On Testing Convexity and Submodularity*

Michal Parnas
The Academic College
of Tel-Aviv-Yaffo
Tel-Aviv, Israel
michalp@mta.ac.il

Dana Ron†
Department of EE – Systems
Tel-Aviv University
Ramat Aviv, Israel
danar@eng.tau.ac.il

Ronitt Rubinfeld
NEC Research Institute
Princeton, NJ
ronitt@research.nj.nec.com

May 19, 2003

Abstract

Convex and Submodular functions play an important role in many applications, and in particular in combinatorial optimization. Here we study two special cases: convexity in one dimension and submodularity in two dimensions. The latter type of functions are equivalent to the well known *Monge matrices*. A matrix $V = \{v_{i,j}\}_{i,j=0}^{i=n_1,j=n_2}$ is called a Monge matrix if for every $0 \leq i < i' \leq n_1$ and $0 \leq j < j' \leq n_2$, we have $v_{i,j} + v_{i',j'} \leq v_{i,j'} + v_{i',j}$. If inequality holds in the opposite direction then V is an *inverse Monge* matrix (supermodular function). Many problems, such as the traveling salesperson problem and various transportation problems, can be solved more efficiently if the input is a Monge matrix.

In this work we present testing algorithms for the above properties. A Testing algorithm for a predetermined property $\mathcal{P}$ is given query access to an unknown function f, and a distance parameter ϵ. The algorithm should accept f with high probability if it has the property $\mathcal{P}$, and reject it with high probability if more than an ϵ-fraction of the function values should be modified so that f obtains the property. Our algorithm for testing whether a one-dimensional function $f : [n] \rightarrow \mathbb{R}$ is convex (concave), has query complexity and running time of $O((\log n)/\epsilon)$. Our algorithm for testing whether an $n_1 \times n_2$ matrix V is a Monge (inverse Monge) matrix has query complexity and running time of $O((\log n_1 \cdot \log n_2)/\epsilon)$.

*Part of this research was done while the first two authors were visiting NEC Resarch Institute.

†Supported by the Israel Science Foundation (grant number 32/00-1).

1 Introduction

Convex functions and their combinatorial analogs, submodular functions, play an important role in many disciplines and applications, including combinatorial optimization, game theory, probability theory, and electronic trade. Such functions exhibit a rich mathematical structure (see Lovász [Lov83]), which often makes it possible to efficiently find their minimum [GLS81, IFF01, Sch00], and thus leads to efficient algorithms for many important optimization problems.

Convex functions over discrete domains are defined as follows.

Definition 1 (Convex and Concave) *Let f be a function defined over a discrete domain X. The function f is* convex *if for all $x, y \in X$ and for all $0 \leq \alpha \leq 1$ such that $\alpha x + (1-\alpha)y \in X$, it holds that $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$. The function f is* concave *if for all $x, y \in X$ and for all $0 \leq \alpha \leq 1$ such that $\alpha x + (1-\alpha)y \in X$, it holds that $f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha) f(y)$.*

Submodular functions are defined as follows: Let $\mathcal{I} = I_1 \times I_2 \times \ldots \times I_d$, $d \geq 2$, be a product space where $I_q \subseteq \mathbb{R}$. In particular, we are interested in discrete domains $I_q = \{0, \ldots, n_q\}$. The *join* and *meet* operations are defined for every $x, y \in \mathcal{I}$:

$$(x_1, \ldots, x_d) \vee (y_1, \ldots, y_d) \stackrel{\text{def}}{=} (\max\{x_1, y_1\}, \ldots, \max\{x_d, y_d\})$$

and

$$(x_1, \ldots, x_d) \wedge (y_1, \ldots, y_d) \stackrel{\text{def}}{=} (\min\{x_1, y_1\}, \ldots, \min\{x_d, y_d\}) ,$$

respectively.

Definition 2 (Submodularity and Supermodularity) *A function $f : \mathcal{I} \to \mathbb{R}$ is* submodular *if for every $x, y \in \mathcal{I}$, $f(x \vee y) + f(x \wedge y) \leq f(x) + f(y)$. The function f is* supermodular *if for every $x, y \in \mathcal{I}$, $f(x \vee y) + f(x \wedge y) \geq f(x) + f(y)$.*

Certain subclasses of submodular functions are of particular interest. One such subclass is that of *submodular set* functions, which are defined over binary domains. That is, $I_q = \{0, 1\}$ for every $1 \leq q \leq d$, and so each $x \in \mathcal{I}$ corresponds to a subset of $\{1, \ldots, d\}$. Such functions are used for example in the scenario of combinatorial auctions on the internet (e.g. [dVV00],[LLN01]).

Another important subclass is the class of *Monge* functions, which are obtained when the domain is large but the dimension is $d = 2$. Since such functions are 2-dimensional, it is convenient to represent them as 2-dimensional matrices, which are referred to as *Monge matrices*. When the function is a 2-dimensional supermodular function the corresponding matrix is called an *inverse Monge matrix*.

The first problem that was shown to be solvable more efficiently if the underlying cost matrix is a Monge matrix is the classical Hitchcock transportation problem (see Hoffman [Hof63]). Since then it has been shown that many other combinatorial optimization problems can be solved more efficiently in this case (e.g. weighted bipartite matching, and NP-hard problems such as the traveling salesperson problem). See [BKR96] for a comprehensive survey on Monge matrices and their applications.

1.1 Testing Convexity and Submodularity

In this paper we approach the questions of convexity and submodularity from within the framework of property testing [RS96, GGR98]. (For surveys on property testing see [Ron01, Fis01].) Let f be a fixed but unknown function, and let $\mathcal{P}$ be a fixed property of functions (such as the convexity or submodularity of a function). A testing algorithm for the property $\mathcal{P}$ should determine, by querying f, whether f has the property $\mathcal{P}$, or whether it is ϵ-*far* from having the property for a given distance parameter ϵ. By ϵ-*far* we mean that more than an ϵ–fraction of the values of f should be modified so that f obtains the desired property $\mathcal{P}$.

Our Results. We present efficient testing algorithms for discrete convexity in one dimension and for Monge matrices. Specifically:

- We describe and analyze an algorithm that tests whether a function $f : [n] \to \mathbb{R}$ is convex (concave). The running time of this algorithm is $O(\log n/\epsilon)$.
- We describe and analyze a testing algorithm for Monge and inverse Monge matrices whose running time is $O((\log n_1 \cdot \log n_2)/\epsilon)$, when given an $n_1 \times n_2$ matrix.

  Furthermore, the testing algorithm for inverse Monge matrices can be used to derive a testing algorithm, with the same complexity, for an important sub-family of Monge matrices, named *distribution matrices*. A matrix $V = \{v_{i,j}\}$ is said to be a distribution matrix, if there exists a non-negative *density matrix* $D = \{d_{i,j}\}$, such that every entry $v_{i,j}$ in V is of the form $v_{i,j} = \sum_{k \le i} \sum_{\ell \le j} d_{k,\ell}$. In other words, the entry $v_{i,j}$ corresponds to the cumulative density of all entries $d_{k,\ell}$ such that $k \le i$ and $\ell \le j$.

In both cases the complexity of the algorithms is linear in $1/\epsilon$ and polylogarithmic in the size of the domain.

1.2 Techniques

Convexity in One Dimension. We start with the following basic observation: A function $f : [n] \to \mathbb{R}$ is convex if and only if for every $1 \le i \le n-1$, $(f(i+1)-f(i))-(f(i)-f(i-1)) \ge 0$. Given this characterization, consider the *difference* function f' which is defined as $f'(i) = f(i) - f(i-1)$. The function f' can be viewed as the discrete analog of the first derivative of f. By the above observation we have that f is convex if an only if f' is monotone non-decreasing. Hence, a tempting approach for testing whether f is convex would be to test whether f' is monotone non-decreasing, where this can be done in time $O(\log n/\epsilon)$ [EKK$^+$00, BRW99, DGL$^+$99].

Unfortunately this approach does not work. There are functions f that are very far from convex but their difference function f' is very close to monotone.[1] Therefore, instead of considering only consecutive points $i, i+1$, we consider pairs of points $i, j \in [n]$ that are not necessarily consecutive. More precisely, we select intervals $\{i, \ldots, j\}$ of varying lengths and check that for each interval selected, certain constraints are satisfied. If f is convex then these constraints are satisfied for every interval. On the other hand, we show that if f is ϵ-far from convex then the probability that we observe a violation of some constraint is sufficiently large.

Monge Matrices. As stated above, it is convenient to represent 2-dimensional submodular functions as 2-dimensional Monge matrices. Thus a function $f : \{0, \ldots, n_1\} \times \{0, \ldots, n_2\} \to \mathbb{R}$ can be

[1] In particular consider the function f such that for every $i \le n/2$, $f(i) = i$, and for $i > n/2$, $f(i) = i - 1$. In other words, $f'(i) = 1$ for every i except $i = n/2$ where $f'(i) = 0$. Then f' is very close to monotone, but it is not hard to verify that f is far from convex.

represented as the matrix $V = \{v_{i,j}\}_{i,j=0}^{i=n_1,j=n_2}$ where $v_{i,j} = f(i,j)$. Observe that for every pair of indices $(i,j'),(i',j)$ such that $i < i'$ and $j < j'$ we have that $(i,j') \vee (i',j) = (i',j')$ and $(i,j') \wedge (i',j) = (i,j)$. It follows from Definition 2 that V is a Monge matrix (f is a 2-dimensional submodular function) if and only if:

$$\forall i,j,i',j' \quad \text{s.t.} \quad i < i',\ j < j' \ : \qquad v_{i,j} + v_{i',j'} \leq v_{i,j'} + v_{i',j}$$

and V is an inverse Monge matrix (f is a 2-dimensional supermodular function) if and only if:

$$\forall i,j,i',j' \quad \text{s.t.} \quad i < i',\ j < j' \ : \qquad v_{i,j} + v_{i',j'} \geq v_{i,j'} + v_{i',j}\ .$$

That is, in both cases we have a constraint for every quadruple $v_{i,j},\ v_{i',j'},\ v_{i,j'},\ v_{i',j}$ such that $i < i'$ and $j < j'$.[2] Our algorithm selects such quadruples according to a particular (non-uniform) distribution and verifies that the constraint is satisfied for every quadruple selected. Clearly the algorithm always accepts Monge matrices. The main thrust of the analysis is in showing that if the matrix V is far from being Monge then the probability of obtaining a "bad" quadruple is sufficiently large.

A central building block in proving the above, is the following combinatorial problem, which may be of independent interest. Let C be a given matrix, possibly containing negative values, and let R be a subset of positions in C. We are interested in refilling the entries of C that reside in R with *non-negative* values, such that the following constraint is satisfied: for every position (i,j) that does not belong to R, the sum of the modified values in C that are below[3] (i,j), is the same as in the original matrix C. That is, the sum of the modified values in entries (k,ℓ), such that $k \leq i$ and $j \leq \ell$, remains as it was.

We provide sufficient conditions on C and R under which the above is possible, and describe the corresponding procedure that refills the entries of C that reside in R. Our starting point is a simple special case in which R corresponds to a sub-matrix of C. In such a case it suffices that for each row and each column in R, the sum of the corresponding entries in the original matrix C is non-negative. Under these conditions a simple greedy algorithm can modify C as required. Our procedure for general subsets R is more involved but uses the sub-matrix case as a subroutine.

1.3 Further Research

We suggest the following open problems. First it remains open to determine the complexity of testing discrete convexity (concavity) when the dimension d of the input domain is greater than 1, and for testing submodular (supermodular) functions when the dimension d is greater than 2. Note that though submodular functions can be viewed as a certain interpretation of convexity in dimensions $d \geq 2$, they do not necessarily satisfy Definition 1.

It seems that our algorithm for testing Monge matrices and its analysis can be extended to work for testing the special case of distribution matrices of dimension $d > 2$, where the complexity of the resulting algorithm is $O\left((\prod_{q=1}^{d} \log n_q)/\epsilon\right)$. However, as opposed to the $d = 2$ case, where Monge matrices are only slightly more general than distribution matrices, for $d > 2$ Monge matrices are more expressive. Hence it is not immediately clear how to adapt our algorithm to testing Monge matrices in higher dimensions.

[2] It is easy to verify that for all other i,j,i',j' (with the exception of the symmetric case where $i' < i$ and $j' < j$), the constraint holds trivially (with equality).

[3] We denote the lower left position of the matrix C by $(0,0)$.

It would also be interesting to find an efficient testing algorithm for the subclass of submodular set functions, which are defined over binary domains.

Finally, in many optimization problems it is enough that the underlying cost matrix is a permutation of a Monge matrix. In such cases it may be useful to test whether a given matrix is a permutation of some Monge matrix or far from any permuted Monge matrix.

Organization. The testing algorithm for convexity is described in Section 2. The remainder of the paper is dedicated to testing Monge matrices. In Section 3 we describe several building blocks that will be used by our testing algorithm for Monge matrices. In Section 4 we describe a testing algorithm for Monge matrices whose complexity is $O(n/\epsilon)$, where we assume for simplicity that the matrix is $n \times n$. Building on this algorithm and its analysis, in Section 5 we present a significantly faster algorithm whose complexity is $O\left((\log^2 n)/\epsilon\right)$. We conclude this section with a short discussion concerning distribution matrices.

