Sparse Solutions of Sparse Linear Systems: Fixed-Parameter Tractability and an Application of Complex Group Testing*

Peter Damaschke†
Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract

A vector with at most k nonzeros is called k-sparse. We show that enumerating the support vectors of k-sparse solutions to a system $Ax = b$ of r-sparse linear equations (i.e., where the rows of A are r-sparse) is fixed-parameter tractable (FPT) in the combined parameter r, k. We give different branching algorithms based on the close relationship to the hitting set problem in fixed-rank hypergraphs. For $r = 2$ the problem is simple. For $0, 1$-matrices A we can also compute an $O(rk^r)$ kernel. For systems of linear inequalities we get an FPT result in the combined parameter d, k, where d is the total number of minimal solutions. This is achieved by interpeting the problem as a case of group testing in the complex model. The problems stem from the reconstruction of chemical mixtures by observable reaction products.

Keywords: sparse vector, linear system, hitting set, parameterized algorithm, enumeration, problem kernel, group testing

1 Introduction

Let A be an $m \times n$ matrix with entries $a_{ij} \geq 0$ ($1 \leq i \leq m$, $1 \leq j \leq n$), and let b be a vector of length m with entries $b_i \geq 0$. A vector with at most k nonzero entries is *k-sparse*. Given a number k, usually much smaller than n, we want to determine the k-sparse nonnegative solutions x of $Ax = b$, where the rows of A are r-sparse. We pose the same problem for systems of linear inequalities, where both relations $\leq$ and $\geq$ may appear mixed in the different rows.

This is certainly a fundamental problem, appearing in machine learning and related areas like inference or reconstruction problems in computational biology, see [13] for an example. A particular application we have in mind is the quantification of proteins in an unknown mixture. (For some background information on protein inference see [8, 16].) There the

*This is an improved and extended version of a paper presented at the *6th International Symposium on Parameterized and Exact Computation IPEC 2011* at *ALGO 2011*, Saarbrücken (Germany), *Lecture Notes in Computer Science* (Springer), vol. 7112, pp. 94–105.

†Tel. 0046-31-772-5405. Fax 0046-31-772-3663.

columns of A correspond to candidate proteins, and the rows correspond to peptides, i.e., products of enzymatic digestion, or just masses of peptides. Entry a_{ij} is the number of occurrences of peptide i in protein j. The a_{ij} are, of course, nonnegative integers, moreover they are mostly 0, and the nonzeros are typically just 1. The real-valued vector b indicates the measured amounts of peptides, obtained by mass spectroscopy. We want to infer which proteins are in the mixture, and their amounts.

A is a matrix of simulated digestion results with several hundreds of thousands of rows and colums. However, some other input parameters are small, which suggests the question of parameterized complexity: After separation procedures some small number k of proteins are present, and a peptide typically appears in a small number r of candidate proteins, thus the rows of A are r-sparse. Due to these facts, only some hundreds of entries of b are nonzero. Of course, we can ignore rows i with $b_i = 0$, and we immediately know $x_j = 0$ if some i exists with $b_i = 0$ but $a_{ij} > 0$. After deletion of these trivial rows and columns there remains a submatrix of manageable size. We still denote the resulting system $Ax = b$, where b is now strictly positive. Simulations with protein data suggest that many rows are only 2-sparse. By solving $Ax = b$ we work under the idealized assumption that b has been accurately measured. Under experimental conditions with much noise it is more realistic to consider inequalities, in the simplest case just with a reliable lower und upper bound for each b_i (so that every peptide gives rise to two inequalities).

Formal notation and results. Let R be any set of rows, and let C be any set of columns of matrix A. We denote by $b[R]$ and $x[C]$ the vector b and x restricted to its entries corresponding to R and C, respectively. $A[R]$ and $A[C]$ denotes the submatrix of A restricted to R and C, respectively, and $A[R, C]$ denotes the submatrix of A being the intersection of $A[R]$ and $A[C]$. Sometimes we identify row and column sets with the sets of their indices, without risk of confusion. For any vector y, as usual, y_i is the entry at index i. The *support* of a vector is the set of indices with nonzero entries. A column set $C \neq \emptyset$ is called *feasible* if the system $A[C] \cdot x[C] = b$ has some solution where all entries of $x[C]$ are positive. A column set C is *minimal feasible* if C is feasible but no $C' \subset C$ is. The definition applies similarly to a system of linear inequalities.

A problem with input size n and some other parameter k is fixed-parameter tractable (FPT) if it is solvable in $f(k) \cdot p(n)$ time, where f is any computable function but p is a polynomial. When the polynomial factor is not in the focus, the time bound is often expressed as $O^*(f(k))$.

Finding a sparsest solution to a linear system is NP-hard in general [11, 15]. In Section 2 we show that enumerating all minimal feasible sets of at most k columns, for systems of linear equations with r-sparse rows, is an FPT problem in the combined parameter r, k. We give two different branching strategies, where the first strategy is superior for small k close to r, and the second one is clearly better when r is constant and k the "actual" parameter. For $r = 2$ the problem is polynomial, even very simple, and this observation may be used in heuristics that speed up the branching in cases where we have $r > 2$ but many 2-sparse rows appear in the matrix. Moreover, we refine the second branching strategy and combine it with a nontrivial algorithm for the closely related hitting set enumeration problem, which results in an improved base in the FPT time bound. Since this needs longer preparations, the

result is presented separately, in Section 3. Moving from algorithms to kernels, we compute in Section 4 an $O(rk^r)$ size full kernel, that is, a set of columns that includes all minimal feasible sets, by adaptation of an earlier result for hitting sets. In Section 5 we show that the same problem for systems of linear inequalities is in FPT in the combined parameter d, k, where d is the number of minimal feasible sets. This result is an application of a strategy for the more abstract *group testing problem in the complex model* (also known as searching for defective hyperedges; we shall give the necessary definitions later). Section 6 concludes the paper with some discussion. The algorithms combine linear algebra observations with FPT techniques. We also remark that d may be exponential in k. However, systems with random matrices often have unique sparse solutions, as is known in the *compressed sensing* literature (see references below). For our problem this gives hope for small d in real data sets. Even if a system has many solutions, one is not forced to list them all.

