This paper was published in the Proceedings of the Meeting on Algorithm Engineering & Experiments, Kyoto, Japan, January 16, 2012 (ALENEX 2012), pp. 45-54, SIAM, 2012.

Leveling the Grid

Sabine Cornelsen* Andreas Karrenbauer† Shujun Li‡

Abstract

Motivated by an application in image processing, we introduce the grid-leveling problem. It turns out to be the dual of a minimum cost flow problem for an apex graph with a grid graph as its basis. We present an $O(n^{3/2})$ algorithm for this problem. The optimum solution recovers missing DC coefficients from image and video coding by Discrete Cosine Transform used in popular standards like JPEG and MPEG. Generally, we prove that there is an $O(n^{3/2})$ min-cost flow algorithm for networks that, after removing one node, are planar, have bounded degrees, and have bounded capacities. The costs may be arbitrary.

1 Introduction

We consider a recovery problem in the context of image and video coding. It would be easy to restore missing image information, if only the brightness of isolated pixels were missing. It is intuitive to set the missing brightness to the average of the neighboring pixels because they are highly correlated. However, image and video data is typically transformed [19] for a higher compression ratio by de-correlating redundant information between neighboring pixels. One of the most used methods is the Discrete Cosine Transform (DCT) [2], which has been adopted in many image and video coding standards like JPEG and MPEG.

Briefly speaking, transform based coding works as follows. First, an image (or a video frame) is partitioned into $N \times N$ blocks, at which N is normally 8. Then, each block is transformed independently by an invertible linear map, which we specify in Sect. 2 precisely. If one of these transformed values is missing, then the whole block will be affected, particularly, when the most important value, called the DC coefficient, which is the average brightness of that block, is concerned.

*Sabine Cornelsen is with Department of Computer & Information Science, University of Konstanz, Germany. Email address: Sabine.Cornelsen@uni-konstanz.de.

†Andreas Karrenbauer is with the Zukunftskolleg and with Department of Computer & Information Science, University of Konstanz, Germany. Email address: Andreas.Karrenbauer@uni-konstanz.de.

‡Shujun Li is with Department of Computing, University of Surrey, Guildford, UK. Email address: Shujun.Li@surrey.ac.uk.

Setting the missing DC coefficients to 0 and computing the inverse transform yields the deviation of each pixel's brightness from the average of the corresponding block. We obtain an initial guess for the original image if we shift each block such that these values just become non-negative (see middle of Fig. 1). The average brightness of a block in this initial guess is certainly a lower bound on the actual average brightness of that block. It thus remains to find a non-negative offset for each block to level out the artifacts along the block boundaries.

Missing DC coefficients can be caused by selective encryption, transmission errors, or deliberate removal by malicious attackers. For instance, light-weight perceptual encryption can be achieved by encrypting important transform coefficients only. Consider a live broadcast of a sports event for paying subscribers. The computational demands on the customers' devices knowing the secret key remains low, while it is sufficient to spoil TV pirates if the effort is too high to recover the stream without the key in acceptable quality and in real-time. On the other hand, if there are efficient algorithms for the recovery problem, then less data can be sent to save bandwidth without reducing the quality noticeably.

1.1 Related Work. For DCT-transformed images and videos, researchers have proposed to implement perceptual encryption by selectively encrypting parts of the transformed values, which include encrypting different subsets and/or certain bits of them [13, 5]. One widely used setting is DC encryption, i.e. all DC coefficients are encrypted. The main purpose of perceptual encryption is twofold: on one side, by encrypting only part of the image and video data the additional computational load is reduced; on the other hand, keeping some DCT coefficients untouched leaves the space for some postprocessing (e.g. watermarking into the unencrypted part) without access to the key. Normally the encryption part is guaranteed by using a cryptographically strong cipher like AES. However, an attacker might cheat by dismissing the encrypted part and instead recovering it from the unencrypted information. For DC encryption, this means recovering missing DC coefficients from known data.

In the image processing field, it was widely thought

Figure 1: The original is on the left, the remainder without the missing DC coefficients is in the middle, and the recovery by solving the grid-leveling problem is on the right.

that missing DC coefficients cannot be effectively recovered, before Uehara *et al.* reported in [22] that one can approximately recover missing DC coefficients by exploiting the fact that the difference of neighboring pixel values in natural images observes a Laplace distribution with zero mean and small variance [18]. Their method generally works well and is computationally efficient, but the perceptual quality of the recovered image is not always good enough.

An improved DC recovery method was reported in [12]. This method is slower, but can produce better recovery results in general. It is not based on an explicit mathematical optimization model, but on an empirical observation. In [14], Li *et al.* further developed the DC recovery problem by modeling it as a linear program, which can produce even better results. It is almost impossible to distinguish the original and the optimum solution in Fig. 1. Moreover, the LP approach allows the recovery of more than only the DC coefficients, which is not possible with the methods proposed before.

1.2 Our contribution. We present the first exact combinatorial algorithm for DC-Recovery by transforming the linear program of [14] to a combinatorial optimization problem called *grid-leveling problem*. We show that it is the dual of a minimum cost flow problem with bounded capacities. This allows us to improve the time to compute an optimum solution by two orders of magnitudes as we demonstrate in the experiments. Moreover, we prove that there is a combinatorial algorithm that finds such a flow of minimum cost in $O(n^{3/2})$ for apex graphs that become planar graphs with bounded degrees after removing one node. The best previously known running time bound for these problems comes from a more general analysis of interior point methods for the minimum cost flow linear program for planar graphs [8]. It is $O(n^{1.594}\sqrt{\log n}\log(n\gamma))$, at which γ is an upper bound on the absolute values of costs and capacities, and generalizes to all graphs with a separator theorem, hence, including the graph classes mentioned above. Our bound does not depend on the costs and it dominates for small capacities or large costs. It becomes $O(c_{\max} \cdot n^{3/2})$ for arbitrary capacities of at most $c_{\max}$. Recently, Cornelsen and Karrenbauer have presented an $O(n^{3/2})$ algorithm for planar graphs with bounded face-degree, bounded costs, and infinite capacities [3]. However, these conditions are not fulfilled in the grid-leveling problem.

