interleaf collision constraint". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 52–88. DOI: `10.1016/j.dam.2004.10.008`.

[80] T. Kalinowski. "Reducing the number of monitor units in multileaf collimator field segmentation". In: *Phys. Med. Biol.* 50.6 (2005), pp. 1147–1161. DOI: `10.1088/0031-9155/50/6/008`.

[81] T. Kalinowski. "Realization of intensity modulated radiation fields using multileaf collimators". In: *Information Transfer and Combinatorics*. Ed. by R. Ahlswede *et al.* Vol. 4123. LNCS. Springer-Verlag, 2006, pp. 1010–1055. DOI: `10.1007/11889342_65`.

[82] T. Kalinowski. "Multileaf collimator shape matrix decomposition". In: *Optimization in Medicine and Biology*. Ed. by G.J. Lim and E.K. Lee. Auerbach Publishers Inc., 2008, pp. 249–282.

[83] T. Kalinowski. "The complexity of minimizing the number of shape matrices subject to minimal beam-on time in multileaf collimator field decomposition with bounded fluence". In: *Discr. Appl. Math.* 157.9 (2009), pp. 2089–2104. DOI: `10.1016/j.dam.2008.06.027`.

[84] S. Kamath, S. Sahni, J. Li, J. Palta and S. Ranka. "Leaf sequencing algorithms for segmented multileaf collimation". In: *Phys. Med. Biol.* 48.3 (2003), pp. 307–324. DOI: `10.1088/0031-9155/48/3/303`.

[85] S. Kamath, S. Sahni, J. Palta and S. Ranka. "Algorithms for optimal sequencing of dynamic multileaf collimators". In: *Phys. Med. Biol.* 49.1 (2004), pp. 33–54. DOI: `10.1088/0031-9155/49/1/003`.

[86] S. Kamath, S. Sartaj, J. Palta, S. Ranka and J. Li. "Optimal leaf sequencing with elimination of tongue–and–groove underdosage". In: *Phys. Med. Biol.* 49 (2004), N7–N19. DOI: `10.1088/0031-9155/49/3/N01`.

[87] H. Kerivin and A.R. Mahjoub. "Design of survivable networks: A survey". In: *Networks* 46.1 (2005), pp. 1–21. DOI: `10.1002/net.20072`.

[88] A. Kiesel and T. Gauer. "Approximated segmentation considering technical and dosimetric constraints in intensity-modulated radiation therapy with electrons". In: *preprint* (2010). `arXiv:1005.0898`.

[89] B.J. Kim, W. Kim and B.H. Song. "Sequencing and scheduling highway network expansion using a discrete network design model". In: *The Annals of Regional Science* 42.3 (2008), pp. 621–642. DOI: `10.1007/s00168-007-0170-2`.

[90] B. Klinz and G.J. Woeginger. "Minimum-cost dynamic flows: The series-parallel case". In: *Networks* 43.3 (2004), pp. 153–162. DOI: `10.1002/net.10112`.

[91] B. Kotnyek. *An annotated overview of dynamic network flows*. Tech. rep. 4936. `http://hal.inria.fr/inria-00071643/` (20 February 2013). INRIA, 2003.

[92] K.-H. Küfer, A. Scherrer, M. Monz, F. Alonso, H. Trinkaus, T. Bortfeld and C. Thieke. "Intensity-modulated radiotherapy–A large scale multi-criteria programming problem". In: *OR spectrum* 25.2 (2003), pp. 223–249. DOI: 10.1007/s00291-003-0125-7.

[93] M. Langer, V. Thai and L. Papiez. "Improved leaf sequencing reduces segments of monitor units needed to deliver IMRT using multileaf collimators". In: *Med. Phys.* 28 (2001), pp. 2450–2458. DOI: 10.1118/1.1420392.

[94] E.K. Lee, T. Fox and I. Crocker. "Integer programming applied to intensity-modulated radiation therapy treatment planning". In: *Annals of Operations Research* 119.1 (2003), pp. 165–181. DOI: 10.1023/A:1022938707934.

[95] W. Lipski, E. Lodi, F. Luccio, C. Mugnai and L. Pagli. "On two dimensional data organization II". In: *Fund. Informaticae* 2 (1979), pp. 245–260.

[96] S. Luan, J. Saia and M. Young. "Approximation algorithms for minimizing segments in radiation therapy". In: *Information processing letters* 101.6 (2007), pp. 239–244. DOI: 10.1016/j.ipl.2006.10.003.

[97] S. Luan, C. Wang, D.Z. Chen, X.S. Hu, S.A. Naqvi, X. Wu and C.X. Yu. "An improved MLC segmentation algorithm and software for step-and-shoot IMRT delivery without tongue-and-groove error". In: *Med. Phys.* 33 (2006), pp. 1199–1212. DOI: 10.1118/1.2188823.

[98] T.L. Magnanti and R.T. Wong. "Network design and transportation planning: Models and algorithms". In: *Transportation Science* 18.1 (1984), pp. 1–55. DOI: 10.1287/trsc.18.1.1.

[99] E. Minieka. "Maximal, lexicographic, and dynamic network flows". In: *Operations Research* 21.2 (1973), pp. 517–527. DOI: 10.1287/opre.21.2.517.

[100] E.F. Moore. "The shortest path through a maze". In: *Proc. International Symposium on the Theory of Switching, Part II.* Ed. by H. Aiken. Vol. XXX. The Annals of the Computation Laboratory of Harvard University. Harvard University Press, 1959, pp. 285–292.

[101] R.G. Ogier. "Minimum-delay routing in continuous-time dynamic networks with piecewise-constant capacities". In: *Networks* 18.4 (1988), pp. 303–318. DOI: 10.1002/net.3230180405.

[102] T. Ohtsuki. "Minimum dissection of rectilinear regions". In: *Proc. IEEE Symposium on Circuits and Systems.* 1982, pp. 1210–1213.

[103] J.B. Orlin. "A faster strongly polynomial minimum cost flow algorithm". In: *Proc. 20th ACM symposium on Theory of computing, STOC 1988.* ACM. 1988, pp. 377–387. DOI: 10.1145/62212.62249.

[104] J.B. Orlin. "A faster strongly polynomial minimum cost flow algorithm". In: *Operations research* 41.2 (1993), pp. 338–350. DOI: 10.1287/opre.41.2.338.

[105] J.B. Orlin. "Max flows in $O(nm)$ time or better". accepted for STOC 2013, online: http://jorlin.scripts.mit.edu/docs/papersfolder/O(nm)MaxFlow.pdf. 2013.

[106] A.B. Philpott. “Continuous-time flows in networks”. In: *Mathematics of Operations Research* 15.4 (1990), pp. 640–661. DOI: 10.1287/moor.15.4.640.

[107] W.B. Powell, P. Jaillet and A. Odoni. “Stochastic and dynamic networks and routing”. In: *Handbooks in Operations Research and Management Science*. Ed. by M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser. Vol. 8. Handbooks in Operations Research and Management Science. Elsevier, 1995. Chap. 3, pp. 141–295. DOI: 10.1016/S0927-0507(05)80107-0.

[108] F. Preciado-Walters, R. Rardin, M. Langer and V. Thai. “A coupled column generation, mixed integer approach to optimal planning of intensity modulated radiation therapy for cancer”. In: *Mathematical Programming B* 101.2 (2004), pp. 319–338. DOI: 10.1007/s10107-004-0527-6.

[109] W. Que, J. Kung and J. Dai. “’Tongue-and-groove’effect in intensity modulated radiotherapy with static multileaf collimator fields”. In: *Phys. Med. Biol.* 49.3 (2004), pp. 399–405. DOI: 10.1088/0031-9155/49/3/004.

[110] H.E. Romeijn, R.K. Ahuja, J.F. Dempsey and A. Kumar. “A Column Generation Approach to Radiation Therapy Treatment Planning Using Aperture Modulation”. In: *SIAM J. on Optimization* 15.3 (2005), pp. 838–862. ISSN: 1052-6234. DOI: 10.1137/040606612.

[111] H.E. Romeijn, R.K. Ahuja, J.F. Dempsey and A. Kumar. “A new linear programming approach to radiation therapy treatment planning problems”. In: *Operations Research* 54.2 (2006), pp. 201–216. DOI: 10.1287/opre.1050.0261.

[112] J.P.C. van Santvoort and B.J.M. Heijmen. “Dynamic multileaf collimation without tongue-and-groove underdosage effects”. In: *Phys. Med. Biol.* 41.10 (1999), pp. 2091–2105. DOI: 10.1088/0031-9155/41/10/017.

[113] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Algorithms and Combinatorics. Springer, 2003.

[114] D.M. Shepard, M.A. Earl, X.A. Li, S. Naqvi and C. Yu. “Direct aperture optimization: a turnkey solution for step-and-shoot IMRT”. In: *Medical physics* 29 (2002), pp. 1007–1018. DOI: 10.1118/1.1477415.

[115] D.M. Shepard, M.C. Ferris, G.H. Olivera and T.R. Mackie. “Optimizing the delivery of radiation therapy to cancer patients”. In: *SIAM Review* 41.4 (1999), pp. 721–744. DOI: 10.1137/S0036144598342032.

[116] G. Singh, D. Sier, A.T. Ernst, O. Gavriliouk, R. Oyston, T. Giles and P. Welgama. “A mixed integer programming model for long term capacity expansion planning: A case study from The Hunter Valley Coal Chain”. In: *European Journal of Operational Research* (2012). DOI: 10.1016/j.ejor.2012.01.012.

[117] R.A.C. Siochi. “Minimizing static intensity modulation delivery time using an intensity solid paradigm”. In: *Int. J. Radiat. Oncol. Biol. Phys.* 43 (1999), pp. 671–680. DOI: 10.1016/S0360-3016(98)00430-1.

[118] M. Skutella. "An introduction to network flows over time". In: *Research Trends in Combinatorial Optimization* (2009), pp. 451–482. DOI: 10.1007/978-3-540-76796-1.

[119] D.D. Sleator and R.E. Tarjan. "A data structure for dynamic trees". In: *Journal of computer and system sciences* 26.3 (1983), pp. 362–391. DOI: 10.1016/0022-0000(83)90006-5.

[120] V. Soltan and A. Gorpinevich. "Minimum Dissection of a Rectilinear Polygon with Arbitrary Holes into Rectangles". In: *Discrete Computat. Geometry* 9 (1993), pp. 57–79. DOI: 10.1007/BF02189307.

[121] S.V. Spirou and C.S. Chui. "Generation of arbitrary intensity profiles by dynamic jaws or multileaf collimators". In: *Medical Physics* 21 (1994), pp. 1031–1041. DOI: 10.1118/1.597345.

[122] R. Svensson, P. Kallman and A. Brahme. "An analytical solution for the dynamic control of multileaf collimators". In: *Phys. Med. Biol.* 39.1 (1999), p. 37. DOI: 10.1088/0031-9155/39/1/003.

[123] É. Tardos. "A strongly polynomial minimum cost circulation algorithm". In: *Combinatorica* 5.3 (1985), pp. 247–255. DOI: 10.1007/BF02579369.

[124] Z.C. Taşkın, J.C. Smith, H.E. Romeijn and J.F. Dempsey. "Optimal multileaf collimator leaf sequencing in IMRT treatment planning". In: *Operations research* 58.3 (2010), pp. 674–690. DOI: 10.1287/opre.1090.0759.

[125] S.V. Ukkusuri and G. Patil. "Multi-period transportation network design under demand uncertainty". In: *Transportation Research Part B: Methodological* 43.6 (2009), pp. 625–642. DOI: 10.1016/j.trb.2009.01.004.

[126] G. Wake, N. Boland and L. Jennings. "Mixed integer programming approaches to exact minimization of total treatment time in cancer radiotherapy using multileaf collimators". In: *Computers & Operations Research* 36.3 (2009), pp. 795–810. DOI: 10.1016/j.cor.2007.10.027.

[127] S. Warshall. "A theorem on Boolean matrices". In: *Journal of the ACM* 9.1 (1962), pp. 11–12. DOI: 10.1145/321105.321107.

[128] S. Webb. *Intensity-modulated radiation therapy.* Taylor & Francis, 2001.

[129] S. Webb, T. Bortfeld, J. Stein and D. Convery. "The effect of stair-step leaf transmission on the tongue-and-groove problem in dynamic radiotherapy with a multileaf collimator". In: *Phys. Med. Biol.* 42.3 (1999), pp. 595–602. DOI: 10.1088/0031-9155/42/3/011.

[130] Steve Webb. "Intensity-modulated radiation therapy using only jaws and a mask". In: *Phys. Med. Biol.* 47.2 (2002), p. 257. DOI: 10.1088/0031-9155/47/2/306.

[131] P. Xia and L. Verhey. "Multileaf collimator leaf–sequencing algorithm for intensity modulated beams with multiple static segments". In: *Med. Phys.* 25 (1998), pp. 1424–1434. DOI: 10.1118/1.598315.

Reducing the tongue-and-groove underdosage in MLC shape matrix decomposition[*]

Thomas Kalinowski

Abstract

We present an algorithm for optimal step-and-shoot intensity modulated radiation therapy minimizing tongue-and-groove effects. Adapting the concepts of [7] we characterize the minimal decomposition time as the maximal weight of a path in a properly constructed weighted digraph. We also show that this decomposition time can be realized by a unidirectional plan, thus proving that the algorithm of Kamath *et al.* [9] is monitor unit optimal in general and not only for unidirectional leaf movement. Our characterization of the minimal decomposition time has the advantage that it can be used to derive a heuristic for the reduction of the number of shape matrices following the ideas of [7].

Key words: leaf sequencing, radiation therapy optimization, intensity modulation, multileaf collimator, IMRT

2000 MSC: 92C50, 90C90

1 Introduction

An important method in cancer treatment is the use of high energetic radiation. In order to kill tumor cells the patient is exposed to radiation that is delivered by a linear accelerator whose beam head can be rotated about the treatment couch. Inevitably the healthy tissue surrounding the tumor is also exposed to some radiation. So the problem arises to arrange the treatment in a way such that the tumor receives a sufficiently high uniform dose while the damage to the normal tissue is as small as possible. The standard approach to this problem is as follows. First the patient body is discretized into so called voxels. The set of voxels is then partitioned into three sets: the clinical target volume, the critical structures and the remaining tissue. There are certain dose constraints for each of these parts. Basically the dose in the target volume has to be sufficient to kill the cancerous cells and the dose in the critical structures must not destroy the functionality of the corresponding organs. The determination of a combination of radiation fields is usually done by inverse methods based on certain physical models of how the radiation passes through a body. In the early 1990s the method of intensity modulated radiation therapy (IMRT) was developed in order to obtain additional flexibility. Using a multileaf collimator (MLC) it is possible to form homogeneous fields of different shapes. By superimposing some homogeneous fields an

[*] *Algorithmic Operations Research* **3**(2):165–174, 2008

intensity modulated field is delivered. An MLC consists of two banks of metal leaves which block the radiation and can be shifted to form irregularly shaped beams (Figure 1).

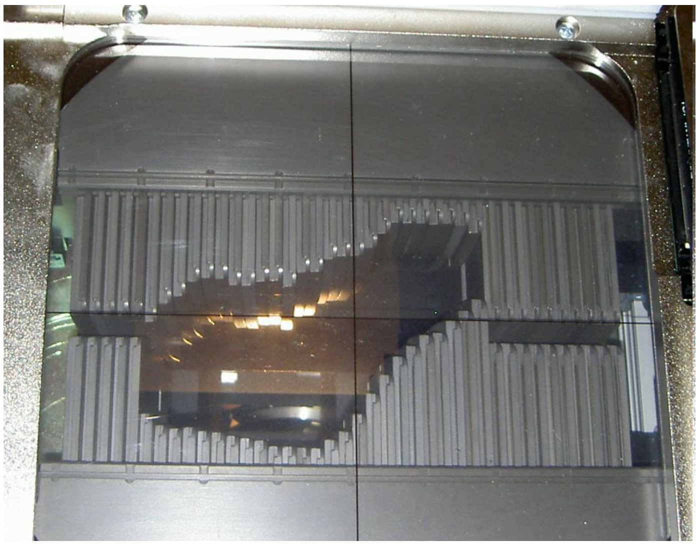

Figure 1: The leaf pairs of a multileaf collimator (MLC).

The most common approach in treatment planning is to divide the optimization into two phases. At first, a set of beam angles and corresponding fluence matrices are determined. In a second step a sequence of leaf positions for the MLC for each of the angles is determined that yields the desired fluence distribution. Very recently there have been attempts to combine both steps into one optimization routine [5, 12].

In this paper we concentrate on the second step, the shape matrix decomposition problem. Suppose we have fixed the beam angles from which the radiation is released, and for each of the beam angles we are given a fluence distribution that we want the patient to be exposed to. After discretizing the beam into bixels we can assume that the fluence distribution is given as a nonnegative integer m nmatrix A. Each row of the matrix corresponds to a pair of leaves of the MLC, and the entry a_{ij} represents the required fluence at bixel (i, j). When the MLC is used in the so called stepandshoot mode the given fluence distribution is realized by superimposing a number of differently shaped homogeneous fields coming from different combinations of the leaf. For example, Figure 2 shows a sequence of leaf positions for the matrix

$$A = \begin{pmatrix} 1 & 3 & 3 & 0 \\ 0 & 2 & 4 & 1 \\ 1 & 1 & 4 & 4 \\ 3 & 3 & 1 & 0 \end{pmatrix} = 2 \cdot \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \tag{1}$$

where the shading indicates the region which is covered by the leaves.

The problem of realizing a given intensity matrix A leads to the problem of representing A as a positive integer combination of certain $(0, 1)$-matrices, called shape matrices, which

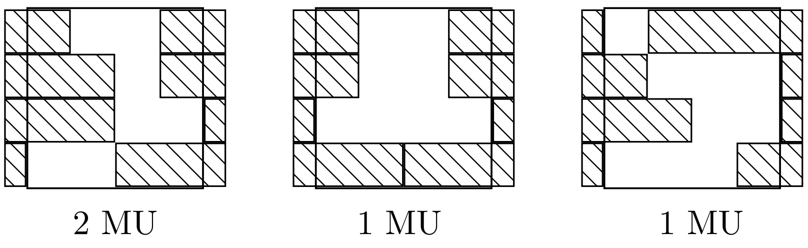

Figure 2: A realization of the intensity matrix A using an MLC. The numbers below the leaf positions indicate the number of monitor units required.

represent the possible leaf positions. So the realization in Figure 2 corresponds to the decomposition in (1). In order to compare different decompositions of an intensity map we consider two quantities (where we adopt the terminology of [1]). For a decomposition $A = \sum_{k=1}^{t} u_k S^{(k)}$, the sum of the coefficients is proportional to the total irradiation time and is called *decomposition time*, $DT = \sum_{k=1}^{1} u_k$. The number k of used shape matrices, called *decomposition cardinality* (DC), influences the total treatment time due to the setup time between the delivery of different shapes. Our objectives in constructing a decomposition are to minimize both DT and DC. In this paper we consider two additional constraints that come from the technical restrictions in many of the available MLCs. The interleaf collision constraint (ICC) forbids the overlapping of opposite leaves in adjacent rows. Another restriction is due to the tongue-and-groove leaf arrangement of the MLCs (see Figure 3). There is a narrow strip in the border region between two adjacent rows that is covered by

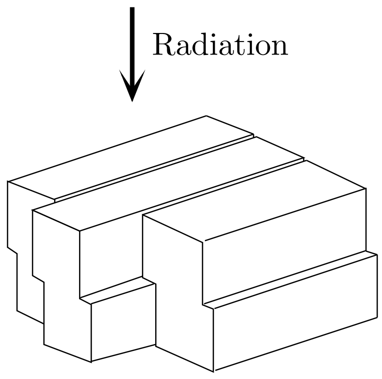

Figure 3: The tongue-and-groove design of the leaves of an MLC.

both leaves and this may lead to underdosage effects in these regions, as is illustrated in Figure 4 for the fluence matrix $A = \left(\begin{smallmatrix} 2 & 3 \\ 3 & 4 \end{smallmatrix}\right)$.

In order to minimize these effects we require that $a_{ij} \leqslant a_{i+1,j}$ implies that bixel $(i+1, j)$ is exposed whenever bixel (i, j) is exposed (similarly for $i-1$ instead of $i+1$). Thus we assure that the overlap region of two bixels always receives the smaller one of the relevant doses. We say that a shape matrix decomposition of A satisfies the tongue-and-

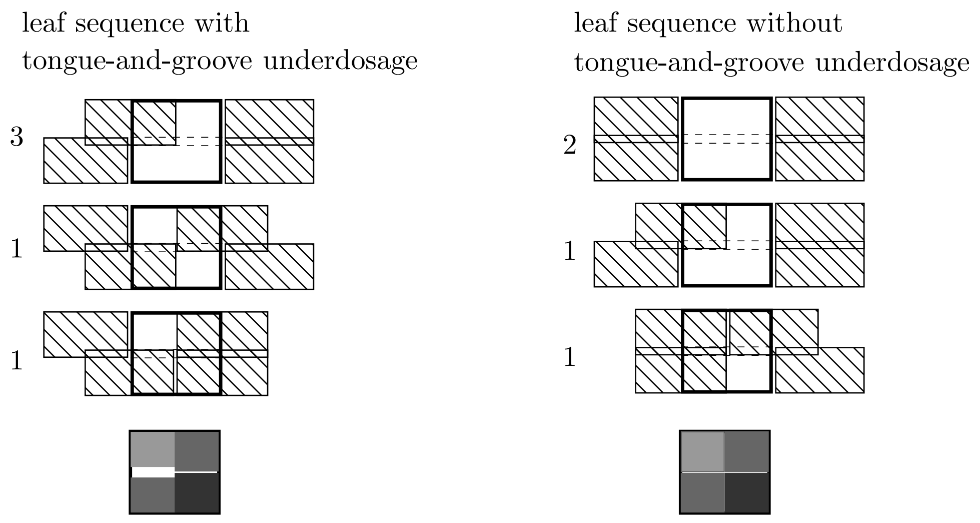

Figure 4: Two different realizations of the same fluence matrix. The numbers next to the leaf positions indicate the irradiation times for the corresponding beams. In the left version the overlap between bixels $(1,1)$ and $(2,1)$ receives no radiation at all.

groove constraint (TGC) if this condition holds for all used shape matrices. This intuitive concept of minimizing underdosage is made more precise in Lemma 1 below. Of course, when the total delivery time increases due to adding the TGC, the total leakage radiation through closed leaves also increases, so there might be a tradeoff between reduction of TG-underdosage and increasing leakage. But numerical experiments indicate that the increase of delivery time compared to the unconstrained case is rather small.

Starting with [3] and [6] several algorithms were proposed for the shape matrix decomposition problem [1, 2, 4, 8, 13, 14]. Methods for eliminating the tongue-and-groove underdosage were presented in [9, 10, 11]. The algorithm from [9] is DT-optimal, as is shown for unidirectional plans in [9] and will be proved without restriction for the leaf movement direction in the present paper. Adapting the approach of [4], in [7] we characterized the minimal DT for the decomposition with ICC as the maximal weight of a path in a certain digraph. In this paper we further modify this approach such that the TGC is included. In addition, we derive a greedy heuristic for the reduction of the number of shape matrices and present some numerical test results.

2 Mathematical formulation of the DT-decomposition problem with ICC and TGC

Throughout the rest of the paper, for a natural number n, $[n]$ denotes the set $\{1, 2, \ldots, n\}$ and for natural numbers $m \leqslant n$, $[m, n]$ denotes the set $\{m, m+1, \ldots, n\}$. In this section we formulate the shape matrix decomposition problem and give a min-max-characterization of the optimal solution very similar to the one used in [7]. We start with a formal characterization of the shape matrices that are allowed in a decomposition of a given intensity matrix A.

Definition 1. Let A be an intensity matrix. A *shape matrix* is an $m \times n$-matrix $S = (s_{ij})$ with entries from $\{0, 1\}$, such that there exist integers l_i, r_i $(i \in [m])$ with the following properties:

$$l_i < r_i \qquad (i \in [m]), \tag{2}$$

$$s_{ij} = \begin{cases} 1 & \text{if } l_i < j < r_i \\ 0 & \text{otherwise} \end{cases} \qquad (i \in [m],\ j \in [n]), \tag{3}$$

$$\text{ICC:} \quad l_i < r_{i+1},\ r_i > l_{i+1} \qquad (i \in [m-1]). \tag{4}$$

A shape matrix is called an *A-shape matrix* if in addition

$$\text{TGC:} \begin{cases} a_{ij} \leqslant a_{i+1,j} \wedge s_{ij} = 1 \implies s_{i+1,j} = 1 \ (i \in [m-1],\ j \in [n]), \\ a_{ij} \leqslant a_{i-1,j} \wedge s_{ij} = 1 \implies s_{i-1,j} = 1 \ (i \in [2, m],\ j \in [n]). \end{cases} \tag{5}$$

A *shape matrix decomposition* of an intensity matrix A is a representation

$$A = \sum_{k=1}^{t} u_k S^{(k)} \tag{6}$$

with positive integers u_k and A-shape matrices $S^{(k)}$ $(k \in [t])$, the decomposition time (DT) of this decomposition is $\sum_{k=1}^{t} u_k$ and shape matrix decomposition problem is to find, for given A, a shape matrix decomposition with minimal DT. We want to give aprecise description of the sense in which condition (5) ensures that the TG-underdosage is minimized. For this purpose we define the tongue and groove error of a decomposition (6) at bixel (i, j) by

$$T(i, j) = \min\{a_{ij}, a_{i+1,j}\} - \sum_{k=1}^{t} u_k s_{ij}^{(k)} s_{i+1,j}^{(k)}.$$

The sum in the right hand side of this equation is the total fluence delivered to the overlap between rows i and $i + 1$ in column j, because this overlap is open in the kth shape if and only if $s_{ij}^{(k)} = s_{i+1,j}^{(k)} = 1$. This sum is at most $\min\{a_{ij}, a_{i+1,j}\}$:

$$a_{ij} = \sum_{k=1}^{t} u_k s_{ij}^{k} \leqslant \sum_{k=1}^{t} u_k s_{ij}^{k} s_{i+1,j}^{k},$$

and similarly for $a_{i+1,j}$. Thus $T(i, j) \geqslant 0$ and every positive value of $T(i, j)$ indicates an underdosage. The following lemma states that the underdosage is minimized for every (i, j) if all the shape matrices satisfy condition (5).

Lemma 1. *For a decomposition $A = \sum_{k=1}^{t} u_k S^{(k)}$ with shape matrices $S^{(k)}$, we have $T(i, j) = 0$ for all $(i, j) \in [m1] \times [n]$ if and only if every shape matrix $S^{(k)}$ (5).*

Proof. By symmetry, we may assume $a_{ij} \leqslant a_{i+1,j}$. We obtain $T(i,j) = 0$ iff and only if

$$a_{ij} = \sum_{k=1}^{t} u_k s_{ij}^{(k)} = \sum_{k=1}^{t} u_k s_{ij}^{(k)} s_{i+1,j}^{(k)},$$

and this is the case if and only if $s_{i+1,j}^{(k)} = 1$ whenever $s_{ij}^{(k)} = 1$. $\square$

In order to characterize the minimal DT we use a similar approach as in [7]. We construct a digraph $G = (V, E)$ as follows.

$$V = \{0,1\} \cup ([m] \times [0, n+1]), \ E = E_1 \cup E_2 \cup E_3 \cup E_4 \ \text{ where}$$

$$\begin{aligned}
E_1 &= \{(0,(i,0)) \ : \ i \in [m]\} \cup \{((i,n+1),1) \ : \ i \in [m]\}, \\
E_2 &= \{((i,j),(i+1,j)) \ : \ i \in [m-1], \ j \in [n-1]\}, \\
E_3 &= \{((i,j),(i-1,j)) \ : \ i \in [2,m], \ j \in [n-1]\}, \\
E_4 &= \{((i,j-1),(i,j)) \ : \ i \in [m], \ j \in [n+1]\}.
\end{aligned}$$

Here 0 and 1 serve as starting and end point, respectively, and the vertices in $[m] \times [n]$ correspond to the entries of A. The two extra columns $[m] \times \{0\}$ and $[m] \times \{n+1\}$ have the purpose to simplify the notation: they assure that for every $(i,j) \in [m] \times [n]$ there are vertices $(i, j-1)$ and $(i, j+1)$. Without this, in several of the arguments below, it would be necessary to treat the first and the last column seperately (then 0 and 1 would have to play the role of $(i,0)$ and $(i,n+1)$, respectively). To be able to treat the first and the n-th column exactly as the remaining columns, we also put $a_{i,0} = a_{i,n+1} = 0$ ($i \in [m]$). Observe that we omit the vertical arcs $((i,n),(i \pm 1, n))$ in the n-th column: this is to make the vertex (i,n) of any $(0,1)$-path unique. This is no loss of generality, because we will see that we are interested only in $(0,1)$-paths of maximal weight, the weights of vertical arcs are nonpositive and the arcs right of column n have weight 0. We define the weight function $w : E \to \mathbb{Z}$:

$$\begin{aligned}
w(0,(i,0)) = w((i,n+1),1) &= 0 && (i \in [m]), \\
w((i,j),(i+1,j)) &= \min\{0, a_{i+1,j} - a_{ij}\} && (i \in [m-1], \ j \in [n-1]), \\
w((i,j),(i-1,j)) &= \min\{0, a_{i-1,j} - a_{ij}\} && (i \in [2,m], \ j \in [n-1]), \\
w((i,j-1),(i,j)) &= \max\{0, a_{ij} - a_{i,j-1}\} && (i \in [m], \ j \in [n+1]).
\end{aligned}$$

Example 1. Figure 5 shows the digraph G corresponding to the matrix $A = \begin{pmatrix} 4&5&0&1&4&5 \\ 2&4&1&3&1&4 \\ 2&3&2&1&2&4 \\ 5&3&3&2&5&3 \end{pmatrix}$.

The following theorem, which is proved in Sections 3 and 4, is the main result of this paper and the basis of the decomposition algorithm.

Theorem 1. *The minimal DT of a shape matrix decomposition of a nonnegative matrix A equals the maximal weight of a $(0,1)-$path in G.*

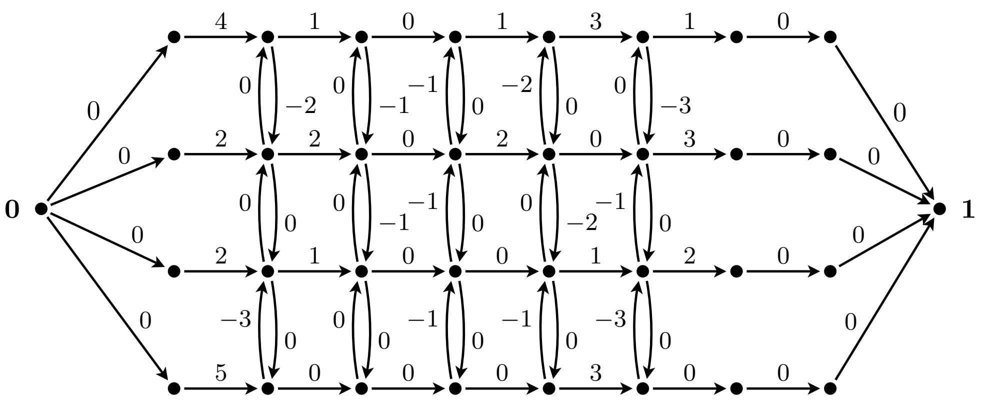
Figure 5: The digraph G corresponding to matrix A.

For convenience we denote this maximal weight by $c(A)$:

$$c(A) = \max\{w(P) \ : \ P \text{ is a } (0,1)-\text{path in } G\}. \tag{7}$$

Observe that the results from [4] and [7] can be seen as characterizations of the minimal DT in terms of maximal path weights for different variants of the problem corresponding to manfacturer specific restrictions.

- MLC without restriction of leaf movement: use the graph G without the vertical arcs.
- MLC with interleaf collision but without tongue and groove: use the same graph G, but with modified weights for the vertical arcs.

So the only case that cannot be treated in this framework is an MLC with tongue and groove and without interleaf collision.

3 The lower bound

In this section we show that the maximal weight of a $(0,1)-$path in G is a lower bound for the DT of a decomposition of A, thus proving the first half of Theorem 1. The basic idea of the proof is a combination of the arguments in [1] and [9], the main difference to [9] being that we do not require the leaf sequence to be unidirectional. For our argument below we need an exact description of how the numbers

$$\alpha(i,j) \leftarrow \max\{w(P) \ : \ P \text{ is a } (0,(i,j))-\text{path in } G\}$$

can be computed. This description is given in Algorithm 1. The underlying principle can be described as follows. We proceed columnwise. Assuming we have already determined the values in column $j-1$ we initialize column j with $\alpha(i,j) \leftarrow \alpha(i,j-1) + w((i,j-1),(i,j))$.

Algorithm 1 Computation of the numbers $\alpha(i,j)$

```
for $i = 1,\dots,m$ do $\alpha(i,1) \leftarrow a_{i1}$
for $j = 2,\dots,n+1$ do
  for $i = 1,\dots,m$ do $\alpha(i,j) \leftarrow \alpha(i,j-1) + w((i,j-1),(i,j))$
  for $i = 2,\dots,m$ do
    if $\alpha(i,j) < \alpha(i-1,j) + w((i-1,j),(i,j))$ then
      $\alpha(i,j) \leftarrow \alpha(i-1,j) + w((i-1,j),(i,j))$
    if $\alpha(i-1,j) < \alpha(ij) + w((i+1,j),(i,j))$ then Update$(i-1)$

Function Update$(k)$
$\alpha(k,j) \leftarrow \alpha(k+1,j) + w((k+1,j),(i,j))$
if $k \geqslant 2$ and $\alpha(k-1,j) < \alpha(kj) + w((k,j),(k-1,j))$ then Update$(k-1)$
```

After that we modify these values in order to satisfy the conditions

$$\begin{aligned} \alpha(i,j) &\geqslant \alpha(i-1,j) + w((i-1,j),(i,j)) && \text{for } i \in [2,m], \\ \alpha(i,j) &\geqslant \alpha(i+1,j) + w((i+1,j),(i,j)) && \text{for } i \in [m-1]. \end{aligned}$$

Now the statement of the following lemma is obvious.

Lemma 2. *Algorithm 1 computes the numbers $\alpha(i,j)$ in time $O(m^2n)$.*

Suppose $A = \sum_{k=1}^{t} S^{(k)}$ is a shape matrix decomposition of A. We characterize the shape matrix $S^{(k)}$ by its left and right leaf positions $l_i^{(k)}$ and $r_i^{(k)}$ ($i \in [m]$). For $(i,j) \in [m] \times [n+1]$, let L_{ij} denote the set of indices k with $l_i^{(k)} < j$, and similarly, let R_{ij} denote the set of indices k with $r_i^{(k)} \leqslant j$. More formally,

$$L_{ij} = \{k \in [t] \ : \ l_i^{(k)} < j\}, \qquad R_{ij} = \{k \in [t] \ : \ r_i^{(k)} \leqslant j\}.$$

Then $|L_{in}|$ is the number of shape matrices which contribute to row i, and $\max_{i\in[m]} |L_{in}|$ is a lower bound for the DT. In the next lemma we collect some simple observations about the sets L_{ij} and R_{ij}.

Lemma 3. *1. For $(i,j) \in [m] \times [n]$, $R_{ij} \subseteq L_{ij}$ and $|L_{ij} \setminus R_{ij}| = a_{ij}$.*

2. For $(i,j) \in [m] \times [n]$, $|L_{ij}| \geqslant |L_{i,j-1}| + \max\{0, a_{ij} - a_{i,j-1}\}$.

3. For $(i,j) \in [2,m] \times [n]$, $R_{i-1,j} \subseteq L_{ij}$ and $R_{ij} \subseteq L_{i-1,j}$.

4. For $(i,j) \in [2,m] \times [n]$,

$$\begin{aligned} a_{i-1,j} \leqslant a_{ij} &\implies L_{i-1,j} \setminus R_{i-1,j} \subseteq L_{ij} \setminus R_{ij} \\ a_{i-1,j} \geqslant a_{ij} &\implies L_{i-1,j} \setminus R_{i-1,j} \supseteq L_{ij} \setminus R_{ij} \end{aligned}$$

Proof. The first statement is a simple consequence of the facts that $r_i^{(k)} \leqslant j$ implies $l_i^{(k)} < j$ and that $s_{ij}^{(k)} = 1$ if and only if $k \in L_{ij} \setminus R_{ij}$. The second statement is clear if $a_{ij} \leqslant a_{i,j-1}$, since $L_{i,j-1} \subseteq L_{ij}$. If $a_{ij} > a_{i,j-1}$, there must be at least $a_{ij} - a_{i,j-1}$ shape matrices $S^{(k)}$ with $s_{ij}^{(k)} = 1$ and $s_{i,j-1}^{(k)} = 0$. For these shape matrices we have $l_i^{(k)} = j-1$, so $k \in L_{ij} \setminus L_{i,j-1}$ and this proves the second claim. Using the ICC we obtain the first inclusion in the third statement:

$$k \in R_{i-1,j} \implies r_{i-1}^{(k)} \leqslant j \implies l_i^{(k)} < j \implies k \in L_{ij},$$

and similarly the second one. For the fourth statement, assume $a_{i-1,j} \leqslant a_{ij}$. Using the TGC we obtain

$$k \in L_{i-1,j} \setminus R_{i-1,j} \implies s_{i-1,j}^{(k)} = 1 \implies s_{ij}^{(k)} = 1 \implies k \in L_{ij} \setminus R_{ij}.$$

This gives the first implication, and the second one is proved similarly. □

Next, we show that the numbers $\alpha_1(i,j)$ bound the cardinalities $|L_{ij}|$ from below.

Lemma 4. *For $(i,j) \in [m] \times [n]$, we have $\alpha(i,j) \leqslant |L_{ij}|$.*

Proof. We proceed by induction. For $j = 1$, $\alpha(i,1) = a_{i1}$ and the claim is obvious, since we need at least a_{i1} shape matrices with $l_i^{(k)} = 0$. Suppose the statement of the lemma is false, and let j be the index of the first column where, for some row i, we have $\alpha(i,j) > |L_{ij}|$. From Lemma 3 we get

$$|L_{ij}| \geqslant |L_{i,j-1}| + \max\{0, a_{ij} - a_{i,j-1}\} \geqslant \alpha(i,j-1) + w((i,j-1),(i,j)).$$

Hence after the initialization of column j in Algorithm 1 (line 3), we still have $\alpha(i,j) \leqslant |L_{ij}|$ for all $i \in [m]$. Now let i be the index of the row where the claim of the lemma is violated for the first time when the algorithm is running. Consider this first violation and assume it occurs in line 6 of Algorithm 1. The case that it occurs in the function `Update`(k) is treated analogously.

Case 1. $a_{i-1,j} \leqslant a_{ij}$. In this case $w((i-1,j),(i,j)) = 0$, hence the updating step of the algorithm is $\alpha(i,j) \leftarrow \alpha(i-1,j)$. By *(iii)* and *(iv)* in Lemma 3 we have

$$R_{i-1,j} \subseteq L_{ij} \qquad \text{and} \qquad L_{i-1,j} \setminus R_{i-1,j} \subseteq L_{ij}.$$

Hence $L_{i-1,j} \subseteq L_{ij}$, and consequently $\alpha(i,j) = \alpha(i-1,j) \leqslant |L_{ij}|$, contradicting the assumption that the step leads to a violation of the claim.

Case 2. $a_{i-1,j} > a_{ij}$. Now the considered step is $\alpha(ij) \leftarrow \alpha(i-1,j) - (a_{i-1,j} - a_{ij})$. Again by *(iii)* and *(iv)* from Lemma 3,

$$R_{i-1,j} \subseteq L_{ij} \qquad \text{and} \qquad L_{ij} \setminus R_{ij} \subseteq L_{i-1,j} \setminus R_{i-1,j}.$$

This implies (using *(i)* from Lemma 3)

$$\begin{aligned} |L_{ij}| \geqslant |R_{i-1,j}| + |L_{ij} \setminus R_{ij}| &= (|L_{i-1,j}| - a_{i-1,j}) + a_{ij} \\ &\geqslant \alpha(i-1,j) - a_{i-1,j} + a_{ij} = \alpha(i,j), \end{aligned}$$

contradicting the assumption. □

Lemma 4 shows that the numbers $\alpha(i,n)$ $(i \in [m])$ are lower bounds for the DT. We state this conclusion as a lemma.

Lemma 5. *For any shape matrix decomposition of an intensity matrix A, we have*

$$DT \geqslant \max_{i\in[m]} \alpha(i,n) = c(A).$$

4 The algorithm

We compute a shape matrix decomposition of A according to Algorithm 2. This is essentially a reformulation of the algorithm of Kamath *et al.* [9], but we need it in this form in order to show that our characterization of the minimal DT in Theorem 1 is correct.

Algorithm 2 DT-optimal shape matrix decomposition

```
for $t = 1,\dots,c(A)$ do
  for $i = 1,\dots,m$ do
    $l_i^{(t)} \leftarrow \max\{j \in [0,n] \ :\ \alpha(i,j) < t \text{ or } j = n\}$
    $r_i^{(t)} \leftarrow \min\{j \in [n+1] \ :\ \alpha(i,j) \geqslant t + a_{ij} \text{ or } j = n+1\}$
  for $(i,j) \in [m]\times[n]$ do
    $s_{ij}^{(t)} \leftarrow \begin{cases} 1 & \text{if } l_i^{(t)} < j < r_i^{(t)} \\ 0 & \text{otherwise} \end{cases}$
```

Lemma 6. *From Algorithm 2 we obtain a shape matrix decomposition of A with $DT = c(A)$.*

Proof. Clearly, the DT of the sum of shape matrices returned by the algorithm is $c(A)$. We divide the proof of the theorem into three parts.

Claim 1. The matrices $S^{(t)}$ form indeed a decomposition of A, that means $A = \sum_{t=1}^{c(A)} S^{(t)}$. Fix some $(i,j) \in [m]\times[n]$. We have

$$\left(l_i^{(t)} < j \iff \alpha(i,j) \geqslant t\right) \quad \text{and} \quad \left(r_i^{(t)} > j \iff \alpha(i,j) < t + a_{ij}\right).$$

Together we obtain $s_{ij}^{(t)} = 1 \iff \alpha(i,j) - a_{ij} < t \leqslant \alpha(i,j)$, hence $\sum_{t=1}^{c(A)} s_{ij}^{(t)} = a_{ij}$, and this proves the claim.

Claim 2. The matrices $S^{(t)}$ satisfy the ICC.
Assume the claim is false. That means, for some $t \in [c(A)]$ and $i \in [m-1]$, $l_i^{(t)} \geqslant r_{i+1}^{(t)}$ or $r_i^{(t)} \leqslant l_{i+1}^{(t)}$. We consider only the first case, since the second one can be treated similarly. We put $j = r_{i+1}^{(t)}$. By construction and our assumption, we have

$$\alpha(i,j) < t \qquad \text{and} \quad \alpha(i+1,j) \geqslant t + a_{i+1,j}.$$

But on the other hand,

$$\alpha(i,j) \geqslant \alpha(i+1,j) + w((i+1,j),(i,j)) = \alpha(i+1,j) + \min\{0, a_{ij} - a_{i+1,j}\},$$

thus $\alpha(i,j) \geqslant t$, and this contradiction proves the claim.

Claim 3. The matrices $S^{(t)}$ satisfy the TGC.
Suppose $a_{ij} \leqslant a_{i+1,j}$ and $s_{ij}^{(t)} = 1$, or equivalently $l_i^{(t)} < j < r_i^{(t)}$. By construction, this implies

$$t \leqslant \alpha(i,j) < t + a_{ij}. \tag{8}$$

Observe, that

$$w((i,j),(i+1,j)) = 0 \quad \text{and} \quad w((i+1,j),(i,j)) = a_{ij} - a_{i+1,j},$$

since $a_{ij} \leqslant a_{i+1,j}$. Using (8), we obtain the bounds

$$\alpha(i+1,j) \geqslant \alpha(i,j) + w((i,j),(i+1,j)) = \alpha(i,j) \geqslant t \qquad \text{and}$$

$$\begin{aligned} t + a_{ij} &> \alpha(i,j) \geqslant \alpha(i+1,j) + w((i+1,j),(i,j)) \\ t + a_{ij} &> \alpha(i+1,j) + (a_{ij} - a_{i+1,j}). \end{aligned}$$

Hence $t \leqslant \alpha(i+1,j) < t + a_{i+1,j}$, and according to Algorithm 2, $s_{i+1,j}^{(t)} = 1$. Thus the first TGC is satisfied, and the second one is proved similarly. □

Together, Lemmas 5 and 6 prove Theorem 1.

5 Minimizing the number of shape matrices

The problem of minimizing the number of shape matrices is NP-hard even for a single row intensity matrix [1]. So it is natural to look for a heuristic approach that yields decompositions with a small number of shape matrices in a reasonable time even if optimality is not always reached. In [7] we used a greedy strategy in order to find a decomposition with minimal DT and a small number of shape matrices for MLCs with ICC but neglecting the TGC. This method can be modified to respect the TGC. In order to characterize the maximal coefficient u for which there is an A-shape matrix S, such that uS can be a term in a DT-optimal decomposition of A, we need the following lemma.

Lemma 7. *Let $A = \sum_{t=1}^{k} u_t S^{(t)}$ be a decomposition of A (i.e. the $S^{(t)}$ are A-shape matrices), and put $A^{(0)} = A$ and $A^{(t)} = A - \sum_{t'=1}^{t} u_{t'} S^{(t')}$ for $t \in [k]$. Then, for every $t \in [k]$ we have*

- $s_{ij}^{(t)} = 1$ *and* $s_{i+1,j}^{(t)} = 0 \;\Rightarrow\; a_{ij}^{(t-1)} \geqslant a_{i+1,j}^{(t-1)} + u \quad (i \in [m-1],\ j \in [n])$,
- $s_{ij}^{(t)} = 1$ *and* $s_{i-1,j}^{(t)} = 0 \;\Rightarrow\; a_{ij}^{(t-1)} \geqslant a_{i-1,j}^{(t-1)} + u \quad (i \in [2,m],\ j \in [n])$.

Informally speaking, if we consider the sequence of matrices starting with A and subtracting one by one the $S^{(t)}$ taking $S^{(t)}$ exactly u_t times, the lemma claims that in each step we subtract an A'-shape matrix, where A' is the resulting matrix after the previous step.

Proof. Assume the contrary and let t be the first index where one of the two claims fails to be true. By symmetry, we assume

$$s_{ij}^{(t)} = 1, \quad s_{i+1,j}^{(t)} = 0, \quad a_{ij}^{(t-1)} < a_{i+1,j}^{(t-1)} + u.$$

Since $S^{(t)}$ is an A-shape matrix, the TGC implies $a_{ij} > a_{i+1,j}$. From our assumption we obtain $a_{ij}^{(t)} < a_{i+1,j}^{(t)}$, hence

$$s_{ij}^{(t')} = 0 \quad \text{and} \quad s_{i+1,j}^{(t')} = 1$$

for some $t' > t$, contradicting the assumption that $S^{(t')}$ is an A-shape matrix. $\square$

We call a pair (u, S) of a positive integer u and an A-shape matrix S an *admissible segmentation pair*, if

- $A - uS$ is nonnegative,
- $s_{ij} = 1$ and $s_{i+1,j} = 0 \quad \Rightarrow \quad a_{ij} \geqslant a_{i+1,j} + u \qquad (i \in [m-1],\ j \in [n])$,
- $s_{ij} = 1$ and $s_{i-1,j} = 0 \quad \Rightarrow \quad a_{ij} \geqslant a_{i-1,j} + u \qquad (i \in [2, m],\ j \in [n])$,
- $c(A - uS) = c(A) - u$.

