Multi-Index Hashing for Information Retrieval*

Dan Greene[†] Michal Parnas[‡] Frances Yao[§]

Abstract

We describe a technique for building hash indices for a large dictionary of strings. This technique permits robust retrieval of strings from the dictionary even when the query pattern has a significant number of errors. This technique is closely related to the classical Turan problem for hypergraphs. We propose a general method of multi-index construction by generalizing certain Turan hypergraphs. We also develop an accompanying theory for analyzing such hashing schemes. The resulting algorithms have been implemented and can be applied to a wide variety of recognition and retrieval problems.

Keywords: Information retrieval, hashing, hypergraphs, error-correction.

*A Preliminary version of this paper appeared in the 35'th Annual Symposium on Foundations of Computer Science, pp. 722-731, 1994.

[†]Xerox, Palo Alto Research Center, 3333 Coyote Hill Road, Palo-Alto, CA 94304, U.S.A. greene@parc.xerox.com, Tel. (650)-812-4476, Fax. (650)-812-4471.

[‡]The Academic College of Tel-Aviv-Yaffo, 4 Antokolsky st., Tel-Aviv 64044, Israel. michalp@server.mta.ac.il, Tel. 972-3-5211864, Fax. 972-3-5211871.

[§]Xerox, Palo Alto Research Center, 3333 Coyote Hill Road, Palo-Alto, CA 94304, U.S.A. ffyao@ibm.net, Tel. (650)-812-4486, Fax. (650)-812-4471.

1 Introduction

One of the classical open problems in the area of data structures is the *Best Match* problem raised by Minsky and Papert [10]. Let D be a dictionary of words of length ℓ, over some finite alphabet Σ, say $\Sigma = \{0, 1\}$. We would like to store D in such a way, that given a query string q, it is possible to find a member $s \in D$, which is closest to q in Hamming distance among all words in D. Minsky and Papert observed that there are no obvious designs which can perform significantly better than an exhaustive search, and asked whether such designs do exist. (See also Rivest [13], Sankoff and Kruskal [14], and Knuth [7], Section 6.5, for related work.)

The AQ Problem. This question has received renewed attention recently (see Dolev, Harari and Parnas [3], and Dolev et al. [2]), where it was referred to as the *Approximate Query* (AQ) problem. The AQ formulation specifies a value d, and asks for all words in D within Hamming distance at most d from the query to be returned. We also say, informally, that an algorithm for the AQ problem *corrects* d errors. The objective of an algorithm is to minimize storage space and query time, while correcting a given number, d, of errors.

The AQ problem has many applications. Spell-checkers, speech and handwriting recognizers are a few examples in which the word length, and therefore the number d of errors in a query, are relatively small. However, in some other applications, such as the analysis of DNA sequences, computer viruses detection, or document recognition, a large number of errors may occur. In such cases it is important to have an algorithm that runs efficiently for large d.

When the dictionary D is small, the AQ problem may be adequately solved by using, for example, the string-matching algorithms of Landau and Vishkin [8], [9] which find all matches with at most d differences. However, the $O(|D|)$ running time of these algorithms makes them unsuitable for large dictionaries.

Corrector Hypergraphs. In this paper, we propose a new class of algorithms for the AQ problem. Our algorithms are based on a *multi-index hashing* approach, where each word and query are hashed m times based on m suitably chosen substrings. Both the space and the query answer time complexity of such an algorithm, depend directly on the number, m, of indices used per word/query. However, the query time also depends on the number of hash collisions per query. Our algorithms try to minimize both m and the number of hash collisions.

We propose a general class of constructions, called *corrector hypergraphs*. These constructions are closely related to the classical Turan problem for hypergraphs, and generalize the known optimal Turan graphs, and certain hypergraphs that are conjectured to be optimal Turan graphs. They can be used to construct efficient *AQ* algorithms, which correct d errors by using $m = O(d2^{dI/\ell})$ indices of size I, where ℓ is the length of the words in D. If the dictionary is uniformly distributed and $I = O(\log |D|)$, then the number of indices per word/query is $m = O(d|D|^{d/\ell})$, and the expected hash retrieval time is constant.

Our algorithms have been implemented and tested successfully on large dictionaries of random strings. We expect these algorithms to be useful for a wide variety of recognition and retrieval problems.

The corrector hypergraphs presented have a rich combinatorial structure and are of independent interest. We compare the correcting capabilities of families of corrector hypergraphs, which have the same number, m, of edges. Our results indicate, that there is a hierarchy between the different corrector hypergraphs, as far as their correcting capabilities outside the guaranteed range of errors corrected.

Organization. The remainder of the paper is organized as follows. In Section 2, we describe the general technique used to generate the indices, and show its connections to the Turan problem. In Section 3, we define the corrector hypergraphs and prove some basic combinatorial properties of these constructions. In Section 4, we show how to use these corrector hypergraphs to design efficient AQ algorithms. We discuss the implementation details, and comment on how to choose the parameters to achieve the best complexity/error tradeoff for a given application. Finally in Section 5, we study further the error-correcting capabilities of corrector hypergraphs, and show that there is a hierarchy between them, as far as their correcting capabilities outside the guaranteed range of errors corrected.

2 Preliminaries

The basic observation used is that if a query resembles a word in D, they must have some common substring. Then it is possible to design an AQ algorithm by partitioning each word and query into $d+1$ substrings (which we call *zones*), and building $d+1$ hash indices based on these substrings (see [3],[13]). If there are at most d errors, then at least one of the indices will be free of errors. Therefore, it is possible to correct d errors, by hashing each word and query using each one of the $d+1$ indices. Note that the zones do not have to be contiguous substrings. (In fact, it was suggested in [3] that, for the English dictionary, indices that include alternating letters will assure less hash collisions.)

As d increases, the indices get smaller and many collisions may result in the hash table. Indeed, good hashing performance requires that the index size be at least $I = \log_2 |D|$. [1] Assuming that the data is well distributed, this would guarantee hash retrieval in constant expected time.