If we do not want to consider the block size $N \times N$ as a constant in the grid-leveling problem, we obtain a running time in $O(nN \log N + n^{3/2}N)$ for our algorithm, at which n is the number of blocks. The $N \log N$ comes from sorting N numbers, which may be reduced to $N \log\log N$ when the input data is integer and integer sorting on a unit-cost-RAM is used [1].

The paper is organized as follows. In Sect. 2, we introduce a mathematical model for the DC-recovery problem. It is then transformed to a combinatorial problem in Sect. 3. In Sect. 4, we show how to efficiently solve the grid-leveling problem as a min-cost flow problem. To this end, we follow a divide and conquer approach in Sect. 4.1 using planar separators, which have been introduced in the seminal work of Lipton and Tarjan [15], and which have been used over the last three decades in various forms for planar flow problems (e.g. [9, 7, 16]), mainly for shortest-path and max-flow. We conclude with the presentation of computational experiments in Sect. 5.

2 The DC-Recovery Problem

To be self-contained, we mention briefly the concept of the block-wise DCT typically used in image coding. First, an image is partitioned into blocks of size $N \times N$. Let $x_{ij} \in [x_{\min}, x_{\max}]$ denote the values of the pixels of such a block, i.e. $0 \le i, j < N$. The DCT coefficients $y_{k\ell}$ are defined by the $N \times N$ two-dimensional DCT as

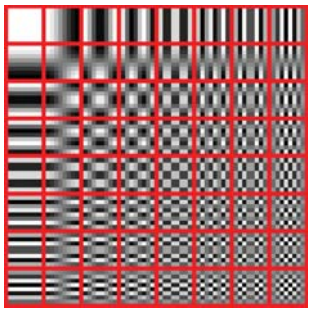


Figure 2: An illustration of the basis vectors of the two-dimensional DCT. The vector corresponding to the DC coefficient is at the top-left, i.e. $k = \ell = 0$.

follows:

$$\begin{aligned} x_{ij} &= \sum_{0 \le k,\ell < N} A(i,j,k,\ell) \cdot y_{k\ell} \\ &= \frac{y_{00}}{N} + \sum_{\substack{0 \le k,\ell < N \\ k+\ell > 0}} A(i,j,k,\ell) \cdot y_{k\ell}, \end{aligned} \tag{2.1}$$

where

$$A(i,j,k,\ell) \;=\; C_k C_\ell \cos\left(\tfrac{(i+0.5)k}{N}\pi\right) \cos\left(\tfrac{(j+0.5)\ell}{N}\pi\right),$$

with $C_k = \sqrt{1/N}$ when $k = 0$ and $\sqrt{2/N}$ when $k > 0$. Note that Eq. (2.1) is a linear map and the indices are relative to each block. We will use $x = A \cdot y$ to denote the block-wise DCT in the following. Since the DCT is invertible, it can be considered as a basis transform of a vector space. An illustration is given in Fig. 2.

Because y_{00} is the coefficient of the uniform vector with $1/N$ in each component, it is called the *DC coefficient* due to the analogy with *direct current*. The others are called *AC coefficients* in reference to *alternating current*. Note that the contribution of a DC coefficient is equal for each pixel in the same block.

The DC recovery problem is based on a property, which is a well-known feature of most natural images [18].

PROPERTY 2.1. *The differences between neighboring pixels are well described by a Laplace distribution with zero mean and a small variance.*

Note that we consider the 4-neighborhood of all the pixels here. That is, the difference with the left, right, top, and bottom neighbor of each pixel is accounted. Hence, each pair of pixels is considered twice with alternating sign. The distribution is therefore symmetric w.r.t. its mean of 0.

We wish to recover all missing DC coefficients such that the resulting pixel values maximize the likelihood according to Property 2.1. That is, we shall minimize the maximum likelihood estimator (MLE) of the standard deviation σ. Since the probability density function of the Laplace distribution with mean 0 is given by $f(z) = \frac{1}{\sigma\sqrt{2}} \exp\left(-\frac{\sqrt{2}}{\sigma}|z|\right)$, the MLE of σ is $\frac{\sqrt{2}}{S}\sum_{i=1}^{S}|z_i|$ where S is the number of observations [17].

Thus, we have to solve the following linear optimization problem [14]:

$$\begin{aligned} \text{min.} \quad & \sum_{i,j} |x_{ij} - x_{i+1,j}| + |x_{ij} - x_{i,j+1}| \\ \text{s.t.} \quad & x = A \cdot y, \\ & x_{\min} \le x_{ij} \le x_{\max}, \\ & y_{k\ell} = y^*_{k\ell} \qquad \text{for all AC coefficients.} \end{aligned} \tag{2.2}$$

The last equality fixes AC coefficient y_{kl} to its known value y^*_{kl} while DC coefficient y_{00} of each block remains to be a variable.

3 Transformation to a Combinatorial Problem

In the following, we first transform the non-combinatorial linear program (2.2) to an LP with an integral constraint matrix. Afterwards, we derive a min-cost network flow problem from its dual. This enables us to solve the recovery problem by an efficient combinatorial algorithm.