Now we proceed exactly as in [7]: we find an admissible segmentation pair (u, S) with maximal u and continue with $A - uS$ until we reach the zero matrix. In order to derive an upper bound for the coefficient u in an admissible segmentation pair (u, S), we use an idea from [2] and identify the set of segments with the set of paths from D to D' in the layered digraph $\Gamma = (W, F)$, constructed as follows. The vertices in the $i-$th layer correspond to the possible leaf positions in row i $(1 \leqslant i \leqslant m)$ and two additional vertices D and D' are added:

$$W = \{(i, l, r) : i \in [m],\ l \in [0, n],\ r \in [l+1, \dots, n+1]\} \cup \{D, D'\}.$$

Between two vertices (i, l, r) and $(i+1, l', r')$ there is an arc if the corresponding leaf positions are consistent with the ICC, i.e. if $l' < r$ and $r' > l$. In addition, the arc set F contains all arcs from D to the first layer and from the last layer m to D', so

$$F = F_+(D) \cup F_-(D') \cup \bigcup_{i=1}^{m-1} F_+(i), \text{ where}$$

$$\begin{aligned} F_+(D) &= \{(D, (1, l, r)) \ :\ (1, l, r) \in W\}, \\ F_-(D) &= \{((m, l, r), D') \ :\ (m, l, r) \in W\}, \\ F_+(i) &= \{((i, l, r), (i+1, l', r')) \ :\ l' < r,\ r' > l\}. \end{aligned}$$

There is a bijection between the possible leaf positions and the paths from D to D' in Γ. This is illustrated in Fig. 6 which shows the paths in Γ for $m=4$, $n=2$, corresponding to the shape matrices

$$\begin{pmatrix} 1&0\\0&1\\1&1\\1&0 \end{pmatrix} \text{ (straight lines) and } \begin{pmatrix} 0&1\\1&1\\1&0\\0&1 \end{pmatrix} \text{ (dashed lines).}$$

For each vertex (i,l,r) let $u_0(i,l,r)$ denote an upper bound for the coefficient in an ad-

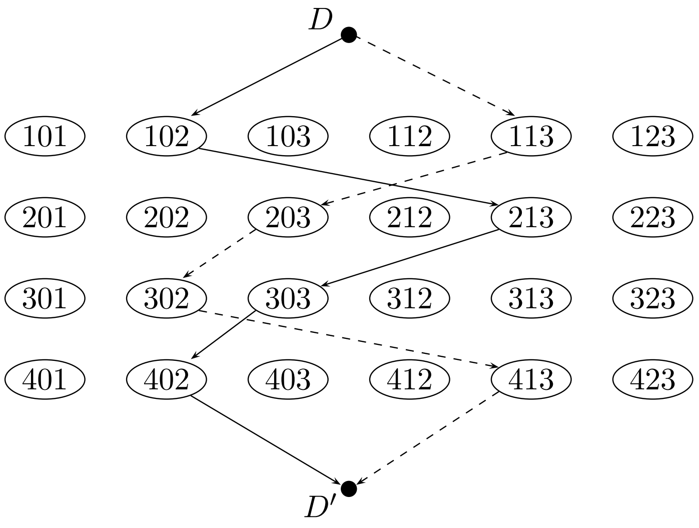

Figure 6: The vertices of Γ for $m=4$, $n=2$ and two (D,D')-paths.

missible segmentation pair (u,S) where S is a shape matrix with $l_i=l$ and $r_i=r$. Then any admissible segmentation pair (u,S) corresponds to a path

$$D,(1,l_1,r_1),(2,l_2,r_2),\dots,(m,l_m,r_m),D'$$

with the following properties.

- For $i\in[m]$, $u_0(i,l_i,r_i)\geqslant u$.
- For $i\in[m-1]$ and $j\in[n]$,

$$\begin{aligned} l_i<j\leqslant l_{i+1} \text{ or } r_{i+1}\leqslant j<r_i &\quad\Longrightarrow\quad a_{ij}\geqslant a_{i+1,j}+u,\\ l_{i+1}<j\leqslant l_i \text{ or } r_i\leqslant j<r_{i+1} &\quad\Longrightarrow\quad a_{i+1,j}\geqslant a_{ij}+u. \end{aligned}$$

If we have good upper bounds $u_0(i,l,r)$, this yields a considerable reduction of the set of shape matrices that have to be considered in the search an admissible segmentation pair. In our implementation we used the bound from the following lemma.

Lemma 8. *For $i \in [m]$, let $g_i = c(A) - \sum_{j=1}^{n} \max\{0, a_{ij} - a_{i,j-1}\}$, and suppose (u, S) is an admissible segmentation pair with parameters l_i, r_i $(i \in [m])$. Then for $i \in [m]$,*

$$u \leqslant g_i \qquad \text{if } r_i = l_i + 1 \tag{9}$$

$$\begin{aligned} u \leqslant \min\{&g_i + \max\{0, a_{i,r-1} - a_{ir}\}, g_i + \max\{0, a_{i,l+1} - a_{il}\}, \\ &\tfrac{1}{2}\left(g_i + \max\{0, a_{i,l+1} - a_{il}\} + \max\{0, a_{i,r-1} - a_{ir}\}\right)\} \qquad \text{if } r_i > l_i + 1. \end{aligned} \tag{10}$$

Proof. For brevity of notation, let $d_{ij} = \max\{0, a_{ij} - a_{i,j-1}\}$ for $(i, j) \in [m] \times [n]$. Observe that $\sum_{j=1}^{n} d_{ij}$ is just the weight of the path

$$0, (i, 0), (i, 1), \dots, (i, n), (i, n + 1), 1$$

in G. The fact that (u, S) is an admissible segmentation pair implies,

$$\sum_{j=1}^{n} d'_{ij} \leqslant c(A) - u, \tag{11}$$

where $A' = (a'_{ij}) = A - uS$ and $d'_{ij} = \max\{0, a'_{ij} - a'_{i,j-1}\}$. If $r_i = l_i + 1$, $a'_{ij} = a_{ij}$ for all j and this implies (9). For (10), observe that

$$\begin{aligned} d'_{i,l_i+1} &= d_{i,l_i+1} - \min\{u, d_{i,l_i+1}\}, \\ d'_{i,r_i} &= d_{i,r_i} + \max\{0, u - \max\{0, a_{i,r_i-1} - a_{ir_i}\}\}, \\ d'_{ij} &= d_{ij} \qquad \text{for } j \notin \{l_{i+1}, r_i\}. \end{aligned}$$

With (11) we obtain

$$\sum_{j=1}^{n} d_{ij} - \min\{u, d_{i,l_i+1}\}\} + \max\{0, u - \max\{0, a_{i,r_i-1} - a_{ir_i}\}\}\} \leqslant c(A) - u,$$

hence

$$u - \min\{u, d_{i,l_i+1}\} + \max\{0, u - \max\{0, a_{i,r_i-1} - a_{ir_i}\}\} \leqslant g_i$$

and this implies (10). □

Algorithm 3 summarizes our greedy approach for the construction of a *DT*-optimal shape matrix decomposition with a small *DC*.

Algorithm 3 DT-optimal shape matrix decomposition with reduced DC

```
while $A \neq 0$ do
  Determine the complexity $c(A)$ and the numbers $u_0(i,l,r)$ for
  $i \in [m]$, $l \in [0,n]$, $r \in [n+1]$ according to Lemma 8
  $u \leftarrow \max\{k$ : There is a path $P$ from $D$ to $D'$ in $\Gamma$
                    with $u_0(i,l,r) \geqslant k$ for all $(i,l,r) \in P\}$
  complete←false
  while (not complete) do
    for the paths $P$ in $\Gamma$ with $u_0(i,l,r) \geqslant k$ for all $(i,l,r) \in P$ do
      Let $S$ be the shape matrix corresponding to $P$
      if $(u,S)$ is an admissible segmentation pair then
        complete←true
    if (not complete) then $u \leftarrow u-1$
  $A \leftarrow A - S$
```

6 Test results

We implemented Algorithm 3 in C++ and computed decompositions for $15 \times 15-$matrices, where the entries are chosen uniformly and independently from $\{0,\dots,L\}$. Table 1 shows the results for different values of L, where for each row of the table we averaged over 1000 sample matrices. In the second column we have the average DT, which is the same as for

L	DT	DC (plain)	DC (reduced)	CPU time (sec)
4	21.2	21.0	18.0	93
7	34.9	34.2	24.1	276
10	48.2	46.3	28.1	399
13	61.7	57.9	31.2	556
16	74.8	68.2	33.5	647

Table 1: Test results for random 15×15-matrices with entries from $\{0,\dots,L\}$.

the algorithm of Kamath *et al.* [9]. The third column shows the DC of a decomposition according to Algorithm 2 (or equivalently the algorithm of Kamath et al.). Clearly, this algorithm just aims at minimizing the DT without taking the DC into account, hence the DC almost equals the DT. In the fourth column we have the DC of the decompositions according to Algorithm 3, and we see that this approach yields considerable savings in terms of the number of used shape matrices. The CPU times (on a 2GHz workstation with 2GB RAM) in the third columns show that the algorithm is practicable for intensity matrices of the considered size (note that the times are for the decomposition of 1000 matrices, so the average time for a single matrix is still below a second). But of course the backtracking for determining the maximal value of u becomes very slow for larger matrices, and more efficient methods are needed for matrix dimensions of practical relevance. In order to evaluate the influence of the TGC, in Table 2 we compare results for different types of constraints.

L	unconstrained		only ICC		ICC and TGC	
	DT	DC	DT	DC	DT	DC
4	17.9	10.9	19.5	14.5	21.2	18.0
7	29.5	13.1	31.7	18.2	34.9	24.1
10	40.9	14.7	43.8	20.7	48.2	28.1
13	52.4	15.8	55.7	22.5	61.7	31.2
16	63.8	16.8	67.7	24.0	74.8	33.5

Table 2: Test results for random 15×15-matrices with entries from $\{0, 1, \ldots, L\}$ for different types of constraints.

Finally, we also tested our algorithm with 13 clinical matrices, each with 10 fluence levels. The results are shown in Table 3. The computation times for these matrices were negligible (less than a second).

		unconstrained		only ICC		ICC and TGC	
no.	size	DT	DC	DT	DC	DT	DC
1	10×11	16	8	16	8	17	11
2	10×9	16	7	16	8	19	13
3	9×9	20	8	20	10	20	12
4	9×9	19	8	19	11	21	15
5	10×8	15	7	18	9	19	11
6	9×9	17	9	17	9	19	11
7	10×8	18	7	18	10	21	12
8	14×12	22	9	22	10	25	14
9	14×10	26	10	30	15	34	19
10	14×10	22	9	23	13	28	15
11	15×10	22	10	22	11	25	16
12	15×11	23	10	23	12	23	16
13	14×10	23	9	24	11	27	17

Table 3: Test results for random 15×15-matrices with entries from $\{0, 1, \ldots, L\}$ for different types of constraints.

Clearly, the addition of the TGC causes an increase in the DT and in the DC. Further investigations are necessary in order to evaluate the potential tradeoff between DT (and corresponding leakage) and tongue-and-groove underdosage.

7 Conclusion

We have presented an algorithm for MLC shape matrix decomposition taking into account the interleaf collision constraint and eliminating tongue-and-groove underdosage effects. We proved that our algorithm is optimal with respect to the total number of monitor units, thus completing the argument of [9] where the optimality was proved only for unidirectional schedules. In addition, we derived a heuristic approach to the reduction of the number of shape matrices. Two open questions arise immediately and are the subject of ongoing research. 1. Is there a nice characterization for the minimal decomposition time if we have no interleaf constraint but still want to eliminate tongue-and-groove underdosage? 2. What about a computationally more efficient heuristic for the decomposition cardinality?

References

[1] D. Baatar, H.W. Hamacher, M. Ehrgott and G.J. Woeginger. "Decomposition of integer matrices and multileaf collimator sequencing". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 6–34. DOI: `10.1016/j.dam.2005.04.008`.

[2] N. Boland, H. W. Hamacher and F. Lenzen. "Minimizing beam-on time in cancer radiation treatment using multileaf collimators". In: *Networks* 43.4 (2004), pp. 226–240. ISSN: 0028-3045. DOI: `10.1002/net.20007`.

[3] T.R. Bortfeld, D.L. Kahler, T.J. Waldron and A.L. Boyer. "X–ray field compensation with multileaf collimators". In: *Int. J. Radiat. Oncol. Biol. Phys.* 28 (1994), pp. 723–730. DOI: `10.1016/0360-3016(94)90200-3`.

[4] K. Engel. "A new algorithm for optimal multileaf collimator field segmentation". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 35–51. DOI: `10.1016/j.dam.2004.10.007`.

[5] K. Engel and E. Tabbert. "Fast Simultaneous Angle, Wedge, and Beam Intensity Optimization in Inverse Radiotherapy Planning". In: *Optimization and Engineering* 6.4 (2005), pp. 393–419. DOI: `10.1007/s11081-005-2065-3`.

[6] J.M. Galvin, X.G. Chen and R.M. Smith. "Combining multileaf fields to modulate fluence distributions". In: *Int. J. Radiat. Oncol. Biol. Phys.* 27 (1993), pp. 697–705. DOI: `10.1016/0360-3016(93)90399-G`.

[7] T. Kalinowski. "A duality based algorithm for multileaf collimator field segmentation with interleaf collision constraint". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 52–88. DOI: `10.1016/j.dam.2004.10.008`.

[8] S. Kamath, S. Sahni, J. Li, J. Palta and S. Ranka. "Leaf sequencing algorithms for segmented multileaf collimation". In: *Phys. Med. Biol.* 48.3 (2003), pp. 307–324. DOI: `10.1088/0031-9155/48/3/303`.

[9] S. Kamath, S. Sartaj, J. Palta, S. Ranka and J. Li. "Optimal leaf sequencing with elimination of tongue–and–groove underdosage". In: *Phys. Med. Biol.* 49 (2004), N7–N19. DOI: `10.1088/0031-9155/49/3/N01`.

[10] S. Luan, C. Wang, D.Z. Chen, X.S. Hu, S.A. Naqvi, X. Wu and C.X. Yu. “An improved MLC segmentation algorithm and software for step-and-shoot IMRT delivery without tongue-and-groove error”. In: *Med. Phys.* 33 (2006), pp. 1199–1212. DOI: `10.1118/1.2188823`.

[11] W. Que, J. Kung and J. Dai. “’Tongue-and-groove’effect in intensity modulated radiotherapy with static multileaf collimator fields”. In: *Phys. Med. Biol.* 49.3 (2004), pp. 399–405. DOI: `10.1088/0031-9155/49/3/004`.

[12] H.E. Romeijn, R.K. Ahuja, J.F. Dempsey and A. Kumar. “A Column Generation Approach to Radiation Therapy Treatment Planning Using Aperture Modulation”. In: *SIAM J. on Optimization* 15.3 (2005), pp. 838–862. ISSN: 1052-6234. DOI: `10.1137/040606612`.

[13] R.A.C. Siochi. “Minimizing static intensity modulation delivery time using an intensity solid paradigm”. In: *Int. J. Radiat. Oncol. Biol. Phys.* 43 (1999), pp. 671–680. DOI: `10.1016/S0360-3016(98)00430-1`.

[14] P. Xia and L. Verhey. “Multileaf collimator leaf–sequencing algorithm for intensity modulated beams with multiple static segments”. In: *Med. Phys.* 25 (1998), pp. 1424–1434. DOI: `10.1118/1.598315`.

Approximated MLC shape matrix decomposition with interleaf collision constraint*

Thomas Kalinowski   Antje Kiesel

Abstract

Shape matrix decomposition is a subproblem in radiation therapy planning. A given fluence matrix A has to be decomposed into a sum of shape matrices corresponding to homogeneous fields that can be shaped by a multileaf collimator (MLC). We solve the problem of minimizing the delivery time for an approximation of A satisfying certain prescribed bounds, under the additional condition that the used MLC requires the interleaf collision constraint.

Key words: Intensity modulated radiation therapy (IMRT); multileaf collimator; combinatorial optimization; programming involving graphs

2010 MSC: 90C35, 92C50, 90C90

1 Introduction

In modern cancer therapy radiation is used to destroy the tumor tissue. At the same time one has to minimize the damage to the healthy tissue, and in particular to sensible structures or organs at risk. Intensity modulated radiation therapy was introduced in order to improve the quality of radiation treatment. In clinical practice it is common to use a linear accelerator which can release radiation from different directions (Figure 1). In addition, a multileaf collimator (MLC) (Figure 2) can be used to protect certain parts of the irradiated area.

For the treatment planning, the first step is to determine a set of directions (typically 3–9), from which radiation is released, given by positions of the isocenter, table angles and gantry angles [5, 12]. In a second step, for each direction the fluence distribution is optimized, subject to required dose distribution in the target. The final step is to determine, for each fluence distribution, a corresponding sequence of MLC leaf positions. Recently, there have been attempts to formulate the optimization problem more globally [5, 13], but most of the widely used treatment planning systems model the three steps independently. In this paper we consider the last step for the MLC in the so called *step-and-shoot* mode. This means the radiation is switched off while the leaves are moving, and so the generated intensity modulated field is just a superposition of finitely many homogeneous fields which are shaped by the MLC. The two most important objectives in the optimization problem

*Algorithmic Operations Research **4**(1):49–57, 2009

Figure 1: A linear accelerator.

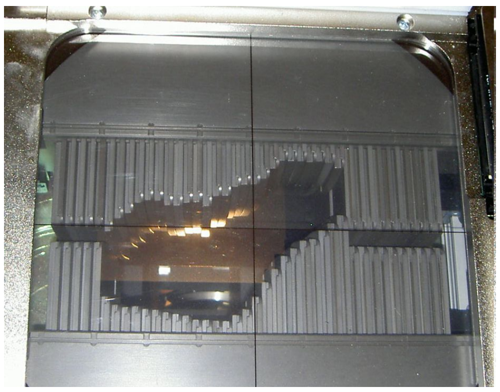
Figure 2: Leaf pairs of a multileaf collimator.

are the total irradiation time, or delivery time (DT), and the number of used fields, or decomposition cardinality (DC). Starting with [2] and [6] there have been proposed several algorithms for this problem [3, 10, 14, 15], taking into account additional machine dependent constraints as the interleaf collision constraint [1, 7] or the tongue-and-groove constraint [11] (see [9] or [8] for a survey).

All of these algorithms start with the given fluence matrix A and construct a sequence of leaf positions realizing this matrix. But from a practical point of view there seem to be some doubts if it is reasonable to consider every entry a_{ij} as fixed once and for all. First, the matrix A is a result of numerical computations which are based on simplified physical models of how the radiation passes through the patients body, and second, the representation of A as a superposition of homogeneous fields is also based on model assumptions which are not strictly correct, for instance the dose delivered to an exposed bixel depends on the shape of the field. So it might be sufficient, to realize (in our model) a matrix that is close to A. It is a natural question, how much the delivery time can be reduced by giving only an approximate representation of A satisfying certain minimum and maximum dose constraints. As an immediate consequence, the next problem arises: find an approximation with this optimal DT which is as close as possible to A. These questions have been answered for unconstrained MLCs in [4], and in the present paper we generalize the ideas from this reference to MLCs with interleaf collision constraint.

In Section 2 we give an precise statement of the problem, Section 3 reviews an exact algorithm for shape matrix decomposition with interleaf collision constraint, in Section 4 we present our graph-theoretical characterization of the minimal DT of an approximation with a constructive proof, in Section 5 we show how the total change can be reduced heuristically, and the final Section 6 contains some test results.

2 Notation and problem formulation

Throughout the rest of the paper, for a natural number n, $[n]$ denotes the set $\{1, 2, \dots, n\}$ and for integers $m < n$, $[m, n]$ denotes the set $\{m, m+1, \dots, n\}$. For integers a, we also use the notation a_+ for the nonnegative part, defined by

$$a_+ = \begin{cases} a & \text{if } a \geqslant 0, \\ 0 & \text{otherwise.} \end{cases}$$

Our starting point is an $m \times n-$matrix A with nonnegative integer entries. The entry a_{ij} represents the desired fluence at bixel (i, j). In addition, for each entry (i, j) we have lower and upper bounds $\underline{a_{ij}}$ and $\overline{a_{ij}}$, such that

$$0 \leqslant \underline{a_{ij}} \leqslant a_{ij} \leqslant \overline{a_{ij}}.$$

Definition 1 (Feasible Approximation)**.** Any integer matrix B with

$$\underline{a_{ij}} \leqslant b_{ij} \leqslant \overline{a_{ij}}$$

is called *feasible approximation* of A. The *total change* $TC(B)$ of a feasible approximation B is defined by

$$TC(B) = \sum_{i=1}^{m} \sum_{j=1}^{n} |b_{ij} - a_{ij}|.$$

The homogeneous fields that can be shaped by the MLC are described by binary matrices of size $m \times n$ which we call *shape matrices.*

Definition 2 (Shape matrix)**.** An $m \times n$ matrix S is a *shape matrix* if there are pairs of integers (l_i, r_i) $(i = 1, \dots, m)$, such that the following conditions are satisfied:

1. $s_{ij} = \begin{cases} 1 & \text{if } l_i < j < r_i, \\ 0 & \text{otherwise.} \end{cases}$
2. $l_i < r_{i+1}$ and $r_i > l_{i+1}$ for all $i \in [m-1]$.

The second condition in Definition 2 is called interleaf collision constraint (ICC). It ensures the left leaf of row i and the right leaf of row $i \pm 1$ do not overlap, which is required by some widely used MLCs, for instance the Elekta MLC. An MLC leaf sequence for A corresponds to a representation of A as a weighted sum of shape matrices.

Definition 3 (Shape matrix decomposition)**.** A *shape matrix decomposition* of A is a representation of A as a positive integer combination of shape matrices

$$A = \sum_{t=1}^{k} u_t S^{(t)}.$$

The *delivery time* (DT) of this decomposition is just the sum of the coefficients,

$$DT = \sum_{t=1}^{k} u_t.$$

Example 1. For the shape matrix decomposition

$$\begin{pmatrix}1&3&3&0\\0&2&4&1\\1&1&4&4\\3&3&1&0\end{pmatrix} = 2\cdot\begin{pmatrix}0&1&1&0\\0&0&1&0\\0&0&1&1\\1&1&0&0\end{pmatrix} + \begin{pmatrix}0&1&1&0\\0&1&1&0\\1&1&1&1\\0&0&0&0\end{pmatrix} + \begin{pmatrix}1&0&0&0\\0&1&1&1\\0&0&1&1\\1&1&1&0\end{pmatrix}$$

we have $DT = 4$.

Now we formulate three optimization problems.

MinDT. Find a shape matrix decomposition $A = \sum_{t=1}^{k} u_t S^{(t)}$ such that $DT = \sum_{t=1}^{k} u_t$ is minimal.

Approx-MinDT. Find a feasible approximation B and a shape matrix decomposition $B = \sum_{t=1}^{k} u_t S^{(t)}$ such that $DT = \sum_{t=1}^{k} u_t$ is minimal.

Approx-MinDT-TC. Find a feasible approximation B and a shape matrix decomposition $B = \sum_{t=1}^{k} u_t S^{(t)}$ such that $DT = \sum_{t=1}^{k} u_t$ is minimal, and under this condition $TC(B)$ is minimal.

The first problem **MinDT** is the exact decomposition problem which can be solved by several efficient algorithms [1, 7, 10]. The idea underlying one of these algorithms is reviewed in the next section because it is the basis for our approach to the second problem **Approx-MinDT**. Finally, we observe that the second part of each of the problems **Approx-MinDt** and **Approx-MinDT-TC**, the search for the shape matrix decomposition, can be ignored safely, because, once the matrix B is fixed, we can apply any exact decomposition algorithm to complete the task.

3 Review of the exact decomposition

The basis of our approach is a characterization of the minimal DT of a decomposition with ICC as the maximal weight of a $q-s-$path in the following digraph $G = (V, E)$ [7, 8].

$$\begin{aligned}
V &= \{q, s\} \cup [m] \times [0, n+1],\\
E &= \{(q,(i,0)) \ : \ i \in [m]\} \cup \{((i,n+1),s) \ : \ i \in [m]\}\\
&\quad \cup \{((i,j),(i,j+1)) \ : \ i \in [m],\ j \in [0,n]\}\\
&\quad \cup \{((i,j),(i+1,j)) \ : \ i \in [m-1],\ j \in [n]\}\\
&\quad \cup \{((i,j),(i-1,j)) \ : \ i \in [2,m],\ j \in [n]\}\,.
\end{aligned}$$

In order to avoid case distinctions, we add two columns to our matrix and put

$$a_{i0} = a_{i,n+1} = 0 \qquad (i \in [m]).$$

Now we can define arc weights by

$$
\begin{aligned}
w(q,(i,0)) = w((i,n+1),s) &= 0 && (i \in [m]) \\
w((i,j-1),(i,j)) &= \max\{0, a_{i,j} - a_{i,j-1}\} && (i \in [m],\ j \in [n+1]) \\
w((i,j),(i+1,j)) &= -a_{ij} && (i \in [m-1],\ j \in [n]) \\
w((i,j),(i-1,j)) &= -a_{ij} && (i \in [2,m],\ j \in [n]).
\end{aligned}
$$

We call this graph the *DT-ICC-graph* for A. Figure 3 shows the DT-ICC-graph for the matrix

$$
A = \begin{pmatrix} 4 & 5 & 0 & 1 & 4 & 5 \\ 2 & 4 & 1 & 3 & 1 & 4 \\ 2 & 3 & 2 & 1 & 2 & 4 \\ 5 & 3 & 3 & 2 & 5 & 3 \end{pmatrix}.
$$

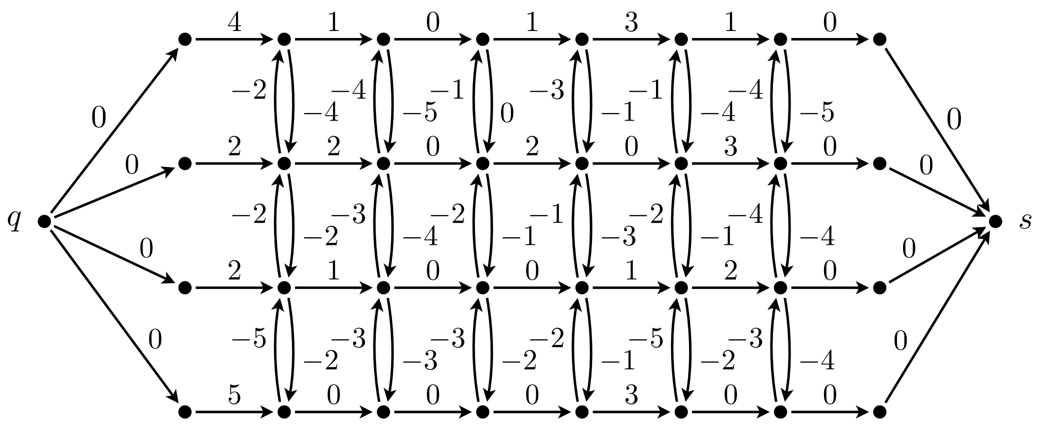

Figure 3: The DT-ICC-Graph for matrix A.

Definition 4. Let A be an intensity matrix, and let G be the DT-ICC-graph for A. The maximal weight of a $q-s-$path in G is called *ICC-complexity* of A and denoted by $c(A)$. More formally,

$$
c(A) = \max\{w(P)\ :\ P \text{ is a } q-s-\text{path in } G.\}.
$$

Using this definition the main result of [7] can be formulated as follows.

Theorem 1. *The minimal DT of a decomposition of A with ICC equals* $c(A)$.

4 Approximation

To simplify our notation, for each $(i,j) \in [m] \times [n]$ we introduce the interval of acceptable fluence values

$$
I_{ij} = \left[\underline{a_{ij}}, \overline{a_{ij}}\right], \qquad \underline{a_{ij}} \leqslant a_{ij} \leqslant \overline{a_{ij}}.
$$

We want to find a matrix B such that

$$b_{ij} \in I_{ij} \quad \text{for } (i,j) \in [m] \times [n] \qquad \text{and} \qquad c(B) \to \min.$$

We follow an approach from [4] and replace every vertex $(i,j) \in [m] \times [n]$ by $|I_{ij}|$ copies, i.e. by the set

$$V_{ij} = \{(i,j)\} \times I_{ij}.$$

In order to avoid case distinctions in the discussion below we also replace the vertices in columns 0 and $n+1$ by

$$V_{i0} = \{(i,0,0)\} \quad \text{and} \quad V_{i,n+1} = \{(i,n+1,0)\}.$$

An arc $((i,j),(i,j+1))$ in the DT-ICC-graph G is replaced by the complete bipartite graph $V_{ij} \times V_{i,j+1}$, and similarly for the arcs $((i,j),(i \pm 1,j))$. The weights of the arcs $((i,j,k),(i,j+1,l))$ should model the approximation matrix B if we choose $b_{ij} = k$ and $b_{i,j+1} = l$, and similarly for the other arc types. Hence we define the arc weights by

$$\begin{aligned}
w(q,(i,0,0)) &= 0 && (i \in [m]),\\
w((i,n+1,0),s) &= 0 && (i \in [m]),\\
w((i,0,0),(i,1,k)) &= k && (i \in [m], k \in I_{i1}),\\
w((i,n,k),(i,n+1,0)) &= 0 && (i \in [m], k \in I_{in}),\\
w((i,j-1,k),(i,j,l)) &= (l-k)_+ && (i \in [m], j \in [n], k \in I_{i,j-1}, l \in I_{ij}),\\
w((i,j,k),(i+1,j,l)) &= -k && (i \in [m-1], j \in [n], k \in I_{ij}, l \in I_{i+1,j}),\\
w((i,j,k),(i-1,j,l)) &= -k && (i \in [2,m], j \in [n], k \in I_{ij}, l \in I_{i-1,j}).
\end{aligned}$$

In order to determine the minimal complexity of an approximation matrix we compute numbers $W(i,j,k)$ such that

$$\begin{aligned}
W(i,j,k) = \max\big\{ & \min_l W(i,j-1,l) + (k-l)_+,\\
& \min_l W(i-1,j,l) - l,\ \min_l W(i+1,j,l) - l\big\}.
\end{aligned}$$

The intuitive idea is that for every feasible approximation B with $b_{ij} = k$, the maximal weight of a $q-(i,j)-$path in the DT-ICC-graph for B is at least $W(i,j,k)$. The numbers $W(i,j,k)$ can be computed efficiently (complexity $O(m^2 n \Delta^2)$, where Δ denotes any upper bound for $|I_{ij}|$) as described in Algorithm 1. Again, in order to avoid case distinctions at the boundaries, we add the values

$$W(0,j,0) = W(m+1,j,0) = a_{0j} = a_{m+1,j} = 0 \qquad (j \in [n]).$$

Definition 5. The *ICC-approximation complexity* of A (with respect to the given intervals I_{ij}) is defined by

$$\tilde{c}(A) = \max_i \min_k W(i,n,k).$$

Clearly, $\tilde{c}(A)$ is a lower bound for the ICC-complexity of a feasible approximation of A. We will show that this bound is sharp by an explicit construction of an approximation matrix B. For the last column we put

$$b_{in} = \begin{cases} a_{in} & \text{if } W(i,n,a_{in}) \leqslant \tilde{c}(A), \\ \max\{k \ : \ W(i,n,k) \leqslant \tilde{c}(A)\} & \text{otherwise.} \end{cases}$$

For $j < n$, we assume that the entries $b_{i,j+1}$ are already determined, and put

$$b_{ij} = \max\left\{k \ : \ W(i,j,k) + (b_{i,j+1} - k)_+ \leqslant W(i,j+1,b_{i,j+1})\right\}.$$

Example 2. We consider the following fluence matrix A with $c(A) = 8$.

$$A = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

We choose the upper and lower bound such that $|b_{ij} - a_{ij}| \leqslant 1$ for every (i,j). The intervals and an optimal approximation are

$$\begin{pmatrix} [3,5] & [0,1] & [0,1] \\ [0,1] & [0,1] & [3,5] \end{pmatrix}, \qquad B = \begin{pmatrix} 3 & 1 & 0 \\ 1 & 1 & 3 \end{pmatrix}$$

with $c(B) = 4$. Our algorithm obtains matrix B as follows. First we compute the numbers $W(i,j,k)$, and obtain, for each (i,j), a vector

$$\left(W_{i,j,\underline{a_{ij}}}, W_{i,j,\underline{a_{ij}}+1}, \dots, W_{i,j,\overline{a_{ij}}}\right).$$

These vectors are collected in the following array.

$$\begin{array}{rrr} (3,4,5) & (3,3) & (3,3) \\ (0,1) & (2,2) & (4,5,6). \end{array}$$

Thus the optimal DT is

$$\max\{\min\{3,3\}, \min\{4,5,6\}\} = 4.$$

For the third column we choose $b_{13} = 0$ and $b_{23} = 3$. For the entry $(1,2)$ we have

$$W(1,2,0) + w((1,2,0),(1,3,0)) = W(1,2,1) + w((1,2,1),(1,3,0)) = W(1,3,0).$$

We choose the maximal possible value $b_{12} = 1$. Observe that $b_{12} = 0$ is indeed not possible, since it leads to an increased DT. For entry $(2,2)$ we have

$$W(2,2,0) + w((2,2,0),(2,3,3)) = 2 + 3 > W(2,3,3),$$

so here $b_{22} = 1$ is the only possible choice. Similarly, we get $b_{11} = 3$ and $b_{21} = 1$. Clearly, the latter one can be replaced by 0.

In order to prove that our method is correct, we need some simple properties of the numbers $W(i,j,k)$.

Lemma 1. *For every $(i,j) \in [m] \times [n]$ and every k such that $(i,j,k),(i,j,k+1) \in V_{ij}$ we have*

$$W(i,j,k) \leqslant W(i,j,k+1) \leqslant W(i,j,k)+1. \tag{1}$$

Furthermore, $W(i,j,k+1) = W(i,j,k)+1$ iff

$$W(i,j,k) = W(i,j-1,l)+(k-l)_+$$

for some $l \in I_{i,j-1}$ with $l \leqslant k$.

Proof. Since

$$W(i,j-1,l)+(k-l)_+ \leqslant W(i,j-1,l)-(k+1-l)_+$$

and using the definition of the $W(i,j,k)$, we conclude $W(i,j,k) \leqslant W(i,j,k+1)$. On the other hand, we have

$$\begin{aligned} W(i,j,k) &= \max\big\{\min_l W(i,j-1,l)+(k-l)_+, \\ &\qquad \min_l W(i-1,j,l)-l,\ \min_l W(i+1,j,l)-l\big\} \\ &\geqslant \max\big\{\min_l W(i,j-1,l)+(k+1-l)_+, \\ &\qquad \min_l W(i-1,j,l)-l,\ \min_l W(i+1,j,l)-l\big\}-1 \\ &= W(i,j,k+1)-1, \end{aligned}$$

where equality occurs iff $W(i,j,k) = W(i,j-1,l)+(k-l)_+$ and $k \geqslant l$. □

The next lemma is the key step of our argument. It asserts that the chosen b_{ij} do not lead to conflicts inside the columns.

Lemma 2. *For all j and all $i \in [m-1]$, we have*

$$W(i,j,b_{ij})-b_{ij} \leqslant W(i+1,j,b_{i+1,j}),$$

and for all j and all $i \in [2,m]$, we have

$$W(i,j,b_{ij})-b_{ij} \leqslant W(i-1,j,b_{i-1,j}).$$

Proof. We only show the first statement, since the second one can be proved similarly. Suppose the statement is false, i.e.

$$W(i,j,b_{ij})-b_{ij} > W(i+1,j,b_{i+1,j}).$$

By construction, there is some $k \in I_{ij}$ such that $W(i,j,k)-k \leqslant W(i+1,j,b_{i+1,j})$.

Case 1. $k < b_{ij}$. Let $\delta = b_{ij}-k > 0$. By Lemma 1 we have

$$W(i,j,k) \geqslant W(i,j,b_{ij})-\delta.$$

But now we obtain

$$W(i,j,k)-k \geqslant (W(i,j,b_{ij})-\delta)-(b_{ij}-\delta) > W(i+1,j,b_{i+1,j}),$$

and this is the required contradiction.

Case 2. $k > b_{ij}$. Let $\delta = k - b_{ij} > 0$. By construction of the numbers b_{ij},

$$W(i,j,b_{ij}) + (b_{i,j+1} - b_{ij})_+ \leqslant W(i,j+1,b_{i,j+1}),$$
$$W(i,j,b_{ij}+1) + (b_{i,j+1} - (b_{ij}+1))_+ > W(i,j+1,b_{i,j+1}).$$

Using Lemma 1, this is possible only if

$$W(i,j,b_{ij}+1) = W(i,j,b_{ij}) + 1.$$

Using Lemma 1 repeatedly, we obtain

$$W(i,j,k) = W(i,j,b_{ij}) + \delta.$$

But together this implies $W(i,j,k) - k = W(i,j,b_{ij}) - b_{ij}$, which is a contradiction. □

Now let G be the DT-ICC-graph for B. Denote by $\alpha_1(i,j)$ the maximal weight of a $q-(i,j)-$path in G. Note that the numbers $\alpha_1(i,j)$ can be computed similarly to the numbers $W(i,j,k)$. Clearly, $\alpha_1(i,1) = b_{i1}$, and the procedure for column $j > 1$ is described in Algorithm 2.

Lemma 3. *For all (i,j) we have $\alpha_1(i,j) \leqslant W(i,j,b_{ij})$.*

Proof. We use induction on j. For $j = 1$ the claim is obvious:

$$\alpha_1(i,1) = W(i,1,b_{i1}) = b_{i1}.$$

Now let $j > 1$. After the initialization of the numbers $\alpha_1(i,j)$ in the first loop of Algorithm 2 we obtain for every i,

$$\alpha_1(i,j) = \alpha_1(i,j-1) + (b_{ij} - b_{i,j-1})_+ \\ \leqslant W(i,j-1,b_{i,j-1}) + (b_{ij} - b_{i,j-1})_+ \leqslant W(i,j,b_{ij}).$$

We just have to check that this inequalities remain valid in every updating step. Suppose the first violation occurs when we replace $\alpha_1(i,j)$ by $\alpha_1(i\pm 1,j) - b_{i\pm 1,j}$. In this case,

$$\alpha_1(i,j) = \alpha_1(i\pm 1,j) - b_{i\pm 1,j} \leqslant W(i\pm 1,j,b_{i\pm 1,j}) - b_{i\pm 1,j} \leqslant W(i,j,b_{ij}),$$

where the last inequality is Lemma 2. So the statement of the lemma remains valid. □

By Lemma 3 (and Theorem 1), matrix B allows a decomposition with $DT \leqslant \tilde{c}(A)$ and this implies the following theorem.

Theorem 2. *The minimal DT of a decomposition of a feasible approximation of A equals $\tilde{c}(A)$ and an approximation matrix B realizing this DT can be constructed as described above in time $O(m^2 n\Delta^2)$.*

Proof. The only thing that is left to prove is the complexity statement. For this it is sufficient to note that the computation of the numbers $W(i,j,k)$ dominates the computation time, since this has complexity $O(m^2 n\Delta^2)$ as can be seen immediately from Algorithm 1. But after the numbers $W(i,j,k)$ have been computed we look at every entry (i,j) only once and in order to fix b_{ij} we have to do at most $|I_{ij}|$ comparisons. So the matrix B is determined in time $O(mn\Delta)$ and this concludes the proof. □

Algorithm 1 Computation of the numbers $W(i,j,k)$

```
for $i \in [m]$ do $W(i,0,0) = 0$
for $j = 1$ to $n$ do
  for $i \in [m]$ do
    for all $k$ do
      $W(i,j,k) = \min_l W(i,j-1,l) + (k-l)_+$
  for $i = 2$ to $m$ do
    for all $k$ do
      $W(i,j,k) = \max\left\{W(i,j,k), \min_l W(i-1,j,l) - l\right\}$
    for $i' = i-1$ downto $1$ do
      for all $k$ do
        $W(i',j,k) = \max\left\{W(i',j,k), \min_l W(i'+1,j,l) - l\right\}$
```

Algorithm 2 Computation of the numbers $\alpha_1(i,j)$ for fixed j

```
for $i \in [m]$ do
  $\alpha_1(i,j) = \alpha_1(i,j-1) + (b_{ij} - b_{i,j-1})_+$
for $i = 2$ to $m$ do
  $\alpha_1(i,j) = \max\{\alpha_1(i,j), \alpha_1(i-1,j) - b_{i-1,j}\}$
for $i' = i-1$ downto $1$ do
  $\alpha_1(i',j) = \max\left\{\alpha_1(i',j), \alpha_1(i'+1,j) - b_{i'+1,j}\right\}$
```

5 Reducing the total change

The construction described in Section 4 leads to an approximation B with minimal delivery time, but a large total change $TC(B)$. The reason is, that we put

$$b_{ij} = \max\left\{k \ : \ W(i,j,k) + (b_{i,j+1} - k) \leqslant W(i,j+1,b_{i,j+1})\right\},$$

even if none of the vertices (i,j,k) is critical, i.e. part of a q-s-path of maximal weight in the DT-ICC-graph of a feasible approximation of A. Thus, the aim is to find an approximation with the same delivery time, but smaller total change. Clearly, we can replace b_{ij} by a value b'_{ij} with $b_{ij} < b'_{ij} \leqslant a_{ij}$ in the case $b_{ij} < a_{ij}$, respectively with $a_{ij} \geqslant b'_{ij} > b_{ij}$ in the case $a_{ij} > b_{ij}$, if this decision does not increase the maximal weight of a q-s-path in the DT-ICC-graph.

Let therefore G be the DT-ICC-graph of B and let $\alpha_1(i,j)$ denote the maximal weight of a q-(i,j)-path in G. Similarly, let $\alpha_2(i,j)$ denote the maximal weight of an (i,j)-s-path in G. The values $\alpha_2(i,j)$ can be computed similarly as the numbers $\alpha_1(i,j)$.

Definition 6. Let B be a feasible approximation of A. For $(i,j) \in [m] \times [n]$, an integer b is called $(i,j)-$*feasible* (with respect to B) if the following conditions are satisfied.

1. $b \in I_{ij}$.
2. $\alpha_1(i,j-1) + (b - b_{i,j-1})_+ + (b_{i,j+1} - b)_+ + \alpha_2(i,j+1) \leqslant \tilde{c}(A)$.
3. $i = 1$ or $\alpha_1(i,j-1) + (b - b_{i,j-1})_+ - b + \alpha_2(i-1,j) \leqslant \tilde{c}(A)$.
4. $i = m$ or $\alpha_1(i,j-1) + (b - b_{i,j-1})_+ - b + \alpha_2(i+1,j) \leqslant \tilde{c}(A)$.
5. $i = 1$ or $\alpha_1(i-1,j) - b_{i-1,j} + (b_{i,j+1} - b)_+ + \alpha_2(i,j+1) \leqslant \tilde{c}(A)$.
6. $i = m$ or $\alpha_1(i+1,j) - b_{i+1,j} + (b_{i,j+1} - b)_+ + \alpha_2(i,j+1) \leqslant \tilde{c}(A)$.
7. $i \in \{1,m\}$ or $\alpha_1(i-1,j) - b_{i-1,j} - b + \alpha_2(i+1,j) \leqslant \tilde{c}(A)$.
8. $i \in \{1,m\}$ or $\alpha_1(i+1,j) - b_{i+1,j} - b + \alpha_2(i-1,j) \leqslant \tilde{c}(A)$.

In other words, b is $(i,j)-$feasible iff we can replace b_{ij} by b without destroying the $DT-$optimality of B. Fig 4 illustrates the different possibilities for a path to pass through vertex (i,j). Each of these possibilities corresponds to one of the conditions 2 through 8 in Definition 6.

We propose a heuristic, formally described in Algorithm 3, to reduce the total change. Clearly, the application of this algorithm can be iterated until no more changes occur.

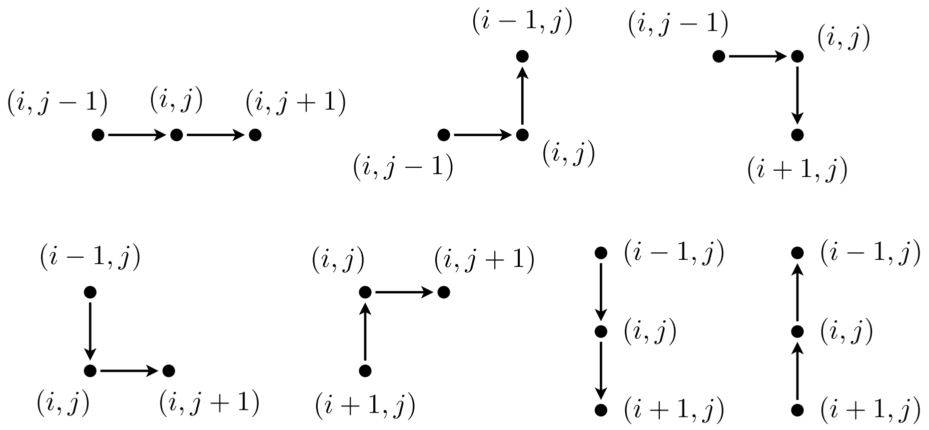

Figure 4: The seven different types of paths that are affected by the choice of b_{ij}.

Algorithm 3 Heuristic for total change minimization

```
for $j = 1$ to $n$ do
  for $i = 1$ to $m$ do
    if $b_{ij} < a_{ij}$ and $b_{ij} + 1$ is $(i,j)-$feasible then $b_{ij}++$
    if $b_{ij} > a_{ij}$ and $b_{ij} - 1$ is $(i,j)-$feasible then $b_{ij}--$
    Update the numbers $\alpha_1(k,l)$ and $\alpha_2(k,l)$
```

6 Test Results

In this section we demonstrate the DT-reduction obtained by the methods from Section 4 and the total change reduction using the heuristic approach from Section 5. We use matrices of size 15×15 and 30×30 with random entries $a_{ij} \in \{0,1,\dots,L\}$ for $L \in \{8,12,16\}$. In our tests we choose the upper and lower bounds for the entries such that each entry is changed by at most 2, i.e. we put

$$\underline{a_{ij}} = (a_{ij} - 2)_+, \qquad \overline{a_{ij}} = a_{ij} + 2.$$

For each L, we construct decompositions of 1000 matrices, and compute the average minimal delivery time $\tilde{c}(A)$ and the total change according to our algorithm from Section 4. Finally, we analyse the total change reduction, that can be achieved using Algorithm 3. The results are shown in Table 1 and 2. For comparison we include the minimal DT for exact decomposition with ICC [7]. Columns 'DT_1' and 'DT_2' contain the average delivery times for the exact and for the approximated decomposition, respectively. Columns 'TC_1' and 'TC_2' contain the total change values before and after the application of Algorithm 3. Our algorithms are completely practicable. On a 3GHz workstation, the computations for the last row, i.e. for the decomposition of 1000 matrices of size 15×15 with entries from $\{0,1,\dots,16\}$ took only 5 seconds for $m = n = 15$ and less than a minute for $m = n = 30$.

L	DT_1	DT_2	TC_1	TC_2
8	35.7	14.6	329.1	188.7
12	51.8	29.2	358.3	140.8
16	67.7	44.6	373.9	112.8

Table 1: Test results for $m = n = 15$.

L	DT_1	DT_2	TC_1	TC_2
8	67.7	24.5	1360.0	837.2
12	97.9	51.4	1484.3	651.3
16	127.7	79.9	1546.2	505.4

Table 2: Test results for $m = n = 30$.

Basically, we can draw two conclusions from our results.

1. The approximation approach leads to an significant DT-reduction: for $L = 16$, allowing a change of at most 2 for each entry reduces the DT by more than 30%.
2. Our heuristic leads to a large total change reduction: for $L = 16$ the total change can be reduced by almost 60%.

7 Summary and discussion

We presented an efficient method to minimize exactly the decomposition time in approximated MLC shape matrix decomposition with interleaf collision constraint. We also described a heuristic for reducing the total approximation error, and demonstrated the proposed algorithms on randomly generated matrices. The obvious next problem, which is the subject of ongoing research, is to find an exact algorithm for the minimization of the total change.

References

[1] D. Baatar, H.W. Hamacher, M. Ehrgott and G.J. Woeginger. "Decomposition of integer matrices and multileaf collimator sequencing". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 6–34. DOI: 10.1016/j.dam.2005.04.008.