To cope with this constraint, it is possible to divide each word and query into $n \geq d+1$ zones. Each zone will now be smaller, but we can combine them in various fashions to yield hash indices of size I. If there are at most d errors, then at least $n-d$ of the zones will be without errors. Therefore, the objective is to build the indices in such a way, that for any pattern of at most d errors, there will exist an index which is composed out of some of the $n-d$ error free zones. The goal is to minimize the total number, m, of indices used (since this number is proportional to the space and time complexity of the algorithm) while correcting a given number of errors d. This turns out to have direct connections with the classical *Turan problem* in graph theory:

Given integers n, c and x, find a hypergraph G with n vertices and edges of size x, such that every subset of c vertices contains an edge, and the number, m, of edges is as small as possible.

The minimum number, m, of hyperedges meeting the above requirement is known as the *Turan number* $T(n, c, x)$.

The AQ problem corresponds exactly to the Turan Problem, if we let the number of vertices, n, correspond to the number of zones into which the words are partitioned; the edge size, x, correspond to the number of zones which compose an index; and $c = n - d$ be the number of correct (error free) zones. Any optimal solution to the Turan problem, can be used to build the indices for the AQ problem, by letting each index correspond to an edge in the optimal graph.

However, optimal hypergraphs for the Turan problem are known only for $x = 2$, and for a few cases of $x = 3$ (see [16], [5], [6], [4], [15]).

[1] We assume that the dictionary is resident in memory. For extremely large dictionaries stored on disks, the hash table may use a bucket size related to the page size.

We propose a general class of constructions, called *corrector hypergraphs*, which generalize the known optimal Turan graphs, and certain hypergraphs that are conjectured to be optimal Turan graphs. These graphs will be used to design efficient AQ algorithms using the method described above. The objectives of the corrector hypergraphs go beyond the classical Turan problem in several respects:

1. The Turan formulation is only concerned with the extremal case. That is, assuring that every subset of c vertices contains an edge. For our hashing application, we are interested in hypergraphs which, in addition, *maximize* the number of subsets of size $c' < c$ that contain an edge. Such constructions will have better error correcting capabilities outside the guaranteed range. This is a desired property assuming the dictionary is not clustered, because then there is a higher probability of finding some answer to queries of distance greater than d from any dictionary word.

2. The space and time complexity of the resulting hashing algorithm depends on the number, m, of indices (hyperedges), but not on the edge size. Hence, for the same error correcting capabilities, there are usually several choices of n, c and x available. Our corrector hypergraphs have an additional parameter t, and offer a wide range of possibilities for choosing the desired tradeoff between algorithm efficiency and the number of errors corrected.

3 Corrector Hypergraphs

A *corrector hypergraph* G has parameters t, k, r and x (all positive integers). Assume that:

$$n \bmod k = 0, \qquad t \leq r, \qquad (x-1) \bmod t = 0, \qquad \frac{x-1}{t} < k.$$

Let $\lambda = \frac{x-1}{t}$. Divide the vertices of G into r equal sets $V_1, ..., V_r$, each with k vertices. The edges of G will be of size x, and will contain vertices from up to t consecutive sets. There will be an edge with $a_1, ..., a_t$ vertices in $V_{(i+1) \bmod r}, ..., V_{(i+t) \bmod r}$, respectively, for $1 \leq i \leq r$, if $(a_1, ..., a_t)$ is a solution to the following set of equations:

$$\begin{cases} a_1 & \leq \lambda, \\ a_1 + a_2 & \leq 2\lambda, \\ \vdots & \\ a_1 + \cdots + a_{t-1} & \leq (t-1)\lambda, \\ a_1 + \cdots + a_{t-1} + a_t & = t\lambda + 1. \end{cases} \tag{1}$$

We call this hypergraph the (t, r, k, x)-*corrector* hypergraph. We will show that it has the following properties:

1. The total number of vertices is $n = rk$.

2. The total number of edges is $m = \frac{r}{t}\binom{tk}{x}$.

3. Every subset of $c = r(x-1)/t + 1$ vertices contains an edge.

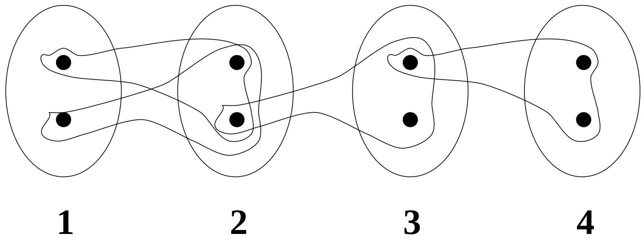

Figure 1: A $(2, 4, 2, 3)$-corrector; not all edges are shown.

3.1 Special Cases of Correctors

We first present a few special cases of corrector hypergraphs.

The Clique Hypergraph ($t = 1$): Divide the vertices of G into r equal sets, each has k vertices and contains all possible $\binom{k}{x}$ edges. Thus:

- The total number of vertices is $n = rk$.
- The total number of edges is $m = r\binom{k}{x}$.
- Every subset of $c = r(x-1)+1$ vertices contains an edge, since at least x of these vertices are contained in one clique.

A design that was considered in [3] corresponds to the Clique graph with $r = 1$.

The Turan Hypergraph ($t = 2$): Divide the vertices of G into r equal sets, $V_1, V_2, \cdots, V_r$, each with k vertices. An edge consists of i vertices from V_j and $x - i$ vertices from $V_{(j+1) \bmod r}$, where $0 \le i \le \lceil \frac{x-1}{2} \rceil$, and $1 \le j \le r$. Thus:

- The total number of vertices is again $n = rk$.
- The total number of edges is $m = r \sum_{i \le \lceil \frac{x-1}{2} \rceil} \binom{k}{i}\binom{k}{x-i}$.
- If the edge size x is odd, then every subset of $c = r(x-1)/2 + 1$ vertices contains an edge (see Theorem 1).

The Clique graph was shown by Turan to be optimal for $T(n, c, 2)$, where $n = k(c-1)$ and $r = c - 1$. The Turan graph is due to Turan, and conjectured to attain the optimal value for $T(n, 4, 3)$, where $n = 3k$ and $r = 3$ (see [16]).

Note that the Clique graph (with r sets of $2k$ vertices) and the Turan graph (with $2r$ sets of k vertices) have the same number of edges and guarantee to correct subsets of the same size c. However, as we will see (by Theorem 9), more subsets with *fewer* than $c = r(x-1)+1$ vertices, contain an edge in the Turan graph than in the Clique graph. Also, the Turan graph is applicable to a wider range of values for n (vertex set size).