2 Testing Convexity in 1-Dimension

As noted in the introduction, in the case that the domain is $X = [n] = \{0, \ldots, n\}$, we get the following characterization for convexity, whose proof is included for completeness.

Claim 1 (1-D Convex) *A function $f : [n] \to \mathbb{R}$ is* convex *if and only if for all $1 \le i \le n-1$, $f(i) - f(i-1) \le f(i+1) - f(i)$.*

Proof: If f is convex then in particular for $x = i-1$, $y = i+1$ and $\alpha = 1/2$ we have $\alpha x + (1-\alpha)y = \frac{i-1}{2} + \frac{i+1}{2} = i$. By Definition 1, $f(i) \le \frac{1}{2}f(i-1) + \frac{1}{2}f(i+1)$, or equivalently, $f(i) - f(i-1) \le f(i+1) - f(i)$.

In the other direction, suppose that $f(i) - f(i-1) \le f(i+1) - f(i)$ for every $1 \le i \le n-1$. Consider any $x, y \in [n]$ and $0 < \alpha < 1$ such that $z = \alpha \cdot x + (1-\alpha) \cdot y$ is an integer. Assume without loss of generality that $x < y$. Now we have that

$$f(y)-f(y-1) \ge f(y-1)-f(y-2) \ge \ldots \ge f(z+1)-f(z) \ge f(z)-f(z-1) \ge \ldots \ge f(x+1)-f(x) .$$

Then, since the differences are monotone non-increasing, the average of the first $\alpha(y-x)$ differences is greater or equal to the average of the next $(1-\alpha)(y-x)$ differences. Since $z = y - \alpha(y-x) = x + (1-\alpha)(y-x)$, we have that

$$\frac{(f(y) - f(y-1)) + (f(y-1) - f(y-2)) + \ldots + (f(z+1) - f(z))}{\alpha(y-x)} \tag{1}$$

$$\ge \frac{(f(z) - f(z-1)) + (f(z-1) - f(z-2)) + \ldots + (f(x+1) - f(x))}{(1-\alpha)(y-x)} . \tag{2}$$

This is equivalent to $(1-\alpha)(f(y) - f(z)) \ge \alpha(f(z) - f(x))$, that is $f(z) \le \alpha f(x) + (1-\alpha)f(y)$, as required. ■

Denote by $I_{i,j}$ the interval $\{i, i+1, \ldots, j\}$ of points. Let $mid = \lfloor (i+j)/2 \rfloor$ be the mid point of $I_{i,j}$.

Definition 3 *For every $0 \le i < j \le n$ such that $j - i > 7$, we say that the interval $I_{i,j}$ is* good with respect to f *if the following holds:*

$$\begin{aligned} f(i+1) - f(i) \;&\le\; \frac{f(mid-1) - f(i+1)}{(mid-1) - (i+1)} \;\le\; f(mid) - f(mid-1) \;\le\; f(mid+1) - f(mid) \\ &\le\; f(mid+2) - f(mid+1) \;\le\; \frac{f(j-1) - f(mid+2)}{(j-1) - (mid+2)} \;\le\; f(j) - f(j-1) \end{aligned}$$

Otherwise we say that the interval is bad with respect to f. *If* $j-i \leq 7$, *then* $I_{i,j}$ *is good with respect to* f *if and only if the function* f *is convex over* $I_{i,j}$.

In order to test if f is convex we test recursively if sub-intervals of $I_{0,n}$ are good.

Algorithm 1 Test-Convex

1. *Repeat* $2/\epsilon$ *times:* Test-Interval($I_{0,n}$).
2. *If all of the tests in Step 1 accepted then accept, otherwise reject.*

Procedure Test-Interval($I_{i,j}$)

1. *Check that* $I_{i,j}$ *is good with respect to* f. *In not, reject.*
2. *If* $j-i > 7$ *then: Uniformly at random call either* Test-Interval($I_{i,mid}$) *or* Test-Interval($I_{mid+1,j}$), *where* $mid = \lfloor (i+j)/2 \rfloor$.
3. *If the test in Step 2 accepted then accept, otherwise reject.*

Theorem 1 *If* f *is convex then Algorithm 1 always accepts, and if* f *is* ϵ*-far from convex then the algorithm rejects with a probability of at least* $2/3$.

Proof: For the sake of brevity, unless stated otherwise, when we say that an interval is good, then we mean with respect to f. If f is convex then all intervals $I_{i,j}$ are good, and hence Algorithm 1 accepts with probability 1. In order to prove that if f is ϵ-far from convex then the algorithm rejects with probability of at least 2/3, we prove the contrapositive statement. Assume that the algorithm accepts with a probability greater than a 1/3. We will show that f is ϵ-close to a convex function.

To this end we define a tree, whose vertices correspond to all possible intervals $I_{i,j}$ that may be tested recursively in calls to Test-Interval($I_{i,j}$). Specifically, the root of the tree corresponds to $I_{0,n}$. The children of the internal vertex corresponding to $I_{i,j}$ are the vertices corresponding to $I_{i,mid}$ and $I_{mid+1,j}$, where $mid = \lfloor (i+j)/2 \rfloor$. The leaves of the tree correspond to the smallest intervals tested, that is, intervals $I_{i,j}$ for which $j-i \leq 7$.

We say that an internal vertex in the tree is good if the corresponding interval is good. We say that a leaf is good if its corresponding interval and all its ancestors are good. Otherwise, the vertex (leaf) is bad. We say that a path from the root to a leaf is good if all vertices along it are good. Otherwise the path is bad. For each level ℓ in the tree, $\ell = 0, \ldots, \log n$, let $\mathcal{B}_\ell$ be the subset of vertices in the ℓ'th level of the tree that are bad but whose ancestors are all good. Let $\mathcal{B} = \bigcup_\ell \mathcal{B}_\ell$, and let ϵ_ℓ be the fraction of vertices in level ℓ of the tree that belong to $\mathcal{B}_\ell$.

Sub-Claim 1 *If Algorithm 1 accepts* f *with a probability greater than a* $1/3$, *then* $\sum_\ell \epsilon_\ell \leq \epsilon$.

Proof: Assume by contradiction that $\sum_\ell \epsilon_\ell > \epsilon$. Observe that by the definition of $\mathcal{B}$, all leaves which are descendents of a vertex in $\mathcal{B}$ are bad, and every bad leaf either belongs to $\mathcal{B}$ or has a single ancestor in $\mathcal{B}$. Therefore, if $\sum_\ell \epsilon_\ell > \epsilon$, then the fraction of bad leaves is greater than ϵ. But in such a case, the probability that the algorithm does not follow a bad path to a bad leaf (passing through a vertex in $\mathcal{B}$), in any one of its $2/\epsilon$ iterations, is at most $(1-\epsilon)^{2/\epsilon} < e^{-2} < 1/3$. This contradicts our assumption that the algorithm accepts with a probability greater than a 1/3. $\square$

Hence we assume from now on that $\sum_\ell \epsilon_\ell \leq \epsilon$. Note also that in this case $I_{0,n} \notin \mathcal{B}$. We show how to modify f in at most $\epsilon \cdot n$ places so that the resulting function, denoted g, is convex. In particular, we shall modify the value of f on every bad interval $I_{i,j}$ whose corresponding vertex in the tree belongs to $\mathcal{B}$. The value of g is defined to be the same as the value of f on all points outside of these intervals. Since $\sum_\ell \epsilon_\ell \leq \epsilon$, the total fraction of points modified is at most ϵ as required. Observe that by the definition of the tree and $\mathcal{B}$, for every two intervals whose corresponding vertices belong to $\mathcal{B}$, the intersection of the intervals is empty. Hence we can modify each one of these intervals independently.

Let $I_{i,j}$ be a bad interval corresponding to a vertex in $\mathcal{B}$. We modify f on points in $I_{i,j}$ as follows:

- $f(i), f(i+1), f(j-1)$ and $f(j)$ remain unchanged. That is, set $g(i) = f(i)$, $g(i+1) = f(i+1)$, $g(j-1) = f(j-1)$ and $g(j) = f(j)$.
- For every t, $i+1 < t < j-1$, set $g(t) = f(i+1) + \frac{f(j-1)-f(i+1)}{(j-1)-(i+1)} \cdot (t-(i+1))$.

Sub-Claim 2 *Let $I_{i,j}$ be a bad interval corresponding to a vertex in $\mathcal{B}$. Then for every $i < t < j$, $g(t) - g(t-1) \leq g(t+1) - g(t)$.*

Proof: By definition of $\mathcal{B}$, the parent of $I_{i,j}$ is good (the parent exists by our assumption that $I_{0,n} \notin \mathcal{B}$). Hence

$$f(i+1) - f(i) \leq \frac{f(j-1) - f(i+1)}{(j-1)-(i+1)} \leq f(j) - f(j-1). \tag{3}$$

By definition of $g(\cdot)$, $g(i+1) - g(i) = f(i+1) - f(i)$, $g(j) - g(j-1) = f(j) - f(j-1)$, and for every $i+1 < t \leq j-1$, $g(t) - g(t-1) = \frac{f(j-1)-f(i+1)}{(j-1)-(i+1)}$. Therefore, for every $i+1 < t < j-1$, $g(t) - g(t-1) = g(t+1) - g(t)$, and for both $t = i+1$ and $t = j-1$, we have $g(t) - g(t-1) \leq g(t+1) - g(t)$, as required. $\square$

Sub-Claim 3 *The function g is convex.*

Proof: We shall first show that all intervals $I_{i,j}$ corresponding to vertices in the tree are good with respect to g, and from this derive the convexity of g.

We start with the first part. Consider any such interval $I_{i,j}$ whose corresponding vertex in the tree is v. Let $Anchor = \{i, i+1, mid-1, mid, mid+1, mid+2, j-1, j\}$ be the set of points which participate in the definition of a good interval $I_{i,j}$. We will show that the value of g on points $p \in Anchor$ is such that the interval $I_{i,j}$ is good with respect to g. There are two cases:

1. The interval $I_{i,j}$ is good with respect to f, and v does not have any ancestors in $\mathcal{B}$. If v also has no descendents in $\mathcal{B}$, then it clearly remains good with respect to g, since no modification is performed on any point in the interval, and so $g(t) = f(t)$ for every $i \leq t \leq j$. Otherwise, v has a descendent in $\mathcal{B}$. In this case, let $p \in Anchor$, let v' be a descendent of v, and let $I_{i',j'}$ denote the interval corresponding to v'. If $i' \leq p \leq j'$, then by definition of the tree, either $p = i'$ or $p = i'+1$ or $p = j'-1$ or $p = j'$. Therefore, even if $v' \in \mathcal{B}$ and the interval $I_{i',j'}$ is modified, then by the definition of g we have that $g(p) = f(p)$ for every $p \in Anchor$. Thus $I_{i,j}$ remains good with respect to g.
2. Either $v \in \mathcal{B}$ or v has an ancestor in $\mathcal{B}$. In the former case, let $v' = v$, and in the latter case let v' be the ancestor that v has in $\mathcal{B}$. Let $I_{i',j'}$ be the corresponding interval of v'. By definition, $I_{i,j} \subseteq I_{i',j'}$. By Sub-Claim 2, $g(t) - g(t-1) \leq g(t+1) - g(t)$ for every $i' < t < j'$, and in particular for every $i < t < j$. It follows that $I_{i,j}$ is good with respect to g.

Hence all intervals corresponding to vertices in the tree are good with respect to g. We now prove that for every $0 < t < n$ it holds that $g(t) - g(t-1) \leq g(t+1) - g(t)$, and thus g is convex. Let $I_{i,j}$ be the smallest interval in the tree such that $i < t < j$. If $j - i \leq 7$ then we are done, since the goodness of $I_{i,j}$ in this case means that g is convex over the whole interval. Otherwise, either $t = mid$ or $t = mid + 1$, where $mid = \lfloor (i+j)/2 \rfloor$. To verify this, note that if this were not the case then either $i < t < mid$ or $mid + 1 < t < j$. Hence t is contained in a smaller interval in the tree, contradicting the minimality of $I_{i,j}$. But since $I_{i,j}$ is good with respect to g, $g(mid) - g(mid-1) \leq g(mid+1) - g(mid)$, and $g(mid+1) - g(mid) \leq g(mid+2) - g(mid+1)$. Thus we are done with the proof of Sub-Claim 3, and Theorem 1 follows. ■

3 Building Blocks for Our Algorithms for Testing Inverse Monge

From this point on we focus on inverse Monge matrices. Analogous claims hold for Monge matrices. We also assume for simplicity that the dimensions of the matrices are $n_1 = n_2 = n$. In what follows we provide a characterization of inverse Monge matrices that is exploited by our algorithms. Given any real valued matrix $V = \{v_{i,j}\}_{i,j=0}^{i,j=n}$ we define an $(n+1) \times (n+1)$ matrix $C'_V = \{c_{i,j}\}_{i,j=0}^{i,j=n}$ as follows:

- $c_{0,0} = v_{0,0}$;
- For $i > 0$: $c_{i,0} = v_{i,0} - v_{i-1,0}$;
- For $j > 0$: $c_{0,j} = v_{0,j} - v_{0,j-1}$;
- And for every $i, j > 0$:

$$\begin{aligned} c_{i,j} &= (v_{i,j} - v_{i-1,j}) - (v_{i,j-1} - v_{i-1,j-1}) \\ &= (v_{i,j} - v_{i,j-1}) - (v_{i-1,j} - v_{i-1,j-1}). \end{aligned} \tag{4}$$

Let $C_V = \{c_{i,j}\}_{i,j=1}^{i,j=n}$ be the sub-matrix of C'_V that includes all but the first (0'th) row and column of C'_V. The following two claims are well known and easy to verify. We include their proofs for completeness.

