POLYNOMIAL-TIME ALGORITHMS FOR REGULAR SET-COVERING AND THRESHOLD SYNTHESIS

Uri N. PELED*
Mathematics, Statistics, and Computer Science Department, University of Illinois at Chicago, Chicago, IL 60680, USA

Bruno SIMEONE**
Department of Statistics, University of Rome 'La Sapienza', Rome, Italy

Received 3 November 1981
Revised 11 December 1984

A set-covering problem is called regular if a cover always remains a cover when any column in it is replaced by an earlier column. From the input of the problem — the coefficient matrix of the set-covering inequalities — it is possible to check in polynomial time whether the problem is regular or can be made regular by permuting the columns. If it is, then all the minimal covers are generated in polynomial time, and one of them is an optimal solution. The algorithm also yields an explicit bound for the number of minimal covers. These results can be used to check in polynomial time whether a given set-covering problem is equivalent to some knapsack problem without additional variables, or equivalently to recognize positive threshold functions in polynomial time. However, the problem of recognizing when an arbitrary Boolean function is threshold is NP-complete. It is also shown that the list of maximal non-covers is essentially the most compact input possible, even if it is known in advance that the problem is regular.

Keywords. Set-covering, regular, knapsack, polynomial algorithm, NP-completeness.

1. Introduction

The set-covering problem is

\begin{align*}
\text{minimize} & \quad cy, \\
\text{subject to} & \quad Ay \geq e, \\
& \quad y_i = 0 \text{ or } 1,
\end{align*}

where \( A \) is a given \( m \times n \) 0–1 matrix, \( c \) is a given row \( n \) vector, and \( e \) denotes the column \( m \) vector of 1's. Here and in what follows, an inequality between vectors denotes the corresponding inequalities between their components. We use the word point to mean a 0–1 \( n \) vector. To avoid variables that can be immediately fixed to

*Partially supported by The National Science Foundation Grant MCS-80-04105 and by NATO Research Grant 105-80.
**Partially supported by NATO Research Grant 105-80.
1 and redundant constraints, we assume that $c \geq 0$ and that no two distinct rows $a, a'$ of $A$ satisfy $a \succeq a'$. A point $y$ satisfying the constraints is called a cover. Thus an optimal solution exists among the minimal covers (covers $y$ such that no other cover $y'$ satisfies $y' \succeq y$). In the terminology of Edmonds and Fulkerson [3], the minimal covers are the blocking clutter to the rows of $A$. The minimal covers can in principle be generated from the rows of $A$, but this cannot be done in time polynomial in the input size $mn$, simply because there may be exponentially many minimal covers. For example, if $n = 2k$, $m = k$, and the rows of $A$ are

$$
\begin{align*}
1 & 1 0 0 \cdots 0 0 \\ 0 & 0 1 1 \cdots 0 0 \\ & \ddots \ddots \ddots \ddots \\ & \ddots \ddots \ddots \ddots \\ 0 & 0 0 0 \cdots 1 1
\end{align*}
$$

then $y$ is a minimal cover if and only if $Ay = e$, and so there are $2^k$ minimal covers. This is not surprising in view of the NP-completeness of the set-covering problem.

However, under additional assumptions to be described below, all the minimal covers can be listed in polynomial time, and hence the set-covering problem can be solved in polynomial time by computing the objective function for each of them. Moreover, the assumptions can be checked in polynomial time.

More specifically, the set-covering problem is called regular when for every cover $y$ with $y_i = 0, y_{i+1} = 1$, the point $y + e_i - e_{i+1}$ is also a cover, where $e_k$ denotes the $k$-th unit vector. Equivalently, for every cover $y$ with $y_i = 0, y_j = 1, i < j$, the point $y + e_i - e_j$ is also a cover. The main result of this work is the following.

**Theorem 1.** There is an algorithm that, given the $m \times n$ matrix $A$ of a regular set-covering problem, lists all the minimal covers in time polynomial in the input size $mn$. In fact, there are no more than $mn + m + n$ minimal covers.

Moreover, given the matrix $A$ of a set-covering problem, it is possible to check in polynomial time whether the problem is regular. Furthermore, it is possible to construct in polynomial time a permutation $\pi$ of the variables such that either $\pi$ transforms the problem into a regular one, or else no permutation transforms the problem into a regular one. The details of how to construct $\pi$ and, after applying it, how to check for regularity are stated in Section 2. The algorithm of Theorem 1 is given in Section 3.

One important consequence of Theorem 1 is the following.

