Perfect Hash Families in Polynomial Time

Charles J. Colbourn[1]

[1] School of Computing, Informatics, and Decision Systems Engineering
Arizona State University

October 2010

Perfect Hash Families

Definition

- A *perfect hash family* $\mathrm{PHF}(N; k, v, t)$ is an $N \times k$ array on v symbols, in which in every $N \times t$ subarray, at least one row consists of distinct symbols.
- The smallest N for which a $\mathrm{PHF}(N; k, v, t)$ exists is the *perfect hash family number*, denoted $\mathrm{PHFN}(k, v, t)$.

Perfect Hash Families

Example PHF$(6; 12, 3, 3)$

$$\begin{bmatrix} 0 & 1 & 2 & 2 & 1 & 2 & 2 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 2 & 2 & 2 & 1 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 2 & 2 & 2 & 1 & 1 & 2 & 1 & 0 & 2 \\ 2 & 0 & 1 & 1 & 2 & 0 & 2 & 0 & 1 & 1 & 2 & 1 \\ 2 & 0 & 2 & 1 & 2 & 1 & 0 & 2 & 2 & 1 & 1 & 0 \\ 2 & 0 & 1 & 2 & 1 & 1 & 2 & 2 & 0 & 1 & 2 & 1 \end{bmatrix}$$

Perfect Hash Families

Example PHF(6; 12, 3, 3)

$$
\begin{array}{c}
\begin{array}{cccccccccccc} & & & & & & & \downarrow & \downarrow & \downarrow & & \end{array} \\
\begin{bmatrix}
0 & 1 & 2 & 2 & 1 & 2 & 2 & 0 & 1 & 1 & 0 & 0 \\
0 & 2 & 1 & 0 & 2 & 2 & 2 & 1 & 0 & 1 & 2 & 1 \\
1 & 0 & 0 & 2 & 2 & 2 & 1 & 1 & 2 & 1 & 0 & 2 \\
2 & 0 & 1 & 1 & 2 & 0 & 2 & 0 & 1 & 1 & 2 & 1 \\
2 & 0 & 2 & 1 & 2 & 1 & 0 & 2 & 2 & 1 & 1 & 0 \\
2 & 0 & 1 & 2 & 1 & 1 & 2 & 2 & 0 & 1 & 2 & 1
\end{bmatrix}
\end{array}
$$

Perfect Hash Families

- It is "well known" that, for fixed v and t, PHFN(k, v, t) grows like $\log k$ (see Mehlhorn 82, Fredman-Komlos 84, Blackburn-Wild 98, for example).
- But constructing specific PHFs remains challenging!
- Why am I (and why should you be) interested?

Covering Array. Definition

- Let N, k, t, and v be positive integers.
- Let C be an $N \times k$ array with entries from an alphabet Σ of size v; we typically take $\Sigma = \{0, \ldots, v-1\}$.
- When $(\nu_1, \ldots, \nu_t)$ is a t-tuple with $\nu_i \in \Sigma$ for $1 \leq i \leq t$, $(c_1, \ldots, c_t)$ is a tuple of t column indices ($c_i \in \{1, \ldots, k\}$), and $c_i \neq c_j$ whenever $\nu_i \neq \nu_j$, the t-tuple $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ is a *t-way interaction*.
- The array *covers* the t-way interaction $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ if, in at least one row ρ of C, the entry in row ρ and column c_i is ν_i for $1 \leq i \leq t$.
- Array C is a *covering array* CA$(N; t, k, v)$ of *strength t* when every t-way interaction is covered.

Covering Array. Definition

- Let N, k, t, and v be positive integers.
- Let C be an $N \times k$ array with entries from an alphabet Σ of size v; we typically take $\Sigma = \{0, \ldots, v-1\}$.
- When $(\nu_1, \ldots, \nu_t)$ is a t-tuple with $\nu_i \in \Sigma$ for $1 \leq i \leq t$, $(c_1, \ldots, c_t)$ is a tuple of t column indices $(c_i \in \{1, \ldots, k\})$, and $c_i \neq c_j$ whenever $\nu_i \neq \nu_j$, the t-tuple $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ is a *t-way interaction*.
- The array *covers* the t-way interaction $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ if, in at least one row ρ of C, the entry in row ρ and column c_i is ν_i for $1 \leq i \leq t$.
- Array C is a *covering array* CA$(N; t, k, v)$ of *strength t* when every t-way interaction is covered.