Related literature. As already indicated, our problem is close to hitting set enumeration in hypergraphs. This problem asks to enumerate the hitting sets of size at most k, that is, vertex sets that intersect all hyperedges, in a hypergraph of rank r, where the rank is the maximum size of the hyperedges. (The precise connection between the two problems will be explained in Section 2.) Hitting set enumeration is in FPT in the combined parameter r, k. More precisely, it can be solved in $O^*(r^k)$ time, based on the observation that some of the (at most) r vertices of any hyperedge must be put in the solution, and this can be done at most k times. Quite some work has been devoted to improved time bounds, see [9, 10]. While our branching approach is superficially similar to that for hitting set enumeration, it is not an immediate generalization.

Despite some connections to hypergraphs and graphs we hope to bring here some fresh contribution to the "not about graphs" direction of parameterized algorithms research that seems to be somewhat neglected.

For certain classes of matrices, such as those used in compressive sensing/sampling [2], sparsest solutions are unique and can be computed surprisingly simply, by a linear program that minimizes the sum of entries of x [1, 7, 14, 19]. However, in our case the matrices A are part of the input and cannot be chosen, thus we cannot assume special structural properties of A, and instead we have to consider the worst case. In general, the sparsest solution is not unique, as the vector b may be in the convex hulls of various small sets of columns of A. (Still we may first test for a given matrix A whether the linear program already yields some sparse solution.)

2 Row-Sparse Linear Systems of Equations

In this section we address the problem of enumerating all minimal feasible sets of at most k columns, thus also determining all k-sparse nonnegative solutions x, to a system $Ax = b$ where every row of A is r-sparse.

Geometrically the first (folklore) lemma says that a vector being in the convex hull of other vectors is already in the convex hull of a linearly independent subset of them.

Lemma 1 *Let C denote any set of columns in A. If C is minimal feasible then C is linearly independent.*

Proof. Since C is minimal feasible, b is a linear combination of the columns in C with positive coefficients, say $b = x_0c_0 + \ldots + x_nc_n$, with $x_i > 0$ for all i. Assume that some $c \in C$ in turn equals a linear combination of other columns in C, say $c_0 = y_1c_1 + \ldots + y_nc_n$. We can rewrite b as $b = (1-t)x_0c_0 + (x_1 + tx_0y_1)c_1 + \ldots + (x_n + tx_0y_n)c_n$, for any $0 \leq t \leq 1$. Starting with $t = 0$ we increase t as long as for every coefficient $x_i + tx_0y_i$ this expression for b remains nonnegative. If we reach $t = 1$ we get rid of c_0 in b. If the coefficient of some other column c_i, $i > 0$, becomes 0 earlier, we get rid of this c_i. Either case contradicts the minimality of C. $\diamond$

Proposition 2 *Every feasible set of columns is the union of some minimal feasible sets.*

Proof. Consider any feasible set C. If C itself is minimal, there is nothing to prove. Otherwise let $D \subset C$ be minimal feasible. Clearly, there exist positive solutions to $A[C] \cdot x[C] = b$ and $A[D] \cdot y[D] = b$. With a slight abuse of notation, let $y[C]$ be the vector obtained from $y[D]$ by filling all entries in $C \setminus D$ with zeros. Note that still $A[C] \cdot y[C] = b$. Since the two matrix-vector products above are equal to b, all numbers are nonnegative, and $D \subset C$, there must exist some index $i \in D$ where $y_i > x_i$. Let t be the largest number such that $x[C] - t \cdot y[C]$ is still nonnegative. Due to the previous observation we have $t < 1$, hence $A[C] \cdot (x[C] - t \cdot y[C]) = (1-t)b$, and multiplication with $1/(1-t)$ yields a nonegative solution z to $Az = b$ whose support C' fulfills $C' \subset C$, $C' \not\supset D$ (since $z_i = 0$), and $C \setminus C' \subseteq D$ (since $z_j > 0$ for all $j \in C \setminus D$). In words: All columns that we removed from the C belong to some minimal feasible set. We repeat this procedure with C' in the role of C, and so on. By an inductive argument, eventually every column of C is in some minimal feasible set. $\diamond$

Proposition 2 and Lemma 1 imply that the minimal feasible sets have the role of vertices of the (convex) space of nonnegative solutions to $Ax = b$. For each minimal feasible set C, the solution vector x with support C is unique (since the columns of C are linearly independent), and trivially, any convex linear combination of any nonegative solutions is a nonnegative solution, too. In this sense we have characterized all nonnegative solutions once we know the minimal feasible sets. This motivates the problem of enumerating these sets.

A tempting idea of a branching algorithm for the task is the following. Recall that all b_i are positive without loss of generality. Pick a row i and decide exactly which of the x_j, $a_{ij} > 0$, shall be positive. At least one of them must be positive, and since the rows are r-sparse, we get an $O(r)$ branching number. When all rows are treated, check whether the columns j of all positive x_j are linearly independent (cf. Lemma 1), and if so, compute the unique nonnegative solution, or find that there is none. It is important to notice the catch: In rows i where some x_j, $a_{ij} > 0$, are already deemed positive, there is an option not to select further positive variables, and then this branch does not reduce the parameter k. It may happen that the above branching rule has to stop, but the obtained set C of columns is not yet feasible. We may still follow the approach, but at this point we have to identify a "small" set of candidates to be added to C. The following lemma is the key.

Lemma 3 *Let C be a set of linearly independent columns such that $A[C] \cdot x[C] = b$ lacks a nonnegative solution. Then there exists a set R of at most $|C| + 1$ rows such that also $A[R, C] \cdot x[C] = b[R]$ lacks a nonnegative solution. Moreover, we can find such R in polynomial time.*

Proof. Since the columns in C are linearly independent, there is a set R' of $|C|$ rows such that $A[R', C]$ has full rank, and R' can be computed in polynomial time, e.g., by Gauss elimination. Note that $A[R', C] \cdot x[C] = b[R']$ has at most one solution, which can be computed in polynomial time. If $x[C]$ does not exist, or if $x[C]$ exists but has some negative entry, we set $R := R'$. Suppose that $x[C]$ does exist and is nonnegative. Due to the assumption on C, this $x[C]$ does not solve $A[C] \cdot x[C] = b$. Hence there is a row index i with $A[i, C] \cdot x[C] \neq b_i$, and we can trivially find such i. Finally set $R := R' \cup \{i\}$. Since already the solution to $A[R', C] \cdot x[C] = b[R']$ was unique, $A[R, C] \cdot x[C] = b[R]$ has no alternative solution either. $\diamond$

Now we get the first main result of this section.

