On Testing Convexity and Submodularity

Michal Parnas[1], Dana Ron[2⋆], and Ronitt Rubinfeld[3]

[1] The Academic College of Tel-Aviv-Yaffo. michalp@mta.ac.il
[2] Department of EE – Systems, Tel-Aviv University. danar@eng.tau.ac.il
[3] NEC Research Institute, Princeton, NJ. ronitt@research.nj.nec.com

Abstract. Submodular and convex 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 $0 \leq i < r \leq n_1$ and $0 \leq j < s \leq n_2$, we have $v_{i,j} + v_{r,s} \leq v_{i,s} + v_{r,j}$. 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 a testing algorithm for Monge and inverse Monge matrices, whose running time is $O((\log n_1 \cdot \log n_2)/\epsilon)$, where ϵ is the distance parameter for testing. In addition we have an algorithm that tests whether a function $f : [n] \to \mathbb{R}$ is convex (concave) with running time of $O((\log n)/\epsilon)$.

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 [14]), which often makes it possible to efficiently find their minimum [10,12,18], and thus leads to efficient algorithms for many important optimization problems.

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 1 (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 $x, y \in \mathcal{I}$, $f(x \vee y) + f(x \wedge y) \geq f(x) + f(y)$.*

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

J.D.P. Rolim and S. Vadhan (Eds.): RANDOM 2002, LNCS 2483, pp. 11–25, 2002.

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 $\{1,\ldots,d\}$. 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 [11]). 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 [2] for a comprehensive survey on Monge matrices and their applications.

Testing Submodularity and Convexity. In this paper we approach the question of submodularity and convexity from within the framework of property testing [17,9]. 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 Monge matrices and for discrete convexity in one dimension. Specifically:

- We describe and analyze a testing algorithm for Monge and inverse Monge matrices whose running time is $O\left((\log n_1 \cdot \log n_2)/\epsilon\right)$, 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 subfamily 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 $v_{i,j} = \sum_{k \leq i} \sum_{\ell \leq j} d_{k,\ell}$. In other words, the entry $v_{i,j}$ corresponds to the cumulative density of all entries $d_{k,\ell}$ such that $k \leq i$ and $\ell \leq j$.
- We provide an algorithm that tests whether a function $f : [n] \to \mathbb{R}$ is convex (concave). The running time of this algorithm is $O\left(\log n/\epsilon\right)$.

Techniques. 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 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

$(i,s),(r,j)$ such that $i < r$ and $j < s$ we have that $(i,s) \vee (r,j) = (r,s)$ and $(i,s) \wedge (r,j) = (i,j)$. It follows from Definition 1 that V is a Monge matrix (f is a 2-dimensional submodular function) if and only if:

$$\forall i,j,r,s \quad \text{s.t. } i < r,\ j < s \ : \quad v_{i,j} + v_{r,s} \leq v_{i,s} + v_{r,j}$$

and V is an inverse Monge matrix (f is a 2-dimensional supermodular function) if and only if: $\forall i,j,r,s \quad \text{s.t. } i < r,\ j < s \ : \quad v_{i,j} + v_{r,s} \geq v_{i,s} + v_{r,j}$.

That is, in both cases we have a constraint for every quadruple $v_{i,j}$, $v_{r,s}$, $v_{i,s}$, $v_{r,j}$ such that $i < r$ and $j < s$.[1] 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[2] (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.

Previous Work. Property testing was first defined in the context of algebraic properties of functions [17]. It was extended in [9] and in particular applied to properties of graphs. In recent years it has been studied in a large variety of contexts. For surveys on property testing see [16,6].

One testing problem that is probably most related to ours is testing whether a function is monotone [8,4,5,7,1]. A function $f : [n] \to \mathbb{R}$ is monotone non-decreasing if for every $0 \leq i < n$, the differences $f(i+1) - f(i)$ are non-negative, whereas f is convex if and only if for every $1 \leq i \leq n-1$, the differences of differences $[f(i+1) - f(i)] - [f(i) - f(i-1)]$ are non-negative. In other words, f is convex if and only if the function $f'(i) = f(i) - f(i-1)$, is monotone non-decreasing. We note though that testing f for convexity cannot be done by

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

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

simply testing f' for monotonicity. Specifically, if f' is close to monotone then it does not necessarily follow that f is close to convex.