Since many variables of (2.2) are pre-determined, we will simplify the formulation as follows. We first note that there is only one free variable for each block. We consider an arbitrary block v in the following. Let x^*_{ij} denote the DC-free part of x_{ij}, i.e. the contribution of all AC coefficients. As mentioned in the introduction, x^*_{ij} is the deviation from the average brightness of that block. Formally, we have $x_{ij} = \frac{y_{00}}{N} + x^*_{ij}$. Let $x^*_{\min}$ be the minimum over all x^* of that block. Since all the pixel values are integers in the range between $x_{\min}$ and $x_{\max}$, typically in $\{0, \ldots, 255\}$, we obtain integers $\hat{x}_{ij} = x^*_{ij} - x^*_{\min} + x_{\min}$, which correspond to the initial guess in the middle of Fig. 1 in the introduction. Hence we may assume that $x_{ij} = h_v + \hat{x}_{ij}$, at which the variable h_v is an integer offset that is added to all shifted DC-free pixel values $\hat{x}_{ij}$ of that block v.

Since the DC value of a block, and thus also the corresponding h-variable, contributes to all pixels of the same block equally, only the pairs of neighboring pixels that belong to different blocks are relevant.

Hence, we may restrict ourselves to these pairs. Put differently, when we consider the 4-neighborhood relation of the pixels as a grid graph, we may contract the edges within each block and obtain a grid multi-graph, say G. The edges of this graph have multiplicity N.

For each pair v and w of neighboring blocks, the N parallel edges between v and w correspond to N pairs $(v,w)_\alpha$ and $(w,v)_\alpha$, $\alpha = 1, \ldots, N$ of neighboring pixels such that $(v,w)_\alpha$ is contained in block v and $(w,v)_\alpha$ is

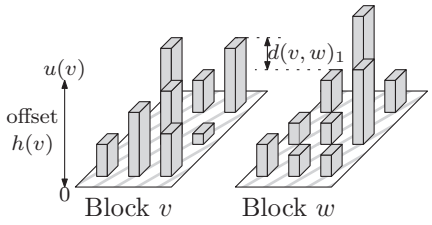


Figure 3: An illustration of the grid-leveling problem.

contained in block w and we set $d(v,w)_\alpha := \hat{x}_{(w,v)_\alpha} - \hat{x}_{(v,w)_\alpha} =: -d(w,v)_\alpha$. See Fig. 3 for an illustration. We are now ready to describe our mathematical model.

So throughout this paper, we consider an $n_1 \times n_2$-*grid* N-*multi-graph* $G = (V, E_N)$ with $n = n_1 \times n_2$ vertices, i.e.

$$\begin{aligned} V &= \{(i,j);\ 1 \le i \le n_1, 1 \le j \le n_2\}, \\ E &= \{((i_1,j_1),(i_2,j_2)) \in V \times V; \\ &\qquad |i_2 - i_1| + |j_2 - j_1| = 1\}, \text{ and} \\ E_N &= \{e_\alpha;\ e \in E, 1 \le \alpha \le N\}. \end{aligned}$$

The *grid-leveling problem* $(G = (V, E_N), u, d)$ is defined as follows. Given $u : V \to \mathbb{Z}_{\ge 0}$ and $d : E_N \to \mathbb{Z}$ with $d(v,w)_\alpha = -d(w,v)_\alpha$ find $h : V \to \mathbb{Z}$ that

$$\text{minimize} \sum_{(v,w)_\alpha \in E_N} \underbrace{\max\{0, h(v) - h(w) - d(v,w)_\alpha\}}_{=:\theta(v,w)_\alpha}$$

subject to

$$0 \le h(v) \le u(v) \quad \text{for } v \in V.$$

Note that $\theta(v,w)_\alpha + \theta(w,v)_\alpha = |h(v) - h(w) - d(v,w)_\alpha| = |x_{(v,w)_\alpha} - x_{(w,v)_\alpha}|$. By adding the inequalities $h(v) - h(w) - d(v,w)_\alpha \le \theta(v,w)_\alpha$ and $\theta(v,w)_\alpha \ge 0$ for $(v,w)_\alpha \in E_N$, we obtain a proper linear programming formulation. As a matter of fact, this is an LP over an integer polyhedron since the constraint matrix is totally unimodular. It is not hard to see that the constraint matrix is made up by the transpose of a network matrix to which identity matrices are attached.

This implies that there is an integer optimum solution vector whenever the optimum is finite. We may compute such a vector in polynomial time by the ellipsoid method [11] or by interior point methods [10]. In general, this yields only a weakly polynomial bound, but using the result of Tardos [20], we obtain a strongly polynomial time algorithm for this problem. This holds in general for computing least absolute deviation regressions of equations with a totally unimodular matrix, and an integer right-hand side, at which the integer variables may be bounded or unbounded integer.

In the remainder of this paper, we give a combinatorial algorithm that solves the grid-leveling problem efficiently. The algorithm is based on the dual that we obtain from a reformulation as a maximization problem, i.e., of the problem in which the objective function is replaced by "max $\sum_{(v,w)_\alpha \in E_N} -\theta(v,w)_\alpha$". We use the primal/dual relation

$$\max\{c^T x : Ax \le b, x \ge 0\} = \min\{b^T f : A^T f \ge c, f \ge 0\},$$

which translates for grid-leveling to finding a function $f : V \cup E_N \to \mathbb{Z}_{\ge 0}$ that minimize

$$\sum_{v \in V} u(v) f(v) + \sum_{(v,w)_\alpha \in E_N} d(v,w)_\alpha f(v,w)_\alpha$$

subject to

$$f(v) + \sum_{(v,w)_\alpha \in E_N} f(v,w)_\alpha - \sum_{(w,v)_\alpha \in E_N} f(w,v)_\alpha \ge 0$$

for $v \in V$ and

$$f(v,w)_\alpha \le 1$$

for $(v,w)_\alpha \in E_N$.

