Resource Bounded Frequency Computations with Three Errors

Ulrich Hertrampf *Christoph Minnameier*

Institut für Formale Methoden der Informatik
Universität Stuttgart
Universitätsstr. 38
D-70569 Stuttgart, Germany
email: `ulrich.hertrampf@informatik.uni-stuttgart.de`

Abstract

We deal with frequency computations in polynomial time, or more general with resource bounded frequency computations. We investigate the first non-trivial case of the Hinrichs-Wechsung conjecture, which states that as soon as we have at least $2^d + d$ inputs to be queried, it does not become harder to get an answer with at most d errors, if we increase the number of inputs to be queried. This conjecture can easily be seen to hold for cases $d < 3$, and it seems very hard to prove in general. We solve the problem affirmatively in the case $d = 3$.

1 Introduction

The concept of frequency computation goes back to G. F. Rose [Ros60]. The idea was as follows: If a function f is not computable in the usual sense it may still be possible to compute it in the following relaxed way: On n (pairwise different) inputs $x_1, \ldots, x_n$, a sequence of outputs $y_1, \ldots, y_n$ shall be produced, which approximates the function in such a way that at least m of the output values are correct, i.e. $\|\{i \in \{1, \ldots, n\} \mid y_i = f(x_i)\}\| \geq m$. The class of functions computable in this way is denoted by (m, n). The same idea can also be applied to sets instead of functions.

Essentially there are three versions of the frequency computation model: The recursion theoretic one, the resource bounded one, and the finite state model. There has been done extensive research in order to solve the relationships between different classes of the form (m, n) in all three areas.

For the recursion theoretic setting, see e.g. [Deg81, KS95a, McN61, Tra63, BGK96]. The resource bounded case was investigated in [KS95a, HW97], and for the finite state model setting, see [Kin76, ADHP00, ADH03]. A lot of related work can be found in the literature (e.g. [HKO92, Kin72, Kin74, Kum92, KS95b, McN95, Tan02]).

As in many other areas it turned out that the recursion theoretic world much resembles the finite state machine world, whereas the resource bounded (for convenience we will in this paper always speak of polynomially time bounded) world looks completely different.

In all models it is clear that $(m + 1, n + 1) \subseteq (m, n)$, because given an $(m + 1, n + 1)$-algorithm one can obtain an (m, n)-algorithm as follows: On input $x_1, \ldots, x_n$ choose any element not equal to any of the x_i and call it x_{n+1}. Now query $x_1, \ldots, x_{n+1}$ to the given $(m + 1, n + 1)$-algorithm and ignore its $(n + 1)$-th output. Of the remaining n outputs no more than $(n + 1) - (m + 1) = n - m$ can be erroneous.

In [HW97] it was shown that in the resource bounded case for all $m < 2^d$ (where $d = n - m$), the class $(m + 1, n + 1)$ is a proper subset of (m, n), thus especially for polynomial time $(m + 1, n + 1)\mathrm{P} \subsetneq (m, n)\mathrm{P}$. Furthermore Hinrichs and Wechsung conjectured that to the contrary, one always obtains equality for $m \geq 2^d$. In this paper we will call that conjecture *the Hinrichs-Wechsung conjecture.*

As a hint on the general computation power of frequency computations we should remark here, that even in the finite state machine setting the classes (m,n) with $2m \leq n$ contain non-countably many languages, i.e. especially they contain non-r.e. sets.

For $d < 3$ the Hinrichs-Wechsung conjecture can easily be proven:

Proposition 1 *Let $m > 2$. Then $(m, m+1)\mathrm{P} = (2,3)\mathrm{P}$.*

Look at four inputs: There are four possibilities to query three of them; if we do that we get at least two correct answers. Thus, altogether we have 12 answers, three for each input, and at most 4 wrong ones. On each of the inputs we have either the same answer from all queries, or we have a split decision, which means either two correct answers and one erroneous, or the other way round. However, if on all inputs we have split decision then we know that if in one of the cases the majority was wrong, then altogether we had at least five wrong answers, which is impossible. So, in this case we can give even 4 correct answers.

If there are either 2 or 3 split decisions, then the non-split decisions must definitely be correct. So at least we can give one definitely correct answer. This leads to a solution by looking at the answer of the query on the remaining inputs.

If there are 0 or 1 split decisions, then we know that at least three of the majority votes would be correct, because otherwise we had at least 5 wrong answers in the four queries.

Thus, we saw $(2,3)\mathrm{P} = (3,4)\mathrm{P}$, and the argument can be generalized to the cases $(m, m+1)\mathrm{P} = (m+1, m+2)\mathrm{P}$ for $m > 2$.

Proposition 2 *Let $m > 4$. Then $(m, m+2)\mathrm{P} = (4,6)\mathrm{P}$.*

This can be proven with a similar reasoning as above, or it can be deduced from [KS95a].

However, for $d = 3$, the claim would be: Let $m > 8$. Then $(m, m+3)\mathrm{P} = (8,11)\mathrm{P}$. No similar proof for this case is known. To provide a proof is the subject of this paper.

2 The problem and some preliminaries

Definition 1 *The class $(m,n)\mathrm{P}$ consists of all languages L, such that a polynomial time algorithm exists, which on every set of n different inputs produces an n-bit vector, one output bit for each of the n inputs, such that these outputs coincide with the characteristic function of L on at least m of the inputs.*

The problem we want to address is the following: Let $m \geq 9$. Let L be a language from $(m, m+3)\mathrm{P}$, witnessed by the algorithm A. Assume we have a set of $m+4$ different inputs $x_1, \ldots, x_{m+4}$, and assume we have obtained $m+4$ results by querying every combination of $m+3$ of these inputs to A. We want to show that, no matter what these results look like, we can find an $(m+4)$-bit vector, which also makes at most 3 errors with respect to the characteristic function of L. Since our result will be obtained from the $m+4$ result vectors of the queries without further access to the inputs themselves, we can view the whole procedure as a polynomial time algorithm itself, thus proving that $(m, m+3)\mathrm{P} \subseteq (m+1, m+4)\mathrm{P}$. In other words, this will prove the Hinrichs-Wechsung conjecture for the case $d = 3$.

Remark 1: In fact this only proves $(9,12)\mathrm{P} = (10,13)\mathrm{P} = \ldots$ (due to the condition $m \geq 9$ above), whereas the conjecture starts with $(8,11)\mathrm{P} = (9,12)\mathrm{P}$. However, starting with the class $(9,12)\mathrm{P}$ is necessary to obtain the extendability of the investigation to higher values (c.f. the discussion in the proof). But, the case $(8,11)\mathrm{P} = (9,12)\mathrm{P}$ can be obtained by a simple adaptation of our proof, so we will state this case as a corollary.

Remark 2: Of course, it is well known that for any fixed value of m, the question whether $(m, m+3)\mathrm{P} = (m+1, m+4)\mathrm{P}$ can be considered to be a finite problem (by the characterization of Kummer and Stephan [KS95a]). But, first of all the instance size already for $m = 9$ is rather big, and it is not at all clear how to investigate this case systematically. Moreover, as soon as we speak of variable m, we would have to consider infinitely many cases, and this considerably increases the difficulty in finding a solution.

The objects we deal with will be 13×13-matrices with entries in $\{0, 1, *\}$, asterisks being exactly in those positions, where row index plus column index equals 14, i.e. in the diagonal from bottom left to top right. All other entries will be 0 or 1. The rows of these matrices shall represent the results of the queries on 12 of the 13 inputs, in the i-th column the resulting bit for input x_i. The asterisk indicates that we have no answer for that input. This means, the first row represents the result on $x_1, \ldots, x_{12}$, the second row the results on $x_1, \ldots, x_{11}, x_{13}$, and so on.

Our proof is structured as follows: We present an algorithm, which checks that for all matrices of the described type, one of the following two facts will be true:

1. The matrix is not consistent, meaning: There is no possibility to obtain the answers from the matrix for any true characteristic sequence without making more than three errors in one row.

   Take as an example a matrix starting with the rows (0,0,0,0,0,0,0,0,0,0,0,0,$*$) and (1,1,1,1,1,1,1,0,0,0,0,$*$,0). No matter what the real characteristic sequence should be, one of the two rows has to make at least 4 errors on the first 7 inputs, because the rows differ on all seven.

   Of course, these matrices will never appear in a computation of the kind we are interested in, as long as the underlying algorithm is a correct $(9,12)$P-algorithm for the language L to be decided.

2. We will be able to produce a 13-bit output, which coincides with the real characteristic vector on the 13 inputs in at least 10 components. Such an output will in the sequel always be called a *solution*.

In fact we will not try to enumerate all such matrices and investigate them – this would last by far too long. Instead, we consider partial matrices consisting of a few rows (always less than 13) and check, whether we can already show that one of the cases given will be true without knowing the other rows of the matrix. Because of this, the second possibility (proving the ability to produce a solution) will be split in two cases: Either we can explicitly give the solution already, or we can prove for a given i, that x_i is definitely in (or definitely not in, resp.) the language L. Then, using at most one more row of the given matrix (namely row $14 - i$), we can obtain a solution.

Moreover our case inspection will take advantage of occuring symmetries: Rearranging the input vector by application of a permutation results in (simultaneous) permutations of rows and columns, without changing the properties investigated: An inconsistent matrix will still be inconsistent after such a permutation, and a solution will be transformed to a solution of the new matrix, if we apply the same permutation to its bits. The second symmetry operation will be what we call bit-flipping: If we change all bits of a given column of our matrix (thus mapping 0 to 1 and 1 to 0, but leaving an asterisk unchanged), we also can see that the properties of inconsistency or having a solution remain the same (only in a solution, the according bit has to be flipped too). We will give details in Section 4.