Claim 2 *For every* $0 \leq i, j \leq n$, $v_{i,j} = \sum_{k=0}^{i} \sum_{\ell=0}^{j} c_{k,\ell}$.

Proof: The claim is proved by induction on i and j.
The base case, $i, j = 0$ holds by definition of $c_{0,0}$.
Consider any $i > 0$ and assume that the claim holds for every $k < i$, $j = 0$. We prove it for i and for $j = 0$. By definition of $c_{i,0}$ we have $v_{i,0} = v_{i-1,0} + c_{i,0}$. By the induction hypothesis, $v_{i-1,0} = \sum_{k=0}^{i-1} c_{k,0}$, and the induction step follows. The claim is similarly proved for every $j > 0$ and $i = 0$.
Finally, consider any $i, j > 0$ and assume that the claim holds for every $k < i$ and $\ell \leq j$, and for every $k \leq i$ and $\ell < j$. We prove it for i, j. By definition of $c_{i,j}$, $v_{i,j} = v_{i-1,j} + (v_{i,j-1} - v_{i-1,j-1}) + c_{i,j}$. By the induction hypothesis,

$$v_{i-1,j} + (v_{i,j-1} - v_{i-1,j-1}) = \sum_{k=0}^{i-1} \sum_{\ell=0}^{j} c_{k,\ell} + \sum_{\ell=0}^{j-1} c_{i,\ell}$$

and the induction step follows. ■

Claim 3 *A matrix V is an inverse Monge matrix if and only if C_V is a non-negative matrix.*

Proof: If V is an inverse Monge matrix, then in particular, for every $i, j \geq 1$ we have that $v_{i,j} + v_{i-1,j-1} \geq v_{i,j-1} + v_{i-1,j}$, which is equivalent to the condition $c_{i,j} \geq 0$.

In the other direction, consider any two points (i,j) and (i',j') such that $0 \leq i < i' \leq n$, $0 \leq j < j' \leq n$. Using Claim 2 we obtain

$$\begin{aligned} & v_{i',j'} - v_{i',j} - v_{i,j'} + v_{i,j} \\ = & \sum_{k=0}^{i'}\sum_{\ell=0}^{j'} c_{k,\ell} - \sum_{k=0}^{i'}\sum_{\ell=0}^{j} c_{k,\ell} - \sum_{k=0}^{i}\sum_{\ell=0}^{j'} c_{k,\ell} + \sum_{k=0}^{i}\sum_{\ell=0}^{j} c_{k,\ell} \\ = & \sum_{k=i+1}^{i'}\sum_{\ell=j+1}^{j'} c_{k,\ell} \end{aligned} \tag{5}$$

But C_V is non-negative and therefore $v_{i',j'} - v_{i',j} - v_{i,j'} + v_{i,j} \geq 0$ as required. ■

It follows from Claim 3 that if we find some entry of C_V that is negative, then we have evidence that V is not an inverse Monge matrix. However, it is not necessarily true that if V is far from being an inverse Monge matrix, then C_V contains many negative entries. For example, suppose that C_V is 1 in all entries except the entry $c_{n/2,n/2}$ which is $-n^2$. Then it can be verified that V is very far from being an inverse Monge matrix (this can be proved by showing that there are $\Theta(n^2)$ disjoint quadruples $v_{i,j}, v_{i',j'}, v_{i,j'}, v_{i',j}$ in V, such that from any such quadruple at least one value should be changed in order to transform V into an inverse Monge matrix). However, as our analysis will show, in such a case there are many sub-matrices in C_V whose sum of elements is negative. Thus our testing algorithms will sample certain sub-matrices of C_V and check that the sum of elements in each sub-matrix sampled is non-negative. We first observe that it is possible to check this efficiently.

Claim 4 *Given access to V it is possible to check in time $O(1)$ if the sum of elements in a given sub-matrix A of C_V is non-negative. In particular, if the lower-left entry of A is (i,j) and its upper-right entry is (i',j') then the sum of elements of A is $v_{i',j'} - v_{i',j-1} - v_{i-1,j'} + v_{i-1,j-1}$.*

Proof: Assume that $A = (c_{k,\ell})_{k=i,\ell=j}^{k=i',\ell=j'}$ is a sub-matrix of C_V. Recall that for any q, p, we have $v_{q,p} = \sum_{k=0}^{q}\sum_{\ell=0}^{p} c_{k,\ell}$. Thus the sum of elements of A is:

$$\begin{aligned} \sum_{k=i}^{i'}\sum_{\ell=j}^{j'} c_{k,\ell} &= \sum_{k=0}^{i'}\sum_{\ell=j}^{j'} c_{k,\ell} - \sum_{k=0}^{i-1}\sum_{\ell=j}^{j'} c_{k,\ell} \\ &= \left(\sum_{k=0}^{i'}\sum_{\ell=0}^{j'} c_{k,\ell} - \sum_{k=0}^{i'}\sum_{\ell=0}^{j-1} c_{k,\ell}\right) - \left(\sum_{k=0}^{i-1}\sum_{\ell=0}^{j'} c_{k,\ell} - \sum_{k=0}^{i-1}\sum_{\ell=0}^{j-1} c_{k,\ell}\right) \\ &= (v_{i',j'} - v_{i',j-1}) - (v_{i-1,j'} - v_{i-1,j-1}) \ . \end{aligned}$$

Therefore computing the sum of elements of any sub-matrix A of C_V, can be done by checking only 4 entries in the matrix V. ■

3.1 Filling Sub-matrices

An important building block for the analysis of our algorithms is a procedure for "filling in" a sub-matrix. That is, given constraints on the sum of elements in each row and column of a given sub-matrix, we are interested in assigning values to the entries of the sub-matrix so that these constraints are met.

Specifically, let $a_1, \ldots, a_s$ and $b_1, \ldots, b_t$ be non-negative real numbers such that $\sum_{i=1}^{s} a_i \geq \sum_{j=1}^{t} b_j$. Then it is possible to construct an $s \times t$ non-negative real matrix T, such that the sum of elements in column j is exactly b_j and the sum of elements in row i is at most a_i. In the special case that $\sum_{i=1}^{s} a_i = \sum_{j=1}^{t} b_j$, the sum of elements in row i will equal a_i. In particular, this can be done by applying the following procedure, which is the same as the one applied to obtain an initial feasible solution for the linear-programming formulation of the transportation problem.

Procedure 1 [Fill Matrix $T = (t_{i,j})_{i,j=1}^{i=s,j=t}$]

Initialize $\bar{a}_i = a_i$ for $i = 1, \ldots, s$ and $\bar{b}_j = b_j$ for $j = 1, \ldots, t$.
(In each of the following iterations, $\bar{a}_i$ is an upper bound on what remains to be filled in row i, and $\bar{b}_j$ is what remains to be filled in column j.)
for j = 1,...,t:
  for i = 1,...,s:
    Assign to entry (i,j) the value $x = \min\{\bar{a}_i, \bar{b}_j\}$
    Update $\bar{a}_i = \bar{a}_i - x$, $\bar{b}_j = \bar{b}_j - x$.

Claim 5 *Procedure 1 fills the matrix T with non-negative values $t_{i,j}$, such that at the end of the procedure, $\sum_{i=1}^{s} t_{i,j} = b_j$ for every $j = 1, \ldots, t$, and $\sum_{j=1}^{t} t_{i,j} \leq a_i$ for every $i = 1, \ldots, s$. If initially $\sum_{j=1}^{t} b_j = \sum_{i=1}^{s} a_i$ then $\sum_{j=1}^{t} t_{i,j} = a_i$ for every $i = 1, \ldots, s$.*

Proof: Notice that initially $\bar{a}_i = a_i \geq 0$ and $\bar{b}_j = b_j \geq 0$. Thus when we update $\bar{a}_i = \bar{a}_i - x = \bar{a}_i - \min\{\bar{a}_i, \bar{b}_j\} \geq 0$ and similarly $\bar{b}_j = \bar{b}_j - x = \bar{b}_j - \min\{\bar{a}_i, \bar{b}_j\} \geq 0$. Therefore the $\bar{a}_i$'s and $\bar{b}_j$'s are always non-negative. Hence all values x filled in T are non-negative, since $x = \min\{\bar{a}_i, \bar{b}_j\} \geq 0$. Furthermore, after each such update the new sum over the $\bar{a}_i$'s equals the old sum over the $\bar{a}_i$'s minus x and a similar statement holds for the sum over the $\bar{b}_j$'s. Thus at all stages of the procedure, $\sum_{i=1}^{s} \bar{a}_i \geq \sum_{j=1}^{t} \bar{b}_j$, and if initially $\sum_{i=1}^{s} a_i = \sum_{j=1}^{t} b_j$ then $\sum_{i=1}^{s} \bar{a}_i = \sum_{j=1}^{t} \bar{b}_j$.

We now show that the sum of elements in each column is as required. Observe that the procedure fills the columns one by one. Therefore when we start to fill column j we have $\bar{b}_j = b_j$. Since $\sum_{i=1}^{s} \bar{a}_i \geq \sum_{j=1}^{t} \bar{b}_j$ at this stage, and all $\bar{a}_i$'s are non-negative, necessarily, $\sum_{i=1}^{s} \bar{a}_i \geq \bar{b}_j = b_j$. Let $1 \leq k \leq s$ be the minimum integer such that $\sum_{i=1}^{k} \bar{a}_i \geq b_j$. Then by definition of the procedure, for every $i < k$, the entry (i,j) is filled with the value $\bar{a}_i$, and the entry $(k+1, j)$ is filled with the value $b_j - \sum_{i=1}^{k} \bar{a}_i$. The total is hence b_j as required.

As for the rows, at all stages $\bar{a}_i$ equals a_i minus the sum of all elements filled so far in row i. Therefore since $\bar{a}_i \geq 0$, then then sum of elements in row i is at most a_i. Furthermore, if initially $\sum_{i=1}^{s} a_i = \sum_{j=1}^{t} b_j$, then the sum of elements in row i will be exactly a_i. To show this note that at the end of the procedure, $\sum_{j=1}^{t} \bar{b}_j = 0$, since each $\bar{b}_j$ equals b_j minus the sum of all elements in column j, and we have shown that the sum of elements in column j is b_j. But $\sum_{i=1}^{s} \bar{a}_i = \sum_{j=1}^{t} \bar{b}_j$, and therefore also $\sum_{i=1}^{s} \bar{a}_i = 0$ at the end. Since $\bar{a}_i \geq 0$, this means that $\bar{a}_i = 0$. Hence the sum of elements in row i must be a_i. ■

4 A Testing Algorithm for Inverse Monge Matrices

We first present a simple algorithm for testing if a matrix V is an inverse Monge Matrix, whose running time is $O(n/\epsilon)$. In the next section we show a significantly faster algorithm that is partly based on the ideas presented here. We may assume without loss of generality that n is a power of 2. This is true since our algorithms probe the coefficients matrix C_V, and we may simply "pad" it by 0's to obtain rows and columns that have lengths which are powers of 2 and run the algorithm with $\epsilon \leftarrow \epsilon/4$. We shall need the following two definitions for both algorithms.

Definition 4 (Sub-Rows, Sub-Columns and Sub-Matrices) *A* sub-row *in an $n \times n$ matrix is a consecutive sequence of entries that belong to the same row. The sub-row $((i,j),(i,j+1),\ldots,(i,j+t-1))$ is denoted by $[\,]^{1,t}_{i,j}$. A* sub-column *is defined analogously, and is denoted by $[\,]^{s,1}_{i,j} = ((i,j),(i+1,j),\ldots,(i+s-1,j))$. More generally, an $s \times t$ sub-matrix whose bottom-left entry is (i,j) is denoted $[\,]^{s,t}_{i,j}$.*

Definition 5 (Legal Sub-Matrices) *A sub-row in an $n \times n$ matrix is a* legal *sub-row if it can result from bisecting the row of length n that contains it in a recursive manner. That is, a complete (length n) row is legal, and if $[\,]^{1,t}_{i,j}$ is legal, then so are $[\,]^{1,t/2}_{i,j}$ and $[\,]^{1,t/2}_{i,j+t/2}$. A legal sub-column is defined analogously. A sub-matrix is* legal *if both its rows and its columns are legal.*

Note that the legality of a sub-row $[\,]^{1,t}_{i,j}$ is independent of the actual row i it belongs to, but rather it depends on its starting position j and ending position $j+t-1$ within its row. An analogous statement holds for legal sub-columns. See also Figure 1 for an illustration of the concept of legal sub-matrices.

