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.

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...
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 sorting $N$ numbers, which may be reduced to $N \log N$ 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 $x_{ij} \in [x_{\text{min}}, x_{\text{max}}]$ denote the values of the pixels of such a block, i.e. $0 \leq i, j < N$. The DCT coefficients $y_{k\ell}$ are defined by the $N \times N$ two-dimensional DCT as
According to Property 2.1. That is, we shall minimize the resulting pixel values maximize the likelihood mean of 0. The distribution is therefore symmetric w.r.t. its edges within each block and obtain a grid multi-graph, which is the coefficient of the uniform vector \( v \) corresponding to the initial frequency of all AC coefficients. As mentioned in the introduction, the others differ, when we consider the 4-neighborhood relation of the pixels as a grid graph, we may contract the AC coefficient is at the top-left, i.e. \( k = \ell = 0 \).

Following:

\[
x_{ij} = \sum_{0 \leq k, \ell < N} A(i, j, k, \ell) \cdot y_{k\ell} = \frac{y_{00}}{N} + \sum_{0 \leq k, \ell < N} A(i, j, k, \ell) \cdot y_{k\ell},
\]

where

\[
A(i, j, k, \ell) = C_k C_\ell \cos \left( \frac{(i+0.5)k \pi}{N} \right) \cos \left( \frac{(j+0.5)\ell \pi}{N} \right),
\]

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 alternating cur-

den 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 contribution of a DC coefficient is well described by a Laplace distribution with mean 0 and a small variance.

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

The last equality fixes AC coefficient \( y_{k\ell} \) to its known value \( y_{k\ell}^* \) 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

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 \( x_{ij} = h_{00} + \hat{x}_{ij} \), at which the variable \( h_{00} \) 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
contained in block \(w\) and we set \(d(v,w)\alpha := \tilde{x}(v,w)\alpha - \tilde{x}(v,w)\) for \((v,w)\) and \(\tilde{x}(v,w)\alpha := -d(w,v)\alpha\). 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.

\[
\begin{align*}
V &= \{(i,j); 1 \leq i \leq n_1, 1 \leq j \leq n_2\}, \\
E &= \{((i_1,j_1),(i_2,j_2)) \in V \times V; \left|i_2 - i_1\right| + \left|j_2 - j_1\right| = 1\}, \text{ and} \\
E_N &= \{\alpha; e \in E, 1 \leq \alpha \leq N\}.
\end{align*}
\]

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

\[
\begin{align*}
\text{minimize} \quad & \sum_{(v,w)\alpha \in E_N} \max\{0, h(v) - h(w) - d(v,w)\alpha\} = \theta(v,w)\alpha \\
\text{subject to} \quad & 0 \leq h(v) \leq u(v) \quad \text{for} \ v \in V.
\end{align*}
\]

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 \(h(v) - h(w) - d(v,w)\alpha \leq \theta(v,w)\alpha\) and \(\theta(w,v)\alpha \geq 0\) 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

\[
\max \{c^T x : Ax \leq b, x \geq 0\} = \min \{b^T f : A^T f \geq c, f \geq 0\},
\]

which translates for grid-leveling to finding a function \(f : V \cup E_N \to \mathbb{Z}_{\geq 0}\) 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

\[
\sum_{(v,w)\alpha \in E_N} f(v,w)\alpha - \sum_{(w,v)\alpha \in E_N} f(w,v)\alpha \geq 0
\]

for \(v \in V\) and

\[
f(v,w)\alpha \leq 1
\]

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

4 Solving the Min-Cost Flow Problem

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

\[
b_f(v) := b(v) + \sum_{e \text{ head of } v} f(e) - \sum_{e \text{ tail of } v} f(e)
\]