4 Solving the Min-Cost Flow Problem

A *min-cost flow network* $\mathcal{N}$ consists of a directed (multi)graph $G = (V, E)$, edge capacities $c : E \to \mathbb{Z}_{\ge 0} \cup \{\infty\}$, node demands $b : V \to \mathbb{Z}$ with $\sum_{v \in V} b(v) = 0$, and edge costs $d : E \to \mathbb{Z}$. A *pseudo-flow* on $\mathcal{N}$ is a map $f : E \to \mathbb{Z}_{\ge 0}$ with $f(e) \le c(e)$ for $e \in E$. A pseudo-flow is a *flow* if the deficiency

$$b_f(v) := b(v) + \sum_{\substack{e \in E \\ v \text{ head of } e}} f(e) - \sum_{\substack{e \in E \\ v \text{ tail of } e}} f(e)$$

of each node $v \in V$ is zero. The map $f : E \to \mathbb{Z}_{\ge 0}$ is a *min-cost flow* on $\mathcal{N}$ if f is a flow on $\mathcal{N}$ that minimizes the *cost* $\sum_{e \in E} f(e) d(e)$ among all flows.

The residual network $(G_f = (V, E_f), c_f, b_f, d_\pi)$ of a min-cost flow network $\mathcal{N} = (G, c, b, d)$, a pseudo-flow f, and *node potentials* $\pi : V \to \mathbb{Z}_{\ge 0}$ is defined as follows. For each edge $e \in E$ from v to w the edge set E_f contains e with $d_\pi(e) := d(e) + \pi(v) - \pi(w)$ if $c_f(e) := c(e) - f(e) > 0$ and a reversed copy $-e$ from w to v with $d_\pi(-e) = -d(e) - \pi(v) + \pi(w)$ if $c_f(-e) := f(e) > 0$. The costs d_π are called the *reduced costs* and c_f are the *residual capacities*. The node potentials are *valid* if $d_\pi(e) \ge 0$ for all $e \in E_f$. Note that a flow has minimum cost if and only it has valid node potentials.

Let p be a path in the residual network $(G_f = (V, E_f), c_f, d_\pi)$ and let $\Delta_f(p)$ be the minimum capacity

on the edges of p. *Augmenting* the pseudo-flow f on the path p by $\Delta \leq \Delta_f(p)$ means adding Δ to f on the edges of p that are original edges of G and subtracting Δ from f on the edges of G such that their reversed copies are edges of p.

The *successive shortest-path algorithm* [6] works as follows. It starts with the node potentials $\pi = 0$ and the pseudo flow f in which all edges with negative costs are saturated, i.e. $f(e) := c(e)$ if $d(e) < 0$ and $f(e) := 0$ otherwise. As long as there is a node s with positive deficiency, the algorithm tries to find a shortest path p in G_f from s to a node t with negative deficiency (where the edge distances are d_π) and augments f on p by $\min\{\Delta_f(p), b_f(s), -b_f(t)\}$. In each step and for each $v \in V$ the length $\text{dist}_{f,\pi}(s, v)$ of a shortest sv-path in (G_f, d_π) is added to $\pi(v)$, thus maintaining π valid. See Algorithm 1 for a pseudocode.

Algorithm 1: Successive Shortest-Path [6]

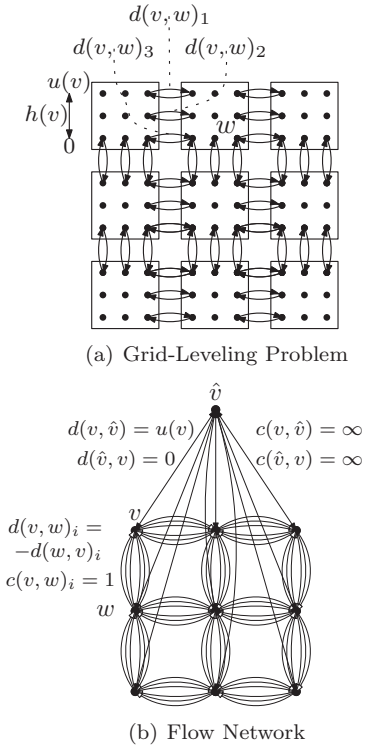
Input : min-cost flow network (G, c, d) with $c(e) < \infty$ if $d(e) < 0$.
Output: min-cost flow $f : E \to \mathbb{Z}_{\geq 0}$ of (G, c, d) with valid node potentials $\pi : V \to \mathbb{Z}_{\geq 0}$, both initialized to 0

```
SuccessiveShortestPath(G, c, d)
    for each edge e with d(e) < 0 do
        f(e) ← c(e);
    while there is a node s with b_f(s) > 0 do
        SingleSourceShortestPath(G_f, d_π, s);
        for v ∈ V do
            π(v) ← π(v) + dist_{f,π}(s, v);
        Augment f by min{Δ_f(p), b_f(s), −b_f(t)} on
        shortest st-path p for some t with b_f(t) < 0;
    return (f, π);
```

For a planar (multi-)graph $G = (V, E)$ let $\hat{G} = (\hat{V}, \hat{E})$ be the *apex graph* in which one additional node $\hat{v}$ is added to V and for each $v \in V$ an edge to and from $\hat{v}$ is added to $\hat{E}$, i.e., $\hat{V} = V \cup \{\hat{v}\}$ and $\hat{E} = E \cup \{(\hat{v}, v), (v, \hat{v});\ v \in V\}$.

Let now $(G = (V, E_N), u, d)$ be an instance of the grid-leveling problem. The dual problem can be modeled as a min-cost flow problem $(\hat{G}, c, b = 0, d)$ as follows. For an edge $e \in \hat{E}$ let $c(e) = 1$ if $e \in E_N$ and infinite otherwise. Further, let $d(v, \hat{v}) = u(v), v \in V$, and $d(\hat{v}, v) = 0, v \in V$. Then an optimum solution of the min-cost flow problem $(\hat{G}, c, b = 0, d)$ is an optimum solution for the dual of the grid-leveling problem. See Fig. 4 for an illustration.

Moreover, the node potentials computed in the successive shortest-path algorithm yield an optimal



(a) Grid-Leveling Problem

(b) Flow Network

Figure 4: From the grid-leveling problem to the associated min-cost flow problem

solution for the dual problem of the min-cost flow problem, and, hence, for our primal problem. We explicitly state this property in our special case, i.e. for the grid-leveling problem. An explicit proof can be found in the appendix.