The Extreme Case ($t = x - 1$): In the extreme case of a corrector with $t = x - 1$, every subset of $r + 1$ vertices contains an edge; thus one can correct any number d of errors by partitioning the vertices into $r = n - 1 - d$ sets.

3.2 Basic Properties of Correctors

To show that the (t, r, k, x)-corrector hypergraph G has certain desired properties, we first focus on sequences that satisfy Eq. (1).

Definition 1 *Let $a_1, \cdots, a_t$ and λ, be nonnegative integers. The sequence $(a_1, \cdots, a_t)$ is called λ-*bounded*, if it satisfies Eq. (1), i.e., $(a_1, \cdots, a_t)$ corresponds to an edge in the (t, r, k, x)-corrector hypergraph for $x = t\lambda + 1$.*

We can rewrite Eq. (1) in the following equivalent form:

$$\begin{cases} a_t & \geq & \lambda + 1, \\ a_{t-1} + a_t & \geq & 2\lambda + 1, \\ \vdots & & \\ a_1 + \cdots + a_{t-1} + a_t & = & t\lambda + 1. \end{cases} \tag{2}$$

Lemma 1 *If $(a_1, \cdots, a_t)$ is λ-bounded, then any of its suffixes $s = (a_i, a_{i+1}, \cdots, a_t)$ dominates (componentwise) a λ-bounded sequence $(a'_i, a'_{i+1}, \cdots, a'_t)$.*

Proof: All partial sums of s satisfy the constraints in Eq. (2) except for the sum $p = a_i + a_{i+1} + \cdots + a_t$, which may be larger than $(t - i + 1)\lambda + 1$. Let $\delta = p - (t - i + 1)\lambda - 1$. Use a greedy algorithm to subtract δ_i, $\delta_{i+1}, \cdots, \delta_t$ from $a_i, a_{i+1}, \cdots, a_t$ such that $\delta_i + \cdots + \delta_t = \delta$. The resulting sequence $(a_i - \delta_i, \cdots, a_t - \delta_t)$ is λ-bounded and dominated by s. ∎

Lemma 2 *Let $a_1, \cdots, a_r$ be a sequence of integers such that $a_1 + \cdots + a_r = r\lambda + 1$. Then there is a unique cyclic permutation π such that $(a_{\pi(1)}, \cdots, a_{\pi(r)})$ is λ-bounded.*

Proof: A non-extendible consecutive sequence $a_{i+1}, a_{i+2}, \cdots, a_{i+s}$ (all subscripts are mod r) of length s, will be called λ-maximal if:

$$\begin{array}{lcl} a_{i+1} & \leq & \lambda, \\ a_{i+1} + a_{i+2} & \leq & 2\lambda, \\ \vdots & & \\ a_{i+1} + \cdots + a_{i+s-1} & \leq & (s-1)\lambda, \\ a_{i+1} + \cdots + a_{i+s-1} + a_{i+s} & \geq & s\lambda + 1. \end{array}$$

A λ-maximal sequence of length $s = 1$ will be just a single element $a_i \geq \lambda + 1$. Notice that a λ-maximal sequence $a_{i+1}, ..., a_{i+s_1}$ of length $s_1 \geq 1$ must exist among $a_1, ..., a_r$, because $a_1 + \cdots + a_r = r\lambda + 1$. We claim that $s_1 = r$. Otherwise, find a λ-maximal sequence of length s_2, among the remaining $a_{i+s_1+1}, ..., a_i$, which ends with a_i (note that $a_i \geq \lambda + 1$, because $a_{i+1}, ...a_{i+s_1}$ was λ-maximal). If none exists, then $a_{i+1}, ..., a_{i+s_1}$ was not λ-maximal, because it can be extended to a λ-maximal sequence starting with a_{i+s_1+1} and ending with a_{i+s_1}. Therefore, a second λ-maximal sequence of length s_2 must exist. Continue to partition the remaining $a'_i s$ in a similar way into λ-maximal sequences, each ending with the previous λ-maximal sequence. But the sum of the elements in each λ-maximal sequence of length s is at least $s\lambda + 1$, and thus $a_1 + \cdots + a_r \geq r\lambda + 2$, a contradiction.

To show that there is only one possible cyclic permutation π, notice that by Eq. (2),

$$\begin{array}{lcl} a_{\pi(r)} & \geq & \lambda+1, \\ a_{\pi(r-1)}+a_{\pi(r)} & \geq & 2\lambda+1, \\ \vdots & & \\ a_{\pi(1)}+\cdots+a_{\pi(r-1)}+a_{\pi(r)}+ & = & r\lambda+1. \end{array}$$

Comparing the i-th constraint above with the i-th constraint in Eq. (1) shows that $a_{\pi(r-i+1)}$ cannot be the 'starting' position of a λ-bounded sequence, for $1 \leq i \leq r-1$. Therefore, the only possible cyclic permutation is the one that starts with $a_{\pi(1)}$. ∎

Theorem 1 *In a (t, r, k, x)-corrector hypergraph, every subset of $c = r(x-1)/t+1$ vertices contains an edge.*

Proof: Assume the subset has $a_1, \cdots, a_r$ vertices in $V_1, ..., V_r$ respectively, where $a_1 + \cdots + a_r = r(x-1)/t+1$. By Lemma 2, there exists a cyclic permutation π such that $p = (a_{\pi(1)}, \cdots, a_{\pi(r)})$ is λ-bounded for $\lambda = \frac{x-1}{t}$. Therefore by Lemma 1, the length-t suffix of p must contain an edge. ∎

Theorem 2 *A (t, r, k, x)-corrector hypergraph has $m = \frac{r}{t}\binom{tk}{x}$ edges.*

Proof: We will show that $V_1, \cdots, V_t$ contain $\frac{1}{t}\binom{tk}{x}$ edges and therefore the whole graph contains $\frac{r}{t}\binom{tk}{x}$ edges. The sets $V_1, \cdots, V_t$ contain a total of tk vertices, and therefore $\binom{tk}{x}$ potential edges. However, any set of $a_1 + \cdots + a_t = x$ vertices, with $a_1, \cdots, a_t$ vertices in $V_1, \cdots, V_t$ respectively, has exactly one cyclic permutation which will result in an edge in the graph (Lemma 2). Therefore, out of every t potential edges, only one actually exists. ∎

4 Implementation Details

The corrector hypergraphs can be used to design efficient AQ algorithms. These graphs have many parameters, which make them very flexible to use and offer a wide range of possibilities to choose the desired tradeoff between algorithm efficiency and the number of errors corrected. However, this flexibility may create confusion on how to choose the different parameters to best suit the implementation. We briefly outline a few subtleties which help make the implementation compact and efficient, and comment on how to choose the parameters of the correctors to achieve the best complexity/error tradeoff for a given application.

