Algorithmica
DOI 10.1007/s00453-008-9252-5

Multiline Addressing by Network Flow

Friedrich Eisenbrand · Andreas Karrenbauer · Martin Skutella · Chihao Xu

Received: 29 November 2006 / Accepted: 28 October 2008

Abstract We consider an optimization problem arising in the design of controllers for OLED displays. Our objective is to minimize amplitude of the electrical current through the diodes which has a direct impact on the lifetime of such a display. Modeling the problem in mathematical terms yields a class of network flow problems where we group the arcs and pay in each group only for the arc carrying the maximum flow. We develop (fully) combinatorial approximation heuristics suitable for being implemented in the hardware of a control device that drives an OLED display.

Keywords Combinatorial optimization · Network design · OLED · Algorithm engineering · Matrix decomposition

1 Introduction

Organic Light Emitting Diode (OLED) displays are considered as the displays of the future. The image and video displayed is brilliant, has a very high contrast and

M. Skutella supported by DFG Research Center MATHEON in Berlin.

F. Eisenbrand
EPFL, 1015 Lausanne, Switzerland
e-mail: friedrich.eisenbrand@epfl.ch

A. Karrenbauer (✉)
Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany
e-mail: karrenba@mpi-inf.mpg.de

M. Skutella
TU Berlin, Institut für Mathematik, Str. des 17, Juni 136, 10623 Berlin, Germany
e-mail: skutella@math.tu-berlin.de

C. Xu
Lehrstuhl für Mikroelektronik, Universität des Saarlandes, Saarbrücken, Germany
e-mail: chihao.xu@lme.uni-saarland.de

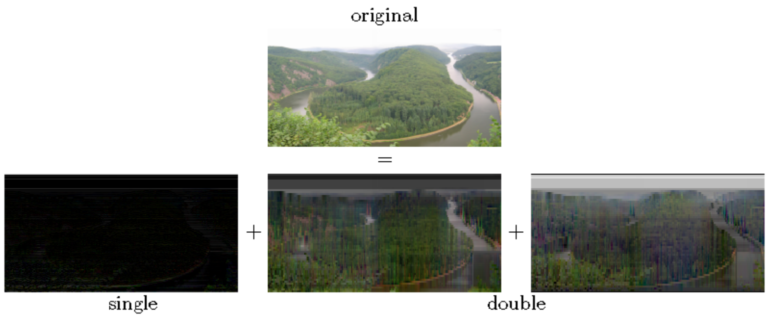

Fig. 1 Decomposition of an image with $k=2$ such that every two rows of the double parts have the same content and the sum of all row maxima is minimized

a viewing angle of nearly 180 degrees. It reacts within 10 microseconds which is much faster than the eye can catch and is therefore perfect for video applications. The display is flexible and above all can be produced at low cost. One major reason why there are only small-size displays on the market is the insufficient lifetime of state of the art OLED displays.

What causes this short lifetime? Briefly, this is because of the high electrical currents through the diodes that occur with the traditional addressing techniques. Though it seems that every pixel shines continuously with a certain brightness, the images are displayed row-after-row. This works at a sufficiently high frame rate since the perception of the eye is the average intensity emitted by each diode. The problem is that this row-by-row activation scheme causes long idle times of the diodes and extreme stress when they are activated.

To overcome this problem one considers now to activate two, or more consecutive rows simultaneously [1]. For passive matrix displays rows can only be simultaneously displayed if their content is equal.

Therefore the goal is to decompose an image into several subframes, the overlay (addition) of which is equal to the original image. Fig. 1 shows such a decomposition. The first image (single) is traditionally displayed row-after-row. In the other two images (double) every two rows have the same content so that one can display these rows simultaneously. As one can see in Fig. 1, the images at the bottom are much darker than the original one. The decomposition should be in such a way that the amplitudes of the electrical current which are needed to display the picture are as small as possible.

In this paper we develop an algorithm to tackle this optimization problem. Our objective is to come up with a combinatorial approximation algorithm that will be implemented in hardware to actually drive such an OLED display. This imposes some restrictions on the methods and techniques we shall use. First of all, such an algorithm has to compute a feasible solution in realtime, i.e. below the perception of a human eye. Moreover, it should be implemented on a chip of low cost meaning that we are not able to e.g. use a general purpose LP solver or a general purpose CPU with an IEEE floating point unit. Therefore, we look for algorithms that are sufficiently simple and easy to implement. Moreover, the algorithms should not suffer from numerical instabilities. Also exact rational arithmetic is not an option for such a realtime application. We rather want to use only fixed precision number types, i.e. integers of fixed size. To meet these economic constraints, we aim at a fully combinatorial algorithm using only addition, subtraction, and comparison.

