Modelling Equidistant Frequency Permutation Arrays: An Application of Constraints to Mathematics

Sophie Huczynska[2], Paul McKay[1], Ian Miguel[1], and Peter Nightingale[1]

[1] School of Computer Science, University of St Andrews, UK
{ianm,pn}@cs.st-andrews.ac.uk, pgm9@st-andrews.ac.uk
[2] School of Mathematics and Statistics, University of St Andrews, UK
sophieh@mcs.st-andrews.ac.uk

Abstract. Equidistant Frequency Permutation Arrays are combinatorial objects of interest in coding theory. A frequency permutation array is a type of constant composition code in which each symbol occurs the same number of times in each codeword. The problem is to find a set of codewords such that any pair of codewords are a given uniform Hamming distance apart. The equidistant case is of special interest given the result that any optimal constant composition code is equidistant. This paper presents, compares and combines a number of different constraint formulations of this problem class, including a new method of representing permutations with constraints. Using these constraint models, we are able to establish several new results, which are contributing directly to mathematical research in this area.[1]

1 Introduction

In this paper we consider Equidistant Frequency Permutation Arrays (EFPAs), combinatorial objects of interest in coding theory. A frequency permutation array (introduced in [1]) is a special kind of constant composition code (CCC), in which each symbol occurs the same number of times in each codeword. CCCs have many applications, for example in powerline communications and balanced scheduling, and have recently been much studied (eg [2], [3]). The situation when CCCs are equidistant is of particular interest, since it is known that any CCC which is optimal must be equidistant. EFPAs are introduced in [4], where various bounds and constructions are obtained; other results on families of such codes can be found in [5].

Informally, the problem is to find a set (often of maximal size) of codewords, such that any pair of codewords are Hamming distance d apart. Each codeword (which may be considered as a sequence) is made up of symbols from the alphabet $\{1, \ldots, q\}$, with each symbol occurring a fixed number λ of times per codeword.

The problem has parameters v, q, λ, d and it is to find a set E of size v, of sequences of length $q\lambda$, such that each sequence contains λ of each symbol in the set $\{1, \ldots, q\}$. For each pair of sequences in E, the pair are Hamming distance d apart (i.e. there are d

[1] Sophie Huczynska is supported by a Royal Society Dorothy Hodgkin Research Fellowship, and Ian Miguel was supported by a UK Royal Academy of Engineering/EPSRC Research Fellowship. Peter Nightingale is supported by EPSRC grant EP/E030394/1.

I.P. Gent (Ed.): CP 2009, LNCS 5732, pp. 50–64, 2009.

Table 1. EFPA example with $v = 5$, $q = 3$, $\lambda = 2$, $d = 4$

c_1	0 0 1 1 2 2
c_2	0 1 0 2 1 2
c_3	0 1 2 0 2 1
c_4	0 2 1 2 0 1
c_5	0 2 2 1 1 0

places where the sequences disagree). For the parameters $v = 5$, $q = 3$, $\lambda = 2$, $d = 4$, Table 1 shows a set $E = \{c_1, c_2, c_3, c_4, c_5\}$.

Computers have long been used to assist in solving related mathematical problems. Slaney et al. found great success in using automated reasoning to attack quasigroup existence problems [6].

We model and solve EFPA in constraints, using the Minion solver [7,8] and the Tailor modelling assistant [9]. Constraint solving proceeds in two phases. First, the problem is *modelled* as a set of discrete decision variables, and a set of constraints (relations, e.g. $x_1 + x_2 \leq x_3$) on those variables that a solution must satisfy. Second, a constraint solver is used to search for solutions to the model: assignments of values to variables satisfying all constraints. Many constraint solvers (including Minion) interleave propagation and depth-first backtracking search. Propagation simplifies the problem by removing values from variable domains, and search assigns values to variables, searching for a solution. A successful model must propagate well, and will also include an ordering of the variables for search.

We investigate six different formulations, starting with two simple models based on two viewpoints (where a viewpoint is a choice of variables and domains sufficient to characterise the problem). A set of implied constraints is derived, which prove to be useful in experiments. Furthermore, we develop a novel method of modelling permutations with constraints.

The work described in this paper has direct application in mathematics. During the course of this work, we generated 24 EFPAs (21 of which were proven to be maximal) to assist with the mathematical research of one of the authors (Huczynska). Some of these EFPAs directly refuted a working conjecture, and others provided supporting evidence that a construction is maximal. This illustrates that, with careful modelling and the power of fast constraint solvers such as Minion and modelling assistants such as Tailor, constraint programming can contribute to research in other disciplines.

Experiments are performed to compare the different models, using 24 carefully chosen instances of EFPA.

2 Modelling the EFPA Problem

First we present two straightforward models based on two viewpoints. One represents the set of sequences explicitly in a two-dimensional table (like Table 1). The other is similar, but extends the two-dimensional table in a third dimension, expanding each original variable into a set of Boolean variables corresponding to each original value. The problem constraints and symmetry breaking constraints are quite different on these

two models. The three-dimensional model is able to break symmetry in three planes rather than two, and the two-dimensional model is able to take advantage of the Global Cardinality Constraint (GCC) [10] to enforce the requirement that there are λ occurrences of each symbol. Both straightforward models perform reasonably well, which reflects well on constraint programming.

The two simple models are developed in various ways to give six variants in total. Firstly, the two models are channelled together, combining the symmetry breaking constraints on the three-dimensional Boolean model with problem constraints on the two-dimensional model. The channelled model is superior to the Boolean model in almost all cases in our experiments.

Secondly, a set of implied constraints are derived for the two-dimensional model. The first row of the table is fixed by the symmetry breaking constraints. All other rows are related to the first by the Hamming distance d, and we derive additional GCC constraints from this. The implied constraints are somewhat similar to those proposed for BIBD [11]. The implied constraints are beneficial in most cases in our experiments.