LEMMA 4.1. *Let $\mathcal{N} = (\hat{G}, c, b = 0, d)$ be the min-cost flow problem for an instance (G, u, d) of the grid-leveling problem. Let f be a flow on $\mathcal{N}$ with valid node potentials π. Then $h(v) = \pi(\hat{v}) - \pi(v)$ is an optimal solution for (G, u, d).*

Proof. Because of their infinite capacities, both $(v, \hat{v})$ and $(\hat{v}, v)$ are in the residual network; it follows that $u(v) + \pi(v) - \pi(\hat{v}) \geq 0$ and $0 + \pi(\hat{v}) - \pi(v) \geq 0$. Thus, $0 \leq h(v) \leq u(v)$ and, hence, h is a feasible solution for the grid-leveling problem. We next show that

$$\sum_{e \in \hat{E}} f(e) d_\pi(e) = \sum_{(v,w)_\alpha \in E_N} -\max\{0, h(v) - h(w) - d(v, w)_\alpha\}.$$

Let $e \in \hat{E}$. Assume first that $e = (v, w)_\alpha \in E_N$. If $f(v, w)_\alpha = 0 < 1$ then $(v, w)_\alpha$ is in the residual network.

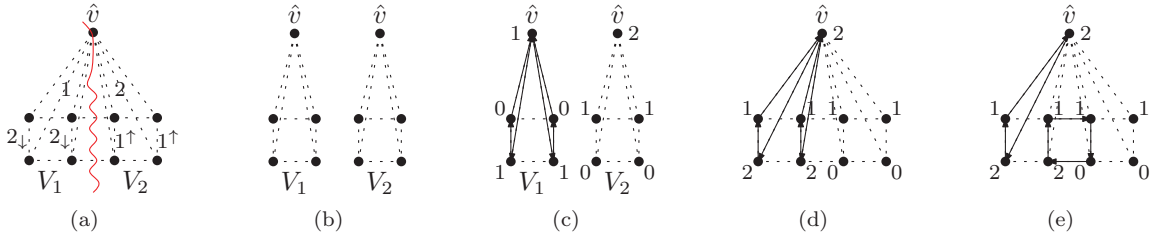


Figure 5: Illustration of Algorithm 2: (a) An instance of the min-cost flow problem associated with the grid-leveling problem on a 2×4-grid 1-multi-graph. Horizontal edges have cost 0. On the left hand side the edges pointing downward have cost 2 and the edges to the apex have cost 1. On the right hand side the edges pointing upward have cost 1 and the edges to the apex have cost 2. All edges are dotted. (b) The grid is divided into two parts, the apex is duplicated. (c) A recursive solution for the two parts. Node labels indicate node potentials. Edges with non-zero flow are solid. (d) The potential of the apex is set to the maximum of the potentials of the two copies, adjusting the potentials in the respective component. (e) The final flow is computed.

Thus, $d(v,w)_\alpha - h(v) + h(w) = d(v,w)_\alpha + \pi(v) - \pi(w) \geq 0$ and, hence, $\max\{0, h(v) - h(w) - d(v,w)_\alpha\} = 0$. If $f(v,w)_\alpha = 1$ then $-(v,w)_\alpha$ is in the residual network and, hence, $0 \leq d(w,v)_\alpha + \pi(w) - \pi(v) = -d(v,w)_\alpha + h(v) - h(w)$. It follows that $-\max\{0, h(v) - h(w) - d(v,w)_\alpha\} = -(h(v) - h(w) - d(v,w)_\alpha) = d_\pi(v,w)_\alpha$. In both cases it follows that

$$f(e)d_\pi(e) = -\max\{0, h(v) - h(w) - d(v,w)_\alpha\}.$$

Assume now that $e \in \hat{E} \setminus E_N$ and that $f(e) \neq 0$. Since e has infinite capacity it follows that both, e and $-e$ are in the residual network. Thus, $d_\pi(e) = 0$. Since f is a flow it follows that

$$\begin{aligned}\sum_{e \in \hat{E}} f(e)d(e) &= \sum_{e \in \hat{E}} f(e)d_\pi(e) \\ &= \sum_{(v,w)_\alpha \in E_N} -\max\{0, h(v) - h(w) - d(v,w)_\alpha\}\end{aligned}$$

and thus, by the weak duality, h is an optimum solution for the grid-leveling problem and f is an optimum solution for the dual min-cost flow problem.

4.1 A Divide and Conquer Approach. In this section, we show how to use a divide and conquer approach to efficiently solve the min-cost flow problem associated with the grid-leveling problem. More generally, we show the following theorem.

THEOREM 4.1. *A min-cost flow on an apex graph for which the removal of one node yields a planar flow network with $\mathcal{O}(n)$ edges, capacities at most $c_{\max}$ and node degree at most Δ can be computed in $\mathcal{O}(c_{\max}\sqrt{\Delta}\, n^{3/2})$ time.*

Proof. In the following, we denote the apex by $\hat{v}$ and we write $b(V') := \sum_{v \in V'} b(v)$ for $V' \subseteq V$. We may assume w.l.o.g. that there are edges $(v, \hat{v})$, $(\hat{v}, v)$ with infinite capacity for all $v \in V$, since we may add such edges with sufficiently large costs, say the sum of the absolute values of the original costs plus 1. These auxiliary edges may lay in parallel to existing edges but still the number of edges remains in $\mathcal{O}(n)$. The instance with the auxiliary edges is always feasible and the original instance is infeasible if and only if an auxiliary edge is used in an optimum solution.