Covering Array. Definition

- Let N, k, t, and v be positive integers.
- Let C be an $N \times k$ array with entries from an alphabet Σ of size v; we typically take $\Sigma = \{0, \ldots, v-1\}$.
- When $(\nu_1, \ldots, \nu_t)$ is a t-tuple with $\nu_i \in \Sigma$ for $1 \leq i \leq t$, $(c_1, \ldots, c_t)$ is a tuple of t column indices $(c_i \in \{1, \ldots, k\})$, and $c_i \neq c_j$ whenever $\nu_i \neq \nu_j$, the t-tuple $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ is a *t-way interaction*.
- The array *covers* the t-way interaction $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ if, in at least one row ρ of C, the entry in row ρ and column c_i is ν_i for $1 \leq i \leq t$.
- Array C is a *covering array* CA$(N; t, k, v)$ of *strength t* when every t-way interaction is covered.

Covering Array

CA(13;3,10,2)

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Covering Array

Motivation *Software interaction testing*

- Construct a large software system by combining software, hardware, and network components each intended to perform some simple function.
- Even when each component operates ‘correctly’, *interactions* among selections for components may cause faults.
- Columns are components or *factors*; selections of particular components are *levels* for the factors.
- Rows are *tests* or *runs*.
- Every t-way interaction is tested in at least one run!
- The *sparsity of effects*...

Perfect Hash Families

The First Connection

Theorem

If a PHF$(s; k, m, t)$ *and a* CA$(N; t, m, v)$ *both exist then a* CA$(sN; t, k, v)$ *exists.*

- B $= (b_{ij})$ is an $s \times k$ array on m symbols forming a PHF$(s; k, m, t)$.
- A $= (a_{ij})$ is an $N \times m$ array on v symbols forming a CA$(N; t, m, v)$.
- Produce an $sN \times k$ array C $= (c_{ij})$ as follows. For each $1 \leq i \leq s$, $1 \leq j \leq N$, and $1 \leq \ell \leq k$, set $c_{(i-1)N+j,\ell} = a_{j,b_{i,\ell}}$.

Perfect Hash Families
Methods

- A number of methods construct PHFs:
  - Direct methods: From codes, orthogonal arrays, finite geometries, modular sequences of integers, no three in arithmetic progression, algebraic curves.
  - Recursive methods: "Cut-and-paste", column replacement techniques.
  - Probabilistic methods: Select an array at random, and if there are enough rows, it "works" with high probability.
  - Computational methods: Random, greedy, local optimization, or metaheuristic search such as simulated annealing, tabu search, genetic algorithms, ...

Perfect Hash Families

Methods

- A number of methods construct PHFs:
  - Direct methods: From codes, orthogonal arrays, finite geometries, modular sequences of integers, no three in arithmetic progression, algebraic curves.
  - Recursive methods: “Cut-and-paste”, column replacement techniques.
  - Probabilistic methods: Select an array at random, and if there are enough rows, it “works” with high probability.
  - Computational methods: Random, greedy, local optimization, or metaheuristic search such as simulated annealing, tabu search, genetic algorithms, ...

Perfect Hash Families

Methods and Limitations

- But there remains a big problem...
  - Direct methods appear to apply for a very limited set of parameters.
  - Recursive methods require very good 'small' ingredients, and appear to work well only when the strength is 'small'.
  - Probabilistic methods ensure the existence of the PHF but do not typically give us the actual array.
  - Computational methods, when sophisticated, do not seem fast enough; and when naive, do not seem to yield results competitive with the direct techniques.
- We need to construct PHFs explicitly.