The Data Structure

The m edges of a given corrector hypergraph can be used to generate m hash indices. However, in practice each word in the dictionary needs to be stored only once. Then the m indices of a given word can be hashed to a hash table of pointers that will refer to that single copy of the word. This reduces significantly the space necessary to store the dictionary, and makes it practical to use a large number of indices and thus to correct a large number of errors. See Figure 2.

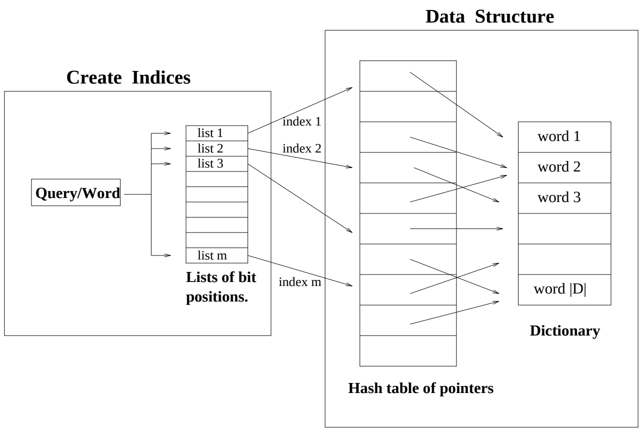

Figure 2: Outline of Algorithm.

Generating the Indices

Generating the indices themselves from the correctors can also be done in an efficient way. It is only necessary to generate the corrector hypergraph once, when the hash table is created, and to provide m templates (of bit positions) for the indices. Thereafter inserting a new word into the dictionary, or answering a query, can be done by referencing these templates. In Appendix A, we present a function in C, which creates templates for the edges of a (t,t,k,x)-corrector. The templates generated by this function, can then be used in an obvious way to generate the edges of a general (t,r,k,x)-corrector, by taking all cyclic permutations of these templates. The function itself generates the edges, in a way similar to that of a ripple counter. It begins with some initial legal edge, and then goes systematically over all other edges. Assume the $n = tk$ vertices of the (t,t,k,x)-corrector are numbered $\{0,1,2,...,tk-1\}$, and partitioned into t sets $V_1, V_2, ..., V_t$, where:

$$V_{i+1} = \{ik, ik+1, ..., (i+1)k-1\}, \qquad for \quad 0 \le i \le t-1.$$

The initial edge E, will include the sets of vertices $E_1, E_2, ..., E_t$:

$$E_{i+1} = \{ik, ik+1, ..., ik+\lambda-1\}, \qquad for \quad 0 \le i \le t-2,$$

$$E_t = \{(t-1)k, (t-1)k+1, ..., (t-1)k+\lambda\}.$$

The function generates the next edge by replacing the largest vertex in the current edge, E, with the next vertex in V_t, and continues to do so as long as possible. It then replaces the two largest vertices in the current edge, E, and so on.

The Complexity of the Algorithm

The question is how to choose the parameters of the (t,r,k,x)-corrector for a given application. In order to avoid collisions in the hash table, the size of an index should be at least $I = \log_2 |D|$, and perhaps even larger if the bits have low information content. Therefore, the index size, I, and the word length, ℓ, are determined by the specific application at hand. However, we are free to choose the number, x, of zones in each index (edge size), and thus the total number, n, of zones (total number of vertices). We should also choose r, k and t.

While our principal concern has been the worst case performance of corrector hypergraphs, Section 5 suggests that when $t = x - 1$, the highest percentage of subsets of size $c' < c$ can be corrected. If the dictionary is not clustered then this is a desired property.

Therefore, let $t = x - 1$, and partition each word into $n = \frac{\ell x}{I}$ equal zones (this would guarantee indices of length I). Then, using Theorem 1 and Theoref 2, it is possible to show that the number, m, of indices (edges) used, and the number, d, of errors corrected is:

$$m = \frac{\ell}{I}\binom{kx - k - 1}{x - 1}, \qquad d = \frac{\ell}{I}\frac{x(k-1)}{k} - 1.$$

The only parameters left to be decided are x and k. It turns out that it is always best to choose the minimal k possible to achieve the desired number of errors, since the number, m, of edges is a monotone increasing function with k, when the ratio $x(k-1)/k$ is fixed. It is often the case that $k = 2$ is the best choice, in the sense that it corrects more errors, using less edges. See for example Table 1.

$k=2$			$k=3$			$k=4$		
x	d	m	x	d	m	x	d	m
2	R-1	R	2	4R/3-1	2 R	2	3R/2 -1	3 R
3	3R/2-1	3R	3	2R -1	10R	3	9 R/4 -1	21 R
4	2R-1	10 R	4	8R/3 -1	56 R	4	3R -1	165 R
5	5R/2 -1	35 R	5	10R/3-1	330 R	5	15 R/4 -1	11365 R

Table 1: The tradeoff between x, d and m for different k, where $R = \ell/I$.

Given the above analysis, we have just proved that:

Theorem 3 *Using corrector hypergraphs it is possible to correct d errors, while hashing each word/query using $m = O(d2^{dI/\ell})$ indices of length I. If the dictionary D is uniformally distributed and $I = O(\log |D|)$, then the number of indices per word/query is $m = O(d|D|^{d/\ell})$, and the expected number of hash collisions per query is constant.*

Proof: Take a (t, r, k, x)-corrector as described above, where $t = x - 1$, $k = 2$ and $n = \frac{\ell x}{I}$ (i.e., $r = \frac{n}{k} = \frac{\ell x}{2I}$). Use the Stirling approximation to estimate the number of edges m. ∎

5 A Hierarchy of Corrector Graphs

The (t, r, k, x)-corrector hypergraphs have a rich combinatorial structure and are of independent interest. In this section we further study these graphs and show that there is a hierarchy between the different corrector hypergraphs, as far as their correcting capabilities outside the guaranteed range of errors corrected. We compare the correcting capabilities of families of corrector graphs, which have the same number, m, of edges. Our results indicate, that a corrector with a larger t parameter in such a family of graphs, will have more subsets that contain an edge, and therefore can correct more pattern of errors.