Let $\mathcal{N} = (\hat{G} = (V \cup \{\hat{v}\}, \hat{E}), c, b, d)$ be the min-cost flow network such that the subgraph $G = (V, E)$ induced by V is planar. The algorithm works as follows (see Algorithm 2 for a pseudocode and Fig. 5 for an illustration).

We compute a cut $(V_1, V_2 = V \setminus V_1)$ of G in linear

Algorithm 2: Recursive Min-Cost Flow

Input : min-cost flow network $\mathcal{N} = (\hat{G} = (V \cup \{\hat{v}\}, \hat{E}), c, b, d)$, with $c(\hat{v}, v) = c(v, \hat{v}) = \infty$, $v \in V$

Output: min-cost flow f on $\mathcal{N}$ and valid node potentials π, both initialized to 0.

```
RecursiveMCF(Ĝ, c, b, d)
    (V1, V2) ← Cut(G);
    for i = 1, 2 do
        (f|_{Ê_i}, π_i) ← RecursiveMCF(Ĝ_i, c,
                b with b(v̂) = −b(V_i), d);
    π(v̂) ← max{π_1(v̂), π_2(v̂)};
    for v ∈ V_i, i = 1, 2 do
        π(v) ← π_i(v) − π_i(v̂) + π(v̂);
    return (f, π)+
      SuccessiveShortestPath(Ĝ_f, c_f, b_f, d_π);
```

time [4] such that $|V_i| \leq \frac{2}{3}|V|$, $i = 1, 2$, and the set of cut-edges $E(V_1, V_2)$, i.e., the set of edges that are incident to both, a vertex in V_1 and V_2, contains $\mathcal{O}(\sqrt{\Delta n})$ edges. Let $\hat{G}_i = (V_i \cup \{\hat{v}\}, \hat{E}_i), i = 1, 2$ be the subgraph of $\hat{G}$ induced by $V_i \cup \{\hat{v}\}$.

Then we recursively compute min-cost flows $f|_{\hat{E}_i}$, $i = 1, 2$ with valid node potentials π_i on $\hat{G}_i$. Note that we modify $b(\hat{v}) = -b(V_i)$ for the respective recursive call. When merging the two recursive solutions, we set $f(e) = 0$ for all $e \in E(V_1, V_2)$. Since $b(\hat{v}) = -b(V_1) - b(V_2)$ it follows that $b_f(\hat{v}) = 0$ and thus f is a flow in $\hat{G}$. Furthermore, we exploit the fact that node potentials remain valid when the same value is added to all of them. This allows us to achieve valid node potentials for all edges in $\hat{E}_1 \cup \hat{E}_2$ by setting $\pi(\hat{v}) = \max\{\pi_1(\hat{v}), \pi_2(\hat{v})\}$ and by adjusting the potentials in the respective component. Note, however, that the edges in $E(V_1, V_2)$ might have negative reduced costs. This will be fixed by a call of the successive shortest path algorithm. After saturating the edges with negative reduced costs, the sum of the deficiencies over all vertices with positive deficiency can be bounded by $\sum_{e \in E(V_1,V_2)} c(e) \in \mathcal{O}(c_{\max}\sqrt{\Delta n})$. Hence, there are at most $\mathcal{O}(c_{\max}\sqrt{\Delta n})$ iterations within SuccessiveShortestPath. Since each shortest-path computation can be performed in linear time [21] each recursive step and, hence, the whole algorithm finishes in $\mathcal{O}(c_{\max}\sqrt{\Delta}\, n^{3/2})$ time.

4.2 Dealing with Multiple Edges. In the grid-leveling problem, we have multiple edges between two vertices. Since for each pair of adjacent vertices v, w only the residual edge from v to w with minimum reduced cost has to be considered, each shortest-path computation has to be performed on a simple directed graph. To decide which edge among the parallel edges between two adjacent vertices has to be considered for the next shortest-path computation, we first sort these edges in totally $\mathcal{O}(nN \log N)$ time and maintain a pointer to the edge with minimum cost among the parallel edges in the residual network. Note that this pointer can be updated with asymptotically no extra costs.

For the bidirected grid graph, which originates from the 4-neighborhood in our application, we have $\Delta = 8$. Since $c_{\max} = 1$ due to the uniform objective of the grid-leveling problem, we obtain the following theorem by applying Theorem 4.1 and multiplying the number of edges in the cut by N.

Theorem 4.2. *The grid-leveling problem on a grid N-multi-graph with n vertices can be solved in $\mathcal{O}(nN \log N + Nn^{3/2})$ time.*

Note that a balanced cut with at most $\mathcal{O}(N\sqrt{n})$ cut edges can be easily constructed on a grid N-multi-graph by dividing the longer of the two sides in the middle. We shall remark that a simple implementation of Dijkstra with binary heaps computes shortest paths in $O(n \log n)$ on graphs with $O(n)$ edges and we thereby only lose a log-factor. That is, we obtain a practical $O(n^{3/2} \log n)$ algorithm for solving the min-cost flow problem in this case. Furthermore, we could use this variant to extend the 4-neighborhood to obtain the 8-neighborhood by adding four nodes at the corners, which is also often used in image processing. Note that it is not planar and not minor closed anymore, but the edge separator is still of size $O(N\sqrt{n})$.

5 Experiments

We have implemented the combinatorial algorithm in C++ custom-tailored to solve the grid-leveling problem efficiently. Since the range of valid pixel values is typically $\{0, \ldots, 255\}$ in our application, the shortest path distances are bound to twice that range. We thus use a bucket queue with queue operations in constant time in our implementation of Dijkstra's shortest path algorithm, which then also runs in $O(n)$ and therefore relieves us from implementing the much more involved algorithm of [21]. Moreover, the edge-separator is trivially computed on a grid by dividing the longer of the two sides in the middle.