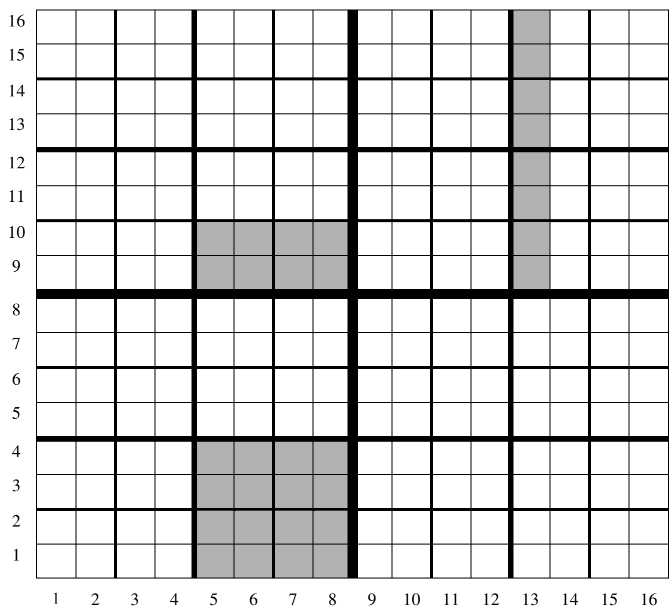

Figure 1: An illustration of three legal sub-matrices. One of the legal sub-matrices is a square (4×4) sub-matrix, and the other two are rectangular (but legal) sub-matrices. The 8×1 sub-matrix on the top-right is a legal sub-column

Although a sub-matrix is just a collection of positions (entries) in an $n \times n$ matrix, we talk throughout the paper about sums of elements in certain sub-matrices A of C_V. In this we mean the sum of elements of C_V determined by the set of positions in A.

Definition 6 (Good and Bad Sub-Matrices) *We say that a sub-matrix A of C_V is* good *if the sum of elements in each row of A is non-negative and the sum of elements in each column of A is non-negative. Otherwise, A is* bad*.*

Definition 7 (Good and Bad Points) *We say that point (i, j) is* good *if all legal* square *sub-matrices A of C_V which contain (i, j) are good. Otherwise, the point is* bad*.*

Algorithm 2 [Test Monge I]

1. *Choose $8/\epsilon$ points in the matrix C_V and check that they are good.*
2. *If all points are good then accept, otherwise reject.*

By Claim 4, it is possible to check in constant time that the sum of elements in a sub-row (sub-column) of C_V is non-negative. Therefore, it is possible to test that an $s \times s$ square sub-matrix A of C_V is good in time $\Theta(s)$. Notice that every point in an $n \times n$ matrix is contained in $\log n$ legal square sub-matrices. Hence the time required to check whether a point is good is $O(n) + O(n/2) + \ldots + O(n/2^i) + \ldots + O(1) = O(n)$, and the complexity of the algorithm is $O(n/\epsilon)$.

Theorem 2 *If V is an inverse Monge matrix then Algorithm 2 always accepts, and if V is ϵ-far from being an inverse Monge matrix, then Algorithm 2 rejects with probability at least $2/3$.*

Proof: The first part of the theorem follows directly from Claim 3. In order to prove the second part of the theorem, we show that if V is ϵ-far from being inverse Monge, then C_V contains more than $(\epsilon/4)n^2$ bad points. The second part of the theorem directly follows because the probability in such a case that no bad point is selected by the algorithm, is at most $(1 - \epsilon/4)^{(8/\epsilon)} < e^{-2} < 1/3$.

Assume contrary to the claim that C_V contains at most $(\epsilon/4)n^2$ bad points. We shall show that by modifying at most ϵn^2 entries in V we obtain an inverse Monge matrix (in contradiction to our assumption concerning V). Let us look at the set of bad points in C_V, and for each such bad point look at the largest bad legal square sub-matrix in C_V that contains this bad point. By our assumption on the number of bad points, it must be the case that the area of all these maximal bad sub-matrices is at most $(\epsilon/4)n^2$, because all the points in a bad sub-matrix are bad.

For each maximal bad legal square sub-matrix B of C_V we will look at the legal square sub-matrix A that contains B. By definition of legal square sub-matrices, the matrix A is uniquely defined. By the maximality of B, the sub-matrix A must be good. Indeed, since B is maximal, if it is of size $s \times s$ where $s < n$, then the legal square sub-matrix of size $2s \times 2s$ that contains it must be good. But if $s = n$, then $B = C_V$ implying that all n^2 points in C_V are bad, contradicting our assumption on the number of bad points.

Next observe that every two different maximal bad legal square sub-matrices B and B' are disjoint. This is true since every two different legal square sub-matrices are either disjoint, or one is contained in the other. Combining this with the fact that for each maximal bad legal square sub-matrix we take the good square legal sub-matrix that is 4 times its size, the area of the union of all these good sub-matrices is at most $4 \cdot (\epsilon/4)n^2 = \epsilon n^2$.

Turning to the collection of resulting good sub-matrices, note that every two of these sub-matrices are either disjoint, or are exactly the same, or one is contained in the other. If a good sub-matrix is strictly contained in another one, then we ignore it, and deal only with the larger good sub-matrix containing it. Thus we have a set of disjoint good sub-matrices that contain all negative entries in the matrix. For each of these good sub-matrices A, we modify A so that it contains only non-negative elements, and the sum of elements in each row and column of A remains as it was. This can be done by applying Procedure 1 to A as described in Section 3.1 (using the actual (non-negative) sums of rows and columns of A as the input to the procedure).

Note that after modifying all these good sub-matrices of C_V, the new matrix C_V is non-negative, and thus the corresponding new matrix V must be an inverse Monge matrix. It remains to show, that at most ϵn^2 values were changed in V following the changes to C_V. Notice that we made sure that the sum of elements in each row and column of each modified sub-matrix A remains as it was. Therefore the values of all points $v_{k,\ell}$ in V that are outside A are not affected by the change to A, since by Claim 2 we have that $v_{k,\ell} = \sum_{i=0}^{k} \sum_{j=0}^{\ell} c_{i,j}$. ■

5 A Faster Algorithm for Inverse Monge Matrices

Algorithm 2 described above has running time linear in n, which is already sub-linear in the size of the matrix, n^2. In this section we show how to significantly improve the dependence on n. We present a variant of the algorithm whose running time is $O(\epsilon^{-1} \log^2 n)$. The new algorithm will be based on a similar principle as that of Algorithm 2. That is, it will uniformly select points and verify that certain sub-matrices that contain them are good. However, there will be two main differences which we now describe briefly.

Algorithm 2 suffers from a relatively slow running time, since for each sub-matrix that the algorithm checks, it verifies that the sum of elements in *every* row and column is non-negative. Therefore, we first relax the concept of a good sub-matrix and demand only that the sum of *all* its elements be non-negative (instead of the sum of every row and column). This change however requires us to check for each point selected by the algorithm, not only that the legal *square* sub-matrices which contain it are good, but rather to verify that *all* legal sub-matrices that contain the point are good. Actually, we check something slightly stronger: The algorithm will verify for each legal sub-matrix T that it examines that the 4 legal equal-size sub-matrices that reside within T and are half of T's length in each dimension, are good as well. In order to formalize the above, we first redefine the concepts of good (bad) sub-matrices and good (bad) points, and introduce the notion of tainted sub-matrices and tainted points.

Definition 8 (Good and Bad Sub-Matrices and Points) *A (legal) sub-matrix T of C_V is* good *if the sum of all its elements is non-negative. Otherwise, T is* bad.

A point is good *if every legal sub-matrix of C_V that contains it is good. Otherwise the point is* bad.

Definition 9 (Tainted Sub-Matrices and Points) *A good legal sub-matrix T of C_V is* tainted *if any one of the four legal sub-matrices that it contains and are half its height and half its width is bad. A point is* tainted *if some legal sub-matrix that contains it is tainted.*

Note that every bad point is tainted, but good points may be tainted as well.

For the sake of the presentation, we shall assume that every row and every column in C_V (that is, every sub-row and sub-column of length n) have non-negative sums. In Subsection 5.2 we explain how to remove this assumption. Note that this assumption implies that every $s \times n$ sub-matrix is good, and similarly every $n \times s$ sub-matrix is good (but of course it has no implications on smaller sub-matrices).

Algorithm 3 [Test Monge II]

1. *Uniformly select $2/\epsilon$ points in the matrix C_V and check for each of them whether it is tainted.*

2. If no point selected is tainted then accept, otherwise reject.

Note that by Definition 5, each point in an $n \times n$ matrix is contained in $O(\log^2 n)$ legal sub-matrices. Thus by Claim 4, checking whether a point is tainted takes time $O(\log^2 n)$. Therefore the running time of the algorithm is $O((\log^2 n)/\epsilon)$.

Theorem 3 *If V is an inverse Monge matrix then Algorithm 3 always accepts, and if V is ϵ-far from being an inverse Monge matrix, then Algorithm 3 rejects with probability at least $2/3$.*

5.1 Outline of the Proof of Theorem 3

If V is an inverse Monge matrix then by Claim 3 all elements in C_V are non-negative. This directly implies that all (legal) sub-matrices and good, and so all points are good and are not tainted. Hence in this case the algorithm always accepts. Suppose that V is ϵ-far from being inverse Monge. We claim that in such a case C_V must contain more than ϵn^2 tainted points, causing the algorithm to reject with probability at least

$$1 - (1-\epsilon)^{(2/\epsilon)} > 1 - e^{-2} > 2/3.$$

Assume contrary to the claim that C_V contains at most ϵn^2 tainted points. Our goal from this point on is to show that in such a case V is ϵ-close to being an inverse Monge matrix.

The proof of this part will follow similar lines to those used in the proof of Theorem 2. That is, we consider all maximal bad legal sub-matrices of C_V, and for each such bad sub-matrix we consider the legal good sub-matrix that is 4 times its area and contains it. Once again, this sub-matrix is unique. By Definition 9, this sub-matrix is tainted. We then take the union of all these good but tainted sub-matrices. By our assumption on the number of tainted points, the area of this union is at most ϵn^2 since all points in the union are tainted.

Finally we show how to modify the values in this union, so that the resulting matrix is an inverse Monge matrix. This time however, since the maximal bad sub-matrices may intersect (which was not the case in the slower algorithm), the good tainted sub-matrices that contain them may intersect in non-trivial ways (that is, not only by coinciding or by strict containment). As a result, the union of the good sub-matrices has a possibly complex structure (and in particular it is no longer a simple union of disjoint sub-matrices), and the process of properly modifying this union is much more involved. We now describe precisely the necessary definitions and proceed with a detailed proof.

Definition 10 (Maximal bad legal sub-matrix) *A bad legal sub-matrix T of C_V is a* maximal bad legal sub-matrix *of C_V if it is not contained in any larger bad legal sub-matrix of C_V.*

Now consider all maximal bad legal sub-matrices of C_V. Note that every negative entry in C_V is contained in the union of these bad sub-matrices. For each such sub-matrix B let us take the (unique) legal sub-matrix T that contains it and has twice the number of rows and twice the number of columns of B (by our assumption that all full rows and columns have a non-negative sum it is indeed possible to double the rows and columns of B). Then by the maximality of B, the resulting sub-matrix is good. We now take the union of all these good (but tainted) legal sub-matrices. Recall that the area of the union of all tainted (legal) sub-matrices of C_V is at most ϵn^2. Denote the union of all these good tainted sub-matrices by R. See for example Figure 2.

In Subsections 5.3 and 5.4 we show that it is possible to change the (at most ϵn^2) entries of C_V within R to non-negative values so that the following property holds:

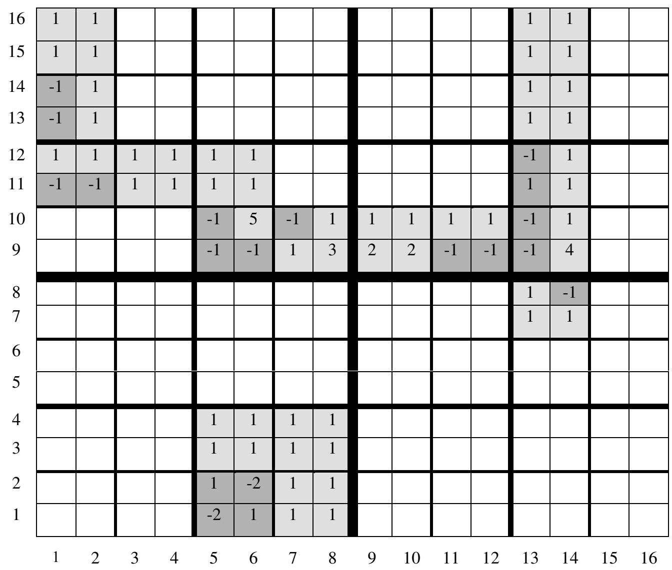

Figure 2: An example of the structure of a subset R, where R is the union of all gray cells in the matrix (both dark and light gray). All values in cells outside of R are non-negative, and are not displayed for sake of simplicity. The bad legal sub-matrices determining R are the dark gray sub-matrices. Each is contained inside a good but tainted legal sub-matrix that has twice the number of rows and twice the number of columns (good tainted sub-matrices are marked both by light and dark gray). For example, there is a bad sub-matrix in column 1, rows 13 and 14, and the good legal sub-matrix containing it is the sub-matrix over columns 1 and 2 and rows 13 through 16. Observe that maximal bad legal sub-matrices may intersect. For example, the bad sub-matrix containing the two cells in row 9 and columns 5 and 6 intersects with the bad sub-matrix containing the two cells in column 5 and rows 9 and 10. Their corresponding good sub-matrices also intersect.