Thirdly, we explicitly model the permutation between each pair of sequences, using a representation of cycle notation. While permutation has been modelled frequently in CSP and SAT (e.g. [12,13]), this is to the best of our knowledge the first time cycle notation has been explicitly modelled using standard CSP integer variables. This approach greatly reduces the search space. Unfortunately this model has a large number of constraints and is not always superior in terms of solver time. However, it shows promise because of the reduction in search space.

To model the EFPA problem, we used the Tailor modelling assistant [9]. We formulated each model in the solver-independent modelling language ESSENCE′ and used Tailor v0.3 to flatten each instance for input to the constraint solver Minion 0.8.1. Tailor provides optional common subexpression elimination (CSE) [14]. Preliminary experiments revealed that CSE improved the speed of Minion by a small margin, without affecting the search tree explored. We use CSE throughout.

2.1 Boolean and Non-Boolean Models

In this section, we investigate two viewpoints and construct two models based on the viewpoints, and a third which channels them together. The three models are compared experimentally.

A Non-Boolean Model. Viewed abstractly, the problem is to find a fixed-size set of codewords, where a codeword is a permutation of a multiset of symbols. Therefore the decisions are how to represent the set, and how to represent the permutations. In the non-Boolean model, we use an explicit representation of the set [15] (i.e. each element is explicitly represented). For each codeword, we use the primal model of a permutation [12]. Each position in the codeword has one variable whose domain is the alphabet. We do not explore the dual model here, because it would complicate the Hamming distance constraints.

There is an additional set of Boolean variables representing where pairs of codewords differ. The variables are as follows.

- $\forall a \in \{1 \dots v\}, \forall i \in \{1 \dots q\lambda\} \ : \ c[a,i] \in \{1 \dots q\}$ representing the set of sequences. a is the sequence number and i is the index into the sequence.
- $\forall a \in \{1 \dots v\}, \forall b \in \{a+1 \dots v\}, \forall i \in \{1 \dots q\lambda\} \ : \ \mathit{diff}[a,b,i] \in \{0,1\}$ representing whether two sequences a and b differ at position i.

For each sequence, a GCC constraint is used to ensure that there are λ occurrences of each symbol in the alphabet. A reified disequality constraint is used to connect the *diff* variables to c, and a sum is used to enforce the Hamming distance d.

- $\forall a \in \{1 \dots v\} \ : \ \text{GCC}(c[a, 1 \dots q\lambda], \langle 1 \dots q \rangle, \langle \lambda \dots \lambda \rangle)$
- $\forall a \in \{1 \dots v\}, \forall b \in \{a+1 \dots v\}, \forall i \in \{1 \dots q\lambda\} \ : \ \mathit{diff}[a,b,i] \Leftrightarrow (c[a,i] \neq c[b,i])$
- $\forall a \in \{1 \dots v\}, \forall b \in \{a+1 \dots v\} \ : \ \sum_{i=1}^{q\lambda} \mathit{diff}[a,b,i] = d$ (any pair of sequences differ in d places)

The matrix c has a number of symmetries. In a solution, rows, columns and symbols of the alphabet may be freely permuted to create other solutions. To break some of the symmetry, we apply lexicographic ordering (lex-ordering) constraints to the rows and columns, following Flener et al. [16].

- $\forall a \in \{2 \dots v\} \ : \ c[a-1, 1 \dots q\lambda] \leq_{lex} c[a, 1 \dots q\lambda]$ (rows are lex-ordered)
- $\forall b \in \{2 \dots q\lambda\} \ : \ c[1 \dots v, b-1] \leq_{lex} c[1 \dots v, b]$ (columns are lex-ordered)

These two constraint sets do not explicitly order the symbols. It would be possible to order the symbols by using value symmetry breaking constraints [18]. However we leave this for future work.

These constraints are all found in Minion 0.8.1. The GCC constraint enforces GAC in this situation (with a fixed number of occurrences of each symbol). The reified not-equal constraint enforces Bounds($\mathbb{Z}$)-consistency [17]. The sum constraint above is decomposed into $\leq d$ and $\geq d$ constraints (sumleq and sumgeq in Minion) which also enforce Bounds($\mathbb{Z}$)-consistency. The lex ordering constraints enforce GAC.

The variable order is row-wise on c, in index order, as follows.

$$c[1,1], \dots, c[1,q\lambda], c[2,1], \dots, c[2,q\lambda], \dots$$

The values are searched in ascending order.

A Boolean Model. In this section we consider another simple model for EFPA, based on a different viewpoint to the one above. The difference is in the representation of each symbol in each codeword using a vector of Boolean variables. The model uses a three-dimensional matrix m of Boolean variables to represent occurrences of the q symbols in the v codewords. The first dimension is the codeword $1 \dots v$, the second is the symbol $1 \dots q$ and the third is the codeword position $1 \dots q\lambda$.

- $\forall i \in \{1 \dots v\}, \forall j \in \{1 \dots q\}, \forall k \in \{1 \dots q\lambda\} \ : \ m[i,j,k] \in \{0,1\}$

Variable $m[i,j,k]$ is 1 iff the codeword i has symbol j at position k.

We must ensure that exactly one symbol appears at each position in each codeword. This is done with the following set of constraints.

- $\forall i \in \{1...v\}, \forall j \in \{1...q\lambda\} \ : \ \sum_{k=1}^{q} m[i,k,j] = 1$

To ensure that there are λ of each symbol in each codeword we post the following set of constraints.