Further Research. We suggest the following open problems. First it remains open to determine the complexity of testing submodular functions when the dimension d of the input domain is greater than 2. It seems that our algorithm 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 seem to be much more expressive. Hence it is not immediately clear how to adapt our algorithm to testing Monge matrices of higher dimensions.

It would also be interesting to find an efficient testing algorithm for the subclass of submodular set functions, which are used for example in combinatorial auctions on the internet (e.g. [3],[13]).

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. In Section 2 we describe several building blocks that will be used by our testing algorithm for Monge matrices. In Section 3 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 4 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. The testing algorithm for convexity can be found in the full version of this paper [15]. All missing details of the analysis together with helpful illustrations appear in [15] as well.

2 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 algorithm. 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:

- $c_{0,0} = v_{0,0}$; $\forall i > 0 : c_{i,0} = v_{i,0} - v_{i-1,0}$; $\forall j > 0 : c_{0,j} = v_{0,j} - v_{0,j-1}$;
- $\forall i, j > 0 : c_{i,j} = (v_{i,j} - v_{i-1,j}) - (v_{i,j-1} - v_{i-1,j-1}) = (v_{i,j} - v_{i,j-1}) - (v_{i-1,j} - v_{i-1,j-1})$.

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.

Claim 1 *For every* $0 \leq i, j \leq n$, $v_{i,j} = \sum_{i'=0}^{i} \sum_{j'=0}^{j} c_{i',j'}$.

Claim 2 *A matrix* V *is an inverse Monge matrix if and only if* C_V *is a non-negative matrix.*

It follows from Claim 2 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 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 $v_{i,j}, v_{r,s}, v_{i,s}, v_{r,j}$ in V such that from any such quadruple at least one value should be changed in order to transform V into a 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 3 *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* (r, s) *then the sum of elements of* A *is* $v_{r,s} - v_{r,j} - v_{i,s} + v_{i,j}$.

2.1 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 $a_1, ..., a_s$ and $b_1, ..., b_t$ 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* $i = 1, ..., s$ *and* $\bar{b}_j = b_j$ *for* $j = 1, ..., t$.
(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 $\bar{a}_i = \bar{a}_i - x$, $\bar{b}_j = \bar{b}_j - x$.

3 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 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 2 (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 $[\,]_{i,j}^{1,t}$. A* sub-column *is defined analogously, and is denoted by $[\,]_{i,j}^{s,1} = ((i,j),(i+1,j),\ldots,(i+s-1,j))$. More generally, an $s \times t$ sub-matrix whose bottom-left entry is (i,j) is denoted $[\,]_{i,j}^{s,t}$.*

Definition 3 (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 $[\,]_{i,j}^{1,t}$ is legal, then so are $[\,]_{i,j}^{1,t/2}$ and $[\,]_{i,j+t/2}^{1,t/2}$. 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 $[\,]_{i,j}^{1,t}$ 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.

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 4 (Good sub-matrix.) *We say that a sub-matrix A of C_V is* good *if the sum of elements in each row and each column of A is non-negative.*

Definition 5 (Good Point.) *We say that point (i,j) is* good *if all legal square sub-matrices A of C_V which contain (i,j) are good.*

Algorithm 1 [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 3, 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 $\log n$ 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 1 *If V is an inverse Monge matrix then it is always accepted, and if V is ϵ-far from being an inverse Monge matrix, then the algorithm rejects with probability at least* $2/3$.

Proof. The first part of the theorem follows directly from Claim 2. 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 square sub-matrix in C_V which 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 smallest good (legal square) sub-matrix A which contains B. First observe that such a good sub-matrix must exist. 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 for every two maximal sub-matrices B and B', the corresponding good sub-matrices A and A' that contain them are either the same sub-matrix, or are totally disjoint. Finally, the sum of areas of all these good sub-matrices is at most $4 \cdot (\epsilon/4)n^2 = \epsilon n^2$.