**Theorem 2.** There is an algorithm that, given the $m \times n$ matrix $A$ of a set-covering problem, decides in time polynomial in $mn$ whether there exists a single linear inequality $dy \geq 1$ having precisely the same $0 \leq y$ solutions $y$ as $Ay \geq e$. If one exists, the algorithm finds such $d$. 
Regular set-covering and threshold synthesis

Theorem 2 enables one to decide constructively in polynomial time whether a given set-covering problem has the same feasible solutions as a knapsack problem in inequality form in the same variables. (Converting the set-covering inequalities into equations by means of surplus variables and aggregating the equations to a single equation does not solve the problem, because of the extra variables).

In Boolean function terminology, Theorem 1 says that there exists a polynomial-time algorithm for dualizing a regular Boolean function in disjunctive normal form (DNF); Theorem 2 says that there is a polynomial-time algorithm for recognizing positive threshold functions in DNF. Theorem 2 answers a long-standing open question concerning the complexity of the classical threshold synthesis problem.

Although every threshold function can be made positive by negating suitable variables, and in contrast with Theorem 2, the threshold recognition problem becomes NP-complete when the positivity assumption is dropped:

**Theorem 3.** It is NP-complete to decide whether an arbitrary DNF represents a threshold function.

Theorems 2 and 3 are treated in Section 4.

Finally we address the question of the input size for set-covering and regular set-covering problems. Above we took $mn$ as the input size. Since $m = O(n^{-1/2}2^n)$, the algorithms do not run in time polynomial in $n$. The question then arises whether $mn$ is an inflated measure of the input size, so that there might be some encoding of the constraints $Ay \geq e$ whose size is polynomial in $n$, say. In Section 5 we use estimates of the number of set-covering and regular set-covering problems on $n$ variables to show that every encoding must use $\Omega(n^{-1/2}2^n)$ bits for some set-covering problem and $\Omega(n^{-3/2}2^n)$ bits for some regular set-covering problem. Therefore the method that encodes the problem by listing the 0–1 matrix $A$ is inflated only by a factor of $O(n)$ or $O(n^2)$ compared to the most compact method possible.

2. Checking for regularity

It is convenient to work with the variables $x_i = 1 - y_i$, $i = 1, \ldots, n$, in terms of which the set-covering constraints read $Ax \leq b$, where $b = Ae - e$. Let $F_A$ denote the set of feasible points, namely those points $x$ satisfying $Ax \leq b$. The $F$ that arise in this way from set-covering problems are precisely the independence systems (if $x' \leq x$ and $x$ is feasible, then $x'$ is feasible). An independence system can be specified by listing its maximal feasible points (MFP), or its minimal infeasible points (MIP) – the rows of $A$ in our case. The support of a point $x$, $\text{supp}(x)$, is defined as $\{j : j = 1, \ldots, n, x_j = 1\}$. We say that $x$ lies above $x'$, or that $x'$ lies below $x$, when $\text{supp}(x') \subseteq \text{supp}(x)$, in other words when $x' \leq x$. We say that $x'$ is a right shift of $x$, or that $x$ is a left shift of $x'$, when $\text{supp}(x) = \{i_1, i_2, \ldots, i_r\}$, $\text{supp}(x') = \{j_1, j_2, \ldots, j_r\}$,
and \( i_1 \leq j_1, i_2 \leq j_2, \ldots, i_r \leq j_r \). It is not hard to see that \( F \) arises from a regular set-covering problem if and only if \( F \) is an independence system closed under right shifts. In that case we say that \( A \) is regular. For regular \( A \), \( F_A \) can be specified by listing its roofs, those MIP \( x \) such that every right shift of \( x \) other than \( x \) itself is feasible. It can also be specified by its ceilings, those MFP \( x \) such that every left shift of \( x \) other than \( x \) itself is infeasible. Roofs and ceilings were introduced by Bradley, Hammer and Wolsey [1].

We define a partial order \( \succeq \) on \( \{1, \ldots, n\} \) with respect to \( A \) by writing, for \( i \neq j \), \( i \succeq j \) when \( F(i,j) \subseteq F(j;i) \). Here \( F(i,j) \) denotes the set of points of \( F_A \cap \{x: x_i = 1, x_j = 0\} \), taken without their \( i \)-th and \( j \)-th components. In other words, \( i \succeq j \) when every feasible point \( x \) satisfying \( x_i = 1, x_j = 0 \) remains feasible when its \( i \)-th and \( j \)-th components are exchanged. Note that \( F(i,j) = F_M \), where the minor \( M \) is obtained from \( A \) by dropping all rows having a 1 in the \( j \)-th column and suppressing the \( i \)-th and \( j \)-th columns. We also use \( i \sim j \) to denote the assertion \( F(i,j) = F(j;i) \). Then \( A \) is regular if and only if \( 1 \succeq 2 \succeq \ldots \succeq n \) with respect to \( A \).

In order to check the relations \( 1 \succeq 2 \succeq \ldots \succeq n \) with respect to \( A \), we can use the following easy result [14].