Finally, we will show (inductively) that, whenever we know that $(9 + k, 12 + k)\mathrm{P} = (10 + k, 13 + k)\mathrm{P}$, then the same proof can be used to show that also $(10 + k, 13 + k)\mathrm{P} = (11 + k, 14 + k)\mathrm{P}$, which will complete the proof of our general result.

3 The maxdist technique

Dealing with the Hinrichs-Wechsung conjecture for quite some time, we found that a combination of theoretical arguments and machine power has to be used to make considerable progress in the direction of an affirmative solution. This means, we have to find a method of dividing the problem in subproblems, which may be easy enough to solve. Our approach does that by partitioning the set of all matrices to be considered according to a parameter called *maxdist*, the maximum distance between any two rows of the matrix. More formally:

Definition 2 *The* distance *of two (equal-sized) vectors over* $\{0, 1, *\}$ *is the number of places, where one vector carries value 0, and the other one carries value 1.*

For example, the two rows considered in the example of Section 2 have distance 7.

Remark 3: The distance between two rows of our matrices will not be changed, if a permutation as described above is applied. The same holds for bit-flipping, because we always flip all bits of a given column.

Definition 3 *The value* maxdist *for a* 13×13*-matrix of the kind considered is the maximum distance that appears between any two rows of the matrix.*

Lemma 1 *If a* 13×13*-matrix of the considered kind has a maxdist value of* 0 *or a maxdist value greater than* 4*, it always satisfies one of our desired facts (being not consistent or allowing a solution).*

Proof: We first look at maxdist at least 7. Then there are two rows in our matrix, which on 7 indices i give different answers to the question "$x_i \in L$?". As in a consistent matrix both should err on at most 3 inputs, this is a contradiction. Thus the matrix has to be inconsistent.

Now, look at maxdist either 5 or 6. We pick two rows with distance maxdist, say the i-th and the j-th row. Perform a permutation that maps i to 1 and j to 2, then the first two rows have distance 5 or 6. Now, the first row is $(a_1, \dots, a_{12}, *)$, the second is $(b_1, \dots, b_{11}, *, b_{13})$, and $a_i = b_i$ for $i \in \{7, \dots, 11\}$, while $a_i \neq b_i$ for $i \in \{1, \dots, 5\}$. Then, on the first 5 inputs, both rows together make exactly 5 errors with respect to the real characteristic sequence on $x_1, \dots, x_5$. It follows that all of $a_7, \dots, a_{11}$ coincide with the characteristic sequence of $x_7, \dots, x_{11}$, because otherwise the two given rows would make at least 7 errors in the sum, contradicting the assumption that each makes at most 3 errors. So we can choose for example row 3, where all but x_{11} are queried, to obtain answers for the other 12 inputs, of which at most 3 are erroneous. With the answer $a_{11} = b_{11}$ for x_{11} we have the desired solution.

Finally, let the matrix have maxdist value 0. Then all rows giving answer for x_i agree on that input (otherwise their distance were greater than 0). We call the according answer a_i. If more than three of the a_is would be wrong, we could choose a row that answers at least four of them. Then, this row would make 4 errors, in contradiction to our assumptions. Thus the vector $a_1, \dots, a_{13}$ is a solution.

$\square$

The case of maxdist 1 is solved in the same way as the cases 2, 3, and 4. However, the number of cases to be considered is rather small. Thus, it is a good opportunity to get some insight, how our computer program has to work. That is why we perform this case now:

Lemma 2 *If a* 13×13*-matrix of the considered kind has a maxdist value of* 1*, then it always satisfies one of our desired facts (being not consistent or allowing a solution).*

Proof: First observe that our matrix has the following property: all pairs of rows have either distance 0 or distance 1.

By a suitable rearrangement (applying a permutation), we make sure that the first two rows have distance 1, and that the difference occurs in the first component. (It is an easy exercise to show that a suitable permutation always exists.)

Now, by bit-flipping we change the matrix in such a way that the first two rows get the following form:

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)

We call this partial matrix 1-2-1, which means: maxdist is 1, depth (number of rows) is 2, and it is the first (in this case the only) matrix with these values of maxdist and depth, which we have to consider. For the third row we get the following possible values:

a) (0,0,0,0,0,0,0,0,0,0,∗,0,0)
b) (1,0,0,0,0,0,0,0,0,0,∗,0,0)
c) (0,0,0,0,0,0,0,0,0,0,∗,1,0)
d) (1,0,0,0,0,0,0,0,0,0,∗,0,1)

All other possible rows would contradict our assumption of maxdist being 1. Now, one can check that case b) can be transformed to case a) by permuting the first two rows (and thus also the last two columns) and bit-flipping in the first column.

The same holds for case d) to case c). So we only need to consider two cases of partial matrices with 3 rows.

Thus we now have two matrices to consider on depth 3, namely

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,0,0)

(denoted as matrix 1-3-1), and

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,1,0)

(denoted as matrix 1-3-2).

We proceed with matrix 1-3-1 and find the following three cases of possible next rows:

a) (0,0,0,0,0,0,0,0,0,∗,0,0,0)
b) (1,0,0,0,0,0,0,0,0,∗,0,0,0)
c) (0,0,0,0,0,0,0,0,0,∗,0,1,0)

All three cases are different and will lead to matrices 1-4-1, 1-4-2, and 1-4-3. But, from matrix 1-3-2 we only have two possible next rows:

a) (0,0,0,0,0,0,0,0,0,∗,0,0,0)
b) (0,0,0,0,0,0,0,0,0,∗,0,1,0)

Here, case a) can be transformed to matrix 1-4-3 (which was case c) above), by swapping rows 3 and 4 (and of course also columns 11 and 10). And also case b) can be transformed to that matrix by a somewhat more complex operation: A circular swap of rows 1, 3, and 4, followed by a bit-flipping in column 12.

Now, we consider depth 4, where we have three matrices to be considered. We start with matrix 1-4-1:

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,0,0)
(0,0,0,0,0,0,0,0,0,∗,0,0,0)

Again, we have three possible next rows, namely:

a) (0,0,0,0,0,0,0,0,∗,0,0,0,0)
b) (1,0,0,0,0,0,0,0,∗,0,0,0,0)
c) (0,0,0,0,0,0,0,0,∗,0,0,1,0)

In cases a) and c) one can check that all possible characteristic sequences, which fit to the 5 answer sequences (i.e. do not differ from any of these in more than 3 bits) have at most 3 bits of value 1. Thus, (0,0,0,0,0,0,0,0,0,0,0,0,0) is a solution for the 13-inputs problem, which we desired.

Case b) has to be considered in the next depth, it will be matrix 1-5-1.

Our next (partial) matrix is 1-4-2:

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,0,0)
(1,0,0,0,0,0,0,0,0,∗,0,0,0)

We have two possible next rows, namely:

a) (0,0,0,0,0,0,0,0,∗,0,0,0,0)
b) (1,0,0,0,0,0,0,0,∗,0,0,0,0)

Both cases can be transformed into matrix 1-5-1, case a) just by swapping rows 4 and 5 (and columns 9 and 10), case b) by the permutation 21534, meaning rows 1 and 2 are swapped, rows 3, 4 and 5 make a circular swap. Of course, also here the respective columns have to be swapped, too. And finally column 1 has to be flipped.

The last matrix on depth 4 is 1-4-3:

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,0,0)
(0,0,0,0,0,0,0,0,0,∗,0,1,0)

We have two possible next rows, namely:

a) (0,0,0,0,0,0,0,0,∗,0,0,0,0)
b) (0,0,0,0,0,0,0,0,∗,0,0,1,0)

In the first case, we have (0,0,0,0,0,0,0,0,0,0,0,0,0) is a solution for the 13-inputs problem, as desired. The second case has to be considered as matrix 1-5-2.

Now, on depth 5, we have matrix 1-5-1 first:

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,0,0)
(0,0,0,0,0,0,0,0,0,∗,0,0,0)
(1,0,0,0,0,0,0,0,∗,0,0,0,0)

There are two possible next rows:

a) (0,0,0,0,0,0,0,∗,0,0,0,0,0)
b) (1,0,0,0,0,0,0,∗,0,0,0,0,0)

The first case leads to the solution (0,0,0,0,0,0,0,0,0,0,0,0,0) for the 13-inputs problem. Case b) will be matrix 1-6-1 to be considered in depth 6.

Further, we have matrix 1-5-2:

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,0,0)
(0,0,0,0,0,0,0,0,0,∗,0,1,0)
(0,0,0,0,0,0,0,0,∗,0,0,1,0)

There are two possible next rows:

a) (0,0,0,0,0,0,0,∗,0,0,0,0,0)
b) (0,0,0,0,0,0,0,∗,0,0,0,1,0)

Both lead to solutions, namely (0,0,0,0,0,0,0,0,0,0,0,0,0) in the first case, and (0,0,0,0,0,0,0,0,0,0,0,1,0) in case b).

Finally, we have to consider matrix 1-6-1:

(0,0,0,0,0,0,0,0,0,0,0,0,∗)
(1,0,0,0,0,0,0,0,0,0,0,∗,0)
(0,0,0,0,0,0,0,0,0,0,∗,0,0)
(0,0,0,0,0,0,0,0,0,∗,0,0,0)
(1,0,0,0,0,0,0,0,∗,0,0,0,0)
(1,0,0,0,0,0,0,∗,0,0,0,0,0)

There are two possible next rows:

a) (0,0,0,0,0,0,∗,0,0,0,0,0,0)
b) (1,0,0,0,0,0,∗,0,0,0,0,0,0)

The first case leads to the solution (0,0,0,0,0,0,0,0,0,0,0,0,0) for the 13-inputs problem, the second one to the solution (1,0,0,0,0,0,0,0,0,0,0,0,0).