5.1 Properties of Sequences

We first prove further properties of subsequences.

Definition 2 *A subset U of vertices in a corrector hypergraph is called a* good *subset if it contains an edge; otherwise U is called a* bad *subset. If U has $a_1, ..., a_r$ vertices in $V_1, ..., V_r$ respectively, then $(a_1, ..., a_r)$ is called the* projection sequence *(or* sequence *for short) of U. A sequence $s = (a_1, ..., a_r)$ is called a* good *sequence if it is the projection sequence of a good subset U; otherwise s is called a* bad *sequence.*

For a sequence $s = (a_1, \cdots, a_r)$, let $s(i) = (a_{i-t+1}, \cdots, a_i)$ denote the length-t subsequence of s ending with a_i (the indices are taken mod r), for $1 \le i \le r$. Thus, using Eq.2, the sequence s is good if and only if some $s(i)$, $1 \le i \le r$, satisfies the following set of constraints.

$$\begin{cases} a_i & \geq \lambda + 1, \\ a_{i-1} + a_i & \geq 2\lambda + 1, \\ \vdots & \\ a_{i-t+1} + \cdots + a_{i-1} + a_i & \geq t\lambda + 1. \end{cases} \tag{3}$$

Let $s = (a_1, \cdots, a_r)$ be a bad sequence. Then for each i, $1 \le i \le r$, $s(i)$ violates one of above constraints, i.e., some partial sum of $s(i)$ is smaller than required. Let $a_j + a_{j+1} + \cdots + a_i$ be such a partial sum where j is as large as possible. We write $p(i) = j$, and call $p(i)$ the *proof number* of i (with respect to the sequence s). We also refer to the interval $[p(i), i]$ as the *proof interval* of i, and the subsequence $a_j a_{j+1} \cdots a_i$ as a *proof pattern* for i. (Thus, a proof pattern is a shortest "certificate" that no edge can end in position i.) Let P denote the collection of all subsequences that may occur as proof patterns (for given values of λ, t and k). For example, when $\lambda = 1$, $t = 3$, and $k \ge 3$, one can show that $P = \{0, 1, 02, 012, 003\}$. The following lemma follows directly from the definition of proof patterns.

Lemma 3 *Every sequence in P has length at most t, and no sequence in P is a suffix of another sequence in P (i.e., P is suffix-free).*

We next show that the proof intervals of a bad sequence must be properly nested.

Lemma 4 *Let $s = (a_1, \cdots, a_r)$ be a bad sequence. Then two proof intervals $[p(i), i]$ and $[p(i'), i']$ of s are either nested or have disjoint interior.*

Proof: We will show that $p(i) \le i' < i$ implies $p(i) \le p(i')$. By the definition of $p(i)$, the partial sum $a_{p(i)} + \cdots + a_i$ is smaller than required by Eq. (3), while the partial sum $a_{i'+1} + \cdots + a_i$ is large enough. Subtracting the latter from the former shows that $a_{p(i)} + \cdots + a_{i'}$ is too small. Hence the proof pattern for i' must be contained in $a_{p(i)} \cdots a_{i'}$, i.e., $p(i) \le p(i')$. ∎

It follows that, by taking the maximal proof intervals of a bad sequence s (viewed cyclically), one obtains a unique partition of s into a cycle of proof patterns. (When written linearly, one of these patterns may be "wrapped around".) This is a necessary and sufficient condition for bad sequences.

Theorem 4 *A sequence s is bad if and only if it can be written uniquely as $s = p_0 p_1 \cdots p_{u-1} p_u$ such that (1) $|p_0| \ge 0$, and $|p_i| > 0$ for $i \ge 1$; (2) $p_1, \cdots, p_{u-1}$ and $p_u p_0$ are in P.*

Thus, the bad sequences can be generated from a set of short patterns. This allows us to enumerate the bad sequences and the bad subsets recursively, as we shall see in the next section.

5.2 Generating Functions for Corrector Hypergraphs

We would like to derive a generating function to enumerate bad subsets, so that we can compare the correcting capabilities of any two corrector hypergraphs.

Definition 3 *Let b_i be the number of bad subsets of size i, in a (t, r, k, x)-corrector. The generating function for the number of bad subsets in this corrector will be:*

$$G_r^{[t]}(z) = \sum_i b_i z^i.$$

Therefore, b_i counts configurations of i error free zones that would not be corrected by the hashing scheme. We make explicit the dependence on r and t, although G will also depend on the other parameters of the corrector hypergraph, x, and k. When the context is clear, the superscript t will be omitted.

Definition 4 *Let $E^{[t]}(u, z)$ be a polynomial that enumerates the proof patterns, with a term*

$$\binom{k}{a_1}\binom{k}{a_2}\cdots\binom{k}{a_j} z^{a_1+a_2+\ldots+a_j} u^j$$

for each proof pattern $a_1 a_2 \cdots a_j$ in P.

For example, for the sample $P = \{0, 1, 02, 012, 003\}$:

$$E^{[3]}(u, z) = u + kzu + \binom{k}{2} z^2u^2 + k\binom{k}{2} z^3u^3 + \binom{k}{3} z^3u^3.$$

Grouping the terms in $E^{[t]}(u, z)$ by pattern length gives:

$$E^{[t]}(u, z) = E_1^{[t]}(z)u + E_2^{[t]}(z)u^2 + \ldots E_t^{[t]}(z)u^t. \tag{4}$$

where $E_j^{[t]}(z)$ is a polynomial that enumerates all proof patterns of length j. Again, when the context is clear, the superscript t will be omitted.

We can now write a recurrence relation for $G_r(z)$.