[2] T.R. Bortfeld, D.L. Kahler, T.J. Waldron and A.L. Boyer. "X–ray field compensation with multileaf collimators". In: *Int. J. Radiat. Oncol. Biol. Phys.* 28 (1994), pp. 723–730. DOI: 10.1016/0360-3016(94)90200-3.

[3] K. Engel. "A new algorithm for optimal multileaf collimator field segmentation". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 35–51. DOI: 10.1016/j.dam.2004.10.007.

[4] K. Engel and A. Kiesel. "Approximated matrix decomposition for IMRT planning with multileaf collimators". In: *OR Spectrum* 33.1 (2011), pp. 149–172. DOI: 10.1007/s00291-009-0168-5.

[5] K. Engel and E. Tabbert. "Fast Simultaneous Angle, Wedge, and Beam Intensity Optimization in Inverse Radiotherapy Planning". In: *Optimization and Engineering* 6.4 (2005), pp. 393–419. DOI: 10.1007/s11081-005-2065-3.

[6] J.M. Galvin, X.G. Chen and R.M. Smith. "Combining multileaf fields to modulate fluence distributions". In: *Int. J. Radiat. Oncol. Biol. Phys.* 27 (1993), pp. 697–705. DOI: 10.1016/0360-3016(93)90399-G.

[7] T. Kalinowski. "A duality based algorithm for multileaf collimator field segmentation with interleaf collision constraint". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 52–88. DOI: 10.1016/j.dam.2004.10.008.

[8] T. Kalinowski. "Multileaf collimator shape matrix decomposition". In: *Optimization in Medicine and Biology.* Ed. by G.J. Lim and E.K. Lee. Auerbach Publishers Inc., 2008, pp. 249–282.

[9] T. Kalinowski. "Realization of intensity modulated radiation fields using multileaf collimators". In: *Information Transfer and Combinatorics.* Ed. by R. Ahlswede *et al.* Vol. 4123. LNCS. Springer-Verlag, 2006, pp. 1010–1055. DOI: 10.1007/11889342_65.

[10] S. Kamath, S. Sahni, J. Li, J. Palta and S. Ranka. "Leaf sequencing algorithms for segmented multileaf collimation". In: *Phys. Med. Biol.* 48.3 (2003), pp. 307–324. DOI: 10.1088/0031-9155/48/3/303.

[11] S. Kamath, S. Sartaj, J. Palta, S. Ranka and J. Li. "Optimal leaf sequencing with elimination of tongue–and–groove underdosage". In: *Phys. Med. Biol.* 49 (2004), N7–N19. DOI: 10.1088/0031-9155/49/3/N01.

[12] G.J. Lim, M.C. Ferris, S.J. Wright, D.M. Shepard and M.A. Earl. "An optimization framework for conformal radiation treatment planning". In: *INFORMS Journal on Computing* 19.3 (2007), pp. 366–380. DOI: 10.1287/ijoc.1060.0179.

[13] H.E. Romeijn, R.K. Ahuja, J.F. Dempsey and A. Kumar. "A Column Generation Approach to Radiation Therapy Treatment Planning Using Aperture Modulation". In: *SIAM J. on Optimization* 15.3 (2005), pp. 838–862. ISSN: 1052-6234. DOI: 10.1137/040606612.

[14] R.A.C. Siochi. "Minimizing static intensity modulation delivery time using an intensity solid paradigm". In: *Int. J. Radiat. Oncol. Biol. Phys.* 43 (1999), pp. 671–680. DOI: 10.1016/S0360-3016(98)00430-1.

[15] P. Xia and L. Verhey. "Multileaf collimator leaf–sequencing algorithm for intensity modulated beams with multiple static segments". In: *Med. Phys.* 25 (1998), pp. 1424–1434. DOI: 10.1118/1.598315.

A Minimum Cost Flow Formulation for Approximated MLC Segmentation[*]

Thomas Kalinowski

Abstract

Shape matrix decomposition is a subproblem in radiation therapy planning. A given fluence matrix A has to be written as a sum of shape matrices corresponding to homogeneous fields that can be shaped by a multileaf collimator (MLC). We solve the problem of finding an approximation B of A satisfying prescribed upper and lower bounds for each entry. The approximation B is determined such that the corresponding fluence can be realized with a prescribed delivery time using a multileaf collimator with an interleaf collision constraint, and under this condition the distance between A and B is minimized.

Keywords: intensity modulated radiation therapy, multileaf collimator, minimum cost flows

1 Introduction

Radiation therapy is an important method in cancer treatment. Basically, the aim is to destroy the tumor while minimizing the damage to the healthy tissue, in particular to sensitive structures or organs at risk. In clinical practice it is common to use a linear accelerator which can release radiation from different directions. In addition, a multileaf collimator (MLC) can be used to cover certain parts of the irradiated area. An MLC consists of two banks of metal leaves that are arranged pairwise such that each pair consists of a left leaf and a right leaf which can be moved into the radiation beam from their respective sides. Figure 1 illustrates how an MLC can be used to modulate the intensity. The subject of the present paper is one step of the treatment planning for an MLC in the *step-and-shoot mode*. That term means the radiation is switched off while the leaves are moving, so the task is to determine finitely many fields such that their superposition yields the required fluence. The two main optimization goals considered in the literature are the minimization of the delivery time (DT) and minimization of the number of used shapes. This problem has been considered by many authors (see [8] and the references therein). There are many algorithms for this task, using different reformulations of the problem and including different technological constraints, such as the interleaf collision constraint (ICC) and the tongue-and-groove constraint. In [5] it was suggested to decompose an approximation of A. This might be necessary if the delivery time for an exact decomposition of A is prohibitively large. It is

[*] *Networks* **57**, 135–140, 2011, `doi:10.1002/net.20394`

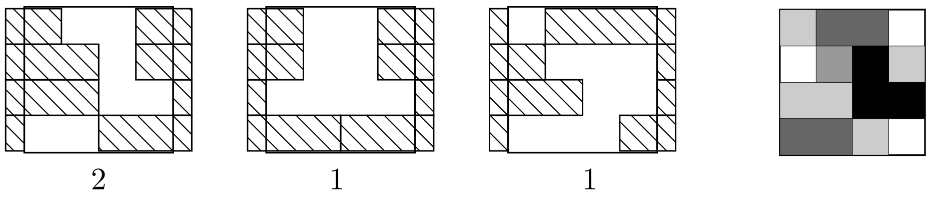

Figure 1: Generating an intensity modulated radiation field by superimposing three homogeneous fields shaped by an MLC. The shaded areas are the regions covered by MLC leaves, the numbers indicate for how long the corresponding field is irradiated, and the greyscales in the rightmost square show the total fluence distribution.

further justified by the fact that A is a result of numerical computations based on simplified models, so there should be an error interval attached to each entry. There are two natural objectives for this approximation problem. First, the delivery time for the approximation matrix should be small and second, a given delivery time should be realized by changing A as little as possible. Both of these problems were solved for single row matrices (and consequently in general for MLCs with independent rows) in [5]. In this paper we consider MLCs that have an interleaf collision constraint, which means that an overlap between opposite leaves in consecutive rows is not allowed. Given a fluence matrix A and upper and lower bounds for the entries of the approximation matrix, the minimal possible delivery time of an approximation was determined in [9], where the authors also described a heuristic method for the reduction of the distance between A and the approximation matrix B (to be defined later). The main result of the present paper is a minimum cost flow formulation of the exact minimization of this distance. In Section 2 we give a precise formulation of the approximation problem and we introduce some notation. Section 3 contains our main result: the approximation problem is dual to a minimum cost flow problem. Finally, Section 4 contains some computational results.

2 Problem formulation

Throughout the paper we use the standard notation

$$[k] = \{1, 2, \dots, k\}, \qquad [k, l] = \{k, k+1, \dots, l\}$$

for integers k and l with $k \leqslant l$. As in [9], we start with a fluence matrix A of size $m \times n$, and two matrices $\underline{A}$ and $\overline{A}$ containing the lower and upper bounds for the entries:

$$0 \leqslant \underline{a}_{ij} \leqslant a_{ij} \leqslant \overline{a}_{ij} \qquad (i, j) \in [m] \times [n].$$

Definition 1 (Feasible Approximation)**.** Any integer matrix B with

$$\underline{a}_{ij} \leqslant b_{ij} \leqslant \overline{a}_{ij} \qquad ((i, j) \in [m] \times [n])$$

is called *feasible approximation* of A. The *total change* $TC(B)$ of a feasible approximation B is defined by

$$TC(B) = \sum_{i=1}^{m} \sum_{j=1}^{n} |b_{ij} - a_{ij}|.$$

The homogeneous fields that can be shaped by the MLC are described by binary matrices of size $m \times n$ which we call *shape matrices*.

Definition 2 (Shape matrix)**.** An $m \times n$ matrix S is a *shape matrix* if there are pairs of integers (l_i, r_i) $(i \in [m])$, such that the following conditions are satisfied:

1. $s_{ij} = \begin{cases} 1 & \text{if } l_i < j < r_i, \\ 0 & \text{otherwise.} \end{cases}$
2. $l_i < r_{i+1}$ and $r_i > l_{i+1}$ for all $i \in [m-1]$.

The first condition is just stating that for each row i, there are a left leaf covering bixels $(i,1),\dots,(i,l_i)$ and a right leaf covering bixels $(i,r_i),\dots,(i,n)$, while the bixels $(i,l_i+1),\dots,(i,r_i-1)$ are exposed to radiation. The second condition is called the interleaf collision constraint (ICC). It ensures the left leaf of row i and the right leaf of row $i \pm 1$ do not overlap, which is required by some widely used MLCs. An MLC leaf sequence for A corresponds to a representation of A as a weighted sum of shape matrices.

Definition 3 (Shape matrix decomposition)**.** A *shape matrix decomposition* of A is a representation of A as a positive integer linear combination of shape matrices

$$A = \sum_{t=1}^{k} u_t S^{(t)}.$$

The *delivery time* (DT) of this decomposition is just the sum of the coefficients,

$$DT = \sum_{t=1}^{k} u_t.$$

Example 1. For the shape matrix decomposition

$$\begin{pmatrix} 1&3&3&0 \\ 0&2&4&1 \\ 1&1&4&4 \\ 3&3&1&0 \end{pmatrix} = 2 \cdot \begin{pmatrix} 0&1&1&0 \\ 0&0&1&0 \\ 0&0&1&1 \\ 1&1&0&0 \end{pmatrix} + \begin{pmatrix} 0&1&1&0 \\ 0&1&1&0 \\ 1&1&1&1 \\ 0&0&0&0 \end{pmatrix} + \begin{pmatrix} 1&0&0&0 \\ 0&1&1&1 \\ 0&0&1&1 \\ 1&1&1&0 \end{pmatrix},$$

corresponding to Figure 1, we have $DT = 4$.

There are 3 natural optimization problems [9].

MinDT. Find a shape matrix decomposition $A = \sum_{t=1}^{k} u_t S^{(t)}$ such that $DT = \sum_{t=1}^{k} u_t$ is minimal.

Approx-MinDT. Find a feasible approximation B and a shape matrix decomposition $B = \sum_{t=1}^{k} u_t S^{(t)}$ such that $DT = \sum_{t=1}^{k} u_t$ is minimal.

Approx-MinTC. For a given delivery time $\tilde{c}$, find a feasible approximation B and a shape matrix decomposition

$$B = \sum_{t=1}^{k} u_t S^{(t)} \tag{1}$$

such that $\sum_{t=1}^{k} u_t \leqslant \tilde{c}$, and under this condition $TC(B)$ is minimal.

The first problem **MinDT** is the exact decomposition problem which can be solved by several efficient algorithms [3, 7, 10]. The second problem **Approx-MinDT** was solved in [9]. In the present paper we consider the third problem **Approx-MinTC**. Throughout the paper, we will always assume that the problem is feasible. In practice that can be realized by solving **Approx-MinDT** first. This yields the minimal possible value for $\tilde{c}$, and for each value as least as large **Approx-MinTC** is feasible.

3 A solution of the problem Approx-MinTC

We start by formulating an LP model for **Approx-MinTC**. Since we are only interested in the sum of the coefficients, we may assume that all the coefficients u_t in (1) are equal to 1 (allowing the same shape matrix $S^{(t)}$ for different values of t). We introduce variables L_{ij} and R_{ij} for $(i,j) \in [m] \times [n]$. Formally, if the shape matrix $S^{(t)}$ in the decomposition $B = \sum_{t=1}^{k} S^{(t)}$ is determined by the parameters $(l_i^{(t)}, r_i^{(t)})$ for $i \in [m]$, our variables are

$$L_{ij} = \left|\left\{t \ : \ l_i^{(t)} < j\right\}\right|, \quad \text{and} \quad R_{ij} = \left|\left\{t \ : \ r_i^{(t)} \leqslant j\right\}\right|.$$

In other words, the variable L_{ij} is the number of shapes where bixel (i,j) is not covered by the left leaf, while R_{ij} counts the shapes where bixel (i,j) is covered by the right leaf. Obviously, this gives $b_{ij} = L_{ij} - R_{ij}$. In addition, we introduce the variables $x_{ij} = |a_{ij} - b_{ij}|$ for $(i,j) \in [m] \times [n]$. Now we can formulate **Approx-MinTC** for a given delivery time $\tilde{c}$ as an LP. To clarify our notation we also write down the dual variables for the constraints.

$$\min \quad \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} \tag{2}$$

$$\begin{array}{rlll}
L_{i,j+1} - L_{ij} \geqslant 0 & \alpha_{ij} \geqslant 0 & ((i,j) \in [m] \times [n-1]), & (3)\\
-L_{in} \geqslant -\tilde{c} & \alpha_{in} \geqslant 0 & (i \in [m]), & (4)\\
R_{i,j+1} - R_{ij} \geqslant 0 & \beta_{ij} \geqslant 0 & ((i,j) \in [m] \times [n-1]), & (5)\\
R_{i1} \geqslant 0 & \beta_{i0} \geqslant 0 & (i \in [m]) & (6)\\
R_{ij} - L_{ij} \geqslant -\overline{a}_{ij} & y_{ij} \geqslant 0 & ((i,j) \in [m] \times [n]), & (7)\\
L_{ij} - R_{ij} \geqslant \underline{a}_{ij} & z_{ij} \geqslant 0 & ((i,j) \in [m] \times [n]), & (8)\\
L_{ij} - R_{i+1,j} \geqslant 0 & u_{ij} \geqslant 0 & ((i,j) \in [m-1] \times [n]), & (9)\\
L_{ij} - R_{i-1,j} \geqslant 0 & v_{ij} \geqslant 0 & ((i,j) \in [2,m] \times [n]), & (10)\\
R_{ij} - L_{ij} + x_{ij} \geqslant -a_{ij} & p_{ij} \geqslant 0 & ((i,j) \in [m] \times [n]), & (11)\\
L_{ij} - R_{ij} + x_{ij} \geqslant a_{ij} & q_{ij} \geqslant 0 & ((i,j) \in [m] \times [n]). & (12)
\end{array}$$

By definition, the variables L_{ij} and R_{ij} should be nonnegative. We do not want to require this explicitly in the LP since we want to have equality constraints in the dual, but note that nonnegativity is implied: constraints (6) together with (5) force all the variables R_{ij} to be nonnegative, and from (8) it follows that also $L_{ij} \geqslant 0$ for all $(i,j) \in [m] \times [n]$. Constraints (3), (5) and (9), (10) are consequences of the inclusions

$$\begin{aligned} &\{t \,:\, l_i^{(t)} < j\} \subseteq \{t \,:\, l_i^{(t)} < j+1\}, && \{t \,:\, r_i^{(t)} \leqslant j\} \subseteq \{t \,:\, r_i^{(t)} \leqslant j+1\} \\ &\{t \,:\, r_{i+1}^{(t)} \leqslant j\} \subseteq \{t \,:\, l_i^{(t)} \leqslant j\}, && \{t \,:\, r_{i-1}^{(t)} \leqslant j\} \subseteq \{t \,:\, l_i^{(t)} \leqslant j\}, \end{aligned}$$

where the inclusions in the second row follow from the interleaf collision constraint. The constraints (7) and (8) ensure that $\underline{a}_{ij} \leqslant b_{ij} \leqslant \overline{a}_{ij}$, while (4) is the constraint that the total number of shapes is at most $\tilde{c}$. Finally, constraints (11) and (12) are equivalent to $x_{ij} \geqslant |a_{ij} - b_{ij}|$, and the objective is to minimize the sum of all the deviations $|a_{ij} - b_{ij}|$.

We remark that the values L_{ij} and R_{ij} do not uniquely determine the shape matrix decomposition, because in the transformation from the shape matrices $S^{(t)}$ to the cardinalities L_{ij} and R_{ij} we lost some information. It is even not completely obvious that a solution of the LP always yields a feasible decomposition. But fortunately, a natural approach to construct appropriate shape matrices works: we define shape matrices $S^{(t)}$ such that the leaves move only from left to right as t increases. More precisely, for a given solution of the problem (2)–12) we consider the sets $\mathcal{L}_{ij} = [L_{ij}]$ and $\mathcal{R}_{ij} = [R_{ij}]$ for $(i,j) \in [m] \times [n]$ and put

$$s_{ij}^{(t)} = \begin{cases} 1 & \text{if } t \in \mathcal{L}_{ij} \setminus \mathcal{R}_{ij} \\ 0 & \text{otherwise} \end{cases} \qquad ((i,j) \in [m] \times [n], t \in [L])\,,$$

where $L = \max\{L_{in} \;:\; i \in [m]\}$. These matrices have the first property required in Definition 2 with parameters

$$\begin{aligned} l_i^{(t)} &= 0 \text{ for } t \leqslant L_{i1} && (i \in [m]), && (13)\\ r_i^{(t)} &= 1 \text{ for } t \leqslant R_{i1} && (i \in [m]), && (14)\\ l_i^{(t)} &= j-1 \text{ for } L_{i,j-1} < t \leqslant L_{ij} && ((i,j) \in [m] \times [2,n]), && (15)\\ r_i^{(t)} &= j \text{ for } R_{i,j-1} < t \leqslant R_{ij} && ((i,j) \in [m] \times [2,n]). && (16) \end{aligned}$$

For $t > L_{in}$, there is a zero row in the i-th row of $S^{(t)}$, which can be realized by parameters $l_i^{(t)} = n$ and $r_i^{(t)} = n+1$. The interleaf collision constraint $l_i^{(t)} < r_{i+1}^{(t)}$ is satisfied for every $t \in [L]$ and every $i \in [m-1]$. If $l_i^{(t)} = 0$ this is obvious. Otherwise we have $l_i^{(t)} = j-1$ for some $j \in [2, n+1]$. Then $t > L_{i,j-1} \geqslant R_{i+1,j-1}$, and consequently $r_{i+1}^{(t)} \geqslant j$. The interleaf collision constraint $l_i^{(t)} < r_{i-1}^{(t)}$ is proved similarly. Finally, using (7) and (8) we have

$$b_{ij} = \sum_{t=1}^{L} s_{ij}^{(t)} = |\mathcal{L}_{ij} \setminus \mathcal{R}_{ij}| = L_{ij} - R_{ij} \in \left[\underline{a}_{ij}, \overline{a}_{ij}\right].$$

Now we dualize the LP (2)– (12) to obtain the problem **TC-Dual**:

$$\begin{aligned}
&\max && \sum_{i=1}^{m}\sum_{j=1}^{n}\big(- a_{ij}p_{ij} + a_{ij}q_{ij} + \underline{a}_{ij}z_{ij} - \overline{a}_{ij}y_{ij}\big) - \tilde{c}\sum_{i=1}^{m}\alpha_{in} && (17)\\
&L_{ij}: && -\alpha_{ij}+\alpha_{i,j-1}+q_{ij}-p_{ij}+z_{ij} && \\
& && \qquad -y_{ij}+u_{ij}+v_{ij}=0 \qquad ((i,j)\in[m]\times[2,n]), && (18)\\
&L_{i1}: && -\alpha_{i1}+q_{i1}-p_{i1}+z_{i1}-y_{i1} && \\
& && \qquad +u_{i1}+v_{i1}=0 \qquad (i\in[m]), && (19)\\
&R_{ij}: && -\beta_{ij}+\beta_{i,j-1}-q_{ij}+p_{ij}-z_{ij} && \\
& && \qquad +y_{ij}-u_{i-1,j}-v_{i+1,j}=0 \qquad ((i,j)\in[m]\times[n-1]), && (20)\\
&R_{in}: && \beta_{i,n-1}-q_{in}+p_{in}-z_{in}+y_{in} && \\
& && \qquad -u_{i-1,n}-v_{i+1,n}=0 \qquad ((i,j)\in[m]\times[n-1]), && (21)\\
&x_{ij}: && p_{ij}+q_{ij}\leqslant 1 \qquad ((i,j)\in[m]\times[n]). && (22)
\end{aligned}$$

Formally, we would have to write down several of the constraints for $i=1$ and $i=m$ separately, since in these cases the variables $u_{i-1,j}$ and v_{ij} (resp. u_{ij} and $v_{i+1,j}$) are missing. In order avoid an unnecessary blowup of the formalism, we use the convention that

$$u_{0j}=u_{mj}=v_{1,j}=v_{m+1,j}=0 \qquad (j\in[n]).$$

Clearly, the objective (17) is equivalent to minimizing

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\big(a_{ij}p_{ij} - a_{ij}q_{ij} - \underline{a}_{ij}z_{ij} + \overline{a}_{ij}y_{ij}\big) + \tilde{c}\sum_{i=1}^{m}\alpha_{in}, \qquad (23)$$

and we will see that **TC-Dual** is equivalent to the problem of finding a $Q-S-$flow of minimum cost in the following network N. The node set V of the underlying digraph consists of two nodes $(i,j,0)$ and $(i,j,1)$ for each bixel $(i,j)\in[m]\times[n]$ and two additional nodes Q and S:

$$V=\{Q,S\}\cup\{(i,j,k)\ :\ (i,j)\in[m]\times[n],\ k\in\{0,1\}\}.$$

The arc set E is constructed corresponding to the variables in **TC-Dual**. From node Q, there is an outgoing arc to every node $(i,1,0)$ with corresponding flow variable β_{i0}. For the nodes $(i,j,1)$ with $(i,j)\in[m]\times[n-1]$, we have an outgoing arc to $(i,j+1,1)$ with corresponding flow variable α_{ij} and two outgoing arcs to $(i,j,0)$ corresponding to flow variables p_{ij} and y_{ij}. Similarly, the nodes $(i,n,1)$ have two outgoing arcs to $(i,n,0)$ with flow variables p_{in} and y_{in}, but their outgoing arc with flow variable α_{in} goes to S. From a node $(i,j,0)$ with $(i,j)\in[m]\times[n-1]$, we have an arc to $(i,j+1,0)$ with flow variable β_{ij} and two arcs to $(i,j,1)$ with flow variables q_{ij} and z_{ij}. For nodes $(i,n,0)$ with $i\in[m]$ the arc to the sink which would correspond to β_{in} is missing, but we still have the two arcs to $(i,n,1)$ with flow variables q_{in} and z_{in}. In addition, for $(i,j,0)$, if $i<m$, there is an arc to

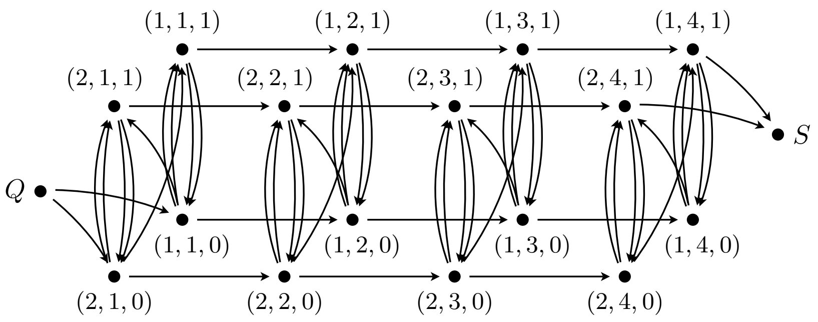

Figure 2: The digraph for the network model of the problem **TC-Dual** for a $2 \times 4-$matrix.

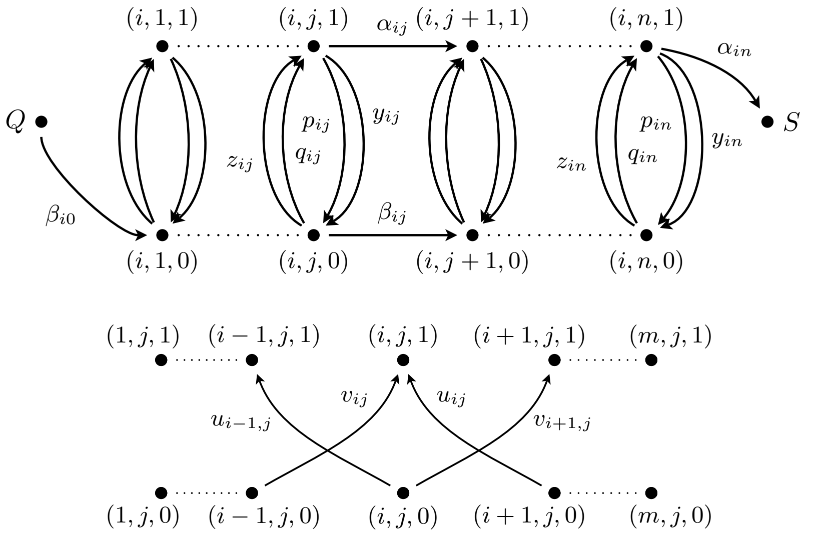

Figure 3: The structure of the digraph for the network model of the problem **TC-Dual**.

$(i+1, j, 1)$ with flow variable $v_{i+1,j}$, and for $i > 1$ there is an arc to $(i-1, j, 1)$ with flow variable $u_{i-1,j}$.

This digraph is illustrated in Figures 2 and 3. The capacity function is very simple: The arcs with flow variables p_{ij} and q_{ij} have capacity 1, and all the other arcs have infinite capacities. The cost function is defined such that the cost of a flow is precisely the objective function (23). Identifying the arcs with their corresponding flow variables, this can be described as follows. For $(i,j) \in [m] \times [n]$, the costs of the arcs p_{ij}, q_{ij}, z_{ij} and y_{ij} are a_{ij}, $-a_{ij}$, $-\underline{a}_{ij}$ and $\overline{a}_{ij}$, respectively. For $i \in [m]$, the cost of arc α_{in} is $\tilde{c}$. All the other arcs have zero cost.

Since the arcs p_{ij} and q_{ij} form a cycle of zero cost, we may assume that $p_{ij}q_{ij} = 0$ for every $(i,j) \in [m] \times [n]$. Under this assumption constraints (22) correspond to the capacity constraints for the arcs p_{ij} and q_{ij}, while the constraints (18)–(21) are the flow conservation constraints at the nodes $x \in V \setminus \{Q, S\}$. So we have proved the following theorem.

Theorem 1. *The problem* ***TC-Dual*** *is equivalent to the minimum cost $Q-S-$flow problem* *in the network N.*

By standard results from network flow theory [1], we obtain a solution of **Approx-MinTC** from a flow $\phi : E \to \mathbb{N}$ of minimum cost as follows. The residual network on the node set V with arc set E' and cost function $\text{cost}' : E' \to \mathbb{Z}$ is defined by

$$\begin{aligned} \phi(xy) < \text{capacity}(xy) \qquad &\Longrightarrow \qquad xy \in E',\ \text{cost}'(xy) = \text{cost}(xy),\\ \phi(xy) > 0 \qquad &\Longrightarrow \qquad yx \in E',\ \text{cost}'(yx) = -\text{cost}(xy). \end{aligned}$$

Recall that L_{ij} and R_{ij} are the dual variables of the flow conservation constraints in $(i,j,1)$ and $(i,j,0)$, respectively, so we can determine them as the negative distances (with respect to cost′) from Q to $(i,j,1)$ and $(i,j,0)$, respectively. We obtain the approximation matrix B by $b_{ij} = L_{ij} - R_{ij}$, and a shape matrix decomposition $B = \sum_{t=1}^{k} S^{(t)}$ with

$$s_{ij}^{(t)} = \begin{cases} 1 & \text{if } l_i^{(t)} < j < r_i^{(t)}, \\ 0 & \text{otherwise,} \end{cases} \qquad ((i,j) \in [m] \times [n])$$

where the parameters $l_i^{(t)}$ and $r_i^{(t)}$ are determined according to (13)–(16).

Example 2. We illustrate the method for $m = 1$, $n = 6$. Suppose we are given the following matrices A, $\underline{A}$ and $\overline{A}$:

$$\begin{pmatrix} 5 & 3 & 3 & 1 & 5 & 5 \end{pmatrix}, \begin{pmatrix} 4 & 2 & 2 & 0 & 4 & 4 \end{pmatrix}, \begin{pmatrix} 6 & 4 & 4 & 2 & 6 & 6 \end{pmatrix}.$$

For matrix A the minimal delivery time is 9, and we want to have an approximation matrix B with a delivery time of 6. The network is shown in Figure 4. The labels on the arcs are the nonzero costs, so a unit flow along the dashed path has cost -3, while a unit flow along the dotted path costs -1. The sum of these two unit flows has a cost of -4 and is optimal.

A shortest path tree in the residual network is shown in Figure 5, and the corresponding

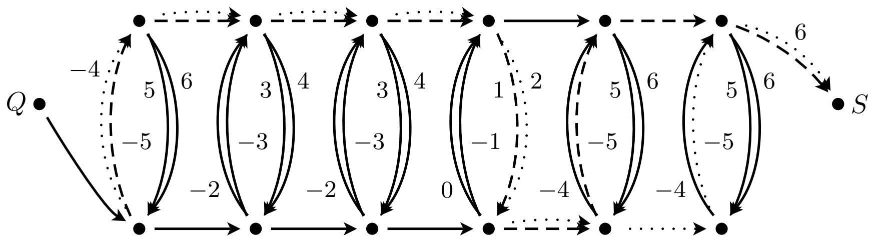

Figure 4: The network for the example.

approximation is

$$\begin{aligned} B = \begin{pmatrix} 4 & 3 & 3 & 2 & 4 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \\ + 2\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} + 2\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{aligned}$$

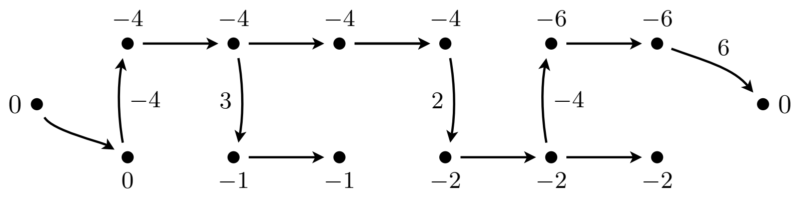

Figure 5: A shortest path tree in the residual network.

We conclude this section with a quick complexity analysis. We have reduced the total change optimal approximation of a matrix of size $m \times n$ to a minimum cost flow problem in a network with $2mn + 2$ nodes and $8mn - 2n$ arcs. Thus, according to [12, 13] the running time of the resulting algorithm is bounded by $O\left((mn)^2 \log^2(mn)\right)$. For comparison, without the interleaf collision constraint the approximation problem can be solved in time $O(mn^2)$ if the differences $\overline{a}_{ij} - \underline{a}_{ij}$ are bounded [5].

4 Test results

We did some computational experiments with a C++-implementation of our algorithm. We did not implement the minimum cost flow algorithm with the theoretically optimal complexity bound. Instead we used the implementation of a primal network simplex method from [11].

We generated matrices of sizes 15×15 and 30×30 with random entries from $\{0, 1, \dots, L\}$ (independent, uniformly distributed) for $L \in \{8, 12, 16\}$. The lower and upper bounds were chosen such that a maximal change of ± 2 is possible for each entry, in other words, we put

$$\underline{a}_{ij} = \max\{0, a_{ij} - 2\}, \qquad \overline{a}_{ij} = a_{ij} + 2$$

for all $(i, j) \in [m] \times [n]$. The results are shown in Table 1, where we averaged over 1000

	$m = n = 15$					$m = n = 30$				
L	DT_1	DT_2	TC_1	TC_2	time	DT_1	DT_2	TC_1	TC_2	time
8	35.7	14.5	188.7	165.3	2	67.7	24.5	837.2	713.9	11
12	51.9	29.3	140.8	125.8	2	97.9	51.4	651.3	559.5	15
16	67.6	44.3	112.8	102.0	3	127.8	79.9	505.4	430.7	17

Table 1: Test results for random matrices.

matrices for each triple (m, n, L). Columns 'DT_1' and 'DT_2' contain the average decomposition times for the exact and the approximated decomposition, respectively. For the minimal possible delivery time $\tilde{c}$, we computed the total change using the heuristic approach from [9] (column '$TC1$') and with our exact method (column '$TC2$'). The final column 'time' contains the computation time (in seconds) for the approximated decomposition of 1000 matrices on a 3GHz workstation with 16GB RAM. We also tested our algorithm for two sets of practical matrices that were used in [2] and [4] and can be found online [6]. The first set contained 20 matrices of size 20×20 with entries from $\{0, 1, \ldots, 15\}$, and the second one consisted of 20 matrices of size 40×40 with entries from $\{0, 1, \ldots, 10\}$. The averaged results are shown in Table 2, where the computation time for the whole table

set	DT_1	DT_2	TC_1	TC_2
1	83.6	51.0	227.0	204.4
2	108.9	47.9	1387.4	1180.7

Table 2: Test results for real-world matrices.

was less than a second. Figure 6 illustrates the tradeoff between delivery time and total

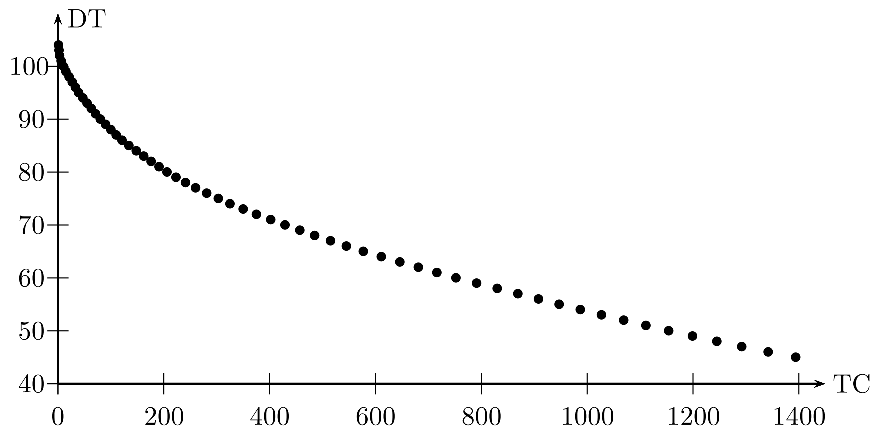

Figure 6: The tradeoff between delivery time and total change.

change for one of the $40 \times 40-$matrices from [6].

5 Summary and discussion

We formulated the approximated MLC shape matrix decomposition with minimal total change as a minimum cost flow problem. This formulation allows us to include the interleaf collision constraint into the model. We demonstrated that this problem can be solved very efficiently using a standard implementation of the network simplex algorithm.

We want to conclude the paper with a short discussion of the relevance of our approximation approach. In some sense, the shape matrix decomposition problem could be considered as solved, since there are many efficient algorithms, even including additional technological constraints. But of course, there is room for improvement. We suggest the following two problems that might arise from a practical point of view.

1. What happens if the delivery time for a leaf sequence obtained by some exact algorithm is considered to be too large for clinical practice?
2. There are certain dosimetric effects of small or narrow fields which are not captured in the mathematical model underlying the standard algorithms. This could lead to significant differences between the planned and the delivered fluence.

We think that the "smoothing" of the fluence that is achieved by our approximation has a positive effect in both of these contexts. It has already been shown that the delivery time can be reduced considerably. Intuitively, the approximation also reduces the number of necessary small shapes with bad dosimetric properties. This deserves further investigations.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin. *Network flows.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[2] D. Baatar, N. Boland, S. Brand and P. Stuckey. "Minimum cardinality matrix decomposition into consecutive-ones matrices: CP and IP approaches". In: *Proc. 4th CPAIOR 2007.* Ed. by P. Van Hentenryck and L. Wolsey. Vol. 4510. LNCS. Springer, 2007, pp. 1–15. DOI: `10.1007/978-3-540-72397-4`.

[3] D. Baatar, H.W. Hamacher, M. Ehrgott and G.J. Woeginger. "Decomposition of integer matrices and multileaf collimator sequencing". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 6–34. DOI: `10.1016/j.dam.2005.04.008`.

[4] H. Cambazard, E. O'Mahony and B. O'Sullivan. "A shortest path-based approach to the multileaf collimator sequencing problem". In: *Proc. 6th CPAIOR 2009.* Ed. by W.-J. van Hoeve and J.N. Hooker. Vol. 5574. LNCS. Springer, 2009, pp. 41–55. DOI: `10.1007/978-3-642-01929-6_5`.

[5] K. Engel and A. Kiesel. "Approximated matrix decomposition for IMRT planning with multileaf collimators". In: *OR Spectrum* 33.1 (2011), pp. 149–172. DOI: `10.1007/s00291-009-0168-5`.

[6] A. Holder. *Intensity map repository.* online repository. `http://holderfamily.dot5hosting.com/aholder/oncology/software/IntensityMaps` (12 March 2013). 2009.

[7] T. Kalinowski. "A duality based algorithm for multileaf collimator field segmentation with interleaf collision constraint". In: *Discr. Appl. Math.* 152.1-3 (2005), pp. 52–88. DOI: `10.1016/j.dam.2004.10.008`.

[8] T. Kalinowski. "Realization of intensity modulated radiation fields using multileaf collimators". In: *Information Transfer and Combinatorics.* Ed. by R. Ahlswede *et al.* Vol. 4123. LNCS. Springer-Verlag, 2006, pp. 1010–1055. DOI: `10.1007/11889342_65`.

[9] T. Kalinowski and A. Kiesel. "Approximated MLC shape matrix decomposition with interleaf collision constraint". In: *Algorithmic Operations Research* 4.1 (2009), pp. 49–57.

[10] S. Kamath, S. Sahni, J. Li, J. Palta and S. Ranka. “Leaf sequencing algorithms for segmented multileaf collimation”. In: *Phys. Med. Biol.* 48.3 (2003), pp. 307–324. DOI: 10.1088/0031-9155/48/3/303.

[11] A. Löbel. *MCF 1.3 – A network simplex implementation.* Available for academic use free of charge at http://www.zib.de. 2004.

[12] J.B. Orlin. “A faster strongly polynomial minimum cost flow algorithm”. In: *Proc. 20th ACM symposium on Theory of computing, STOC 1988.* ACM. 1988, pp. 377–387. DOI: 10.1145/62212.62249.

[13] J.B. Orlin. “A faster strongly polynomial minimum cost flow algorithm”. In: *Operations research* 41.2 (1993), pp. 338–350. DOI: 10.1287/opre.41.2.338.

A dual of the rectangle-segmentation problem for binary matrices[*]

Thomas Kalinowski

Abstract

We consider the problem to decompose a binary matrix into a small number of binary matrices whose 1-entries form a rectangle. We show that the linear relaxation of this problem has an optimal integral solution corresponding to a well known geometric result on the decomposition of rectilinear polygons.

MSC: 90C27, 90C46

1 Introduction

In the context of intensity modulated radiation therapy several decomposition problems for nonnegative integer matrices have been considered. One of these is the decomposition into a small number of binary matrices whose 1-entries form a rectangle. There is an example showing that in general the linear relaxation of this problem has no optimal integral solution [1]. On the other hand, the same paper contains an algorithm based on the revised simplex method that uses only very few Gomory cuts. In computational experiments, this algorithm provided exact solutions for matrices of reasonable size in short time. In the present paper we consider the special case that the input matrix is already binary: $A \in \{0,1\}^{m\times n}$. In this case the integer optimization problem is equivalent to a well studied geometric problem: the decomposition of a rectilinear polygon into the minimal number of rectangles. Our main result is that the minimal number of rectangles in such a decomposition equals the optimal objective in the relaxed matrix decomposition problem. In other words, the integrality gap vanishes, provided the input matrix is binary. This solves Problem 2 of [1].

2 Notation and problem formulation

Let A be a binary matrix of size $m \times n$. A *rectangle matrix* is an $m \times n-$matrix $S = (s_{ij})$ such that for some integers k_1, k_2, l_1 and l_2 with $1 \leqslant k_1 \leqslant k_2 \leqslant m$ and $1 \leqslant l_1 \leqslant l_2 \leqslant n$, we have

$$s_{ij} = \begin{cases} 1 & \text{if } k_1 \leqslant i \leqslant k_2 \text{ and } l_1 \leqslant j \leqslant l_2, \\ 0 & \text{otherwise.} \end{cases}$$

[*] *The Electronic Journal of Combinatorics* **16**:R89, 2009

The *rectangle segmentation problem* is the following:

RSP. Find a decomposition $A = S_1 + \cdots + S_t$ with rectangle matrices $S_1, \dots, S_t$ such that t is minimal.

A linear relaxation of this problem ist the following.

RSP-Relax. Find a linear combination $A = x_1 S_1 + \cdots + x_t S_t$ with rectangle matrices $S_1, \dots, S_t$ and $0 \leqslant x_i \leqslant 1$ for $i = 1, \dots, t$ such that $x_1 + \cdots + x_t$ is minimal.

The integral problem **RSP** can be formulated in a geometric setup as follows. We associate A with a rectangular $m \times n$-array of unit squares in the plane. The set $P = \{(i,j) \ : \ a_{ij} = 1\}$ corresponds to a rectilinear polygon whose boundary consists of line segments with integer coordinates. Clearly, a solution of the problem **RSP** is precisely the decomposition of P into the minimal number of rectangles. In order to state the solution to the polygon decomposition problem we need some notation. Let N, c and k be the number of vertices, connected components and holes of P, respectively. This notation has to be clarified by two remarks (see Figure 1 for illustrations).

1. If P can be decomposed into two or more polygons which intersect pairwise in isolated vertices these vertices are counted twice and we consider the parts as different connected components.

2. Similarly, if the boundary of a hole intersects the outer boundary or another hole only in isolated vertices these vertices are counted twice.

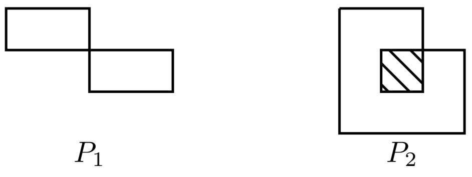

Figure 1: Two polygons. The parameters are $N = 8$, $c = 2$ and $k = 0$ for P_1, and $N = 10$, $c = 1$, $k = 1$ for P_2.

We call a vertex of P *convex* if the interior angle at this vertex is 90° and *concave* if it is 270°. A *chord* of P is a line segment that lies completely inside P, connects two concave vertices and is parallel to one of the coordinate axes. The chords parallel to the x-axis and the y-axis are called *horizontal* and *vertical*, respectively. We associate a bipartite graph $G = (H \cup V, E)$ with P. The vertex sets H and V are the sets of horizontal and vertical chords, respectively, and two chords are connected by an edge if they intersect (Figure 2). Let α be the maximal cardinality of an independent set in the graph associated to P. The following theorem of Lipski et al. (which was reproved by several authors), characterizes the minimal number of rectangles in a decomposition of P.

Theorem 1 ([2, 3, 4])**.** *The minimal number of rectangles in a decomposition of P equals*

$$\frac{N}{2} - c + k - \alpha.$$

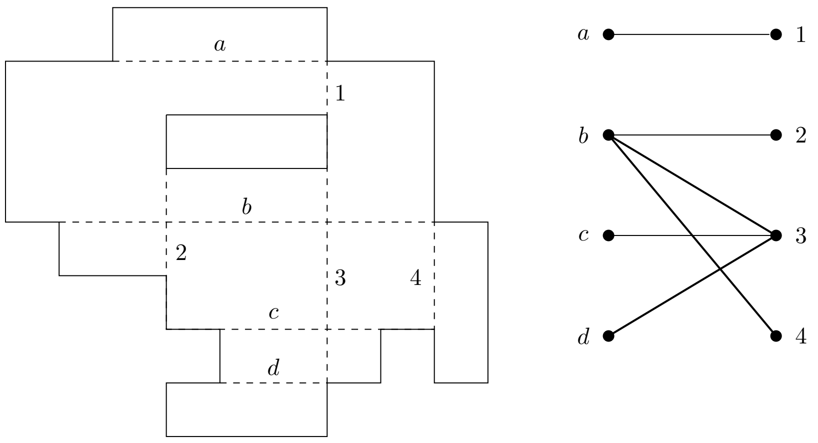

Figure 2: The graph associated to a polygon.

We want to show that there is no integrality gap in the rectangle segmentation problem for binary matrices. This can be done by proving that the optimal objective value for **RSP-Relax** is at least $\frac{N}{2}-c+k-\alpha$. In order to do this we will present a feasible solution for the dual problem with exactly this objective value. For a binary matrix A, the set of variables in **RSP-Relax** corresponds to the set of rectangles that are completely contained in P. Indexing these rectangles by the numbers $1,\dots,T$, i.e. $S^{(1)},\dots,S^{(T)}$ are precisely the rectangle matrices whose 1-entries are contained in P, we can reformulate **RSP-Relax** as follows.

$$\begin{aligned} \sum_{t=1}^{T} s_{ij}^{(t)} x_t &= 1 && \text{for } (i,j)\in P,\\ x_t &\geqslant 0 && \text{for } t=1,\dots,T,\\ \sum_{t=1}^{T} x_t &\to \min. \end{aligned}$$

Dualizing, we obtain the problem **RSP-Dual**:

$$\begin{aligned} \sum_{i=k_1}^{k_2}\sum_{j=l_1}^{l_2} y_{ij} &\leqslant 1 \qquad \text{for } [k_1,k_2]\times[l_1,l_2]\subseteq P,\\ \sum_{(i,j)\in P} y_{ij} &\to \max. \end{aligned}$$

This dual problem has a nice interpretation. The polygon P is considered as a set of unit squares and we want to fill these squares with numbers such that the sum over every rectangle contained in P is bounded by 1, and the total sum is maximized under this constraint. We start with a simple observation which allows us to restrict our attention to dual solutions of a special type. Let $\mathcal{R}$ be the set of rectangles into which P is decomposed when the boundary lines at the concave vertices are extended until they meet the opposite boundary of P. For an illustration see Figure 3, where $|\mathcal{R}| = 16$ and, for instance, the big square in the middle is the element $[2,4] \times [4,6] \in \mathcal{R}$. We call the elements of $\mathcal{R}$ *basic rectangles*. The following lemma asserts that we may assume that only in the upper left

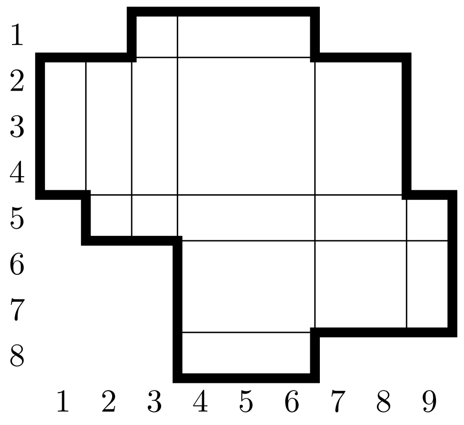

Figure 3: Decomposition of a polygon P into basic rectangles.

corner of a basic rectangle the entry of $\boldsymbol{y}$ is nonzero.

Lemma 1. *Suppose $\boldsymbol{y'} = (y'_{ij})_{(i,j)\in P}$ is feasible for* ***RSP-Dual****, and define $\boldsymbol{y}$ as follows. For every $[i_1,i_2] \times [j_1,j_2] \in \mathcal{R}$ put*

$$y_{ij} = \begin{cases} \sum\limits_{i'=i_1}^{i_2} \sum\limits_{j'=j_1}^{j_2} y'_{i'j'} & \text{for } (i,j) = (i_1,j_1), \\ 0 & \text{for } (i,j) \in ([i_1,i_2] \times [j_1,j_2]) \setminus \{(i_1,j_1)\}\,. \end{cases}$$

Then $\boldsymbol{y}$ is also feasible, and the objective value for $\boldsymbol{y}$ is the same as for $\boldsymbol{y'}$.