**Theorem 4 (Quine).** Let \( A \) and \( B \) be 0–1 matrices with the same number of columns. Then \( F_B \subseteq F_A \) if and only if every MIP of \( A \) lies above some MIP of \( B \).

It is easy to use this theorem to check in polynomial time whether \( A \) is regular. For \( i = 1, \ldots, n - 1 \) we check \( F(i,i+1) \subseteq F(i+1;i) \) by using the condition in Theorem 4. To do so requires no more than comparing each MIP of \( F(i+1;i) \) with each MIP of \( F(i;i+1) \).

It is possible that \( A \) is not regular, but some permutation of its columns makes it regular. To find such a permutation we use a result of Winder [17], a necessary condition for \( i \succeq j \) with respect to a 0–1 matrix \( A \) with \( n \) columns. Let \( C \) be the \( n \times n \) matrix having in row \( i \) and column \( j \) the number of MIP \( x \) of \( A \) satisfying \( x_i = 1, x_1 + \ldots + x_n = j \). This matrix can be computed in time \( O(mn) \) by examining each MIP and collecting contributions to the entries of \( C \).

**Theorem 5.** (Winder). Let \( C \) be computed for the 0–1 matrix \( A \) as above, and assume \( i \succeq j \) with respect to \( A \). If \( i \sim j \) holds, then the \( i \)-th and the \( j \)-th rows of \( C \) are equal. If not, then the \( i \)-th row of \( C \) is lexicographically greater than the \( j \)-th row.

To use Theorem 5 we merely sort the rows of \( C \) lexicographically in time \( O(n^2 \log n) \), so that the \( \pi(1) \)-th row is largest, the \( \pi(2) \)-th row is second largest, and so on. Then we check the relations \( \pi(1) \succeq \pi(2) \succeq \ldots \succeq \pi(n) \) with respect to \( A \) using Theorem 4. (We only have to check \( \pi(i) \succeq \pi(i+1) \) when the \( i \)-th and \( (i+1) \)-st rows of \( C \) are unequal.) If all these relations hold, then the permutation \( \pi \) of the columns of \( A \) makes \( A \) regular. If not, then no permutation makes \( A \) regular.
3. The algorithm

Given an $m \times n$ regular 0-1 matrix $A$, we show here how to list all the MFP of $A$ in polynomial time, and obtain a simple bound on their number in terms of $n$ and $m$.

Let us introduce a total (linear) order on the $2^n$ points $x$. The *positional representation* of $x$ is the $n$-vector whose components are the elements of $\text{supp}(x)$ in increasing order followed by zeros. For example, the positional representation of $(0,1,0,1,0)$ is $(2,4,0,0,0)$. The total order of the points is defined to be the same as the lexicographical order of their positional representations. For example, for $n = 3$ the order is $(0,0,0), (1,0,0), (1,1,0), (1,1,1), (1,0,1), (0,1,0), (0,1,1), (0,0,1)$. The immediate successor of $x$ in this order is denoted by $\text{succ}(x)$ and the immediate predecessor by $\text{pred}(x)$.

To give an explicit formula for $\text{succ}(x)$, and to express the algorithm below, we introduce some more notation. For any point $x$, let $b(x)$ be the largest index $j$ such that $x_j = 1$ ($b(x) = 0$ if no such $j$ exists, namely if $x = 0$), and let $a(x)$ be the largest index $j$ such that $x_j = 0$ and $x_{j+1} = 1$ ($a(x) = 0$ if no such $j$ exists). The *fill-up* of $x$, denoted $\text{fill}(x)$, is $x + e_{b(x)} - 1$ (undefined if $b(x) = n$). The *bottom right shift* of $x$, denoted $\text{brs}(x)$, is $x - e_{b(x)} + e_{b(x)+1}$ (undefined if $b(x)$ is 0 or $n$). The *truncation* of $x$, denoted $\text{trunc}(x)$, is $x - \sum \{e_i : a(x) + 1 \leq i \leq b(x)\}$. It is easy to see that

$$\text{succ}(x) = \begin{cases} \text{fill}(x), & \text{if } x_n = 0, \\ \text{brs}(x - e_n), & \text{if } x_n = 1. \end{cases}$$

In particular, $\text{succ}(x)$ is undefined only for $x = e_n$, which is the last point; and every point is a successor except 0, which is the first point.

Another concept used by the algorithm below is that of a shelter of a regular $A$, which is a MIP having some of the properties of a roof. A roof has been defined as a MIP $x$ such that every right shift of $x$ other than $x$ itself is feasible. A *shelter* is a MIP $x$ such that $\text{brs}(x)$, if defined, is not a MIP. Therefore we can generate the shelters of $A$ from the list of the $m$ MIP in time $O(nm^2)$. When we have the shelters, we can sort them according to the total order introduced above in time $O(nq \log q)$, where $q$ is the number of shelters and $q \leq m$. The sorted list of shelters, followed by a dummy shelter, is the input to the algorithm below. The idea behind the algorithm is to
scan all the points in the total order, skipping over intervals that cannot contain any MFP. The information imbedded in the sorted list of shelters enables us to do this in polynomial time.