Perfect Hash Families

Methods and Limitations

- But there remains a big problem...
  - Direct methods appear to apply for a very limited set of parameters.
  - Recursive methods require very good ‘small’ ingredients, and appear to work well only when the strength is ‘small’.
  - Probabilistic methods ensure the existence of the PHF but do not typically give us the actual array.
  - Computational methods, when sophisticated, do not seem fast enough; and when naive, do not seem to yield results competitive with the direct techniques.
- We need to construct PHFs explicitly.

Perfect Hash Families

Methods and Limitations

- But there remains a big problem...
  - Direct methods appear to apply for a very limited set of parameters.
  - Recursive methods require very good 'small' ingredients, and appear to work well only when the strength is 'small'.
  - Probabilistic methods ensure the existence of the PHF but do not typically give us the actual array.
  - Computational methods, when sophisticated, do not seem fast enough; and when naive, do not seem to yield results competitive with the direct techniques.
- We need to construct PHFs explicitly.

A Random Method

- Choose an array from $\{1, \ldots, v\}^{N \times k}$ uniformly at random.
- For any set of t columns, the probability that it is not separated is $\left(1 - \frac{\prod_{i=1}^{t} v+1-i}{v^t}\right)^N$.
- So the expected number of sets of t columns not separated is $\binom{k}{t} \left(1 - \frac{\prod_{i=1}^{t} v+1-i}{v^t}\right)^N$.
- When this expected number is less than 1, some array has all sets of t columns separated!

A Random Method

- Fix t independent of n.
- Take logarithms of $\binom{k}{t}\left(1 - \frac{\prod_{i=1}^{t} v+1-i}{v^t}\right)^N < 1$ to get

$$N > ct \log k$$

for a constant c depending only on t and v.

- This shows us the right growth rate for PHFN(k, v, t).

Derandomizing

The Stein-Lovász Method

- Instead of generating the array at random $\{1, \ldots, v\}^{N \times k}$, generate one row at a time at random from $\{1, \ldots, v\}^k$.
- After ρ rows have been generated, keep track of the number of sets of t columns separated so far.
- For an as-yet-unseparated set of columns, what is the probability that the next row chosen separates it?
- Because the row is selected at random, this is just $\frac{\prod_{i=1}^{t} v+1-i}{v^t}$.

Derandomizing

The Stein-Lovász Method

- Now count in two ways all possible ways to choose a row and a t-set of columns separated by that row for the first time. Suppose that the number of t-sets not yet separated is σ.
- First, if the expected number of t-sets separated by a row is ψ then the number of (row,separated column) pairs is ψv^k.
- Secondly, for any specific t-set T that is not yet separated, the number of rows separating it is $v^{k-t} \prod_{i=1}^{t} v + 1 - i$, so the number of (row,separated column) pairs is $\sigma v^{k-t} \prod_{i=1}^{t} v + 1 - i$.
- So $\psi = \frac{\prod_{i=1}^{t} v+1-i}{v^t} \sigma$.

Derandomizing

The Stein-Lovász Method

- So in every row, if σ t-sets of columns were not yet separated before the row, the expected number still not separated after the row is

$$\sigma - \frac{\prod_{i=1}^{t} v+1-i}{v^t}\sigma = \frac{v^t - \prod_{i=1}^{t} v+1-i}{v^t}\sigma.$$

- To derandomize, choose the row that separates the largest number of previously unseparated t-sets.
- Let σ_i be the number of as-yet-unseparated t-sets after i rows are selected. Then $\sigma_0 = \binom{k}{t}$ and $\sigma_{i+1} \leq \frac{v^t - \prod_{i=1}^{t} v+1-i}{v^t}\sigma_i$ for $i > 0$.

Derandomizing

The Stein-Lovász Method

- So $\sigma_m \leq \binom{k}{t} \left(\frac{v^t - \prod_{i=1}^{t} v+1-i}{v^t} \right)^m$.
- Solve for m in $\sigma_m < 1$.
- But how can we choose the 'best' row at each stage?