Proof. Let $R := [k_1,k_2] \times [l_1,l_2] \subseteq P$, and let $\mathcal{R}_0$ denote the set of basic rectangles having their upper left corner in R. More formally,

$$\mathcal{R}_0 = \{[i_1,i_2] \times [j_1,j_2] \in \mathcal{R} \ : \ (i_1,j_1) \in R\}.$$

The union of the elements of $\mathcal{R}_0$ is a rectangle $R' = [k'_1,k'_2] \times [l'_1,l'_2] \subseteq P$ with

$$\sum_{i=k_1}^{k_2} \sum_{j=l_1}^{l_2} y_{ij} = \sum_{i=k'_1}^{k'_2} \sum_{j=l'_1}^{l'_2} y'_{ij}.$$

Figure 4 illustrates the step from R to R' for $R = [2,5] \times [3,7]$ and $R' = [3,6] \times [3,8]$. Now the feasibility of $\boldsymbol{y'}$ implies

$$\sum_{i=k_1}^{k_2} \sum_{j=l_1}^{l_2} y_{ij} \leqslant 1,$$

and consequently, the feasibility of $\boldsymbol{y}$. The final statement about the objective values is obvious. □

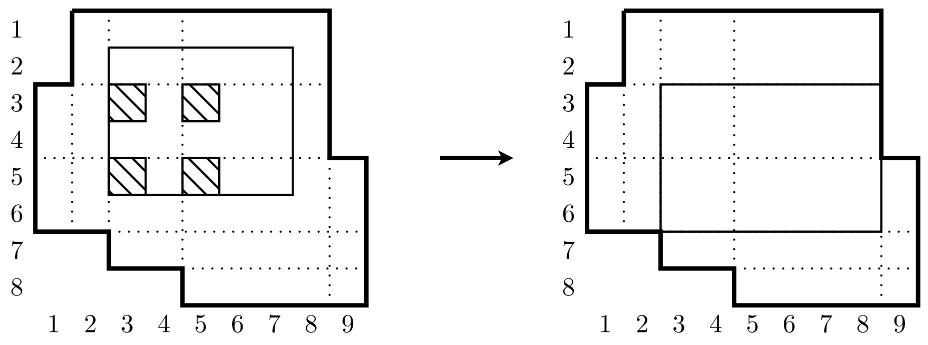

Figure 4: The transition from R to R'. The upper left corners of the elements of $\mathcal{R}_0$ are shaded in the left drawing.

Clearly, a solution $\boldsymbol{y}$ of the form described in Lemma 1 can be identified with the function

$$g : \mathcal{R} \to \mathbb{R}, \quad [k_1, k_2] \times [l_1, l_2] \mapsto y_{k_1,l_1}.$$

This is illustrated in Figure 5, where all the values of g are in $\{0, \pm 1\}$. It will turn out that these special values are sufficient to define an optimal solution for **RSP-Dual**. The

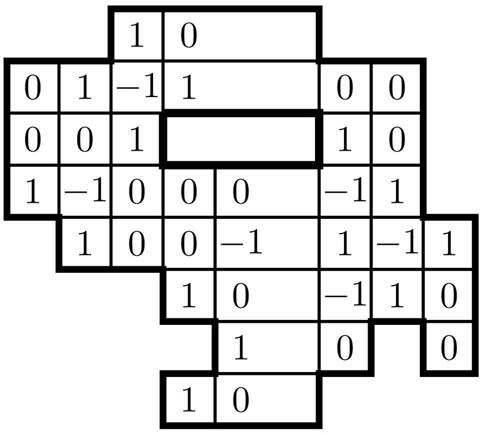

Figure 5: An example solution for **RSP-Dual**.

same argument as in the proof of Lemma 1 shows that for checking the feasibility of such a function g it is sufficient to consider the constraints for rectangles that are unions of basic rectangles. We call these rectangles *essential*.

3 The case $\alpha = 0$

Let us assume $\alpha(G) = 0$, i.e. P has no chords at all. We fix an orientation for the lines that are used to decompose P into basic rectangles. The orientation is defined by pointing away from the concave vertices towards the interior of P (see Figure 6 for an illustration).

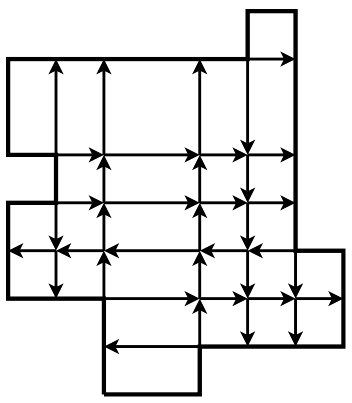
Figure 6: The orientation of the boundaries of the basic rectangles.

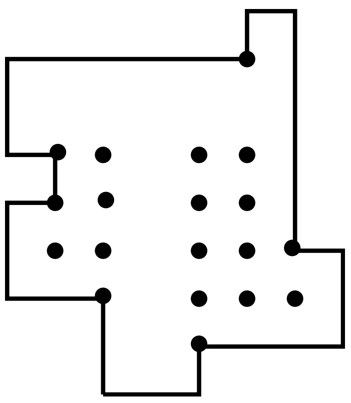
Figure 7: The intersection points.

We will define the values $g(R)$ for basic rectangles R depending on the orientation of the boundary of R. First, we need some additional notation. The vertices of basic rectangles that are either in the interior of P or concave vertices of P are called *intersection points*. The set of all intersection points is denoted by I. In Figure 7 the intersection points are marked by dots.

Definition 1. A vertex a of an essential rectangle $\mathcal{A} \subseteq \mathcal{R}$ is called a *source* (with respect to $\mathcal{A}$) if the two line segments on the boundary of $\mathcal{A}$ that start from a are oriented away from a. In addition, let $q(\mathcal{A}) \in \{0, 1, 2\}$ be the number of sources for $\mathcal{A}$.

Now we can define a function $g : \mathcal{R} \to \{0, \pm 1\}$ which turns out to be an optimal solution for **RSP-Dual**:

$$g(R) = 1 - q(R) \qquad (R \in \mathcal{R}). \tag{1}$$

In order to show the feasibility of this function we observe that the value extends to essential rectangles.

Lemma 2. *For any essential rectangle $\mathcal{A} \subseteq \mathcal{R}$, we have*

$$\sum_{R \in \mathcal{A}} g(R) = 1 - q(\mathcal{A}).$$

*In particular, $g : \mathcal{R} \to \{0, \pm 1\}$ defines a feasible solution for **RSP-Dual**.*

Proof. We proceed by induction on $|\mathcal{A}|$. For $|\mathcal{A}| = 1$, the statement is precisely the definition of g. For $|\mathcal{A}| > 1$, $\mathcal{A}$ is the union of two rectangles $\mathcal{A}_1 \cup \mathcal{A}_2$ as indicated in Figure 8. By induction, we have

$$\sum_{R \in \mathcal{A}_i} g(R) = 1 - q(\mathcal{A}_i) \qquad (i \in \{1, 2\}).$$

The vertices a, b, c and d are sources for $\mathcal{A}$ iff they are sources for the respective $\mathcal{A}_i$. The vertex e is not a source for any of the considered rectangles. Finally, it is easy to see that

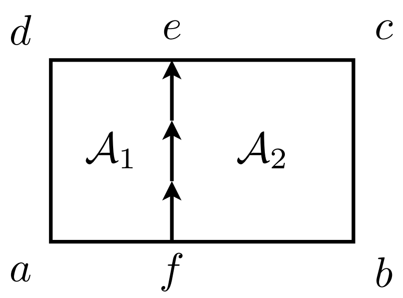

Figure 8: The induction step in the proof of Lemma 2.

f is a source in precisely one of the rectangles $\mathcal{A}_i$, because it is an intersection point. This yields $q(\mathcal{A}) = q(\mathcal{A}_1) + q(\mathcal{A}_2) - 1$, hence

$$\sum_{R\in\mathcal{A}} g(R) = \sum_{R\in\mathcal{A}_1} g(R) + \sum_{R\in\mathcal{A}_2} g(R) = (1 - q(\mathcal{A}_1)) + (1 - q(\mathcal{A}_1))$$
$$= 1 - q(\mathcal{A}). \quad \square$$

We observe that every intersection point a is a source for exactly one basic rectangle with vertex a. Hence by a simple double counting argument, the objective value for the function g is

$$\sum_{R\in\mathcal{R}} g(R) = |\mathcal{R}| - \sum_{R\in\mathcal{R}} q(R) = |\mathcal{R}| - |I|.$$

Our next lemma shows that g is indeed an optimal solution.

Lemma 3. *The objective value for g equals the upper bound from the minimal decomposition of P into rectangles. In other words,*

$$|\mathcal{R}| - |I| = \frac{N}{2} - c + k.$$

Proof. Clearly, we may assume $c = 1$. We proceed by induction on the objective value $h := \frac{N}{2} - 1 + k$. The value $h = 1$ is possible only if P is a single rectangle, and in this case $|\mathcal{R}| - |I| = 1 - 0 = 1$. If $h > 0$, P has at least one concave vertex. Let a be a concave vertex such that no concave vertex is right of a. Along the vertical line through a we cut the polygon P into two polygons P_1 and P_2. The three possible situations are illustrated in Figure 9.

For $i \in \{1, 2\}$, let N_i and k_i be the numbers of vertices and holes of the respective polygons, $\mathcal{R}_i$ the sets of basic rectangles, I_i the sets of intersection points, and let $h_i = N_i/2 - 1 + k_i$ be the corresponding objective values.

Case (a). We have $N = N_1 + N_2 - 2$ and $k = k_1 + k_2$, thus $h = h_1 + h_2$. So h_1 and h_2 are smaller than h and we can apply the induction hypothesis to P_1 and P_2. Let t be the number of intersection points for P that are not in $I_1 \cup I_2$. Clearly, these points lie on the horizontal or on the vertical line through a, as indicated in Figure 10. Let t_1

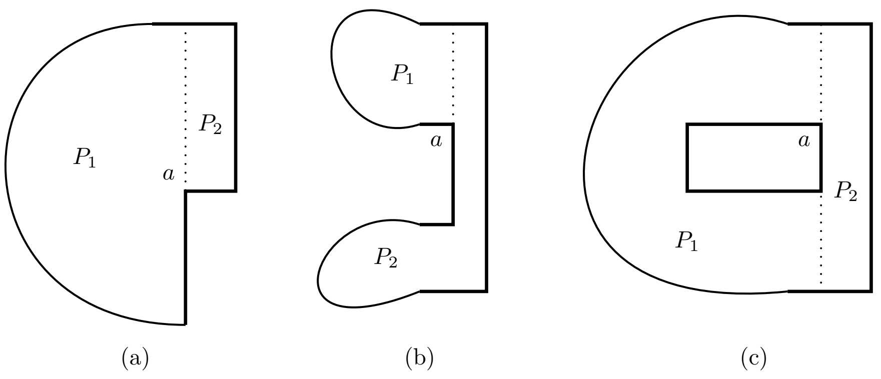

Figure 9: The possible cuts (dotted lines) in the proof of Lemma 3.

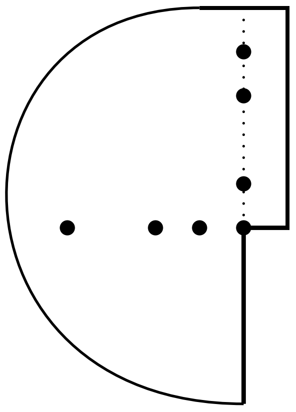

Figure 10: The new intersection points in Case (a).

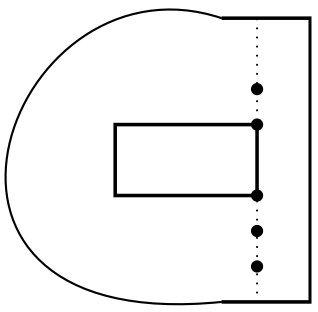

Figure 11: The new intersection points in Case (c).

be the number of new intersection points on the vertical line, except a itself, and let t_2 be the number of new intersection points on the horizontal line (including a), so $t = t_1 + t_2$. In P, P_2 is divided into $t_1 + 1$ basic rectangles (which gives t_1 additional basic rectangles), and exactly t_2 basic rectangles of P_1 are cut into two parts. We obtain

$$|\mathcal{R}| = |\mathcal{R}_1| + |\mathcal{R}_2| + t, \qquad |I| = |I_1| + |I_2| + t,$$

and the claim follows by induction.

Case (b). As in case (a), $h = h_1 + h_2$, and we can apply the induction hypothesis to P_1 and P_2. Again, we see that every new intersection point (all on the vertical line through a) creates a new basic rectangle, and the argument is completed as before.

Case (c). This time we have $N = N_1 + N_2 - 4$ and $k = k_1 + k_2 + 1$, but again $h = h_1 + h_2$, so the induction hypothesis applies. Again the t new intersection points lie on the vertical line through a, and P_2 is divided into $t+1$ basic rectangles (see Figure 11), and this concludes the proof. $\square$

4 The general case

In this section we show how the general case can be handled. In the first subsection we describe the solution g in the general case, while the second subsection is devoted to the proof of the main theorem.

4.1 Definition of the solution

In order to define the solution as in (1), we have to put an orientation on the chords. To fix such an orientation let $Q = H_0 \cup V_0$ be an independent set of size $|Q| = \alpha(G)$, where $H_0 \subseteq H$ and $V_0 \subseteq V$. In addition, let $H_1 = H \setminus H_0$ and $V_1 = V \setminus V_0$. By maximality of Q and Hall's theorem, H_1 can be matched into V_0, and V_1 can be matched into H_0. Let $M \subseteq (H_0 \times V_1) \cup (H_1 \times V_0)$ be such a matching, i.e. $|M| = |H_1| + |V_1|$. Let $Q_0 \subseteq Q$ be the subset of vertices that are not matched in M. In particular, we have $\alpha = |Q_0| + |M|$. We call the elements of Q_0 *isolated chords.* Now the vertical and horizontal isolated chords are oriented from top to bottom and from left to right, respectively. For an edge $xy \in M$, we direct every segment of the chords x and y towards the intersection point of x and y. Figure 12 illustrates this for the polygon in Figure 2 where the underlying matching is $M = \{a1, b2, c3\}$ and the isolated chords are 4 and d (see Figure 2 for the labeling of the chords). Now we can define $g : \mathcal{R} \to \{0, \pm 1\}$ by $g(R) = 1 - q(R)$ as before.

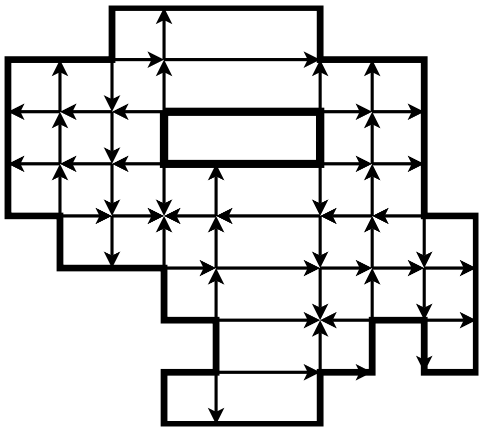

Figure 12: The orientation in the general case.

4.2 Proof of the main theorem

When we try to prove the feasibility of g by induction on $|\mathcal{A}|$ as in Lemma 2, a problem arises in the case that f is the endpoint of a horizontal isolated chord as indicated in the left hand side of Figure 13. Then f is not a source for any of the rectangles $\mathcal{A}_1$ and $\mathcal{A}_2$.

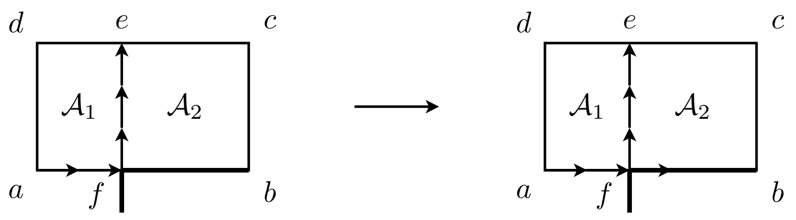

Figure 13: The problem in the induction step for the feasibility of g (and its solution).

The right hand side of the same figure shows a solution for this problem: we just extend the orientation to the first segment of the boundary immediately following the isolated chord. Of course, we have to do this for every isolated chord. Observe that in Figure 12 this is done for both isolated chords (which are 4 and d in this example). After this modification the argument for Lemma 2 proves the following general version.

Lemma 4. *For any essential rectangle $\mathcal{A} \subseteq \mathcal{R}$, we have*

$$\sum_{R\in\mathcal{A}} g(R) = 1 - q(\mathcal{A}).$$

*In particular, $g : \mathcal{R} \to \{0, \pm 1\}$ defines a feasible solution for **RSP-Dual**.*

Now it remains to prove the optimality of g. There are some special intersection points: the points where two chords meet which are matched in M. Clearly, these are never a source for any incident basic rectangle. But each of the remaining intersection points is a source for precisely one basic rectangle. So we obtain the objective value

$$\sum_{R\in\mathcal{R}} g(R) = |\mathcal{R}| - |I| + |M|.$$

Lemma 5. *We have*

$$|\mathcal{R}| - |I| + |M| = \frac{N}{2} - c + k - \alpha.$$

Proof. As in the proof of Lemma 3 we may assume $c = 1$. We proceed by induction on α. The case $\alpha = 0$ was treated in Section 3. So assume $\alpha > 0$, let Q be some independent set of chords of size $|Q| = \alpha$, and choose any chord $ab \in Q$ (w.l.o.g. horizontal). Now we cut the polygon P along the chord ab. We have to distinguish two different cases: either the cut divides P into two polygons P_1 and P_2, or P stays connected, but the number of holes decreases by 1 (see Figure 14 and 17).

Case 1 (Figure 14). For $i \in \{1, 2\}$, denote the parameters of P_i by N_i, k_i and α_i. Similarly, the corresponding sets of basic rectangles and intersection points are denoted by $\mathcal{R}_i$ and I_i, respectively. Observe that P_i inherits a maximum independent set Q_i and a corresponding Matching M_i from P: Q_i is the set of chords in P that are also

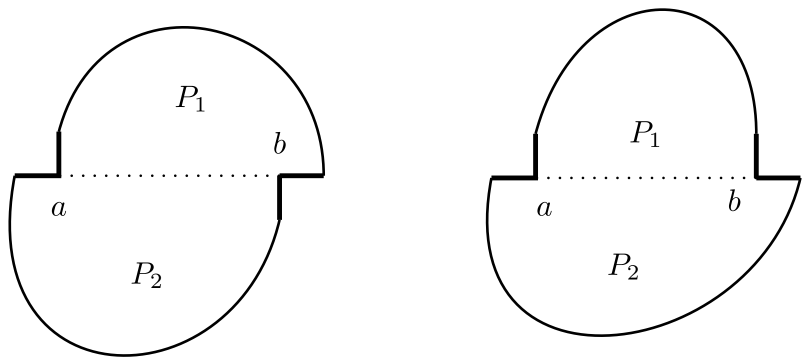

Figure 14: A cut dividing P into two parts (Case 1).

chords in P_i, and M_i is the set of elements $xy \in M$ such that x and y are both in Q_i. We have

$$N = N_1 + N_2, \quad k = k_1 + k_2, \quad \alpha = \alpha_1 + \alpha_2 + 1.$$

Hence, by induction, it is sufficient to prove that

$$|\mathcal{R}| - |I| + |M| = (|\mathcal{R}_1| - |I_1| + |M_1|) + (|\mathcal{R}_2| - |I_2| + |M_2|)\,. \tag{2}$$

Let T be the set of intersection points on the chord ab. This set splits into three subsets. For $i \in \{1, 2\}$, we put

$$T_i := \{u \in T \ : \ u \text{ is a vertex of an element of } \mathcal{R}_i, \text{ but not of an element of } \mathcal{R}_{3-i}.\},$$

and in addition $T_3 = T \setminus (T_1 \cup T_2)$. Observe that T_3 is the set of points where the chord ab meets other chords. For instance, in Figure 15, we have

$$T_1 = \{a_1, a_3\}, \quad T_2 = \{a_2, a_4\}, \qquad T_3 = \{a_5, a_6\}.$$

Next we define a function $\phi : T_1 \cup T_2 \to \mathbb{N}$. For u in T_1, let $\phi(u)$ be the number of intersection points for P on the vertical line through u lying below u. Similarly, for u in T_2, let $\phi(u)$ be the number of intersection points for P on the vertical line through u lying above u. In both cases, the point u itself is included. In the example from Figure 15, we have $\phi(a_1) = 4$, $\phi(a_2) = 2$ and $\phi(a_3) = \phi(a_4) = 3$ (see Figure 16). The function ϕ counts the intersection points of P that are not intersection points of P_1 or P_2:

$$|I| = |I_1| + |I_2| + \sum_{c \in T_1 \cup T_2} \phi(c) + |T_3|. \tag{3}$$

On the other hand, ϕ can be used to count the additional basic rectangles. The vertical line through $u \in T_1$ divides $\phi(u)$ basic rectangles of P_2 into two parts, and the vertical line through $u \in T_2$ divides $\phi(u)$ basic rectangles of P_1 into two parts, hence

$$|\mathcal{R}| = |\mathcal{R}_1| + |\mathcal{R}_2| + \sum_{u \in T_1 \cup T_2} \phi(u). \tag{4}$$

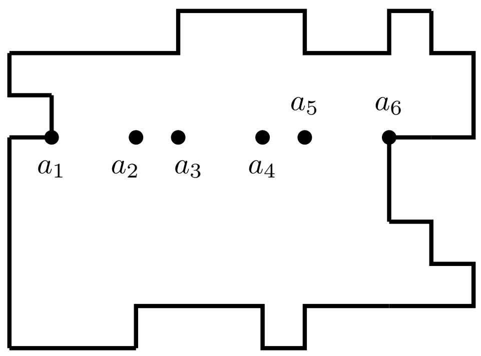

Figure 15: The intersection points on the cutting chord.

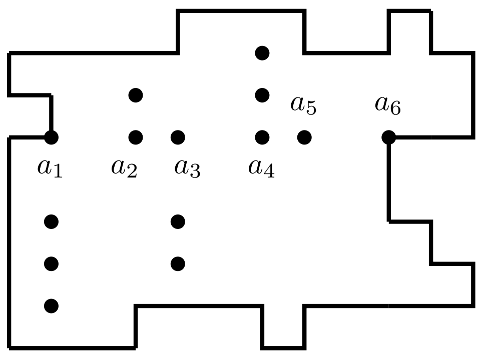

Figure 16: The additional intersection points.

The matching M can be written as a disjoint union $M = M_1 \cup M_2 \cup M_3$ as follows. For $i \in \{1, 2\}$, M_i is the set of edges $xy \in M$ such that $x \in H$, $y \in V$, the chord y does not intersect ab, and both vertices of y are in P_i. This is consistent with the above description of the matchings M_1 and M_2. The remaining matching edges $xy \in M$ such that $x \in H$, $y \in V$, and the chord y intersects ab, are collected in M_3. Clearly, for every such matching edge $xy \in M_3$, the vertical chord y intersects the chord ab in some point from T_3. On the other hand, for every point $a' \in T_3$, the chord y that intersects ab in a' does not belong to our maximal independent set Q, so it is matched in M. Consequently, there is a one-to-one correspondence between M_3 and T_3, and we obtain

$$|M| = |M_1| + |M_2| + |T_3|. \tag{5}$$

Putting together equations (3), (4) and (5), we obtain

$$\begin{aligned}|\mathcal{R}| - |I| + |M| = &\left(|\mathcal{R}_1| + |\mathcal{R}_2| + \sum_{u \in T_1 \cup T_2} \phi(u)\right) \\ &- \left(|I_1| + |I_2| + \sum_{u \in T_1 \cup T_2} \phi(u) + |T_3|\right) + \left(|M_1| + |M_2| + |T_3|\right).\end{aligned}$$

This is (2), and thus concludes the proof in this case.

Case 2 (Figure 17). Essentially the proof is the same as in Case 1. For the parameters of P', we obtain

$$N' = N, \qquad k' = k - 1, \qquad \alpha' = \alpha - 1.$$

So induction applies to P', and we have to show that

$$|\mathcal{R}| - |I| + |M| = |\mathcal{R}'| - |I'| + |M'|. \tag{6}$$

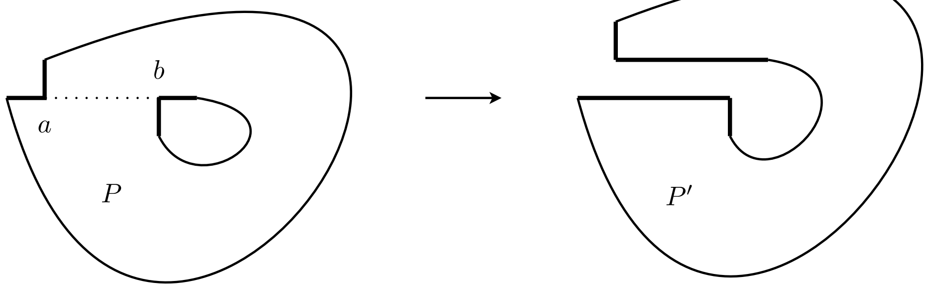

Figure 17: A cut that kills a hole (Case 2).

As before, T is the set of intersection points on the chord ab. In analogy to Case 1, T_1 is the set of $u \in T$ such that u sees a concave vertex when it looks upwards, but u does not see a concave vertex when it looks downwards. Similarly, T_2 is the set of $u \in T$ such that u sees a concave vertex when it looks downwards, but u does not see a concave vertex when it looks upwards, and finally $T_3 = T \setminus (T_1 \cup T_2)$, the set of points where ab meets other chords. For $u \in T_1$, let $\phi(u)$ be the number of intersection points on the vertical line through u lying below u, and for $u \in T_2$, let $\phi(u)$ be the number of intersection points on the vertical line through u lying above u (in both cases we include u itself). By the same counting arguments as in Case 1 we obtain

$$\begin{aligned}
|I| &= |I'| + \sum_{u\in T_1\cup T_2} \phi(u) + |T_3|,\\
|\mathcal{R}| &= |\mathcal{R}'| + \sum_{u\in T_1\cup T_2} \phi(u),\\
|M| &= |M'| - |T_3|.
\end{aligned}$$

These equations imply (6), and this concludes the proof. $\square$

As a consequence of Lemmas 4 and 5, we obtain that g solves **RSP-Dual**.

Theorem 2. *The function $g : \mathcal{R} \to \{0, \pm 1\}$ with $g(A) = 1 - q(A)$ defines an optimal solution for **RSP-Dual**.*

Corollary 1. *There is no integrality gap in the rectangle segmentation problem for binary input matrices.*

References

[1] K. Engel. “Optimal matrix-segmentation by rectangles”. In: *Discr. Appl. Math.* 157.9 (2009), pp. 2015–2030. DOI: 10.1016/j.dam.2008.12.008.

[2] L. Ferrari, P.V. Sankar and J. Sklansky. “Minimal Rectangular Partitions of Digitized Blobs”. In: *Computer Vision, Graphics and Image Processing* 28 (1984), pp. 58–71. DOI: 10.1016/0734-189X(84)90139-7.

[3] W. Lipski, E. Lodi, F. Luccio, C. Mugnai and L. Pagli. “On two dimensional data organization II”. In: *Fund. Informaticae* 2 (1979), pp. 245–260.

[4] T. Ohtsuki. “Minimum dissection of rectilinear regions”. In: *Proc. IEEE Symposium on Circuits and Systems*. 1982, pp. 1210–1213.

Scheduling arc maintenance jobs in a network to maximize total flow over time*

Natashia Boland Thomas Kalinowski Hamish Waterer Lanbo Zheng

Abstract

We consider the problem of scheduling a set of maintenance jobs on the arcs of a network so that the total flow over the planning time horizon is maximized. A maintenance job causes an arc outage for its duration, potentially reducing the capacity of the network. The problem can be expected to have applications across a range of network infrastructures critical to modern life. For example, utilities such as water, sewerage and electricity all flow over networks. Products are manufactured and transported via supply chain networks. Such networks need regular, planned maintenance in order to continue to function. However the coordinated timing of maintenance jobs can have a major impact on the network capacity lost to maintenance. Here we describe the background to the problem, define it, prove it is strongly NP-hard, and derive four local search-based heuristic methods. These methods integrate exact maximum flow solutions within a local search framework. The availability of both primal and dual solvers, and dual information from the maximum flow solver, is exploited to gain efficiency in the algorithms. The performance of the heuristics is evaluated on both randomly generated instances, and on instances derived from real-world data. These are compared with a state-of-the-art integer programming solver.

1 Introduction

We consider a problem in which a network with arc capacities is given, together with, for each arc of the network, a set of maintenance jobs that need to be carried out on the arc. Each maintenance job has a duration, and a time window during which it must start. A maintenance job cannot be pre-empted; once started it will continue for its duration. This situation could arise in a range of network infrastructure settings, for example, when considering maintenance on pipe sections in a water network, or track sections in a rail network. Such maintenance causes network arc outages, leading to capacity reduction in the network. Here we measure network capacity as the value of the maximum flow in the network. This has the advantage of being the simplest way of measuring network capacity. It is also the approach taken by our industry partner in the application that motivated this research. The objective of the problem is to schedule all the maintenance jobs so that the total flow over time is maximized.

*<i></i>*Discrete Applied Mathematics* **163**, 34–52, `doi:10.1016/j.dam.2012.05.027`

We were led to consider this problem through our collaboration with the Hunter Valley Coal Chain Coordinator Limited (HVCCC). The Hunter Valley Coal Chain (HVCC) constitutes mining companies, rail operators, rail track owners and terminal operators, together forming the world's largest coal export facility. In 2008, the throughput of the HVCC was about 92 million tonnes, or more than 10% of the world's total trade in coal for that year. The coal export operation generates around \$15 billion in annual export income for Australia. As demand has increased significantly in recent years and is expected to increase further in the future, efficient supply chain management is crucial. Our industry partner, the HVCCC was founded to enable integrated planning and coordination of the interests of all involved parties, so as to improve the efficiency of the system as a whole. More details on the HVCC can be found in [1].

The problem discussed in this paper was motivated by the annual maintenance planning process carried out by the HVCCC. Supply chain components such as railway track sections, terminal equipment and load points have to undergo regular preventive and corrective maintenance, causing a significant loss in system capacity (up to 15%). The HVCCC had observed that careful scheduling of the maintenance jobs – good alignment of them – could reduce the impact of maintenance on the network capacity, and established a regular planning activity to carry it out, called "capacity alignment". Currently capacity alignment for the approximately 1500 maintenance jobs planned each year is a labour-intensive, largely manual process, achieved by iterative negotiation between the HVCCC and the individual operators.

The HVCCC currently uses an automated rule-based calculator to evaluate the quality of candidate maintenance schedules. In-depth analysis of both the calculator and the HVCC coal handling system revealed this to be well modelled as a maximum flow problem in a network in which the coal flows from the mines to the ships. The arcs represent the relevant pieces of infrastructure: load points, rail track and different machines at the terminals. A maintenance job on a piece of the infrastructure simply means that the corresponding arc cannot carry any flow for the duration of the job. The natural objective is to schedule the maintenance tasks such that the total flow over the time horizon is maximized. This corresponds to, e.g., annual throughput capacity of the HVCC.

The maintenance jobs themselves are scheduled initially according to standard equipment requirements, which typically dictate particular types of maintenance jobs be performed at particular time points. After discussions with the maintenance planners, it emerged that they would be prepared to move the jobs, usually for intervals of plus or minus 7 days, in order to achieve better overall throughput of the system. We initially expected there would be some inter-maintenance constraints, for example, that a type of job carried out at four-week intervals could not be carried out more than 5 weeks apart. But the maintenance planners were not concerned about this issue, and preferred the simple assumption that jobs could not deviate more than some fixed number of days around their initial scheduled time. This gives rise to a simple release date and due date job scheduling structure.

The problem of scheduling maintenance jobs in a network so as to maximize the total flow over time has some aspects of dynamic maximum flow. The concept was introduced by Ford and Fulkerson [5]: given a network with transit times on the arcs, determine the maximum flow that can be sent from a source to a sink in T time units. In the application of interest to us, there are no transit times on arcs, but the capacities vary over time. This leads to a different type of dynamic flow problem. Variations of the dynamic maximum flow problem with zero transit times are discussed in [3, 8, 9], while piecewise constant capacities are investigated by Ogier [14] and Fleischer [4]. For a comprehensive survey on dynamic network flow problems we refer the reader to [11, 18], and for a recent, very general treatment of maximum flows over time to [10]. For a given maintenance schedule, the capacities on the arcs jump between zero and their natural capacity, and so are piecewise constant. Thus the problem of evaluating a maintenance schedule could be viewed as a dynamic maximum flow problem of this type. However, in our case the piecewise constant function is a function of the maintenance schedule, and hence of the schedule decision variables. This makes our problem quite different.

The problem does have a superficial resemblance to machine scheduling problems (see, e.g., the book by Pinedo [15]), but there is no underlying machine, and the association of jobs with network arcs and a maximum flow objective give it quite a different character. Classical machine scheduling seeks to carry out jobs as quickly as possible (in some sense). The maximum flow objective motivates quite different strategies. For example, if arcs are "in sequence" in some sense, it is better to overlap the corresponding maintenance jobs in time as much possible, whereas if they are "in parallel", it is better to schedule them with as little overlap as possible.

There is also some resemblance to network design problems (fixed charge network flows), see e.g. Nemhauser and Wolsey [12] and references therein, but in such problems the arcs are either designed in, or out, of the network in a single-period setting. Even a multi-period variant (see for example the recent work of Toriello *et al.* [19]) would not capture the need for consecutive period outages implied by a maintenance activity.

An emerging research area that also blends network flow and scheduling elements arises in restoration of network services in the wake of a major disruption. For example, Nurre *et al.* [2] schedule arc restoration tasks so as to maximize total weighted flow over time. They consider dispatch rule based heuristics and integer programming approaches. The latter performed well in sewerage, small power infrastructure, and emergency supply chain cases, solving most instances to optimality in a matter of seconds, but the heuristic was competitive in terms of more quickly finding good quality solutions. The heuristic was also especially effective in a large power infrastructure case, finding nearly as good solutions as the exact approach in far less time (see also [13]). We note that the scheduling part of the problem considered in [2] is more similar to a classical scheduling setting than ours is: the restoration activity for each arc needs to be scheduled on a machine, (work group), and one wants to complete all jobs as quickly as possible.

Thus although there are connections of our problem to existing problems, we believe that this is the first time that the problem has been considered. We believe it has a wide range of natural applications, a very attractive structure, with tractable special cases (a few of which we discuss), and some interesting extensions. We thus hope that this paper will stimulate further research on the problem and its variants. Our contributions in this paper are first to define and introduce the problem, prove it is strongly NP-hard, and discuss some tractable special cases. We then propose four different local search heuristics. The heuristics integrate exact maximum flow solutions within a local search framework, exploiting the max flow objective function structure, the availability of both primal and dual solvers, and dual information, to gain efficiency in the algorithms. The heuristics proved to be very effective on both randomly generated and real-world instances, significantly out-performing a pure integer programming approach, particularly on larger, harder problems.

The paper is organized as follows. In Section 2, the problem is formally defined, formulated as an integer program, and proved to be NP-hard. We also outline some tractable special cases. In Section 3, our local search algorithms for solving the problem are presented. Section 4 contains computational results on randomly generated test instances, as well as on two instance derived from real world data. Finally, we summarize the paper in Section 5 and point out some directions for further investigation.

2 Problem Definition and Complexity Results

Throughout we use the notation $[k,l] = \{k, k+1, \dots, l\}$ and $[k] = \{1, 2, \dots, k\}$ for $k, l \in \mathbb{Z}$. Let (N, A, s, s', u) be a network with node set N, arc set A, source s and sink s', and capacities $u_a \in \mathbb{N}$ for $a \in A$. Also, for a node $v \in N$ let $\delta^-(v)$ and $\delta^+(v)$ denote the set of arcs entering and leaving node v, respectively. We consider the network over a time horizon $[T]$. A *maintenance job* j is specified by its associated arc $a_j \in A$, its processing time $p_j \in \mathbb{N}$, its release date $r_j \in [T]$, and its deadline $d_j \in [T]$. Let J be a set of maintenance jobs, and let let J_a denote the set of jobs $j \in J$ with $a_j = a$. For each job $j \in J$ we have to choose a start time $S_j \in [r_j, d_j - p_j + 1]$ within the *time window* for the job. In our model, jobs cannot be preempted, i.e. scheduling a maintenance job to start at time S_j makes the arc a_j unavailable at times S_j, $S_j + 1, \dots, S_j + p_j - 1$. Thus for a given maintenance job schedule $(S_j)_{j \in J}$, the arc a has capacity zero at time t if for some $j \in J_a$, $t \in [S_j, S_j + p_j - 1]$, and u_a otherwise, for each time $t \in [T]$. The problem we consider is to schedule a set J of maintenance jobs so as to maximize the total throughput over the interval $[T]$, i.e. so as to maximize the sum over t of the flows that can be sent from s to s' in the network, given the arc capacities at time $t \in [T]$ implied by the maintenance schedule. In this paper we assume unlimited resources in terms of workforce and machines, i.e. all jobs could be processed at the same time as far as their time windows allow. It is in principle straightforward to add constraints, for instance limiting the number of jobs requiring use of a given resource processed at any given time. We did not do that because in the HVCCC context the input for the optimization consists of initial maintenance schedules for the different parts of the system (rail network and terminals) with relevant resource constraints already taken into account.

For this paper we make the additional assumption that the different jobs associated with an arc do not overlap, i.e. we assume that for any two jobs j and j' on arc a, $[r_j, d_j] \cap [r_{j'}, d_{j'}] = \varnothing$. This assumption can be made without loss of generality, as the general case can be reduced to this case by replacing any arc violating the assumption by a path, distributing the intersecting jobs among the arcs of the path. The reason for making the assumption is to simplify the presentation of the heuristics below: the local effect of moving a job j (i.e. the effect on the capacity of the arc associated with j) depends only on job j.

We formally define the problem via an integer programming formulation, which we also use to provide a baseline for computational testing. We introduce the following variables.

- For $a \in A$ and $t \in [T]$
  - $\phi_{at} \in \mathbb{R}_+$ is the flow on arc a over time interval t,
  - $x_{at} \in \{0,1\}$ indicates the availability of arc a at time t. These variables are not strictly needed, but are included for convenience.
- For $j \in J$ and $t \in [r_j, d_j - p_j + 1]$, $y_{jt} \in \{0,1\}$ indicates if job j starts at time t.

Now we can write down the problem *maximum total flow with flexible arc outages* (**MaxTFFAO**).

$$z = \max \sum_{t=1}^{T} \sum_{a \in \delta^+(s)} \phi_{at} \tag{1}$$

$$\text{s.t.} \sum_{a \in \delta^-(v)} \phi_{at} - \sum_{a \in \delta^+(v)} \phi_{at} = 0 \qquad \big(v \in N \setminus \{s, s'\},\ t \in [T]\big)\,, \tag{2}$$

$$\phi_{at} \leqslant u_a x_{at} \qquad (a \in A,\ t \in [T])\,, \tag{3}$$

$$\sum_{t=r_j}^{d_j - p_j + 1} y_{jt} = 1 \qquad (j \in J)\,, \tag{4}$$

$$x_{at} + \sum_{t'=\max\{r_j, t-p_j+1\}}^{\min\{t, d_j\}} y_{jt'} \leqslant 1 \qquad (a \in A,\ t \in [T],\ j \in J_a)\,. \tag{5}$$

The objective (1) is to maximize the total throughput. Constraints (2) and (3) are flow conservation and capacity constraints, respectively, (4) requires that every job j is scheduled exactly once, and (5) ensures that an arc is not available while a job is being processed.

Example 1. Consider the network in Figure 1 over a time horizon $T = 6$ with the job list given in Table 1. Figure 2 shows that the total throughput can vary significantly depending on the scheduling of the jobs. Observation of this example shows that, all other things being equal, it is better for jobs on arcs that are "in series" to overlap as much as possible, and for jobs on arcs that are "in parallel" to overlap as little as possible. Thus the job on d should overlap as little as possible with the jobs on e and f, which should overlap as much as possible, and the job on a should overlap as much as possible with those on e and f. This is achieved in the second schedule in Figure 2. Of course the situation is more complex for general networks, but the insight can be useful.

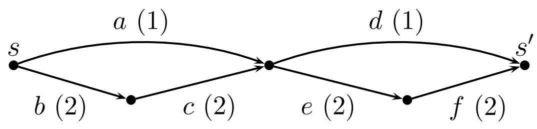

Figure 1: An example network. Capacities are indicated in brackets.

j	arc	p_j	r_j	d_j
1	a	3	1	5
2	d	2	2	5
3	e	2	2	5
4	f	2	3	6

Table 1: Example job list.

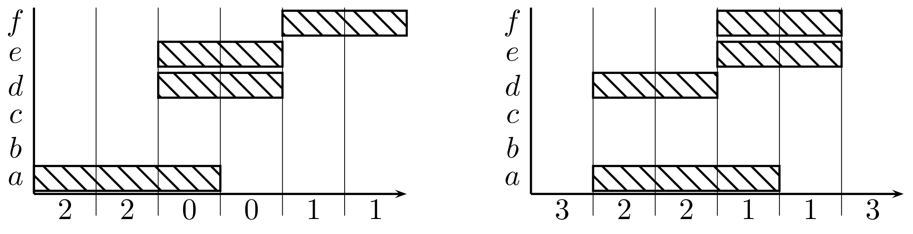

Figure 2: Two schedules for the example problem. In the horizontal direction, we have the 6 unit time intervals, and in the vertical direction there are the 6 arcs. The shaded rectangles indicate the jobs, and below the x-axis is the maximum flow for each time period. The left schedule yields a total flow of 6, while for the right schedule we obtain a total flow of 12.

Next we observe that the problem **MaxTFFAO** is strongly NP-hard, suggesting that in order to tackle instances of practical relevance efficient heuristics might be needed.

Proposition 1. *The problem **MaxTFFAO** is strongly NP-hard.*

Proof. Reduction from 3-partition (see [6]).

Instance. $B \in \mathbb{N}$, $u_1, \dots, u_{3m} \in \mathbb{N}$ with $B/4 < u_i < B/2$ for all i and $\sum_{i=1}^{3m} u_i = mB$.

Problem. Is there a partition of $[3m]$ into m triples (i, j, k) with $u_i + u_j + u_k = B$?

The corresponding network has 3 nodes: s, v and s'. There are $3m$ arcs from s to v with capacities u_i $(i = 1, \dots, 3m)$ and one arc from v to s' with capacity $(m-1)B$ (see Fig. 3).

There is one job with unit processing time for each arc from s to v, with release dates $r_j = 1$ and deadlines $d_j = m$ for all j. It is easy to see that the 3-partition instance has a positive answer if and only if there is a schedule allowing a total flow of $m(m-1)B$. If there is a 3-partition then the i-th of the m triples corresponds to three jobs to be processed in time period i. □

We conclude this section with some remarks on certain special cases.

1. If the network is a directed path and all the jobs have release date $r_j = 1$ and deadline $d_j = T$, it is optimal to start all jobs at the same time, say $S_j = 1$ for all j. This

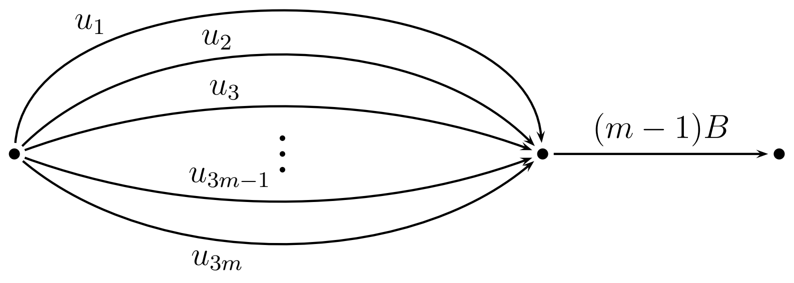

Figure 3: The network for the NP-hardness proof.

follows since the max flow equals the minimum of the arc capacities if all arcs are available, and 0 otherwise. So

$$\min_{a\in A} u_a \cdot \left(T - \max_{j\in J} p_j\right)$$

is an upper bound for the objective z which is attained for the described solution. More generally, if $\bigcap_{j\in J}[r_j, d_j - p_j + 1] \neq \varnothing$, any element t of this intersection determines an optimal solution by putting $S_j = t$ for all $j \in J$.

2. In general, if the network is a path and all jobs have unit processing time, the problem is equivalent to the vertex cover problem on the hypergraph with vertex set $[T]$ and edge set $\{[r_j, d_j] \ : \ j \in J\}$. This is a 0-1 integer programming problem with an interval matrix as coefficient matrix. So it is totally unimodular and can be solved efficiently by linear programming. Another interpretation of this case is that we are looking for a smallest set of time periods, such that all jobs can start at a time given in the set.

3. Inspired by the construction in the hardness proof in Proposition 1, we can ask under what conditions an instance of **MaxTFFAO** with unit processing times and jobs that can move freely ($r_j = 1$ and $d_j = T$) is optimally solved by scheduling all jobs at the same time. For a set $A' \subseteq A$ of arcs let $z_{A'}$ denote the max flow in the network with arc set $A \setminus A'$. Then scheduling all jobs at the same time is always optimal iff

$$\forall A_1, A_2 \subseteq A \qquad A_1 \cap A_2 = \varnothing \implies z_\varnothing + z_{A_1\cup A_2} \geqslant z_{A_1} + z_{A_2}. \tag{6}$$

The if part follows, since if the implication is true, and we are given a solution scheduling jobs at times $t_1 \neq t_2$, we can always shift all the jobs scheduled at t_2 to t_1 without decreasing the objective function. Conversely, if there are disjoint arc sets A_1 and A_2 with $z_\varnothing + z_{A_1\cup A_2} < z_{A_1} + z_{A_2}$, then for an instance with one job on every arc in $A_1 \cup A_2$, it is better to schedule the jobs on A_1 at a different time than the jobs on A_2. Note that the first example of the directed path is a special case of this.

4. Using the characterization (6), we can generalize the path example. Suppose the network $N - s'$ (i.e. the original network without the sink) is a tree, all arcs pointing

away from the source, and in the full network precisely the leaves of this tree are connected to the sink. Assume also that there are no bottlenecks, i.e. for every node $v \neq s$ the capacity of the arc entering v is at least as large as the sum of the capacities of the arcs leaving v. Under these conditions (6) is satisfied, so freely movable jobs with unit processing times should be scheduled at the same time.

3 Local search for MaxTFFAO

3.1 Evaluating the objective function

We consider a solution of **MaxTFFAO** to be specified by the start time indicator variables y_{jt} for all jobs $j \in J$. For given $\boldsymbol{y}$, the values x_{at} can be fixed by

$$x_{at} = 1 - \max_{j \in J_a} \sum_{t'=\max\{r_j, t-p_j+1\}}^{\min\{t, d_j-p_j+1\}} y_{jt'},$$

and then the best solution for the given $\boldsymbol{y}$ can be determined by solving T max flow problems. As a local search framework requires the frequent evaluation of the objective function, we try to make use of the problem structure to design a more efficient method. The following four simple observations indicate potential for such an improvement.

1. A time interval with constant network structure requires only a single max flow computation.
2. If there is a change in the network between time t and time $t+1$, the solution for t can be used as a warm start for $t+1$. As a consequence, the objective function can be evaluated by solving at most $2|J|+1$ max flow problems, and if this number is really necessary, consecutive networks differ in exactly one arc.
3. To update the flows after a change of the schedule we can restrict our attention to the time intervals where the network structure actually changed due to the modification. That means the effect of local changes in the schedule can be determined by solving a short sequence of max flow problems on very similar networks.
4. How the similarity of the networks for different time periods can be used depends on the way the max flow problems are solved. In our LP based implementation (see discussion in Section 3.2 for details) it is natural to reoptimize from the current solution using the primal simplex method if an arc is added and the dual simplex method if an arc is removed.