Contributions of this Paper

First, we model this optimization problem as a certain network design problem. We present a concise formulation where the arcs are partitioned into groups and only the arc with the highest flow in each group is charged and one as a covering (integer) linear program with an exponential number of constraints. For the latter case, we develop linear time separation routines which are required to solve the LP relaxation with the ellipsoid method [2, 3] or with a cutting-plane approach [4] or to solve it approximately [5] with known frameworks. We then propose an efficient fully combinatorial heuristic which is based on the separation of these constraints and satisfies the above requirements. Our implementation shows that this heuristic is very close to the optimum.

Related Work

Subsequently, the models and algorithms presented in this paper have served as the basis for further developments [6, 7]. In particular, the special case of black/white images is considered in [7]. For that case, a polynomial-time algorithm and tight competitive ratios for the natural online problem with fixed look-ahead are presented, which helped to improve the heuristics for the general case introduced in this paper. Other approaches based on *Non-negative Matrix Factorization* [8, 9] have been outlined in [10] and [11]. Network flow techniques have been applied to other matrix decomposition problems occurring in cancer therapy [12].

2 Technical Background

To understand the objective of our optimization problem, we need to explain in an informal way how OLED displays work. An OLED display has a matrix structure with n rows and m columns. At any crossover between a row and a column there is a vertical diode which works as a pixel, see Fig. 2.

The image itself is given as an integral $n \times m$ matrix $(r_{ij}) \in \{0, \ldots, \varrho\}^{n\times m}$ representing its RGB values. The number ϱ determines the *color depth*, e.g. $\varrho = 255$ for 16.7 million colors. Since there are only $n + m$ contacts available, a specific addressing technique is needed. We explain one technique (*pulse width modulation*) in a simplified way in the following. Consider the contacts for the rows and columns as switches. If the switch of row i and column j is closed, the pixel (i, j) shines with a brightness or *intensity* I which is common to all row-column pairs. An image has to be displayed within a certain time frame T_f. The value r_{ij} determines that within the time-frame T_f, the switches i and j have to be simultaneously closed for the time

$$t_{ij} = r_{ij} \cdot \frac{T_f}{I} \tag{1}$$

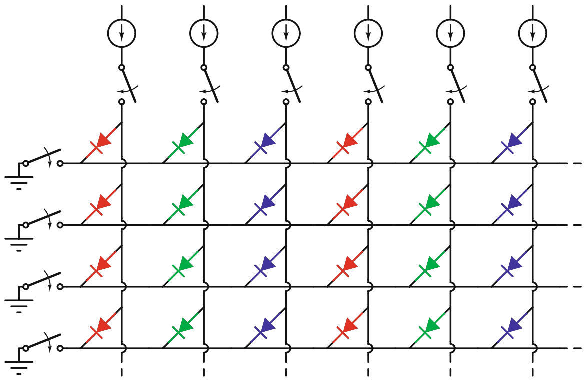

Fig. 2 Scheme of the electrical circuit of a display

$$\begin{bmatrix} 109 & 238 & 28 \\ 112 & 237 & 28 \\ 150 & 234 & 25 \\ 189 & 232 & 22 \\ 227 & 229 & 19 \end{bmatrix} = \begin{bmatrix} 0 & 82 & 25 \\ 0 & 82 & 25 \\ 0 & 41 & 22 \\ 0 & 41 & 22 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 112 & 155 & 3 \\ 112 & 155 & 3 \\ 189 & 191 & 0 \\ 189 & 191 & 0 \end{bmatrix} + \begin{bmatrix} 109 & 156 & 3 \\ 0 & 0 & 0 \\ 38 & 38 & 0 \\ 0 & 0 & 0 \\ 38 & 38 & 19 \end{bmatrix}$$

Fig. 3 An example decomposition

in total. At a sufficient high frame rate e.g. 50 Hz, the perception by the eye is the average value of the light.

Currently, drivers for OLED displays display the image in a row-by-row fashion. This means that the switches for the rows are activated one after the other. While row i is active, column j has to be active for the time t_{ij} so row i is finished after $\max\{t_{ij} \mid j = 1, \dots, m\}$ time units. The time which is required to display the image is consequently $T(I) = \sum_{i=1}^{n} \max\{t_{ij} \mid j = 1, \dots, m\}$. The intensity I has to be high enough such that $T(I) \leq T_f$ holds.

Equation (1) shows that the time t_{ij} is inversely proportional to the value I. The aforementioned short lifetime of today's OLED displays is mainly due to the high value of I which is necessary to display images with a sufficient frame rate. This means that the peak energy which has to be emitted by the diodes of the display is very high while on the other hand, the diodes stay idle most of the time, see [13]. The high amplitudes of electrical current which alternate with long idle times put the diodes under a lot of stress, which results in a short lifetime.

In this paper, we aim to overcome this problem by a different driving mechanism. The simple but crucial observation is that, if two rows have the same content, we could drive them simultaneously and thereby we would save half of the time necessary for the two rows. Therefore the value of I could be reduced until $T(I) = T_f$.