Theorem 5 $G_r^{[t]}(z) = \sum_{i=1}^{t} E_i^{[t]}(z) G_{r-i}^{[t]}(z)$, *for $r > t$.*

Proof: By Theorem 4, if s is a bad sequence, then it can be written uniquely as $s = p_0p_1...p_{u-1}p_u$, where p_i are proof patterns. Notice that whenever $|s| > t$ there will be at least two maximal proof patterns in s. Hence, we can shorten s by removing p_1, the first pattern that does not wrap around s, and reduce $|s|$ by as much as t, giving a t-th order recurrence. ∎

Definition 5 $(1 + z)^k \bmod z^i \stackrel{\text{def}}{=} 1 + kz + \binom{k}{2}z^2 + \ldots \binom{k}{i-1}z^{i-1}$.

We can now give the initial conditions for the recurrence in Theorem 5:

Theorem 6 $G_t^{[t]}(z) = (1 + z)^{kt} \bmod z^x$.

Proof: When there are exactly t groups, we know by Lemma 2 that every possible edge of size x is present. Thus, the construction is equivalent to a single hyperclique on kt vertices. The generating function enumerates all ways of choosing fewer than x vertices from this hyperclique. ∎

In fact Theorem 6 is a special case of a more general result:

Theorem 7 $G_q^{[t]}(z) = (1+z)^{kq} \bmod z^{\lambda q+1}$ *when* $q \le t$, *and* $\lambda = (x-1)/t$.

Proof: Note that if $q < t$, then a sequence $s = (a_1, ..., a_q)$ is bad, if all of its subsequences of length t (wrapped around), violate one of the constraints in Eq. 3. Therefore, it is clear that if $a_1 + ... + a_q \le \lambda q$, then s is bad. We have to show that all sequences s with $a_1 + ... + a_q \ge \lambda q + 1$ are good. Assume with out loss of generality that $a_1 + ... + a_q = \lambda q + 1$. By Lemma 2, there exists a unique cyclic permutation π, such that $(a_{\pi(1)}, ..., a_{\pi(q)})$ is λ-bounded. Then, the wrapped around subsequence of s of length t, which ends with $a_{\pi(q)}$ satisfies Eq. 3, and therefore s is good. Thus, $G_q^{[t]}(z)$ enumerates all subsets with at most λq vertices. ∎

Theorem 5 gives us a useful recurrence relation for $G_r(z)$. However, in order to better understand $E(u,z)$ we will derive $G_r(z)$ using another technique. First we define a trivariate generating function.

Definition 6 *Let* $F(u,w,z) = \sum_{i,\ell,r>0} f_{i\ell r} u^r w^i z^\ell$, *where* $f_{i\ell r}$ *is the number of bad sequences* $(a_1, ..., a_r)$, *with* i *proof patterns, and* $a_1 + ... + a_r = \ell$.

We are not particularly interested in enumerating according to the number of proof patterns; the variable w will be used at intermediate stages in the analysis and eventually be set to one to obtain the generating function defined at the beginning of this section:

$$F(u,1,z) = \sum_{r>0} G_r(z) u^r.$$

We can also define a similar trivariate generating function, $H(u,w,z)$, for non-empty bad sequences that begin with a proof pattern. (By contrast, Theorem 4 permits one proof pattern of sequences in $F(u,w,z)$ to wrap around from the beginning to the end of the sequence.) The function $H(u,w,z)$ is not the enumeration we want, but it does have a simple expression in terms of the enumerator of proof patterns, $E(u,z)$, defined above.

Lemma 5 $H(u,w,z) = \frac{wE(u,z)}{1-wE(u,z)}$.

The generating functions F and H are related by the following Lemma.

Lemma 6 $w\frac{\partial}{\partial w}F(u,w,z) = u\frac{\partial}{\partial u}H(u,w,z)$.

Proof: Let the bad sequences $(a_1, \ldots, a_r)$ be arranged on a circle, so that there is no origin. The generating function F enumerates sequences where a single position is chosen as the origin, whereas the generating function H enumerates sequences where a single proof pattern is chosen as the origin. We now enumerate sequences where both a single position and a single proof pattern are designated on the circle. The Lemma results from two ways of enumerating these "double origin" sequences; we can take a sequence in F and choose any proof pattern for the other origin, or we can take a sequence in H and choose any position for the other origin. ∎

Integrating with respect to u and w, gives another expression for G in terms of E.

Theorem 8 $1 - E^{[t]}(u,z) = e^{-G_1^{[t]}(z)u - G_2^{[t]}(z)\frac{u^2}{2} - ... - G_r^{[t]}(z)\frac{u^r}{r} - ...}$.

Since the polynomial on the left hand side of the equation in Theorem 8, is t-th order in u, we can omit the high order terms from the right hand side of the equation. This way $E(u,z)$ and the recurrence in Theorem 5, can be computed from the initial values of $G_r(z)$ given in Theorem 7. When $t < 5$ we can solve the recurrence relation for all $G_r(z)$. For example:

Corollary 1 *If $t = 2$ and $x = 3$ (i.e. Turan's construction) then $G_r^{[2]}(z) = v_1^r + v_2^r$,, where*

$$v_1, v_2 = \frac{1 + kz \pm \sqrt{2k(k-1)z^2 + (1+kz)^2}}{2}.$$

Proof: By Theorem 8

$$1 - E^{[2]}(u, z) = e^{-G_1^{[2]}(z)u - G_2^{[2]}(z)\frac{u^2}{2}} \bmod u^3 = 1 - (1 + kz)u - \binom{k}{2} z^2 u^2.$$

Solving the recurrence in Theorem 5, gives the desired result. ∎

In practice most constructions have small enough parameters such that the recurrence relation can be used to compute the desired generating functions. For example, we can find the generating function $G_r^{[t]}$ that enumerates bad subsets for three different corrector hypergraphs, each with $n = kr = 16$ vertices, edges of size $x = 5$, and a total of $m = 112$ edges:

- $r = 2, k = 8, t = 1$:

$$G_2^{[1]}(z) = 1 + 16z + 120z^2 + 560z^3 + 1820z^4 + 4256z^5 + 7056z^6 + 7840z^7 + 4900z^8.$$

- $r = 4, k = 4, t = 2$:

$$G_4^{[2]}(z) = 1 + 16z + 120z^2 + 560z^3 + 1820z^4 + 4256z^5 + 7024z^6 + 7568z^7 + 4322z^8.$$

- $r = 8, k = 2, t = 4$:

$$G_8^{[4]}(z) = 1 + 16z + 120z^2 + 560z^3 + 1820z^4 + 4256z^5 + 7000z^6 + 7472z^7 + 4078z^8.$$

For reference, the binomial expansion of $(1 + z)^{16}$ is:

$$1 + 16z + 120z^2 + 560z^3 + 1820z^4 + 4368z^5 + 8008z^6 + 11440z^7 + 12870z^8 + 11440z^9 + \ldots$$

Note that while all the above correctors guarantee to correct subsets with $c = 9$ or more vertices, there is a considerable difference in their correcting capability, when there are slightly fewer than 9 vertices. All other parameters being equivalent, it appears to be better to choose t as large as possible, when the dictionary is not clustered. We prove this for a few cases in the next subsection.