To make this more precise we introduce more notation for the start times associated with a solution vector $\boldsymbol{y}$: Let $S_j(\boldsymbol{y})$ be the start time of job j, i.e. $S_j(\boldsymbol{y})$ is the unique t with $y_{jt} = 1$. If there is no danger of confusion we will omit the argument $\boldsymbol{y}$ in the notation and just write S_j. Now we can associate with each solution a set of times

$$R = \{S_j, S_j + p_j \ : \ j \in J\} \cup \{1, T+1\}$$

containing exactly the set of times t such that there is a change of capacity on at least one arc between time $t-1$ and time t. The times $t=1$ and $t'=T+1$ can be interpreted this way by adding virtual networks at times 0 and $T+1$ with zero capacities. We denote the elements of R by

$$1 = t_0 < t_1 < \cdots < t_{M-1} < t_M = T+1.$$

The set $[t_{i-1}, t_i - 1]$ is called *time slice* i and its length is denoted by $l_i = t_i - t_{i-1}$. The time slicing is illustrated in Figure 4. In this setup the above observations imply that the

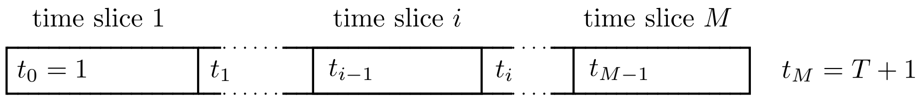

Figure 4: Time slicing.

objective function can be evaluated as described in Algorithm 1.

Algorithm 1 Objective evaluation.

```
Input: Schedule given by $S_j$ for $j \in J$
$R = \{S_j, S_j + p_j \ : \ j \in J\} \cup \{1, T+1\} = \{1 = t_0 < t_1 < \cdots < t_{M-1} < t_M = T+1\}$
Construct the network $(N, A, s, s', u)$
for $i = 1$ to $M$ do
    Update upper bounds of the flow variables according to the outages in time slice $i$
    (Re)solve the network flow problem and store the max flow $z_i$
Output: $z = \sum_{i=1}^{M} z_i \cdot l_i$
```

3.2 Moving single jobs

The feasible region is the set of all binary vectors $\boldsymbol{y} = (y_{jt})_{j\in J, r_j \leqslant t \leqslant d_j}$ satisfying (4). Note that the generation of an initial solution is easy, as we can choose arbitrary start times in the corresponding time windows. A simple neighbourhood is one that is induced by single job movements:

$$N_1(\boldsymbol{y}) = \left\{\boldsymbol{y'} \ : \ S_j(\boldsymbol{y'}) \neq S_j(\boldsymbol{y}) \text{ for at most one job } j\right\}.$$

The size of this neigbourhood is

$$|N_1(\boldsymbol{y})| = 1 + \sum_{j\in J} (d_j - p_j - r_j + 1).$$

In the following we give a characterization of the optimal neighbours, implying an exact method to determine an optimal neighbour.

3.2.1 Preliminary considerations

Moving a job from its current start time S_j to another start time S'_j has two different effects:

1. For any time $t \in [S_j, S_j + p_j - 1] \setminus [S'_j, S'_j + p_j - 1]$ the arc a_j is released and we gain capacity on this arc which could lead to an increase in the max flow for time t.
2. For any time $t \in [S'_j, S'_j + p_j - 1] \setminus [S_j, S_j + p_j - 1]$ we lose the arc, and if it has positive flow in the current max flow, the max flow for time t might decrease.

In order to characterize the impact of a job movement on the objective value we introduce the following parameters measuring the effect of changing the availability status of arc a in time slice i for all $a \in A$ and $i \in [M]$:

- z^+_{ai} is the max flow in the network of time slice i, with arc a added (with capacity u_a) if it is missing in the current solution.
- z^-_{ai} is the max flow in the network of time slice i, with arc a removed if it is present in the current solution.

We start with some simple observations.

- $z^-_{ai} \leqslant z_i \leqslant z^+_{ai}$ for all $a \in A$ and $i \in [M]$.
- $x_{at} = 1$ for $t \in [t_{i-1}, t_i - 1] \implies z^+_{ai} = z_i$.
- $x_{at} = 0$ for $t \in [t_{i-1}, t_i - 1] \implies z^-_{ai} = z_i$.
- For an unavailable arc a (i.e. $x_{at} = 0$ for $t \in [t_{i-1}, t_i - 1]$), releasing arc a increases the max flow by $\Delta^+_{ai} := z^+_{ai} - z_i$.
- For an available arc a (i.e. $x_{at} = 1$ for $t \in [t_{i-1}, t_i - 1]$), removing arc a decreases the max flow by $\Delta^-_{ai} := z_i - z^-_{ai}$.

To efficiently calculate the net effect on the objective, $\Delta_j(S'_j)$, of moving a job j from start time S_j to start time S'_j, one need only consider the set of time slices $\tau^+_j(S'_j)$, defined to be those which are covered by $[S_j, S_j + p_j - 1]$ and that will be (at least partially) uncovered by the move, and the set of time slices $\tau^-_j(S'_j)$, defined to be those which are not covered by $[S_j, S_j + p_j - 1]$ but that will be (at least partially) covered by $[S'_j, S'_j + p_j - 1]$. We also need for each $i \in \tau^+_j(S'_j) \cup \tau^-_j(S'_j)$, the length of the time slice covered by $[S'_j, S'_j + p_j - 1]$, denoted by $l^-_{ij}(S'_j)$. Then

$$\Delta_j(S'_j) = \sum_{i \in \tau^+_j(S'_j)} \Delta^+_{ai} \cdot (l_i - l^-_{ij}(S'_j)) - \sum_{i \in \tau^-_j(S'_j)} \Delta^-_{ai} \cdot l^-_{ij}(S'_j). \tag{7}$$

Provided Δ^+_{ai} and Δ^-_{ai} have been calculated for the appropriate time slices, it is thus straightforward to calculate $\Delta_j(S'_j)$ for any j and S'_j, and hence to determine an optimal neighbour.

Proposition 2. *Finding an optimal neighbour of the given schedule $(S_j)_{j\in J}$ is equivalent to*

$$\max\left\{\Delta_j(S'_j)\ :\ j\in J,\ S'_j\in[r_j,d_j-p_j+1]\right\}.$$

If $\Delta_j(S'_j)\leqslant 0$ for all pairs (j,S'_j), there is no improving solution in the neighbourhood of the current schedule.

3.2.2 The basic method

Proposition 2 immediately suggests a local search strategy: compute $\Delta_j(S'_j)$ for (a subset of) all pairs (j,S'_j), choose one or more pairs with a high value of this bound, perform the corresponding changes of the schedule, and iterate. This could be done naively by first calculating Δ^+_{ai} and Δ^-_{ai} for each time slice i and arc a. The formula (7) shows that we can then easily calculate $\Delta_j(S'_j)$ as required. This approach would appear at first sight to be computationally prohibitive, requiring the solution of two max flow problems to calculate z^+_{ai} and z^-_{ai} for each arc a and time slice i. A number of mitigating factors make this approach more attractive than appearances suggest. First, each arc in a given time slice is either missing or present in the current solution, and so from observations in the previous section, one of z^+_{ai} and z^-_{ai} is given by z_i; it is only for the other that a max flow problem needs to be solved. More importantly,

1. if an arc is added in a time slice where it was previously blocked, the flow stays primal feasible but might no longer be optimal, and
2. similarly, if an arc with nonzero flow is taken out, the dual stays feasible.

Thus the maximum flow problems to be solved in calculating z^+_{ai} and z^-_{ai} can use a primal (dual) method respectively "hot started" from the existing solution for the time slice i. We also observe from (7) that only jobs j with $\Delta^+_{a_ji}>0$ for some time slices i covered by $[S_j,S_j+p_j-1]$ can be moved to give a better solution: these are the *promising* jobs. So we should first determine Δ^+_{ai} to discover the promising jobs, and then only calculate Δ^-_{ai} values as needed for these jobs. Furthermore, Δ^+_{ai} can only be positive if the reduced cost of arc a in the maximum flow problem is positive; otherwise it must be zero. Thus even if the arc is missing from the network, as long as it is included in the original max flow calculation (with zero capacity), and we use a max flow method which makes reduced costs available, we can avoid further max flow calculations (z^+_{ai} can simply be set to z_i if the reduced cost of a in time slice i is not positive).

Algorithm 2 describes how the effects Δ^+_{ai} (adding an arc) and Δ^-_{ai} (blocking an arc) are determined. We do not make explicit here how z^+_{ai} and z^-_{ai} are calculated, since these depend on the specific max flow method used; these implementation issues are discussed in Section 4.1.2. Finally, Algorithm 3 describes the complete procedure of the greedy rescheduling algorithm which will be denoted by `GreedyResched`. In our implementation we use the following three stopping criteria:

1. a time limit,
2. 100 iterations without improvement, and

Algorithm 2 Effects of change.

```
PromisingJobs= $\varnothing$
**for** $i = 1$ **to** $M$ **do**
    $A_i^{\text{out}} = \{a \in A \ : \ x_{at} = 0 \text{ for } t \in [t_{i-1}, t_i - 1]\}$
    **for** $a \in A_i^{\text{out}}$ **do**
        $\Delta_{ai}^+ = z_{ai}^+ - z_i$
        **if** $\Delta_{ai}^+ > 0$ **then**
            Add the job $j$ with $a_j = a$ and time window containing slice $i$ to PromisingJobs
**for** $j \in$ PromisingJobs **do**
    Put $i_0 = \min\{i \ : \ t_i \geqslant r_j\}$ and $i_1 = \max\{i \ : \ t_i \leqslant d_j + 1\} - 1$
    **for** $i = i_0$ **to** $i_1$ **do** $\Delta_{a_j i}^- = z_i - z_{ai}^-$
**Output:** $\Delta_{ai}^+$ – benefit (per time unit) of releasing arc $a$ in time slice $i$
         $\Delta_{ai}^-$ – loss (per time unit) of removing arc $a$ in time slice $i$
         PromisingJobs – set of jobs whose movement could give an improvement
```

Algorithm 3 `GreedyResched`.

```
Initialize time slicing and flow problems (Algorithm 1)
**while** not STOP **do**
    Determine PromisingJobs and the values $\Delta_{ai}^+$ and $\Delta_{ai}^-$ (Algorithm 2)
    **for** $j \in$ PromisingJobs and $S_j' \in [r_j, d_j - p_j + 1]$ **do** calculate $\Delta_j(S_j')$
    **if** $\max_{j,S_j'} \Delta_j(S_j') < 0$ **then** STOP
    **else**
        Choose $(j, S_j')$ with maximal $\Delta_j(S_j')$
        Update time slicing and and resolve the max flow problems with changed input data
```

3. 2 consecutive iterations without improvement and only a single pair (j, S_j') with $\Delta_j(S_j') = 0$.

The reason for the last criterion is that in this situation the algorithm just alternates between two solutions having the same objective value.

3.2.3 Variations

Here we present some natural modifications of the algorithm `GreedyResched`.

Randomization. Instead of choosing the best neighbour in each iteration one can choose randomly from a set of candidates, similar to the strategy applied in the construction phase of *greedy randomized adaptive search procedures* [16]. More precisely, we order the pairs (j, S_j') by nonincreasing value of $\Delta_j(S_j')$ and choose randomly from the first k of this list (uniformly distributed), where k depends on the total number of possibilities, for instance with $K = \#\{(j, S'_j) \;:\; \Delta_j(S'_j) \geqslant 0\}$ denoting the number of moves with nondecreasing objective value we can take

$$k = \max\{\min\{\kappa_1, K\}, \lceil \kappa_2 K \rceil\},$$

where $\kappa_1 \in \mathbb{N}$ and $\kappa_2 \in \{\kappa \in \mathbb{R} \;:\; 0 \leqslant \kappa \leqslant 1\}$ are parameters of the algorithm. After satisfying the stopping criterion, we can restart the algorithm from the initial solution, iterate this, and finally choose the best solution from all runs. We denote this randomized variant by `GreedyRandResched`(κ_1, κ_2). In Figure 5 we plot the behaviour of K in `GreedyResched` for the randomly generated instances we used in our computational experiments (see Section 4). For these experiments we choose $(\kappa_1, \kappa_2) = (5, 0.15)$. Some further possible modifications

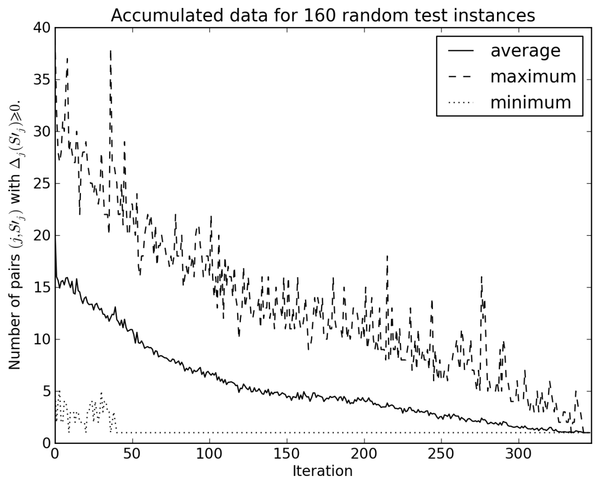

Figure 5: Number of possible moves, i.e. pairs (j, S'_j) with $\Delta_j(S'_j) \geqslant 0$.

to randomization are as follows.

1. Instead of going back to the initial solution each time the stopping criterion is met, we can collect the intermediate solutions with a large value of K (indicating many improving directions) in a solution pool, and choose the starting point for each run from the solution pool (randomly or deterministically).

2. If the computation time until reaching the stopping criterion is large the following combination of the ideas underlying `GreedyResched` and `GreedyRandResched` might be beneficial.

   (a) Do a small number, say k_1, improvements randomly choosing from the improving moves (as in `GreedyRandResched`).

   (b) Repeat the step (a) a small number, say k_2, times.

   (c) Choose the best of the k_2 solutions obtained and continue with step (a).

Testing the effectiveness of these further ideas will be the subject of future research.

Making several moves at a time. In order to speed up the progress of the method we can do several moves corresponding to pairs (j, S'_j) with nonnegative value of $\Delta_j(S'_j)$ simultaneously, if they do not affect the same time slices. This can be implemented by looping over the list of pairs (j, S'_j), ordered by nonincreasing $\Delta_j(S'_j)$, and choosing a pair if its affected time slices do not overlap with those of the pairs already chosen. The benefit of this approach is that it saves recalculations of the values $\Delta_j(S'_j)$, which may be relatively expensive. An iteration of this algorithm can be considered as a greedy accumulation of `GreedyResched` steps, and so we denote the algorithm by `GreedyAccResched`. We also consider a randomized version of this approach. While the list of pairs (j, S'_j), ordered by nonincreasing $\Delta_j(S'_j)$, is non-empty, we choose at random a pair from the first k in the list, and then remove from the list all pairs with affected time slices overlapping those of the chosen pair, before looping again to choose a random pair from the first k. We call this the `GreedyRandAccResched` algorithm.

3.3 Moving multiple jobs

Clearly, there are some limitations in the approach to consider only movements of single jobs. It is easy to construct examples where no single job movement yields an improvement, but moving two jobs at the same time does. However, the benefit of moving jobs simultaneously is only of interest if the jobs interact, in the sense of overlapping in time. We thus propose to search neighbourhoods of the form:

$$\tilde{N}_{j_0}(\boldsymbol{y}) = \left\{\boldsymbol{y'} \ : \ S_j(\boldsymbol{y'}) \neq S_j(\boldsymbol{y}) \text{ only for jobs } j \in J(j_0)\right\},$$

for some $j_0 \in J$, where $J(j_0)$ is the set of jobs whose time window (plus processing time) overlaps with that of j_0, i.e.

$$J(j_0) = \{j \in J \ : \ [r_j, d_j] \cap [r_{j_0}, d_{j_0}] \neq \varnothing\}.$$

The size of $\tilde{N}_{j_0}(\boldsymbol{y})$ is $\prod_{j \in J(j_0)}(d_j - p_j - r_j + 2)$. This is exponential in the number of jobs that have at least two possible start times and overlap with j_0. In particular, the instance used in the proof of Proposition 1 has the property that all job pairs overlap. Thus in general it is NP-hard to optimize over this neighbourhood, and we propose to explore it via a randomized approach as follows.

We consider each job in turn as the *base job*, j_0, and systematically search a selection $C(j_0) \subseteq [r_{j_0}, d_{j_0}]$ of its possible start times. Our idea is that $C(j_0)$ should start small, allowing a "rough" exploration of the alternatives, and increase as the algorithm progresses, thus refining the search. We explain this more precisely later. For each possible start time $S'_{j_0} \in C(j_0)$, we would like to know how "good" that choice of start time is, taking into account interactions of j_0 with other jobs, i.e. we would like to find the best $\boldsymbol{y'}$ such that $S_j(\boldsymbol{y'}) \neq S_j(\boldsymbol{y})$ only for jobs $j \in J(j_0)$. Equivalently, we would like to simultaneously optimize the start times of all jobs in $J(j_0)$, finding a local optimum with respect to j_0. However we expect that doing so would be prohibitive in terms of computational time. Thus we sample from a restricted neighbourhood, restricting the possible start times of jobs in $J(j_0) \setminus \{j_0\}$ heuristically, using the intuition that jobs should either overlap as little, or as much, as possible to get best results. To see where this intuition comes from consider two arcs a and a' with the property that every source-sink path through a also contains a'. If these are the only two arcs with maintenance jobs it is clearly best possible to maximize the overlap between jobs on these arcs. On the other hand, if there is no path containing both arcs a and a', then the total throughput is maximized when the jobs overlap is minimized. Each $j \in J(j_0) \setminus \{j_0\}$ has a set of (up to four) possible start times $C(j)$, so that either the job's start or end aligns with the start or end of job j_0, (assuming j_0 starts at S'_{j_0}). This choice of $C(j)$ is motivated by the fact that in general, there is always an optimal solution such that for every job j one of the following is true.

- Job j starts at its earliest possible start time r_j, or
- job j starts at its latest possible start time $d_j - p_j + 1$, or
- there is a job j' that starts at the same time $S_j = S_{j'}$, or
- there is a job j' that ends at the same time $S_j + p_j - 1 = S_{j'} + p_{j'} - 1$, or
- there is a job j' such that the start of job j aligns with the end of job j', i.e. $S_j = S_{j'} + p_{j'}$, or
- there is a job j' such that the end of job j aligns with the start of job j', i.e. $S_j + p_j = S_{j'}$.

We simply sample randomly from the neighbourhood σ times, choosing the best, for σ an algorithm parameter. This randomized method for moving multiple jobs, denoted by `RandMultiJob`, is described more formally in Algorithm 4.

To implement the method the choice of the candidate start sets $C(j_0)$ has to be specified. For our experiments, $C(j_0)$ consists of k evenly spaced elements in the interval $[r_j, d_j]$, where $k \in \mathbb{N}$. k starts small (at $k = 1$), and increases by one whenever no improvement has been found for a number of consecutive iterations. In our experiments, we use $|J|$ iterations for the increment criterion. Since each job may be the base job multiple times for the same value of k, we want to avoid choosing the same subset of start times every time. Thus we include a mechanism for cycling through sets of k evenly spaced points, modulo the time window width. More precisely, in the m-th run through the outer loop of Algorithm 4, we put

- $W = d_{j_0} - p_{j_0} - r_{j_0} + 2$ (width of the time window of job j_0),
- $\theta = \lfloor (W-1)/k \rfloor$, and
- $C(j_0) = \begin{cases} \{r_{j_0} + (m + i\theta) \pmod W \ : \ i = 0, \dots, k-1\} & \text{if } \theta > 1, \\ [r_{j_0}, d_{j_0} + p_{j_0} + 1] & \text{if } \theta = 1. \end{cases}$

Note that W and θ vary with j_0, but we forgo using a j_0 subscript to improve readability. To illustrate how this works, consider the case that $W = 7$, $k = 3$ and take $r_{j_0} = 1$. Then $\theta = 2$ and when $m \equiv 0 \pmod W$ we get $C(j_0) = \{1, 3, 5\}$, when $m \equiv 1 \pmod W$ we get

Algorithm 4 `RandMultiJob`.

```
Input: A feasible solution $(S_j)_{j\in J}$ with objective value $z$ and parameter $\sigma$
while not Stop do
    for $j_0 \in J$ do
        choose a subset $C(j_0) \subseteq [r_{j_0}, d_{j_0} - p_{j_0} + 1]$
        $J(j_0) = \{j \in J\ :\ [r_j, d_j] \cap [r_{j_0}, d_{j_0}] \neq \varnothing\}$
        Put $\boldsymbol{S} = (S_j)_{j\in J(j_0)}$
        for $S'_{j_0} \in C(j_0)$
            for $j \in J(j_0) \setminus \{j_0\}$ do
                set $C(j) = [r_j, d_j - p_j + 1] \cap \left\{S'_{j_0}, S'_{j_0} - p_j, S'_{j_0} + p_{j_0}, S'_{j_0} + p_{j_0} - p_j + 1\right\}$
            repeat
                for $j \in J(j_0) \setminus \{j_0\}$ do
                    if $C(j) \neq \varnothing$ do choose random $S'_j$ from $C(j)$   else $S'_j = S_j$
                compute the objective $z'$ for starting job $j$ at time $S'_j$ for all $j \in J(j_0)$
                if $z' > z$ then replace $\boldsymbol{S}$ by $(S'_j)_{j\in J(j_0)}$ and $z$ by $z'$
            until done $\sigma$ times

            if enough consecutive iterations with no improvement have passed then increase $k$
Output: An improved solution $(S_j)_{j\in J}$
```

$C(j_0) = \{2, 4, 6\}$, when $m \equiv 2 \pmod W$ we get $C(j_0) = \{3, 5, 7\}$, when $m \equiv 3 \pmod W$ we get $C(j_0) = \{1, 4, 6\}$, etc.

In future work, we will consider allowing σ to vary during the course of the algorithm, by making it dependent on the size of the neighbourhood $\tilde{N}_{j_0}(\mathbf{y})$ at the current solution $\mathbf{y}$, so that more samples are taken from larger neighbourhoods.

4 Computational Experiments

In this section we report on the results of computational tests of our proposed algorithm variants. The first subsection is concerned with randomly generated instances, while the second subsection contains results for two instances coming from real world data.

4.1 Randomly generated instances

We first describe how our random test instances have been generated, then we present the details of the experiments that have been run, and finally, we compare the performance of the considered algorithms.

4.1.1 Instance generation

Our tests are carried out for a time horizon with $T = 1,000$. We need to generate networks and job lists. We generate eight different networks using the RMFGEN generator of Goldfarb and Grigoriadis [7]. For parameters a, b, c_1 and c_2 the generated network has a^2b nodes arranged in b frames of a^2 nodes each. The capacities between frames are randomly chosen from $[c_1, c_2]$, while all capacities inside frames are c_2a^2. We generated 8 different networks for the parameter pairs

$$(a,b) \in \big\{(2,3),\ (2,4),\ (3,2),\ (3,3),\ (3,4),\ (4,2),\ (4,3),\ (4,4)\big\}$$

with $c_1 = 10$ and $c_2 = 20$.

In order to generate a job list for a given network, for each arc we first choose α, the number of jobs. Then we divide the time horizon into α equal subintervals, each of them associated with one of the jobs to be created. For each job we choose a processing time and a number of start time candidates randomly. Finally, we choose a random release date, making sure that it is compatible with the job being completed in its subinterval. This is made more precise in Algorithm 5 where the input parameters are

- X — set of possible number of jobs per arc,
- Y — set of possible processing times, and
- Z — set of possible sizes for start time windows.

Algorithm 5 Generate JobList (X, Y, Z) $\qquad (X, Y, Z \subseteq \mathbb{N})$

```
for $a \in A$ do
  Initialize $J_a = \varnothing$
  choose random $\alpha \in X$ and put $\mu = \lfloor T/\alpha \rfloor$
  for $\eta = 1$ to $\alpha$ do
    choose random $\beta \in Y$ and $\gamma \in Z$
    choose random $r \in \{1, 2, \dots, \mu - \beta - \gamma + 2\}$
    add job with processing time $\beta$, release date $r_j = (\eta - 1)\mu + r$
    and deadline $r_j + \beta + \gamma - 2$ to $J_a$
```

Let $\overline{\alpha}$, $\overline{\beta}$ and $\overline{\gamma}$ be the maximum elements of X, Y and Z, respectively. In order to guarantee feasible job lists we must have

$$(\overline{\gamma} + \overline{\beta} - 1)\overline{\alpha} \leqslant T.$$

As the number of binary variables in the MIP model (1)–(5) is determined by the sizes of the time windows we decided to focus on studying the influence of the set Z. So we fix $X = [5, 15]$, $Y = [10, 30]$ and test two variants for Z.

1. There are a variety of time window sizes: $Z_1 = [1, 35]$ (the *first* instance set).
2. All time windows are large: $Z_2 = [25, 35]$ (the *second* instance set).

For each network and each triple (X, Y, Z_i) we generated 10 instances, giving a total of 160 instances. The network sizes and the average numbers of jobs obtained in this way are shown in Table 2. In Tables 3 and 4 we report the average problem sizes for the MIP formulation (1)–(5).

Small networks				Large networks			
Network	Nodes	Arcs	Jobs	Network	Nodes	Arcs	jobs
1	12	32	303.2	5	36	123	1159.8
2	16	44	421.0	6	32	92	870.7
3	18	57	542.4	7	48	176	1674.2
4	27	90	847.5	8	64	240	2278.0

Table 2: The sizes of the random networks.

Network	# Rows	# Columns	# Nonzeros	# Binaries	Root relaxation solution time (s)
1	50,810	69,925	226,402	36,925	0.5
2	69,714	95,381	312,255	50,381	1.4
3	87,589	122,784	404,722	64,784	1.6
4	137,141	192,544	638,215	101,544	7.0
5	187,607	262,799	886,817	138,799	21.4
6	145,025	197,037	658,174	104,037	5.5
7	265,983	375,552	1,278,099	198,552	37.8
8	361,293	511,157	1,746,054	270,157	92.1

Table 3: Average problem sizes for the first instance set ($Z = [1, 35]$).

4.1.2 Experimental setup

Each of the generated instances is solved by the following methods:

Algorithm CPX. CPLEX with default settings applied to the formulation (1)–(5),

Algorithm GR. `GreedyResched` using CPLEX to solve the max flow subproblems,

Algorithm GRR. `GreedyRandResched` (Section 3.2.3) with parameters $(\kappa_1, \kappa_2) = (5, 0.15)$,

Algorithm GAR. `GreedyAccResched` (Section 3.2.3),

Algorithm GRAR. `GreedyRandAccResched` (Section 3.2.3) with the same parameter values as for GRR, and

Algorithm RMJ σ. `RandMultiJob` with parameter σ.

Network	# Rows	# Columns	# Nonzeros	# Binaries	Root relaxation solution time (s)
1	57,763	75,215	322,204	42,215	1.1
2	79,481	102,783	448,435	57,783	5.8
3	100,483	132,480	583,899	74,480	1.4
4	157,521	207,922	920,092	116,922	33.6
5	214,415	283,148	1,257,642	159,148	427.8
6	165,634	212,561	944,456	119,561	204.6
7	303,721	404,331	1,800,871	227,331	207.5
8	413,640	550,940	2,469,028	309,940	828.7

Table 4: Average problem sizes for the second instance set ($Z = [25, 35]$).

For ease of implementation, and so as to more readily exploit reduced cost information, and access primal and dual algorithm variants, we decided to solve the max flow subproblems in the algorithms `GreedyResched`, `GreedyRandResched`, `GreedyAccResched` and `GreedyRandAccResched` using CPLEX as LP solver instead of implementing combinatorial algorithms. The following three remarks on the implementation of Algorithm 2, which underlies the three greedy approaches, are based on the observations in Section 3.2.2.

1. The value Δ^+_{ai} is computed only if the reduced cost of the arc a is positive in the current solution for time slice i as otherwise there is no potential for improvement.

2. For calculating the values z^+_{ai}, i.e. the gain obtained by adding in arc a, we use the primal simplex method starting from the current max flow for time slice i.

3. For calculating the values z^-_{ai}, i.e. the loss due to taking out arc a, we use the dual simplex method starting from the current max flow for time slice i.

For `GreedyResched`, we also note that among the pairs (j, S'_j) with maximal value of $\Delta_j(S'_j)$ it is always possible to choose one causing at most one time slice split. A simple way to achieve this is to choose the pair with the smallest S'_j: This ensures that S'_j or S'_j+p_j is one of the breakpoints t_i in the current time slicing. We use this approach in our implementation. For `RandMultiJob` we solve the max flow problems using the implementation of the push-relabel algorithm in the Boost library [17]. Here we don't take advantage of the similarities between the networks of different time slices, but we still use the third observation in Section 3.1 that after changing the schedule we only have to reevaluate the flow for time slices that are actually affected by the change. We experimented with $\sigma = 1, 2, 4$ and 8. We found that results for $\sigma = 8$ were dominated by the other values, and that whilst $\sigma = 4$ did give better values than $\sigma = 1$ or 2 on a very small proportion of instances, it did not give the best value over all algorithms for any instance (random or real world). Thus we only present detailed results for $\sigma = 1$ and 2.

For all algorithms, we impose a time limit of 30 minutes, and all of them start with an initial solution given by

$$S_j = \left\lfloor \frac{r_j + d_j}{2} \right\rfloor .$$

All computations are done on a Dell PowerEdge 2950 with dual quad core 3.16GHz Intel Xeon X5460 processors and 64GB of RAM running Red Hat Enterprise Linux 5. CPLEX v12.1 was running in deterministic mode using a single thread.

4.1.3 Results

As a performance measure to compare algorithms we use the relative gap between the algorithm's solution value and the best known solution value over all algorithms. If the best known solution value for an instance I is z^{best} and the current algorithm returns z, its performance measure on that instance is given by

$$\mathcal{P}(I) = \frac{z^{\text{best}} - z}{z^{\text{best}}}.$$

In Figures 6 to 9, we plot for CPLEX and the `Greedy` algorithms the proportion of instances for which the solution found by the algorithm is within a factor of $1 - \tau$ of the best, for increasing values of τ, i.e. we plot

$$\frac{1}{n} \cdot \#\left\{I \ : \ I \text{ instance with } \mathcal{P}(I) \leqslant \tau\right\}$$

as a function of τ, where n is the total number of instances (in our case 80 for each instance set). Note that for the 5 minute plots (Figures 6 and 8) we take z^{best} to be the best known solution over all algorithms after 30 minutes. Tables 5 and 6 contain the average number

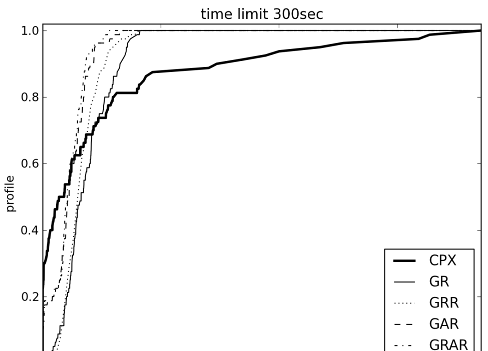

Figure 6: Performance profiles for the first instance set ($Z = [1, 35]$) with computation time limited to 5 minutes.

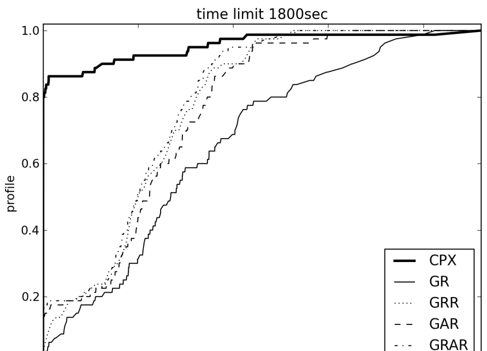

Figure 7: Performance profiles for the first instance set ($Z = [1, 35]$) with computation time limited to 30 minutes.

of max flow problems solved for each of the local search algorithms and every network. For algorithms GR and GAR we also report the run times, GRR, GRAR and RMJ 2 ran for the whole 30 minutes. Tables 7 to 10 provide information about the relative gaps (average and maximal) and the numbers of instances where each algorithm found the best solution, for all algorithms. Here the relative gap is computed as $(z' - z)/z'$ where z' is the best

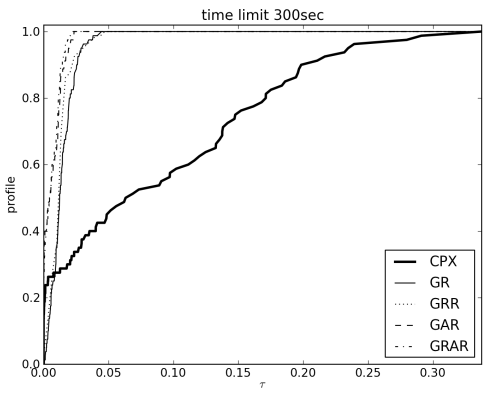

Figure 8: Performance profiles for the second instance set ($Z = [25, 35]$) with computation time limited to 5 minutes.

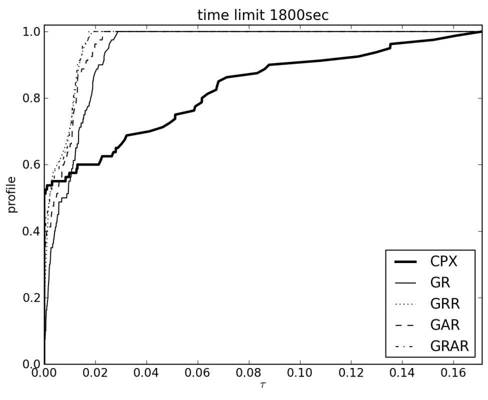

Figure 9: Performance profiles for the second instance set ($Z = [25, 35]$) with computation time limited to 30 minutes.

	GR		GRR	GAR		GRAR	RMJ 2
Network	Time(s)	mf calls	mf calls	Time(s)	mf calls	mf calls	mf calls
1	12	210	31,296	17	179	16,500	96,618
2	25	351	22,741	30	287	13,772	74,739
3	25	460	29,168	40	344	16,386	61,270
4	128	1,256	15,201	100	677	9,681	38,703
5	308	2,007	10,693	324	1,247	7,320	25,605
6	115	838	11,574	115	585	7,041	37,588
7	524	2,984	8,871	519	2,010	6,146	14,827
8	825	3,256	6,607	605	1,540	4,090	8,276

Table 5: Average numbers of max flow problems (divided by 1,000) for the first instance set ($Z = [1, 35]$).

upper bound obtained by CPLEX in 30 minutes, and z is the objective value of the best solution found by the considered algorithm in the respective time (5 or 30 minutes).

We make the following observations.

- For the first instance set CPLEX outperforms all other algorithms, but on the large networks the heuristics, in particular `GreedyAccResched` and `GreedyRandAccResched`, arequite good in providing a reasonably good solution in a short time. The 5 minute performance profiles both show the distinct advantage of `GreedyAccResched` and `GreedyRandAccResched` over the other methods for short run times. For long run times, the 30 minute performance profiles show CPLEX to be the clear winner for the first instance set, with `GreedyRandResched` and `GreedyRandAccResched` best for the second instance set.

	GR		GRR	GAR		GRAR	RMJ 2
Network	time [sec]	mf calls	mf calls	time [sec]	mf calls	mf calls	mf calls
1	14	321	37,818	12	214	30,966	83,170
2	36	627	28,411	23	382	25,900	65,355
3	28	724	38,786	26	490	32,253	54,133
4	160	1,982	19,304	84	1,129	19,702	35,745
5	367	2,928	13,907	229	1,966	7,793	23,858
6	197	1,762	15,022	96	776	7,603	33,656
7	709	4,704	11,133	499	3,485	5,714	14,388
8	956	4,723	8,147	536	2,334	3,808	8,056

Table 6: Average numbers of max flow problems (divided by 1000) for the second instance set ($Z = [25, 35]$).

- For the second instance set, on the small networks CPLEX is still superior, but on the larger networks, the local search heuristics outperform CPLEX, with all heuristics giving solutions with smaller gaps on average for all (large) instances over short run times, and all `Greedy` heuristics giving smaller gaps on average over long run times – significantly smaller for 3 out of the 4 (largest) networks.

- Comparing `GreedyResched` and `GreedyAccResched` we see that in all cases it pays off to save the time for reevaluating the possible moves after each step and thus being able to make more moves in the same amount of time. A similar observation applies to `GreedyRandResched` and `GreedyRandAccResched`, but the benefits of the latter are less pronounced.

- Across the board, randomized greedy algorithms give better results than their non-random counterparts, due to the possibility to escape local minima.

- `RandMultiJob` performs better with $\sigma = 1$ than 2, particularly for larger networks in the second instance set, and for the first instance set, with the shorter run time. On the first instance set with the longer run time, the two are difficult to separate, but $\sigma = 2$ gives better results on more networks, and in particular does better on the difficult case of the sixth network. As might be expected, the `RandMultiJob` algorithms show more significant improvement than the greedy heuristics when given more run time. However in no case do the `RandMultiJob` algorithms outperform the greedy heuristics. (Hence we omit their profiles from Figures 6 to 9, to avoid cluttering them.)

4.2 Instances derived from real world data

The real world maintenance scheduling problem is complicated by additional constraints imposed, for example, by daylight restrictions, availability of equipment or labour force to carry out the maintenance, incompatibility issues between jobs, or conflicts with other

		1	2	3	4	5	6	7	8
	avg gap	0.0	0.4	0.3	1.1	3.0	3.7	3.9	12.9
CPX	max gap	0.1	0.8	1.7	2.6	6.1	4.6	5.4	20.2
	# best sol	10	10	9	10	7	10	8	3
	avg gap	2.1	2.8	1.9	1.6	2.3	4.9	2.1	3.3
GR	max gap	4.1	4.0	3.5	1.9	2.5	5.5	2.7	5.3
	# best sol	0	0	0	0	0	0	0	0
	avg gap	1.5	2.0	1.6	1.5	2.4	4.1	2.4	3.7
GRR	max gap	2.2	3.1	2.4	2.0	2.7	5.2	3.1	6.1
	# best sol	0	0	0	0	0	0	0	0
	avg gap	1.5	2.0	1.5	1.4	2.1	4.0	1.5	1.5
GAR	max gap	1.9	2.4	2.2	1.6	2.5	4.8	1.9	2.2
	# best sol	0	0	1	0	2	0	2	6
	avg gap	1.4	1.8	1.4	1.4	2.0	3.8	1.5	1.5
GRAR	max gap	1.8	2.2	2.2	1.6	2.4	4.2	1.8	2.2
	# best sol	0	0	1	0	1	0	0	1
	avg gap	3.0	5.1	2.0	4.5	7.3	11.1	5.2	5.7
RMJ 1	max gap	4.9	7.1	2.8	6.9	8.5	12.5	6.5	7.6
	# best sol	0	0	0	0	0	0	0	0
	avg gap	2.5	4.6	1.9	4.8	7.3	11.2	5.3	5.9
RMJ 2	max gap	3.8	6.0	2.6	7.1	8.1	13.2	6.7	8.0
	# best sol	0	0	0	0	0	0	0	0

Table 7: Average and maximal relative gaps and number of best solutions found on the first instance set, $Z = [1, 35]$ (runtime 5 minutes).

users of the infrastructure. All of this can be modelled in an MIP framework and taken into account in a local search, both of which are the subject of ongoing work. For the present paper, we ignore the additional constraints and conduct some experiments on pure **MaxTFFAO** instances derived from real world data. The network shown in Figure 10 is a simplified version of the real situation. We generate two instances using the maintenance job lists and the actual maintenance schedules for 2010 and 2011. These job lists contain $1,457$ and $1,234$ jobs, respectively. Based on the level of detail occurring in practice, we use a time discretization of 1 hour, leading to instances with time horizons $T = 365 \cdot 24 = 8,760$. The processing times vary between an hour and several days, while 75% of the jobs have a processing time between 1 and 18 hours. For every job we assume a time window of two weeks, i.e. $d_j = r_j + p_j + 14 \cdot 24 - 2$ for all j. This model leads to really large problems as indicated in Table 11, containing the problem sizes. As a start solution we used a snapshot of the HVCCC maintenance scheduling process. We increased the time limit to 2 hours, and the results are shown in Figures 11 and 12. For clarity, the same results for CPLEX and a selection of the better algorithms is given in Figures 13 and 14.

		1	2	3	4	5	6	7	8
	avg gap	0.0	0.2	0.3	0.3	1.6	2.0	0.9	2.8
CPX	max gap	0.0	0.4	1.7	0.8	2.8	2.7	2.1	6.1
	# best sol	10	10	9	10	7	10	8	3
	avg gap	2.1	2.8	1.9	1.6	2.3	4.9	1.6	1.6
GR	max gap	4.1	4.0	3.5	1.9	2.5	5.5	2.1	2.3
	# best sol	0	0	0	0	0	0	0	0
	avg gap	1.4	1.7	1.5	1.4	2.0	3.7	1.4	1.5
GRR	max gap	2.2	2.2	2.3	1.9	2.3	4.9	2.0	2.2
	# best sol	0	0	0	0	0	0	0	0
	avg gap	1.5	2.0	1.5	1.4	2.1	4.0	1.5	1.5
GAR	max gap	1.9	2.4	2.2	1.6	2.5	4.8	1.9	2.2
	# best sol	0	0	1	0	2	0	2	6
	avg gap	1.3	1.7	1.4	1.3	2.0	3.7	1.4	1.5
GRAR	max gap	1.7	2.1	2.2	1.5	2.4	4.2	1.8	2.2
	# best sol	0	0	1	0	1	0	0	1
	avg gap	2.9	5.0	2.0	3.9	6.3	9.9	4.2	4.6
RMJ 1	max gap	4.9	6.9	2.8	5.7	6.8	11.9	5.7	6.2
	# best sol	0	0	0	0	0	0	0	0
	avg gap	2.5	4.5	1.8	3.9	6.3	9.5	4.2	4.6
RMJ 2	max gap	3.8	5.9	2.6	6.1	7.0	11.0	5.0	6.5
	# best sol	0	0	0	0	0	0	0	0

Table 8: Average and maximal relative gaps and number of best solutions found on the first instance set, $Z = [1, 35]$ (runtime 30 minutes).

We observe that the MIP seems to be really hard. For the 2010 data, CPLEX finds one integer solution with better objective value than the start solution, and for 2011 no improving solution can be found at all. Confirming the results for the random instances, the greedy approaches perform very well in terms of finding high quality solutions quickly. The impacts, i.e. the annual capacity reductions due to maintenance, for the start solutions were 37.6Mt (2010) and 32.5 Mt (2011). Table 12 shows the impact reductions achieved by the different algorithms. We note two features that seem to be different to the behaviour for the random instances.

1. Randomization does not always improve the greedy heuristics. Of course, looking at two instances is very limited evidence, but for the 2011 data the randomized variants give slightly worse results. `GreedyAccResched` gives the best result for this instance.

2. `RandMultiJob` keeps improving even after 2 hours, while the other local search strategies seem to get trapped in local optima comparatively early. Both $\sigma = 1$ and 2 values give better results than any of the greedy heuristics or CPLEX for the 2010 instance,

		1	2	3	4	5	6	7	8
CPX	avg gap	0.1	3.9	0.0	6.3	20.5	21.9	16.1	17.0
	max gap	0.4	6.2	0.1	11.0	25.3	39.5	22.8	24.8
	# best sol	10	10	10	9	0	3	0	0
GR	avg gap	1.8	4.0	1.6	1.9	3.4	8.2	2.5	3.6
	max gap	2.5	4.7	2.1	2.8	4.6	10.0	3.4	5.6
	# best sol	0	0	0	0	1	0	0	2
GRR	avg gap	1.4	3.2	1.2	1.9	3.5	7.7	2.6	4.0
	max gap	2.1	4.0	1.4	2.7	4.8	10.8	3.2	5.8
	# best sol	0	0	0	0	1	2	0	1
GAR	avg gap	1.4	3.4	1.2	1.8	2.9	7.7	1.4	1.3
	max gap	2.3	4.5	1.4	2.5	3.5	9.1	1.8	1.9
	# best sol	0	0	0	0	7	3	4	4
GRAR	avg gap	1.3	3.0	1.2	1.8	2.8	7.7	1.4	1.4
	max gap	2.3	3.9	1.4	2.4	3.5	9.4	1.8	1.9
	# best sol	0	0	0	1	1	2	6	3
RMJ 1	avg gap	2.0	6.5	1.4	4.6	8.1	15.3	5.1	4.9
	max gap	3.0	8.2	1.7	8.3	9.3	18.8	5.9	6.6
	# best sol	0	0	0	0	0	0	0	0
RMJ 2	avg gap	1.9	6.5	1.4	4.7	8.2	15.5	5.0	5.0
	max gap	2.8	8.2	1.7	8.7	10.0	18.4	5.9	6.6
	# best sol	0	0	0	0	0	0	0	0

Table 9: Average and maximal relative gaps and number of best solutions found on the second instance set, $Z = [25, 35]$ (runtime 5 minutes).

with $\sigma = 2$ giving the best result overall.

		1	2	3	4	5	6	7	8
CPX	avg gap	0.0	1.9	0.0	1.6	7.4	8.9	6.6	13.3
	max gap	0.1	3.3	0.1	4.9	10.8	14.2	10.1	18.3
	# best sol	10	10	10	9	0	3	0	0
GR	avg gap	1.8	4.0	1.6	1.9	3.1	8.2	1.5	1.3
	max gap	2.5	4.7	2.1	2.8	4.1	10.0	2.0	1.6
	# best sol	0	0	0	0	1	0	0	2
GRR	avg gap	1.3	3.1	1.2	1.8	2.8	7.2	1.4	1.3
	max gap	1.8	3.9	1.4	2.5	3.5	9.0	1.8	1.5
	# best sol	0	0	0	0	1	2	0	1
GAR	avg gap	1.4	3.4	1.2	1.8	2.9	7.7	1.4	1.3
	max gap	2.3	4.5	1.4	2.5	3.5	9.1	1.7	1.9
	# best sol	0	0	0	0	7	3	4	4
GRAR	avg gap	1.3	3.0	1.2	1.7	2.8	7.3	1.4	1.3
	max gap	1.8	3.8	1.4	2.3	3.5	8.5	1.8	1.9
	# best sol	0	0	0	1	1	2	6	3
RMJ 1	avg gap	2.0	6.4	1.4	3.9	7.0	13.9	4.1	4.0
	max gap	3.0	8.2	1.6	7.2	7.9	17.4	5.2	5.8
	# best sol	0	0	0	0	0	0	0	0
RMJ 2	avg gap	1.9	6.2	1.3	3.8	7.1	14.1	4.2	4.0
	max gap	2.8	8.2	1.7	7.0	8.2	17.1	4.9	5.7
	# best sol	0	0	0	0	0	0	0	0

Table 10: Average and maximal relative gaps and number of best solutions found on the second instance set, $Z = [25, 35]$ (runtime 30 minutes).

	# Rows	# Columns	# Nonzeros	# Binaries	Root relaxation solution time (s)
2010	2,741,944	3,317,433	13,500,830	1,775,673	3,823
2011	2,735,919	3,310,352	13,226,945	1,768,592	7,154

Table 11: Problem sizes for the instances derived from real world data.