- $\forall i \in \{1...v\}, \forall j \in \{1...q\} \ : \ \sum_{k=1}^{q\lambda} m[i,j,k] = \lambda$

The final problem constraint set states that the Hamming distance between any pair of codewords is exactly d. The two codewords are represented as planes in the matrix. For each position where the pair of codewords differ, the planes in m differ in two places corresponding to one symbol being removed and another inserted. Therefore the number of places where the two planes differ is $2d$.

- $\forall i \in \{1...v\}, \forall j \in \{i+1...v\} \ : \ [\sum_{k=1}^{q\lambda} \sum_{l=1}^{q} (m[i,l,k] \neq m[j,l,k])] = 2d$

Symbols, codewords and positions may all be freely permuted. In order to break some of this symmetry, we lexicographically order (lex order) planes of the matrix in all three dimensions, using the technique of Flener et al. [16]. (We rely on Tailor to vectorize the planes in a consistent manner.) This set of symmetry breaking constraints orders the symbols, in contrast to those of the non-Boolean model.

- $\forall i \in \{1...q\lambda - 1\} \ : \ m[1 \ldots v, 1 \ldots q, i] \leq_{lex} m[1 \ldots v, 1 \ldots q, i+1]$
- $\forall i \in \{1...q-1\} \ : \ m[1 \ldots v, i, 1 \ldots q\lambda] \leq_{lex} m[1 \ldots v, i+1, 1 \ldots q\lambda]$
- $\forall i \in \{1 \ldots v-1\} \ : \ m[i, 1 \ldots q, 1 \ldots q\lambda] \leq_{lex} m[i+1, 1 \ldots q, 1 \ldots q\lambda]$

Preliminary experiments reveal that these three constraint sets drastically improve performance. In one instance the addition of symmetry breaking constraints improved the performance of the model by approximately 40 times. The variable ordering is as follows. For each i in ascending order: for each j in ascending order: for each k in ascending order: $m[i,j,k]$. To illustrate:

$$m[1,1,1], \ldots, m[1,1,q\lambda], m[1,2,1], \ldots, m[1,q,q\lambda], m[2,1,1], \ldots$$

For all variables, value 0 is branched on first.

Channelling Boolean and Non-Boolean Models. The Boolean model may have better symmetry breaking than the non-Boolean model, because the value symmetry of the non-Boolean model is transformed into variable symmetry [16] and broken using lex constraints. However, in the non-Boolean model, the first row is invariant because of the column lex constraints and therefore some of the value symmetry is broken there.

The non-Boolean model is able to exploit the GCC constraint on the rows, and also has a neater representation of the Hamming distance requirement. In this section we aim to gain the advantages of both models by connecting the two with channelling constraints, given below.

- $\forall i \in \{1 \ldots v\}, \forall j \in \{1 \ldots q\}, \forall k \in \{1 \ldots q\lambda\} \ : \ m[i,j,k] \Leftrightarrow (c[i,k] = j)$

The symmetry breaking constraints in the non-Boolean model are removed, because they contradict those in the Boolean model.

The Boolean model has three sets of constraints other than the symmetry breaking constraints. Inspection of each set suggests that they will provide no useful propagation, because the non-Boolean representation of the same constraint set is equivalent or stronger. Preliminary experimentation on instance $d = \lambda = q = 4$, $v = 9$ showed that removing all three sets does not affect the node count (2,350,155) but does reduce the time taken from 86 s to 52 s. Therefore we do not include the three sets of constraints.

Preliminary experiments suggest that searching on the c (non-Boolean) variables is not effective when channelling, using either an ascending or descending value ordering (with the variable ordering described for c above). Therefore we search on m, using the same variable and value ordering as the standard Boolean model. Given that the constraints on the non-Boolean formulation appear to be stronger, we expect that the channelled model will improve on the Boolean model in terms of search nodes.

Empirical Evaluation. To compare the three models empirically, we picked twelve tuples $\langle d, \lambda, q\rangle$ with a range of different values of d, λ and q. For each parameter set, the usual task is to find the maximal set of codewords. This can be done by solving iteratively, increasing v until the instance is unsatisfiable. This provides a maximal set of codewords, and a proof that there is no larger set. Typically the unsatisfiable instance is much more difficult than the others, because of the need to exhaust the search space. Instances are identified by the tuple $\langle d, \lambda, q, v\rangle$.

For each parameter set, we use two consecutive values of v such that the smaller instance is satisfiable and the larger one is unsatisfiable or it takes longer than the time limit of 2 hours to solve. This provides 12 satisfiable instances, 11 unsatisfiable instances and one $\langle 6, 4, 4, 14\rangle$ which is unknown[2]. We used Minion 0.8.1 on an Intel Xeon E5430 2.66 GHz 8-core machine, using all cores. The three models are named as follows.

Non-Boolean refers to the two-dimensional model.
Boolean refers to the three-dimensional model.
Channelled refers to the combined model described in the section above.

We enabled the SAC [19] preprocessing option of Minion to be consistent with our other experiments presented below. SAC preprocessing is cheap, taking less than 0.2 s on the largest instances.

Both the non-Boolean and Channelled models include some variables which are not necessary here, but are required for the implied constraints and permutation model described below[3]. For satisfiable instances, these spurious variables are set at the end of the search process. Assigning each variable takes 1 search node, and a very small amount

[2] The problem instances are available at
`http://minion.sourceforge.net/benchmarks.html`

[3] Several models were expressed in one Essence′ file for convenience. Essence′ allows constraints to be included or excluded as required, and we used this to generate the different models, however the set of variables remains the same.

of time. For unsatisfiable instances, these variables do not affect search or propagation in any way.

Figure 1 shows our results for the three models. Instance $\langle 6, 4, 4, 14\rangle$ times out for all three models, and is the only time-out for the channelled model. The non-Boolean model times out on five instances. The channelled model improves upon the Boolean model in both nodes and time, except for the very easy satisfiable instance $\langle 4, 4, 3, 7\rangle$. In this case, there are 327 spurious variables, and the channelled model explores 396 nodes. Therefore the spurious variables account for most of the nodes, and this instance should be disregarded for comparing Boolean and channelled models.