Theorem 4 *For systems $Ax = b$ where all rows of A are r-sparse, we can enumerate all minimal feasible sets of size at most k in $O^*(r^k k!)$ time. In particular, this problem is in FPT, in the combined parameter r, k.*

Proof. Starting from a family with one member $C = \emptyset$ we generate a family of sets C of linearly independent columns. This is done as follows. We pick any C from this family. For $C \neq \emptyset$ we check in polynomial time, by linear programming, whether $A[C] \cdot x[C] = b$ has a nonnegative solution. If so, we remove C from the family and put it aside. (We know that C contains some feasible set, hence we also know that extending C by further columns cannot generate new minimal feasible sets.) If not, or if $C = \emptyset$, it is clear that C does not include any feasible subset. Then we find a small family E of columns, with the property that every feasible set containing C as a subset must also contain some of the columns from E. (This step will be detailed below.) Then we check linear independence of every such set $C \cup \{j\}$, $j \in E$, and we replace C in our family with all $C \cup \{j\}$ that pass this test.

Since we are only interested in minimal feasible sets of size at most k, we also throw away sets that exceed the size limit. It is easy to see that we cannot miss any solution: By Lemma 1, only linearly independent column sets need to be considered, and their subsets are linearly independent as well. We keep all column sets that are candidates for being extendible to a minimal feasible set.

In order to find a set E as specified above, we apply Lemma 3. With $c := |C|$, we determine a set R of at most $c + 1$ rows such that $A[R, C] \cdot x[C] = b[R]$ lacks a nonnegative solution. Since the rows of A are r-sparse, $A[R]$ has nonzeros in a set E of at most $(c + 1)r$ columns. We extend C with any one column from E, that is, we generate at most $(c+1)r$ new column sets. Note that at least some column of E must be inserted in C to make $A[R, C] \cdot x[C] = b[R]$ solvable, which is a necessary condition for feasibility.

On every path of the search tree generating our sets C, their cardinalities c grow from 0 to at most $k-1$. Since the outdegrees of search tree nodes are at most $(c+1)r$, the number of leaves of the search tree is bounded by $(r)(2r)(3r) \cdot \ldots \cdot (kr) = r^k k!$ This bounds the number of column sets C we have put aside, and all minimal feasible sets of size at most k are among them. It remains to check the minimality of every candidate C. Recall that C is linearly independent, hence it comes with a unique solution to $A[C] \cdot x[C] = b$. Thus C is minimal if and only if $x[C]$ has only positive entries. $\diamond$

So far we have silently assumed a model of computation with precise real numbers. To turn the algorithm into a practical method, note that a vector b being in the convex hull of

some set C of columns is, after a small perturbation, still close to a point in the convex hull. Instead of looking for an exact solution to $A[C] \cdot x[C] = b$ we append all unit vectors to the matrix and compute, still by a linear program, a solution that minimizes the coefficients of these extra vectors, and we accept solutions within some tolerance. Similarly we relax the equality tests by some tolerance. This way we can still recover minimal feasible sets after small perturbations of b. Solutions are changed only if, roughly speaking, the noise is comparable to the distance to the next candidate solutions.

The $k!$ term in the time bound is somewhat unsatisfactory. Before we give an alternative branching strategy avoiding that, we introduce the following notion.

Definition 5 *We define the hypergraph H* associated *with the system $Ax = b$ to be the hypergraph whose vertices and hyperedges are the columns and rows of A, respectively, and vertex j belongs to edge i iff $a_{ij} \neq 0$.*

Observe that every minimal feasible set of columns of A *contains* some minimal hitting set of H (but is not necessarily *equal* to some minimal hitting set of H). Therefore we may first enumerate the minimal hitting sets C of H, which can be trivially done with branching number r, and then start the procedure of Theorem 4 from these sets C rather than from the empty set. The associated hypergraph is not only useful in this heuristic. We also use it in the following theorem.

Theorem 6 *For systems $Ax = b$ where all rows of A are r-sparse, we can enumerate all minimal feasible sets of size at most k in $O^*(r^{2k})$ time.*

Proof. We generate a family of records, where every record consists of a set C of columns and a system of linear equations $Qx = s$, such that the rows of Q are linearly independent, and Q has nonzeros in $Q[C]$ only. Note that, consequently, Q has at most $|C|$ rows. We start from a family with one record where C and the row set of Q are empty; then the above condition is vacuously true. The family evolves as follows. We pick any record $(C, Qx = s)$. If $C \neq \emptyset$, we check in polynomial time, by linear programming, whether $(A[C] \cdot x[C] = b,\ Q[C] \cdot x[C] = s)$ has a nonnegative solution. If so, we remove C from the family and put it aside. (As before, we know that C contains some feasible set and needs no further extension.) If not, or if $C = \emptyset$, clearly C does not include any feasible subset that also satisfies the extra system $Qx = s$. Then we find a small family E of columns and a new linear equation, with the property that every feasible set of the compound system $(Ax = b,\ Qx = s)$ containing C as a subset must also contain some of the columns from E or must have a solution that fulfills the new equation. (This step will be detailed below.) Then we extend our record in all possible ways, that is, we either replace C with some $C \cup \{j\}$, $j \in E$, or we keep C but insert the new linear equation. Again we abandon records where C is not linearly independent or exceeds the size limit k, and again, the "exhaustive" branching ensures that we cannot miss any solution.

Specifically, in order to do the branching we fix some row i of A that has some nonzero outside C. (Such a row exists, as we can w.l.o.g. assume that no column of A is the zero vector, and we can trivially stop if C is already the full set of columns.) Let E be the set of columns $j \notin C$ where $a_{ij} > 0$ in this fixed row i. If we decide to append none of the $j \in E$ to C, the equation in row i must be fulfilled already by nonzero variables in C, formally

$A[i, C] \cdot x[C] = b_i$. We append this equation to the extra system $Qx = s$, provided that it is linearly independent of those already being in $Qx = s$. In the opposite case it follows $x_j = 0$ for all $j \in E$ (since the equation is already enforced, and everything is nonnegative). Then we fix another row i and repeat the procedure until we either find an i and E for branching, or all variables outside C are fixed to 0, and thus C is a dead end. Since the rows of A are r-sparse, obviously the branching number is at most $r + 1$, where the worst case is that the selected row i has all its r nonzeros outside C.