Consider the example in Fig. 3. Suppose here that initially $T_f/I = 1$. If the image is displayed row-by-row, then the minimum time which is needed to display the image is $238 + 237 + 234 + 232 + 229 = 1170$ time units. But we can do better by a suitable decomposition of the image into three matrices. In the first one every even row is equal to its odd predecessor and in the second one every odd row is equal to its even predecessor with zero-rows, where there is no predecessor available respectively. The remainder is put into an offset matrix that is driven in the traditional way. By driving the equal rows simultaneously, we require only $82 + 41 + 155 + 191 + 156 + 38 +$

$$F^{(2)} = \begin{pmatrix} 0 & 82 & 25 \\ 112 & 155 & 3 \\ 0 & 41 & 22 \\ 189 & 191 & 0 \end{pmatrix}, \qquad F^{(1)} = \begin{pmatrix} 109 & 156 & 3 \\ 0 & 0 & 0 \\ 38 & 38 & 0 \\ 0 & 0 & 0 \\ 38 & 38 & 19 \end{pmatrix}$$

Fig. 4 Compact representation of the example decomposition of Fig. 3

$38 = 701$ time units. This means that we could reduce I and therefore the amplitude of the electrical current by roughly 40%.

We could save even more by driving $3, 4, 5, \ldots$ rows simultaneously. Of course there is a saturation somewhere, unless the image is totally homogeneous. On our benchmark pictures, we observed that it is not worth to consider more than 6 simultaneously driven rows.

Since we only need to store the contents of the common part once, a decomposition where up to k consecutive rows are combined can be stored in k matrices $F^{(1)}, \ldots, F^{(k)}$ not larger than the original image (see Fig. 4).

3 The Network Model

For the sake of simplicity, we first consider the case in which two consecutive rows can be activated simultaneously. Let $R = (r_{ij}) \in \{0, \ldots, \varrho\}^{n \times m}$ be the matrix representing the picture. To decompose R we need to find matrices $F^{(1)} = (f_{ij}^{(1)})$ and $F^{(2)} = (f_{ij}^{(2)})$ where $F^{(1)}$ represents the offset part and $F^{(2)}$ the common part. More precisely, the i-th row of matrix $F^{(2)}$ represents the common part of rows i and $i+1$. In order to get a valid decomposition of R, the matrices $F^{(1)}$ and $F^{(2)}$ must fulfill the constraint $f_{ij}^{(1)} + f_{i-1,j}^{(2)} + f_{ij}^{(2)} = r_{ij}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, where we now and in the following use the convention to simply omit terms with indices running out of bounds. The fixed boundary conditions in our application require $f_{1j}^{(1)} + f_{1j}^{(2)} = r_{1j}$ and $f_{nj}^{(1)} + f_{n-1,j}^{(2)} = r_{nj}$ for the first and the last row, respectively. Notice that the matrix $F^{(2)}$ has only $n-1$ rows. We sometimes assume that there is in addition a row numbered n containing only zeros.

Since we cannot produce "negative" light we require also non-negativity of the variables $f_{ij}^{(\ell)} \geq 0$ where we now and in the following use the superscript $\ell = 1, 2$ for statements that hold for both matrices. The goal is to find an integral decomposition that minimizes

$$\sum_{i=1}^{n} \max\{f_{ij}^{(1)} : 1 \leq j \leq m\} + \sum_{i=1}^{n-1} \max\{f_{ij}^{(2)} : 1 \leq j \leq m\}$$

This problem can be formulated as an integer linear program by replacing the objective by $\sum_{i=1}^{n} u_i^{(1)} + \sum_{i=1}^{n-1} u_i^{(2)}$ and by adding the constraints $f_{ij}^{(\ell)} \le u_i^{(\ell)}$. This yields

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} u_i^{(1)} + \sum_{i=1}^{n-1} u_i^{(2)} \\ \text{s.t.} \quad & f_{ij}^{(1)} + f_{i-1,j}^{(2)} + f_{ij}^{(2)} = r_{ij} \quad \text{for all } i, j \\ & f_{ij}^{(\ell)} \le u_i^{(\ell)} \quad \text{for all } i, j, \ell \\ & f_{ij}^{(\ell)} \in \mathbb{Z}_{\ge 0} \quad \text{for all } i, j, \ell \end{aligned} \tag{2}$$

The corresponding linear programming relaxation is not integral in general as an example with

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

shows. The optimal solution is obtained by setting each $u_i^{(\ell)} = \frac{1}{2}$ yielding an objective value of $\frac{5}{2}$. However, requiring the f-variables to be integral, yields that in every optimal solution the u-variables are integral, too. Conversely, if we have an optimal integral assignment to the u-variables, then every feasible basic solution of the f-variables is also integral. To see this property, we shall investigate the constraints a bit deeper.