**Hop-Skip-and-Jump Algorithm**

\[ s := \text{first shelter on the list}; \]

START: if \( s = 0 \) then stop \( \{ f = 1 \} \) else \( x := 0; \)

while true do

begin \{outer while\}

while \( x \neq s - e_b(s) \) do \{inner while\}

if \( x_n = 0 \)

then FILL-UP: \( x := \text{fill}(x) \)

else SKIP: begin \{skip\}

output \( x; \)

\( y := \text{trunc}(x); \)

if \( y = 0 \) then stop else \( x := \text{brs}(y) \)

end; \{skip\}

\{end inner while\}

LEAP: begin \{leap\}

if \( s_n = 0 \) then HOP: \( x := \text{brs}(s) \)

else JUMP: begin \{jump\}

output \( x; \)

if \( s = e_n \) then stop else \( x := \text{succ}(s) \)

end; \{jump\}

\( s := \text{next shelter on the list} \)

end \{leap\}

end \{outer while\}

**Remark.** If \( s \) is the dummy shelter, then \( x \neq s - e_b(s) \) is considered to be true.

The following invariant assertion will be used to prove the validity of the algorithm.

**Lemma 2.** Immediately after \( \text{START} \), and after each iteration of the inner while loop and of \( \text{LEAP} \), \( x \) is feasible and \( s \) is the first point following \( x \) that is a shelter (the first shelter that follows \( x \)). Hence the algorithm outputs only feasible points.

**Proof.** The assertion certainly holds just after \( \text{START} \): as soon as \( x \) is defined, \( s \neq 0 \) which means that there are feasible points, so \( x = 0 \) is obviously feasible. We shall assume the assertion to be true for the previous \( x \) and \( s \), which we denote by \( \xi \) and \( \sigma \), respectively, and prove it for the present \( x \) and \( s \). We distinguish two cases by which point in the algorithm is considered.

Case 1: Just after the inner while loop. In this case \( s = \sigma \).
Case 1a: $x$ was generated in SKIP. Then $\xi_n = 1$ and the points $\text{succ}(\xi)$, $\text{succ}(\text{succ}(\xi)), \ldots, x$ are right shifts of points lying below $\xi$. By regularity and the feasibility of $\xi$, all these points are feasible. Therefore $x$ is feasible and there are no shelters between $\xi$ and $x$. Therefore $s = \sigma$ is the first shelter that follows $x$.

Case 1b: $x$ was generated in FILL-UP. Then $x = \text{succ}(\xi) = \text{fill}(\xi)$ and we only have to prove that $x$ is feasible. Assume that $x$ is infeasible. Then $x$ is a MIP, by the feasibility of $\xi$ and regularity. Let $y^0 = x$. If $y^0$ is not a shelter, then $y^1 = \text{brs}(y^0)$ exists and is infeasible by definition of a shelter, hence $y^1$ is a MIP by Lemma 1. If $y^1$ too is not a shelter, then $y^2 = \text{brs}(y^1)$ exists and is a MIP, and so on. The sequence $y^0, y^1, \ldots$ must terminate because $\text{brs}$ cannot be applied indefinitely, hence one of the $y^k$ is a shelter, possibly equal to $x$. For $i < k$, the points $\text{succ}(y^i)$, $\text{succ}(\text{succ}(y^i)), \ldots, \text{pred}(y^{i+1})$ lie above $y^i$ and therefore are not MIP and are not shelters. Hence $y^k$ is the first shelter that follows $\xi$. But this is $\sigma$ by assumption, and by the construction of $y^k$, $\xi = \sigma - e_{b(\sigma)}$. In that case the algorithm should have leaped from the inner while loop. This contradiction proves that $x$ is feasible.

Case 2: Just after LEAP. In this case $\xi = \sigma - e_{b(\sigma)}$.

Case 2a: $x$ was generated in JUMP. Then $x = \text{succ}(\sigma) = \text{brs}(\sigma - e_n)$ is feasible since $\sigma$ is a MIP and by regularity. Since $s$ is the first shelter that follows $\sigma$, $s$ is the first shelter that follows $\text{succ}(\sigma) = x$.

Case 2b: $x$ was generated in HOP. Then $x = \text{brs}(\sigma)$ is feasible by the definition of shelter. The points $\text{succ}(\sigma)$, $\text{succ}(\text{succ}(\sigma)), \ldots, \text{pred}(x)$ all lie above $\sigma$ and are different from $\sigma$, hence they are not MIP and are not shelters. Therefore the first shelter that follows $x$ is the first shelter that follows $\sigma$, which is $s$. □

Lemma 3. The Hop-Skip-and-Jump Algorithm outputs only MFPs.