The minimality of every candidate C is checked as before. For the time analysis we use an auxiliary parameter that is initially $2k$. We deduct 1 from this parameter, for every column inserted in C and for every equation inserted in $Qx = s$. Since Q has at most $|C| \leq k$ rows, in fact we deduct never more than $2k$. We also remark that the extra equations have been introduced only for the sake of a simple analysis. Of course, it is equivalent to fix the variables x_j, $j \in E$ to 0 in the affected branches.

This gives only a time bound of $O^*((r + 1)^{2k})$, but we can easily improve the algorithm to achieve $O^*(r^{2k})$ as follows. Instead of starting from scratch, we first take the hypergraph associated with $Ax = b$ and enumerate all minimal hitting sets C of size at most k. Every feasible set must contain some of them, and the branching number is trivially r. For all these C, the extra system $Qx = s$ is still empty. After that we continue as above. Since now every row has at most $r - 1$ nonzeros outside C, the branching number is r. $\diamond$

Obviously $O^*(r^{2k})$ beats $O^*(r^k k!)$ when k grows, in relation to r. Still the former branching strategy could be faster for k close to r, as one can see from Stirling's formula. A direct comparison in theory is difficult, as the hidden polynomial factors depend on implementation details. Moreover, additional heuristics may be applied that sometimes allow cheaper branchings also in Theorem 4. We discuss some observation below.

We may look for pairs of rows i, i' where $b_i > b_{i'}$ but $a_{ij} \leq a_{i'j}$ for all $j \in C$. Then we add a column j with $a_{ij} > a_{i'j}$, clearly some of them must be put in C. Note that these are at most r columns. When all these conditions for pairs of rows are fulfilled, we similarly look for branchings based on triples of rows, etc. If we are lucky, we can grow our sets C by moderate branchings. Another improvement comes from the special case $r = 2$ which is interesting in itself.

Theorem 7 *For systems $Ax = b$ where all rows of A are 2-sparse, we get an implicit enumeration of all minimal feasible sets in polynomial time.*

Proof. Every 1-sparse row is a linear equation with only one variable, and its unique solution value is obtained instantly. Hence we can remove all 1-sparse rows from A, as well as all columns where these rows have their nonzero entries. That is, these columns must appear in every minimal feasible set. Now we have exactly two positive entries in each row.

We construct a graph with every column being a vertex, and every row being an edge that joins the vertices representing the columns of the two nonzeros. Solving $Ax = b$ is now equivalent to a graph problem: Given a graph where the edges e are labeled with real numbers b_e and the vertex-edge pairs (v, e), $v \in e$, are weighted with real numbers a_{ev}, the task is to label the vertices with real numbers $x_v \geq 0$, such that $a_{eu}x_u + a_{ev}x_v = b_e$ holds on every edge $e = uv$.

But this graph problem is rather simple, too, as we discuss now. First assume that the graph is connected. Then any label x_v determines, by propagation, the x_u labels of all vertices u. Hence it suffices to try every vertex v and set its label zero, and check whether the unique solution forced by that choice is nonnegative. This yields all minimal feasible sets, unless the test fails for every v. In the latter case, the entire vertex set is the only candidate feasible solution. If so, by Lemma 1, the columns are linearly independent, hence we can determine the unique solution by Gauss elimination and check for nonnegativity.

Finally, if the graph is not connected, the reasoning applies to every connected component independently, and all combinations of minimal feasible sets of the components are exactly the minimal feasible sets in the whole instance. In this way we can implicitly describe all solutions in polynomial time, although their number is, of course, not polynomial in general. $\diamond$

We remark that a closer look at the graph problem also reveals the structure of the solution space: In any connected component of the graph containing an odd cycle, the values of the corresponding variables are uniquely determined, and the solution space of any bipartite component is at most one-dimensional. This follows easily from the aforementioned propagation of values.

When matrix A is r-sparse for some $r > 2$, we can still begin and apply the method in Theorem 7 to the 2-sparse rows, temporarily ignoring the other rows, to find all possible solutions on the affected entries of x. Then we may branch on these solutions, remove the settled rows and columns, and apply the method iteratively to the remaining systems, as long as new 2-sparse rows are obtained. From the aforementioned structure of the solution space of 2-sparse systems one can derive that the branching number in this phase is only 2, or better. (The worst case is graphs consisting of isolated edges.) Since this is only an additional heuristic and no theoretical bounds are provided, we skip the details.

3 An Improved Branching Algorithm

The aim of this section is to further reduce the base in Theorem 6 by more sophisticated branching, but still without extensive case inspections. First we review a search tree algorithm that has been proposed in [17] for computing a minimum hitting set with at most k vertices in hypergraphs of any fixed rank r, and later adapted in [5] for counting and implicit enumeration of all these minimal hitting sets. Later we have to modify the algorithm somewhat, to meet the special needs of our problem to enumerate the minimal feasible sets. We will refer to the algorithm as HS-Order.

Some terminology first: The degree $deg(v)$ of a vertex v is the number of hyperedges v belongs to. We call vertices *equivalent* if they belong to exactly the same hyperedges. Equivalent vertices can be instantly merged into one *multiple vertex*, because they can replace each other in a hitting set. If no equivalent vertices exist, then $deg(u) \geq deg(v)$ implies the existence of some hyperedge that contains u but not v. This is trivial if $deg(u) > deg(v)$, and in case $deg(u) = deg(v)$ note that u and v are not in exactly the same hyperedges, hence the claim follows as well. A hyperedge with strictly fewer than r vertices (where any multiple vertex counts like a normal vertex) is called *small*, otherwise it is called *large*. Now we phrase

HS-Order as a sequence of rules, applied as long as possible in this priority order:
(1) If equivalent vertices exist, merge them accordingly into multiple vertices.
(2) If some small hyperedge H with at least two vertices exists, then order the vertices in H as $v_1, v_2, v_3, \ldots$ such that $deg(v_1)$ is the highest degree in H. In a branching step, decide on the smallest j such that v_j is added to the solution. (This creates one branch for every j.) Delete all vertices v_i, $i < j$, and remove all hyperedges containing v_j.
(3) If some large hyperedge H exists, proceed as in (2).
The search tree construction stops when only isolated multiple vertices, i.e., pairwise disjoint hyperedges, remain.