No more case is left, so the breadth-first search of the tree of possible answer sequences in the case maxdist value 1 is complete. This proves the lemma.

□

4 Our algorithm

Now we want to complete the proof of $(9,12)\mathrm{P} = (10,13)\mathrm{P}$. Therefor we have to consider the cases of maxdist value 2, 3, or 4. This now is definitely a case for the computer. But, in order to reduce computation time, we want to reduce the problem as far as possible.

In Section 2 we already mentioned that we will use transformations, and we did apply this technique in Section 3 for the easy cases. We justify this by the following discussion.

Let $x_1, \ldots, x_{13}$ be any sequence of (pairwise different) inputs, which are to be checked for membership in L, the given language from $(9,12)\mathrm{P}$. By 13 queries to the according algorithm we will obtain 13 times 12 output bits for the possible combinations of 12 out of the 13 inputs. We will know that at least 9 output bits in each answer sequence are correct. (But of course, we do not a priori know which ones!)

Thus the output could look like:

(0,1,∗,0,1,1,0,0,1,1,0,0,1)
(1,0,1,0,1,∗,1,1,1,0,1,0,1)
(0,0,1,0,1,0,0,0,1,0,0,1,∗)
(0,0,0,0,0,1,∗,0,1,0,0,0,1)
(0,0,1,∗,1,0,1,0,1,0,0,0,1)
(0,0,1,0,∗,0,0,0,1,0,0,0,1)
(0,0,1,0,1,1,1,∗,1,1,1,0,1)
(0,0,0,0,0,1,1,1,1,0,0,∗,1)
(∗,0,0,0,0,1,1,0,1,0,0,1,1)
(0,0,1,0,1,1,1,1,1,0,∗,1,1)
(0,∗,1,1,1,1,1,0,1,0,0,1,1)
(0,0,1,0,1,0,1,0,1,∗,0,1,1)
(0,0,1,0,0,0,1,0,∗,0,0,1,1)

Clearly, we can choose to write any row first, then any other row second, and so forth. But it is obvious that there is only one way to arrange the rows, where the asterisks appear in the diagonal from top right to bottom left:

$$
\begin{array}{c}
(0,0,1,0,1,0,0,0,1,0,0,1,*)\\
(0,0,0,0,0,1,1,1,1,0,0,*,1)\\
(0,0,1,0,1,1,1,1,1,0,*,1,1)\\
(0,0,1,0,1,0,1,0,1,*,0,1,1)\\
(0,0,1,0,0,0,1,0,*,0,0,1,1)\\
(0,0,1,0,1,1,1,*,1,1,1,0,1)\\
(0,0,0,0,0,1,*,0,1,0,0,0,1)\\
(1,0,1,0,1,*,1,1,1,0,1,0,1)\\
(0,0,1,0,*,0,0,0,1,0,0,0,1)\\
(0,0,1,*,1,0,1,0,1,0,0,0,1)\\
(0,1,*,0,1,1,0,0,1,1,0,0,1)\\
(0,*,1,1,1,1,1,0,1,0,0,1,1)\\
(*,0,0,0,0,1,1,0,1,0,0,1,1)
\end{array}
$$

We always choose the sequence of rows in this way and call that a normalization. Now, whenever we swap two rows, or more generally whenever we perform a permutation on the rows, we have to swap columns too, in order to keep our normalized form. Note, that swapping of rows i and j has to be followed by swapping of columns $14-i$ and $14-j$ (for $1 \leq i < j \leq 13$), and similarly for general permutations.

However, while the ordering of the rows technically means nothing, the ordering of the columns refers to the ordering of the queried inputs $x_1, \ldots, x_{13}$. Thus, one should keep in mind that such an operation always means a rearrangement of the inputs.

Lemma 3 *If the normalized matrix A can be transformed into the normalized matrix B by a sequence of row and column permutations, then B is consistent if and only if A is, and B allows a solution, if and only if A does.*

Proof: Let us consider any bit sequence $a_1, \ldots, a_{13}$, and let d be the maximum distance of this sequence to any row of A. The value of d does not change by a row permutation. As for column permutations, we observe that the sequence $b_1, \ldots, b_{13}$, obtained from the original sequence by applying the same column permutation, will have the same maximum distance to the permuted matrix as had the original sequence to the original matrix. Thus, there will be a sequence of distance less than 4 to all rows of B, if and only if there is a sequence of distance less than 4 to all rows of A. What we showed is thus: There is a possible truth according to the answer sequence of A if and only if there is a possible truth according to the answer sequence of B, and so A is consistent if and only if B is.

But in fact, the set of all possible truths (i.e. all 13-bit vectors which could according to the answer sequence A still be the correct characteristic sequences) transforms by the same column permutation used for the matrices exactly to the set of all possible truths according to B. In other words, there is a solution for A if and only if there is a solution for B.

□

Corollary 1 *If the normalized partial matrix A (i.e. an upper part of a normalized complete matrix) can be transformed into the normalized partial matrix B by a sequence of row and column permutations, then B can be extended to a consistent matrix if and only if A can, and B already allows for a solution, if and only if A does.*

We will normalize the matrices further: We want to consider only such matrices, where the first row is $(0,0,0,0,0,0,0,0,0,0,0,0,*)$, and the second row has a 0 in the 13-th place. This can be arranged by bit-flipping of all columns, where the first row had a 1, resp. of column 13, if the second row had a 1 there. This operation may be done because of the following lemma:

Lemma 4 *If the normalized matrix A can be transformed to B by bit-flipping on one or more columns, then B is consistent if and only if A is, and B allows a solution, if and only if A does.*

Proof: It suffices to prove the claim for bit-flipping on one column. Let column i be flipped. If $a_1, \ldots, a_{13}$ is a possible truth for the original matrix A, then the sequence obtained by flipping a_i is a possible truth for the new matrix B, and vice versa. Thus, A is consistent if and only if B is. If a solution for A is given, then it can be changed to a solution for B by flipping the i-th bit, and also vice versa. Thus, a solution for A exists, if and only if a solution for B exists.

□

Notation: The relation between (partial) matrices, given in such a way that A is related to B if and only if A can be transformed to B by the above introduced transformation rules, is an equivalence relation. We will often call matrices related in this way to be *symmetric* to each other.

By the above normalization rules we will always be able to transform a given matrix to the following form:

- The asterisk of row i is in column $14 - i$.
- The first row consists only of 0 entries (and the asterisk in column 13).
- The second row has a 0 in column 13.
- The distance between rows 1 and 2 is exactly the value of maxdist.
- In each row, the following holds: If columns j and $j + 1$ are exactly equal in all rows above the current one, then in the current row we may not have a 0 in column j and a 1 in column $j + 1$. (We call that the monotonicity rule; it could also be called the "1s to the left" rule.)
- The monotonicity rule, together with the rule about the distance between rows 1 and 2 implies for row 2 that there are 1-entries exactly in columns 1 to maxdist, and all other entries in row 2 are 0 (except for the asterisk in column 12).

The discussion proved that in order to show that all possible (that is consistent) matrices have solutions, it is sufficient to only consider matrices in the described form.

Now we are ready to introduce our algorithm for the cases of maxdist values 2, 3, and 4. The pseudo-code formulation of the algorithm is given in Appendix A. An exe-file with our implementation (and the source code in ADA) can be found on the web page

`http://www.fmi.uni-stuttgart.de/ti/personen/Hertrampf/frequency/`

Essentially, the algorithm works exactly as the explicit procedure we performed for the case of maxdist value 1 in Lemma 2. For a given value of maxdist (2, 3, or 4) it starts with the first two rows, which are uniquely determined by the above described form and the value of maxdist. We call that stage "depth 2" and we initialize a list of matrices to be considered with that one matrix of two rows. Moreover we compute the set of possible truths, i.e. the set of all 13-bit vectors which have distance at most 3 to both rows of that matrix, and we attach that set to the matrix.

Now for a given depth (starting from depth 2), we take all matrices from the list, compute all candidates for next rows, which have a distance less than or equal to maxdist to all rows already in the matrix. For each of these candidate rows, we check, whether the row would obey the monotonicity rule. If it does, it is output as a new row to be considered.

Now, the algorithm has to cancel out all so far possible truths, which contradict the new row (by having distance greater than 3 to it), and then examine the remaining set of possible truths, in order to find out, whether one of the following cases happens:

a) The (partial) matrix becomes inconsistent (set of possible truths is empty), or
b) All possible truths have the same value in one position (this will lead to a solution), or
c) We can find a 13-bit vector, which has distance less than or equal to 3 for all possible truths, or
d) None of these cases, but in our list for the next depth, we can already find a matrix which can be obtained from the current one by row permutation and bit-flipping operations, or
e) None of the other cases, so we have to append the new matrix to the list for the next depth.

The most complex test to be performed here is for case d). We could try to apply all possible transformations in order to get any of the matrices already in the list. However, we do it the other way round: We feed all pairs, built of the current matrix and one matrix of the list, into a symmetry detecting procedure called *the matcher*, which uses additional structural properties of the matrices to speed up detection of nonsymmetry in many cases: The mainly used property is the vector of numbers of pairs of a given distance. If e.g. a 3 rows matrix has distance 1 between rows 1 and 2, distance 2 between rows 1 and 3, and distance 1 between rows 2 and 3, then it can only be symmetric to 3 rows matrices, where also distance 1 occurs twice, and distance 2 occurs once, because neither row or column swapping nor bit-flipping on columns can change the multiset of pairwise row distances.