For the non-Boolean and channelled models, neither always dominates the other, either in nodes or time. The non-Boolean model is faster on 13 instances, and the channelled model on 10 instances. For the instance that timed out, we can observe that the channelled model explores fewer nodes than the other two. For this instance, the channelling has some overhead, as one would expect.

2.2 Extensions of the Non-Boolean Model

In this section we explore two extensions of the non-Boolean model, both of which exploit knowledge about the permutation of d elements between pairs of codewords.

Implied Constraints. It is possible to derive some implied constraints between pairs of sequences. Consider the sequence $\langle 1, 1, 1, 2, 2, 2, 3, 3, 3\rangle$, and assume that $d = 4$. To construct another sequence with the appropriate Hamming distance, we can swap two 1's with two 2's: $\langle 1, 2, 2, 1, 1, 2, 3, 3, 3\rangle$. However, it is not possible to move all three 1's, since that would cause six disagreements. In general, for each symbol, the maximum number which can be moved is $\lfloor \frac{d}{2} \rfloor$.

If $\lfloor \frac{d}{2} \rfloor < \lambda$, then this observation allows us to add useful constraints to the model. Between any pair of sequences in the set, and for each symbol a, at least $\lambda - \lfloor \frac{d}{2} \rfloor$ instances of a must remain in the same place. We do not exploit this observation for every pair of rows, but only for the pairs containing the first row. This is because the first row is fixed and this makes the statement of the constraints considerably simpler.

The first row is fixed by the combination of column lex ordering constraints and the cardinality constraint. If $q = \lambda = 3$ then the first row is $\langle 1, 1, 1, 2, 2, 2, 3, 3, 3\rangle$ in the non-Boolean model. It is arranged in q blocks of length λ, and each of the other sequences is divided into q blocks in the same way, as shown in the table below.

$c[1, 1 \dots q\lambda]$	1 1 1	2 2 2	3 3 3
$c[2, 1 \dots q\lambda]$	Block 1	Block 2	Block 3

To state the constraints, we have auxiliary variables $occ^a_{b,d} \in \{0 \dots \lambda\}$ representing the number of occurrences of value d in sequence a, block b (where the blocks are numbered $1 \dots q$ in index order). We post q GCC constraints to count the symbols in each block, as follows. Also, we constrain the occurrences of the relevant *occ* variables.

- $\forall a \in \{2 \dots v\}, \forall b \in \{1 \dots q\} \ : \ \text{GCC}(c^a_{(b-1)\lambda+1 \dots b\lambda}, \langle 1 \dots q\rangle, occ^a_{b,1 \dots q})$
- $\forall a \in \{2 \dots v\}, \forall b \in \{1 \dots q\} \ : \ occ^a_{b,b} \geq \lambda - \lfloor \frac{d}{2} \rfloor$