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

The residual network \((G_f = (V,E_f),c_f,b_f,d_f)\) of a min-cost flow network \(N\) is \((G,c,b,d)\), a pseudo-flow \(f\), and node potentials \(\pi : V \to \mathbb{Z}_{\geq 0}\) is defined as follows. For each \(e \in E\) from \(v\) to \(w\) the edge set \(E_f\) contains \(e\) with \(d_f(e) := d(e) + \pi(v) - \pi(w)\) if \(c_f(e) := c(e) - f(e) > 0\) and a reversed copy \(-e\) from \(w\) to \(v\) with \(d_f(-e) := -d(e) - \pi(v) + \pi(w)\) if \(c_f(-e) := f(e) > 0\). The costs \(d_f\) are called the \(\text{reduced costs}\) and \(c_f\) are the \(\text{residual capacities}\). The node potentials are \(\text{valid}\) if \(d_f(e) \geq 0\) 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 \((G_f = (V,E_f),c_f,d_f)\) 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 $\Delta$ to $f$ on the edges of $p$ that are original edges of $G$ and subtracting $\Delta$ 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 $d_\pi$) 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 $(G_f, d_\pi)$ is added to $\pi(v)$, thus maintaining $\pi$ 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.

1. **SuccessiveShortestPath**$(G, c, d)$
2. for each edge $e$ with $d(e) < 0$
   1. $f(e) \leftarrow c(e);$
3. while there is a node $s$ with $b_f(s) > 0$
   1. **SingleSourceShortestPath**$(G_f, d_\pi, s);$
   2. for $v \in V$
      1. $\pi(v) \leftarrow \pi(v) + \text{dist}_{f,\pi}(s, v);$  
5. Augment $f$ by $\min\{\Delta_f(p), b_f(s), -b_f(t)\}$ on shortest $st$-path $p$ for some $t$ with $b_f(t) < 0$;
6. return $(f, \pi);$
Figure 5: Illustration of Algorithm 2: (a) An instance of the min-cost flow problem associated with the grid-
leveling problem on a $2 \times 4$-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.

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

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

Assume now that $e \in \bar{E} \setminus E_N$ 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 

$$\sum_{e \in E} f(e) d_{\pi}(e) = \sum_{e \in E} f(e) d_{\pi}(e)$$

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 $O(n)$ edges, capacities at most $c_{\text{max}}$ and node 
degrees at most $\Delta$ can be computed in $O(c_{\text{max}} \sqrt{\Delta} n^{3/2})$ 
time.

**Proof.** In the following, we denote the apex by $\hat{v}$ and we 
write $b(V^i) := \sum_{v \in V^i} b(v)$ for $V^i \subseteq V$. We may assume 

\[ \text{w.l.o.g. that there are edges } (v, \hat{v}), (\hat{v}, v) \text{ with infinite}\]
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 lie in parallel to existing edges but still the 
number of edges remains in $O(n)$. 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 $N = (\hat{G} = (V \cup \{\hat{v}\}, \bar{E}), c, b, d)$ 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 
time [4] such that $|V_i| \leq \frac{3}{4}|V|, i = 1, 2$, 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 \( O(\sqrt{\Delta n}) \) 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 \( \pi_i \) on \( \hat{G}_i \). Note that we modify \( b(\hat{v}) = -b(V_1) \) 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 \( b(\hat{v}) = 0 \) 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 \( \sum_{\hat{e} \in E(V_1, V_2)} c(\hat{e}) \in O(c_{\text{max}} \sqrt{\Delta n}) \). Hence, there are at most \( O(c_{\text{max}} \sqrt{\Delta n}) \) iterations within \textsc{SuccessiveShortestPath}. Since each shortest-path computation can be performed in linear time \( [21] \) each recursive step and, hence, the whole algorithm finishes in \( O(c_{\text{max}} \sqrt{\Delta n} n^{3/2}) \) 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 \( O(nN \log N) \) 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 \( c_{\text{max}} = 1 \) 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 \( O(nN \log N + Nn^{3/2}) \) time.

Note that a balanced cut with at most \( O(N \sqrt{n}) \) 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 is 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 average than the interior point method to solve the equiva-
lent LP for images up to a size of $512 \times 512$. Although
the fitted exponent for the combinatorial algorithm is
larger than the one for CPLEX, the two curves inter-
sect 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 more-
ever, the memory requirement for CPLEX is more than
1 GB for images of size $512 \times 512$. 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 op-
timum 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 $512 \times 512$ 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 meaning-
ful way. However, it is tempting to say that our ex-
periments suggest that even a non-recursive successive
shortest path algorithm may achieve an $O(n^{3/2})$ run-
ing 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 inter-
esting 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
is easy to generate a new one for each child instance.
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.

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