Observe that the equality constraints of (2) can be represented by a blockdiagonal 0/1-matrix with one block for each $j = 1, \dots, m$. We thus have m blocks with identical structure of the form illustrated on the left of expression (3)

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$
$$\rightsquigarrow \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix} \tag{3}$$

Notice that the columns have the consecutive ones property. Hence, there is a natural transformation by row-operations (preserving the solution space like in Gaussian elimination) into a node-arc incidence matrix, see, e.g. [14]. In other words, we add a zero dummy row at the end and subtract from each row its predecessor and obtain in each column exactly one 1 and one -1 as depicted on the right in (3). Recall that

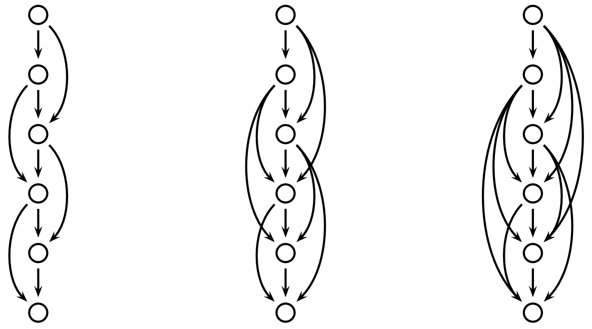
Fig. 5 Examples for a display graph with $n = 5$, $k = 2, 3, 4$

this matrix is just the block for one $j \in \{1, \dots, m\}$. The resulting graph $G = (V, A)$, which is common to all j, is called the *display graph*; see Fig. 5 for an illustration. The display graph has node set $V = \{1, \dots, n+1\}$ and arcs $(i, i+1)$ for $i = 1, \dots, n$ and $(i, i+2)$ for $i = 1, \dots, n-1$.

In the forthcoming, we refer to the u variables as *capacities*. The new right-hand sides of the equality constraints which we call *demands* are given by $d_j(i) = r_{ij} - r_{i-1,j}$. The generalization when we drive $k \geq 2$ consecutive lines together is straightforward and depicted in Fig. 5.

The optimization problem can now be understood as follows. Assign integral capacities $u : A \to \mathbb{Z}_{\geq 0}$ to the arcs of the display graph at minimum cost (each unit of capacity costs 1 for each arc) such that the network flow problem defined by the display graph together with the demands $d_j : V \to \mathbb{Z}$ has a feasible solution for each $j = 1, \dots, m$. Let $\delta^{out}(X)$ denote the outgoing arcs of node set $X \subset V$. We use the standard notation $u(\delta^{out}(X))$ and $d(X)$ for the sums over the corresponding capacities and demands respectively. It follows now from MaxFlow/MinCut duality that our optimization problem can be rewritten as (cf. Chap. 11 in [15])

$$\begin{aligned} \min \quad & \sum_{a \in A} u(a) \\ \text{s.t.} \quad & u(\delta^{out}(X)) \geq d_j(X) \quad \text{for all } X \subset V, \text{for all } j \\ & u \in \mathbb{Z}_{\geq 0}{}^{|A|} \end{aligned} \tag{4}$$

For general graphs, this problem contains DirectedSteinerTree as a special case. The complexity of the optimization problem (4) restricted to display graphs remains open. For $k = 2$ and monochrome images, a polynomial-time algorithm is presented in [7]. But regardless whether the problem is solvable in polynomial time for graphs with our structure or it remains NP-complete, we shall focus on fast approximation algorithms. To this end, we will discuss the *separation problem* for the linear program in the next section. Among other reasons, it is necessary since the number of cuts in a graph is exponential in its size. Moreover, fast separation permits fast heuristics in practice.

4 Efficient Algorithms for the Separation Problem

Finding violated inequalities for a given assignment to the variables is a key idea for solving linear programs. It is well known [16] that the *linear optimization* problem over a given polyhedron is polynomial time equivalent to the *separation* problem for this polyhedron. Also our heuristics (see Sect. 5) rely on the solution of the separation problem for inequalities (4) and updating the solution iteratively until we have found a feasible solution. In our setting, the separation problem is the following:

> Given a capacity assignment $u \geq 0$ of the display graph, determine whether u is feasible and if not, compute a subset $X \subset V$ of the nodes in the display graph such that there is a $j \in \{1, \dots, m\}$ with $u(\delta^{out}(X)) < d_j(X)$.

4.1 Separation by MaxFlow/MinCut

Observe that we can consider the separation problem for each column $j \in \{1, \dots, m\}$ independently. For a given j this can be done with a MaxFlow computation as follows. We construct a network G_j by adding new vertices s and t to the display graph. The capacities of the arcs in the display graph are given by u. There is an arc (s, i) if $d_j(i) > 0$ with capacity $d_j(i)$ and there is an arc (i, t) if $d_j(i) < 0$ with capacity $-d_j(i)$.