Derandomizing

Hypergraph Colouring

- At any stage the set of t-sets remaining to distinguish forms a t-uniform hypergraph on k vertices.
- When all remaining t-sets are to be separated by the next row, the row must form a colouring of the k vertices in v colours.
- Every t-set must be polychromatic ('rainbow'), receiving t different colours.
- The *strong chromatic number* is the minimum number of colours in such a strong colouring of the hypergraph.

Derandomizing
Hypergraph Colouring

- But computing the strong chromatic number is NP-hard in general...
- So although we have found a natural greedy method, its running time remains exponential in k.

Derandomizing

Average is Good Enough

- A simple but key observation... *Our analysis did not depend on picking the best row, just on picking one at least as good as the average!*
- But can we choose a row that is at least average? Evidently we can compute the average, and we can compute the number of newly separated t-sets for any specific candidate row, so given one we could certify that it is at least average (or that it is not).
- Generate candidate rows at random? But then we have reintroduced randomness to the method.

Density

- A *partial row* R is a vector in $(\{1, \ldots, v\} \cup \{\star\})^k$. Think of $\star$ as meaning 'not yet determined'.
- We can ask: If we fill in the $\star$ entries in R randomly, what is the expected number of t-sets newly separated? Call this the *density* for R.
- When R and R' are partial rows, write $R \to R'$ when R' is obtained from R by changing one $\star$ to a value from $\{1, \ldots, v\}$.
- A *fill sequence* is a collection $R_k, \ldots, R_0$ of partial rows where R_i contains exactly i $\star$ entries and $R_i \to R_{i-1}$ for $1 \leq i \leq k$.

Density

- Consider a fill sequence $R_k \to R_{i-1} \to \cdots \to R_0$.
  - If the density of R_{i-1} is at least that of R_i for $1 \leq i \leq k$, then because
  - the density of R_k is $\frac{\prod_{i=1}^{t} v+1-i}{v^t}\sigma$, which is exactly the average number of previously unseparated t-sets separated by a random row, then
  - the density of R_0 is at least the average number of previously unseparated t-sets separated by a random row —
  - but R_0 has no $\star$ entries, and hence its density is the *actual* number of previously unseparated t-sets separated by this row.

Density

- So all we need to do is find a way to get R_{i-1} from R_i so that the density does not decrease, and to do this efficiently.
- Consider R_i. Let the indices of the $\star$ entries be *free* and the remainder *fixed*.
- Choose one free index. There are v ways to change the $\star$ here to an entry.
- For each of the $\binom{k-1}{t-1}$ ways to select $t-1$ other indices, consider the t-set containing those $t-1$ together with the chosen free index.

Density

- For each way to choose a symbol to place in the free index, determine the expectation that the t-set is separated for the first time, conditioned on fixing the chosen symbol in the free index.
- Then for every choice of symbol s of the free index, form the sum δ_s of these conditional expectations over all $\binom{k-1}{t-1}$ ways to select $t-1$ other indices.
- Select a symbol $\overline{s}$ whose sum is at least the average!
- (Indeed if we carry out the same computation of the sum $\delta_\star$ of conditional expectations by placing a $\star$ again in the free index, the change in density from R_i to R_{i-1} is $\delta_{\overline{s}} - \delta_\star$, but $\delta_\star = \frac{1}{v}\sum_{i=1}^{v} \delta_i$.)

Density

- When t is fixed, the effort to make a new row that is at least as good as average is polynomial in k.
- But beware: t is in the exponent, so for practical reasons t had better be small, not just 'fixed'.

Density

- This method is greedy in its selection of rows, and greedy in its selection of symbols within a row.
- Its efficiency results from backing off from requiring a best row, and settling for an average one.
- We did not do this to get a method that was intended to be practical, but here comes the surprise.