Property 1 (Sum Property for R) *For every point (i, j) outside of R, the sum of the elements in the modified entries (i', j') within R such that $i' \leq i$ and $j' \leq j$ is the same as in the original matrix C_V.*

Let $\tilde{C}_V$ be the matrix obtained from C_V by modifying R so that Property 1 holds, and let $\tilde{V}$ be the matrix which corresponds to $\tilde{C}_V$. Then it follows from Claim 2 that $\tilde{V}$ is at most ϵ-far from the original matrix V, and this completes the proof of Theorem 3. Before we continue with showing how to obtain Property 1, we explain shortly how to remove the assumption that all (full) rows and columns in C_V have a non-negative sum.

5.2 Dealing with Rows/Columns Having a Negative Sum

Suppose first that $\epsilon \leq 4/n$. Then we may directly check in time $O(1/\epsilon)$ that in fact all rows and columns of the matrix C_V have non-negative sums (using Claim 4), and reject if some row or column has a negative sum. Hence in this case our assumption is valid. Thus assume that $\epsilon > 4/n$.

First we slightly modify Algorithm 3 so that it uniformly selects $4/\epsilon$ points in C_V (instead of $2/\epsilon$). In such a case, if C_V contains more than $(\epsilon/2)n^2$ tainted points then the algorithm rejects with probability at least $2/3$. We thus assume that C_V contains at most $(\epsilon/2)n^2$ tainted points and strive to show that in such a case V is ϵ-close to being an inverse Monge matrix. Since we do not assume that every row and column in C_V has a non-negative sum, we first modify C_V so that it has this property.

Consider each row i in C_V whose sum of elements in negative. Suppose that we modify the last entry in the row, $c_{i,n}$, so that the new sum of all elements is 0. Similarly, we modify the last entry $c_{n,j}$ in each column j that has a negative sum. Let $\bar{C}_V$ be the resulting matrix, and let $\bar{V}$ be the matrix corresponding to $\bar{C}_V$. Then all rows and columns in $\bar{C}_V$ have a non-negative sum, and by Claim (2) $\bar{V}$ and V differ on at most $2n-1 < (\epsilon/2)n^2$ entries (at most all elements in the last column and last row).

Now we may define the region R as we did in the previous subsection. Note that in this case the area of the region R is at most $(\epsilon/2)n^2$. We can therefore continue in proving that it is possible to modify only the elements within R so that they are all non-negative and Property 1 holds. This will imply that the total number of entries that should be modified (first to obtain non-negative rows and columns, and then to refill R) is at most ϵn^2, as desired.

5.3 Refilling R to Obtain Property 1

Let R be as defined in Section 5.1. Recall that R consists of a union of good legal sub-matrices. (The fact that they are tainted is no longer relevant.) In the following discussion, when we talk about elements in sub-matrices of R we mean the elements in C_V determined by the corresponding set of positions in R.

We are interested in refilling the entries in R with *non-negative* values, so that Property 1 will hold. Note that if R is just a sub-matrix (block) of C_V then we can use Procedure 1 to refill R as desired. However, in general the structure of R is more complex. We show that there is a way to partition R into disjoint blocks and refill each block using Procedure 1. In Subsection 5.3.1 we define precisely what blocks are and present several other notions that are needed for the refilling procedure. The refilling procedure for R is described in Subsection 5.3.2 and its correctness is proved in Subsection 5.4.

5.3.1 Preliminaries for the Refilling Procedure

As stated above, the refilling procedure will partition R into disjoint blocks (sub-matrices) and fill each block separately with non-negative values, so that Property 1 is maintained. We start with defining the following term that will be needed to define blocks.

Definition 11 (Maximal (legal) sub-row/column) *Given a subset R of entries in an $n \times n$ matrix, a sub-row T is a* maximal (legal) sub-row with respect to *R if T is contained in R and there is no larger (legal) sub-row T' such that $T \subset T' \subseteq R$. A* maximal (legal) sub-column with respect to *R is defined analogously.*

For sake of succinctness, whenever it is clear what R is, we shall just say maximal (legal) sub-row and drop the suffix, "with respect to R". Note that a maximal sub-row is simply a maximal consecutive sequence of entries in R that belong to the same row, while a maximal legal sub-row is a more constrained notion. In particular, a maximal sub-row may be a concatenation of several maximal legal sub-rows. We can now define blocks as follows.

Definition 12 (Maximal Block) *A maximal block $B = [\;]_{i,j}^{s,t}$ in R is a sub-matrix contained in R which has the following property: It consists of a maximal consecutive sequence of maximal legal sub-columns of the same height. The maximality of each sub-column is as in Definition 11. That is, for every $j \le r \le j+t-1$, the column $[\;]_{i,r}^{s,1}$ is a maximal legal sub-column (with respect to R).*

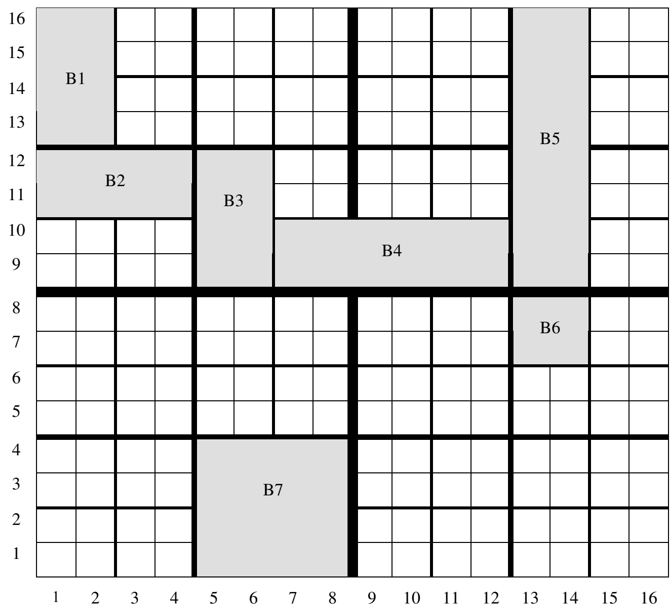

Figure 3: An example of the partition of R shown in Figure 2 into maximal blocks (numbered B_1–B_7). Note that the ratio between the heights of any two blocks is always a power of 2. Furthermore, the blocks are aligned in the following way. Suppose a block B has height s and a block B' has height $s' \leq s$ and some of their sub-rows belong to the same row of the matrix (e.g., B_3 and B_4, or B_4 and B_5). Then the shorter block B' must be aligned either with the first or second half of B, or with one of the quarters of B, or with one of its eighth's, etc.

The height *of a maximal block B is the height of the columns in B (equivalently, the number of rows in B).*

The maximality of the sequence of sub-columns in a block $B = [\ ]_{i,j}^{s,t}$ means that we cannot extend the sequence of columns neither to the left nor to the right. That is, neither $[\ ]_{i,j-1}^{s,1}$ nor $[\ ]_{i,j+t}^{s,1}$ are maximal legal sub-columns in R. (Specifically, each is either not fully contained in R or R contains a larger legal sub-column that contains it.)

We shall sometimes refer to maximal blocks simply as blocks. Observe that by this definition, R is indeed partitioned in a unique way into maximal disjoint blocks. See Figure 3 for an illustration to how the subset R from Figure 2 is partitioned into maximal blocks.

Three additional notions that will be needed for the refilling procedure are defined below. The first two are illustrated in Figure 4.

Definition 13 (Covers) *We say that a sub-matrix A* covers *a given block B with respect to R, if $B \subseteq A \subseteq R$ and the number of rows in A equals the height of B.*

We say that A is a maximal row-cover *with respect to R, if A consists of maximal sub-rows with respect to R.*

Definition 14 (Borders) *We say that a sub-matrix $T = [\ ]_{i,j}^{s,t}$* borders *another sub-matrix $T' = [\ ]_{i',j'}^{s',t'}$, if $i' \leq i + s - 1$ and $i \leq i' + s' - 1$, and either $j' = j + t$ (so that T is to the left of T'), or $j' + t' = j$ (so that T is to the right of T').*

Definition 15 (Sums) *For a given sub-matrix T, we denote the sum of the elements in T by $sum(T)$.*

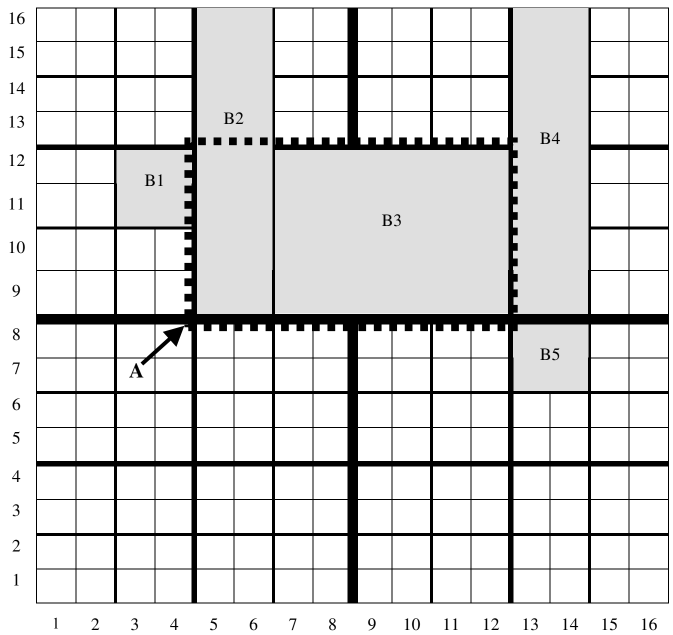

Figure 4: An illustration of the notions of covers and borders. Here the sub-matrix A (extending from row 9 to 12 and from column 5 to 12) covers the block B_3 (but is not a maximal row-cover with respect to R). The sub-matrix A borders block B_1 (from the left of A), and block B_4 (from the right of A).

5.3.2 The Procedure for Refilling R

We now describe the procedure that refills the entries of R with non-negative values so as to obtain Property 1. Recall that R is a disjoint union of maximal blocks. Hence if we remove a maximal block from R, then the maximal blocks of the remaining structure are simply the remaining maximal blocks of R. For simplicity of this introductory discussion, after removing a block from R, we refer to the remaining structure as R. The procedure described below will remove the blocks of R one by one, in order of increasing (non-decreasing) height, and refill each block separately using Procedure 1.

Recall that when (re)filling an $s \times t$ sub-matrix, Procedure 1 is provided with non-negative values $a_1, \ldots, a_s$ and $b_1, \ldots, b_t$ such that $\sum_{i=1}^{s} a_i \geq \sum_{j=1}^{t} b_j$. It then fills the sub-matrix with non-negative values so that the sum of elements in column j is exactly b_j and the sum of elements in row i is at most a_i. Whenever we apply Procedure 1 to a block B, the column sums $b_1, \ldots, b_t$ are simply set to be the sums of the elements in the corresponding sub-columns of B in C_V. By definition of (maximal) blocks, these sub-columns are maximal legal sub-columns, and as we show in Subsection 5.4.1, this ensures that their sums are non-negative.

The setting of the upper bounds $a_1, \ldots, a_s$ for the row sums is a little more involved. At any point in the algorithm, each maximal sub-row L is associated with a *designated* sum, denoted $\overline{sum}(L)$. This is the sum we intend it to have when the refilling procedure terminates. Initially, for every maximal sub-row L in R, we set $\overline{sum}(L) = sum(L)$. That is, $\overline{sum}(L)$ is equal to the original sum of sub-row L in C_V. In Subsection 5.4.1 we show that these sums are all non-negative. When refilling a block B, we first find the row-cover A of B that is a maximal row-cover with respect to (the current) R. Since the blocks are filled by order of height, and blocks are removed after they are filled, such a maximal row-cover must exist when B is covered, and is unique. We then use the designated sums of the (maximal) rows of A as the upper bounds $a_1, \ldots, a_s$ for the sums of rows of B. As we prove subsequently, it always holds that $\sum_{i=1}^{s} a_i \geq \sum_{j=1}^{t} b_j$, as required by Procedure 1. After removing a block B from R, we obtain new, shorter, maximal sub-rows in the remaining structure $R \setminus B$, and we must associate with these shorter sub-rows new designated sums. Procedure 1 is used here as well to determine how to set these designated row sums, in a manner explained in detail in Step 3c below.