Now, if one of cases a), b), c) or d) happens, the algorithm produces an output telling exactly which case happened, and moreover in case b) the special position and the value that all truths have in that position, in case c) the solution, and in case d) the row permutation that leads to the other matrix, and the number of that matrix in our list (note, that the according column permutation and the bit-flipping operations are implicitly given by our normalization conditions).

When all partial matrices of a given depth are done, the algorithm starts to examine the list for the next depth. If that is empty, the algorithm terminates.

The algorithm can be performed on maxdist values 2, 3 and 4, and it runs very fast. Its output is partially listed in Appendix B. The complete output can also be found on the above named web page. By inspection of the output one can see that in all three cases the maximum depth to be considered is depth 8. (In fact, for maxdist 2 the maximum depth is 7, for maxdist 3 it is 8, and for maxdist 4 the maximum depth is 4.) At the maximum depth the algorithm terminates in each case, meaning there are nor more matrices to consider. Thus we obtain:

Theorem 1 $(9,12)\mathrm{P} = (10,13)\mathrm{P}$.

One can observe that all (partial) matrices occuring in the execution of the algorithm have at least one column of all zeroes. If we omit one such column in every considered matrix, the whole procedure does exactly the same, only it now works on only 12 inputs. Thus, as a corollary we get:

Corollary 2 $(8,11)\mathrm{P} = (9,12)\mathrm{P}$.

5 Extendability

As we observed in the end of Section 4, every matrix used in the execution of our algorithm has at least one column of all zeroes. To be more exact: In cases maxdist value 3 or 4, we always have at least one such column, and in case maxdist value 2 (and maxdist value 1 as well, as investigated in Lemma 2), we always have at least two such columns. These zero columns will be the key in the proof, that the result of Theorem 1 can be extended to all cases with more inputs and up to three errors. Together with Corollary 2 this will complete the solution of the $d = 3$ instance of the Hinrichs-Wechsung conjecture.

To emphasize the role of zero columns we state this explicitly:

Observation: In the computation of our algorithm all partial matrices that have to be considered in cases of maxdist $\in \{3, 4\}$ have at least one zero column. All partial matrices that have to be considered in cases of maxdist $\in \{1, 2\}$ have at least two zero columns.

Now we want to explain, how the algorithm has to be interpreted for a case $(9+k, 12+k)\mathrm{P}$ vs. $(10+k, 13+k)\mathrm{P}$ for $k > 0$. We inductively assume that the whole algorithm with $k-1$ extra columns (which are all zero columns) is given and is proven to solve the case $(8 + k, 11 + k)\mathrm{P}$ vs. $(9 + k, 12 + k)\mathrm{P}$ (note that in the base case $k = 1$ this is just the claim of Theorem 1).

Clearly, our lemmas and corollaries about transformations remain valid in the general case, too. Observe that we did never really exploit the fact that we have 13×13-matrices in the respective proofs.

What we have to show is: Our case inspection is still complete, i.e. of all consistent matrices we can make sure that either an upper part of the matrix itself or an upper part of a transformed form is found to have a solution.

The main aspect here is to make sure that we only stop investigation (because of either inconsistency, transformation to other (further investigated) cases, or provable existence of solution) if the respective reason for stopping is also given in the extended case.

The second aspect is to make sure that the set of candidates in the extended case is identical (apart from the extra zero bit) with the set of candidates in the other case.

We start the proof by some easy facts:

Definition 4 *For a matrix C, we denote by C^+ the matrix obtained from C by adding a column, where all bits are zero. (Note that the new column has no asterisk in any row.)*

Lemma 5 *If A can be transformed to B by our symmetry operations, then also A^+ can be transformed to B^+, essentially by the same operations.*

Proof: Assume w.l.o.g. the new column in A^+ and B^+ to be the rightmost one. Then, application of the same transformations that transform A to B also transforms A^+ to B^+, because the zero column will not be altered by any row or column permutation, and bit-flipping can only be performed on the columns already present in A.

□

Lemma 6 *If a partial matrix in the algorithm leads to a solution via case b) (which means, we definitely know the characteristic function w.r.t. L on one x_i), then we can also conclude that all x_j such that the j-th column is a zero column will have value 0 in all possible truths.*

Proof: If there was a possible truth T, which has a 1, where our matrix has a particular zero column, then we could define a vector T', which coincides with T, except that in the i-th column it carries the opposite value, and has a 0 at the particular zero column. The distance between T' and the rows of our matrix is not greater than the distance between T and the rows. Thus, as T is a possible truth, also T' is one, contradicting the assumption that in all possible truths the i-th bit is equal.

□

Now we can proof the main lemma of this section:

Lemma 7 *Let A be any (partial) matrix which has to be considered in the algorithm for the extended problem (for maxdist values 1, 2, 3 or 4). Then there is a matrix B, which will be considered in the non-extended case, such that*

1) A is symmetric to B^+

2) If B satisfies case a), b), c) or d) then A satisfies the same case.

Proof: Assume there is a matrix A, which contradicts the claim. Then we can choose such an A of minimal depth. A must have been constructed by a matrix A' which fulfilled the claim, but had one less rows. (A has more than two rows, because there is always only one matrix of depth two, i.e. with two rows, to be considered.)

A' does not satisfy any of cases a), b), c) or d), because otherwise A would not have to be considered. So, by the minimality of A, there is a matrix B', such that A' is symmetric to B'^+, and B' does not satisfy any of cases a), b), c) or d).

Now, there are two cases:

case 1: A has no zero column. Then, as A can be obtained from A' by adding a possible next row, there is a matrix B'' that can be obtained from B'^+ by adding a possible next row, and B'' is symmetric to A. Moreover, if we omit one column (without asterisk) of B'', then we obtain a matrix to be considered in the non-extended case. But, since A has no zero columns, B'' has neither, thus also in the non-extended case there would be a matrix without zero columns that has to be considered. This contradicts the (inductive) assumption that every matrix to be considered has at least one zero column.

case 2: A has a zero column. Then, omitting one such column we obtain a matrix $\tilde{B}$, such that A is symmetric to $\tilde{B}^+$.

It remains to show that if $\tilde{B}$ satisfies case a), b), c), or d), then $\tilde{B}^+$ does so, too.

We start with case d): If $\tilde{B}$ can be transformed to a matrix C, which has been added to the list before, then $\tilde{B}^+$ can be transformed to C^+, which will be added to the list in the extended problem, before we consider $\tilde{B}^+$.

Now, we turn to case a): If $\tilde{B}$ is consistent, i.e. there is a possible truth $(b_1, \ldots, b_{12+k})$ w.r.t. $\tilde{B}$, then $(b_1, \ldots, b_{12+k}, 0)$ is a possible truth w.r.t. $\tilde{B}^+$, so $\tilde{B}^+$ is consistent. This settles the case.

As to case b), assume that $\tilde{B}$ leads to knowledge of a certain output bit (say the i-th bit). Then all possible truths for $\tilde{B}$ have the same i-th bit, so also all possible truths for $\tilde{B}^+$ have the same i-th bit, because every such truth by omitting the last bit is a truth for $\tilde{B}$. So, also $\tilde{B}^+$ leads to knowledge of a certain output bit.

It remains to show the claim for case c): Assume that $\tilde{B}$ allows for a solution, i.e. all possible truths have distance at most 3 to a given vector $(b_1, \ldots, b_{12+k})$. We want to show that $(b_1, \ldots, b_{12+k}, 0)$ has distance at most 3 to all possible truths w.r.t. $\tilde{B}^+$, thus also $\tilde{B}^+$ allows for a solution. Now assume a possible truth w' w.r.t. $\tilde{B}^+$, which has distance at least 4 to $(b_1, \ldots, b_{12+k}, 0)$. Certainly, the restriction w of that truth to the first $12+k$ components is a possible truth for $\tilde{B}$, and thus has distance at most 3 to $(b_1, \ldots, b_{12+k})$. So, we have to have bit $13+k$ to be 1 in w', and moreover the distance of w to $(b_1, \ldots, b_{12+k})$ must be exactly 3. We will distinguish two cases again:

First, assume that there is a zero column of $\tilde{B}$, such that in the according position, say position i, w has a zero. Thus we know that $\tilde{B}^+$ has at least two zero columns, namely columns i and $13+k$, and in w' in the i-th position we have value 0, but in the $13+k$-th position value 1. Now, exchanging the bits i and $13+k$ in w' must also lead to a truth for $\tilde{B}^+$, so the restriction to the first $12+k$ bits is a truth for $\tilde{B}$. On the other hand, the distance of that new truth to $\tilde{B}$ is greater than the one of w to $\tilde{B}$, because we just changed one bit in a zero column to value 1. But then we have a truth with distance 4 w.r.t. to $\tilde{B}$, a contradiction. So, our assumption about a zero column, where w has a zero must be wrong.

Second, assume that at all zero columns of $\tilde{B}$, the according position in w has value 1. If we have maxdist at least 3, then in the first two rows of $\tilde{B}$, this truth makes at least 5 errors, because we have at least one zero column, where it makes two errors, and the first three columns, where it makes three more errors. Then, w' makes 7 errors in the first two rows, and so can not be a truth for $\tilde{B}^+$. If otherwise maxdist is at most 2, then we always have at least two zero columns in $\tilde{B}$ (see the Observation in the beginning of this section). Thus, in cases of maxdist 1 or maxdist 2, we have again at least 5 errors of w against $\tilde{B}$ in the first two rows, and thus 7 errors of w' against $\tilde{B}^+$. Again a contradiction.