We now correct each such good sub-matrix 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 2.1.

Note that after correcting 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. We must show however, 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 corrected 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 1 we have that $v_{k,\ell} = \sum_{i=0}^{k} \sum_{j=0}^{\ell} c_{i,j}$.

4 A Faster Algorithm for Inverse Monge Matrices

Though the above algorithm has running time sub-linear in the size of the matrix, which is n^2, we would further like to improve its dependence on n. We next suggest a variant of the algorithm whose running time is $O(\log^2 n/\epsilon)$ and explain what needs to be proved in order to argue its correctness. We first redefine the concepts of good sub-matrices and good points.

Definition 6 (Good sub-matrix.) *A (legal) sub-matrix T of C_V is* good *if the sum of all its elements is non-negative. Otherwise, T is* bad.

Definition 7 (Good Point.) *We say that a point is* good *if every legal sub-matrix of C_V that contains it is good. Otherwise the point is* bad.

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 a non-negative sum. In the full version of this paper [15] we explain how this assumption can be easily removed. Note that this assumption implies that every $s \times n$ sub-matrix is good, and similarly for every $n \times s$ sub-matrix (but of course it has no implications on smaller sub-matrices).

Algorithm 2 [Test Monge II.]

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

Note that by Definition 3, each point in an $n \times n$ matrix is contained in $O(\log^2 n)$ legal sub-matrices. Thus by Claim 3, checking that a point is good takes time $O(\log^2 n)$. Therefore the running time of the algorithm is $O((\log^2 n)/\epsilon)$.

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

4.1 Outline of the Proof of Theorem 2

If V is an inverse Monge matrix then by Claim 2 all elements in C_V are non-negative, and so the algorithm always accepts. Suppose V is ϵ-far from being inverse Monge. We claim that in such a case C_V must contain more than $(\epsilon/4)n^2$ bad points, causing the algorithm to reject with probability at least $1 - (1 - \epsilon/4)^{(8/\epsilon)} > 1 - e^{-2} > 2/3$. Assume contrary to the claim that C_V contains at most $(\epsilon/4)n^2$ bad 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.

Consider the union of all bad legal sub-matrices of C_V. Since within each bad legal sub-matrix, all points are bad, then the area occupied by this union is at most $(\epsilon/4)n^2$.

Definition 8 (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 such maximal bad legal sub-matrices of C_V. For each such sub-matrix B let us take the legal sub-matrix that contains it and has twice the number of rows and twice the number of columns. Then by the maximality of B

(and our assumption that all full rows and columns have a non-negative sum), the resulting sub-matrix is good. We now take the union of all these good legal sub-matrices, and get a total area of size at most ϵn^2. Denote the union of all these sub-matrices by R. See for example Figure 1.

Fig. 1. An example of the structure of a subset R (outlined by a bold line). The bad legal sub-matrices determining R are the gray sub-matrices. Each is contained inside a good legal sub-matrix that has twice the number of rows and twice the number of columns (marked by dashed rectangles). Observe that maximal bad-legal sub-matrices may overlap.

Definition 9 (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 would like to 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:

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 as it was 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 1 that $\tilde{V}$ is at most ϵ-far from the original matrix V.

4.2 Fixing R

Let R be the subset of entries in the matrix C_V that consists of a union of good legal sub-matrices. 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.

Lemma 4 *The sum of elements in every maximal legal sub-row and every maximal legal sub-column in R is non-negative.*

Proof. Assume, contrary to the claim, that R contains some maximal legal sub-row $L = [\ ]_{i,j}^{1,t}$ 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 $T = [\ ]_{i',j}^{s,t}$ 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.

Maximal Blocks. We will partition R into disjoint blocks (sub-matrices) and fill each block separately with non-negative values, so that the sum property for R is maintained (see Property 1). We define blocks as follows.