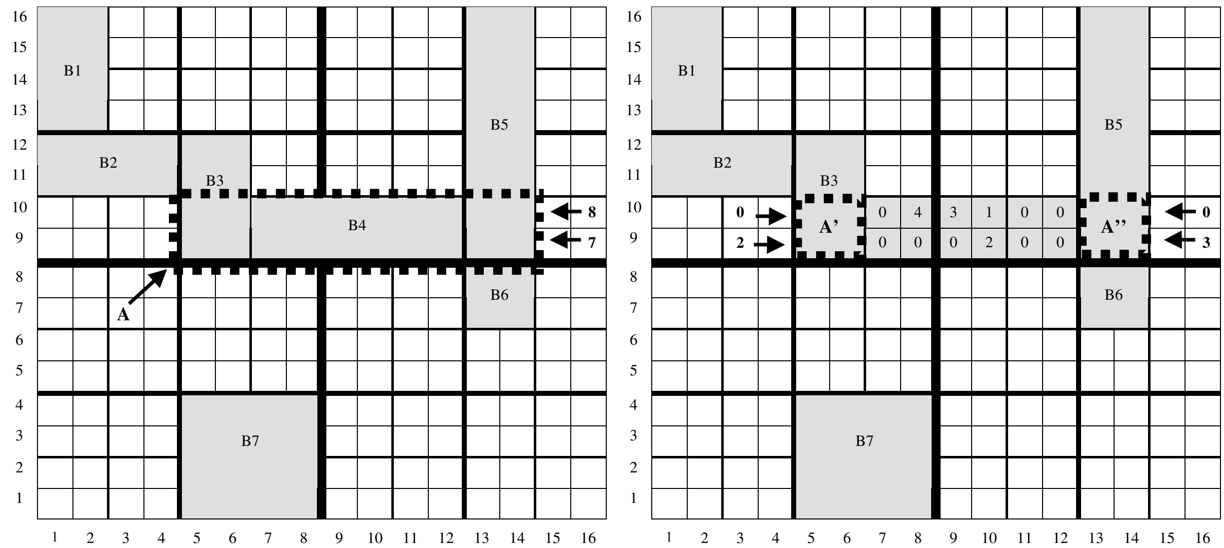

Figure 5: An illustration of one step of the refilling procedure, where we apply it to the matrix illustrated in Figures 2 and 3. The first block filled may be either B_2, B_4, or B_6 (all three have height 2, which is the minimum among all blocks). Here we have selected to refill B_4 first. On the left we see the maximal row-cover A that covers B_4, where the designated sums of the two rows of A are 8 and 7 (in accordance with the values appearing in Figure 2). On the right we see the values that the Procedure 1 has entered in the cells of B_4. We also see the two sub-matrices, A' and A'', that remain of A after B_4 is removed from R, and the designated sums of the new maximal rows in A' and A''.

Procedure 2 [Refill R]

1. *We assign each maximal sub-row L in R a designated sum of elements for that row, which is denoted by $\overline{sum}(L)$. Initially we set $\overline{sum}(L)$ to be $sum(L)$.*

2. *Let m be the number of maximal blocks in R, and let $R_1 = R$.*

3. *for $p = 1, \ldots, m$:*

   (a) *Let B_p be a maximal block in R_p whose height is minimum among all maximal blocks of R_p, and assume that B_p is an $s \times t$ sub-matrix. Let A_p be a maximal row-cover of B_p with respect to R_p. For $1 \le \ell \le s$, let L_ℓ denote the sub-row of A_p that covers the ℓ'th sub-row of B_p.*

   (b) *Refill B_p by applying Procedure 1 (see Section 3.1), where the sum filled in the k'th sub-column of B_p, $1 \le k \le t$, should be the original sum of this sub-column in C_V, and the sum filled in the ℓ'th sub-row of B_p, $1 \le \ell \le s$, is at most $\overline{sum}(L_\ell)$.*

   For each $1 \le \ell \le s$, let x_ℓ denote the sum of elements filled by Procedure 1 in the ℓ'th sub-row of B_p.

(c) *Let* $R_{p+1} = R_p \setminus B_p$. *We next assign designated sums to the rows of* R_{p+1} *that have been either shortened or broken into two parts by the removal of* B_p *from* R_p. *This is done as follows:*

The set $A_p \setminus B_p$ *is the union of two non-consecutive sub-matrices,* A' *and* A'', *so that* A' *borders* B_p *from the left of* B_p *and* A'' *borders* B_p *from the right of* B_p *(where it is possible that one or both of these sub-matrices does not exist). Let* L'_ℓ *and* L''_ℓ *be the sub-rows in* A' *and* A'' *respectively that are contained in sub-row* L_ℓ *of* A_p. *We assign to* L'_ℓ *and* L''_ℓ *non-negative designated sums,* $\overline{sum}(L'_\ell)$ *and* $\overline{sum}(L''_\ell)$, *that satisfy the following:*

$$\overline{sum}(L'_\ell) + \overline{sum}(L''_\ell) = \overline{sum}(L_\ell) - x_\ell,$$

and furthermore,

$$\sum_{row\ L \in A'} \overline{sum}(L) = sum(A'), \qquad \sum_{row\ L \in A''} \overline{sum}(L) = sum(A'').$$

This is done by applying Procedure 1 to a $2 \times s$ *matrix whose sums of columns are* $sum(A')$ *and* $sum(A'')$ *and sums of rows are* $\overline{sum}(L_\ell) - x_\ell$, *where* $1 \leq \ell \leq s$.

(Note that one or both of A' *and* A'' *may not exist. This can happen if* B_p *bordered* $A_p \setminus B_p$ *on one side and its boundary coincided with* R_p, *or if* $A_p = B_p$. *In this case, if, for example,* A' *does not exist then we view it as a sub-matrix of height* 0 *where* $sum(A') = 0$.*)*

5.4 Proving that Procedure 2 is Correct

In order to prove that Procedure 2 is correct we have to prove two claims. First we have to show that the procedure does not "get stuck". Namely, that all iterations of the procedure can be completed. Second, we have to prove that at the end of the procedure, the refilled structure R has Property 1. Before we prove these two claims we first prove some properties relating to the sum of elements in maximal blocks and other sub-matrices of R. These properties will be used to show that the procedure does not get stuck.

5.4.1 Sums of Blocks and Other Sub-Matrices

Lemma 6 *The sum of elements in every maximal legal sub-row and every maximal legal sub-column in* R *is non-negative.*

Proof: We prove the lemma for maximal legal sub-rows. The claim for maximal legal sub-columns is analogous. Assume, contrary to the claim, that R contains some maximal legal sub-row $L = [\ ]^{1,t}_{i,j}$ whose sum of elements is negative. Let T be the maximal bad legal sub-matrix in C_V that contains L. By the maximality of L, necessarily $T = [\ ]^{s,t}_{i',j}$ for some $i' \leq i$ and $s \geq 1$. That is, the rows of T (one of which is L) are of length t. By the construction of R, R must contain a good legal sub-matrix T' that contains T and is twice as large in each dimension. But this contradicts the maximality of L. ■

It directly follows from Lemma 6 that every maximal row in R has a non-negative sum, and that every maximal block has a non-negative sum. We would like to characterize other sub-matrices of R whose sum is necessarily non-negative.

Lemma 7 *Consider any two maximal blocks $B = [\,]_{i,j}^{s,t}$ and $B' = [\,]_{i',j'}^{s',t'}$ where $i \le i' \le i + s - 1$, $i' + s' \le i + s$. That is, B has height s and B' has height $s' \le s$, and B' starts at row $i' \ge i$ and ends at row $i' + s' - 1 \le i + s - 1$. Consider the sub-matrix T of height s "between them". That is, $T = [\,]_{i,j+t}^{s,j'-(j+t)}$ or $T = [\,]_{i,j'+t'}^{s,j-(j'+t')}$. Suppose that $T \subset R$. Then $sum(T) \ge 0$.*

See Figure 6 for a illustration of the lemma and its proof.

Proof: Assume without loss of generality that B' is to the right of B (that is, $j' \ge j + t$ and $T = [\,]_{i,j+t}^{s,j'-(j+t)}$). If T is empty then the claim follows trivially since $sum(T) = 0$. Hence we may assume from now on that T is not empty and we separate the proof into two cases.

Case 1: T is a legal sub-matrix. Assume, contrary to the claim, that $sum(T) < 0$. That is, T is a bad legal sub-matrix. Let T' be the maximal bad legal sub-matrix containing T (where T' may equal T). By construction of R, R should contain a good legal sub-matrix T'' that contains T' and has twice the number of rows and twice the number of columns. But this would contradict the maximality of the sub-columns of B or of B'. To see why this is true, assume without loss of generality that for any legal sub-column $[\,]_{i,r}^{s,1}$, the legal column that is twice its height is $[\,]_{i,r}^{2s,1}$ (the case in which it is $[\,]_{i-s,r}^{2s,1}$, is treated analogously). Then T'' must contain either the sub-column $[\,]_{i,j'}^{2s,1}$, or the sub-column $[\,]_{i,j+t-1}^{2s,1}$ (depending on the identity of the legal sub-rows that are twice the length of the rows of T). In the first case we would get a contradiction to the fact that B' is a maximal block, and in the second case we would get a contradiction to the fact that B is a maximal block.

Case 2: T is not a legal sub-matrix. Observe that its columns are necessarily legal sub-columns (given that the columns of B are legal). Hence, only its rows are not legal sub-rows. Therefore, T can be partitioned into sub-matrices $T_1, \ldots, T_k$, such that each is of height s, and is a maximal legal sub-matrix *with respect to* T. We claim that for every T_ℓ, $sum(T_\ell) \ge 0$. Consider any fixed T_ℓ. By its maximality with respect to T, we know that the legal sub-rows that contain the rows of T_ℓ and are twice their length, are not strictly contained in T, but rather extend either to the right or to the left of T. Hence these rows (or some of them in case the height of B' is strictly smaller than the height of T_ℓ), must intersect either B or B'. Assume, contrary to what we claim, that $sum(T_\ell) < 0$. Let T'_ℓ be the maximal bad legal sub-matrix with respect to R that contains T_ℓ, and let T''_ℓ be the good legal sub-matrix that contains T'_ℓ and has twice its height and twice its width. Then T''_ℓ intersects either B or B', and in this intersection, the (legal) sub-columns of T''_ℓ strictly contain the sub-columns of B or B' (as in the case considered in the previous paragraph). But this contradicts the maximality of B or B'. ■

By Lemma 7, we get the following corollary whose proof is illustrated in Figure 7.

Corollary 8 *Let A be a sub-matrix of R that covers a given block B. If on each of its sides A either borders a block with height smaller than the height of B or its border coincides with the border of R, then $sum(A) \ge sum(B)$.*

Proof: Let $B_1, \ldots, B_k$ be the set of maximal blocks that are covered by A (where $B = B_i$ for some $1 \le i \le k$). Note that by definition of maximal blocks and covers, they are all of the same height, which is the height of A. Let D_1 and D_2 be two shorter blocks that border A on the left side and the right side of A, respectively. (If there is no such block on one of the sides, then we think of the corresponding D_i as having height 0). Let $T_0, \ldots, T_k$ be the sub-matrices between these blocks (that have the same height as the blocks). That is, T_0 is between D_1 and B_1, T_k is between B_k and D_2, and for $1 \le i \le k-1$, T_i is between B_i and B_{i+1}. Then, by Lemma 7 and the fact that every

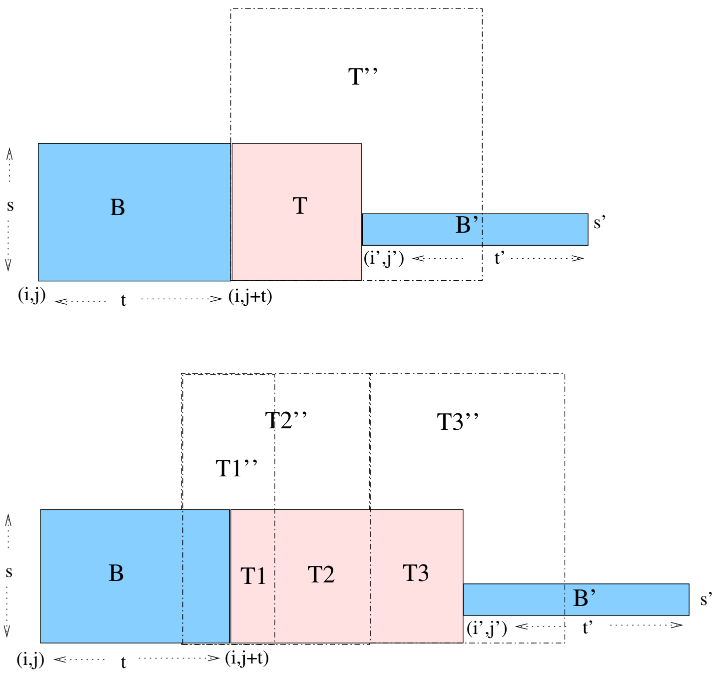

Figure 6: An illustration for Lemma 7. The figure on the top illustrates the case, in the proof of Lemma 7, where T is a legal sub-matrix (for simplicity, we assume $T' = T$). The figure on the bottom illustrates the second case in the proof when T is a union of legal sub-matrices (all having the height of B).