□

From Lemma 7, we immediately obtain:

Theorem 2 *For all $k > 0$, we have $(8+k, 11+k)\mathrm{P} = (8, 11)\mathrm{P}$.*

Proof: The extendability of our algorithmic proof of $(9,12)\mathrm{P} = (10,13)\mathrm{P}$ for the cases of maxdist 1, 2, 3 or 4 has been shown in Lemma 7. For the other cases, observe that Lemma 1 can easily be adapted to be valid for the higher cases too. Finally add Corollary 2 to obtain equality with $(8,11)\mathrm{P}$.

□

6 Further work

One obvious next step would be an adaptation of the proof to case $d = 4$. We plan to do this by a simple extension of our algorithm, which will be added to the above indicated web page, as soon as it is available. If further evidence is needed that the Hinrichs-Wechsung conjecture is reasonable, we feel that this evidence would be given, if also in that case the correctness can be shown.

But, certainly more interesting, a (theoretical) proof should be given for the following conjecture:

Conjecture: For any $m < n$ we have:

$$(m,n)\mathrm{P} = (m+1,n+1)\mathrm{P} \Longrightarrow (m+1,n+1)\mathrm{P} = (m+2,n+2)\mathrm{P}$$

Clearly, a proof of this conjecture would make lengthy considerations like the one in Section 5 obsolete. Then, in order to prove for any d that the Hinrichs-Wechsung conjecture holds, it would suffice to just show that $(2^d, 2^d + d)\mathrm{P} = (2^d + 1, 2^d + d + 1)\mathrm{P}$. However, although the above conjecture looks very plausible, it seems very hard to prove.

Finally, we should remark that a lot more questions about relations between resource bounded frequency classes have been open a long time: E.g. can one prove or disprove that for all pairs (m, n) with $5m \geq 3n \geq 15$ we have $(3, 5)\mathrm{P} \subseteq (m, n)\mathrm{P}$?

References

[ADH03] Holger Austinat, Volker Diekert, and Ulrich Hertrampf. A structural property of regular frequency computations. *Theoretical Computer Science*, 292(1):33–43, 2003.

[ADHP00] Holger Austinat, Volker Diekert, Ulrich Hertrampf, and Holger Petersen. Regular frequency computations. In *Proceedings of the RIMS Symposium on Algebraic Systems, Formal Languages and Computation*, pages 35–42, Kyoto, Japan, 2000.

[BGK96] Richard Beigel, William I. Gasarch, and Efim B. Kinber. Frequency computation and bounded queries. *Theoretical Computer Science*, 163(1–2):177–192, 1996.

[Deg81] A. N. Degtev. On (m, n)-computable sets. *Algebraic Systems*, pages 88–99, 1981. (In Russian).

[HKO92] Valentina S. Harizanov, Martin Kummer, and James C. Owings. Frequency computations and the cardinality theorem. *Journal of Symbolic Logic*, 57(2):682–687, 1992.

[HW97] Maren Hinrichs and Gerd Wechsung. Time bounded frequency computations. *Information and Computation*, 139(2):234–257, 1997.

[Kin72] Efim B. Kinber. Frequency calculations of general recursive predicates and frequency enumeration of sets. *Soviet Mathematics Doklady*, 13:873–876, 1972.

[Kin74] Efim B. Kinber. On frequency-enumerable sets. *Algebra i Logika*, 13:398–419, 1974. (In Russian; English translation in *Algebra and Logic*, 13:226–237, 1974).

[Kin76] Efim B. Kinber. Frequency computations in finite automata. *Kibernetika*, 2:7–15, 1976. (In Russian; English translation in *Cybernetics*, 12:179–187, 1976).

[KS95a] Martin Kummer and Frank Stephan. The power of frequency computation. In Horst Reichel, editor, *Proceedings of the Tenth International Congress on Fundamentals of Computation Theory (FCT 1995)*, LNCS 969, pages 323–332, 1995.

[KS95b] Martin Kummer and Frank Stephan. Recursion theoretic properties of frequency computation and bounded queries. *Information and Computation*, 120:59–77, 1995.

[Kum92] Martin Kummer. A proof of Beigel's cardinality conjecture. *Journal of Symbolic Logic*, 57(2):677–681, 1992.

[McN61] Robert McNaughton. The theory of automata, a survey. *Advances in Computers*, 2:379–421, 1961.

[McN95] Timothy McNicholl. *The Inclusion Problem for Generalized Frequency Classes*. PhD thesis, George Washington University, Washington D.C., 1995.

[Ros60] Gene F. Rose. An extended notion of computability. In *Abstracts of the International Congress for Logic, Methodology, and Philosophy of Science*, page 14, Stanford, California, 1960.

[Tan02] Till Tantau. Towards a cardinality theorem for finite automata. In Krzysztof Diks and Wojciech Rytter, editors, *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS 2002), Warsaw, Poland*, LNCS 2420, pages 625–636, 2002.

[Tra63] Boris A. Trakhtenbrot. On the frequency computation of functions. *Algebra i Logika*, 2:25–32, 1963. (In Russian).

Appendix A: The Algorithm in Pseudo-Code

```
% function definitions:
% ---------------------
% breadth_first_tree_search   : establishes a whole maxdist-tree depth by depth
% init_truth_administration   : establishes a truth administration regarding a single row
%                               establishes an array of all vectors
%                                   with a distance $leq$ 3 to the regarded row
%                               counts the truths
%                               counts the Bits valued 1 in truths for each position.
% update_truth_administration : updates a truth administration regarding another row
%                               deletes each truth with a distance > 3 to the regarded row
%                               counts the residual truths
%                               counts the Bits valued 1 in truths for each position.
% next_depth                  : computes the next depth using the last depth
% case_abc                    : checks if any of the cases a), b), c) occurs for
%                                  a matrix by the matrix's truth_administration
% try_and_find_solution       : tries to find a solution for a given set of truths
% the_matcher                 : check's if two matrices are symmetric
% appropriate_permutations    : establishes a list of permutations for which two
%                                  pairs of distance_matrices would match
% permute_distance_matrix     : "permutes" a distance_matrix
%                               given a distance_matrix A' of a matrix A and a permutation p
%                                  this function directly (not using B) computes the
%                                  distance_matrix B' for a matrix B := (A under Permutation p)
% permute_&_restructure_matrix: applies a row permutation to a matrix and swaps the columns in
%                                  the right part in order to restore normalized form
% restore_matrix_header       : bitflips and swaps columns in order to renormalize
%                                  the first two rows of a matrix
% try_and_equalize_left_part  : swaps columns in the left part of a matrix
%                                  in order to complete a matching contempt

---------------------------------------------------------------------------------------------
-----------------------------------   main function   ---------------------------------------
---------------------------------------------------------------------------------------------
PROCEDURE search_all_trees IS
BEGIN
FOR maxdist IN 2..4 DO
        breadth_first_tree_search;
OD;
END search_all_trees;

---------------------------------------------------------------------------------------------
------------------------- breadth_first_tree_search and subfunctions ----------------------
---------------------------------------------------------------------------------------------
PROCEDURE breadth_first_tree_search RETURN boolean IS
BEGIN
create new output_file and file header;
initialize row 1 and 2 of a matrix in last_depth according to current maxdist;
establish a truth administration regarding row 1;
   % use function "init_truth_administration" for that
update truth administration regarding row 2;
   % use function "update_truth_administration" for that
WHILE last_depth not_empty DO
        set depth-header in output_file;
        compute next depth and assign: last_depth := next_depth;
           % use function "next_depth" for that
OD;
set end_of_file mark in output_file;
END breadth_first_tree_search;
```

```
-------------------------------------------------------------------------------------------
FUNCTION init_truth_administration RETURN truth_administration IS
output : truth_administration;
BEGIN
FOR every vector DO
        check distance from  (0,0,0,0,0,0,0,0,0,0,0,0,*);
           % if d > 3 skip vector
        affiliate vector in truth array;
        increment number of truths;
        increment the componentes of 1-count-vector which are valued 1 in truth;
OD;
RETURN output;
END init_truth_administration;

-------------------------------------------------------------------------------------------
FUNCTION update_truth_administration(old_administration: truth_administration;
                                     answer: row) RETURN truth_administration IS
BEGIN
For all truths IN old_administration DO
        check distance to answer;
           % If d $\leq$ 3 skip truth
        decrement the componentes of 1-count-vector which are valued 1 in truth;
        delete truth in possible truths;
        decrement number of truths;
OD;
RETURN old;
END update_truth_administration;

-------------------------------------------------------------------------------------------
FUNCTION next_depth(last_depth: matrix_list) RETURN matrix_list IS
next_depth : matrix_list;
BEGIN
FOR all matrices IN last_depth DO
        FOR all possible next rows DO
                append row to matrix;
                check maxdist;
                   % If maxdist>3 skip this row
                Check monotonicity;
                   % If monotonicity violated skip this row
                Update truth administration;
                   % Delete all vectors with distance > 3
                   % w.r.t. current row
                   % use function "update_truth_administration" for that
                Check cases a, b, and c;
                   % If such a case occurs, write that
                   % to the output, and skip
                   % use function "case_abc" for that
                Check symmetry to a former case;
                   % If such a symmetry occurs, write that
                   % to the output, and skip
                   % use function "the_matcher" for all matrices in next_depth list for this
                Append new matrix to next_depth list;
                   % matrix is to be investigated further
        OD
OD
RETURN next_depth;
END next_depth;
```

```
----------------------------------------------------------------------------------------------
FUNCTION case_abc(truth_administration : truth_administration) RETURN case IS
BEGIN
IF number_of_truths = 0 THEN RETURN a; FI;
IF any component of 1_count_vector is valued 0 or valued number_of_truths THEN RETURN b; FI;
IF a solution can be found THEN RETURN c; FI;
   % use function "try_and_find_solution" for that
RETURN d; -- which means neither of the cases a,b,c
END case_abc;

----------------------------------------------------------------------------------------------
FUNCTION try_and_find_solution(truth_administration: truth_administration)
                               RETURN Boolean IS
BEGIN
v := the first truth in truth administration;
For v' with d(v',v) $\leq$ 3 DO
        IF the gained vector is a solution THEN RETURN True;
OD;
RETURN False;
END try_and_find_solution;

==============================================================================================
--------------------------------    The Matcher    -------------------------------------------
==============================================================================================
FUNCTION the_matcher(X,Y : Matrix) RETURN Boolean IS
permutations: permuations_list := NEW permuations_list_element;
correct_row_permutation: permutation;
symmetry : Boolean := FALSE;
BEGIN
Establish the row distance matrices for the left and the right part of X and Y;
Check if the the left part matrices have the same entries
   % ELSE RETURN False;
Check if the the right part matrices have the same entries
   % ELSE RETURN False;
Establish a list of row-permutations that could possibly make the distance matrices match
   % use function "appropriate_permutations" for this
FOR all these row-permutations DO
        apply the permutation to matrix Y;
           % use function "permute_&_restructure_matrix" for this
           % the function also sorts the columns in the right part according to norm 1
        restore the correct form of the first two rows by symmetrie-operations
           % use function "restore_matrix_header" for this
        check if right parts of the matrices match
           % If they don't, skip this permutation
        try to attain a matching for the left parts of the matrices
           % use function "try_and_equalize_left_part" for this
           % If possible then write the permutation into the output_file
OD
END the_matcher;

----------------------------------------------------------------------------------------------
FUNCTION appropriate_permutations(A1,B1,A2,B2: Halbmatrix)
                                  RETURN permutations_list IS
BEGIN
For every permutation DO
        Apply Permutation to B1 and B2
           % use function "permute_distance_matrix" for this
        IF A1 = B1 AND A2 = B2 THEN Append permutation to list; FI;
OD;
RETURN permutations;
End appropriate_permutations;
```

```
-------------------------------------------------------------------------------------------
FUNCTION permute_distance_matrix(A: Halbmatrix; P: Permutation)
                                 RETURN Halbmatrix IS
BEGIN
For all entries in A DO
        A'(i,j) := A(P(i),P(j))
           % An entry (i,j) in A marks the distance between row i an j
           % After the permutation it is meant to mark the distance between p(i) and p(j)
OD
RETURN A';
END permute_distance_matrix;

-------------------------------------------------------------------------------------------
FUNCTION permute_&_restructure_matrix(A: Matrix; Q: Permutation) RETURN Matrix IS
BEGIN
        Apply Q to the rows of A
           % A'(i) := A(P(i)); for rows
        Sort the rows of the right part according to Norm 1
           % A'(14-i) := A(14-Q(i)); for columns
RETURN A';
END permute_&_restructure_matrix;

-------------------------------------------------------------------------------------------
FUNCTION restore_matrix_header(A: Matrix) RETURN Matrix IS
BEGIN
        edit A in order to make row 1 "000000000000*"
           % Flip columns i with A(1)(i) = 1
        edit A in order to make row 2 end with "..*0"
           % IF A(2)(13) = 1 THEN Flip column 13
        edit A in order to make row 2 "11..00*0"
           % simply sort the columns 1..11
RETURN A;
END restore_matrix_header;

-------------------------------------------------------------------------------------------
FUNCTION try_and_equalize_left_part(A,B: Matrix) RETURN Boolean IS
BEGIN
% at the beginning the unequalized part is to be left part of the matrices
WHILE the unequalized part isn't empty DO
        take the most left column of the unequalized part in A;
        IF you can find this column in the unequalized part of B
                THEN Swap this column in B to the left (enlarging the equalized part);
                ELSE RETURN False;
        FI;
OD;
RETURN True;
END try_and_equalize_left_part;

-------------------------------------------------------------------------------------------
---------------------------------    END OF PSEUDOCODE    ---------------------------------
-------------------------------------------------------------------------------------------
```