HS-Order was already analyzed in the aforementioned papers. The key observation leading to a branching number significantly better than r is that, in (2) and (3), all branches but one diminish another hyperedge, thus ensure the existence of some small hyperedge. This happens since, for every $j > 1$, in particular v_1 is deleted, and some hyperedge contains v_1 but not v_j. The time complexity is described by the coupled recurrences $T(k) = (r-1)B(k-1)+T(k-1)$ and $B(k) = (r-2)B(k-1)+T(k-1)$. Here, $T(k)$ is the leaf number of a search tree when the parameter value is k, and $B(k)$ is defined similarly, but under the additional assumption that some small hyperedge exists.

Lemma 8 ([17]) *The coupled recurrence given by $T(k) = (r-1)B(k-1)+T(k-1)$ and $B(k) = (r-2)B(k-1)+T(k-1)$ has the solution $T(k) = O((r-1+1/(r-1))^k)$.* $\diamond$

For reasons that we discuss later, we can guarantee a small hyperedge only in all branches with two exceptions, therefore we work with a weaker recurrence:

Lemma 9 *The coupled recurrence given by $T(k) = (r-2)B(k-1)+2T(k-1)$ and $B(k) = (r-3)B(k-1)+2T(k-1)$ has the solution $T(k) = O((r-1+2/(r-1))^k)$.*

Proof. With $T(k) = x^k$ and $B(k) = y^k$ we rewrite the recurrence as $x^k = (r-2)y^{k-1}+2x^{k-1}$ and $y^k = (r-3)y^{k-1}+2x^{k-1}$. Assuming that $x > r-3$, iterated substitution of the second equation into itself yields

$$y^k < (r-3)^k + 2(r-3)^{k-1}\left(\left(\frac{x}{r-3}\right)^{k-1} + \left(\frac{x}{r-3}\right)^{k-2} + \left(\frac{x}{r-3}\right)^{k-3} + \ldots\right)$$

$$< (r-3)^k + 2(r-3)^{k-1}\left(\frac{(x/(r-3))^k}{x/(r-3)-1}\right) = (r-3)^k + 2x^k/(x-r+3).$$

Since we are proving an upper bound, we can increase y such that $y^k = (r-3)^k+2x^k/(x-r+3)$. Since k is arbitrary, replace k with $k-1$ and get $y^{k-1} = (r-3)^{k-1}+2x^{k-1}/(x-r+3)$. Substitute this into $x^k = (r-2)y^{k-1}+2x^{k-1}$ and obtain

$$x^k = (r-2)(r-3)^{k-1} + 2(r-2)x^{k-1}/(x-r+3) + 2x^{k-1}.$$

Divide by x^{k-1} and omit the terms that go to 0 as k grows. It remains

$$x = 2(r-2)/(x-r+3) + 2,$$

hence

$$x^2 - (r-3)x = 2(r-2) + 2x - 2(r-3),$$

and finally $x^2 - (r-1)x - 2 = 0$, with the solution

$$x = (r-1)/2 + \sqrt{(r-1)^2/4 + 2} = r - 1 + \sqrt{(r-1)^2/4 + 2} - (r-1)/2$$

$$= r - 1 + \frac{(r-1)^2/4 + 2 - (r-1)^2/4}{\sqrt{(r-1)^2/4 + 2} + (r-1)/2} < r - 1 + 2/(r-1).$$

$\diamond$

Next we introduce an "interactive" (or "online") version of the hitting set problem that might appear artificial at first glance, but it will fit well in our actual algebraic problem. Suppose that we are asked to select vertices for the hitting set or discard vertices, one by one (as it actually happens in a branching algorithm). Hyperedges hit by a selected vertex are removed, and equivalent vertices are merged. Furthermore, not all hyperedges are known from the beginning, but new hyperedges may be revealed to us after every decision. These new hyperedges may even pop up depending on our choices. However, any new hyperedge must fulfill a crucial condition: It must *respect equivalence*, that is, not distinguish vertices that are equivalent. In other words, either all vertices forming a multiple vertex, or none of them, must belong to any new hyperedge. Similarly, some hyperedges may disappear after a decision, even if they are not hit by a selected vertex. Our task in this game is still to hit all hyperedges (that did not disappear spontaneously), using at most k vertices in total.

Algorithm HS-Order can still be applied. If no hyperedges disappear, it has the same complexity bound as in [17]. The reason is simple: The original complexity analysis is solely based on the branching numbers of the rules actually applied to selected hyperedges, and new hyperedges never split a multiple vertex. Thus it is immaterial whether there exist further, yet unknown hyperedges behind the scenes; they are treated at a later time anyway. The issue of disappearing hyperedges is more tricky. The complexity bound from [17]relies on the fact that the existence of a small hyperedge is guaranteed in all branches but one, when the branching rule is applied. In our "online" version it may happen in the worst case that all small hyperedges disappear, and then we only get the trivial branching number r. We will have to control the existence of small hyperedges when we apply HS-Order to feasible set enumeration.

After these preparations we now devise our improved algorithm.

Theorem 10 *For systems $Ax = b$ where all rows of A are r-sparse, we can compute a succinct enumeration of all minimal feasible sets of size at most k in $O^*((r - 1 + 2/(r-1))^{2k})$ time.*

Proof. As earlier, we start our search tree construction with the associated hypergraph. With a slight abuse of notation, the terms vertex and column and variable, as well as hyperedge and row and equation, are used interchangeably without risk of confusion.

We apply HS-Order to generate subsets C of potential solutions, each accompanied by an extra system of linear equations $Qx = b$. At the same time we also modify the residual hypergraph according to the following rules.

When a hyperedge H is hit for the first time, that is, some vertex v of H is put in C, we add a symbolic vertex u, called an annulator, to the rest of H. (Its role will become clear below.) Formally, we replace H with a new hyperedge $(H \setminus \{v\}) \cup \{u\}$. Similarly, for any

further vertex v of H that respresents a column and is put in C, we replace H with $H \setminus \{v\}$. Only when the annulator of a hyperedge H is put in C, we remove H altogether. Moreover, in this case we also delete all remaining vertices in H.

Note that the new hyperedges respect equivalence: Since every annulator is a new vertex, and the rest of the affected hyperedge is merely copied, we never distinguish vertices that are rendered equivalent.