block has a non-negative sum we get that:

$$sum(A) = \sum_{i=1}^{k} sum(B_i) + \sum_{i=0}^{k} sum(T_i) \geq sum(B). \tag{6}$$

■

5.4.2 Proving that Procedure 2 Does not Get Stuck

Recall that for each $1 \leq p \leq m$, R_p is what remains of R at the start of the p'th iteration of Procedure 2. In particular, $R_1 = R$. In this section we show that the procedure does not "get stuck". That is, for each iteration p, Procedure 1 can be applied to the block B_p selected in this

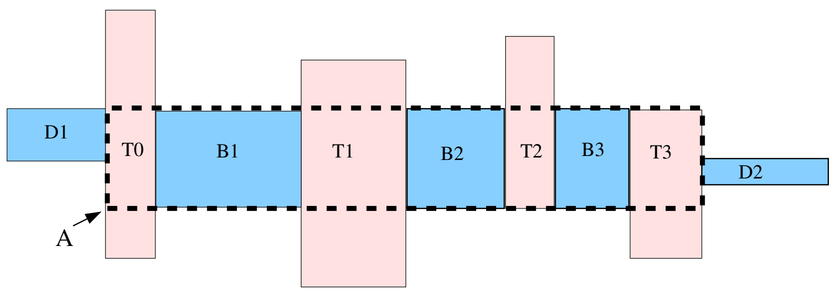

Figure 7: An illustration for Corollary 8. Here A covers the blocks B_1, B_2 and B_3, and borders the blocks D_1 and D_2. The sub-matrices T_0–T_4 are parts of larger blocks (that extend above and/or below A).

iteration, and it is possible to update the designated sums of the rows that have been shortened by the removal of B_p. Note that since the blocks are selected according to increasing (non-decreasing) height, then in each iteration there indeed exists a unique cover A_p of B_p that is a maximal row-cover with respect to R_p.

For every $1 \leq p \leq m$, let s_p be the minimum height of the maximal blocks of R_p, and let $s_0 = 1$. Observe that whenever s_p increases, it does so by a factor of 2^k for some k. This is true because the columns of maximal blocks are legal sub-columns.

Lemma 9 *For every $1 \leq p \leq m$, Procedure 1 can be applied to the block B_p selected in R_p, and the updating process of the designated sum of rows can be applied. Moreover, if A is a sub-matrix of R_p with height of at least s_{p-1}, whose columns are legal sub-columns and whose rows are maximal rows with respect to R_p, then $\sum_{row\ L \in A} \overline{sum}(L) = sum(A)$.*

Proof: Let B_p be the block selected in iteration p, where B_p is an $s \times t$ sub-matrix, and let A_p be the maximal row-cover of B_p with respect to R_p. As noted in Subsection 3.1, all that is required for Procedure 1 to work is:

(1) For every column K in B_p, $sum(K) \geq 0$.

(2) For every row L in A_p, $\overline{sum}(L) \geq 0$.

(3) $\sum_{\text{row } L \in A_p} \overline{sum}(L) \geq \sum_{\text{column } K \in B_p} sum(K)$.

In order for the updating process to succeed in Step 3c of Procedure 2, we must have that:

(4) For each $1 \leq \ell \leq s$, let x_ℓ be the sum of elements filled in the ℓ'th sub-row of B_p, and let L_ℓ be the sub-row of A_p that covers this sub-row of B_p. Then, $\overline{sum}(L_\ell) - x_\ell \geq 0$.

(5) If $A_p \setminus B_p$ consists of the two sub-matrices A' and A'' (between which resided B), then $sum(A') \geq 0$, $sum(A'') \geq 0$, and

$$\sum_{\text{row } L_\ell \in A_p} (\overline{sum}(L_\ell) - x_\ell) = sum(A') + sum(A'').$$

By Lemma 6, Item (1) holds at the start of every iteration. In order to prove the other items for every p, we first extend and generalize Item (2):

(2') Let A be any sub-matrix in R_p having height at least s_{p-1} whose columns are legal sub-columns and whose rows are maximal rows with respect to R_p. Then for every row L of A we have $\overline{sum}(L) \geq 0$, and $\sum_{\text{row } L \in A} \overline{sum}(L) = sum(A)$.

Observe that if Item (2') holds at the start of iteration p, then in particular it holds for A_p. Hence by Corollary 8

$$\sum_{\text{row } L \in A_p} \overline{sum}(L) = sum(A_p) \geq sum(B_p) \tag{7}$$

and so Item (3) holds as well.

Furthermore, if Items (1)–(3) hold at the start of iteration p, then Procedure 1 can be applied successfully. Thus Item (4) necessarily holds by definition of Procedure 1. The first part of Item (5), concerning the non-negativity of A' and A'', follows from Lemma 7 very similarly to the way

Corollary 8 follows from this lemma. The second part of Item (5) follows from Item (2') holding for A_p and the fact that $\sum_{\ell=1}^{s} x_\ell = sum(B_p)$ (since Procedure 1 completed successfully). Hence,

$$\sum_{\text{row } L_\ell \in A_p} (\overline{sum}(L_\ell) - x_\ell) = sum(A_p) - sum(B_p) = sum(A') + sum(A'') \tag{8}$$

as required.

Hence, it remains to prove that Item (2') holds at the start of every iteration p. We do so by induction on p. Consider the base case, $p = 1$, so that $R_p = R_1 = R$. By the initialization of Procedure 2, for every maximal sub-row L of R, $\overline{sum}(L) = sum(L)$. By Lemma 6 (applied to the maximal legal sub-rows that partition L), we know that $\overline{sum}(L) \geq 0$. Furthermore, for every sub-matrix A of R having height of at least $s_{p-1} = s_0 = 1$ and whose rows are maximal sub-rows of R,

$$\sum_{\text{row } L \in A} \overline{sum}(L) = \sum_{\text{row } L \in A} sum(L) = sum(A) \tag{9}$$

as required.

Assuming that the induction claim holds for $p - 1$, we prove it for p. Consider any sub-matrix A having height at least s_{p-1}, whose columns are legal sub-columns and whose rows are maximal sub-rows with respect to R_p. If A also consisted of maximal sub-rows with respect to R_{p-1}, then we are done by the induction hypothesis.

Otherwise, the block B_{p-1} of height s_{p-1} that was removed from R_{p-1}, bordered A on one of its sides. Let $A^1, \ldots, A^q$ be the disjoint sub-matrices of height s_{p-1} such that $A = \cup_{h=1}^{q} A^h$. That is, $A^1, \ldots, A^q$ are located one on top of the other (for an illustration, see Figure 8). In this case, all but at most one of these sub-matrices, say A^q, consisted of maximal sub-rows with respect to R_{p-1}, and B_{p-1} bordered A^q.

For each of the sub-matrices $A^1, \ldots, A^{q-1}$ we can apply the induction hypothesis (Item (2')). We get that for each such A^h: (a) For every row L in A^h, $\overline{sum}(L) \geq 0$; and (b) $\sum_{\text{row } L \in A^h} \overline{sum}(L) = sum(A^h)$.

As for A^q, assume without loss of generality that B_{p-1} bordered A^q from the right of A^q. Let A' be the sub-matrix that bordered B_{p-1} from the right of B_{p-1} (A' may be empty). This means that A_{p-1} is of the form $A_{p-1} = A^q \cup B_{p-1} \cup A'$ (see Figure 8). But then, by definition of the updating rule and since it succeeded by the induction hypothesis (Items (4) and (5)), we have that for every row L in A^q, $\overline{sum}(L) \geq 0$ and $\sum_{\text{row } L \in A^q} \overline{sum}(L) = sum(A^q)$.

It follows that for every row L in A we have $\overline{sum}(L) \geq 0$ and

$$\sum_{\text{row } L \in A} \overline{sum}(L) = \sum_{h=1}^{q} \sum_{\text{row} L \in A^h} \overline{sum}(L) = \sum_{h=1}^{q} sum(A^h) = sum(A) \ . \tag{10}$$

The induction step is proven. ■

5.4.3 Proving that Property 1 holds at the end of Procedure 2

Finally, we have to show that when Procedure 2 terminates and R is refilled with non-negative values, then Property 1 holds. This will complete the proof of Theorem 3.

Let $\tilde{C}_V = \{\tilde{c}_{i,j}\}$ be the matrix resulting from the application of Procedure 2 to the matrix $C_V = \{c_{i,j}\}$. For any sub-matrix T of C_V (and in particular of R), we let $\widetilde{sum}(T)$ denote the sum of elements of T in $\tilde{C}_V$. By definition of the procedure, $\widetilde{sum}(K) = sum(K)$ for every maximal legal sub-column K of R. Hence this holds also for every maximal sub-column of R. We next prove a related claim concerning rows.

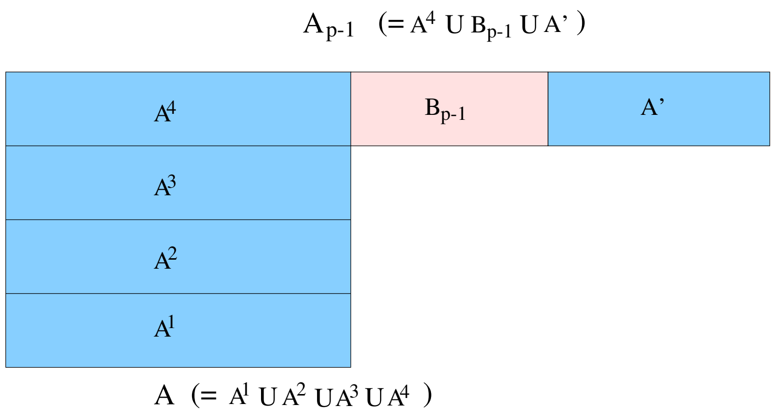

Figure 8: An illustration for the induction step in the proof of Lemma 9 (where $q = 4$).

Lemma 10 *For every sub-row L in R, such that L is assigned $\overline{sum}(L)$ as a designated sum at some iteration of Procedure 2, we have that $\widetilde{sum}(L) = \overline{sum}(L)$.*

Observe that by combining Lemma 10 with Lemma 6 we get that for every maximal sub-row L of R, $\widetilde{sum}(L) = \overline{sum}(L) = sum(L)$.

Proof: Let $\mathcal{L}$ the the set of sub-rows L of R, such that L is assigned $\overline{sum}(L)$ as a designated sum at some iteration of Procedure 2. Observe that the set $\mathcal{L}$ consists exactly of those rows that are maximal sub-rows for some R_p. We prove the lemma by induction on the length of $L \in \mathcal{L}$. For the base of the induction, consider any sub-row $L \in \mathcal{L}$ that is shortest among all sub-rows in $\mathcal{L}$. Since L is shortest, it must be completely filled in a single iteration as part of a block B (or otherwise there would be a shorter $L' \subset L$ with a designated sum $\overline{sum}(L')$). But by definition of the procedure, we get that $\widetilde{sum}(L) = \overline{sum}(L)$ as required.

Assume that the claim holds for every L of length less than ℓ, we prove it for L having length ℓ. Consider the first iteration after which L became a maximal sub-row (and so received the designated sum $\overline{sum}(L)$) in which part of L is filled. If all of L is filled, then the induction claim follows as in the base case. Otherwise, let x be the sum of elements that was filled in the part $P \subset L$. Let L' and L'' be what remains of L to the left and right of P respectively. Then the procedure sets $\overline{sum}(L') + \overline{sum}(L'') = \overline{sum}(L) - x$. But L' and L'' are strictly shorter than L, and therefore by the induction hypothesis $\widetilde{sum}(L') = \overline{sum}(L')$ and $\widetilde{sum}(L'') = \overline{sum}(L'')$. Thus $\widetilde{sum}(L) = \widetilde{sum}(L') + \widetilde{sum}(L'') + x = \overline{sum}(L') + \overline{sum}(L'') + x = \overline{sum}(L)$ as required. ■

Definition 16 (Boundary) *We say that a point (i, j) is on the* boundary *of R if $(i, j) \in R$, but either $(i + 1, j) \notin R$, or $(i, j + 1) \notin R$, or $(i + 1, j + 1) \notin R$. We denote the set of boundary points by $\mathcal{B}$.*

Definition 17 *For a point (i, j), $1 \leq i, j \leq n$ let $R^{\leq}(i, j)$ denote the subset of points $(i', j') \in R$, $i' \leq i, j' \leq j$, and let $sum^R(i, j) = \sum_{(i',j') \in R^{\leq}(i,j)} c_{i',j'}$ and $\widetilde{sum}^R(i, j) = \sum_{(i',j') \in R^{\leq}(i,j)} \tilde{c}_{i',j'}$.*

Property 1 and therefore Theorem 3 will follow directly from the next two lemmas.