Appendix B: The Algorithm's Output

```
      -------------------------------------------------
      |             FILE = (Maxdist = 2)              |
      -------------------------------------------------

Depth 2
---------------------------------------------------------------
000000000000*
11000000000*0 Matrix 2-2-1

0000000000*00 Proceed with 2-3-1
1000000000*00 Proceed with 2-3-2
1100000000*00 symmetric to 2-3-1 via permutation 213
1010000000*00 Proceed with 2-3-3
0000000000*10 Proceed with 2-3-4
1000000000*10 Proceed with 2-3-5
1000000000*01 symmetric to 2-3-5 via permutation 213
1100000000*01 symmetric to 2-3-4 via permutation 213
1000000000*11 Proceed with 2-3-6

Depth 3
---------------------------------------------------------------
000000000000*
11000000000*0
0000000000*00 Matrix 2-3-1

000000000*000 solution 0000000000000
100000000*000 Proceed with 2-4-1
110000000*000 Proceed with 2-4-2
101000000*000 solution 0000000000000
100000000*100 Proceed with 2-4-3
000000000*010 Proceed with 2-4-4
100000000*010 Proceed with 2-4-5
100000000*001 symmetric to 2-4-3 via permutation 3214

000000000000*
11000000000*0
1000000000*00 Matrix 2-3-2

000000000*000 symmetric to 2-4-1 via permutation 1243
100000000*000 Proceed with 2-4-6
010000000*000 Proceed with 2-4-7
110000000*000 symmetric to 2-4-1 via permutation 2143
101000000*000 Proceed with 2-4-8
100000000*100 Proceed with 2-4-9
010000000*100 Proceed with 2-4-10
000000000*010 Proceed with 2-4-11
100000000*010 Proceed with 2-4-12
100000000*001 symmetric to 2-4-12 via permutation 2134
110000000*001 symmetric to 2-4-11 via permutation 2134
100000000*011 solution 1000000000000

000000000000*
11000000000*0
1010000000*00 Matrix 2-3-3

000000000*000 solution 0000000000000
100000000*000 symmetric to 2-4-8 via permutation 1243
110000000*000 solution 1100000000000
101000000*000 solution 1010000000000
011000000*000 Proceed with 2-4-13
100100000*000 solution 0000000000000
100000000*100 solution 1000000000100
100000000*010 solution 1000000000010
100000000*001 solution 1000000000001

000000000000*
11000000000*0
0000000000*10 Matrix 2-3-4

000000000*000 symmetric to 2-4-4 via permutation 1243
100000000*000 symmetric to 2-4-11 via permutation 1243
100000000*100 Proceed with 2-4-14
000000000*010 symmetric to 2-4-4 via permutation 3241
100000000*010 symmetric to 2-4-11 via permutation 3241
100000000*011 symmetric to 2-4-14 via permutation 3214

000000000000*
11000000000*0
1000000000*10 Matrix 2-3-5

000000000*000 symmetric to 2-4-5 via permutation 1243
100000000*000 symmetric to 2-4-12 via permutation 1243
110000000*000 symmetric to 2-4-3 via permutation 4123
101000000*000 solution 1000000000010
100000000*100 Proceed with 2-4-15
000000000*010 symmetric to 2-4-11 via permutation 4231
100000000*010 solution 1000000000010
010000000*010 symmetric to 2-4-10 via permutation 3421
100000000*001 Proceed with 2-4-16
100000000*011 Proceed with 2-4-17

000000000000*
11000000000*0
1000000000*11 Matrix 2-3-6

100000000*000 solution 1000000000000
100000000*100 Proceed with 2-4-18
000000000*010 symmetric to 2-4-14 via permutation 4213
100000000*010 symmetric to 2-4-17 via permutation 1243
100000000*001 symmetric to 2-4-17 via permutation 2143
110000000*001 symmetric to 2-4-14 via permutation 4123
100000000*011 Proceed with 2-4-19
010000000*011 Proceed with 2-4-20

Depth 4
---------------------------------------------------------------

...

Depth 7
---------------------------------------------------------------
000000000000*
11000000000*0
0000000000*00
100000000*000
10000000*0000
0100000*00000
010000*000000 Matrix 2-7-1

00000*0000000 solution 0000000000000
10000*0000000 solution 1000000000000
01000*0000000 solution 0100000000000
11000*0000000 Proceed with 2-8-1
00000*0000010 solution 0000000000000

000000000000*
11000000000*0
0000000000*00
100000000*000
00000000*0010
0000000*00010
100000*000010 Matrix 2-7-2

00000*0000000 solution 0000000000000
10000*0000000 solution 1000000000000
00000*0000010 solution 0000000000010
10000*0000010 solution 1000000000010

Depth 8
---------------------------------------------------------------
000000000000*
11000000000*0
0000000000*00
100000000*000
10000000*0000
0100000*00000
010000*000000
11000*0000000 Matrix 2-8-1

0000*00000000 solution 0000000000000
1000*00000000 solution 1000000000000
0100*00000000 solution 0100000000000
1100*00000000 solution 1100000000000

----- End of File -----
```

```
        --------------------------------------------------
        |              FILE = (Maxdist = 3)              |
        --------------------------------------------------

Depth 2
------------------------------------------------------------------
000000000000*
11100000000*0 Matrix 3-2-1

0000000000*00 Proceed with 3-3-1
1000000000*00 Proceed with 3-3-2
1100000000*00 symmetric to 3-3-2 via permutation 213
1110000000*00 symmetric to 3-3-1 via permutation 213
1001000000*00 Proceed with 3-3-3
1101000000*00 symmetric to 3-3-3 via permutation 213
0000000000*10 Proceed with 3-3-4
1000000000*10 Proceed with 3-3-5
1100000000*10 Proceed with 3-3-6
1001000000*10 Proceed with 3-3-7
1000000000*01 symmetric to 3-3-6 via permutation 213
1100000000*01 symmetric to 3-3-5 via permutation 213
1110000000*01 symmetric to 3-3-4 via permutation 213
1101000000*01 symmetric to 3-3-7 via permutation 213
1000000000*11 Proceed with 3-3-8
1100000000*11 symmetric to 3-3-8 via permutation 213

Depth 3
------------------------------------------------------------------
000000000000*
11100000000*0
0000000000*00 Matrix 3-3-1

000000000*000 solution 0000000000000
100000000*000 Proceed with 3-4-1
110000000*000 Proceed with 3-4-2
111000000*000 Proceed with 3-4-3
100100000*000 solution 0000000000000
110100000*000 Proceed with 3-4-4
100000000*100 Proceed with 3-4-5
110000000*100 Proceed with 3-4-6
000000000*010 solution 0000000000000
100000000*010 solution 0000000000000
110000000*010 Proceed with 3-4-7
100100000*010 solution 0000000000000
100000000*110 Proceed with 3-4-8
100000000*001 symmetric to 3-4-5 via permutation 3214
110000000*001 symmetric to 3-4-6 via permutation 3214