If the maximum s, t-flow in this network is less than $\delta_j = \sum_{d_j(i)>0} d_j(i)$, then the vertices of a corresponding MinCut which belong to the display graph comprise a set $X \subset V$ with $u(\delta^{out}(X)) < d_j(X)$. If the value of a MaxFlow in G_j is equal to δ_j for all $j = 1, \dots, m$, then the capacity assignment u is feasible.

In our implementation we iteratively increase the capacity of one arc by some integral constant c. We use a *Blocking Flow* approach in our implementation. Thereby, we can benefit from an efficient treatment of capacity adjacent problems, see, e.g. [14]. Note that in this case there only exist at most c augmenting paths and moreover these paths have to take the arc whose capacity has increased. Since each of these paths can be found in linear time by depth-first search, the update takes only $O(n)$ for one column. In practice, the performance is even better since on average the paths are rather short. Moreover, we have to consider only the columns that have this arc in their current MinCut since otherwise the additional capacity would not have any effect.

Theorem 1 *Suppose that one is given two adjacent capacity assignments* $0 \leq u \leq \bar{u}$, *i.e.* u *and* $\bar{u}$ *differ in exactly one arc by a constant* c, *and a maximum flow* f *w.r.t.* u, *then the separation problem w.r.t.* $\bar{u}$ *can be solved in linear time. More precisely one can compute a maximum flow* $\bar{f} \leq \bar{u}$ *and a minimum cut* $\bar{X} \subset V$ *with respect to* $\bar{u}$ *in linear time.*

One issue of this approach is that we maintain the flow variables f. On ordinary PC hardware this is an advantage since we have to compute the decomposition of the image which is represented by the flow variables at the end anyways. But we need roughly k times more memory than the input size which makes the implementation on a chip more expensive. We address this issue in the next section.

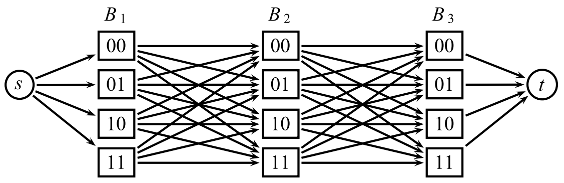

Fig. 6 Solving the separation problem by a shortest path computation in a DAG (here $k = 2$)

4.2 A Linear Time Algorithm for Fixed k

The number of simultaneously activated lines k is relatively small, in fact up to 6. We now show how to solve the separation problem for fixed k in linear time. The key feature of the display graph which allows such an efficient algorithm is the following. The arcs are of the form (i, i'), where $i' \leq i + k$. It is sufficient to find for each $j = 1, \dots, m$ a subset $X \subset V$ such that $u(\delta^{out}(X)) - d_j(X)$ is as small as possible. If one of these values is negative, we have found a violated inequality. Otherwise, all constraints are fulfilled.

In order to find such a subset X, we partition the vertices $V = \{1, \dots, n + 1\}$ into consecutive blocks $B_1, \dots, B_{\lceil (n+1)/k \rceil}$ of size k. That is, $B_i := \{(i - 1) \cdot k + 1, (i - 1) \cdot k + 2, \dots, i \cdot k\}$ for $i = 1, \dots, \lfloor (n + 1)/k \rfloor$ and $B_{\lceil (n+1)/k \rceil} := \{\lfloor (n + 1)/k \rfloor k + 1, \dots, n + 1\}$ if k does not divide $n + 1$.

We now consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where the vertex set $\mathcal{V}$ contains all subsets of the sets B_i and there is an arc (S_1, S_2) if there exists an index i such that $S_1 \subseteq B_i$ and $S_2 \subseteq B_{i+1}$. Furthermore $\mathcal{G}$ has an additional vertex s and an additional vertex t together with arcs (s, S) for each $S \subseteq B_1$ and (S, t) for each $S \subseteq B_{\lceil (n+1)/k \rceil}$. An example is shown in Fig. 6. A path from s to t specifies a subset $X \subseteq V$ of the display graph in a natural way (and also vice versa): Given a path from s to t, take the union of the subsets represented by the inner vertices of the path.

It remains to define arc weights in $\mathcal{G}$ such that the weight of such a path is exactly $u(\delta^{out}(X)) - d_j(X)$. For this consider an arc (S_1, S_2) with head and tail different from s and t respectively. The weight of this arc is defined as $-\sum_{i \in S_2} d_j(i) + u(S_1 : S_2)$, where $S_1 : S_2$ denotes the subset of arcs of the display graph that have their tails in S_1 and their heads are neither in S_1 nor in S_2. The weight of an arc (s, S) is defined as $-\sum_{i \in S} d_j(i)$ and the weight of an arc (S, t) is $u(\delta^{out}(S))$.