Lemma 11 *For every point $(i, j) \in \mathcal{B}$, $\widetilde{sum}^R(i, j) = sum^R(i, j)$.*

Proof: Consider any point $(i, j) \in \mathcal{B}$ and let $U = R^{\leq}(i, j)$. Let $\mathcal{C}(U) = \{B_1, \ldots, B_q\}$ be the minimal set of (maximal) blocks whose union contains U. For each $B_h \in \mathcal{C}(U)$ we know that

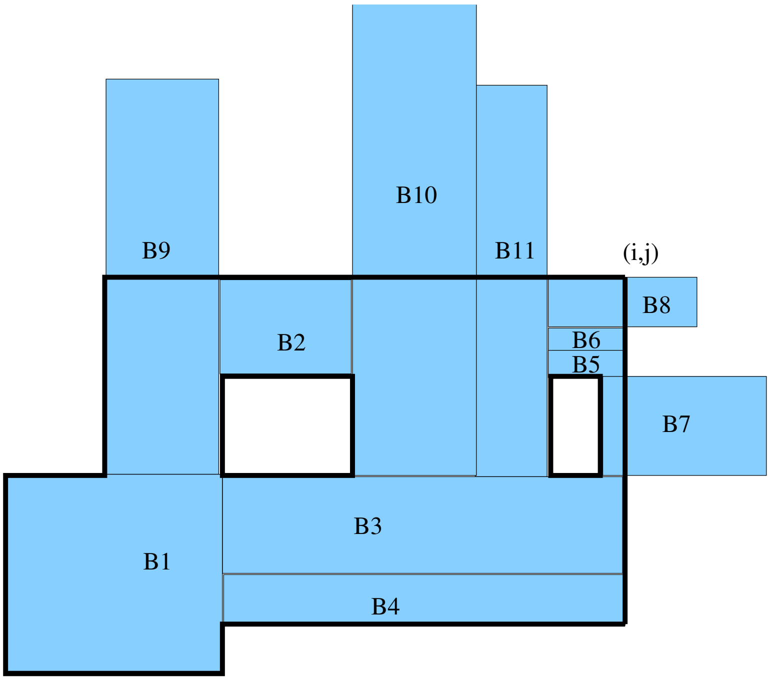

Figure 9: An illustration for the proof of Lemma 11. The solid line denotes the outline of $U = R^{\leq}(i,j)$, where point (i,j) is in the top-right corner. Blocks B_1–B_6 are fully contained in U and therefore belong to $\mathcal{C}_1(U)$. Blocks B_7 and B_8 belong to $\mathcal{C}_2(U)$ and blocks B_9–B_{11} belong to $\mathcal{C}_3(U)$. Block B_{10} is twice the height of B_9 and B_{11} and so "extends out of the figure".

$\widetilde{sum}(B_h) = sum(B_h)$. In particular this is true for every $B_h \subset U$. Let $\mathcal{C}_1(U) = \{B_h \in \mathcal{C}(U) : B_h \subset U\}$. Hence we have that

$$\sum_{B_h \in \mathcal{C}_1(U)} \widetilde{sum}(B_h \cap U) = \sum_{B_h \in \mathcal{C}_1(U)} \widetilde{sum}(B_h) = \sum_{B_h \in \mathcal{C}_1(U)} sum(B_h) . \tag{11}$$

If every $B_h \in \mathcal{C}(U)$ is fully contained in U then $\mathcal{C}_1(U) = \mathcal{C}(U)$ and we are done.

Otherwise, consider the remaining B_h's in $\mathcal{C}(U) \setminus \mathcal{C}_1(U)$ (i.e., blocks that are not fully contained in U but rather intersect it). Each of them either contains a column that is a sub-column of column $j+1$, or a row that is a sub-row of row $i+1$ (recall that $U = R^{\leq}(i,j)$). Let the former subset be denoted $\mathcal{C}_2(U)$ and the latter $\mathcal{C}_3(U)$. Thus $\mathcal{C}_2(U)$ contains blocks that "intersect U from the right", and $\mathcal{C}_3(U)$ contain blocks that "intersect U from the top". See for example Figure 9.

It is important to note that $\mathcal{C}_2(U) \cap \mathcal{C}_3(U) = \emptyset$: If there existed a block $B_h \in \mathcal{C}_2(U) \cap \mathcal{C}_2(U)$, it would necessarily contain both (i,j), and the three neighboring points, $(i+1,j)$, $(i,j+1)$ and $(i+1,j+1)$. But this contradicts the fact that (i,j) is a boundary point.

For each $B_h \in \mathcal{C}_2(U)$, $B_h \cap U$ is a subset of maximal legal sub-columns with respect to R (since each $B_h \in \mathcal{C}_2(U)$ cannot extend beyond row i). Let $\mathcal{K}_2(U)$ denote the set of all maximal legal sub-columns that belong to $\bigcup_{B_h \in \mathcal{C}_2(U)} (B_h \cap U)$. Since for every maximal legal sub-column K, it holds that $\widetilde{sum}(K) = sum(K)$, we have that

$$\sum_{B_h \in \mathcal{C}_2(U)} \widetilde{sum}(B_h \cap U) = \sum_{K \in \mathcal{K}_2(U)} \widetilde{sum}(K) = \sum_{K \in \mathcal{K}_2(U)} sum(K). \tag{12}$$

Next consider the blocks $B_h \in \mathcal{C}_3(U)$. Let $\mathcal{L}_3(U)$ be the set of sub-rows in U that are maximal sub-rows with respect to $\bigcup_{B_h \in \mathcal{C}_3(U)} (B_h \cap U)$. Thus, $\bigcup_{B_h \in \mathcal{C}_3(U)} (B_h \cap U) = \bigcup_{L \in \mathcal{L}_3} L$. We next observe that for every $B_h \in \mathcal{C}_3(U)$, all blocks that border B_h and belong either to $\mathcal{C}_1(U)$ or to $\mathcal{C}_2(U)$, must be strictly shorter than B_h. This follows from the definition of legal sub-columns. Hence, the blocks in $\mathcal{C}_1(U)$ and $\mathcal{C}_2(U)$ are all removed before the blocks in $\mathcal{C}_3(U)$.

For each sub-row in $\mathcal{L}_3(U)$ there exists the first iteration p in which it becomes a maximal sub-row with respect to R_p (following the removal of some block in $\mathcal{C}_1(U) \cup \mathcal{C}_2(U)$ from R_{p-1}). We partition the rows in $\mathcal{L}_3(U)$ accordingly. Let $\mathcal{L}_3^p(U)$ denote all sub-rows in $\mathcal{L}_3(U)$, that are maximal sub-rows with respect to R_p but were not maximal sub-rows with respect to R_{p-1}. Observe that in particular, $\mathcal{L}_3^1(U)$ is the set of sub-rows in $\mathcal{L}_3(U)$ that were already maximal sub-rows with respect to R. By this definition the sub-rows in $\mathcal{L}_3^p(U)$ constitute a sub-matrix of height s_{p-1}. By the second part of Lemma 9, $\sum_{L\in\mathcal{L}_3^p(U)} \overline{sum}(L) = \sum_{L\in\mathcal{L}_3^p(U)} sum(L)$, and by applying Lemma 10 we get that $\sum_{L\in\mathcal{L}_3(U)} \widetilde{sum}(L) = \sum_{L\in\mathcal{L}_3(U)} sum(L)$. Therefore,

$$\sum_{B_h\in\mathcal{C}_3(U)} \widetilde{sum}(B_h \cap U) = \sum_{L\in\mathcal{L}_3(U)} \widetilde{sum}(L) = \sum_{L\in\mathcal{L}_3(U)} sum(L). \tag{13}$$

By combining Equations (11)–(13) we get

$$\begin{aligned}
\widetilde{sum}(U) &= \sum_{B_h\in\mathcal{C}(U)} \widetilde{sum}(B_h \cap U) \\
&= \sum_{q=1}^{3} \sum_{B_h\in\mathcal{C}_q(U)} \widetilde{sum}(B_h \cap U) \\
&= \sum_{B_h\in\mathcal{C}_1(U)} sum(B_h) + \sum_{K\in\mathcal{K}_2(U)} sum(K) + \sum_{L\in\mathcal{L}_3(U)} sum(L) \\
&= sum(U)
\end{aligned}$$

■

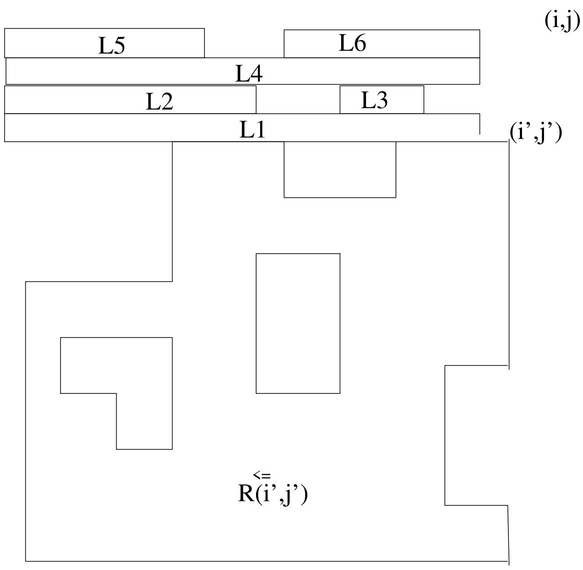

Figure 10: An illustration for the proof of Lemma 12. The point (i', j') is as defined in the proof, and the rows $L_1, \ldots, L_6$ are all maximal sub-rows of R that belong to rows $i'+1, \ldots, i$ and end by column j (that is, the set $\mathcal{L}(i, i', j)$).

Lemma 12 *Let (i,j) be any point such that $(i,j) \notin R$. Then $\widetilde{sum}^R(i,j) = sum^R(i,j)$.*

Proof: Let $(i',j') \in R$, $i' < i$, $j' \leq j$, be the point for which j' is maximized, and if there are several such points, let it be the one amongst them for which i' is maximized. Thus, (i',j') is maximal in the sense that for every (i'',j''), $i'' < i$, $j'' \leq j$ such that $(i'',j'') > (i',j')$ it holds that $(i'',j'') \notin R$. Furthermore, among all such maximal points it is the right-most one (i.e., it belongs to the column with the highest index). By definition, (i',j') belongs to $\mathcal{B}$, since $(i'+1,j')$ necessarily does not belong to R. Let $\mathcal{L}(i,i',j)$ be the subset of all maximal sub-rows of R that belong to rows $i'+1,\ldots,i$, and end by column j. Then $\widetilde{sum}^R(i,j) = \widetilde{sum}^R(i',j') + \sum_{L \in \mathcal{L}(i,i',j)} \widetilde{sum}(L)$. By applying Lemma 11 and Lemma 10, we get that $\widetilde{sum}^R(i,j) = sum^R(i,j)$. ■

5.5 Distribution Matrices

As noted in the introduction, a sub-family of inverse Monge matrices that is of particular interest is the class of *distribution matrices*. A matrix $V = \{v_{i,j}\}$ is said to be a distribution matrix, if there exists a non-negative *density matrix* $D = \{d_{i,j}\}$, such that every entry $v_{i,j}$ in V is of the form $v_{i,j} = \sum_{k \leq i} \sum_{\ell \leq j} d_{k,\ell}$. In particular, if V is a distribution matrix then the corresponding density matrix D is simply the matrix C'_V (as defined in Section 3). Hence, in order to test that V is a distribution matrix, we simply run our algorithm for inverse Monge matrix on C'_V instead of C_V.

Acknowledgments

We would like to thank Noam Nisan for suggesting to examine combinatorial auctions in the context of property testing. We would also like to thank the anonymous referees for their comments which helped us improve the presentation of this paper.

References

[BKR96] R.E. Burkard, B. Klinz, and R. Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 70, 1996.

[BRW99] T. Batu, R. Rubinfeld, and P. White. Fast approximate PCPs for multidimensional bin-packing problems. In *Proceedings of RANDOM*, pages 245–256, 1999.

[DGL+99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonocity. In *Proceedings of RANDOM*, pages 97–108, 1999.

[dVV00] S. de Vries and R. Vohra. Combinatorial auctions: a survey. available from: `http://www.kellogg.nwu.edu/faculty/vohra/htm/res.htm`, 2000.

[EKK+00] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *JCSS*, 60(3):717–751, 2000.

[Fis01] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, 2001.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653–750, 1998.

[GLS81] M. Grotschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1, 1981.

[Hof63] A.J. Hoffman. On simple linear programming problems. In *In Proceedings of Symposia in Pure Mathematics, Convexity*, volume 7, pages 317–327, 1963. American Mathematical Society.

[IFF01] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *JACM*, 48(4):761–777, 2001.

[LLN01] B. Lehmann, D. Lehmann, and N. Nisan. Combinatorial auctions with decreasing marginal utilities. In *ACM Conference on Electronic Commerce*, 2001.

[Lov83] L. Lovász. Submodular functions and convexity. *Mathematical Programming: The State of the Art*, pages 235–257, 1983.

[Ron01] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001.

[RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[Sch00] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory B*, 80:346–355, 2000.