We also do a few more things after every branching, that are discussed now.
(i) We check every new C whether it is minimal feasible, and if so, we abort the corresponding path of the search tree. (Note that this cannot raise the time bound.)
(ii) We update the system $Qx = s$: Whenever the last decision was to take the annulator of some hyperedge, say, of row i, we set all variables $x_j := 0$, where $j \notin C$ and $a_{ij} > 0$. Thus we stipulate the new equation $A[i, C] \cdot x[C] = b_i$.
(iii) We delete all rows i where the compound vector $(A[i, C], b_i)$ depends linearly on the rows of the current system $Qx = s$. (In particular, this situation implies that $A[i, C] \cdot x[C] = b_i$ is already enforced by $Qx = s$). Again we set $x_j := 0$ for all j with $j \notin C$ and $a_{ij} > 0$ in this case.

Deletion of the rows specified in (iii) ensures that the equations of the extra system $Qx = b$ always remain linearly independent when the system is extended in step (ii) by a new equation. Trivially they remain linearly independent also when C is extended by new columns.

Now we argue that certain branches in rule (2) and (3) of HS-Order leave us with some small hyperedge. First observe that, if H has already an annulator, we never declare it v_1: The annulator has degree 1. If some "regular" vertex of H has a larger degree, then some of them becomes v_1, and if all vertices of H have degree 1, they have been already merged according to rule (1) of HS-Order, such that rule (2) or (3) is not further applied to H. Hence, in all branches $j > 2$, vertex v_1 is deleted from some hyperedge H_j containing v_1 but not v_j, therefore H_j becomes small. (Note that it is not possible that H_j obtains an annulator at this moment, since an annulator is caused only by selecting some vertex of H_j for C for the first time, but not by vertex deletion.)

Now step (iii) might cause, in the worst case, the loss of H_j, even the loss of all small hyperedges, after the branching step. However, a new vector $(A[i, C], b_i)$ can become linearly dependent of $Qx = s$ only when a new equation has been appended to $Qx = s$, and this can happen only if the annulator of H has been chosen, see (ii). Hence we obtain a small hyperedge in all branches but at most two, and the recursion in Lemma 9 applies.

Our hitting set here has size limit $2k$ rather than k, for essentially the same reason as before: We use the initial parameter value $2k$ and deduct 1 for every vertex put in the hitting set, that is, for every column inserted in C and for every annulator chosen in a branching step. Due to (ii), every choice of an annulator in a branching step inserts an equation in $Qx = s$, and due to (iii), every new row entering this system is linearly independent of the others. Again, since $|C| \leq k$, and Q exclusively contains variables from C with nonzero coefficients, it follows that Q has at most k rows, hence we deduct at most $2k$ from the parameter. This completes the analysis.

Every leaf of the search tree generated so far represents an instance where the vertices outside the current C form disjoint multiple vertices. That is, the variables outside C are

divided in pairwise disjoint groups, such that the variables of each group appear (with nonzero coefficients) in only one equation, say, in row i. We represent the multiple vertex of row i by one variable y_i which equals the linear combination of the original variables x_j with the given coefficients $a_{ij} > 0$. In a last branching phase we decide for every such i whether we take the annulator of row i, hence $y_i := 0$, or put the variable y_i in C. Trivially the branching number is only 2. Note that the measures in (i)–(iii) are applied also in this phase, such that still everything happens within the $2k$ bound. $\diamond$

We suspect that our analysis is still too conservative. Recall that, when a hyperedge disappears due to the choice of its annulator, we also delete its vertices, which should in general leave a number of small hyperedges, but our analysis does not make use of that. It is hard to see when the worst case (all small hyperedges disappear) really occurs, although one gets the feeling that it should be rare. A more ambitious goal is an algorithm with complexity $O^*((cr)^k)$ for some constant c. It is not even clear whether feasible set enumeration is really harder than hitting set enumeration, since the quantitative constraints might be helpful, rather than adding complexity. We have only argued that algorithms for hitting set enumeration cannot be straightforwardly extended to feasible set enumeration, but this does not exclude totally different approaches.

4 A Problem Kernel for Binary Matrices

In [4] we introduced the notion of a *full kernel* of an FPT enumeration problem, which is a set that contains all minimal solutions. For the minimal hitting sets of size at most k in hypergraphs with hyperedges of size r there is an $(r-1)k^r + k$ size full kernel, and there exist hypergraphs with full kernel size $\Omega(k^r)$ [4]. In order to establish similar bounds for the present problem we have to adapt the proofs in [4], i.e., the next theorem is not just a consequence of the old result. The following result holds when all entries in A are 0 or 1.

Theorem 11 *For systems $Ax = b$ with 0-1-matrices A where all rows are r-sparse, all minimal feasible sets of at most k columns are contained in a set of $(r-1)k^r + k$ columns. The bound $O(k^r)$ is tight, up to some factor depending on r but not on k.*

Proof. First we need some notation: The *hyperedges* are those of the associated hypergraph defined above. With respect to a solution vector x, we call a vertex *positive* if the corresponding variable x_j is positive. The sum $s(C)$ of a set C of vertices is defined by $s(C) := \sum_{j \in C} x_j$.

As an inductive hypothesis we suppose that every set C of $r - i$ vertices (i.e., columns) is contained in at most k^i hyperedges, or there is no feasible set at all. To establish the induction base $i = 0$, note that a set of r vertices can be in only one hyperedge, due to r-sparsity. (Otherwise the system $Ax = b$ has identical rows, and all copies but one can be deleted.)

For the induction step, with a fixed i, assume that some set C of $r - i$ vertices belongs to $k^i + 1$ or more hyperedges. By the inductive hypothesis, every set $C \cup \{v\}$ is in at most k^{i-1} hyperedges. It follows that k vertices outside C would not be enough to hit all $R \setminus C$, for the hyperedges $R \supset C$. In a k-sparse nonnegative solution we cannot have $s(C) < s(R)$

for every hyperedge $R \supset C$, since then we need a positive vertex in every $R \setminus C$, but these would be more than k positive vertices, due to the previous observation. Furthermore, in a nonnegative solution, $s(C) > s(R)$ is not possible either, for any $R \supset C$. Thus, in every k-sparse nonnegative solution, $s(C)$ must be equal to the (given) smallest $s(R)$, $R \supset C$. This establishes a new equation for a new hyperedge C, whereas all hyperedges $R \supset C$ can be replaced with $R \setminus C$, where the required sums are adjusted in the obvious way. In summary, if some set C of $r - i$ vertices belongs to $k^i + 1$ or more hyperedges, we get a system with the same k-sparse nonnegative solutions, where C is no longer a subset of other hyperedges, and the total size of all hyperedges, i.e., the number of 1 entries in A, has strictly decreased. Hence this transformation can be done only finitely many times, and eventually the inductive hypothesis holds for i, in the transformed system.