Density in Practice

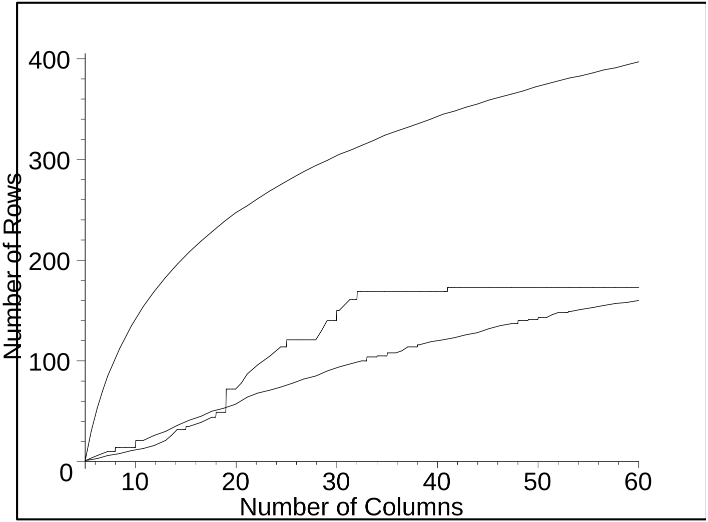

Figure: PHFN(k, 5, 5)

Density in Practice

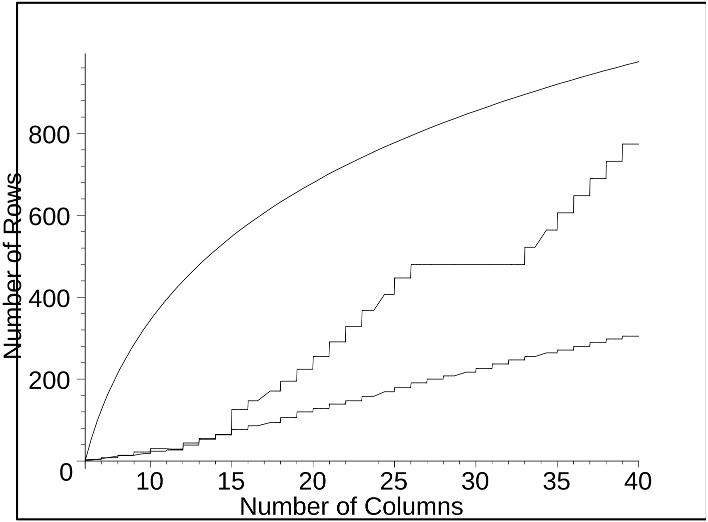

Figure: PHFN(k, 6, 6)

Density in Practice

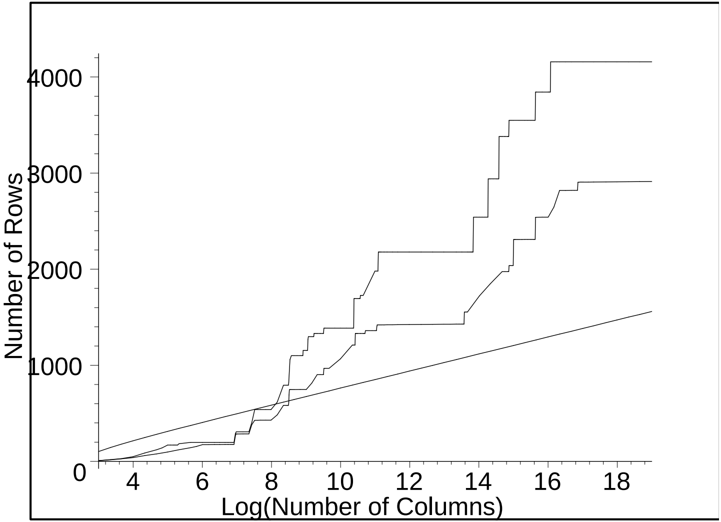

Figure: PHFN$(k, 5, 5)$

Conclusion

- Derandomizing a greedy randomized algorithm leads to an efficient deterministic algorithm for generating PHFs, and
- perhaps more surprisingly, this gives the best current general method for making them!