Definition 10 (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 9. That is, for every $j \leq r \leq j+t-1$, the column $[\ ]_{i,r}^{s,1}$ is a maximal legal sub-column (with respect to R).*

The maximality of the sequence of sub-columns in a block 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 2 for an example of R and its partition into maximal blocks.

Definition 11 (Size of a Maximal Block.) *Let B be a maximal block. The* size *of B is the height of the columns in B (equivalently, the number of rows in B).*

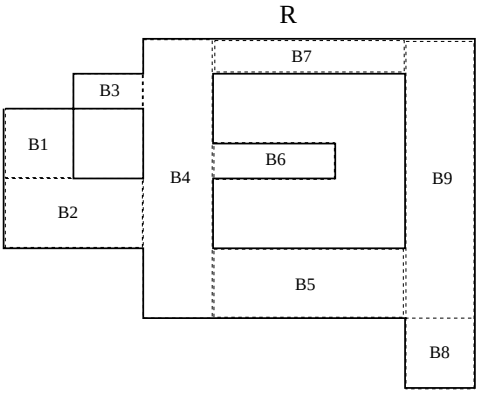

Fig. 2. An example of the partition of R into maximal blocks (numbered B_1–B_9). Note that the ratio between the sizes of any two blocks is always a power of 2. Furthermore, the blocks are "oriented" in the following sense. Suppose a block B has size s and a block B' has size $s' \leq s$ and some of their sub-rows belong to the same row of the matrix (e.g., B_4 and B_2, or B_9 and B_6). Then the smaller 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.

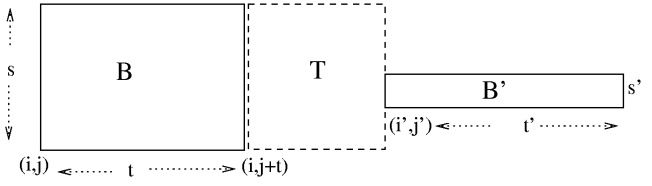

Fig. 3. An illustrations for Lemma 5.

Bounded Sub-matrices. As we have shown in Lemma 4, the sum of elements in every maximal legal sub-column in R is non-negative. It directly follows 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.

Definition 12 *For a given sub-matrix T, we denote the sum of the elements in T by $sum(T)$.*

Lemma 5 *Consider any two maximal blocks $B = [\,]_{i,j}^{s,t}$ and $B' = [\,]_{i,j'}^{s,t'}$ where $j' > j + t$. That is, both blocks have the same size s and both start at row i and end at row $i + s - 1$. Consider the sub-matrix $T = [\,]_{i,j+t}^{s,j'-(j+t)}$ "between them". Suppose that $T \subset R$. Then $sum(T) \geq 0$.*

See Figure 3 for an illustration of Lemma 5.

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

size 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 $A = [\,]_{i,j}^{s,t}$ in R,* borders *a maximal block $B = [\,]_{i',j'}^{s',t'}$ if $i \leq i' \leq i+s-1$, $i'+s' \leq i+s$, and either $j' = j+t$ (so that A borders B from the left), or $j'+t' = j$ (so that A borders B from then right).*

By Lemma 5 and using the above terminology, we get the following corollary whose proof is illustrated in Figure 4.

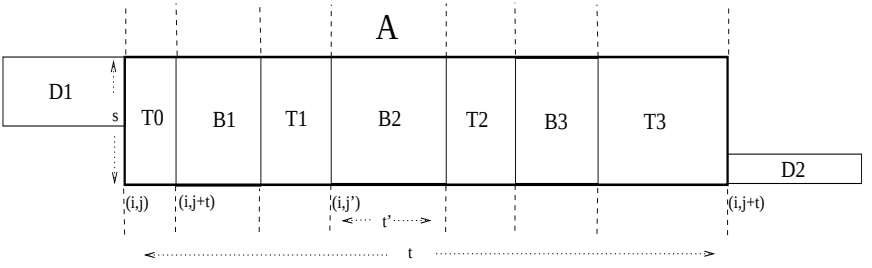

Fig. 4. An illustration for Corollary 6. Here A covers the blocks B_1, B_2 and B_3, and borders the blocks D_1 and D_2. The sub-matrices T_0–T_4 are parts of larger blocks (that extend above and/or below A).