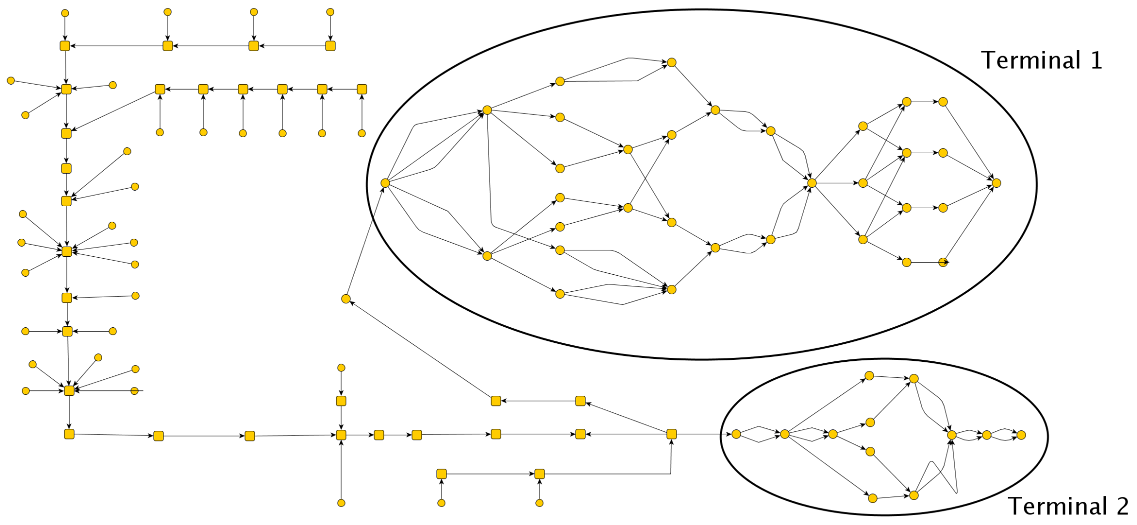

Figure 10: The HVCC network. The circled parts of the network represent the flow of coal through terminal handling equipment. The rest represents the rail network, sourcing coal from 33 coal load points.

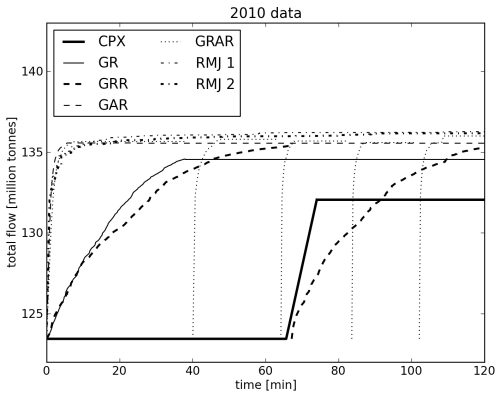

Figure 11: Progress for the 2010 data.

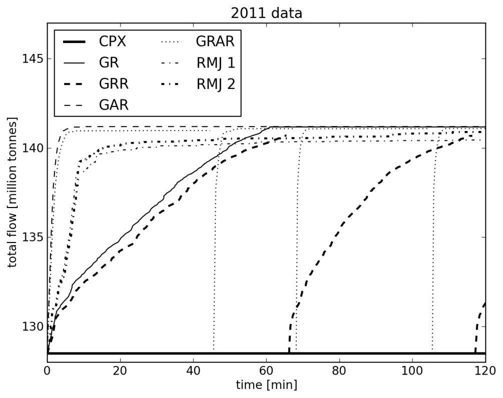

Figure 12: Progress for the 2011 data.

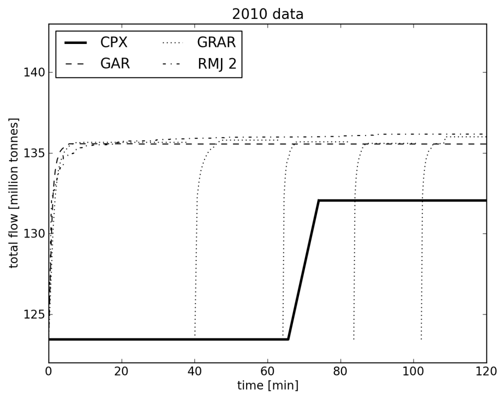

Figure 13: Progress for the 2010 data (selected algorithms).

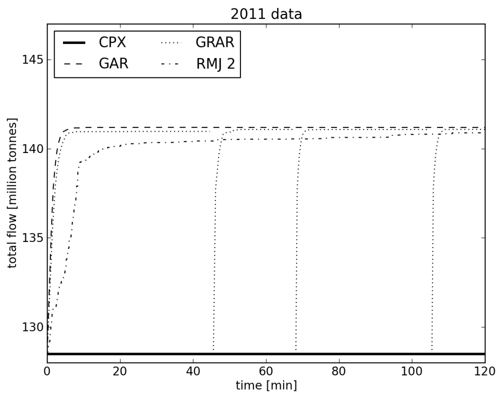

Figure 14: Progress for the 2011 data (selected algorithms).

	2010			2011		
	Impact (Mt)	Reduction (%)	Gap (%)	Impact (Mt)	Reduction (%)	Gap (%)
CPX	28.9	22.9	7.8	32.5	0.0	13.4
GR	26.4	29.6	6.1	19.8	39.0	4.9
GRR	25.6	31.8	5.5	20.3	37.5	5.2
GAR	25.4	32.3	5.4	19.8	39.1	4.9
GRAR	25.0	33.4	5.1	19.9	38.7	4.9
RMJ 1	24.8	33.9	4.9	20.5	36.8	5.4
RMJ 2	24.6	34.6	4.8	20.4	37.3	5.3

Table 12: Impact reduction obtained by the different local search strategies. The column labeled "Gap" contains the relative gap to the best known upper bound from CPLEX.

5 Future directions

We want to point out three directions for further investigation, other than those already indicated in the paper.

1. A natural idea is to develop the local search towards a *Greedy Randomized Adaptive Search Procedure* (GRASP) [16]. That means, instead of using a fixed start solution, start solutions are constructed in a randomized greedy manner.

2. As the general problem is NP-hard, it is interesting to look for special cases (special in terms of the network structure and/or in terms of properties of the job list) that can be solved efficiently.

3. In the other direction, there might be generalizations of the problem that are worth studying, for instance allowing

    - arbitrary subsets of $[T]$ as sets of possible start times (not only intervals $[r_j, d_j]$), and
    - job processing to only reduce the arc capacity by some fraction, rather than taking it out completely.

    The former arises in the Hunter Valley coal chain application in respect of rail track maintenance, where crews must work during daylight hours of the working week. The latter obviously arises in contexts such as highway maintenance, where lane closures and slow-downs come into effect.

These examples of possible future directions illustrate what an exciting new problem we believe maximum total flow with flexible arc outages to be, with great potential for both theoretical and practical development.

Acknowledgment

We like to acknowledge the valuable contributions of Jonathon Vandervoort, Rob Oyston, Tracey Giles, and the Capacity Planning Team from the Hunter Valley Coal Chain Coordinator (HVCCC) P/L. Without their patience, support, and feedback, this research could not have occurred. We also thank the HVCCC and the Australian Research Council for their joint funding under the ARC Linkage Grant no. LP0990739. Furthermore, we are very grateful to the anonymous referees for their helpful comments and suggestions.

References

[1] N. Boland and M. Savelsbergh. “Optimizing the Hunter Valley coal chain”. In: *Supply Chain Disruptions: Theory and Practice of Managing Risk.* Ed. by H. Gurnani, A. Mehrotra and S. Ray. Springer-Verlag London Ltd., 2011, pp. 275–302.

[2] B. Cavdaroglu, J.E. Mitchell, S.G. Nurre, T.C. Sharkey and W.A. Wallace. *Restoring infrastructure systems: An integrated network design and scheduling problem.* Tech. rep. `www.rpi.edu/~sharkt/RIS.pdf` (13 November 2011). Rensselaer Polytechnic Institute, 2010.

[3] L. Fleischer. "Faster algorithms for the quickest transshipment problem". In: *SIAM journal on Optimization* 12.1 (2001), pp. 18–35. DOI: `10.1137/S1052623497327295`.

[4] L. Fleischer. "Universally maximum flow with piecewise-constant capacities". In: *Networks* 38.3 (2001), pp. 115–125. DOI: `10.1002/net.1030`.

[5] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton, N.J.: Princeton Univ. Press, 1962.

[6] M.R. Garey and D.S. Johnson. *Computers and intractability, a guide to the theory of NP–completeness.* W.H. Freeman, 1979.

[7] D. Goldfarb and M.D. Grigoriadis. "A computational comparison of the Dinic and network simplex methods for maximum flow". In: *Annals of Operations Research* 13.1 (1988), pp. 81–123. DOI: `10.1007/BF02288321`.

[8] B. Hajek and R.G. Ogier. "Optimal dynamic routing in communication networks with continuous traffic". In: *Networks* 14.3 (1984), pp. 457–487. DOI: `10.1002/net.3230140308`.

[9] B. Hoppe and É. Tardos. "Polynomial time algorithms for some evacuation problems". In: *Proc. 5th ACM-SIAM symposium on discrete algorithms SODA 1994.* Society for Industrial and Applied Mathematics. 1994, pp. 433–441.

[10] R. Koch, E. Nasrabadi and M. Skutella. "Continuous and discrete flows over time". In: *Mathematical Methods of Operations Research* 73.3 (2011), pp. 301–337. DOI: `10.1007/s00186-011-0357-2`.

[11] B. Kotnyek. *An annotated overview of dynamic network flows.* Tech. rep. 4936. `http://hal.inria.fr/inria-00071643/` (20 February 2013). INRIA, 2003.

[12] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[13] S.G. Nurre and T.C. Sharkey. "Restoring infrastructure systems: An integrated network design and scheduling problem". In: *Proceedings of the 2010 Industrial Engineering Research Conference.* 2010.

[14] R.G. Ogier. "Minimum-delay routing in continuous-time dynamic networks with piecewise-constant capacities". In: *Networks* 18.4 (1988), pp. 303–318. DOI: `10.1002/net.3230180405`.

[15] M.L. Pinedo. *Scheduling: theory, algorithms, and systems.* Springer, 2012.

[16] L.S. Pitsoulis and M.G.C. Resende. "Greedy randomized adaptive search procedures". In: *Handbook of applied optimization.* Ed. by P.M. Pardalos and M.G.C. Resende. Oxford University Press, 2002, pp. 168–183.

[17] J.G. Siek, L.-Q. Lee and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. C++ In-Depth. Addison-Wesley Professional, 2001.

[18] M. Skutella. "An introduction to network flows over time". In: *Research Trends in Combinatorial Optimization* (2009), pp. 451–482. DOI: `10.1007/978-3-540-76796-1`.

[19] A. Toriello, G. Nemhauser and M. Savelsbergh. "Decomposing inventory routing problems with approximate value functions". In: *Naval Research Logistics* 57.8 (2010), pp. 718–727. DOI: `10.1002/nav.20433`.

Mixed integer programming based maintenance scheduling for the Hunter Valley coal chain*

Natashia Boland Thomas Kalinowski Hamish Waterer Lanbo Zheng

Abstract

We consider the scheduling of the annual maintenance for the Hunter Valley Coal Chain. The coal chain is a system comprising load points, railway track and different types of terminal equipment, interacting in a complex way. A variety of maintenance tasks have to be performed on all parts of the infrastructure on a regular basis in order to assure the operation of the system as a whole. The main objective in the planning of these maintenance jobs is to maximize the total annual throughput. Based on a network flow model of the system we propose a mixed integer programming formulation for this planning task. In order to deal with the resulting large scale model which cannot be solved directly by a general purpose solver, we propose two steps. The number of binary variables is reduced by choosing a representative subset of the variables of the original problem, and a rolling horizon approach enables the approximation of the long term (i.e. annual) problem by a sequence of shorter problems (for instance monthly).

Keywords. maintenance scheduling, coal supply chain, capacity alignment, network flow, mixed integer programming

1 Introduction

The Hunter Valley Coal Chain (HVCC) consists of mining companies, rail operators, rail track owners and terminal operators, together forming the world's largest coal export facility. In 2008, the throughput of the HVCC was about 92 million tonnes, or more than 10 per cent of the world's total trade in coal for that year. The coal export operation generates around $15 billion in annual export income for Australia. As demand has increased significantly in recent years and is expected to increase further in the future, efficient supply chain management is crucial. Our industry partner, the Hunter Valley Coal Chain Coordinator Limited (HVCCC) was founded to enable integrated planning and coordination of the interests of all involved parties, so as to improve the efficiency of the system as a whole. More details on the HVCC can be found in [4].

In this paper we are concerned with the infrastructure that is necessary to bring the coal all the way from the mining areas in the Hunter Valley onto vessels transporting it to the final destination. The coal has to go by rail to one of the terminals in the port of Newcastle, where it is assembled and finally loaded onto a vessel. There is a natural subdivision of the chain into three parts.

Journal of Scheduling **16**(6), 649–659, doi:10.1007/s10951-012-0284-y

1. The rail network between the load points at the mines and the terminals.
2. The *inbound* part of the terminals. The coal is unloaded from the trains at dump stations, transported to the stockyard via conveyor belts and stacked onto pads in the stockyard.
3. The *outbound* part of the terminals. The coal is reclaimed from the stockyard, and loaded onto the vessel at the berth.

We discuss the annual maintenance planning process carried out by the HVCCC. Supply chain components such as railway track sections and terminal equipment have to undergo regular preventive and corrective maintenance, causing a significant loss in system capacity (up to 15%). The HVCCC had observed that careful scheduling of the maintenance jobs – good alignment of them – could reduce the impact of maintenance on the network capacity, and establish a regular planning activity to carry it out, called "capacity alignment". Currently capacity alignment for the approximately 1,500 maintenance jobs planned each year is a labour-intensive, largely manual process, achieved by iterative negotiation between the HVCCC and the individual operators. The items whose maintenance is considered in the process come in three groups corresponding to the above partition of the coal chain:

1. railway track sections,
2. terminal inbound, in particular, dump stations and stackers, and
3. terminal outbound, in particular, reclaimers, ship loaders and berths.

The HVCCC currently uses an impact calculator written in a business rules management system to evaluate the quality of proposed maintenance schedules. This calculator is integral to the HVCCC Capacity Model, a software developed by the HVCCC. For a given set of maintenance activities, it determines three numbers: a rail track impact, a terminal inbound impact and a terminal outbound impact. The total system impact is taken to be the maximum of these three numbers. Summing over the time intervals of constant maintenance yields a single total impact for the full time horizon. In-depth analysis of rules for the terminal impacts and the HVCC coal handling system revealed that the rules can be well captured by solving maximum flow problems in certain networks. The arcs in these networks represent the different terminal machines, and a maintenance job simply means that the corresponding arc cannot carry any flow for the duration of the job. The railway track network is represented very coarsely in the HVCCC impact calculator. Basically, the impact of a track section outage is taken to be the sum of the expected demands (scaled to the duration of the outage) of the load points L with the property that the outage prevents trains from going between L and the terminals. This is motivated by the fact that, due to the tree structure of the rail network, there is a unique route from each load point to each terminal, and the terminals are very close to each other. This means that as long as the railway is not a bottleneck the sum of the affected load point demands corresponds indeed to the reduction of the system capacity. In the future, with increasing demands the rail network will become a bottleneck, so it is more accurate to model it also as a network, the nodes representing junctions and the arcs representing track sections. The additional conceptual advantage of having network models for all parts of the system is that they can easily be connected to capture the interaction between different parts. In particular, the link between the inbound and the outbound part of the coal chain is an important feature of our proposed model that is missing in the current impact calculator. So the buffering function of the stockyard, where the coal typically stays for between three and ten days, can be taken into account. This has the potential to enable more efficient coordination of inbound and outbound outages.

The maintenance jobs are scheduled initially according to standard equipment requirements, which typically dictate particular types of maintenance jobs to be performed at particular time points. Initial schedules are produced by the providers (track owners and terminal operators) more or less independently of each other. Based on this an iterative process begins, in which the coordinator evaluates the schedule, works out options for modifications that can release capacity, and negotiates these proposed changes with the providers. The purpose of our model is to support this process by providing an efficient way to explore and evaluate a variety of different rescheduling options.

Our paper is organized as follows. After a brief review of the relevant literature in Section 2, Section 3 contains a precise definition of the considered maintenance scheduling problem. This includes the underlying network model, the form of the given initial maintenance schedule, and the rescheduling rules that have to be followed by the optimizer. In Section 4, a mixed integer programming (MIP) formulation is given, and Section 5 contains two heuristic reduction steps necessary in order to make the problem computationally tractable. Computational results for two different real world data sets are presented in Section 6, and the final Section 7 contains concluding remarks and some directions for further investigation.

2 Literature Review

Maintenance in general is an essential activity of many production and transportations systems, and in many cases accounts for a significant part of the cost of the operation. Maintenance is usually categorized as either preventive or corrective. The former is a regular, planned activity, whereas the latter is carried out in response to a failure or breakdown. A third category, known as predictive maintenance, concerns the use of measurement to predict and diagnose equipment condition. We refer the reader to Sharma *et al.* [13] for an overview of maintenance activities and their optimization. In this work we are concerned with planning preventive maintenance activities.

Much of the literature on preventive maintenance concerns identification of maintenance policies: how often to perform each type of maintenance, or under what conditions. In addition to the review of Sharma *et al.* [13], we refer the reader to the review of Budai *et al.* [6], which recognizes the significant advantages that can be realized in taking into account the impact on production when planning maintenance. This paper categorizes approaches as either taking into account the production impact of maintenance in maintenance planning, taking into account resource implications (e.g. manpower) in maintenance scheduling, and production planning subject to maintenance requirements. Three streams of research are identified: those focused on costing maintenance activity, those investigating the impact of carrying out maintenance at opportune moments (when a breakdown or other interruption has occurred), and those which schedule maintenance in line with production. The HVCCC process has elements of the first and third: (1) the "cost" of a maintenance plan is assessed in terms of its impact on throughput, and is used to compare alternative plans; and (2) by seeking to re-time maintenance (align it, discussed further in the next paragraph), the HVCCC is seeking to schedule maintenance in line with production.

In our work, the preventive maintenance policy is already set, which yields initial schedules for all components of the system. Rail and terminal equipment have initial maintenance schedules largely directed by their corresponding maintenance policy. However the maintenance policies do not determine the times for individual maintenance events exactly. There is some flexibility which can be exploited to re-time certain jobs, or align them, so as to reduce the impact of the maintenance on the system capacity. Most of the literature that considers scheduling maintenance in line with production addresses maintenance policy setting. Exceptions occur in power generation, when maintenance outages must be scheduled, for example, as in [8, 9]. These are primarily concerned with minimizing the overall cost of maintenance and generation, while ensuring generation is sufficient to meet demand; in our context, by contrast, the cost of maintenance activities is fixed, and we want to maximize system capacity. Other exceptions can be found in the transport sector, for example, buses or airlines. These usually focus on the resource implications of maintenance, as in [10] and [11], or on how to cover the scheduled transport operations while still meeting maintenance requirements, as in [1]. The focus is on scheduling maintenance for the transport equipment, as opposed to the infrastructure over which it moves; the latter is closer to our case of interest.

Rail track and road maintenance does address transport infrastructure, and a number of interesting studies are available, particularly for rail maintenance, see, for example, [5]. In these cases, the focus is on minimizing the disruption to scheduled activities, not on maximizing the number of trains that could be pushed through the system; the latter would be closer to the situation of interest to us. However there are still some relevant papers: we discuss the most closely relevant and recent one, that of Budai *et al.* [7], in some detail further below.

In production systems, which perhaps more closely resemble the HVCCC situation, multi-component system maintenance would appear to be relevant, since it is the interaction of subsystems and their maintenance schedules that produces the benefit of alignment. Multi-component maintenance models are defined to be those in which a system consists of multiple units (equipment or machines) that may depend on each other economically, stochastically or structurally (see [12] and references therein). Economic dependence is focussed on the direct cost of the maintenance activity, and whether or not carry out maintenance on multiple units simultaneously can decrease costs, e.g. via economies of scale, or increase them, e.g. via the need to employ extra resources. Stochastic dependence concerns the failure probabilities and whether or not these are related across units. Structural dependence applies, for example, when the part needing maintenance is inside or connected to other components, so in order to have maintenance on one component, others may also need maintenance, or at the least dismantling. Clearly these latter two types are not relevant to the HVCCC situation: here the failure probabilities are already accounted for in the preventive maintenance policies, which are set, and any structural connection between maintenance tasks simply implies a constraint, indicating that some maintenance tasks must be scheduled together. Neither of these addresses the primary mechanism of interest: that the alignment of maintenance tasks (by re-timing) can release capacity in the system. Clearly economic benefit is realized by releasing capacity, thus we would hope that work categorized under economic dependence would address the HVCCC context. However that does not seem to be the case, and all work highlighting the economic benefits of maintenance scheduling in multi-component systems focusses on the direct costs of the maintenance activities themselves, not on the consequent production benefits. We thus believe that a third type of economic dependence warrants attention, and that is when scheduling maintenance tasks either together or apart affects the production capacity, and hence realizable revenue, of the system.

The only work we are aware of which articulates the benefits of alignment as identified in our context is that of [7] on preventive scheduling of railway track maintenance. As in our context, maintenance is planned over a finite time horizon. They consider maintenance of two types: routine work and projects. The former are cyclic, with given period, and so are not similar to the situation we consider. The projects are similar: input data is a list of maintenance projects, and for each, its duration, together with its earliest and latest start time. However these are relatively infrequent (for each type, performed once every 6 months to a year), and a major part of the contribution of their paper is to explore the interaction between projects and routine work, which is an interesting challenge. Their objective is to minimize the combination of track possession costs, which reflect the train schedule disruption, and the direct costs of the maintenance activities. In our setting, the latter are irrelevant, but the former could be viewed as a proxy for track capacity. They recognize that by clustering jobs at the same time, the track possession cost may be minimized: this corresponds to one type of alignment in our context. Since their model focusses on a single link in the network, the other type is not noticed. They develop an integer programming model, and for the scale of problem they consider (15 types of maintenance over a 3-4 year horizon), the solution times are too long, indeed for their randomly generated problems over 2-year horizons, less than 30% of instances are solved within three hours. Thus they consider a restricted integer program, and four heuristics, to get good quality solutions in reasonable time.

In this paper, we do not have cyclic maintenance tasks, but we do consider simultaneous scheduling over the whole system, not a single link. Most significantly, the cost of the schedule is not a simple linear function such as the track possession cost. Instead, because of the system-wide effects, and the interdependence of subsystems in achieving system capacity, we require solution of an optimization model to determine the impact of a combination of maintenance tasks at scheduled times. Also important is the treatment of time. Budai *et al.* [7] are able to use a fairly coarse time discretization (weeks), needing only the order of 100 periods in their model. In our context, maintenance tasks have start time specified to the nearest 15 minutes in some cases, (to the nearest hour in others), and are of the order of hours to days in duration. Thus we need to address the challenge of how to handle time without leading to an extremely large number of variables.

A simplified version of the problem considered in the present work has been introduced in a more abstract setting in [3]. That paper omits some of the complicating constraints specific to the coal chain maintenance scheduling to arrive at a problem that might be applied in different network related contexts. Even more simplified special cases (unit processing time jobs with arbitrary start times) are studied from a computational complexity viewpoint in [2]. In contrast, the present work focuses on modelling the actual problem of the planners at HVCCC as closely as possible, and proposes solutions of immediate practical relevance.

To conclude, we believe our paper makes a quite unique contribution to the maintenance optimization literature. It considers a multi-component system, where maintenance activities on the components can be scheduled so as to produce economic benefit of a new type, not previously considered. It exploits a relationship between production and maintenance in what appears to be a new way. There does not seem to be any prior work that considers scheduling maintenance activities so as to maximize the production capacity of the system, unless one interprets minimizing track possession costs in rail track maintenance in that light, and in that case we see our paper makes further significant new contributions. In particular, maintenance is scheduled system-wide and the objective is a complex function of the interaction of the system as a whole, not a simple linear function. Furthermore, we provide insights and ideas for handling scheduling problems of this type when maintenance tasks are of widely varying durations, and their start times are fine-grained relative to the planning time horizon.

We expect the models and methods presented here to have broader applicability beyond the HVCC: they could be applied in any setting in which production is reasonably modelled as a flow in a network, throughput (total production rate) is the key objective to be maximized, and regular maintenance needs to be scheduled on network components. Most mining supply chains, whether for coal, iron ore, or other minerals, well fit this description. Applications to other bulk goods supply chains, such as for fertilizer or wheat, are also likely to be possible.

3 Problem description

In this section we set the scene. The first subsection contains details on the underlying network while the second subsection introduces the actual scheduling problem.

3.1 The network model

The network representing the HVCC consists of subnetworks for railway track and for terminals.

1. The railway track network has nodes for load points and for junctions, and the arcs represent rail track sections.
2. There are two terminal networks whose arcs correspond to terminal equipment.

The full network is shown in Figure 1. The ellipses indicate the terminals and the part

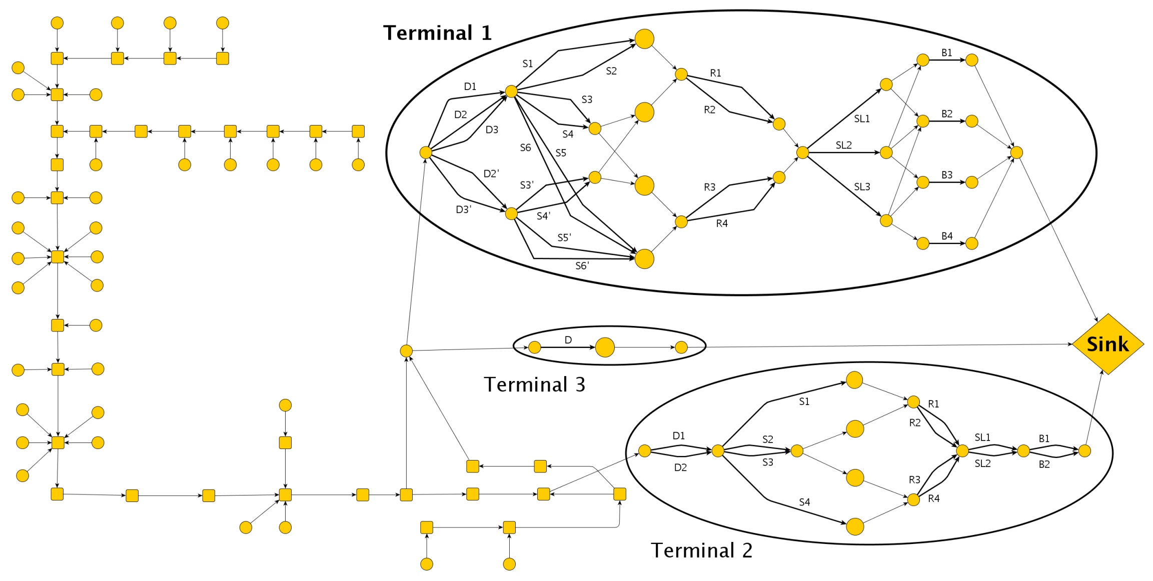

Figure 1: The HVCC network. The terminal subnetworks are indicated by ellipses. For the rail network, rectangular nodes represent junctions while load points correspond to circle nodes.

outside of them is the rail network, where square nodes represent junctions while circle nodes are load points. The load point nodes are included in Figure 1 just for illustration. In the actual model they are identified to form a single source node, and for every load point there is a corresponding arc linking the source to the respective junction. The capacities of these load point arcs are the demand forecasts, which may vary over the time horizon. Arcs between junctions have capacities determined by the number of trains that can pass the corresponding piece of track per day.

For the terminal modelling we focus on Terminals 1 and 2, as Terminal 3 was commissioned only recently, and we did not implement a more detailed model yet. Both of the first two terminals have a column of four larger nodes in the middle, representing the pads on the stockyard where coal can be stored. In fact, real world operation can be reflected quite accurately by requiring that the coal has to stay on the pad for a certain time. This dwelling time is restricted to be between two parameters D_1 and D_2 which might be taken to be three and ten days to capture what typically happens in practice. The inbound and the outbound part of the terminals are to the left and to the right of the pad nodes, respectively. Labeled arcs represent terminal machines where the labels “D”, “S”, “R”, “SL” and “B” stand for dump station, stacker, reclaimer, ship loader and berth, respectively. Note that in the Terminal 1 network there are arcs labeled D2 and D2′, both corresponding to the same dump station, and similarly for dump station 3 and stackers 3 to 6. This is necessary to capture the following aspect of the operational practice at the terminal. The six stackers are grouped into four *stacker streams*: $\{1, 2\}$, $\{3\}$, $\{4\}$ and $\{5, 6\}$. A stream is available if at least one of its stackers is available. The inbound capacity is determined by the following rules, where $c_1 = 85$, $c_2 = 17.5$ and $c_3 = 15$ are capacity parameters.

1. Every dump station can be combined with every stacker stream.
2. The basic capacity of a (dump station, stacker stream)-pair is c_1.
3. Every combination of dump station 2 or 3 with a stacker stream different from $\{1,2\}$ (*high-throughput pair*) releases an additional capacity of c_2.
4. If the number of available dump stations is greater than or equal to the number of available stacker streams, and exactly one of the stackers 1 and 2 is available, there is a reduction of c_3 due to inefficiency.

For the given values of the capacity parameters, the inbound capacity of Terminal 1 can be characterized more formally as follows. For $i \in \{1,2,3\}$ and $j \in \{1,2,\dots,6\}$, let a_i and b_j indicate the availability of dump station i and stacker j, respectively, i.e.

$$a_i = \begin{cases} 1 & \text{if dump station } i \text{ is available,} \\ 0 & \text{otherwise,} \end{cases}$$

$$b_j = \begin{cases} 1 & \text{if stacker } j \text{ is available,} \\ 0 & \text{otherwise.} \end{cases}$$

Then the inbound capacity equals the optimal value of the integer program (1)–(10).

$$\begin{aligned}
\max \quad & c_1y_1 + c_2y_2 - c_3y_3 && (1)\\
\text{s.t.} \quad & x_{12} \leqslant b_1 + b_2, && (2)\\
& x_{56} \leqslant b_5 + b_6, && (3)\\
& y_1 \leqslant a_1 + a_2 + a_3, && (4)\\
& y_1 \leqslant x_{12} + b_3 + b_4 + x_{56}, && (5)\\
& y_2 \leqslant a_2 + a_3, && (6)\\
& y_2 \leqslant b_3 + b_4 + x_{56}, && (7)\\
& 3y_3 \geqslant (a_1 + a_2 + a_3) - y_1 + 2x_{12} - (b_1 + b_2), && (8)\\
& x_{12}, x_{56}, y_3 \in \{0,1\}, && (9)\\
& y_1 \in \{0,1,2,3\},\ y_2 \in \{0,1,2\}. && (10)
\end{aligned}$$

By (2) and (3), x_{12} and x_{56} are the indicator variables for the availability of the streams $\{1,2\}$ and $\{5,6\}$, respectively. Constraints (4) and (5) make y_1 a bound for the number of available (dump station, stacker)-pairs, and similarly, y_2 bounds the number of available high-throughput pairs by (6) and (7). Finally, (8) ensures that $y_3 = 1$ if and only if the inefficiency condition holds, because in this case

$$2 \geqslant (a_1 + a_2 + a_3) - y_1 \geqslant 0 \quad \text{and} \quad 2x_{12} - (b_1 + b_2) = 1.$$

In order for this characterization to be valid (i.e. the equivalence between the four inbound capacity rules and the integer program (1)–(10)), certain assumptions on the parameters

c_1, c_2 and c_3 are necessary. For instance, if c_3 were larger than c_1, the availability of stacker 1 does not enforce $x_{12} = 1$: In the situation

$$a_1 = a_2 = a_3 = b_1 = b_3 = b_4 = 1,\ b_2 = b_5 = b_6 = 0,$$

the optimal solution of (1)–(10) is

$$x_{12} = x_{56} = y_3 = 0, \qquad y_1 = y_2 = 2$$

with objective value $2(c_1 + c_2)$, while the capacity determined according to the rules is $3c_1 + 2c_2 - c_3$: There are three (dump station, stacker stream)-pairs available, two of them high-throughput, but one of them inefficient.

We conclude, that the operational terminal logic is captured by the following capacities on the inbound arcs in the network for Terminal 1:

- capacity 85 for arcs D1, D2, D3, S3, S4, S5, S6,
- capacity 70 for arcs S1 and S2, and
- capacity 17.5 for arcs D2′, D3′, S3′, S4′, S5′, S6′.

Using these types of considerations, where the one explained in detail is by far the most involved, the full terminal network structure including capacities can be derived.

3.2 Maintenance scheduling

The initial schedules from the track owners and the terminal operators are given as a list of maintenance jobs. Every entry of this list consists of the name of the asset to be maintained, the start and the end time of the maintenance activity, and possibly an additional entry indicating the type of work to be done. At the terminals an outage simply makes the corresponding arcs unavailable for the job's duration. In most cases that means the deletion of a single arc, but in some cases for Terminal 1 two arcs may be affected as described in the previous subsection. For the rail network a single asset can be associated with a sequence of arcs in the network, and the effect of the outage on the capacity can be specified for each of the affected arcs separately. For instance, if there is double track available an outage might reduce the capacity to 50%, while on a single track it is reduced to zero. There are also certain track inspection jobs that do not block the track completely for their whole duration, but still cause delays. The effect of these jobs is taken into account by small capacity reductions: the exact value of this reduction is given as input data based on the practical experience of maintenance planners at HVCCC and lies typically between 10% and 20%.

For a fixed schedule, we can collect the start and end times of the jobs and order the list of all these times. This defines a partition of the time horizon into intervals of constant maintenance activity. We call this partition the *time slicing* associated with the schedule, and the intervals *time slices*. In order to measure the quality of the schedule we construct a time expanded network containing one copy of the basic network per time slice. Flows in this network represent total tonnes of coal transported during the time slice. The network copies for consecutive time slices are connected via arcs linking the corresponding copies of pad nodes. Flows on these linking arcs represent coal present on the pads at the transition time between the time slices. The arc capacities in a time slice are taken to be the capacities of the basic network – which express upper bounds on the rate of flow in each arc – discounted by the capacity reduction factor for any maintenance job occurring on the arc during the time slice, and multiplied by the length of the time slice. The results are upper bounds on the amount of coal that can move along each arc during the time slice. We solve a max flow problem in this expanded network, the value of which is interpreted as a measure for the total system capacity. We take this as our primary optimization objective. In fact, it is more complicated than a pure max flow problem, as there are side constraints from the requirement that the coal stays on the ground for some time.

In discussions with the maintenance planners, it emerged that they would be prepared to move the jobs, usually for intervals of plus or minus 7 days. We initially expected there would be some inter-maintenance constraints, for example, that a type of job carried out at 4-weekly intervals could not be carried out more than 5 weeks apart. But the maintenance planners were not concerned about this issue, and preferred the simple assumption that jobs could not deviate more than some fixed number of days around their initial scheduled time. The arising optimization problem is to take an input schedule and modify it according to certain rules such that the total system capacity is maximized. The scheduling rules can be summarized as follows.

1. No job can be moved by more than 7 days.
2. Major track outages (rail outages with a duration of more than 24 hours) must not be moved.
3. The track inspection jobs and jobs with certain specific work type tags must not be moved.
4. Rail jobs initially scheduled on a weekday (Monday to Friday) have to stay on a weekday.
5. Rail jobs initially scheduled between 7:00am and 4:30pm have to stay in this time window.
6. Jobs on the same item that do not overlap in the initial schedule are not allowed to overlap in the optimized schedule.
7. Some jobs on stackers and reclaimers have an associated so-called washdown job which immediately precedes them. These job pairs can only be moved together.

Our primary objective is to maximize the throughput, but from a practical point of view it is also desirable not to deviate too much from the initial input schedule, as this was the result of independent decision processes of the providers. So the final goal should be to treat the maintenance scheduling in a bi-objective framework with the objectives total throughput and (weighted) number of job movements. As a first step in that direction we propose a lexicographic optimization: in a two-phase approach we first maximize the total throughput and then minimize the number of moved jobs subject to a lower bound on the throughput.

4 A mixed integer programming formulation

In this section we present a MIP formulation of the maintenance scheduling problem described in Section 3. Let (N, A, s, s', u) denote the network with node set N, arc set A, source s and sink s', and capacities $u_a \in \mathbb{R}_+$ for $a \in A$. We denote the set of incoming and outgoing arcs of a node $v \in V$ by $\delta^-(v)$ and $\delta^+(v)$, respectively. Recall that the source s replaces the load point nodes in Figure 1. In addition, let $N_P \subseteq N$ denote the set of nodes corresponding to the pads on the terminal stockyards. They are special in that they allow the storage of flow, and each of them has associated upper and lower capacities u_v^{lower} and u_v^{upper}, representing the acceptable variation of the amount of coal on the pad. We represent the considered time horizon by a real interval $[0, T]$ where time is measured in days, so for the complete annual problem, $T = 365$. A maintenance job j is specified by

- its arc set $A_j \subseteq A$ with associated capacity reduction factors $\rho_{ja} \in [0, 1]$ for $a \in A_j$, meaning that during the processing of job j the capacity of arc a is reduced to $(1 - \rho_{ja})u_a$,
- a processing time $p_j \in \mathbb{R}_+$,
- a finite set of possible start times $\mathcal{S}_j \subseteq [0, T]$, and
- an initial start time $S_j^0 \in \mathcal{S}_j$.

Note that the first five scheduling rules listed in Section 3.2 will not appear as constraints in the MIP as they can be enforced by simply restricting the sets $\mathcal{S}_j$ appropriately. Scheduling a job to start at time $S_j \in \mathcal{S}_j$ reduces the capacity of arcs $a \in A_j$ in the time interval $[S_j, S_j + p_j]$. We have to schedule a set J of maintenance jobs in such a way that the total throughput over the interval $[0, T]$ is maximized. We denote the set of job pairs that are not allowed to overlap due to rule 5 by $R \subseteq \binom{J}{2}$, and we call a maximal clique $\mathcal{C}$ in the graph with vertex set J and edge set R a *conflict clique*. Then the scheduling rule just says that for any conflict clique $\mathcal{C}$, at any point of time at most one of the jobs in $\mathcal{C}$ can be processed.

In order to formulate the MIP, we need some more notation. Let $\mathcal{T} = \{0 = t_0 < t_1 < \cdots < t_M = T\}$ be the set of all times relevant for the problem, i.e.

$$\mathcal{T} = \big([0, T] \cap \mathbb{Z}\big) \cup \bigcup_{j \in J} \bigcup_{a \in A_j} \mathcal{S}_j \cup (\mathcal{S}_j + p_j)\,.$$

Note that this implicitly defines M, the number of possible time slices that could occur in the resulting maintenance schedule. We require $\mathcal{T}$ to contain all integers in the time horizon in order to control the daily balances of in- and outflow at the stockyard. As for a fixed schedule, this is a time slicing. In fact, it is a refinement of the time slicing associated with the initial schedule (assuming that the initial start time of job j is contained in $\mathcal{S}_j$ for every job j). Now we can define the variables of the model.

- For $a \in A$ and $i \in [M]$, $x_{ai} \in \mathbb{R}_+$ is the flow on arc a in the i-th time slice $[t_{i-1}, t_i]$.
- For $v \in N_P$ and $i \in [0, M]$, $x_{vi} \in \mathbb{R}_+$ is the pad level at time t_i.
- For $v \in N_P$ and $d, d' \in [T]$ with $d' - d \in \{D_1, D_1 + 1, \dots, D_2\} \pmod T$, $X_v^{dd'} \in \mathbb{R}_+$ is the amount of flow entering node v on day d and leaving on day d'.
- For $j \in J$ and $t \in \mathcal{S}_j$, y_{jt} is the indicator variable for job j starting at time t, i.e.

$$y_{jt} = \begin{cases} 1 & \text{job } j \text{ starts at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

- For $a \in A$, $i \in [M]$ and impact factor γ, the variable $w^i_{a\gamma} \in \{0,1\}$ indicates if in time slice i arc a is affected by a job with impact γ, i.e.

$$w^i_{a\gamma} \in \{0,1\} = \begin{cases} 0 & \text{if arc } a \text{ between } t_{i-1} \text{ and } t_i \text{ is affected} \\ & \text{by a job with reduction factor } \gamma, \\ 1 & \text{otherwise.} \end{cases}$$

In order to formulate the constraints it is convenient to denote the set of relevant jobs for an arc $a \in A$ by J_a , i.e. $J_a = \{j \in J : a \in A_j\}$, and the set of possible capacity discount (impact) factors by $\Pi_a = \{\rho_{ja} : j \in J_a\}$. We can now write down a MIP for the maintenance scheduling problem. At first, our objective is to maximize the total flow

$$\max\ z = \sum_{i=1}^{M} \sum_{a \in \delta^+(s)} x_{ai} \tag{11}$$

subject to the following constraints.

Flow conservation constraints. Except at source and sink and at the pad nodes, where flow between time slices is possible, we have flow conservation per time slice. For every node $v \in N \setminus (N_P \cup \{s, s'\})$ and every $i \in [M]$, we have

$$\sum_{a \in \delta^-(v)} x_{ai} - \sum_{a \in \delta^+(v)} x_{ai} = 0, \tag{12}$$

and for $v \in N_P$, $i \in [M]$

$$\left(\sum_{a \in \delta^-(v)} x_{ai} - \sum_{a \in \delta^+(v)} x_{ai} \right) = x_{vi} - x_{v,i-1}. \tag{13}$$

We also include periodic boundary conditions

$$x_{vM} = x_{v0} \qquad\qquad (v \in N_P) \tag{14}$$

in this group of constraints.

Capacity constraints. There are arc capacities

$$\begin{aligned} x_{ai} &\leqslant u_a(t_i - t_{i-1})(1 - \gamma(1 - w^i_{a\gamma})) \\ &= u_a(t_i - t_{i-1})(1-\gamma) + u_a(t_i - t_{i-1})\gamma w^i_{a\gamma} \end{aligned} \tag{15}$$

for all $a \in A$, $i \in [M]$ and impact factors $\gamma \in \Pi_a$. Note that this constraint and the use of the w variables exposes the fixed charge network flow structure in the problem. The job start indicator variables are linked to the impact indicators via constraints

$$w^i_{a\gamma} \leqslant 1 - \sum_{t \in \mathcal{S}_j \,:\, t_i - p_j \leqslant t \leqslant t_{i-1}} y_{jt} \tag{16}$$

for all arcs $a \in A$, all impact factors $\gamma \in \Pi_a$, all time slice indices $i \in [M]$ and all jobs $j \in J$ with $\rho_{ja} = \gamma$. In addition, we have node capacities

$$u_v^{\text{lower}} \leqslant x_{vi} \leqslant u_v^{\text{upper}} \tag{17}$$

for all $v \in N_P$ and $i \in [M]$. Note that these are the only lower bounds on flow in the model, so a flow of zero on all arcs, and a flow which is between these bounds and identical for all variables linking a storage node across time slices, provides a feasible solution.

Scheduling constraints. Every job has to be scheduled exactly once, and the processing periods of incompatible jobs must not overlap.

$$\sum_{t \in \mathcal{S}_j} y_{jt} = 1 \qquad (j \in J), \tag{18}$$

$$\sum_{j \in \mathcal{C}} \sum_{\substack{t \in \mathcal{S}_j \\ t < t_i \leqslant t + p_j}} y_{jt} \leqslant 1 \qquad (i \in [M],\ \mathcal{C} \text{ conflict clique}), \tag{19}$$

Dwell time constraints. The values of the flow variables x_{ai} determine the total daily in- and outflows at the pad nodes. Now the inflow of day d has to leave between day $d + D_1$ and day $d + D_2$. This is enforced by the nonnegativity of the variables $X_v^{dd'}$ and the constraints

$$\sum_{d'=d+D_1}^{d+D_2} X_v^{dd'} = \sum_{i \,:\, \lceil t_i \rceil = d} \sum_{a \in \delta^-(v)} x_{ai}, \tag{20}$$

$$\sum_{d'=d-D_2}^{d-D_1} X_v^{d'd} = \sum_{i \,:\, \lceil t_i \rceil = d} \sum_{a \in \delta^+(v)} x_{ai} \tag{21}$$

for $v \in N_P$ and $d \in \{1, 2, \ldots, T\}$.

Variable domains. The flow variables are nonnegative reals and the job start indicators are binary.

$$x_{ai},\, x_{vi},\, X_v^{dd'} \geqslant 0 \qquad (a \in A,\, v \in N_P,\, i \in [M],\; d, d' \in [T]), \tag{22}$$

$$y_{jt} \in \{0,1\} \qquad (j \in J,\, t \in \mathcal{S}_j), \tag{23}$$

$$w^i_{a\gamma} \in \{0,1\} \qquad (a \in A,\, i \in [M],\, \gamma \in \Pi_a). \tag{24}$$

In a second phase we change the objective function to maximize the number of jobs starting at their initially scheduled start time S_j^0:

$$\max \sum_{j \in J} y_{jS_j^0}, \tag{25}$$

and we add a lower bound for the total throughput, i.e. a constraint of the form

$$\sum_{i=1}^{M} \sum_{a \in \delta^+(s)} x_{ai} \geqslant B, \tag{26}$$

where the bound B is a function of the best objective value obtained in the first phase. For our computational experiments we just multiplied the maximal throughput by 0.999.

5 Solution strategies

In Section 4 we formulated a large scale MIP for the maintenance scheduling of the HVCC. In this section we present some strategies for coping with this large problem. We focus on Phase 1 of the lexicographic optimization, i.e. on the maximization of the total throughput, as this is the primary objective in practice. For the annual planning more than 2,000 jobs have to be scheduled. After some preprocessing taking into account the rules described in Section 3.2 (fixed jobs, washdowns, etc.) there are still about 1,000 jobs contributing binary variables y_{jt}. In practice, jobs are scheduled by the half-hour or on even finer time scales. If this is accurately modelled, allowing every half-hour in the time window as a potential start time, a job without additional daylight or weekday constraints has about $14 \cdot 48 = 672$ potential start times, corresponding to binary variables. In Subsection 5.1 we describe a heuristic method to choose a representative subset of the start times. Even with these reduced candidate start time sets initial computational test reveal that the problem for the complete annual time horizon cannot be solved by simple application of a commercial MIP solver (in our case CPLEX 12.3). On the other hand, we obtain promising results if the problem is restricted to a shorter time horizon. This motivates a rolling horizon approach to the full problem as is described in Subsection 5.2.

5.1 Reducing the number of potential start times

Intuitively, focussing on two jobs j and j', it seems one would always try to minimize their overlap or to maximize their overlap. Consider for example the three arcs in Figure 2. If j

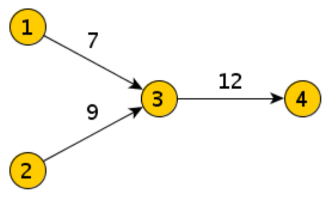

Figure 2: Three arcs from a network. Capacities are indicated by arc labels.

and j' operate on arcs $(1,3)$ and $(3,4)$, respectively, they should be scheduled to overlap as much as possible, while jobs on arcs $(1,3)$ and $(2,3)$ should be separated. To give a more specific example, suppose the processing times are $p_j = 2$ and $p_{j'} = 3$ and the candidate start time sets are $\mathcal{S}_j = \{1,2\}$ and $\mathcal{S}_{j'} = \{2,3\}$.

1. If the arcs for j and j' are $(1,3)$ and $(3,4)$ it is a good idea (at least locally) to schedule both jobs to start at time $S_j = S_{j'} = 2$, giving a total capacity of $2 \cdot 12 + 3 \cdot 0 + 12 = 36$ over the time interval $[0,6]$.

2. If the arcs for j and j' are $(1,3)$ and $(2,3)$ the local analysis suggests to schedule the jobs to start at times $S_j = 1$ and $S_{j'} = 3$ with a total capacity of $1 \cdot 12 + 2 \cdot 9 + 3 \cdot 7 = 51$ over the time interval $[0,6]$.

Of course the situation becomes more complicated with more jobs involved. But still, the intuition is that there should be an optimal schedule with the property that every job j starts at its earliest or at its latest possible start time, or its start or completion time coincides with the start or completion time of some other job j'. For networks without storage at nodes, this intuition can be converted into a rigorous argument, which is the subject of ongoing research. For the present work we adopt a more pragmatic viewpoint and generate candidate start time sets $\mathcal{S}_j$ by a heuristic based on the described intuition. Let $J_0 \subseteq J$ be the set of jobs that are not fixed by a scheduling rule. For $j \in J_0$, we initialize $\mathcal{S}_j$ with the initial start time S_j^0 and the earliest and latest possible start times, i.e. $S_j^0 \pm 7$. For $j \in J \setminus J_0$, clearly $\mathcal{S}_j = \{S_j^0\}$. Then we iteratively add candidate start times to the sets $\mathcal{S}_j$ that could potentially align job j with the start or end of some job j'. In order to keep the sets $\mathcal{S}_j$ reasonably small we ensure that every job gets at most 2 candidate start times per day. The details of this procedure are given in Algorithm 1.