In this way, the weight of a path from s to t in $\mathcal{G}$ is equal to the value $u(\delta^{out}(X)) - d_j(X)$, where X is the set which is represented by the path. Thus, the separation problem can be reduced to m shortest path problems in graph $\mathcal{G}$ with roughly $2^k \cdot (n + 1)/k = O(n)$ vertices, if k is fixed. We have proved the following theorem.

Theorem 2 *The separation problem for the inequalities* (4) *can be solved in linear time for fixed k.*

5 Implementation

In practice we have a display of fixed size and the requirement on the running time to be below the perception of a human eye. According to this time constraint, we have to

Fig. 7 Framework for the approximation heuristics

```
Input: $d = (d_1, \ldots, d_m)$
Output: $f$
$f = 0$
$u = \text{INITIALIZE}(d)$
while($f$ is not feasible){
      $f = \text{MAXFLOW}(V, A, d, u)$
      $C = \text{MINCUT}(V, A, d, u)$
      for all($a \in A$){
            $u(a) = u(a) + \Delta u(a, C)$
      }
}
return $f$
```

design our algorithm such that the cost of the integrated circuit which implements it is minimized. Parallelization, e.g. on several columns concurrently, and reusing results of certain computations, e.g. shortest path trees of previous iterations, decrease the running time but increase the complexity and memory usage of the circuit and hence the production costs.

Since we use only fixed precision datatypes, we consider the variables to be integral all the time. A higher intermediate precision is easily achieved by scaling the constraints, i.e. the demands. Briefly speaking our heuristics work as follows. We start with an initial assignment of the capacities returned by the function INITIALIZE. We will discuss it later. Consider it to simply return the zero vector for now. Afterwards, we iterate until we find a feasible solution. In each iteration, we first solve the separation problem. Since we have to compute the lifting to the flow-variables at the end we use the blocking flow approach for this task. Depending on the outcome and on the chosen strategy, we augment one or several capacities. In Fig. 7, we describe this general framework of our heuristics with pseudocode.

In short, d denotes the demands of the nodes and thereby implicitly defines the underlying graph as we consider k to be fixed. The notion of the function Δu covers several variants of our heuristics for augmenting the capacities.

5.1 Capacity Augmentation

After having found a violated cut, the question arises which capacity variables to augment. Unlike in the framework of Garg and Könemann [5] where all capacities of the cut would be multiplied with a constant, we select only one variable to augment since we want to benefit from the capacity adjacency mentioned before.

Our strategy is to augment the most promising variable that is in any of the cuts of the different columns. We measure the potential impact of a capacity by the number of different columns, i.e. cuts, it appears in. This would be equivalent to summing up all the cuts over all columns which gives us a valid violated inequality too. The *basic greedy* approach selects the capacity having the highest potential impact, i.e. to increase the capacity with the highest coefficient in the sum of the cuts. A slightly different variant of this approach takes only the variables into account which appear in the cut of a column with the maximum deficit (referred to as *max-column greedy update*), i.e. that attains the minimum in the separation problem.

Since we maintain the integrality of the variables throughout the algorithm, we have to increase the variables at least by 1 in each iteration. With this value, the running time is then proportional to the size of the display and the difference between the achieved objective value and the one from the initial solution. By adding a greater constant or a certain fraction of the previous capacity like in the framework of Garg and Könemann, the running time would improve, however we observed that the quality of our approximation deteriorated.

5.2 Initialization

The naive way to initialize the capacities is to set them to the zero vector. The other extremal case would be to solve the LP and round up each fractional capacity with the result that we have just one iteration where we compute the flow variables. Although we are within an additive error not greater than the number of variables then, we could round down instead of rounding up and solve or approximate the remaining $0/1$ integer program with a possibly better solution for the whole problem. However, as mentioned before solving (or approximating) linear programs involving fractional numbers is not really what we want. It is natural to ask whether there is a way in between that allows us to stick with integers and can be attacked with a fully combinatorial algorithm.

Indeed, if we restrict ourselves to *easy* constraints, i.e. constraints describing an integral polyhedron, we accomplish the first goal of remaining integral. If we only consider constraints permitting fully combinatorial algorithms (e.g. flow problems or their duals) then we could also achieve the second goal.

Having solved such an easy subproblem, we can initialize our heuristics with the thereby computed solution. The next theorem identifies such an easy subset of the constraints.

Theorem 3 *The linear program* (4) *restricted to the set system*

$$\mathcal{C} = \bigcup_{i=1}^{n} \{X_i, Y_i, X_i \cap Y_i, X_i \cup Y_i : X_i = \{1, \dots, i\}, Y_i = \{i, i+2, i+3, \dots, n+1\}\}$$

has an integral optimal solution for $k = 2$ which can be found in $O(n)$ time by a fully combinatorial algorithm.