110000000*101 Proceed with 3-4-9
100000000*011 symmetric to 3-4-8 via permutation 3214

000000000000*
11100000000*0
1000000000*00 Matrix 3-3-2

000000000*000 symmetric to 3-4-1 via permutation 1243
100000000*000 solution 1000000000000
010000000*000 Proceed with 3-4-10
110000000*000 Proceed with 3-4-11
011000000*000 Proceed with 3-4-12
111000000*000 symmetric to 3-4-2 via permutation 2143
100100000*000 solution 1000000000000
010100000*000 Proceed with 3-4-13
110100000*000 solution 1000000000000
100000000*100 Proceed with 3-4-14
010000000*100 Proceed with 3-4-15
110000000*100 Proceed with 3-4-16
011000000*100 Proceed with 3-4-17
000000000*010 Proceed with 3-4-18
100000000*010 Proceed with 3-4-19
010000000*010 Proceed with 3-4-20
110000000*010 Proceed with 3-4-21
100100000*010 solution 1000000000000
100000000*110 Proceed with 3-4-22
010000000*110 Proceed with 3-4-23
100000000*001 solution 1000000000000
010000000*001 Proceed with 3-4-24
110000000*001 solution 1000000000000
111000000*001 Proceed with 3-4-25
110100000*001 solution 1000000000000
110000000*101 Proceed with 3-4-26
100000000*011 solution 1000000000000
110000000*011 Proceed with 3-4-27

000000000000*
11100000000*0
1001000000*00 Matrix 3-3-3

000000000*000 solution 0000000000000
100000000*000 solution 1000000000000
010000000*000 symmetric to 3-4-13 via permutation 1243
110000000*000 solution 1100000000000
111000000*000 symmetric to 3-4-4 via permutation 2143
100100000*000 solution 1001000000000
010100000*000 Proceed with 3-4-28
110100000*000 symmetric to 3-4-13 via permutation 3241
011100000*000 Proceed with 3-4-29
100010000*000 solution 0000000000000
110010000*000 solution 1000000000000
100000000*100 solution 1000000000000
010000000*100 Proceed with 3-4-30
110000000*100 solution 1000000000000
000000000*010 solution 0000000000000
100000000*010 solution 0000000000000
110000000*010 solution 1100000000000
100100000*010 solution 1001000000000
010100000*010 Proceed with 3-4-31
100010000*010 solution 0000000000000
100000000*110 solution 1000000000000
100000000*001 solution 1000000000000
110000000*001 solution 1000000000000
110100000*001 symmetric to 3-4-30 via permutation 3214
110000000*101 solution 1000000000000
100000000*011 solution 1000000000000

000000000000*
11100000000*0
0000000000*10 Matrix 3-3-4

000000000*000 solution 0000000000000
100000000*000 symmetric to 3-4-18 via permutation 1243
110000000*000 symmetric to 3-4-25 via permutation 2143
100100000*000 solution 0000000000000
100000000*100 solution 0000000000000
110000000*100 Proceed with 3-4-32
000000000*010 solution 0000000000010
100000000*010 symmetric to 3-4-18 via permutation 3241
110000000*010 symmetric to 3-4-25 via permutation 2341
100100000*010 solution 0000000000010
100000000*110 Proceed with 3-4-33
100000000*001 symmetric to 3-4-33 via permutation 3214
100000000*011 solution 0000000000010
110000000*011 symmetric to 3-4-32 via permutation 3214

000000000000*
11100000000*0
1000000000*10 Matrix 3-3-5

000000000*000 solution 0000000000000
100000000*000 symmetric to 3-4-19 via permutation 1243
010000000*000 symmetric to 3-4-20 via permutation 1243
110000000*000 solution 1100000000000
111000000*000 symmetric to 3-4-6 via permutation 4123
100100000*000 solution 0000000000000
110100000*000 solution 1100000000000
100000000*100 solution 1000000000100
010000000*100 Proceed with 3-4-34
110000000*100 Proceed with 3-4-35
000000000*010 symmetric to 3-4-18 via permutation 4231
100000000*010 solution 1000000000010
010000000*010 Proceed with 3-4-36
110000000*010 Proceed with 3-4-37
011000000*010 symmetric to 3-4-17 via permutation 4321
100100000*010 solution 1000000000010
010100000*010 symmetric to 3-4-30 via permutation 3421
100000000*110 Proceed with 3-4-38
010000000*110 Proceed with 3-4-39
100000000*001 solution 1000000000001
110000000*001 Proceed with 3-4-40
110000000*101 Proceed with 3-4-41
100000000*011 Proceed with 3-4-42
010000000*011 Proceed with 3-4-43
110000000*011 Proceed with 3-4-44

000000000000*
11100000000*0
1100000000*10 Matrix 3-3-6

000000000*000 symmetric to 3-4-7 via permutation 1243
100000000*000 symmetric to 3-4-21 via permutation 1243
110000000*000 solution 1100000000000
101000000*000 symmetric to 3-4-24 via permutation 2143
111000000*000 symmetric to 3-4-5 via permutation 4123
100100000*000 solution 1100000000000
110100000*000 solution 1100000000000
100000000*100 Proceed with 3-4-45
110000000*100 solution 1100000000000
101000000*100 Proceed with 3-4-46
000000000*010 symmetric to 3-4-25 via permutation 2431
100000000*010 symmetric to 3-4-37 via permutation 1243
110000000*010 solution 1100000000010
001000000*010 symmetric to 3-4-17 via permutation 3421
101000000*010 Proceed with 3-4-47
100100000*010 solution 1100000000010
100000000*110 Proceed with 3-4-48
001000000*110 Bit 4 is always valued 0
100000000*001 Proceed with 3-4-49
110000000*001 solution 1100000000010
111000000*001 symmetric to 3-4-33 via permutation 4123
110100000*001 solution 1100000000010
110000000*101 solution 1100000000000
100000000*011 solution 1100000000010
```

```
110000000*011 solution 1100000000010
101000000*011 Proceed with 3-4-50

000000000000*
11100000000*0
1001000000*10 Matrix 3-3-7

000000000*000 solution 0000000000000
100000000*000 solution 1000000000000
110000000*000 solution 1100000000000
100100000*000 solution 1001000000000
010100000*000 symmetric to 3-4-31 via permutation 1243
110100000*000 symmetric to 3-4-30 via permutation 4123
100010000*000 solution 0000000000000
100000000*100 solution 0000000000000
110000000*100 solution 1100000000000
000000000*010 solution 0000000000010
100000000*010 solution 1000000000010
010000000*010 symmetric to 3-4-30 via permutation 4321
110000000*010 solution 1100000000010
100100000*010 solution 1001000000010
010100000*010 symmetric to 3-4-31 via permutation 3241
100010000*010 solution 1000000000000
100000000*110 solution 1000000000110
010000000*110 Proceed with 3-4-51
100000000*001 solution 1000000000000
110100000*001 symmetric to 3-4-51 via permutation 3214
100000000*011 solution 1000000000000
110000000*011 solution 1100000000010

...
```