Our test set consists of 105 instances that result from 21 images of original size 512×512 that have been down-scaled to square images with side lengths of 256, 128, 64, and 32 pixels to measure the scaling behavior. The experiments have been carried out on a Dell XT2 laptop with an Intel dual core CPU (U9600 with 1.60GHz) and 5 GB RAM. We first compare the computation times of our implementation of the combinatorial algorithm with solving the LP model by the CPLEX 12.2 Concert C++ framework (see Fig. 6). The measured computation times comprise the time to solve an instance in RAM. That is, we exclude the time for reading and writing image data from and to disk, and the setup time of the corresponding data structures. For a fair comparison, we take the CPU time that is consumed by the barrier optimizer (which the fastest of the available LP algorithms in CPLEX 12.2 for this type of problem) and the CPU time that is consumed by the combinatorial algorithm. For the small instances and the new algorithm, the time is measured over 10000 runs on the same instance and scaled down accordingly because of the lower precision of the built-in timing functionality (Linux kernel 3.0, g++ compiler 4.5.1).

As one can see in Fig. 6, the combinatorial algorithm is about two orders of magnitudes faster on aver-

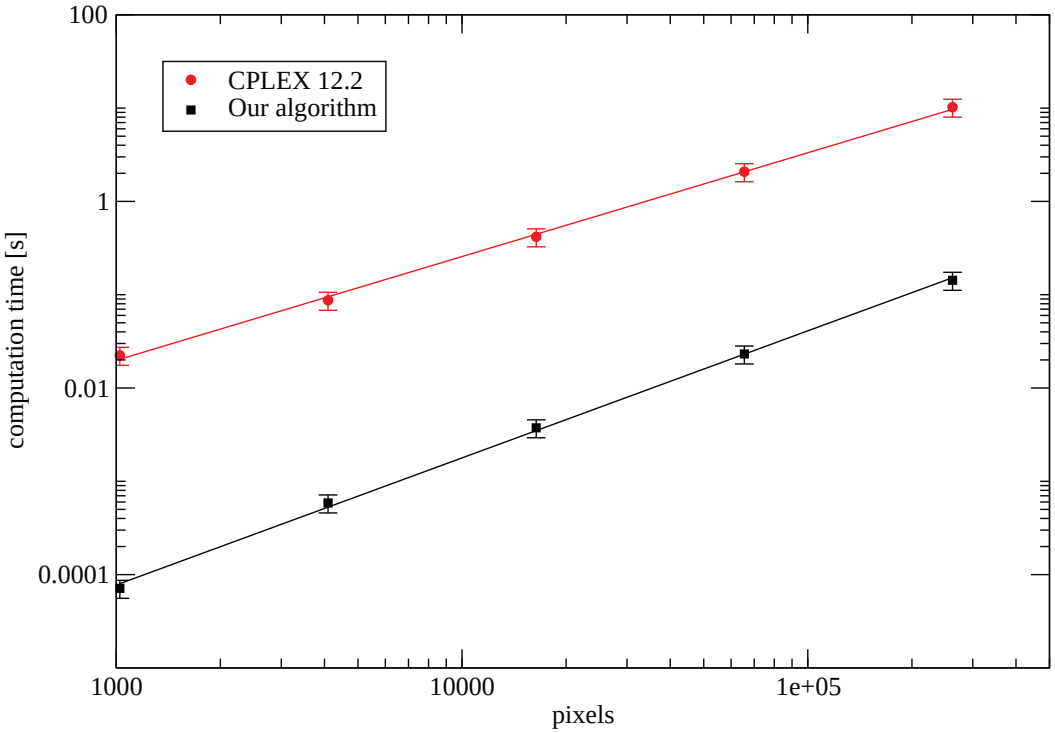


Figure 6: Comparison between the LP-model solved by CPLEX 12.2 and the flow model solved by our implementation. The two lines correspond to the fitted functions $10^{-5} \cdot x^{1.1(1)}$ and $10^{-8} \cdot x^{1.3(6)}$ at top and bottom, respectively. The parentheses indicate uncertain digits due to the statistical error. The error bars show the standard errors of the means.

age than the interior point method to solve the equivalent LP for images up to a size of 512×512. Although the fitted exponent for the combinatorial algorithm is larger than the one for CPLEX, the two curves intersect at about 10^{12} pixels, i.e. for images with height and width of one million. Needless to say that this will not happen in practice in the near future, since moreover, the memory requirement for CPLEX is more than 1 GB for images of size 512×512. Whereas less than 10 MB are allocated by the combinatorial algorithm for images of that size. Note that both algorithms produce nearly the same results w.r.t. image quality compared to the original images (the average SSIM is 96.6% for both approaches and the average PSNR is 26.34 for the combinatorial algorithm and 26.39 for LP). Although both algorithms compute an optimum solution, the optimum does not have to be unique. This explains the tiny difference in the image quality measures.

It remains to discuss the depth of recursion at which one should compute the min-cost flow of the subinstance directly. The data in Fig. 6 is for recursion level 1, which we define as no recursion. Since the sample means for more levels of recursion almost coincide with the plotted ones, we do not show them to avoid clutter. We rather concentrate on the images of size 512×512 and demonstrate the effect of the recursion depth in Fig. 7. It is negligible for the mean computation time, but the variance decreases until stagnation after recursion level 3.

Note that we can only test the scaling behavior of average computation times in a statistical meaningful way. However, it is tempting to say that our experiments suggest that even a non-recursive successive shortest path algorithm may achieve an $O(n^{3/2})$ running time bound. Note that we stop each shortest path search immediately when we have found a sink. The computational effort in practice is proportional to the number of discovered nodes. The theoretical difficulty is that we have to charge $O(n)$ nevertheless. It is an interesting future research question whether it is necessary to actually compute the edge separators or whether it is sufficient to consider them only virtually in the analysis to bound the number of queue operations.