Proof We order the variables $u_i^{(\ell)}$ lexicographically with respect to $[i, l]$. The constraints corresponding to the cuts of $\mathcal{C}$ are

$$\begin{aligned} u(\delta^{out}(X_i)) &= u_{i-1}^{(2)} + u_i^{(1)} + u_i^{(2)} \\ u(\delta^{out}(X_i \cup Y_i)) &= u_{i-1}^{(2)} + u_i^{(1)} \\ u(\delta^{out}(X_i \cap Y_i)) &= u_i^{(1)} + u_i^{(2)} \\ u(\delta^{out}(Y_i)) &= u_i^{(1)} \end{aligned}$$

Hence, the constraint matrix has the consecutive ones property in every row. Since our objective is the all-ones vector, we can solve the problem by a shortest path computation in a directed acyclic graph of $O(n)$ nodes and $|\mathcal{C}| = O(n)$ edges. The transformation is performed similar to the one illustrated in (3). We add a dummy variable at the end and apply a unimodular transformation to the variables (like we did it with the rows in Sect. 3), we get $\geq$ constraints containing exactly one 1 and one -1 in each row. The costs of the edges are given by the negated right-hand side of the respective constraints. The costs of the shortest path from the first to the last node yield the negated optimal objective value subject to the subsystem $\mathcal{C}$. The corresponding capacities can be computed from the difference of the distance labels of the nodes by reversing the transformation. All these operations take $O(n)$ time in total. $\square$

Note that this consecutive ones property does not hold for $k > 2$. However, we can restore it by adding the missing capacities on the left-hand-side of the inequalities and the corresponding lower bounds, i.e.

$$u_i^{(1)} \geq d_j(Y_i), \qquad u_i^{(2)} \geq 0, \tag{5}$$

on the right-hand-side giving (weaker) valid inequalities yielding an approximate solution.

The objective value of a capacity assignment satisfying the easy constraints is a lower bound on the optimum value with respect to all constraints. However, some of the capacities of an optimum solution for all constraints may be lower than those optimizing only the easy constraints. Since our heuristics only increase variables, we may overshoot the mark for certain capacities. To be on the safe side, one could use only the constraints (5), to which we refer as *safe lower bounds* in the experimental evaluation of our approach in the next section.

6 Computational Results

As of yet the computational results are based on ordinary PC hardware. Therefore, we only present the running times of the variant that performs best on a Pentium M with 2 GHz and 2 MB L2 cache.

As a test set we used the portraits of 197 employees of the MPI. While the original images have a resolution of 180×240 RGB pixels (i.e. 720 diodes per row), we scaled them down to $n = 60, 90, 120, 150$ keeping the aspect ratio such that we have $m = 4n$ for all images. We observed that, among the different initialization and augmentation strategies, a combination of the aforementioned max-column greedy update and the initialization using the easy constraints performs best. We recommend these strategies on the basis of an implementation on PC-hardware.

On the left of Fig. 8, one can see the dependence of the running time on the input size together with the distribution of the instances. We connected the median running time to guide the eye. The fit of these medians with respect to a power function yields $t = 21\ \mu\text{s} \cdot n^{2.23}$ which is almost linear in the number of pixels. The graph on the right of Fig. 8 has on its x-axis the initial intensity I and on the y-axis the reduced intensity I' which we achieve with multiline addressing using our algorithm. The black dots

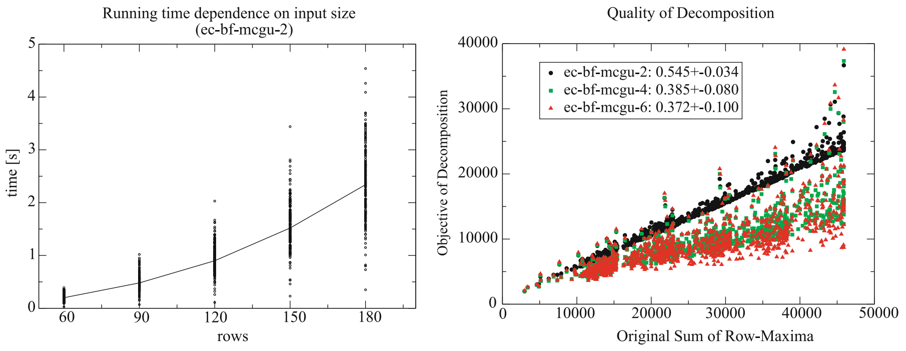

Fig. 8 We initialized the capacities by the easy constraints (ec), used a blocking flow algorithm (bf), performed the capacity augmentation by the maximum column greedy update method (mcgu), and $k = 2$, 4, 6. The *left* plot shows the dependence of the running time on n in case of $k = 2$. On the *right*, each mark represents an instance with its original value on the x-axis and the objective on the y-axis. The inset shows the average ratios and their standard deviations for $k = 2$, 4, 6 respectively