5.2 A rolling horizon approach

Computational tests revealed that even after the reduction of the number of binary variables the complete annual problem is very hard. As the performance on restrictions of the problem to shorter time horizons is better, the iterative solution of smaller subproblems is a natural approach to solving the problem, which is also supported by the intuition that rescheduling of a job should have mainly local effects: the system's performance in September should be largely independent of rescheduling of jobs in March. This suggests the following approach. Fix the binary start indicator variables for all jobs outside a time window $[t,t']$ and fix all

Algorithm 1 Generating start time sets.

```
**for** $j \in J_0$ **do** $\mathcal{S}_j \leftarrow \left\{S_j^0, S_j^0 - 7, S_j^0 + 7\right\}$
$\mathcal{S} \leftarrow \bigcup_{j \in J} \mathcal{S}_j \times \{j\}$
**while** not STOP **do**
    **for** $j \in J_0$ **do**
        **for** $(t, j') \in \mathcal{S}$ with $j' \neq j$ **do**
            **for** $t' \in \{t, t + p_{j'}, t - p_j, t + p_{j'} - p_j\}$ **do**
                **if** $t'$ is a feasible start time for job $j$ and $\left|\mathcal{S}_j \cap \left[\lfloor t' \rfloor, \lfloor t' + 1 \rfloor\right]\right| < 2$ **then**
                    Add $t'$ to $\mathcal{S}_j$ and $(t', j)$ to $\mathcal{S}$
    **if** none of the sets $\mathcal{S}_j$ changed **then** STOP
```

flow variables outside a slightly larger time window $[t - \delta, t + \delta]$, solve the resulting MIP, shift the time windows by some value σ,and iterate. The whole procedure is described more precisely in Algorithm 2.

Algorithm 2 The rolling horizon algorithm.

```
**Parameters:** $w$ – inner time window width
                $\delta$ – margin between inner and outer time window
                $\sigma$ – time window shift
Initialize the MIP (11)–(24)
**for** $j \in J$ **do**
    **for** $t \in \mathcal{S}_j$ **do**
        **if** $t = S_j^0$ **then** $y_{jt} \leftarrow 1$ **else** $y_{jt} \leftarrow 0$
Generate an initial solution from the current values of the variables $y_{jt}$
**while** not STOP
    $t \leftarrow 0$;    $t' \leftarrow w$
    **while** $t' < T$ **do**
        fix all binary variables for jobs $j$ outside $[t, t']$
        fix all continuous variables for time slices outside $[t - \delta, t' + \delta]$
        solve the problem
        update the start times of jobs $j$ that are not fixed
        unfix all fixed variables
        $t \leftarrow t + \sigma$;    $t' \leftarrow t' + \sigma$
```

6 Computational Results

In this section we present computational results for two real world data sets. We use the raw schedules for the years 2010 and 2011 as inputs. The capacities for the load point arcs are determined from the annual capacity forecast numbers for these years. They specify for every load point on a quarterly level the expected daily demands. For 2010 this amounts to a total daily demand of 375kt and for 2011 it is 475kt in total. After preprocessing, the 2010 schedule contains 1,277 jobs (197 of them fixed), and for 2011 there are 986 jobs (159 of them fixed). Figure 3 shows the distribution of the processing times.

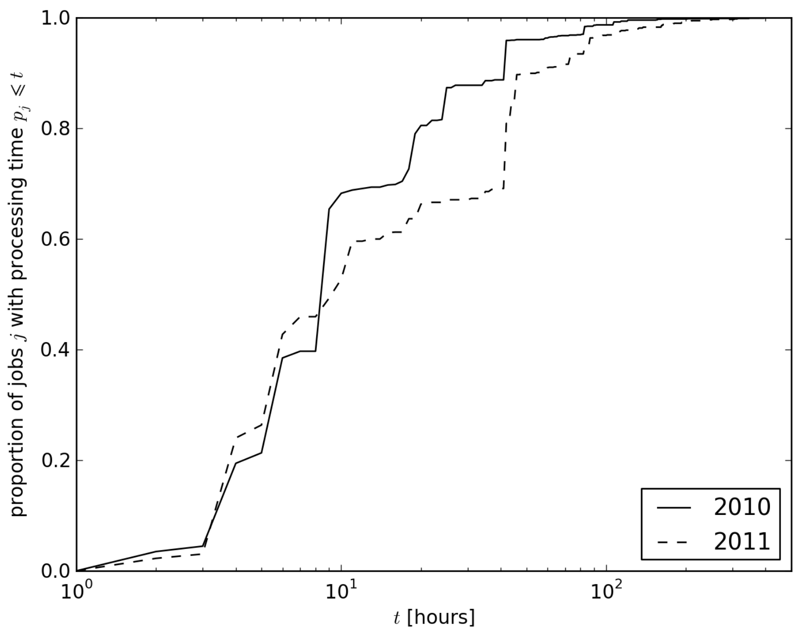

Figure 3: Distributions of the job processing times.

All our computations are done on a Dell PowerEdge R710 with dual hex core 3.06GHz Intel Xeon X5675 Processors and 96GB RAM running Red Hat Enterprise Linux 6. CPLEX v12.3 is used in deterministic mode with a single thread.

From the 2010 and 2011 data we obtain eight quarterly and two annual instances. For each of these instances we compare the performance of CPLEX with default settings on the complete problem to the rolling horizon heuristic in Algorithm 2 with parameters $w = 15$, $\delta = 10$ and $\sigma = 10$ (all times in days). For the quarterly problems, we impose a time limit of six hours for each phase. In the rolling horizon algorithm the time limit of six hours is the stopping criterion for the while loop in Algorithm 2, and for each CPLEX call we use a time limit of 25 minutes. For the annual problems we increase the time limit per phase to 24 hours, and the time limit per CPLEX call to 40 minutes. The results are presented in Tables 1 and 2. Table 1 shows the relative improvements in throughput that are obtained by the two methods as well as an upper bound for the throughput gain. More precisely, if the total througput for the initial schedule is z^{init}, the best schedules for the two methods give throughput values z^{full} and z^{rolling}, respectively, and the best upper bound from CPLEX on

	Full problem	Rolling horizon	Upper bound
2010 Q1	2.21%	3.01%	4.14%
2010 Q2	1.90%	2.77%	3.28%
2010 Q3	1.70%	2.73%	3.10%
2010 Q4	1.47%	1.82%	2.27%
2010 full	1.61%	2.61%	3.40%
2011 Q1	2.24%	2.53%	2.60%
2011 Q2	1.58%	2.73%	2.96%
2011 Q3	2.39%	3.92%	4.27%
2011 Q4	1.49%	2.13%	2.90%
2011 full	0.00%	1.77%	2.76%

Table 1: Throughput improvements in Phase 1. All numbers are relative improvements compared to the initial schedule that is used as a start solution. The last column contains upper bounds for the throughput gains.

the whole quarterly problem is z^{bound}, then the reported numbers are

$$\frac{z^{\text{full}} - z^{\text{init}}}{z^{\text{init}}}, \qquad \frac{z^{\text{rolling}} - z^{\text{init}}}{z^{\text{init}}}, \qquad \frac{z^{\text{bound}} - z^{\text{init}}}{z^{\text{init}}}.$$

Note that the total throughput is about 120 megatonnes for 2010 and 140 megatonnes for 2011, so an improvement of 1% corresponds to an additional throughput of 1.2 million tonnes and 1.4 million tonnes, respectively. So the improvement obtained with our model makes a significant difference in practice. Table 2 contains results on the numbers of moved jobs. We make the following observations. For all ten instances the rolling horizon approach yields a significantly larger total throughput than CPLEX on the complete problem. For the full problem, the number of moved jobs can be reduced significantly in Phase 2, while for the rolling horizon method in nine of the ten instances we do not find any alternative schedule with a smaller number of moved jobs that yields at least 99.9% of the throughput achieved in Phase 1. In Phase 2 the difference between "full problem" and "rolling horizon" is that the "full problem" has a weaker lower bound on the throughput from the result of Phase 1. A comparison of the lower bound columns "LB" in Table 2 indicates that there might be a tradeoff between throughput and the number of moved jobs which should be studied in more detail in future work.

		Full problem			Rolling horizon		
	Total	P1	P2	LB	P1	P2	LB
2010 Q1	342	284	20	14	289	39	26
2010 Q2	384	324	29	17	319	319	41
2010 Q3	293	248	60	16	242	242	40
2010 Q4	266	215	43	15	200	200	26
2010 full	1,277	1,049	1,049	46	1,040	1,040	108
2011 Q1	247	189	18	12	184	184	16
2011 Q2	257	198	11	10	207	207	32
2011 Q3	268	219	16	13	218	218	45
2011 Q4	222	175	17	11	257	257	23
2011 full	986	0	0	0	704	704	40

Table 2: Numbers of moved jobs. We report the total number of jobs (column "Total"), the numbers of moved jobs for the final schedule after Phase 1 and Phase 2 (columns "P1" and "P2") and the lower bound for the number of moved jobs to achieve the throughput that is enforced in Phase 2 (column "LB").

7 Concluding remarks

In this paper we present a MIP model for the maintenance scheduling at the HVCC, where the primary objective is to maximize the throughput, and the secondary objective is to minimize deviations from a given initial schedule. The resulting large scale problems cannot be solved directly, so efficient solution strategies are needed. We propose an iterative approach based on solving subproblems obtained by fixing most of the binary variables. Our computational tests show that this rolling horizon approach outperforms plain CPLEX in terms of the obtained total throughput. Also the experimental results indicate a tradeoff between throughput and the number of moved jobs.

Clearly, more work is necessary in order to improve the performance of the model. Our experiments indicate that the initial LP bounds are rather weak, so adding appropriate cutting planes is a promising approach. Another option for reducing the complexity of the problem is to put an explicit bound on the number of moved jobs. This seems to be a reasonable direction, especially as one outcome of discussions with the maintenance planners was that the option to add more specific constraints on the set of allowed job movements is a desirable feature. Another step towards a tool that is applicable in practice is the development of a true bi-objective framework to find (or at least approximate) the set of efficient solutions for the objectives "total throughput" and "number of schedule modifications".

Acknowledgement We like to acknowledge the valuable contributions of Jonathon Vandervoort, Rob Oyston, Tracey Giles, and the Annual Capacity Alignment Team from the Hunter Valley Coal Chain Coordinator (HVCCC) P/L. Without their patience, support, and feedback, this research could not have occurred. We also thank the HVCCC and the Australian Research Council for their joint funding under the ARC Linkage Grant no. LP0990739.

References

[1] C. Barnhart, N.L. Boland, L.W. Clarke, E.L. Johnson, G.L. Nemhauser and R.G. Shenoi. "Flight string models for aircraft fleeting and routing". In: *Transportation Science* 32.3 (1998), pp. 208–220. DOI: `10.1287/trsc.32.3.208`.

[2] N. Boland, T. Kalinowski, R. Kapoor and S. Kaur. "Scheduling unit processing time arc shutdown jobs to maximize network flow over time: complexity results". In: *Networks* 63.2 (2014), pp. 196–202. DOI: `10.1002/net.21536`.

[3] N. Boland, T. Kalinowski, H. Waterer and L. Zheng. "Scheduling arc maintenance jobs in a network to maximize total flow over time". In: *Discr. Appl. Math.* 163 (2014), pp. 34–52. DOI: `10.1016/j.dam.2012.05.027`.

[4] N. Boland and M. Savelsbergh. "Optimizing the Hunter Valley coal chain". In: *Supply Chain Disruptions: Theory and Practice of Managing Risk.* Ed. by H. Gurnani, A. Mehrotra and S. Ray. Springer-Verlag London Ltd., 2011, pp. 275–302.

[5] G. Budai and R. Dekker. "An overview of techniques used in planning railway infrastucture maintenance". In: *Proceedings of IFRIMmmm (maintenance modelling and management) Conference.* Ed. by W. Geraerds and D. Sherwin. 2002, pp. 1–8.

[6] G. Budai, R. Dekker and R.P. Nicolai. "Maintenance and production: a review of planning models". In: *Complex Systems Maintenance Handbook, Part D.* Ed. by K.A.H. Kobbacy and D.N.P. Murthy. Series in Reliability Engineering. Springer, 2008. Chap. 13, pp. 321–344. DOI: `10.1007/978-1-84800-011-7`.

[7] G. Budai, D. Huisman and R. Dekker. "Scheduling preventive railway maintenance activities". In: *Journal of the Operational Research Society* 53 (2006), pp. 1035–1044. DOI: `10.1057/palgrave.jors.2602085`.

[8] D. Frost and R. Dechter. "Optimizing with Constraints: A Case Study in Scheduling Maintenance of Electric Power Units". In: *Proc. 5th International Symposium on Artificial Intelligence and Mathematics.* 1998, pp. 1–20.

[9] D. Frost and R. Dechter. "Optimizing with constraints: a case study in scheduling maintenance of electric power units". In: *Proc. 5th Int. Conf. on Principles and Practice of Constraint Programming – CP 1998.* Ed. by M. Maher and J.-F. Puget. Vol. 1520. LNCS. Springer, 1998, p. 469. DOI: `10.1007/3-540-49481-2`.

[10] A. Haghani and Y. Shafahi. "Bus maintenance systems and maintenance scheduling: model formulations and solutions". In: *Transportation Research Part A: Policy and Practice* 36.5 (2002), pp. 453–482. DOI: `10.1016/S0965-8564(01)00014-3`.

[11] G. Keysan, G.L. Nemhauser and M.W.P. Savelsbergh. “Tactical and operational planning of scheduled maintenance for per-seat, on-demand air transportation”. In: *Transportation Science* 44.3 (2010), pp. 291–306. DOI: `10.1287/trsc.1090.0311`.

[12] R.P. Nicolai and R. Dekker. “Optimal maintenance of multi-component systems: a review”. In: *Complex Systems Maintenance Handbook, Part D*. Ed. by K.A.H. Kobbacy and D.N.P. Murthy. Series in Reliability Engineering. Springer, 2008. Chap. 11, pp. 263–286.

[13] A. Sharma, G.S. Yadava and S.G. Deshmukh. “A literature review and future perspectives on maintenance optimization”. In: *Journal of Quality in Maintenance Engineering* 17.1 (2011), pp. 5–25. DOI: `10.1108/13552511111116222`.

Scheduling unit processing time arc shutdown jobs to maximize network flow over time: complexity results*

Natashia Boland Thomas Kalinowski Reena Kapoor Simranjit Kaur

Abstract

We study the problem of scheduling maintenance on arcs of a capacitated network so as to maximize the total flow from a source node to a sink node over a set of time periods. Maintenance on an arc shuts down the arc for the duration of the period in which its maintenance is scheduled, making its capacity zero for that period. A set of arcs is designated to have maintenance during the planning period, which will require each to be shut down for exactly one time period. In general this problem is known to be NP-hard. Here we identify a number of characteristics that are relevant for the complexity of instance classes. In particular, we discuss instances with restrictions on the set of arcs that have maintenance to be scheduled; series parallel networks; capacities that are balanced, in the sense that the total capacity of arcs entering a (non-terminal) node equals the total capacity of arcs leaving the node; and identical capacities on all arcs.

Introduction

Many real life systems can be viewed as a network with arc capacities, supporting the flow of a commodity. For example, transportation networks, or supply chains, may on occasion be viewed this way. We were motivated by a particular coal export supply chain [4], in which maximizing throughput is a key concern. Whilst this suggests a maximum flow model would be appropriate, in fact, the real network is not static: capacities change over time, and in particular, some arcs are shut down for maintenance at certain times. Often there is some flexibility in the time when maintenance jobs can be scheduled. Every maintenance schedule will incur some loss in the total throughput of the network. To obtain maximum throughput, it is important to select the schedule that leads to minimum loss of flow. For example consider the network in Figure 1 with three nodes $\{s, v, t\}$, three arcs $\{a, b, c\}$ and given arc capacities. The total throughput possible in two time periods when no arcs are on maintenance is 14 units. Suppose that arc a and b have to go on maintenance for a unit period of processing time in a time horizon of two periods. The two possible schedules are either put both the arcs a and b on maintenance together in the first time period giving a total throughput of 7 units in two time periods or put the arc a in first time period and arc b in second time period giving the total throughput of 9 units in the two time periods.

*Networks **63**(2), 196–202, 2014, `doi:10.1002/net.21536`

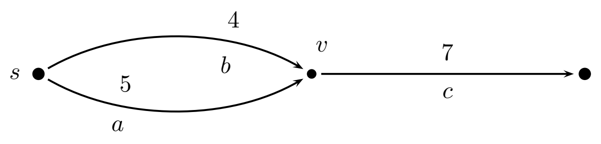

Figure 1: Example network.

Clearly the second schedule is better than the first one as it is giving less loss of flow. This leads to a model in which arc maintenance jobs need to be scheduled so as to maximize the total flow in the network over time [1, 2, 3].

In this paper we consider the case of this problem in which all maintenance jobs have unit processing time. The problem is defined over a network $N = (V, A, s, t, u)$ with node set V, arc set A, source $s \in V$, sink $t \in V$ and nonnegative integral capacity vector $u = (u_a)_{a\in A}$. Note that we permit parallel arcs, i.e. there may exist more than one arc in A having the same start and end node, so A is a multiset. By $\delta^-(v)$ and $\delta^+(v)$ we denote the set of incoming and outgoing arcs of node v, respectively. We consider this network over a set of T time periods indexed by the set $[T] := \{1, 2, \dots, T\}$, and our objective is to maximize the total flow from s to t. In addition, we are given a subset $J \subseteq A$ of arcs that have to be shut down for exactly one time period in the time horizon. In other words, there is a set of maintenance jobs, one for each arc in J, each with unit processing time. Our optimization problem is to choose these outage time periods in such a way that the total flow from s to t is maximized. More formally, this can be written as a mixed binary program as follows:

$$\max\ z = \sum_{i=1}^{T}\left(\sum_{a\in\delta^+(s)} x_{ai} - \sum_{a\in\delta^-(s)} x_{ai}\right) \tag{1}$$

$$\text{s.t.}\quad x_{ai} \leqslant u_a \qquad a \in A\setminus J,\ i\in[T], \tag{2}$$

$$x_{ai} \leqslant u_a y_{ai} \qquad a\in J,\ i\in[T], \tag{3}$$

$$\sum_{i=1}^{T} y_{ai} = T-1 \qquad a\in J, \tag{4}$$

$$\sum_{a\in\delta^-(v)} x_{ai} = \sum_{a\in\delta^+(v)} x_{ai} \qquad v\in N\setminus\{s,t\},\ i\in[T], \tag{5}$$

$$x_{ai} \geqslant 0 \qquad a\in A,\ i\in[T], \tag{6}$$

$$y_{ai}\in\{0,1\} \qquad a\in J,\ i\in[T], \tag{7}$$

where $x_{ai} \geqslant 0$ for $a \in A$ and $i \in [T]$ denotes the flow on arc a in time period i, and $y_{ai} \in \{0, 1\}$ for $a \in J$ and $i \in [T]$ indicates when the arc a is *not* shut down for maintenance in time period i, i.e. $y_{ai} = 0$ in the period i in which the outage for arc a is scheduled.

To the best of our knowledge, Boland et al. [1, 2, 3] initiated study on the problem with general processing times. In [1, 2], the coal supply chain application, which has a number of additional side constraints, is modelled and solved using a rolling time horizon mixed integer programming approach. In [3], the complexity of the general problem is established, and four local search heuristics are developed and compared. We are not aware of any other studies on this problem. Several authors have studied dynamic network flows. For instance [6] studied the problem of finding the maximum flow that can be sent from a source to a sink in T time units, in a network with transit times on the arcs. Variations of the dynamic maximum flow problem with zero transit times are discussed in [5], [8], and [9]. None of these have a scheduling component. Machine scheduling problems have received a great deal of attention in the literature [10], but in the problem we study here, there is no underlying machine, and the association of jobs with network arcs and a maximum flow objective give it quite a different character. The closest work we can find is that in the recent paper of Tawarmalani and Li [11], which considers multiperiod maintenance scheduling over a network, in which the objective is based on multicommodity flows (with origin-destination demands, but without arc capacities), there is a limit on the number of arcs that can be shut down in any one period, and the network's structure is restricted to a tree. Complexity results are provided for linear networks, with a polynomial algorithm in the case of (nearly) uniform commodity demands, and a proof that the case of general demands is strongly NP-hard. Integer programming models are also considered, and polyhedral analysis carried out. The lack of previous attention to the trade-off between maintenance scheduling and network flow reduction in the literature is also noted in [11].

Our key contribution in this paper is an analysis of how the complexity of the problem depends on important characteristics: (i) the case that the set of arcs with a job contains a minimum cut of the network, (ii) *balanced* networks, in which the capacity into and out of each (non-terminal, i.e. transhipment) node is equal, (iii) networks that are series-parallel, (iv) the number of time periods is treated as a fixed parameter, and (v) the case that all arcs have the same capacity. We show for case (i) that it is optimal to schedule all jobs in the same time period, and that this is also true if the network is both balanced and series-parallel. However if the network is balanced but not necessarily series-parallel, then the problem is strongly NP-hard. We provide an approximation ratio for scheduling all jobs in the same time period in the general case, which shows this is asymptotically optimal as T approaches infinity. For case (iv), we show that even if $T = 2$ and the network contains only a single transhipment node, the problem is weakly NP-hard, and we give an algorithm for series-parallel networks that has pseudopolynomial complexity for T fixed (but is exponential in T). In case (v), if all arcs have the same capacity, we prove that the problem can be reduced to a maximum flow problem and T additional linear programs, and hence can be solved in polynomial time. In this case it is not necessarily optimal to schedule all jobs at the same time.

The paper is organized as follows. Section 1 contains a discussion of cases of the network with a single transhipment node. In Section 2 we explore general networks, and in Section 3 we consider the case that all arcs have the same capacity. Finally, in Section 4 we suggest some future directions for study of this problem.

1 Networks with single transhipment node

The problem in general is NP-hard [3]. In this proof, the reduction gave rise to a network with a single transhipment node, which was not balanced, and in which the set of arcs with associated jobs did not contain a minimum cut. This left open the complexity of the cases that all arcs in a minimum cut have an associated outage, or the network is balanced. This section gives a result that describes a class of networks with single transhipment node that covers the above-mentioned cases and is easy to resolve. Consider a network having only one transhipment node, say v. Let

$$C_1^- = \sum_{a\in\delta^-(v)} u_a, \qquad C_1^+ = \sum_{a\in\delta^+(v)} u_a,$$
$$C_2^- = \sum_{a\in\delta^-(v)\setminus J} u_a, \qquad C_2^+ = \sum_{a\in\delta^+(v)\setminus J} u_a.$$

If all jobs are scheduled at the same time, say in time period 1, we obtain a total throughput of

$$\min\left\{C_2^-,\, C_2^+\right\} + (T-1)\min\left\{C_1^-,\, C_1^+\right\}.$$

On the other hand, using $\sum_{a\in\delta^-(v)\cap J} u_a = C_1^- - C_2^-$ and $\sum_{a\in\delta^+(v)\cap J} u_a = C_1^+ - C_2^+$ we obtain an upper bound of

$$\begin{aligned}\min\left\{TC_2^- + (T-1)(C_1^- - C_2^-),\, TC_2^+ + (T-1)(C_1^+ - C_2^+)\right\}&\\ = \min\left\{C_2^- + (T-1)C_1^-,\, C_2^+ + (T-1)C_1^+\right\}&.\end{aligned}$$

If (i) $C_1^- \leqslant C_1^+$ and $C_2^- \leqslant C_2^+$ or (ii) $C_1^+ \leqslant C_1^-$ and $C_2^+ \leqslant C_2^-$ then the upper bound equals the lower bound, and this proves the following sufficient optimality condition.

Proposition 1. *If (i) $C_1^- \leqslant C_1^+$ and $C_2^- \leqslant C_2^+$ or (ii) $C_1^+ \leqslant C_1^-$ and $C_2^+ \leqslant C_2^-$, then it is optimal to schedule all jobs at the same time.*

As a simple consequence we note that in the following situations it is optimal to schedule all jobs at the same time:

- $C_1^- = C_1^+$, so the network is balanced, or
- $C_1^- \leqslant C_1^+$ and $J \supseteq \delta^-(v)$, or
- $C_1^+ \leqslant C_1^-$ and $J \supseteq \delta^+(v)$.

For a time horizon of two time periods the problem asks for a partition of the job set $J = J_1 \cup J_2$ into two parts such that the total flow is maximized, i.e. we want to find

$$\max_{J_1\cup J_2=J}\left[\min\left\{\sum_{a\in\delta^-(v)\setminus J_1} u_a,\ \sum_{a\in\delta^+(v)\setminus J_1} u_a\right\} + \min\left\{\sum_{a\in\delta^-(v)\setminus J_2} u_a,\ \sum_{a\in\delta^+(v)\setminus J_2} u_a\right\}\right].$$

The following proposition shows that it is NP-hard to decide if the trivial partition $J_1 = J$ and $J_2 = \varnothing$ is optimal.

Proposition 2. *For a network with one transhipment node and a time horizon of two periods it is NP-hard to decide if it is optimal to schedule all jobs at time 1.*

Proof. Reduction from PARTITION (see [7]). An instance is given by a set $D = \{d_1, \dots, d_m\}$ of positive integers with $\sum_{i=1}^m d_i = 2B$, and the problem is to decide if there is a partition $D = D_1 \cup D_2$ such that $\sum_{d\in D_1} d = \sum_{d\in D_2} d = B$. We consider the network shown in Figure 2 where every arc except the bold arc from v to t has an associated job. Scheduling

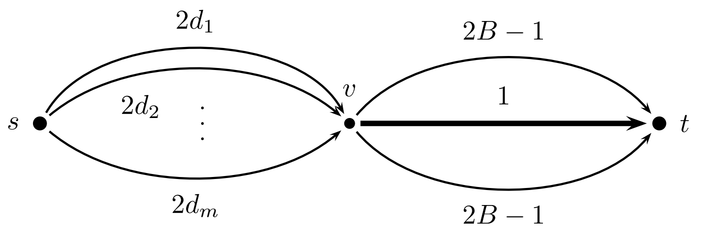

Figure 2: The network for the reduction from PARTITION. Arcs are labeled with capacities.

all jobs at time 1 gives a total flow of $4B - 1$. A total flow of $4B$ is possible if and only if there is a flow of $2B$ in each time period, and this is equivalent to a positive solution for the PARTITION instance. □

The reduction from PARTITION suggests the use of dynamic programming to obtain a pseudopolynomial algorithm for the single node problem. This is indeed possible, and in fact can be done more generally for series-parallel networks. This more general approach is presented in the next section (see Corollary 1).

2 General Networks

In this section we explore complexity issues for networks with more than one transhipment node and also discuss some of its tractable subclasses. We start with a lemma generalizing the upper bound in the single node case.

Lemma 1. *Let $S \subseteq A$ be any s-t cut in the network. The objective value for problem (1) – (7) is bounded above by*

$$T \sum_{a\in S\setminus J} u_a + (T-1) \sum_{a\in S\cap J} u_a.$$

Proof. Since S is a cut, the total flow over the whole time horizon is bounded above by

$$\sum_{i=1}^T \sum_{a\in S} x_{ai} = \sum_{a\in S} \sum_{i=1}^T x_{ai} = \sum_{a\in S\setminus J} \sum_{i=1}^T x_{ai} + \sum_{a\in S\cap J} \sum_{i=1}^T x_{ai} \leqslant \sum_{a\in S\setminus J} Tu_a + \sum_{a\in S\cap J} (T-1)u_a,$$

by the combination of (2) — (4). The result follows. □

As an immediate consequence we obtain that the problem is tractable when the set of arcs that have to undergo maintenance contains a minimum cut.

Proposition 3. *If J contains a minimum cut S of the network then it is optimal to schedule all jobs at the same time.*

Proof. Since S is a minimum cut, the maximum flow in any period in which no maintenance is scheduled is $\sum_{a\in S} u_a$, so scheduling all jobs at time 1 gives a total flow of $(T-1)\sum_{a\in S} u_a$, which achieves the upper bound from Lemma 1. □

In Section 1, we showed that the case of single-node networks with balanced capacities is easy. The following theorem shows that the balanced property alone is not enough.

Proposition 4. *The problem is strongly NP-hard for balanced networks.*

Proof. Reduction from 3-Partition (see [7]). A 3-Partition instance is given by an integer B and a set $\{d_1,\dots,d_{3m}\}$ of integers with $B/4 < d_i < B/2$ for all i and $\sum_{i=1}^{3m} d_i = mB$. The problem is to decide if there is a partition of the set $\{d_1,\dots,d_{3m}\}$ into m triples such that the sum of each triple equals B. Consider the network shown in Figure 3, where the arc labels indicate capacities, and the bold arcs don't have jobs associated with them. Also let the time horizon be $T = m$. By Lemma 1 applied to the cut $(\{s\},\{v_1,v_2,t\})$, the

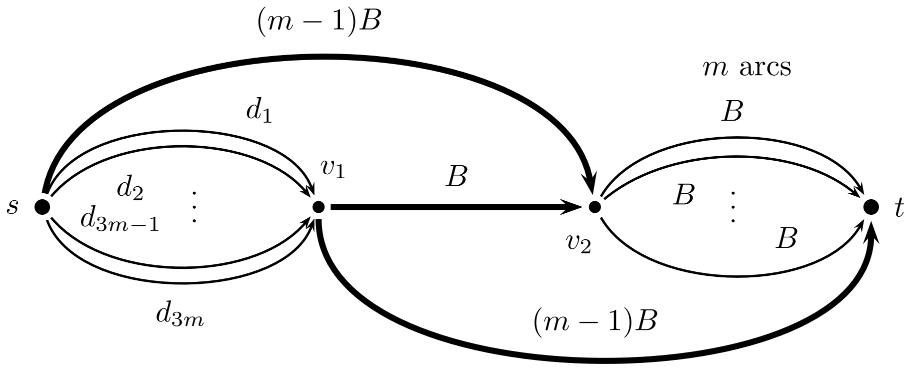

Figure 3: The network for the reduction from 3-Partition.

total flow is bounded by

$$T(m-1)B + (T-1)\sum_{i=1}^{3m} d_i = 2m(m-1)B.$$

To achieve the bound $2m(m-1)B$ the arc (s,v_2) is at capacity in every time period. This implies that we have to schedule exactly one job on the arcs between v_2 and t in each time period. Now flow conservation in node v_2 implies that the flow on the arc (v_1,v_2) is zero in every time period. Considering the cut $(\{s,v_1,v_2\},\{t\})$ the bound $2m(m-1)B$ can be achieved only if the arc (v_1,t) is at capacity in every time period. Using flow conservation in node v_1 we can now conclude that in order to achieve the bound $2m(m-1)B$ it is necessary and sufficient to send in each time period $(m-1)B$ units of flow from s to v_1, and this can be done if and only if the answer for the 3-PARTITION instance is YES. □

As already mentioned, not all instances of the general balanced network are hard. The single-node variant is easy and is in fact a special case of a series-parallel network. Note that the network constructed in the above NP-hardness proof is not series-parallel. We show below (Proposition 5) that indeed the case of series-parallel balanced networks is easy. However we first make precise our definition of series-parallel. Throughout this paper, by *series-parallel network* we mean a *two-terminal series-parallel network*: a network that has a single source and single sink and is constructed by a sequence of series and parallel compositions starting from single arcs. For two networks N_1 and N_2 the *parallel composition* of N_1 and N_2 is obtained by identifying the source node s_1 and sink node t_1 of N_1 with the source node s_2 and sink node t_2 of N_2, respectively. The *series composition* of N_1 and N_2 is obtained by identifying the sink node t_1 of N_1 with the source node s_2 of N_2. We denote these compositions by $N_1 \oplus_P N_2$ and $N_1 \oplus_S N_2$, respectively. The next proposition shows that series-parallel balanced networks are tractable.

Proposition 5. *If the network is series-parallel and balanced then it is optimal to schedule all jobs at the same time.*

Proof. For a network $N=(V,A,s,t,u)$ and a subset $J \subseteq A$ let $F_{N,J}$ denote the maximum flow value in the network $N=(V, A\setminus J, s, t, u\,|_{A\setminus J})$. The statement that it is optimal to schedule all jobs at the same time is equivalent to

$$F_{N,J\cup J'} + F_{N,\varnothing} \geqslant F_{N,J} + F_{N,J'}$$

for all $J, J' \subseteq A$ (see [3]). We prove the proposition by induction on the structure of the graph. The claim holds for the base case of a single arc. So assume that N is a series-parallel network that is not a single arc. Then $N = N_1 \oplus_P N_2$ or $N = N_1 \oplus_S N_2$ for some smaller networks $N_i = (V_i, A_i, s_i, t_i, u_i)$ $(i \in \{1,2\})$, and by induction

$$F_{N_i,J_i\cup J_i'} + F_{N_i,\varnothing} \geqslant F_{N_i,J_i} + F_{N,J_i'}$$

for all $J_i, J_i' \subseteq A_i$. Now let $J, J' \subseteq A = A_1 \cup A_2$ be arbitrary and put $J_i = J \cap A_i$ and $J_i' = J' \cap A_i$ for $i \in \{1,2\}$.

Case 1. $N = N_1 \oplus_P N_2$. Then

$$\begin{aligned} F_{N,J\cup J'} + F_{N,\varnothing} &= F_{N_1,J_1\cup J_1'} + F_{N_1,\varnothing} + F_{N_2,J_2\cup J_2'} + F_{N_2,\varnothing} \\ &\geqslant F_{N_1,J_1} + F_{N_1,J_1'} + F_{N_2,J_2} + F_{N_2,J_2'} = F_{N,J} + F_{N,J'}. \end{aligned}$$

Case 2. $N = N_1 \oplus_S N_2$. By the assumption that N is balanced, we have $F_{N,\varnothing} = F_{N_1,\varnothing} = F_{N_2,\varnothing}$, and we denote this common value by F. Now

$$\begin{aligned} F_{N,J\cup J'}+F &= \min\{F_{N_1,J_1\cup J_1'},\, F_{N_2,J_2\cup J_2'}\}+F \geqslant \min\{F_{N_1,J_1}+F_{N_1,J_1'},\, F_{N_2,J_2}+F_{N_2,J_2'}\} \\ &\geqslant \min\{F_{N_1,J_1},\, F_{N_2,J_2}\} + \min\{F_{N_1,J_1'},\, F_{N_2,J_2'}\} = F_{N,J} + F_{N,J'}. \end{aligned}$$ □

Since scheduling all jobs in the same period seems to be optimal in some cases, we now ask how well it performs as an approximation algorithm in the general case.

Proposition 6. *Scheduling all jobs in the same period gives an approximation ratio no less than* $\frac{(T-1)}{T}$.

Proof. Let z^* denote the optimal value and $\tilde{z}$ denote the throughput obtained by scheduling all arcs in the same period. Clearly $\tilde{z} \geqslant (T-1)F$ and $z^* \leqslant TF$, so

$$\frac{\tilde{z}}{z^*} \geqslant \frac{T-1}{T}. \qquad \square$$

Thus scheduling all jobs in the same period is asymptotically optimal in the sense that the approximation ratio approaches 1 as T tends to infinity. In general the analysis in the proof of Proposition 6 is tight as can be seen by considering the network in Figure 4 where all arcs have unit capacity and the set J of arcs with a job is the set of the two arcs from s to v. Then scheduling both outages at the same time yields a total throughput of $T-1$ while for $T \geqslant 2$ the outages can be scheduled in different time periods which yields a total throughput of T, and the approximation ratio in this case is $(T-1)/T$. For certain

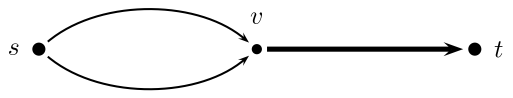

Figure 4: A network where the bound of Proposition 6 is tight.

instances the analysis of the approximation ratio can be slightly improved. For this let

$$L = (T-1)F + \min_{S\in\mathcal{S}} \sum_{a\in S\setminus J} u_a, \qquad U = \min_{S\in\mathcal{S}} \left(T\sum_{a\in S} u_a - \sum_{a\in S\cap J} u_a \right)$$

where the minima are over the the set $\mathcal{S}$ of all s-t-cuts in the network. Clearly, L is the objective value for scheduling all jobs in the same time period, and from Lemma 1 it follows that U is an upper bound for the optimal objective value. Thus L/U is a lower bound for the approximation ratio, and since $L \geqslant (T-1)F$ and $U \leqslant TF$ this is at least as good as the bound from Proposition 6. Note that this generalizes Proposition 3: if J contains a min cut S then both of the minima in the definitions of L and U are obtained for S, and we get $L = U = (T-1)F$, the approximation ratio is 1, in other words it is optimal to schedule all jobs in the same time period.

Next we present an algorithm for general series-parallel networks, which for the instance used in the proof of Proposition 2 coincides with the well known dynamic programming algorithm for Partition. With feasible values for the binary variables y_{ai} ($a \in J$, $i \in [T]$) we can associate a vector $z^y = (z_i^y)_{i=1,\dots,T}$ where z_i^y denotes the maximum flow in the network with arc set $A \setminus \{a \ : \ y_{ai} = 0\}$. By symmetry we may assume that $z_1^y \geqslant z_2^y \geqslant \cdots \geqslant z_T^y$. Our algorithm exploits the fact that many different maintenance schedules y may give rise to the same vector z^y to gain efficiency over naive enumeration of schedules. The algorithm computes the possible vectors z for subnetworks of the network N, starting from single arcs. To do this we use *sp-trees* which encode the construction of series-parallel networks. An sp-tree for a series-parallel network N is a full binary tree in which the leaves correspond to the arcs of N, any internal node corresponds to the composition of its two child nodes, and the type of composition (series or parallel) is indicated by a node label ('S' or 'P', respectively). Figure 5 shows a network and the corresponding sp-tree. Recognition of

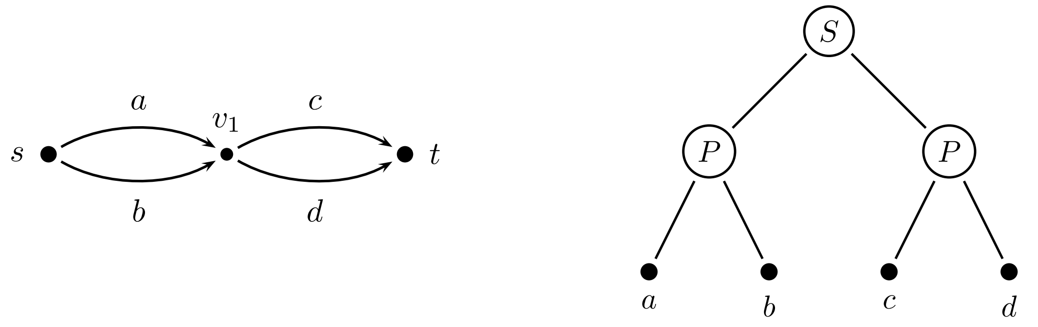

Figure 5: A series-parallel network and the corresponding sp-tree.

series-parallel networks and construction of an sp-tree can be done in linear time [12]. So assume we are given the sp-tree with node set $\mathcal{V} = \mathcal{L} \cup \mathcal{W}$, where $\mathcal{L}$ is the set of leaves and $\mathcal{W}$ is the set of internal nodes. The set $\mathcal{W}$ is partitioned into level sets $\mathcal{W}_i$ where $\mathcal{W}_i$ is the set of internal nodes at distance i from the root. Let d be the largest index such that $\mathcal{W}_d \neq \varnothing$. The lists of possible maximum flow vectors z are initialized at the leaves by assigning a list with a single element to the leaf corresponding to arc a. The unique element in this list is $(u_a, u_a, \dots, u_a, 0)$ if $a \in J$ and $(u_a, u_a, \dots, u_a, u_a)$ if $a \notin J$. Then the lists for the internal nodes are computed going up in the tree as described in Algorithm 1. The list generated for each node $v \in \mathcal{V}$ is denoted by L_v. This algorithm returns the maximum total throughput, and it is easy to see how to keep track of corresponding schedules for all the elements of the lists in the internal nodes.

Example 1. Suppose for the network in Figure 5 the capacities are $u_a = 4$, $u_b = 1$, $u_c = u_d = 2$, the set of arcs with a job is $J = \{a, b, c\}$, and the time horizon is $T = 3$. Figure 6 illustrates how the lists for the internal nodes are computed. The optimal vector

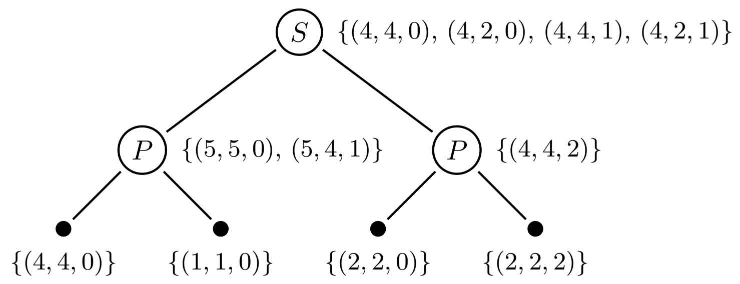

Figure 6: Computation of the possible maximum flow vectors.

Algorithm 1 Maximizing total throughput for series-parallel networks

```
for $v \in \mathcal{L}$ do
    Let $a \in A$ be the arc corresponding to $v$
    if $a \in J$ then $L_v \leftarrow \{(u_a, u_a, \dots, u_a, 0)\}$ else $L_v \leftarrow \{(u_a, u_a, \dots, u_a, u_a)\}$
for $i = d, d-1, \dots, 0$ do
    for $v \in \mathcal{W}_i$ do
        $L_v \leftarrow \{\}$      /* initialize empty list*/
        Let $u$ and $w$ be the child nodes of $v$
        for each $z \in L_u$, $z' \in L_w$ and $\pi$ a permutation of $\{1, 2 \dots, T\}$ do
            if $v$ is a parallel composition node then
                for $i \in [T]$ do $z''_i = z_i + z'_{\pi(i)}$
            else /* $v$ is a series composition node */
                for $i \in [T]$ do $z''_i = \min\{z_i,\ z'_{\pi(i)}\}$
            sort the components of $z''$ in non-increasing order
            if $z'' \notin L_v$ then add $z''$ to $L_v$
Let $v$ be the root node of the sp-tree and return $\max\limits_{z \in L_v} \sum\limits_{i=1}^{T} z_i$
```

in the root node is $(4, 4, 1)$ giving a total throughput of 9, and this can be obtained by scheduling the job on arc b for the second time period and the jobs for arcs a and c for the third time period.

Bounding the runtime of Algorithm 1 we obtain the following complexity result.

Proposition 7. *For series-parallel networks with m arcs the problem can be solved in time $O\left(mT^{T+3/2}e^{-T}\log(T)(mB+1)^{2T}\right)$, where B is an upper bound for the capacities.*

Proof. The entries of the vectors in the lists at the internal nodes are bounded by mB, hence every list can contain at most $(mB+1)^T$ elements. Thus the loop over $(z, z') \in L_u \times L_w$ and permutations π is over at most $T!(mB+1)^{2T}$ elements. Inside this loop is another one giving an additional factor T, and a sorting operation which is at most a factor of $T\log(T)$. With the use of hash tables, checking z'' is not already in the list prior to insertion will not worsen the complexity of operations inside this loop, which is thus $T\log(T)$. In total there are $m-1$ internal nodes, thus the runtime is $O(T\log(T)T!(mB+1)^{2T}(m-1))$ from which the result follows when $T!$ is bounded using Stirling's formula. □

Corollary 1. *For series-parallel networks and fixed time horizon T the problem can be solved in time $O\left(m(mB)^{2T}\right)$.*

We add two remarks on an efficient implementation of Algorithm 1.

1. Any vector z that is dominated by another vector z' in the list, meaning that $z_i \leqslant z'_i$ for all $i \in \{1, 2 \dots, T\}$, can be removed immediately.

2. In the loop over $(z, z') \in L_u \times L_w$ and permutations π it is necessary to loop over all permutations only if the entries of the vectors z and z' are pairwise distinct. An efficient implementation detects the occurrence of multiple entries and restricts the range of the considered permutations accordingly.

3 Networks with all arcs having unit capacity

In this section we study the case that the capacity of every arc equals 1. We can aggregate all time periods and solve a standard max flow problem to get an upper bound. The max flow problem is

$$\begin{aligned}
\max \sum_{a\in\delta^+(s)} X_a - \sum_{a\in\delta^-(s)} X_a & & & (8)\\
\text{s.t.} \sum_{a\in\delta^+(v)} X_a = \sum_{a\in\delta^-(v)} X_a & & v \in N \setminus \{s,t\}, & (9)\\
X_a \leqslant T & & a \in A \setminus J, & (10)\\
X_a \leqslant T-1 & & a \in J, & (11)\\
X_a \geqslant 0 & & a \in A. & (12)
\end{aligned}$$

We will show that this upper bound is actually tight. This follows by induction once we can find a max flow X^* and cover all the arcs carrying flow T by a collection of arc disjoint s-t-paths. Given any max flow we can reduce the flow along any cycles carrying flow, and we can remove arcs with zero flow. So in order to prove that the upper bound is tight it is sufficient to prove the following result.

Proposition 8. *Let (V, A, s, t) be an acyclic network with source $s \in V$ and sink $t \in V$, and suppose $X^* : A \to [T]$ satisfies the flow conservation constraints*

$$\sum_{a\in\delta^-(v)} X_a^* = \sum_{a\in\delta^+(v)} X_a^* \qquad \textit{for all } v \in V \setminus \{s,t\}.$$

Then there is a collection $\mathcal{P}$ of arc-disjoint s-t-paths such that $A^* \subseteq \bigcup_{P\in\mathcal{P}} P$, where $A^* = \{a \in A \ :\ X_a^* = T\}$ is the set of arcs carrying flow T.

Proof. We consider the following binary program, in which $(\xi_a)_{a\in A}$ induces a set of arc-disjoint s-t-paths:

$$\begin{aligned}
\max \sum_{a\in A^*} \xi_a & & & (13)\\
\text{s.t.} \quad \sum_{a\in\delta^+(v)} \xi_a - \sum_{a\in\delta^-(v)} \xi_a = 0 & & v \in V \setminus \{s,t\}, & (14)\\
\xi_a \in \{0,1\} & & a \in A. & (15)
\end{aligned}$$

We have to prove that the optimal objective value for the problem (13) — (15) is $|A^*|$. The flow conservation constraints (14) form a network matrix, hence we do not lose anything by relaxing integrality, i.e. we can replace (15) by $0 \leqslant \xi_a \leqslant 1$ for all $a \in A$. The dual problem can be written in the form

$$\begin{aligned} \min \sum_{a\in A} \eta_a & & (16)\\ \text{s.t.}\quad \pi_v - \pi_w + \eta_a &\geqslant 0 & a=(v,w)\in A\setminus A^*, \qquad (17)\\ \pi_v - \pi_w + \eta_a &\geqslant 1 & a=(v,w)\in A^*, \qquad (18)\\ \pi_s = \pi_t &= 0, & (19)\\ \eta_a &\geqslant 0 & a\in A. \qquad (20) \end{aligned}$$

A feasible solution with objective value $|A^*|$ is given by $\pi_v = 0$ for all $v \in V$, $\eta_a = 0$ for $a \in A \setminus A^*$, and $\eta_a = 1$ for $a \in A^*$. In order to prove our claim we have to check that $|A^*|$ is a lower bound, i.e. that $\sum_{a\in A} \eta_a \geqslant |A^*|$ for every feasible solution. To see this let $\mathcal{P}'$ be any decomposition of the flow X^* into paths, that is a collection of s-t-paths such that every arc a is contained in exactly X^*_a paths $P \in \mathcal{P}'$. Adding up constraints (17) and (18) over the arcs of any path $P \in \mathcal{P}'$, we obtain $\sum\limits_{a\in P} \eta_a \geqslant |P \cap A^*|$, hence

$$\sum_{a\in A} X^*_a \eta_a = \sum_{P\in\mathcal{P}'} \sum_{a\in P} \eta_a \geqslant \sum_{P\in\mathcal{P}'} |P\cap A^*| = T|A^*|.$$

Finally, using $X^*_a \leqslant T$ for all $a \in A$,

$$\sum_{a\in A} \eta_a \geqslant \sum_{a\in A} \frac{X^*_a}{T} \eta_a \geqslant |A^*|. \qquad \square$$

To summarize, if $u_a = 1$ for all $a \in A$ then the problem can be reduced to solving the max flow problem (8) – (12) followed by T instances of (the linear relaxation of) the problem (13) – (15). Consequently, these instances can be solved in time polynomial in the size of the network and the time horizon T.