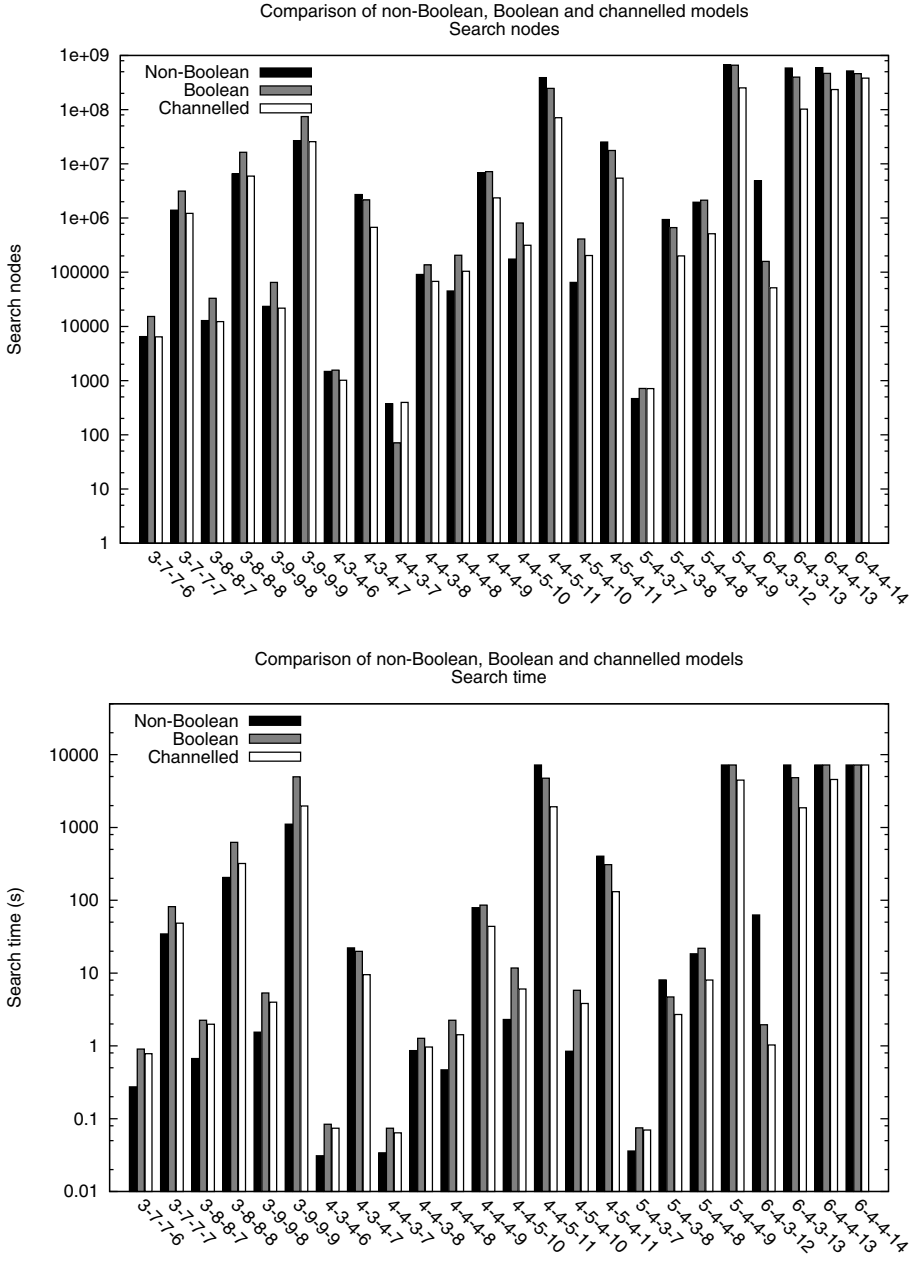

Fig. 1. Comparison of non-Boolean, Boolean and channelled models

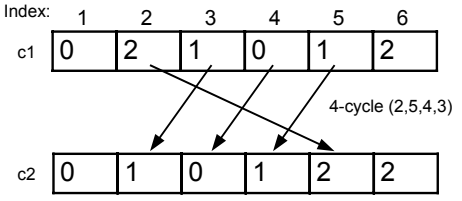

Fig. 2. The action of permutation (2,5,4,3) on codeword c_1 to form c_2

To improve the propagation of these constraints, we also state that for each symbol, the occurrence variables for all blocks in a sequence must sum to λ.

– $\forall a \in \{2 \dots v\}, \forall b \in \{1 \dots q\} \ : \ \sum occ^{a}_{1 \dots q,b} = \lambda$

The GCC constraint (with variables as its third argument) performs a hybrid consistency. It reads the bounds of the occurrence variables, and performs GAC over the target variables only, using Régin's algorithm [10]. Also, it uses a simple counting algorithm to prune the occurrence variables. For example, if two target variables are assigned to 0, and five variables have 0 in their domain, then the lower bound for occurrences of 0 is two, and the upper bound is five. This implementation of GCC is named gccweak in Minion.

Note that this constraint set cannot be applied to the channelled model, because it relies on the symmetry breaking constraints of the non-Boolean model. In the channelling model, the first row is in descending order.

Modelling Permutations. Between any pair of codewords c_1 and c_2, there are d indices where they differ. Since the two codewords have the same multiset of symbols, the difference between them can be represented as a permutation. In order to have d points of disagreement, d symbols in the first codeword must be moved to a different position in the second codeword. We consider permutations of the indices $1 \dots q\lambda$, specifying which indices of the first codeword are moved to form the second. Any permutation can be represented in *cycle notation* — for example, the cycle $(1, 3, 2)$ moves the symbol at index 1 to index 3, 3 to 2 and 2 to 1. We do not allow the permutation to permute index i to j if $c_1[i] = c_1[j]$, since this would leave c_1 and c_2 the same at position j.

Figure 2 shows an example of a permutation in cycle form acting on a codeword to create another codeword with Hamming distance $d = 4$.

From any sequence to any other, there is a permutation of d indices. The implied constraints in the section above make some use of this fact between the first sequence and all others. In this section we consider all pairs of sequences, and we explicitly model the cycle notation, using an array of d variables containing indices into the sequences. A further variable *cform* represents the form of the cycle notation. For example, when $d = 4$, there are two possible forms of the cycle notation: (p, q, r, s) and $(p, q)(r, s)$, therefore *cform* has two values. In fact we have only implemented the permutation model for $d = 4$. The subclass of EFPA where $d = 4$ is of interest to mathematicians as it is the smallest value of d where the precise upper bound for the size of an EFPA is not obvious (due to the fact that there is more than one possible cycle structure for a derangement of 4 points).

Table 2. Example of p

		1	2	3	4
1	$(1,3)(4,6)$	$s_1[3]=2$	$s_1[1]=1$	$s_1[6]=3$	$s_1[4]=2$
2	$(1,3,4,6)$	$s_1[6]=3$	$s_1[1]=1$	$s_1[3]=2$	$s_1[4]=2$

- $\forall e \in \{1 \dots d\} \ : \ \mathit{perm}[e] \in \{1 \dots q\lambda\}$
- $\mathit{cform} \in \{1 \dots \mathit{cforms}\}$ where *cforms* is the number of cycle forms.

To allow us to map from one sequence to another using the permutation, we introduce a table of variables p. There are *cforms* rows and d columns in p. The rows correspond to different forms of the cycle notation. Each row contains elements of the first sequence (those elements indexed by *perm*) permuted according to the form of the cycle notation.

- $\forall i \in \{1 \dots \mathit{cforms}\}, \forall j \in \{1 \dots d\} \ : \ p[i,j] \in \{1 \dots q\}$

For example, if the first sequence is $s_1 = \langle 1,1,2,2,3,3 \rangle$, $d = 4$ and *perm* is $\langle 1,3,4,6 \rangle$, then p is given in Table 2. In the first row, indices for each pair are swapped, and in the second row the indices are rotated according to the inverse of the 4-cycle.

For the second sequence s_2, positions 1,3,4 and 6 (i.e. the values of *perm*) must equal the appropriate value from p. *cform* is used to select the appropriate row in p. For position 1, $s_2[1] = p[\mathit{cform}, 1]$. Also, constraints are posted stating that s_1 and s_2 are equal at all positions not in *perm*. In this example, if $\mathit{cform} = 1$ then $s_2 = \langle 2,1,1,3,3,2 \rangle$, and if $\mathit{cform} = 2$ then $s_2 = \langle 3,1,1,2,3,2 \rangle$.