Corollary 6 *Let A be a sub-matrix of R which covers a given block B. If on each of its sides A either borders a block smaller than B or its border coincides with the border of R, then $sum(A) \geq sum(B)$.*

The Procedure for Refilling R. We now describe the procedure that refills the entries of R with non-negative values. 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. The procedure described below will remove the blocks of R one by one, in order of increasing size, and refill each block separately using Procedure 1. After removing each block, the sum of the elements in each remaining column in R remains the same, however the row sums must be updated. Procedure 1 is used here as well.

Procedure 2 [Refill R.]

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

2. *Let m be the number of maximal blocks in R, and let $R_1 = R$.*
3. *for $p = 1, ..., m$:*
   a) *Let B_p be a maximal block in R_p whose size 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 $1 \leq \ell \leq s$, 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 2.1), where the sum filled in the k'th sub-column of B_p, $1 \leq k \leq t$, 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, $1 \leq \ell \leq s$, is at most $\overline{sum}(L_\ell)$.*
   For each $1 \leq \ell \leq s$, let x_ℓ denote the sum of elements filled by Procedure 1 in the ℓ'th sub-row of B_p.
   c) *Let $R_{p+1} = R_p \setminus B_p$ and 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 and A'' borders B_p from the right. 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 size 0 where $sum(A') = 0$.)

4.3 Proving That Procedure 2 Works

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, $R_1 = R$. We would first like to 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 iteration, and the updating of the designated sum for the rows that have been shortened by the removal of B_p can be performed. Note that since the blocks are selected according to increasing size, then in each iteration the maximal row cover A_p of B_p must actually be a sub-matrix.

Proving that Procedure 2 Does not Get Stuck. For every $1 \leq p \leq m$, let s_p be the minimum size of the maximal blocks of R_p, where $s_0 = 1$. 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 7 *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 $\sum_{row\ L \in A} \overline{sum}(L) = sum(A)$.*

Proving that Procedure 2 is Correct. 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 state a related claim concerning rows.

Lemma 8 *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 in particular we get that for every maximal sub-row L of R, $\widetilde{sum}(L) = \overline{sum}(L) = sum(L)$.

Definition 15 (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 16 *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 2 will follow directly from the next two lemmas.

Lemma 9 *For every point $(i,j) \in \mathcal{B}$, $\widetilde{sum}^R(i,j) = sum^R(i,j)$.*

Lemma 10 *Let (i,j) be any point such that $(i,j) \notin R$. Then $\widetilde{sum}^R(i,j) = sum^R(i,j)$.*

4.4 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 2). 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.

References

1. T. Batu, R. Rubinfeld, and P. White. Fast approximate pcps for multidimensional bin-packing problems. In *Proceedings of RANDOM*, pages 245–256, 1999.
2. 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.
3. S. de Vries and R. Vohra. Combinatorial auctions: a survey. available from: `http://www.kellogg.nwu.edu/faculty/vohra/htm/res.htm`, 2000.
4. 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.
5. F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, pages 259–268, 1998.
6. 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.
7. E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samrodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing*, 2002.
8. O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.
9. O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653–750, 1998.
10. M. Grotschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1, 1981.
11. 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.
12. S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 96–107, 2000. To appear in JACM.
13. B. Lehmann, D. Lehmann, and N. Nisan. Combinatorial auctions with decreasing marginal utilities. In *ACM Conference on Electronic Commerce*, pages –, 2001.
14. L. Lovász. Submodular functions and convexity. *Mathematical Programming: The State of the Art*, pages 235–257, 1983.
15. M. Parnas, D. Ron, and R. Rubinfeld. On testing convexity and submodularity. Available from: `http://www.eng.tau.ac.il/~danar/papers.html`, 2002.
16. D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001.
17. R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.
18. A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory B*, 80:346–355, 2000.