Proof. Let $x$ be a point output by the algorithm. By Lemma 2, $x$ is feasible and we only have to prove that it is maximal. We distinguish two cases by where $x$ is output.

Case 1: $x$ is output in SKIP. Then $x_n = 1$. If the algorithm has never been in LEAP before, then $x = e_1 + \ldots + e_n$, which is clearly maximal. Therefore we assume that the algorithm has been in LEAP and that $a(x) \neq 0$. By regularity, to prove that $x$ is a MFP, it suffices to show that $x + e_{a(x)}$ is infeasible. Let $\sigma$ denote the value of $s$ just before the most recent LEAP, and let $\xi$ denote the value of $x$ just after this LEAP. Then $x$ has been obtained from $\xi$ by a sequence of FILL-UPs, hence

$$x = \xi + \sum \{e_i : b(\xi) + 1 \leq i \leq n\} \quad \text{and} \quad a(x) = a(\xi).$$

Also $\xi$ is either $\text{brs}(\sigma)$ if $\sigma_n = 0$ or $\text{brs}(\sigma - e_n)$ if $\sigma_n = 1$. In both cases $a(\xi) = b(\sigma)$, and so $x + e_{a(x)}$ lies above $\sigma$ and is infeasible.

Case 2: $x$ is output in JUMP. Then $s_n = 1$ and $x = s - e_n$. Since $x + e_n = s$ is infeasible, every point different from $x$ and lying above $x$ is infeasible by regularity, so $x$ is a MFP. □

Lemma 4. The Hop-Skip-and-Jump Algorithm outputs every MFP.
**Proof.** If \( x^1, x^2, \ldots, x^r \) are generated by successive FILL-UPs, then for \( i < r \), \( x^i \) is not output, but it lies below \( x^{i+1} \) and therefore is not a MFP. The points omitted by SKIP are not MFP, because they lie (strictly) below the feasible point that SKIP outputs. Consider the points omitted by LEAP. Let \( x \) and \( s \) have their values just before the LEAP, so that \( x = s - e_{b(s)} \). If \( s_n = 0 \), then the omitted points are \( x, \text{succ}(x), \ldots, \text{pred}(brs(s)) \). Each of these points is either a right shift of \( x \) or a left shift of a point lying above \( s \). In the first case the omitted point is feasible, but not maximal, because \( x \) lies (strictly) below the feasible point \( brs(s) \). In the second case the omitted point is infeasible. If \( s_n = 1 \), then the omitted points are \( \text{succ}(x), \text{succ}(\text{succ}(x)), \ldots, s \). Each of these points lies above \( s \) and is infeasible. □

Lemma 3 and 4 show that the algorithm is correct.

**Lemma 5.** The Hop-Skip-and-Jump Algorithm runs in \( O(n^3m) \) time.

**Proof.** At most \( n \) consecutive FILL-UPs can be executed before a SKIP or LEAP occurs, since every FILL-UP increases the sum \( x_1 + \ldots + x_n \). At most \( n \) SKIPS can be executed before a LEAP occurs, because for two consecutive SKIPS, possibly separated by FILL-UPs but not by LEAP, the sum \( x_1 + \ldots + x_n \) just after the later SKIP is smaller than just after the earlier SKIP. It follows that the inner while loop can be iterated no more than \( n^2 \) consecutive times before a LEAP occurs. But every LEAP changes the shelter, so that there are at most \( q \) LEAPs. After the last LEAP there can be at most \( n^2 \) additional iterations of the inner while loop before termination, when \( s \) is the dummy shelter. Therefore the total number of iterations is \( O(n^2(q + 1)) = O(n^2m) \). Each iteration involves simple checks and operations like \( \text{brs} \), \( \text{trunc} \), and so on that can be implemented in time \( O(n) \), from which the time bound in the lemma follows. □

**Theorem 6.** If a regular \( m \times n \) matrix has \( r \) MFP, then \( r \leq mn + m + n \).

**Proof.** From the proof of Lemma 5, there are at most \( n \) SKIPS before the next LEAP and after the last LEAP, and every SKIP outputs just one point. Also every LEAP outputs at most one point and there are at most \( q \) LEAPs. Therefore the algorithm outputs at most \( n(q + 1) + q \) points. □

Lemmas 3, 4, 5 and Theorem 6 constitute a proof of Theorem 1.

The Hop-Skip-and-Jump Algorithm is an improved version of the algorithm of Hammer, Peled and Pollatschek [6] to dualize a regular function. Their algorithm outputs some non-maximal feasible points in addition to all the MFP. They did not analyze the running time, but observed empirically its linear dependence on \( m \), which indicates that the algorithm tends to process at least a fixed fraction of the shelters before termination. Our algorithm enabled us to obtain Theorem 6 directly.