The basic set of constraints is given below.

- $\forall i \in \{1 \dots \mathit{cforms}\}, \forall j \in \{1 \dots d\} : p[i,j] = s_1[k]$ where k is the inverse mapping of j by the permutation (*perm* with cycle form i).
- $\forall i \in \{1 \dots d\} \ : \ s_2[\mathit{perm}[i]] = p[\mathit{cform}, i]$
- $\forall i \in \{1 \dots q\lambda\} \ : \ (\bigwedge_{j=1}^{d} \mathit{perm}[j] \neq i) \Leftrightarrow (s_1[i] = s_2[i])$ (If index i is not present in *perm*, then the value at position i remains the same, and vice versa)

We add symmetry breaking constraints to *perm*. In general, within each cycle, the smallest element is placed at the front, and cycles of equal length are ordered by their first element. The ordering constraints are shown below for $d = 4$. Only one of the constraints is conditional on *cform*; the other three are true for either cycle form.

- $(\mathit{cform} = 1) \Rightarrow (\mathit{perm}[3] < \mathit{perm}[4])$
- $(\mathit{perm}[1] < \mathit{perm}[2]) \wedge (\mathit{perm}[1] < \mathit{perm}[3]) \wedge (\mathit{perm}[1] < \mathit{perm}[4])$

As a special case for $d = 4$, we add an allDifferent to the values permuted by the 4-cycle form. The reason is that if the values are not all different, the 4-cycle can be reformulated as two 2-cycles.

- $(\mathit{cform} = 2) \Rightarrow \text{allDifferent}(p[2, 1 \dots d])$

The expression above is decomposed as a reified allDifferent and an implies constraint on two Boolean variables. GAC is enforced on both these constraints, which is equivalent to GAC on the original expression.

When $cform = 1$, the transposed values must be different, therefore we add the following constraints. These constraints are also true when $cform = 2$, so there is no need for them to be conditional.

- $p[1,1] \neq p[1,2] \wedge p[1,3] \neq p[1,4]$

Empirical Evaluation. In this section we compare the following four models, using the same set of instances and experimental details as in the previous experiment.

Non-Boolean refers to the two-dimensional model.
Implied is the non-Boolean model with additional constraints described in the section with heading Implied Constraints.
Permutation is the non-Boolean model with additional permutation constraints described in the Modelling Permutations section.
Implied+Perm is the non-Boolean model with both Implied and Permutation constraint sets.

All the above models have the same set of variables, including all variables used by the implied constraints and the permutation model. For solvable instances, the unused variables are enumerated at the end of the search process, adding a small constant to the node count. The permutation models are only used where $d = 4$, since they are not defined for other values.

We found it important to perform singleton consistency (SAC) [19] preprocessing. It is a very cheap preprocessing step which is nevertheless very important for the permutation model. Table 3 shows that SAC preprocessing is important on the instance where $d = \lambda = q = 4$ and $v = 9$, for two of the four models listed above. For both models that include the permutation constraint set, the preprocessing is vital. Therefore we use SAC preprocessing throughout.

Figure 3 shows our results for these models. It is clear that both the implied constraints and the permutation model are effective in reducing the number of search nodes, and in most cases the combined model gives a further reduction. The permutation model is particularly effective. For example, on the instance $\langle 4, 5, 4, 11\rangle$ the non-Boolean model takes 25,271,680 nodes, the implied model takes 14,607,410 nodes and the permutation model takes 6,032,900 nodes, a reduction of 76%. Where $d = 4$, there is a clear ordering among the four models.

Table 3. Search nodes with and without SAC preprocessing, $d = \lambda = q = 4$, $v = 9$

Search nodes	Non-Boolean	Implied	Permutation	Implied+Perm
No preprocessing	6,788,586	4,032,510	6,021,363	3,471,775
SAC preprocessing	6,788,361	4,032,306	1,674,826	1,340,546