For $i = r - 1$ the inductive hypothesis says that every vertex is contained in at most k^{r-1} hyperedges. Since every feasible solution is also a hitting set, and at most k positive vertices are permitted, at most k^r hyperedges remain, or no feasible set can exist. Now the size bound follows obviously.

To prove tightness, we construct a linear system with $\Omega(k^r)$ columns, for any fixed r, such that every column appears in some minimal feasible set. It suffices to consider k divisible by r. We take a complete d-ary tree of depth r where all non-leaf vertices have outdegree $d := k/r$. Every vertex in the tree, except the root, becomes a variable. For every leaf v, let $P(v)$ denote the path from the root to v, let $S'(v)$ be the set of all siblings of vertices on $P(v)$, and $S(v) = S'(v) \cup \{v\}$. Note that $S(v)$ does not include the root. For every leaf v we set up an r-sparse equation containing all variables on $P(v)$ with coefficients 1; all other coefficients are 0. The right-hand side is 1. Observe that $S(v)$ contains exactly one vertex of $P(u)$, for every leaf u. It follows that every set $S(v)$ (more precisely, the set of corresponding columns) is a feasible set, in particular, it gives rise to a 0-1 solution vector, where exactly the variables in $S(v)$ have value 1. Every $S(v)$ is also minimal feasible, because $S(v) \setminus \{w\}$ for any vertex $w \in S(v)$ is disjoint to some path $P(v)$, hence some equation cannot be satisfied. Every $S(v)$ has size $rd = k$, and the tree has $d^r = (1/r^r)k^r$ leaves, hence $\Omega(k^r)$ vertices. Obviously, every vertex is also member of some $S(v)$. $\diamond$

Note that the upper-bound proof also describes an efficient method to obtain a full kernel of the claimed size, and that only subsets C of the given hyperedges need to be examined; these are at most 2^r per hyperedge. But the proof does not apply to matrices A with arbitrary positive coefficients, because then the step where new equations on smaller hyperedges are enforced does not work.

We also remark that we improved in [5] the full kernel size bound for hitting set enumeration to $(1 + o(1))k^r$, using more sophisticated counting arguments. A natural question is whether these techniques can be applied also to r-sparse linear systems. For the problem of finding just some hitting set of size k in a hypergraph of rank r, no kernel with $k^{r-\epsilon}$ hyperedges can exist, under common complexity-theoretic assumptions [6]. One could ask whether a similar lower-bound result holds for finding some sparse solution of a linear system.

5 Sparse Nonnegative Solutions of Systems of Linear Inequalities via the Complex Model of Group Testing

Our FPT result for systems of linear equations do not immediately generalize to inequalities. In this section we give a completely different approach for systems of linear inequalities. Recall the notion of a minimal feasible column set, and the basic fact that, given the support of x, some solution x can be computed by linear programming. Hence we have a test that tells us, for any given set C of columns, whether C contains some minimal feasible set. The goal is to find all minimal feasible sets. This is a problem known elsewhere, as the *complex model of group testing* or *searching for defective edges*: In a set, an unknown family of subsets are *defective*, and a *group test* (for brevity: *test*) on a subset C answers positively if C contains (entirely) some defective set, otherwise it answers negatively. Accordingly, a tested set C is also called a positive or negative *pool*. The term "defective" is common in group testing and comes from applications in, e.g., fault diagnosis. Instead of *defective set* we speak of a *complex* in this section.

It follows from [3] that all complexes can be found using $kd \log_2 n + k^{k/2} d^k + o(d^k)$ tests, where n is the number of elements, k the maximum size of a complex, and d the number of complexes. No previous knowledge of d is assumed. However, it is assumed that all complexes have at most a prescribed number k of elements. In our application the problem is slightly more general: For a given k we wish to enumerate all complexes of size at most k, but there may exist larger complexes as well, and clearly they can affect the test results. Therefore the algorithm from [3] does not carry over to our problem, moreover, the algorithm and its analysis are intricate. Below we give an algorithm that has a somewhat worse dependency on parameters k and d, but it also works in our case and is conceptually simpler. We start with an adaptation of the Triesch-Johann procedure [18, 12] used in [3]. In the following, a *k-complex* is either a complex with at most k elements, or a k-element subset of a larger complex.

Lemma 12 *Some k-complex can be found by $k \log_2 n$ tests.*

Proof. First consider the problem of finding a complex, rather than a k-complex. We index the elements arbitrarily by $v_1, \ldots, v_n$. By binary search using $\log_2 n$ tests we determine the largest j such that $\{v_j, \ldots, v_n\}$ is a positive pool. Note that $j = n$ if $\{v_n\}$ is still a positive pool, and then this is also a complex. In the following consider $j < n$. Clearly $\{v_j, \ldots, v_n\}$ contains a complex while $\{v_{j+1}, \ldots, v_n\}$ does not. Hence all complexes in $\{v_j, \ldots, v_n\}$ necessarily contain v_j, and such a complex does exist. In order to find a complex of this form, we fix v_j, that is, we henceforth add v_j to every pool, and we search for a complex in $\{v_{j+1}, \ldots, v_n\}$ relative to that. (In other words, we search for a positive pool such that removal of any element v_l, $l > j$, yields a negative pool. We know already that removal of v_j yields a negative pool.)

Iterating this reasoning, we successively fix elements, each by at most $\log_2 n$ tests, that together form a subset of a complex. After each round we test the current subset C. If C is a negative pool, we continue the search on the suffix after the last inserted element. If C is a positive pool, we have finished a complex.

When searching for a k-complex we proceed in the same way, we just stop after k rounds if the complex is not yet completed. $\diamond$

Now we are using this routine in our enumeration algorithm.

Theorem 13 *Given an integer k we can determine all complexes of size at most k using $kd \log_2 n + \min(k^k d^{k+1}, dk^d)$ tests, if d complexes exist. No previous knowledge of d is assumed.*

Proof. Let V denote the set of all elements. We maintain a family $\mathcal{C}$ of k-complexes and their union U. Initially, $\mathcal{C}$ and U are empty.