It is interesting to compare Theorem 1 with the results of Lawler, Lenstra and
Rinnooy Kan [10] about knapsack problems. Their argument, slightly generalized to regular set-covering problems, shows that if a regular set-covering problem with \( n \) columns has \( r \) MFP and is specified by a unit-time oracle to test feasibility, then its \( r \) MFP can be generated in time polynomial in \( n \) and \( r \). Our results are that if the problem is specified by the list of its \( m \) MIP, then its \( r \) MFP can be generated in time polynomial in \( n \) and \( m \), and we have an explicit polynomial bound for \( r \) in terms of \( m \) and \( n \).

4. Polynomial-time recognition of threshold functions

In this section we show how to check in polynomial time whether a given set-covering problem is equivalent to a knapsack problem in inequality form in the same variables, in the sense of having the same feasible points. In Boolean function terms this amounts to checking in polynomial time whether a Boolean function in positive DNF is a threshold function. We also show that the corresponding problem for arbitrary DNF is NP-complete.

Consider again the set-covering constraints in the form \( Ax \leq b \) as in Section 2. \( A \) will be called a **knapsack matrix** when there exists a single linear inequality \( w_1 x_1 + \ldots + w_n x_n \leq t \) that is satisfied by the points of \( F_A \) and violated by the points not in \( F_A \). This is equivalent to the existence of a hyperplane separating the convex hull of the feasible points from the convex hull of the infeasible points. It follows from the separation theorem for polytopes that \( A \) is not a knapsack matrix if and only if there exist an integer \( k \geq 2 \), \( k \) feasible points, not necessarily distinct, \( p^1, p^2, \ldots, p^k \), and \( k \) infeasible points, not necessarily distinct, \( q^1, q^2, \ldots, q^k \), such that \( p^1 + \ldots + p^k = q^1 + \ldots + q^k \) (the summations taken as for vectors in \( R^n \)). This condition is not very useful for recognizing knapsack matrices.

There are other, more useful, necessary conditions for being a knapsack matrix. Let \( A \) be a 0–1 matrix with \( n \) columns and let \( I \) and \( J \) be disjoint subsets of the index set \( \{1, \ldots, n\} \). We generalize the notation \( F(i; j) \) of Section 2, denoting by \( F(I; J) \) the set of points of \( F_A \cap \{x: x_i = 1 \text{ for all } i \in I \text{ and } x_j = 0 \text{ for all } j \in J\} \), written without the components indexed by \( I \) and \( J \). We use obvious simplified notations when \( I \) or \( J \) is a singleton. We say that \( A \) is \( k \)-monotonic if \( F(I; J) \subseteq F(J; I) \) or \( F(J; I) \subseteq F(I; J) \) whenever \( I \) and \( J \) are disjoint sets satisfying \( |I| + |J| \leq k \). Every knapsack matrix is \( k \)-monotonic for every \( k \). Indeed, if \( w_1 x_1 + \ldots + w_n x_n \leq t \) is a separating hyperplane for \( A \), then \( \sum \{ w_i : i \in I \} = \sum \{ w_j : j \in J \} \) implies that \( F(I; J) \subseteq F(J; I) \). However, there exist non-knapsack matrices that are \( k \)-monotonic for every \( k \) [16].

In some special cases there are known polynomial-time algorithms for recognizing knapsack matrices. One case occurs when each row of \( A \) has exactly two ones, that is, when the constraints \( Ax \leq b = e \) are vertex-packing constraints in a graph. (Set-packing constraints \( Ax \leq e \) for an arbitrary 0–1 matrix \( A \) can easily be converted into vertex-packing constraints in the same variables). Chvátal and Hammer proved [2]
that in this particular case \( A \) is a knapsack matrix if and only if \( A \) is 2-monotonic. They also gave an efficient recognition algorithm, several graph-theoretic characterizations and a method to construct a separating hyperplane. A positive 0-1 matrix \( A \) is called matroidal when \( F_A \) is the collection of the independent sets of a matroid, meaning that for each point \( a \) there is a number \( r(a) \) such that all the maximal solutions of \( x \in F_A \) and \( x \leq a \) satisfy \( x_1 + \ldots + x_n = r(a). \) Edmonds, Wolsey [18] proved that a regular matrix is matroidal if and only if it has a unique ceiling. See Euler [4] for a substantial generalization. Giles and Kannan [5] proved that a matroidal matrix is knapsack if and only if it is 3-monotonic.