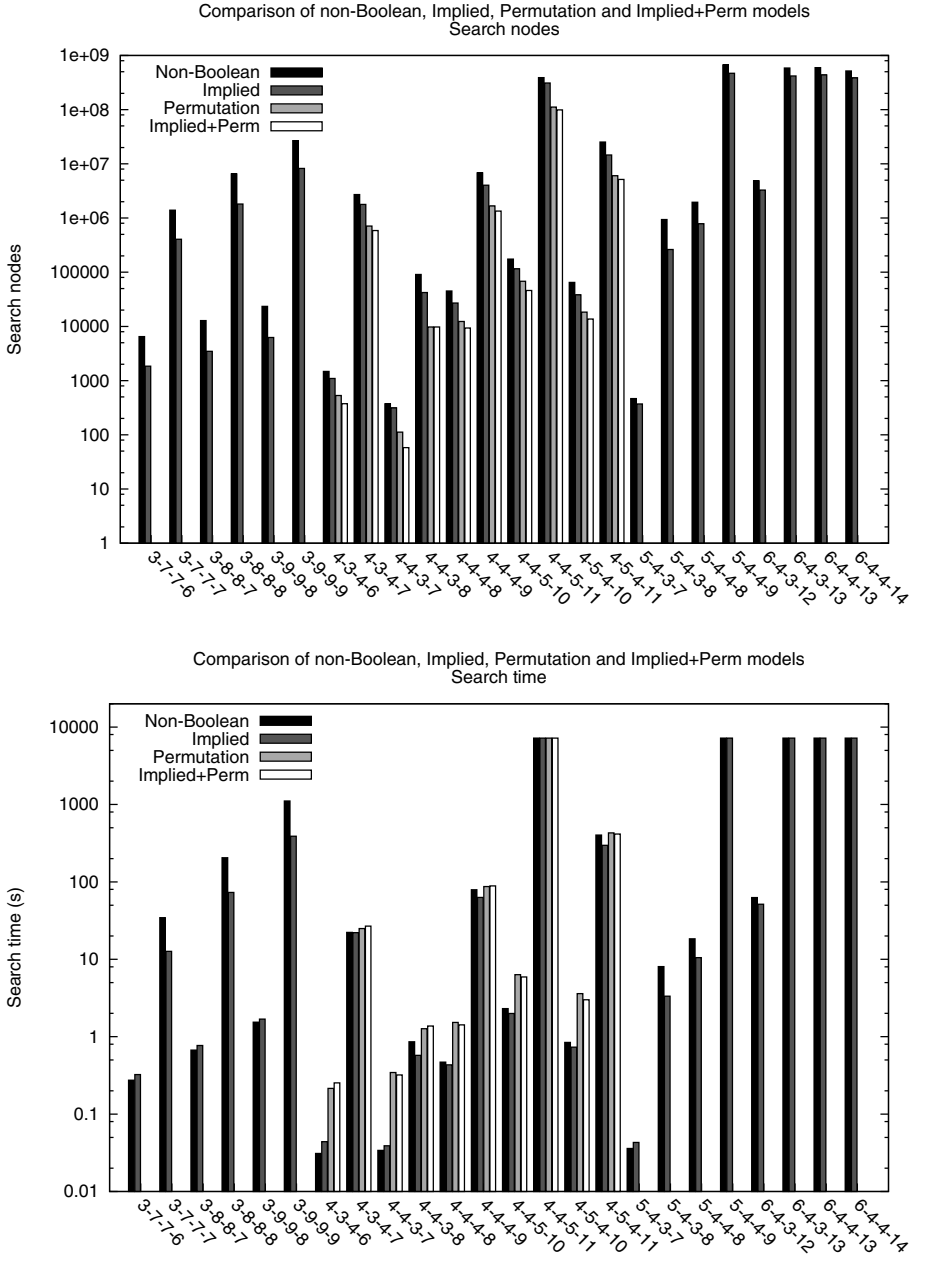

Fig. 3. Comparison of non-Boolean, Implied, Permutation and Implied+Perm models

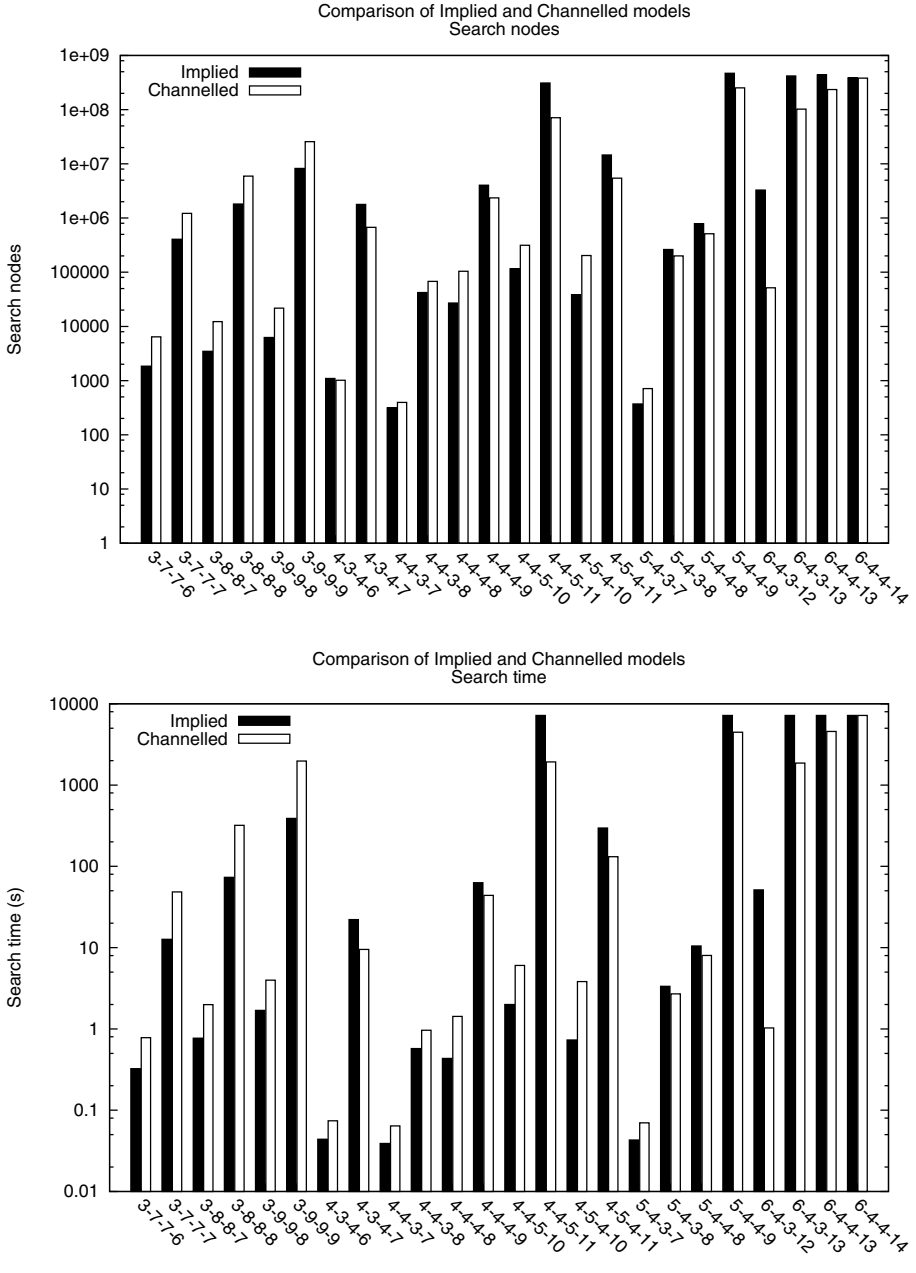

Fig. 4. Comparison of Implied and Channelled models

However, the search times are not so straightforward. In most cases, the implied constraints are worthwhile. However, the permutation model is never worthwhile, and likewise for the Implied+Perm model. It is clear that the permutation constraint set adds a considerable overhead to the search process, and therefore it takes longer to solve even though it is exploring many fewer nodes.