Depth 6

```
----------------------------------------------------------------
000000000000*
11100000000*0
0000000000*00
100000000*000
01000000*0000
1100000*00000 Matrix 3-6-1

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 0100000000000
110000*000000 solution 1100000000000
001000*000000 solution 0000000000000
101000*000000 solution 0000000000000
011000*000000 solution 0000000000000
111000*000000 solution 1110000000000
100100*000000 solution 0000000000000
010100*000000 solution 0000000000000
110100*000000 solution 1000000000000
100000*100000 solution 0000000000000
010000*100000 solution 0000000000000
110000*100000 solution 1000000000000
001000*100000 Bit 4 is always valued 0
100000*010000 solution 0000000000000
010000*010000 solution 0000000000000
110000*010000 solution 1000000000000
101000*010000 solution 0000000000000
100000*001000 solution 0000000000000
010000*001000 solution 0000000000000
110000*001000 solution 1100000000000
011000*001000 solution 0000000000000
100000*000100 solution 1000000000000
010000*000100 solution 1100000000000
110000*000100 solution 1000000000000
000000*000010 solution 0000000000000
100000*000010 solution 0000000000000
010000*000010 solution 0000000000000
110000*000010 solution 1100000000000
100000*010010 solution 0000000000000
010000*001010 solution 0000000000000
100000*000001 solution 1000000000000
010000*000001 solution 1100000000000
110000*000001 solution 1000000000000
110000*000101 solution 1100001000000

000000000000*
11100000000*0
0000000000*00
100000000*000
11000000*0000
1110000*00000 Matrix 3-6-2

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 1110000000000
110000*000000 solution 1100000000000
001000*000000 solution 0000000000000
101000*000000 solution 0000000000000
011000*000000 solution 0000000000000
111000*000000 solution 1110000000000
100100*000000 solution 0000000000000
010100*000000 solution 0000000000000
110100*000000 solution 1000000000000
101100*000000 solution 0000000000000
100000*100000 solution 0000000000000
010000*100000 solution 0000000000000
110000*100000 solution 1000000000000
101000*100000 solution 0000000000000
100000*010000 solution 0000000000000
010000*010000 solution 0000000000000
110000*010000 solution 1000000000000
001000*010000 solution 0000000000000
101000*010000 solution 0000000000000
100000*001000 solution 0000000000000
010000*001000 solution 0000000000000
110000*001000 solution 1100000000000
101000*001000 solution 0000000000000
011000*001000 solution 0000000000000
100000*000100 solution 1000000000000
010000*000100 solution 1100000000000
110000*000100 solution 1000000000000
101000*000100 solution 1000000000000
100000*000010 solution 0000000000000
010000*000010 solution 0000000000000
110000*000010 solution 1000000000000
101000*000010 solution 0000000000000
100000*100010 solution 0000000000000
100000*000001 solution 1000000000000
010000*000001 solution 1100000000000
110000*000001 solution 1000000000000
101000*000001 solution 1000000000000
110000*000101 solution 1000000000000

000000000000*
11100000000*0
1000000000*00
010000000*000
11000000*0000
0010000*00000 Matrix 3-6-3

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 0100000000000
110000*000000 solution 1100000000000
001000*000000 solution 0010000000000
101000*000000 Proceed with 3-7-1
011000*000000 symmetric to 3-7-1 via permutation 1243567
111000*000000 solution 1110000000000
100100*000000 solution 0000000000000
010100*000000 solution 0000000000000
100000*100000 solution 0000000000000
010000*100000 solution 0000000000000
110000*100000 solution 1000000000000
100000*010000 solution 0000000000000
010000*010000 solution 0000000000000
001000*010000 Bit 4 is always valued 0
100000*001000 solution 0000000000000
010000*001000 solution 0000000000000
101000*001000 solution 1000000000000
100000*000100 solution 0000000000000
010000*000100 solution 0000000000000
011000*000100 solution 1110000000000
000000*000010 solution 0000000000000
100000*000010 solution 0000000000000
010000*000010 solution 0000000000000
100000*000001 solution 1000000000000
010000*000001 solution 1100000000000
111000*000001 Bit 4 is always valued 0

000000000000*
11100000000*0
1000000000*00
010000000*000
11000000*0000
1010000*00000 Matrix 3-6-4

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 0100000000000
110000*000000 solution 1100000000000
001000*000000 symmetric to 3-7-1 via permutation 1234576
101000*000000 solution 1010000000000
011000*000000 symmetric to 3-7-1 via permutation 2156374
111000*000000 solution 1110000000000
100100*000000 solution 0000000000000
110100*000000 solution 1000000000000
100000*100000 solution 1000000000000
010000*100000 solution 1100000000000
110000*100000 solution 1000000000000
100000*010000 solution 0000000000000
110000*010000 solution 1000000000000
001000*010000 Bit 4 is always valued 0
100000*001000 solution 0000000000000
110000*001000 solution 1000000000000
101000*001000 solution 1000000000000
100000*000100 solution 0000000000000
110000*000100 solution 1100000000000
011000*000100 Bit 4 is always valued 0
000000*000010 solution 0000000000000
```

```
100000*000010 solution 0000000000000
110000*000010 solution 1000000000000
100000*001010 solution 0000000000000
100000*000001 solution 1000000000000
110000*000001 solution 1000000000000
111000*000001 solution 1110000000000
110000*100001 solution 1100000000100

000000000000*
11100000000*0
1000000000*00
010000000*000
11000000*0000
0000000*00010 Matrix 3-6-5

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 0100000000000
110000*000000 solution 1100000000000
001000*000000 solution 0000000000000
101000*000000 solution 0000000000000
011000*000000 solution 0000000000000
100100*000000 solution 0000000000000
010100*000000 solution 0000000000000
100000*100000 solution 0000000000000
010000*100000 solution 0000000000000
110000*100000 solution 1000000000000
100000*010000 solution 0000000000000
010000*010000 solution 0000000000000
001000*010000 Bit 4 is always valued 0
100000*001000 solution 0000000000000
010000*001000 solution 0000000000000
100000*000100 solution 0000000000000
010000*000100 solution 0000000000000
000000*000010 solution 0000000000010
100000*000010 solution 1100000000000
010000*000010 solution 1100000000000
110000*000010 solution 1100000000000
100000*001010 solution 1000000000000
010000*000110 solution 1100000000000
100000*000001 solution 1000000000000
010000*000001 solution 1100000000000
110000*000011 Bit 3 is always valued 0

000000000000*
11100000000*0
1000000000*00
010000000*000
10100000*0000
0110000*00000 Matrix 3-6-6

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 0100000000000
110000*000000 symmetric to 3-7-1 via permutation 2175364
001000*000000 symmetric to 3-7-1 via permutation 1273546
101000*000000 solution 1010000000000
011000*000000 solution 0110000000000
111000*000000 solution 1110000000000
110100*000000 solution 1000000000000
001100*000000 solution 0000000000000
100000*100000 solution 0000000000000
110000*100000 solution 1000000000000
001000*100000 solution 0000000000000
010000*010000 solution 0000000000000
110000*010000 solution 1000000000000
001000*010000 solution 0000000000000
110000*001000 solution 1000000000000
001000*001000 solution 0000000000000
101000*001000 solution 1000000000000
110000*000100 solution 1110000000000
001000*000100 solution 0000000000000
011000*000100 solution 1110000000000
000000*000010 solution 0000000000000
110000*000010 solution 1000000000000
001000*000010 solution 0000000000000
110000*000001 solution 1000000000000
001000*000001 solution 1010000000000
111000*000001 solution 0110000000000

000000000000*
11100000000*0
1000000000*00
010000000*000
00000000*0010
1000000*00010 Matrix 3-6-7

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 0100000000000
110000*000000 solution 1100000000000
001000*000000 solution 0000000000000
101000*000000 solution 0000000000000
100100*000000 solution 0000000000000
100000*100000 solution 0000000000000
010000*100000 Bit 4 is always valued 0
001000*100000 Bit 4 is always valued 0
100000*010000 solution 0000000000000
110000*010000 solution 1000000000000
100000*001000 solution 0000000000000
100000*000100 solution 0000000000000
000000*000010 solution 0000000000010
100000*000010 solution 1000000000010
010000*000010 Bit 4 is always valued 0
110000*000010 solution 1100000000010
001000*000010 Bit 4 is always valued 0
100000*001010 solution 1000000000000
010000*000110 Bit 4 is always valued 0
100000*000001 solution 1000000000000
110000*000011 Bit 3 is always valued 0

000000000000*
11100000000*0
1000000000*00
010000000*000
10000000*0010
0100000*00010 Matrix 3-6-8

000000*000000 solution 0000000000000
100000*000000 solution 1000000000000
010000*000000 solution 0100000000000
110000*000000 solution 1100000000000
001000*000000 Bit 4 is always valued 0
111000*000000 solution 1000000000000
110100*000000 solution 1000000000000
100000*100000 solution 0000000000000
110000*100000 solution 1000000000000
010000*010000 solution 0000000000000
110000*010000 solution 1000000000000
110000*001000 solution 1000000000000
110000*000100 solution 1100000000000
000000*000010 Bit 4 is always valued 0
100000*000010 solution 1000000000010
010000*000010 solution 0100000000010
110000*000010 solution 1100000000010
001000*000010 Bit 4 is always valued 0
100000*001010 Bit 4 is always valued 0
010000*000110 Bit 4 is always valued 0
110000*000001 solution 1000000000000
110000*000011 solution 0100000000010

Depth 7
----------------------------------------------------------------
000000000000*
11100000000*0
1000000000*00
010000000*000
11000000*0000
0010000*00000
101000*000000 Matrix 3-7-1

00000*0000000 solution 0000000000000
10000*0000000 solution 1000000000000
01000*0000000 solution 0100000000000
11000*0000000 solution 1100000000000
00100*0000000 solution 0010000000000
10100*0000000 solution 1010000000000
01100*0000000 Bit 4 is always valued 0
11100*0000000 solution 1110000000000
10010*0000000 solution 0000000000000
10000*1000000 solution 0000000000000
01000*1000000 Bit 4 is always valued 0
10000*0100000 solution 0000000000000
11000*0100000 solution 1000000000000
10000*0010000 solution 0000000000000
00100*0010000 Bit 4 is always valued 0
10000*0001000 solution 0000000000000
10100*0001000 solution 1000000000000
10000*0000100 solution 0000000000000
01100*0000100 Bit 4 is always valued 0
00000*0000010 solution 0000000000000
10000*0000010 solution 0000000000000
10000*0000001 solution 1000000000000
11100*0000001 Bit 4 is always valued 0

----- End of File -----



       --------------------------------------------------
       |              FILE = (Maxdist = 4)              |
       --------------------------------------------------

Depth 2
----------------------------------------------------------------

...
```