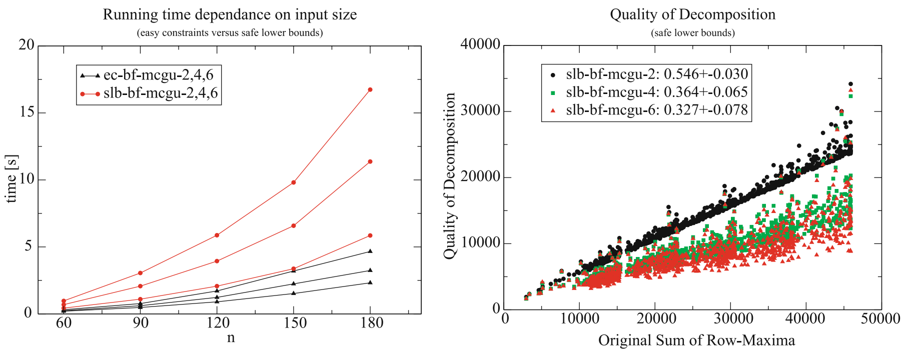

Fig. 9 Using safe lower bounds for initialization yields better ratios (*right*) but worse running times (*left*)

are the results obtained by addressing two rows simultaneously, i.e. for $k = 2$. Here one can see that the average ratio I'/I is roughly 0.545 with a standard deviation of 0.034. The green squares are the results for $k = 4$. Here the average ratio I'/I is 0.385 ± 0.065. The red triangles represent $k = 6$ with $I'/I = 0.372 \pm 0.100$. The theoretical lower bound for these ratios is $1/k$.

Not using the easy constraints but safe lower bounds yields slightly better ratios as depicted in the inset of the right graph of Fig. 9. But the running time grows faster with the number of pixels as one can see in the left graph of Fig. 9 where the three top curves belong to the median runtimes using safe lower bounds with $k = 2, 4, 6$ respectively, whereas the median runtimes using the easy constraint initialization and $k = 2, 4, 6$ yield the three lower curves.

References

1. Xu, C., Wahl, J., Eisenbrand, F., Karrenbauer, A., Soh, K.M., Hitzelberger, C.: Method for triggering matrix displays. German Patent Application 10 2005 063 159 PCT/EP2006/012362 (filed 12/30/2005, pending)
2. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Algorithms and Combinatorics, vol. 2. Springer, Berlin (1988)
3. Khachiyan, L.: A polynomial algorithm in linear programming. Dokl. Akad. Nauk SSSR **244**, 1093–1097 (1979)
4. Dantzig, G., Fulkerson, R., Johnson, S.: Solution of a large-scale traveling-salesman problem. J. Oper. Res. Soc. Am. **2**, 393–410 (1954)
5. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems, FOCS pp. 300–309 (1998)
6. Xu, C., Karrenbauer, A., Soh, K.M., Wahl, J.: A new addressing scheme for PM OLED display. In: Morreale, J. (ed.) SID 2007 International Symposium Digest of Technical Papers. Society for Information Display, vol. XXXVIII, pp. 97–100. Long Beach, USA (2007)
7. Eisenbrand, F., Karrenbauer, A., Xu, C.: Algorithms for longer OLED lifetime. In: Demetrescu, C. (ed.) WEA 2007. Lecture Notes in Computer Science, vol. 4525, pp. 338–351. Springer, Berlin (2007)
8. Paatero, P., Tapper, U.: Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. Environmetrics **5**, 111–126 (1994)
9. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization, Adv. Neural Inf. Process. Syst. **13** (2001)
10. Smith, E., Routley, P., Foden, C.: Processing digital data using non-negative matrix factorization. Patent GB 2421604A, pending (2005)
11. Smith, E.C.: Total matrix addressing (TMA™). In: Morreale, J. (ed.) SID 2007 International Symposium Digest of Technical Papers. Society for Information Display, vol. XXXVIII, pp. 93–96. Long Beach, USA (2007)
12. Ehrgott, M., Hamacher, H.W., Nußbaum, M.: Decomposition of matrices and static multileaf collimators: a survey, vol. 12, pp. 25–46 (2007)
13. Murano, S., Burghart, M., Birnstock, J., Wellmann, P., Vehse, M., Werner, A., Canzler, T., Stübinger, T., He, G., Pfeiffer, M., Boerner, H.: Highly efficient white OLEDs for lighting applications. SPIE, San Diego (2005)
14. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)
15. Schrijver, A.: Combinatorial Optimization—Polyhedra and Efficiency. Algorithms and Combinatorics, vol. 24. Springer, Berlin (2003)
16. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica **1**(2), 169–197 (1981)