Finally, we compare the Implied model with the Channelled model from the previous experiment. These models use a different variable and value ordering as well as a different constraint set, so there is no reason to expect one to always dominate the other in terms of search nodes. The Implied model is the most efficient when searching on c variables, and likewise the channelled model is the most efficient when searching on m, therefore this makes an interesting final comparison. The data are plotted in Figure 4. Recall that the implied constraint set cannot be added to the channelling model, because it is incompatible with the symmetry breaking constraints.

The two models behave remarkably similarly, given their considerable differences. For the instance which times out, it can be seen that the two models explored a similar number of nodes, indicating a similar node rate. The implied model was faster for 13 instances, and the channelled model was faster for 10. However, the implied model timed out on five instances, and the channelled model timed out on one.

3 Conclusions

We have modelled the equidistant frequency permutation array problem using constraint programming, investigating a range of models. We devised a channelled model which exploits symmetry breaking constraints on one viewpoint and problem constraints on the other viewpoint. We invented a set of implied constraints and showed their benefit in most cases. This set of constraints may generalise to other problems involving fixed Hamming distance. As a potential item of future work for EFPA, the implied constraints could be reformulated to be compatible with the channelled model.

We gave a novel representation of cycle notation, modelling a permutation with a fixed number of move points. This was shown to be very effective for cutting down the search space, which indicates its potential. However, the overhead of the additional constraints negated the benefits. With a different formulation or a different constraint solver, the permutation model could prove to be beneficial. Also, it may apply to other problems involving fixed Hamming distance. It would be interesting to investigate this further.

Our work has direct application in mathematics. One of the authors (Huczynska) is a mathematician and is exploiting our novel results in her own theoretical investigations [4]. This illustrates that, with careful modelling and the power of fast constraint solvers such as Minion and modelling assistants such as Tailor, constraint programming can contribute to research in other disciplines.

References

1. Huczynska, S., Mullen, G.: Frequency permutation arrays. J. Combin. Des. 14, 463–478 (2006)
2. Chu, W., Colbourn, C., Dukes, P.: Constructions for permutation codes in powerline communications. Designs, Codes and Cryptography 32, 51–64 (2004)

3. Chu, W., Colbourn, C., Dukes, P.: On constant composition codes. Discrete Applied Math. 154, 912–929 (2006)
4. Huczynska, S.: Equidistant frequency permutation arrays and related constant composition codes. Designs, Codes and Cryptography (to appear, 2009)
5. Ding, C., Yin, J.: A construction of optimal constant composition codes. Designs, Codes and Cryptography 40, 157–165 (2006)
6. Slaney, J., Fujita, M., Stickel, M.: Automated reasoning and exhaustive search: Quasigroup existence problems. Computers and Mathematics with Applications 29, 115–132 (1995)
7. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast, scalable, constraint solver. In: Proceedings 17th European Conference on Artificial Intelligence (ECAI 2006), pp. 98–102 (2006)
8. Gent, I.P., Jefferson, C., Miguel, I.: Watched literals for constraint propagation in minion. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 182–197. Springer, Heidelberg (2006)
9. Gent, I.P., Miguel, I., Rendl, A.: Tailoring solver-independent constraint models: A case study with Essence′ and Minion. In: Miguel, I., Ruml, W. (eds.) SARA 2007. LNCS (LNAI), vol. 4612, pp. 184–199. Springer, Heidelberg (2007)
10. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: Proceedings 13th National Conference on Artificial Intelligence (AAAI 1996), pp. 209–215 (1996)
11. Frisch, A.M., Jefferson, C., Miguel, I.: Symmetry breaking as a prelude to implied constraints: A constraint modelling pattern. In: Proceedings ECAI 2004, pp. 171–175 (2004)
12. Hnich, B., Smith, B.M., Walsh, T.: Dual modelling of permutation and injection problems. JAIR 21, 357–391 (2004)
13. Velev, M., Gao, P.: Efficient SAT techniques for absolute encoding of permutation problems: Application to hamiltonian cycles. In: Proceedings SARA 2009 (to appear, 2009)
14. Rendl, A., Miguel, I., Gent, I.P., Jefferson, C.: Automatically enhancing constraint model instances during tailoring. In: Proceedings of the Eighth International Symposium on Abstraction, Reformulation and Approximation, SARA 2009 (to appear, 2009)
15. Jefferson, C.: Representations in Constraint Programming. PhD thesis, Computer Science Department, York University, UK (2007)
16. Flener, P., Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetries in matrix models. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 462–476. Springer, Heidelberg (2002)
17. Choi, C.W., Harvey, W., Lee, J.H.M., Stuckey, P.J.: Finite domain bounds consistency revisited. In: Sattar, A., Kang, B.-h. (eds.) AI 2006. LNCS (LNAI), vol. 4304, pp. 49–58. Springer, Heidelberg (2006)
18. Walsh, T.: Breaking value symmetry. In: Proceedings of AAAI 2008 (2008)
19. Debruyne, R., Bessière, C.: Some practicable filtering techniques for the constraint satisfaction problem. In: Proceedings 15th International Joint Conference on Artificial Intelligence (IJCAI 1997), pp. 412–417 (1997)