On the other hand, the divide and conquer scheme allows a very straight-forward parallel implementation. Since the subgrids are disjoint for each of the child processes, there is no need to worry about exclusive writes. The only shared node is the apex. But it

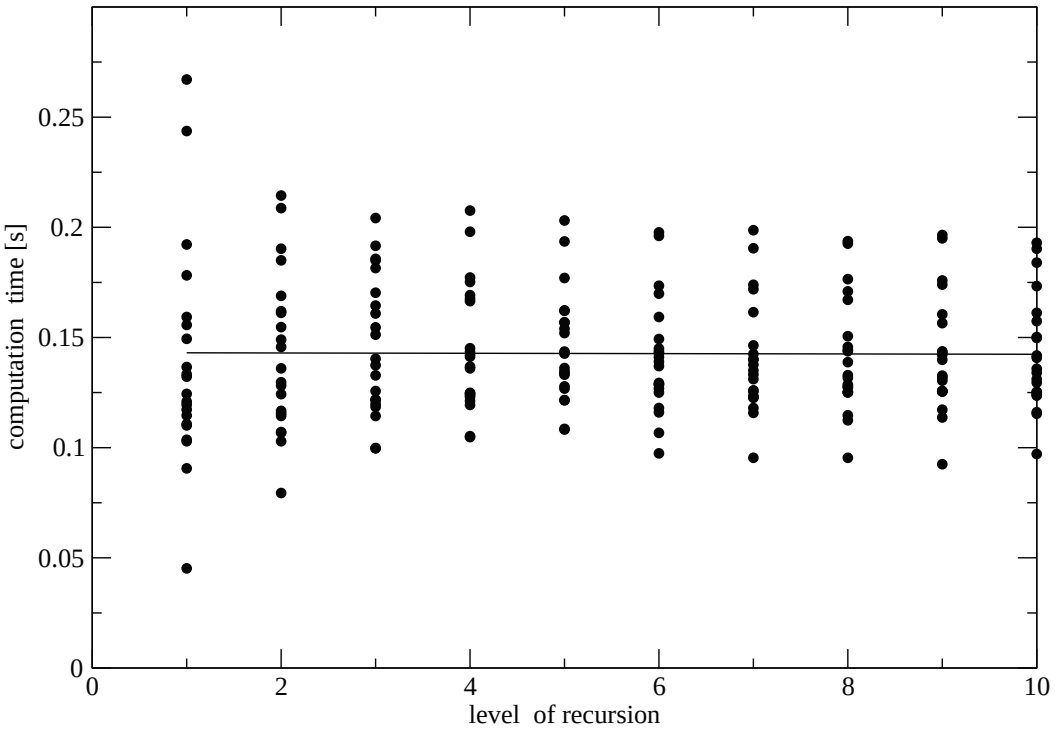


Figure 7: Average computation time w.r.t. the number of recursive calls on 21 images of size 512x512. Level 1 means no recursion. The line corresponds to a linear regression with $0.14(3) - 0.000(0) \cdot x$ with a slope within ± 0.001.

is easy to generate a new one for each child instance. Furthermore, the data for the arcs from and to the apex are stored at the corresponding grid nodes. The parallelization aspect is for example relevant for an implementation on the graphics card, when it comes to real-time decoding of HD videos.

References

[1] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57(1):74–93, 1998.

[2] V. Britanak, P. C.Yip, and K. R. Rao. *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations.* Academic Press, 2006.

[3] S. Cornelsen and A. Karrenbauer. Accelerated bend minimization. In *Proceedings of the 19th International Symposium on Graph Drawing*, 2011, to appear.

[4] K. Diks, H. N. Djidjev, O. Sykora, and I. Vrto. Edge separators of planar and outerplanar graphs with applications. *Journal of Algorithms*, 14(2):258–279, 1993.

[5] F. Dufaux and T. Ebrahimi. Scrambling for privacy protection in video surveillance systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8):1168–1174, 2008.

[6] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[7] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55:3–23, 1997. Special Issue on Selected Papers from STOC 1994.

[8] H. Imai and K. Iwano. Efficient sequential and parallel algorithms for planar minimum cost flow. In *SIGAL International Symposium on Algorithms*, LNCS 450, pages 21–30. Springer, 1990.

[9] D. B. Johnson and S. M. Venkatesan. Partition of planar flow networks. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 259–264. IEEE Computer Society, 1983.

[10] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[11] L. G. Khachiyan. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244:1093–1097, 1979.

[12] S. Li, J. J. Ahmad, D. Saupe, and C.-C. J. Kuo. An improved DC recovery method from AC coefficients of DCT-transformed images. In *Proceedings of the 17th IEEE International Conference on Image Processing (ICIP)*, pages 2085–2088, 2010.

[13] S. Li, G. Chen, A. Cheung, B. Bhargava, and K.-T. Lo. On the design of perceptual MPEG-video encryption algorithms. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(2):214–223, 2007.

[14] S. Li, A. Karrenbauer, D. Saupe, and C.-C. J. Kuo. Recovering missing coefficients in DCT-transformed images. In *Proceedings of the 18th IEEE International Conference on Image Processing (ICIP)*, pages 1569–1572, 2011.

[15] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[16] S. Mozes and C. Wulff-Nilsen. Shortest Paths in Planar Graphs with Real Lengths in $O(n \log^2 n / \log \log n)$ Time. In *Algorithms – ESA 2010*, volume 6347 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2010.

[17] J. K. Ord. Laplace distribution. *Encyclopedia of Statistical Sciences*, 6:3961–3962, 2006.

[18] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 3rd edition, 2005.

[19] Y. Q. Shi and H. Sun. *Image and Video Compression for Multimedia Engineering*. CRC Press, 2nd edition, 2008.

[20] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

[21] S. Tazari and M. Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation. *Discrete Applied Mathematics*, 157(4):673–684, 2009.

[22] T. Uehara, R. Safavi-Naini, and P. Ogunbona. Recovering DC coefficients in block-based DCT. *IEEE Transactions on Image Processing*, 15(11):3592–3596, 2006.