In each round of the algorithm we probe all sets S that have the following properties: (1) $S \subseteq U$, (2) $|S| \leq k$, (3) S contains none of the $C \in \mathcal{C}$ as a subset. To probe S means to test $S \cup (V \setminus U)$. If this pool is positive, it contains a complex which is not already in $\mathcal{C}$ nor extends any k-complex from $\mathcal{C}$, due to (3). Thus, by applying Lemma 12 we find either another complex (of size at most k) or another k-complex C'. In the latter case we have $C' \subset C''$ for some complex C'' not already "covered" by $\mathcal{C}$, as said before. This fact is important, as it implies that every round deals with some new complex, hence the routine of Lemma 12 is called at most d times. If all probes give a negative answer, then every complex C of size at most k appears already in $\mathcal{C}$: If not, then $S := C \cap U$ fulfills (1)–(3), hence S would be probed and answer positively, a contradiction.

All calls of Lemma 12 need at most $kd \log_2 n$ tests. Since $|C| \leq k$ for all $C \in \mathcal{C}$, we always have $|U| \leq kd$. Due to (1) and (2) we probe fewer than $(kd)^k$ sets S in every round, furthermore each S is probed at most d times. This yields the first bound $kd \log_2 n + k^k d^{k+1}$.

An extension of this strategy gives the second part of the bound. In case $d \leq k$ we can bound the number of probes in a round by k^d, which is smaller than $(kd)^k$: Property (3) requires that some element from each $C \in \mathcal{C}$ be excluded from S. In the worst case we have k^d different choices, and then we take S as U minus the excluded elements. Now S can be larger than k, however, property (2) above was only used to bound the number of probes.

Thus, we finally proceed as follows: In the first k rounds we apply the probing strategy that excludes a hitting set of $\mathcal{C}$, and then we switch to the former strategy that guesses the intersection of a new complex with U. $\diamond$

Corollary 14 *For systems of linear inequalities with d minimal feasible sets, the problem of enumerating all minimal feasible sets of size at most k is in FPT, in the combined parameter d, k.* $\diamond$

A concern is that the parameter d could be too large to be practical. But the famous results about unique sparsest solutions for some natural classes of random matrices [1, 7, 14, 19] give hope that one would actually encounter small d in real data. This question needs experimental research.

To summarize, we used complex group testing where the tests correspond to linear programs. Other than that, our current algorithm does not further use the mere fact that we are working on linear systems. Whereas this modularity of our algorithm might be appealing, some clever use of polyhedral combinatorics instead of group testing might lead to more efficient algorithms.

6 Conclusions

It would be interesting to prove even better FPT time bounds and a kernel size bound for non-binary matrices, and to extend the polynomial result for 2-sparse equations. Specific research questions were already discussed in the technical sections. One could also think of other meaningful problem versions that bridge between strict equations, arbitrary inequalities, and the Boolean case (hitting set problem). Furthermore, we have not considered the question of output-sensitive algorithms for sparse solution of linear systems. Finally, implementation of the methods, taking the numerical issues into account, and experiments on protein mixture data would give important insights.

Acknowledgments

This work has been supported by the Swedish Research Council (Vetenskapsrådet), grant no. 2010-4661 "Generalized and fast search strategies for parameterized problems", and the questions were inspired by discussions with Leonid Molokov (PhD student in Computational Systems Biology). The author would like to thank the referees for careful reading.

References

[1] A.M. Bruckstein, M. Elad, M. Zibulevsky, On the uniqueness of nonnegative sparse solutions to underdetermined systems of equations, IEEE Trans. on Info. Theory 54 (2008) 4813–4820.

[2] E.J. Candés, M.B. Wakin, An introduction to compressive sampling, IEEE Signal Proc. Magazine, March 2008, 21–30.

[3] T. Chen, F.K. Hwang, A competitive algorithm in searching for many edges in a hypergraph, Discr. Appl. Math. 155 (2007) 566–571.

[4] P. Damaschke, Parameterized enumeration, transversals, and imperfect phylogeny reconstruction, Theor. Comput. Sci. 351 (2006) 337–350.

[5] P. Damaschke, L. Molokov, The union of minimal hitting sets: parameterized combinatorial bounds and counting, J. Discr. Algor. 7 (2009) 391–401.

[6] H. Dell, D. van Melkebeek, Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses, in: L.J. Schulman (Ed.), ACM STOC 2010, pp. 251–260.

[7] D.L. Donoho, J., Tanner, Sparse nonnegative solution of underdetermined linear equations by linear programming, Proc. Nat. Acad. of Sciences 102 (2005) 9446–9451.

[8] B. Dost, V. Bafna, N. Bandeira, X. Li, Z. Shen, S. Briggs, Shared peptides in mass spectrometry based protein quantification, in: S. Bazoglou (Ed.) RECOMB 2009, LNCS, vol. 5541, Springer, Heidelberg, 2009, pp. 356–371.

[9] H. Fernau, Parameterized algorithms for d-hitting set: the weighted case, Theor. Comput. Sci. 411 (2010) 1698–1713.

[10] H. Fernau, A top-down approach to search-trees: improved algorithmics for 3-hitting set, Algorithmica 57 (2010) 97–118.

[11] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, Freeman and Company, New York, 1979.

[12] P. Johann, A group testing problem for graphs with several defective edges, Discr. Appl. Math. 117 (2002) 99–108.

[13] V. Lacroix, M. Sammeth, R. Guigó, A. Bergeron, Exact transcriptome reconstruction from short sequence reads, in: K.A. Crandall, J. Lagergren (Eds.) WABI 2008, LNCS, vol. 5251, Springer, Heidelberg, 2008, pp. 50–63.

[14] M.J. Lai, On sparse solutions of underdetermined linear systems, J. Concr. Applic. Math. 8 (2010) 296–327.

[15] B.K. Natarajan, Sparse approximate solutions to linear systems, SIAM J. Comput. 24 (1995) 227–234.

[16] A.I. Nesvizhskii, R. Aebersold, Interpretation of shotgun proteomic data: the protein inference problem, Mol. Cellular Proteomics 4 (2005) 1419–1440.

[17] R. Niedermeier, P. Rossmanith, An efficient fixed-parameter algorithm for 3-hitting set. J. Discr. Algor. 1 (2003) 89–102.

[18] E. Triesch, A group testing problem for hypergraphs of bounded rank, Discr. Appl. Math. 66 (1996) 185–188.

[19] M. Wang, W. Xu, A. Tang, A unique "nonnegative" solution to an underdetermined system: from vectors to matrices. IEEE Trans. Signal Proc. 59 (2011), 1007–1016.