**Proof of Theorem 2.** It is not hard to see that every knapsack matrix \( A \) has a separating hyperplane of the form \( dx = d_1 x_1 + \ldots + d_n x_n = 1 \) with all \( d_i > 0 \) that does not pass through any 0–1 point. The coefficients \( d_i \) are then characterized as the solutions of the following system of strict linear inequalities with 0, ±1 coefficients.

\[
\begin{align*}
dx &< 1 \quad \text{for every MFP } x, \\
*dx &> 1 \quad \text{for every MIP } x, \\
d_i &> 0 \quad \text{for } i = 1, \ldots, n.
\end{align*}
\]

If \( A \) has \( m \) rows and \( q \) MFP, then by Theorem 1, \( q \leq mn + m + n \) and the \( q \) MFP can be obtained from \( A \) in polynomial time. Karmarkar’s algorithm for linear programming [8] can find a solution \( d \) of the above inequalities or show that none exists, in which case \( A \) is not a knapsack matrix. Its running time is \( O(n^{3.5} L^2) \), where \( L \) is the total number of bits needed to write the coefficients of the inequalities in absolute value. In our case there are \( q + m + n \) inequalities requiring at most \( n + 1 \) bits each, so that \( L \leq (n + 1)(mn + 2m + 2n) = O(mn^2) \). This completes the proof. \( \square \)

In actual computations it is much better to use the following facts. Let \( dx = 1 \) be a separating hyperplane for the regular knapsack matrix \( A \) with \( n \) columns. If for some \( i < n \) it is not the case that \( i \sim i + 1 \), then \( d_i > d_{i+1} \). Also if \( F(\emptyset; n) \neq F(n; \emptyset) \) – which means that the value of \( x_n \) can affect feasibility – then \( d_n > 0 \). On the other hand there exists a solution \( d \) satisfying \( d_i = d_{i+1} \) for all \( i < n \) such that \( i \sim i + 1 \), and also \( d_n = 0 \) if \( F(\emptyset; n) = F(n; \emptyset) \). The conditions \( d_i \geq d_{i+1} \geq \ldots \geq d_n \geq 0 \), which may be imposed on \( d \) by the above, make the inequalities \( dx < 1 \) (\( dx > 1 \)) redundant unless \( x \) is a ceiling (roof). This leads to a great reduction in the number of inequalities. Furthermore, we may keep only one representative index from each equivalence class under \( \sim \) (and drop completely the last class if \( F(\emptyset; n) = F(n; \emptyset) \)). This reduces further the number of unknowns. The new inequalities now acquire integer coefficients other than 0, ±1, but on the whole \( L \) still decreases.

Theorems 1 and 2 have significant formulations in terms of Boolean functions. For a 0–1 matrix \( A \), let

\[
f(x) = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n} x_i^{a_{ij}}.
\]
Clearly $f(x) = 0$ if and only if $Ax \leq b$, namely if and only if $x \in F_A$. The Boolean function $f$ is positive ($x \leq x'$ implies $f(x) \leq f(x')$). The function $f$ is called a threshold function when $A$ is a knapsack matrix. The notions of MIP, MFP, regularity and $k$-monotonicity for $f$ correspond to the same notions for $A$. The 1-monotonic functions are called unate. If $f$ is unate, it becomes positive when suitable variables are negated. The positive 2-monotonic functions become regular when a suitable permutation, described in Section 2, is applied to their variables. Thus the positive (regular) functions can serve as canonical forms of the 1-monotonic (2-monotonic) functions under the operations of negations (negations and permutations) of the variables. The dual Boolean function to $f$ is defined by $f^d(x) = 1 - f(e - x)$, and one sees immediately that $x$ is a MIP of $f^d$ if and only if $e - x$ is a MFP of $f$. Hence Theorem 1 can be restated as follows:

'There is an algorithm that, given a regular Boolean function $f$ in DNF, computes $f^d$ in DNF in polynomial time.'

Theorem 2 can be restated as follows:

'There is an algorithm that, given a positive Boolean function $f$ in DNF, decides in polynomial time whether $f$ is a threshold function, and if so, constructs a separating hyperplane.'

The problem of recognizing a threshold function and constructing a separating hyperplane is known as the synthesis problem of threshold logic [11,16]. A number of algorithms have appeared for this problem, but to the best of our knowledge, none has been proved to run in polynomial time.