5.3 Monotonicity of Correctors

We conjecture that, in general, a (t, r, k, x)-corrector hypergraph has fewest bad subsets (for fixed number of vertices $n = rk$ and edge size x) when t is made as large as possible. In the next theorem we establish the monotonicity claim for the case of $t = 2$ versus $t = 1$. That is we prove that the $(2, 2r, k, x)$-corrector hypergraph (Turan graph) corrects more patterns of errors than the $(1, r, 2k, x)$-corrector hypergraph (Clique graph) for odd x, although in both graphs $m = r\binom{2k}{x}$ and $c = r(x-1) + 1$.

Theorem 9 *The $(2, 2r, k, x)$-corrector has fewer bad subsets than the $(1, r, 2k, x)$-corrector, for odd x.*

Proof: First note that by Theorem 6, the initial conditions $G_1^{[1]}(z)$ and $G_2^{[2]}(z)$ agree (they are both hypercliques on $2k$ points). We wish to compare the even terms of the expansion

$$-\log(1-E^{[2]}(u,z)) = G_1^{[2]}(z)u + G_2^{[2]}(z)u^2/2 + G_3^{[2]}(z)u^3/3 + \ldots$$

with terms corresponding to half as many groups in a similar expansion for $t=1$. Let:

$$Q^{[i]}(u) = 1 - E^{[i]}(u,z).$$

We wish to show that $\log(Q^{[2]}(u)Q^{[2]}(-u)/Q^{[1]}(u^2))$ has positive coefficients. Since

$$Q^{[2]}(u)Q^{[2]}(-u) = e^{-G_2^{[2]}(z)u^2 - G_4^{[2]}(z)u^4/2 - \ldots}$$

and $G_1^{[1]}(z)$ and $G_2^{[2]}(z)$ agree, the low order terms are equal:

$$Q^{[2]}(u)Q^{[2]}(-u) = Q^{[1]}(u^2) + R(z)u^4,$$

where $R(z) = (E_2^{[2]}(z))^2$, and $E_2^{[2]}(z)$ must have positive coefficients by definition (Eq. (4)). The theorem follows from the positive coefficients of

$$\log(Q^{[2]}(u)Q^{[2]}(-u)/Q^{[1]}(u^2)) = -\log(1 - \frac{R(z)u^4}{Q^{[2]}(u)Q^{[2]}(-u)}).$$

∎

An alternative proof of this last theorem, which uses direct counting techniques can be found in [11]. In the next theorem, we extend the monotonicity to $2t$ versus t for any t, with edge size $x = 2t+1$. We will need two lemmas.

Lemma 7 *When $\lambda = 1$, we have $E_i^{[t]}(z) = \frac{1}{i-1}\binom{(i-1)k}{i} z^i$ for $2 \le i \le t$.*

Proof: It is easy to verify that, for $\lambda = 1$, a sequence $a_1 a_2 \cdots a_i$ of length i ($2 \le i \le t$) is a proof pattern if and only if 1) $a_1 = 0$ and 2) $(a_2, \cdots, a_i)$ corresponds to an edge pattern in the $(i-1, r, k, i)$-corrector hypergraph. The lemma then follows from Theorem 2. ∎

Lemma 8

$$\binom{ik}{i}\binom{jk}{j} \le \binom{(i-1)k}{i-1}\binom{(j+1)k}{j+1}$$

for $2 \le i \le j$.

Proof: We will show that

$$\frac{\binom{jk}{j}}{\binom{jk+k}{j+1}} \le \frac{\binom{ik-k}{i-1}}{\binom{ik}{i}}. \tag{5}$$

Expanding both sides of Eq. (5) gives

$$\frac{j+1}{jk+k}\frac{(jk-j+1)(jk-j+2)\ldots(jk-j+k-1)}{(jk+1)(jk+2)\ldots(jk+k-1)} \le$$

$$\frac{i}{ik}\frac{(ik-i-k+2)(ik-i-k+3)\ldots(ik-i)}{(ik-k+1)(ik-k+2)\ldots(ik-k+k-1)}.$$

This can be rewritten as

$$\frac{1}{k}\prod_{\ell=1}^{k-1}\frac{jk-j+\ell}{jk+\ell} \leq \frac{1}{k}\prod_{\ell=1}^{k-1}\frac{(i-1)k-i+\ell+1}{(i-1)k+\ell}$$

or

$$\prod_{\ell=1}^{k-1} 1-\frac{j}{jk+\ell} \leq \prod_{\ell=1}^{k-1} 1-\frac{i-1}{(i-1)k+\ell}.$$

But the last inequality holds since

$$\frac{j}{jk+\ell} > \frac{j-1}{(j-1)k+\ell}$$

for any $\ell, k, j \geq 1$. ∎

Theorem 10 *The $(2t, 2r, k, x)$-corrector has fewer bad subsets than the $(t, r, 2k, x)$-corrector for $x = 2t+1$.*

Proof: As in the proof of Theorem 9, we will show that the polynomial

$$R(u,z) = Q^{[2t]}(u)Q^{[2t]}(-u) - Q^{[t]}(u^2),$$

has all positive coefficients. As before, we note that $R(u,z)$ has nonzero coefficients only for terms u^i with $i > 2t$. Since

$$R(u,z) = (1-E^{[2t]}(u,z))(1-E^{[2t]}(-u,z)) - (1-E^{[t]}(u^2,z)),$$

the higher order terms of $R(u,z)$ arise solely from the product $E^{[2t]}(u,z)E^{[2t]}(-u,z)$. Hence we will focus on the latter product. As all odd terms in $E^{[2t]}(u,z)E^{[2t]}(-u,z)$ vanish, let $c_{2\ell}(z)$ be the coefficient of the term $u^{2\ell}$ for $2t+2 \leq 2\ell \leq 4t$. Thus,

$$c_{2\ell}(z) = \sum_{\substack{i+j=2\ell \\ 2\leq i,j\leq 2t}} (-1)^i E_i^{[2t]}(z)E_j^{[2t]}(z).$$