Remark 1. It is straightforward to generalize the result of this section to the problem where every arc can have several unit processing time jobs which must not overlap. The only necessary modification in this case is to replace the right-hand side of constraint (11) by $T - m_a$ where m_a is the number of jobs that have to be scheduled on arc a.

4 Future Work

It would be interesting to find a more combinatorial proof of Proposition 8, rather than resorting to solution of linear programs. A combinatorial proof may suggest combinatorial algorithms for constructing the arc-disjoint paths that cover all arcs with flow T. Practical algorithms for more general problems are also of interest.

References

[1] N. Boland, T. Kalinowski, H. Waterer and L. Zheng. “An optimisation approach to maintenance scheduling for capacity alignment in the Hunter Valley coal chain”. In: *Proc. 35th APCOM Symposium: Applications of Computers and Operations Research in the Minerals Industry.* Ed. by E.Y. Baafi, R.J. Kininmonth and I. Porter. The Australasian Institute of Mining and Metallurgy Publication Series 11. Wollongong, Australia, 2011, pp. 887–897.

[2] N. Boland, T. Kalinowski, H. Waterer and L. Zheng. “Mixed integer programming based maintenance scheduling for the Hunter Valley Coal Chain”. In: *Journal of Scheduling* 16.6 (2013), pp. 649–659. DOI: 10.1007/s10951-012-0284-y.

[3] N. Boland, T. Kalinowski, H. Waterer and L. Zheng. “Scheduling arc maintenance jobs in a network to maximize total flow over time”. In: *Discr. Appl. Math.* 163 (2014), pp. 34–52. DOI: 10.1016/j.dam.2012.05.027.

[4] N. Boland and M. Savelsbergh. “Optimizing the Hunter Valley coal chain”. In: *Supply Chain Disruptions: Theory and Practice of Managing Risk.* Ed. by H. Gurnani, A. Mehrotra and S. Ray. Springer-Verlag London Ltd., 2011, pp. 275–302.

[5] L. Fleischer. “Universally maximum flow with piecewise-constant capacities”. In: *Networks* 38.3 (2001), pp. 115–125. DOI: 10.1002/net.1030.

[6] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton, N.J.: Princeton Univ. Press, 1962.

[7] M.R. Garey and D.S. Johnson. *Computers and intractability, a guide to the theory of NP–completeness.* W.H. Freeman, 1979.

[8] B. Hajek and R.G. Ogier. “Optimal dynamic routing in communication networks with continuous traffic”. In: *Networks* 14.3 (1984), pp. 457–487. DOI: 10.1002/net.3230140308.

[9] B. Hoppe and É. Tardos. “Polynomial time algorithms for some evacuation problems”. In: *Proc. 5th ACM-SIAM symposium on discrete algorithms SODA 1994.* Society for Industrial and Applied Mathematics. 1994, pp. 433–441.

[10] M.L. Pinedo. *Scheduling: theory, algorithms, and systems.* Springer, 2012.

[11] Mohit Tawarmalani and Yanjun Li. “Multi-period maintenance scheduling of tree networks with minimum flow disruption”. In: *Naval Research Logistics (NRL)* 58.5 (2011), pp. 507–530. DOI: 10.1002/nav.20455.

[12] Jacobo Valdes, Robert E Tarjan and Eugene L Lawler. “The recognition of series parallel digraphs”. In: *Proc. 11th ACM symposium on Theory of computing, STOC 1979.* ACM. 1979, pp. 1–12. DOI: 10.1145/800135.804393.

Incremental network design with shortest paths[*]

Matthew Baxter Tarek Elgindy Andreas Ernst Thomas Kalinowski
Martin Savelsbergh

Abstract

We introduce a class of incremental network design problems focused on investigating the optimal choice and timing of network expansions. We concentrate on an incremental network design problem with shortest paths. We investigate structural properties of optimal solutions, we show that the simplest variant is NP-hard, we analyze the worst-case performance of natural greedy heuristics, and we derive a 4-approximation algorithm.

1 Introduction

Consider a network optimization problem, e.g., a shortest path problem, a maximum flow problem, or a traveling salesman problem. Next, assume that this optimization problem has to be solved in a number of consecutive time periods and that in each time period the value of an optimal solution is recorded, e.g., the cost of an $s-t$ path, the value of an $s-t$ flow, or the cost of a TSP tour. Let the objective be to minimize or maximize the total cost or value over the planning horizon. At this point, that simply means solving the network optimization problem and multiplying the value of an optimal solution with the number of time periods in the planning horizon. It becomes more interesting when a budget is available in each time period to expand the network, i.e., to build additional links. Expanding the network may improve the cost or value of an optimal solution to the network optimization problem in future time periods and thus may improve the total cost or value over the planning horizon. However, deciding which links to build and the sequence in which to build them is nontrivial. In part, because in some situations the benefits of building a link will only materialize when other links have been built as well, e.g., adding a single link to the network does not lead to a shorter TSP tour, but adding two links to the network does.

We introduce a class of incremental network design problems focused on the optimal choice and timing of network expansions given that these network expansions impact the value a solution to an optimization problem that is solved on the network in each of the periods of the planning horizon. We concentrate on the incremental network design problem with shortest paths. We investigate structural properties of optimal solutions, we show that even the simplest variant is NP-hard, we establish a class of instances that can be solved in

[*] *European Journal of Operational Research* **238**(3), 675–684, 2014, doi:10.1016/j.ejor.2014.04.018

polynomial time, we analyze the worst-case performance of natural greedy heuristics, and we derive a 4-approximation algorithm.

Even though single-stage or single-period network design problems have been studied extensively, multi-stage or multi-period network design problems, which occur just as often in practice, have received much less attention. We hope that our investigation demonstrates that multi-period network design problems present interesting challenges and can produce intriguing and surprising results.

The remainder of the paper is organized as follows. In Section 2, we introduce the class of incremental network design problems. In Section 3, we present a brief literature review. In Section 4, we introduce the incremental network design problem with shortest paths. In Sections 5 and 6, we analyze the complexity of the incremental network design problem with shortest paths, and we explore the performance of natural greedy heuristics, respectively. In Section 7, we develop a 4-approximation algorithm for the incremental network design problem with shortest paths. Finally, in Section 8, we discuss possible extensions and future research.

2 A Class of Incremental Network Design Problems

Incremental network design problems have two characteristic features: a design feature, since we are deciding which arcs will be part of a network, and a multi-period feature, since the ultimate network design is built over a number of time periods.

The general structure of an incremental network design problem is as follows. We are given a network $D = (N, A)$ with node set N and arc set $A = A_e \cup A_p$, where A_e contains *existing* arcs and A_p contains *potential* arcs. Each arc $a \in A$ has a capacity C_a. Let T be the planning horizon. A budget B^t is available in every time period $t \in \{1, \dots, T\}$. The budget can be used to build potential arcs $a \in A_p$, which will be available for use in the following period. For each potential arc $a \in A_p$, there is an associated build-cost $c_a \leqslant B$. Let y_a^t be a 0-1 variable indicating whether arc $a \in A_p$ has been built in or before time period t, with all potential arcs initially unbuilt ($y_a^0 = 0$). Thus, $y_a^t - y_a^{t-1} = 1$ indicates that arc a is built in time period t and can be utilized in period $t+1$. In every time period, a network optimization problem P has to be solved over the usable arcs in time period t, i.e., the existing arcs and the potential arcs that have been built before time period t. Let x_a^t represent the flow on arc $a \in A$ in time period $t \in \{1, \dots, T\}$ in an optimal solution to the network optimization problem. Let $F(P)$ define the “structure” of feasible solutions to the network optimization problem, i.e., the set of constraints imposed on the flow variables (that it has to be an $s - t$ path, $s - t$ flow, a TSP tour, etc.). The value of an optimal solution to the network optimization P in time period t is function of the flows on the arcs in that period and denoted by $c(x^t)$. The objective is to minimize the total cost over the planning period. Thus, the generic formulation of an incremental network design problem is as follows:

$$\begin{aligned}
\min \sum_{t\in\{1,\ldots,T\}} c(x^t) + & \sum_{t\in\{1,\ldots,T\},a\in A_p} c_a(y_a^t - y_a^{t-1}) & \\
\text{s.t.}\quad & x^t \in F(P) && \text{for all } t \in \{1,\ldots,T\},\\
& x_a^t \leqslant C_a y_a^{t-1} && \text{for all } a \in A_p, t \in \{1,\ldots,T\},\\
\sum_{a\in A_p} c_a(y_a^t - y_a^{t-1}) & \leqslant B^t && \text{for all } t \in \{1,\ldots,T\},\\
& y_a^t \geqslant y_a^{t-1} && \text{for all } a \in A_p, t \in \{2,\ldots,T\}
\end{aligned}$$

An incremental network design problem has characteristics in common with network design problems and with dynamic facility location problems. A brief review of some relevant literature is given below.

3 Literature review

Network design is a fundamental optimization problem and has a rich research tradition. The seminal paper by Magnanti and Wong [8] discusses many of its features, applications, models, and algorithms, with an emphasis on network design in transportation planning. Kerivin and Mahjoub [5] survey many of network design problems studied in telecommunications. The paper by Magnanti and Wong [8] mentions “Time Scale” as one of characteristics of a network design problem that can vary in different planning environments, e.g., transportation and water resource design decisions have long-term effects whereas communication system designs frequently are more readily altered. Not withstanding, the paper focuses exclusively on single-period or single-stage network design problems. Recently, the interest in multi-period or multi-stage network design problems in the area of transportation planning has picked up, partly because it better meets practitioners needs, as in many environments network design decisions span planning periods of up to 25 years and the intermediate network configurations are of concern as well as the final network configuration (see for example [6] and [11]). A class of network design problems where construction over time has been studied extensively is dynamic facility location (the recent review of Arabani and Farahani [1] is completely dedicated to dynamic facility location).

Studying approximation algorithms for network design problems has been popular as well, especially in the computer science community. Two prime examples are the papers by Goemans *et al.* [3] and Gupta *et al.* [4]. These approximation algorithms are for single-period network design problems. Multi-period or incremental approximation has been introduced in the context of facility location by Mettu and Plaxton [9]. They consider a situation where a company is building facilities in order to supply its customers, but because of capital considerations, the company will build the facilities over time. The question asked is whether the company can plan its future expansion in such a way that when it has opened the first k facilities, it is close, in value, to that of an optimal solution in which any choice of k facilities is opened. This problem is known as the incremental k-median problem. More formally, in the incremental k-median problem, we are given the input of the k-median problem without the parameter k and must produce a sequence of the facilities. For each k, consider the ratio of the cost of opening the first k facilities in the ordering to the cost of an optimal k-median solution. The goal of the problem is to find an ordering that minimizes the maximum of this ratio over all values of k. See Lin *et al.* [7] for more on incremental optimization problems.

This brief literature review has covered areas of research that are relevant to the study of the class of incremental network design problems introduced in this paper. The review identified survey papers on single-period network design, recent papers discussing the practical importance of investigating multi-period network design, and fundamental papers on approximation algorithms for network design.

For the remainder of the paper, we focus on one particular incremental network design problem, namely the incremental network design problem with shortest paths (INDP-SP).

4 The Incremental Network Design Problem with Shortest Paths

We are given a network $D = (N, A)$ with node set N ($|N| = n$) and arc set $A = A_e \cup A_p$ ($|A| = m$), where A_e is the set of *existing* arcs and A_p is the set of *potential* arcs, as well as a source $s \in N$ and sink $t \in N$. For each arc $a \in A$, we are given a length $l_a \geqslant 0$. Let $T = |A_p| + 1$ be the planning horizon. In every time period, we have the option to expand the usable network, which initially consists of only the existing arcs, by "building" a single potential arc $a \in A_p$, which will be available for use in the following period. In every period, the cost (or length) of a shortest $s - t$ path is incurred (using only usable arcs, i.e., existing arcs and potential arcs that have been built in previous periods). The objective is to minimize the total cost over the planning horizon. Note that the length of the planning horizon ensures that every potential arc can be built. This also implies that a shortest $s - t$ path in D will always be built. We will refer to such a shortest $s - t$ path as an *ultimate* shortest $s - t$ path to distinguish it from a shortest $s - t$ path in a time period, which depends on the potential arcs that have been built up to and including that period. Note that there may be many ultimate shortest paths in the network, and the sequence of potential arcs built in an optimal solution may not be unique. This problem is related to, but also quite different from, shortest path re-optimization problems as studied by Gallo [2] and extended by Pallottino and Scutellà [10].

An example. Consider the network shown in Figure 1, where solid arcs represent existingarcs and dashed arcs represent potential arcs. A path of length 52 can be built in three periods and the ultimate shortest path of length 4 can be constructed in 4 periods. However, the optimal solution is to first build arcs a, b, and c to give a path of length 54, and then to build arcs d, e, and f to complete the ultimate shortest path for a total cost over the planning horizon of 3174.

We start by observing a useful property of an optimal solution. Let $\bar{T}$ denote the time period in which an ultimate shortest path is completed in an optimal solution.

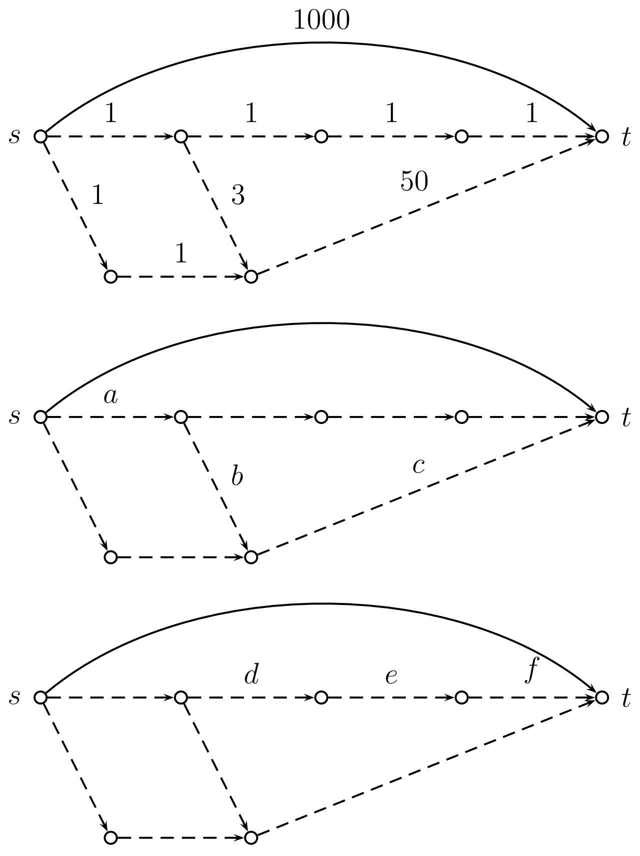

Figure 1: An example.

Observation 1. Let $(a_1, a_2, \ldots, a_{\bar{T}})$ be an optimal sequence of potential arcs to build $(a_i \in A_p)$. Let $(P_1, P_2, \ldots, P_{\bar{T}}, P_{\bar{T}+1})$ be a sequence of associated shortest paths, where P_i is the path used while a_i is being constructed. Let c_i be the cost of path P_i. By grouping paths with identical costs, we obtain subsequences $S_1, S_2, \ldots, S_K$. Let P_{S_i} denote a path associated with subsequence S_i. Let A_{S_i} denote the potential arcs built during subsequence S_i. Then we have

$$A_{S_i} = P_{S_{i+1}} \cap A_p \setminus (P_{S_1} \cup \cdots \cup P_{S_i}) \text{ for } i = 1, \ldots, K-1.$$

Proof. Each arc a_j for $j = 1, \ldots, \bar{T}$ must contribute to the construction of some path P_{S_i} with $i \in \{1, \ldots, K\}$, otherwise an improved solution is obtained by simply removing a_j from $(a_1, a_2, \ldots, a_{\bar{T}})$. Furthermore, suppose there exist j and j' with $j < j'$ and i and i' with $i < i'$ such that $a_{j'}$ contributes to the construction of path P_{S_i} and a_j contributes to the construction of path $P_{S_{i'}}$, but not to the construction of paths P_{S_k} with $k \in \{i, i+1, \ldots, i'-1\}$. By interchanging a_j and $a_{j'}$, the paths P_{S_k} with $k \in \{i, i+1, \ldots, i'-1\}$ are completed earlier, while all other paths are completed at the same time, which reduces the total cost. □

Thus, the problem can be viewed as seeking a sequence of potential arcs to build, but also as seeking a sequence of paths to be constructed, and the last property establishes that in an optimal solution any potential arc that is built before an ultimate shortest path is completed will be part of the next shortest path.

5 Complexity

Since the shortest path problem is polynomially solvable, the complexity of the incremental network design problem with shortest paths is not obvious. The following theorem shows that even the simplest variant of the incremental network design problem with shortest paths as defined above is NP-hard.

Theorem 1. *The incremental incremental network design problem with shortest paths is NP-hard.*

Proof. Reduction from 3-SAT. Consider an instance of 3-SAT with m clauses, i.e., $c_1 \wedge c_2 \wedge \cdots \wedge c_m$, each of which contains three literals, i.e., $c_i = (l_{i1} \vee l_{i2} \vee l_{i3})$, where each literal is a Boolean variable x_j or its negation. Let there be n Boolean variables. We construct the instance of the incremental network design problem with shortest paths given in Figure 2. As before, existing arcs are shown as solid arcs and potential arcs are shown as dashed arcs. Let $M_m = 1$, $M_{m-1} = 2$, …, $M_1 = m$, $M_0 = m + 1$ and $I = \lceil \frac{(m+1)(n+1)}{2} \rceil + 1$, and all other arcs have a cost of 0. Observe that initially there is only a single path from s to t with cost I. Next observe that a shorter path of length M_0 can be constructed in n periods by building one of the two potential arcs associated with a variable x_i (i.e., x_i or $\neg x_i$) for each of the n variables. Next observe that a path from the source to the sink associated with clause c_i requires that at least one of the potential arcs associated with the literals in c_i has to be build and that the cost of such a path is M_i. It is now easy to see that if the instance of SAT-3 can be satisfied, a solution to the incremental shortest path problem with value $nI + (n + 1)M_0 + M_1 + M_2 + \cdots + M_m$ exists (all but n of the potential arcs, all associated with the Boolean variables, will be build and the ultimate shortest path, with length 0, will be used in the last in periods). Furthermore, if the instance of 3-SAT cannot be satisfied, a solution with strictly greater value will be obtained. Note that sequence of potential arcs through nodes $s_1, s_2, \ldots, s_n$ is introduced to ensure that it is always advantages to build n potential arcs associated with the Boolean variables before building a path associated with a clause. □

Even though the incremental network design problem with shortest paths is NP-hard, we next show that there exist special cases which are polynomially solvable. Consider the special case of *disjoint paths* graphs, in which the arcs $a \in A$ form node-disjoint paths from s to t. More specifically, consider a disjoint paths graph with $r + 1$ paths $P_0, P_1, \ldots, P_r$. Because the paths are disjoint, the order in which the potential arcs on a path are build is immaterial. Therefore, for $i \in \{0, 1, \ldots, r\}$, let q_i denote the number of potential arcs on P_i, and, as before, let c_i denote the cost of path P_i. It is easy to see that if $c_i > c_j$ and

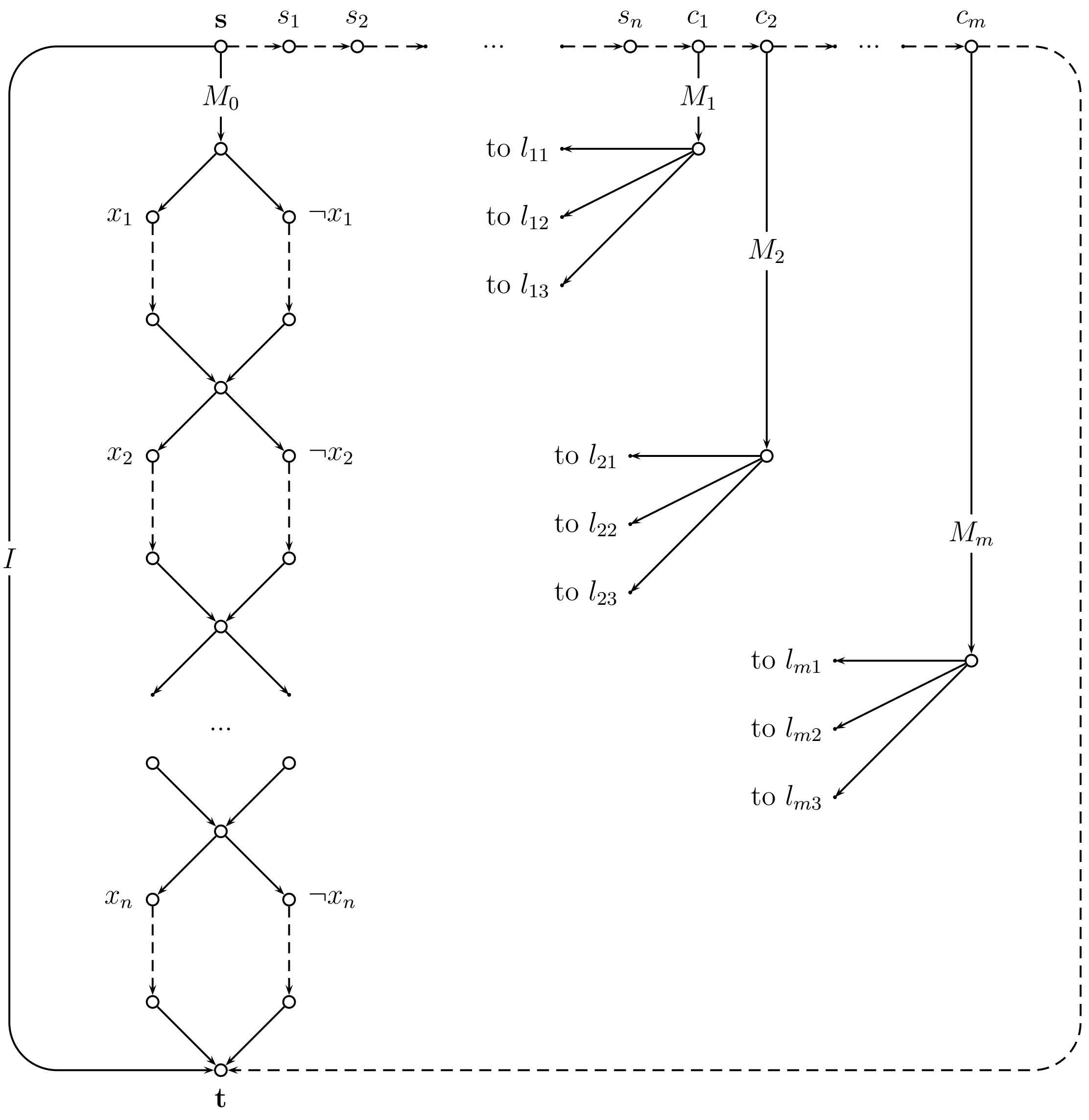

Figure 2: Instance of INDP-SP used in the reduction from 3-SAT.

$q_i \geqslant q_j$, then the path P_i will not be constructed in an optimal solution. Therefore, we may assume that $0 = q_0 < q_1 < \cdots < q_r$ and $c_0 > c_1 > \cdots > c_r$. We construct an auxiliary digraph $\bar{D}$ with node set $\{0, 1, \dots, r+1\}$. An arc set of $(r, r+1)$ with weight c_r, and arcs (i, j) for $0 \leqslant i < j \leqslant r$ with weights

$$w(i, j) = q_j c_i + (\sum_{k=i+1}^{j-1} q_k) c_r.$$

The paths from 0 to r in $\bar{D}$ are in one-to-one correspondence to solutions of the given disjoint paths instance: a path $(0 = i_0, i_1, \dots, i_s = r)$ corresponds to building the paths $P_{i_1}, P_{i_2}, \dots, P_{i_s}$ in this order, and the weight of the path in $\bar{D}$ equals the objective value for the disjoint paths instance. Note that $q_{i_j} c_{i_{j-1}}$ for $j = 1, \dots, s$ captures the cost of building path P_{i_j} and $(\sum_{i \in \{1,\dots,r\} \setminus \{i_1, i_2, \dots, i_s\}} q_i) c_r$ captures the cost incurred after the ultimate shortest path has been built. Thus, we have proved the following result.

Proposition 1. *An instance associated with a disjoint paths graph with $r + 1$ paths and characterized by vectors $(q_1, \dots, q_r)$ and $(c_0, \dots, c_r)$ can be solved in time $O(r^2)$.*

In the following proposition, we completely characterize the instances associated with disjoint paths graphs where all potential arcs need to be build.

Proposition 2. *For an instance associated with a disjoint paths graph with $r+1$ paths and characterized by vectors $(q_1, \dots, q_r)$ and $(c_0, \dots, c_r)$ all potential arcs have to be built in any optimal solution if and only if*

$$(q_{i+1} - q_i) c_{i-1} + q_i c_r > q_{i+1} c_i \tag{1}$$

for all $i \in \{1, 2, \dots, r-1\}$.

Proof. Suppose the inequality (1) holds for all $i \in \{1, \dots, r\}$ and there is an optimal solution with $I \subseteq \{1, \dots, r\}$ being the set of indices i such that P_i is built in this solution. Writing $I = \{i_1 < i_2 < \cdots < i_s = r\}$ and putting $i_0 = 0$, the optimal objective value is

$$f(I) = \sum_{k=1}^{s} q_{i_k} c_{i_{k-1}} + c_r \left(\sum_{i \in \{1,\dots,r\} \setminus I} q_i + 1 \right)$$

Suppose there is some $i \in \{1, \dots, r-1\}$ with $i \notin I$. Then we may assume that $i = i_l - 1 > i_{l-1}$ for some $l \in \{1, \dots, s\}$ and the objective value for the index set $I' = I \cup \{i\}$ is

$$f(I') = f(I) - (q_{i+1} - q_i) c_{i_{l-1}} + q_{i+1} c_i - c_r q_i < f(I)$$

and this contradicts the optimality of I.

Conversely, suppose $I = \{1, 2, \dots, r\}$ is the index set of the paths build in the unique optimal solution with objective value $f(I)$ and that there is an $i \in \{1, \dots, r-1\}$ violating (1). The objective value for building the paths with with indices in $I' = I \setminus \{i\}$ is

$$f(I') = f(I) + (q_{i+1} - q_i) c_{i-1} - q_{i+1} c_i + q_i c_r \leqslant f(I)$$

contradicting the assumption that all paths have to be build in any optimal solution. $\square$

One possible way to satisfy the condition in Proposition 2 is to put $q_i = i$ for $i = 0, 1, 2, \ldots, r$, and to define c_r recursively by $c_r = 0$ and $c_{i-1} = (i+1)c_i + 1$ for $i = r-1, r-2, \ldots, 0$. This is illustrated in Figure 3 for $r = 5$.

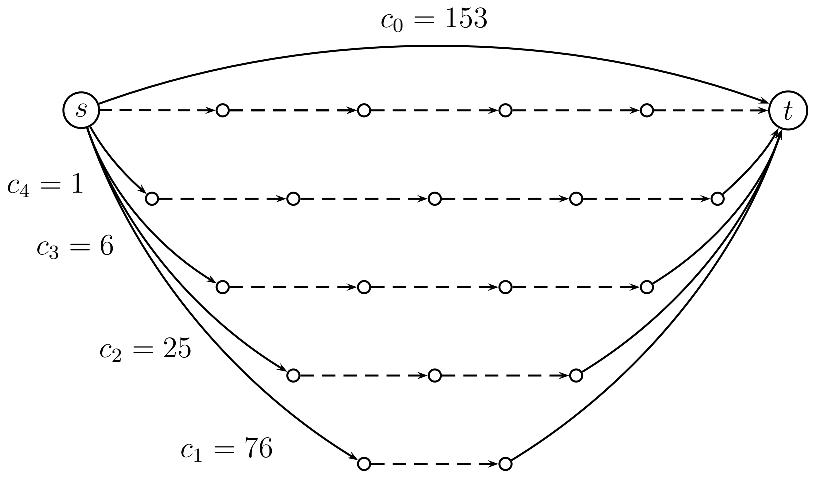

Figure 3: An instance associated with a disjoint paths graph in which all potential arcs need to be build in an optimal solution.

6 Greedy Heuristics

Two simple natural greedy heuristics are to

1. Always build arcs that lead to a shorter $s - t$ path as quickly as possible (H_1); and

2. Build an ultimate shortest path as quickly as possible (H_2).

Next, we show that the performance of both these heuristics, as well as of the heuristic where we take the best of the two solutions (H_3), can be bad. Let the sequence (c_k) be given by values $c_0 = 1$, $c_1 = 1$, $c_2 = 4$, and the recursion $c_k = 5c_{k-1} - 3c_{k-2}$ for $k > 2$.

Lemma 1. *We have* $c_k = \sum_{i=1}^{k-1} (i+2)c_{k-i} + c_0$ *for all* k.

Proof. Induction on k. For $k \leqslant 2$ it's easy to check, and for $k > 2$ we get

$$\begin{aligned}
c_k = 5c_{k-1} - 3c_{k-2} &= 3c_{k-1} + 2\left(\sum_{i=1}^{k-2}(i+2)c_{k-1-i} + c_0\right) - 2c_{k-2} - \left(\sum_{i=1}^{k-3}(i+2)c_{k-2-i} + c_0\right)\\
&= 3c_{k-1} + 4c_{k-2} + \left(\sum_{i=2}^{k-2} 2(i+2)c_{k-1-i} - \sum_{i=1}^{k-3}(i+2)c_{k-2-i}\right) + c_0\\
&= 3c_{k-1} + 4c_{k-2} + \left(\sum_{i=3}^{k-1} 2(i+1)c_{k-i} - \sum_{i=3}^{k-1} ic_{k-i}\right) + c_0\\
&= \sum_{i=1}^{k-1}(i+2)c_{k-i} + c_0. \qquad \square
\end{aligned}$$

Now consider the instance given in Figure 4. We have $T = |A_p| + 1 = 2 + 3 + \cdots + (k + 1) + T/2 + 1$, hence $T = (k+1)(k+2)$. We compare the following three strategies

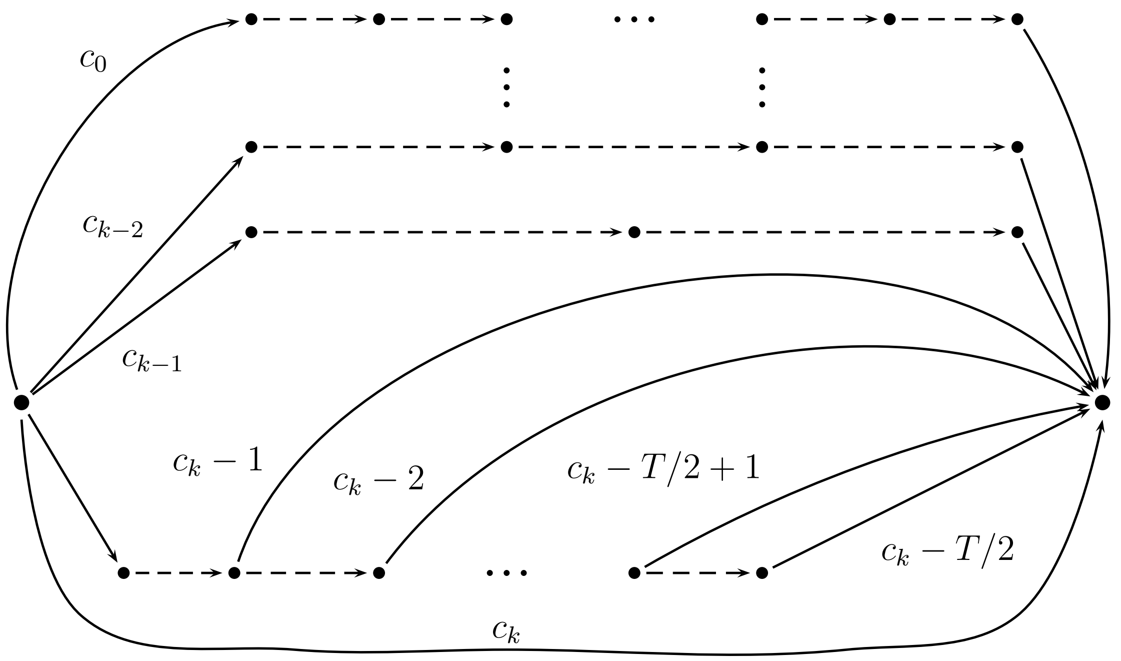

Figure 4: Instance on which natural greedy heuristics perform bad.

1. H_1, i.e. build the bottom path first, and then the remaining paths from bottom to top;
2. H_2, i.e., build only the top path; and
3. Build all paths except the bottom one.

For large enough k, we can bound the corresponding objective function values as follows.

1. For H_1 we get

$$\sum_{i=0}^{T/2}(c_k - i) + (c_k - T/2) + \sum_{i=1}^{k-1}(i+2)c_{k-i} + c_0 = \left(\frac{T}{2}+3\right)c_k - \frac{T(T+2)}{8} - \frac{T}{2} \geqslant kc_k.$$

2. For H_2 we get

$$(k+1)c_k + (T-k-1)c_0 \geqslant kc_k.$$

3. For the third strategy we get

$$2c_k + \sum_{i=1}^{k-1}(i+2)c_{k-i} + c_0 + \frac{T}{2}c_0 = 3c_k + \frac{T}{2}c_0 \leqslant 4c_k.$$

This implies that the two simple natural greedy heuristics produce solutions with values that are at least $\frac{k}{4}$ times the optimal value for this instance. The above shows that the two simple natural greedy heuristics do not provide constant-factor approximation algorithms.

7 Approximation Algorithm

For $k = 0, 1, \ldots, T$, let d_k deonte the cost of the shortest path from source to sink using at most k arcs from A_p. The value of d_k can be computed using the modified Bellman-Ford algorithm shown in Algorithm 1, where $d_k(u,v)$ is the cost of a shortest $u - v$ path using at most k arcs from A_p. Set $\Delta' = d_T$ to be the cost of an ultimate shortest path, and $\Delta = d_0 - d_T$ be the cost of the initial shortest path, less the cost of an ultimate shortest path.

Algorithm 1 k-costs from the source.

```
for $k = 0, \ldots, T$ do
    if $k = 0$ then
        $d_k(s,s) \leftarrow 0$
        $d_k(s,v) \leftarrow \infty$ for all $v \neq s$
    else
        $d_k(s,v) \leftarrow d_{k-1}(s,v)$ for all $v \in N$
        for $a = (v,w) \in A_p$ do
            $d_k(s,w) \leftarrow \min\{d_k(s,w),\ d_k(s,v) + c_a\}$
    for $i = 1, \ldots, n$ do
        for $a = (v,w) \in A_e$
            $d_k(s,w) \leftarrow \min\{d_k(s,w),\ d_k(s,v) + c_a\}$
return $d_k(s,t)$ as $d_k$ for all $k$
```

Denote the value of κ_i as the minimum number of arcs that need to be constructed to obtain an $s-t$ path with cost less than $\Delta' + \frac{\Delta}{2^i}$. For convenience, we also introduce parameters $k_0 = \kappa_0$ and $k_i = \kappa_i - \kappa_{i-1}$ for $i \geqslant 1$, which gives $\kappa_i = k_0 + k_1 + \cdots + k_i$. Let K be the minimal index k with $d_k = d_T$, and r be the smallest integer with $d_{K-1} \geqslant \Delta' + \frac{\Delta}{2^r}$. This gives $\kappa_r = k_0 + k_1 + \cdots + k_r = K$.

Algorithm 2 Approximately solving the incremental network design problem with shortest paths.

```
Compute the values $d_k$ using Algorithm 1
$K \leftarrow \min\{k \ : \ d_k = d_T\}$
$I = \{i$ such that $k_i > 0\}$
**for** $i = 0, \dots, r$ **do**
    $\kappa_i \leftarrow \min\{k \ : \ d_k < d_T + (d_0 - d_T)/2^i\}$
$j \leftarrow 0$ {number of added arcs}
$\overline{A} \leftarrow \varnothing$ {set of added arcs}
**for** $i \in I$ **do**
    Determine an $s$-$t$ path $P$ of cost $d_{\kappa_i}$ using at most $\kappa_i$ arcs from $A_p$
    **for** $a \in (P \cap A_p) \setminus \overline{A}$ **do**
        $j \leftarrow j + 1$
        $a_j \leftarrow a$
        $\overline{A} \leftarrow \overline{A} \cup \{a\}$
Build arcs $a_1, a_2, \dots, a_j$ in that order.
```

The approach that is taken in Algorithm 2, is to firstly determine the smallest number of arcs that can be constructed such that cost of a shortest $s-t$ path is less than $\frac{\Delta}{2^i} + \Delta'$ for each $i = 0, 1, \dots, r$. This defines the values of κ_i and k_i. If $k_i > 0$, then $\frac{\Delta}{2^i} + \Delta'$ provides a lower bound on the cost incurred while constructing a path that has k_i more potential arcs than the previously used shortest $s-t$ path. Furthermore, $\frac{\Delta}{2^{i-1}} + \Delta'$ provides an upper bound on the cost incurred while constructing a path that has k_i more potential arcs then the previously used shortest $s-t$ path. The number of arcs that must be constructed to complete this path depends on the number of arcs previously constructed on the $s-t$ path that is utilized. The lower bound on the number of arcs that need to be constructed is k_i, and the upper bound is given by κ_i. The upper bound represents the case where the path is disjoint, and requires all potential arcs in the path to be constructed. In Figure 5, these two cases refer to the top branch and the bottom $r+1$ branches respectively. We can combine the bounds on both the number of arcs constructed and their cost to obtain upper and lower bounds on the solution returned by Algorithm 2.

Lower bound A cost of at least Δ' is incurred every time period so the baseline cost is $T\Delta'$. As a lower bound, $\frac{\Delta}{2^i}$ is incurred for k_i periods, while constructing the path that has k_i more potential arcs than the previously used shortest $s-t$ path. The total cost of construction is bounded below by $L = T\Delta' + \sum_{i \in I} k_i \frac{\Delta}{2^i}$.

Upper bound Again, the baseline cost is $T\Delta'$. As an upper bound, $\frac{\Delta}{2^{i-1}}$ is incurred for κ_i periods, while constructing the path that has k_i more potential arcs than the previously used shortest $s{-}t$ path. The total cost of construction is bounded above by $U = T\Delta' + \sum_{i\in I} \kappa_i \frac{\Delta}{2^{i-1}}$.

Combining these two bounds, we can bound the approximation ratio of Algorithm 2.

Proposition 3. *Algorithm 2 is a 4-approximation for the incremental network design problem with shortest paths.*

Proof. We can now demonstrate that Algorithm 2 is a 4-approximation, by showing that $U \leqslant 4L$:

$$U = T\Delta' + \sum_{i\in I} \kappa_i \frac{\Delta}{2^{i-1}} = T\Delta' + \sum_{i\in I} \left[\left(\sum_{j=0}^{i} k_j \right) \frac{\Delta}{2^{i-1}} \right]$$

$$= T\Delta' + \sum_{j=0}^{r} \left[\left(\sum_{i\in I,\, i\geq j} \frac{\Delta}{2^{i-1}} \right) k_j \right] \leqslant T\Delta' + \sum_{j=0}^{r} \frac{\Delta}{2^{j-2}} k_j \leqslant 4T\Delta' + 4\sum_{j=0}^{r} \frac{\Delta}{2^j} k_j = 4L. \quad \square$$

The worst case behaviour for this algorithm can be observed in Figure 5. The optimal solution is to build the upper path which gives an objective value of

$$2 \cdot 2^r + k\left(2^{r-1} + 2^{r-2} + \cdots + 2^1 + 2^0\right) = (k+2)2^r - k.$$

But Algorithm 2 builds the other paths from top to bottom and this yields

$$2^r + \sum_{i=1}^{r+1} (ik+1)(2^{r-i+1} - 1) = (4k+3)2^r - k\frac{(r+2)(r+3)}{2} - k - r - 2.$$

Now the claim follows since for sufficiently large $k = r$,

$$(4k+3)2^r - k\frac{(r+2)(r+3)}{2} - k - r - 2 > (4-\varepsilon)\left[(k+2)2^r - k\right].$$

8 Conclusions and Extensions

In this paper, we have introduced a class of incremental network design problems and studied one member of the class, the incremental network design problem with shortest paths, in more detail. We have established that the problem is NP-hard, but have also identified a polynomially solvable special case. Natural greedy heuristics have been analyzed and a 4-approximation algorithm has been developed.

A natural extension to the variant of the incremental network design problem with shortest paths considered in this paper is obtained by introducing arc construction costs and construction budgets. In that case, we are also given for each arc $a \in A_p$ a construction cost $\bar{c}_a$ and for each period t a budget B_t and the option to expand the network in each

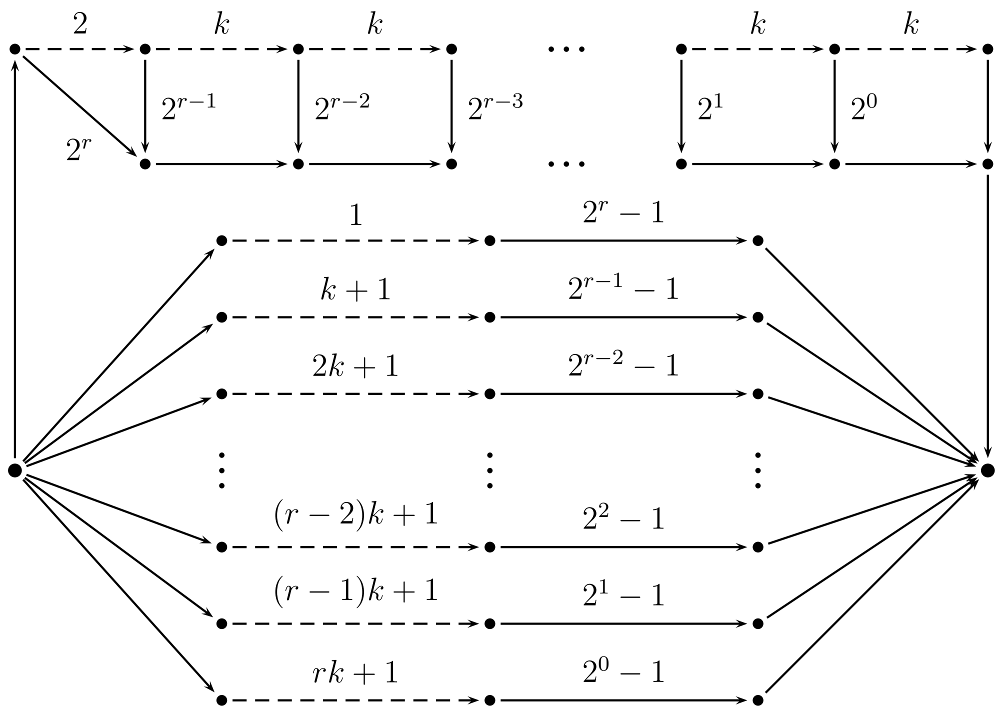

Figure 5: Instance on which Algorithm 2 performs badly. A dashed arc with label i indicates a path of i potential arcs, and a solid arc with label i indicates an existing arc with cost i. All potential arcs and the unlabeled solid arcs have zero cost.

period by building potential arcs $a \in A_p$ subject to the constraint that the construction costs do not exceed the budget. As before, the objective is to minimize the total cost over the planning period, which now includes construction costs and shortest $s-t$ path costs.

Another natural extension is to consider multiple commodities, where each commodity i requires one unit of flow to be send from source s_i to sink t_i. This incremental network design problem with multi-commodity flows can be uncapacitated, i.e., there is no limit on the flow on an arc, or capacitated, where the flow on arc has to be less than or equal to a given capacity. When the capacity is equal to one for all arcs, the subproblem in each period reduces to finding arc-disjoint paths.

Rather than considering network expansion, we can consider network maintenance. A class of multi-period network maintenance problems can be defined in a similar way to the class of incremental network design problems. In this setting, arcs are required to undergo maintenance, either once or with a certain frequency, and when an arc undergoes maintenance it is unavailable and cannot be used in the network optimization problem that has to be solved each period. This class of multi-period network maintenance problems also offers a rich environment for further research.

Finally, it is not difficult to formulate incremental network design problems and multi-period network maintenance problems as integer programs. We are currently exploring whether integer programming formulations of the incremental network design problem with shortest paths and the incremental network design problem with multi-commodity flows are computationally tractable and whether valid or facet inducing inequalities can be derived from the multi-period structure.

References

[1] A.B. Arabani and R.Z. Farahani. “Facility location dynamics: An overview of classifications and applications”. In: *Computers & Industrial Engineering* (2011), pp. 408–420. DOI: 10.1016/j.cie.2011.09.018.

[2] G. Gallo. “Reoptimization procedures in shortest path problem”. In: *Decisions in Economics and Finance* 3.1 (1980), pp. 3–13.

[3] M.X. Goemans, A.V. Goldberg, S. Plotkin, D.B. Shmoys, E. Tardos and D.P. Williamson. “Improved approximation algorithms for network design problems”. In: *Proc. 5th ACM-SIAM symposium on Discrete algorithms SODA 1994*. Society for Industrial and Applied Mathematics. 1994, pp. 223–232.

[4] A. Gupta, A. Kumar and T. Roughgarden. “Simpler and better approximation algorithms for network design”. In: *Proc. 35th ACM symposium on Theory of computing STOC 2003*. ACM. 2003, pp. 365–372. DOI: 10.1145/780542.780597.

[5] H. Kerivin and A.R. Mahjoub. “Design of survivable networks: A survey”. In: *Networks* 46.1 (2005), pp. 1–21. DOI: 10.1002/net.20072.

[6] B.J. Kim, W. Kim and B.H. Song. “Sequencing and scheduling highway network expansion using a discrete network design model”. In: *The Annals of Regional Science* 42.3 (2008), pp. 621–642. DOI: 10.1007/s00168-007-0170-2.

[7] G. Lin, C. Nagarajan, R. Rajaraman and D.P. Williamson. “A general approach for incremental approximation and hierarchical clustering”. In: *SIAM Journal on Computing* 39.8 (2010), pp. 3633–3669. DOI: 10.1137/070698257.

[8] T.L. Magnanti and R.T. Wong. “Network design and transportation planning: Models and algorithms”. In: *Transportation Science* 18.1 (1984), pp. 1–55. DOI: 10.1287/trsc.18.1.1.

[9] R.R. Mettu and C.G. Plaxton. “The online median problem”. In: *SIAM Journal on Computing* 32.3 (2003), pp. 816–832. DOI: 10.1137/S0097539701383443.

[10] S. Pallottino and M.G. Scutellà. “A new algorithm for reoptimizing shortest paths when the arc costs change”. In: *Operations Research Letters* 31.2 (2003), pp. 149–160. DOI: 10.1016/S0167-6377(02)00192-X.

[11] S.V. Ukkusuri and G. Patil. “Multi-period transportation network design under demand uncertainty”. In: *Transportation Research Part B: Methodological* 43.6 (2009), pp. 625–642. DOI: 10.1016/j.trb.2009.01.004.