**Proof of Theorem 3.** Consider the following problems.

**DNF-SAT:** Input is a Boolean function in DNF. The property to check is whether $f \equiv 1$.

**DNF-THR:** Input is a Boolean function in DNF. The property to check is whether $f$ is a threshold function.

DNF-SAT is equivalent to the SATISFIABILITY problem (check whether a conjunctive normal form is identically zero). Theorem 3 is equivalent to the NP-completeness of DNF-THR. We reduce DNF-SAT to DNF-THR as follows. Given a DNF $f(x)$, construct in linear time the DNF $g(x, a_1, a_2) = f(x) a_2 \vee a_1 a_2 \vee a_1 a_2$. If $f \equiv 1$, then $g(x, a_1, a_2) = a_2 \vee a_1$ is a threshold function. If not, let $f(x^0) = 0$, then $g(x^0, 0, 0) = g(x^0, 1, 1) = 1$, $g(x^0, 0, 1) = g(x^0, 1, 0) = 0$ and $(x^0, 0, 0) + (x^0, 1, 1) = (x^0, 0, 1) + (x^0, 1, 0)$ show that $g$ is not a threshold function. This completes the proof. \[\square\]

**Remark.** The same construction shows that it is NP-complete to decide whether a given DNF, not necessarily in unate form, represents a unate function. The problem
of Theorem 3 remains NP-complete even if it is known in advance that \( f \) is unate, but not which variables should be negated to make it positive [7].

5. The input size

In this section we ask whether \( mn \) is an appropriate measure of the input size needed to specify a positive or regular function on \( n \) variables having \( m \) MIP. Consider first the positive functions. Every positive function is determined by its MIP, and a set of points constitute the MIP of some positive function if and only if none of them lies below another one. It follows that \( \psi(n) \), the number of positive functions on \( n \) variables, is equal to the number of antichains of the poset \( B(n) \) of all the \( 2^n \) points ordered by the relation ‘lies below’, namely the Boolean algebra on \( n \) atoms. By Sperner’s theorem, the largest size of an antichain in \( B(n) \) is \( E_n = C(n, \lfloor n/2 \rfloor) \), where \( C \) denotes the binomial coefficient and \( \lfloor a \rfloor \) denotes the greatest integer not exceeding \( a \). The Stirling approximation gives \( E_n = O(n^{-1/2} 2^n) \).

An antichain of size \( E_n \) is the set \( A \) of all points with exactly \( \lfloor n/2 \rfloor \) ones. Since every subset of \( A \) is also an antichain, we have \( \log_2 \psi(n) \geq E_n \). Since every encoding of a general positive function must be able to distinguish between \( \psi(n) \) different functions, it must use at least \( \log_2 \psi(n) \) bits of information for some function. Since \( E_n \geq m \) by Sperner’s theorem, the encoding that uses \( mn \) bits by listing the \( m \) MIP is inflated by a factor of only \( O(n) \) compared to the most compact encoding possible. We remark that the estimate \( E_n \) for \( \log_2 \psi(n) \) is very accurate, and in fact [9]

\[
(1 + c' \log n/n)E_n \geq \log_2 \psi(n) \geq (1 + c 2^{-n/2})E_n.
\]

Let us now turn to the regular functions. Every regular function is determined by its roofs, and a set of points constitute the roofs of some regular function if and only if none of them lies below a right shift of another one. Therefore \( \varrho(n) \), the number of regular functions on \( n \) variables, is equal to the number of antichains in the poset \( M(n) \) of all the \( 2^n \) points ordered by the relation ‘lies below a right shift of’. As before, \( \log_2 \varrho(n) \) is bounded below by the largest size of an antichain in \( M(n) \). \( M(n) \) is a ranked poset in the sense that if the rank of the point \( (x_1, \ldots, x_n) \) is defined as \( x_1 + 2x_2 + \ldots + nx_n \), then a point can cover only points whose rank is one less than its own. Let \( A_r \) denote the set of points of rank \( r \), \( r = 0, \ldots, n(n+1)/2 \). Then \( A_r \) is an antichain having the same size as \( A_s \), where \( s = n(n+1)/2 - r \). Stanley showed [15] that the sequence \( |A_r| \) is unimodal (first nondecreasing and then nonincreasing), so that the largest \( A_r \) occurs when \( r = \lceil n(n+1)/4 \rceil \), and that the largest \( A_r \) is a largest antichain of \( M(n) \). See also Proctor [13] for a proof using linear algebra. Thus the largest size of an antichain in \( M(n) \) is the number of points \( (x_1, \ldots, x_n) \) satisfying \( x_1 + 2x_2 + \ldots + nx_n = [n(n+1)/4] \), or the middle coefficient (coefficients) of the polynomial \( (1+q)(1+q^2)\ldots(1+q^n) \). From the results of Odlyzko and Richmond [12, Theorem 3], this coefficient is \( \sim (2/3\pi)^{1/2} n^{-3/2} 2^n \).
follows that every encoding of a general regular function on \( n \) variables must use at least \( \log_2 Q(n) \geq cn^{-3/2}2^n \) bits of information for some function. Therefore the encoding that lists all the MIP – roofs and nonroofs alike – is inflated by a factor of only \( O(n^2) \) compared to the most compact encoding possible.

Acknowledgement

We are grateful to T. Ibaraki for illuminating discussions on the complexity issues concerning Boolean functions.

References