By Lemma 7,

$$c_{2\ell}(z) = \sum_{\substack{i+j=2\ell \\ 2\leq i,j\leq 2t}} \frac{(-1)^i}{(i-1)(j-1)}\binom{(i-1)k}{i}\binom{(j-1)k}{j} z^{2\ell}.$$

Let,

$$d_{ij} = \frac{1}{(i-1)(j-1)}\binom{(i-1)k}{i}\binom{(j-1)k}{j}.$$

We will show that $\sum(-1)^i d_{ij} > 0$. Note that $d_{ij} = d_{ji}$. Furthermore, it can be derived easily from Lemma 8, that

$$d_{i,j} \leq d_{i-1,j+1} \quad for \quad 3 \leq i \leq j.$$

We break $\sum(-1)^i d_{ij}$ into pair-wise sums as follows:

$$\sum_{\substack{i+j=2\ell \\ 2\leq i,j\leq 2t}} (-1)^i d_{ij} = (d_{2,2\ell-2} - d_{3,2\ell-3}) + \cdots + (-d_{2\ell-3,3} + d_{2\ell-2,2}).$$

That is, the $2\ell-3$ terms are grouped into pairs, with one positive, central term left by itself ($(-1)^\ell d_{\ell,\ell}$ in case ℓ is even, and $(-1)^{\ell+1} d_{\ell+1,\ell-1}$ in case ℓ is odd). Since each pair yields a nonnegative difference, it follows that $\sum(-1)^i d_{ij} > 0$. This completes the proof of the theorem. ∎

Conclusion

We have presented a family of hypergraph constructions that are useful in efficient multiple-index hashing schemes for approximate query problems. These hypergraphs appear to have an interesting combinatorial structure. It is an open problem to determine the full hierarchy between the corrector hypergraphs.

Acknowledgement

We would like to thank Noga Alon and Ronald Graham for pointing us to the literature on the Turan problem. We would also like to thank Mike Paterson for helpful discussions.

References

[1] W.G. Brown. *On an Open Problem of Paul Turan Concerning 3-Graphs.* Studies in Pure Mathematics, pp. 91–93.

[2] D. Dolev, Y. Harari, N. Linial, N. Nisan and M. Parnas. *Neighborhood preserving hashing and approximate queries.* 5th ACM Symp. On Discrete Algorithms, pp. 251–259, 1994.

[3] D. Dolev, Y. Harari, M. Parnas. *Finding the neighborhood of a query in a dictionary.* 2nd Israel Symp. on Theory of Computing and Systems, pp. 33–42, 1993.

[4] D.G. Fon-Der-Flaass. *A Method for Construction of* $(3,4)$*-Graphs.* Matematicheskie Zametki, 44(4), pp. 546–550, 1988, (translated in Mathematical Notes of the Academy of Sciences of the USSR).

[5] G. Kalai. *A new Approach to Turan's Conjecture.* Graphs and Combinatorics 1, pp. 107–109, 1985.

[6] A.V. Kostochka. *A class of Constructions for Turan's (3,4)-Problem.* Combinatorica, 2(2), pp. 187–192, 1982.

[7] D. E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, 1975.

[8] G. M. Landau and U. Vishkin. *Efficient String Matching in the Presence of Errors.* Proc. 26th IEEE Symposium on Foundations of Computer Science, 1985.

[9] G.M. Landau and U. Vishkin. *Introducing Efficient Parallelism into Approximate String Matching and a New Serial Algorithm.* Proc. 18th Annual ACM Symposium on Theory of Computing, 1986.

[10] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, Massachusetts, 1969.

[11] M. Parnas. *Robust Algorithms and Data Structures for Information Retrieval.* PHD Thesis, 1994.

[12] J. Peterson. *Computer Programs for Detecting and Correcting Spelling Errors.* Communications of the ACM, 23(12), pp. 676–686, 1980.

[13] R. Rivest. *Partial-Match Retrieval Algorithms.* SIAM J. Computing, 5(1), pp. 19–50, 1976.

[14] D. Sankoff and J.B. Kruskal. *Time Warps, Strings Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley Publishing Company, Inc., 1983.

[15] A.F. Sidorenko. *Precise Values of Turan Numbers.* Matematicheskie Zametki, 42(5), pp. 751–760, 1987, (translated in Mathematical Notes of the Academy of Sciences of the USSR).

[16] P. Turan. *Research Problem.* Közl MTA Mat. Kutató Int. 6, pp. 417–423, 1961.

A Generating the Indices

```
/**********************************************************/
/* Generate the edges of a (t,t,k,x)-corrector, where x = t*λ+1. */
/* Store resulting edges in array edges[m][x], where m is the    */
/* number of edges, and x is the size of an edge.                */
/**********************************************************/


create_edges(edges,k,t,x,λ)
{
int current[x];                       /* Holds current edge. */
int stop = 1;                         /* Flag. */
int num = 0;                          /* Counts edges. */
int i,j;


for ( i = 0 ; i < t ; i++ )           /* Initialize first edge */
    for ( j = 0 ; j < λ ; j++ )
        edges[0][i*λ+j] = current[i*λ+j] = i*k+j;
    edges[0][x-1] = current[x-1] = (t-1)*k+λ;



if (current[0] == t*k-x) stop = 0;          /* Generate the edges */
while (stop == 1) {
        j = x-1; current[x-1]++;
        while (current[j] ≥ (t*k-x)+j+1){
                j--; current[j]++;
        }
        while (j < x-1){
                j++;
                if ((j mod λ == 0) & (j ≠ x-1))
                   current[j] = (current[j-1]+1 > (j/λ)*k) ? current[j-1] +1 : (j/λ)*k;
                else current[j] = current[j-1] +1 ;
        }
        num++;                        /* Copy 'current' to 'edges' */
        for (j = 0; j < x; j++)
             edges[num][j] = current[j];
        if (current[0] == t*k-x) stop = 0;
}
}
```