Hash functions for binary and ternary words

Warren D. Smith*
wds@research.NJ.NEC.COM

27 May 1992

Abstract — **We describe a simple integer-valued "hash" function $h(x)$, where x is a binary word of some fixed length. This hash function has the properties that**

1. **$h(x)$ may be computed in $O(n)$ steps where n is the length of x,**
2. **$h(y)$ may be computed in $O(d)$ time, if $h(x)$ is known and y and x differ at d (known) bits.**
3. **h is "locally collisionless:" $h(x) \neq h(y)$ if $x \neq y$ and x and y are Hamming distance $\leq k$ apart.**

These properties of our hash functions emerge from simple number theoretic arguments.

Modifications of the same idea work for ternary words, and binary words of constant weight. We mention two applications of this hash function in computer programming: to playing the Oriental game of Go, and to a new heuristic method of finding good local optima for intractible combinatorial optimization problems. As an application of these properties in mathematics, we obtain new error correcting codes to radix B, with minimal distance 3. We find a lower bound on the number of codewords which implies, subject to a certain number-theoretic conjecture, that these codes include an infinite number of new record-cardinality distance-3 codes. Regardless of the fate of that conjecture, our construction certainly leads to thousands of particular new record ternary codes.

Keywords — Go, combinational optimization, local optima, hash function, difference sets, prime numbers, ternary error-correcting codes, simulated annealing.

1 Introduction

One often wants to know whether two binary words x and y are identical, but it often is too laborious to always answer this question by comparing every bit of the two words. A faster method of answering this question can be to compute "hash functions" $h(x)$ and $h(y)$ and observe that if $h(x) \neq h(y)$, then $x \neq y$. In the unlikely event that $h(x) = h(y)$, then we can go on to perform a bitwise comparison, or, alternatively, we could compute some additional hash functions $h_k(x)$ and $h_k(y)$, $k = 1, 2, 3, \ldots$ and if equality holds for all the values of k we try, and the hash functions themselves were good ones, we could conclude that it was very probable that $x = y$. Sometimes this nonrigorous conclusion of "very probable equality" is sufficient for one's purposes.

1.1 Application 1: Go

We now mention two practical applications in which, in fact, this approach yields tremendous time savings.

The first is the Oriental game of Go. Go is played on a 19×19 grid, each of whose vertices is either "black," "white," or "uncolored." There are thus approximately 3^{361} possible Go positions, $361 = 19^2$, and each position may be regarded as a 361-trit long ternary word. At each successive move in the game, one additional uncolored vertex is colored; except that (comparatively rare) "capturing" moves can occur. Capturing moves also result in the *un*coloring of a certain number (usually 1 or 2, but conceivably much larger) of colored vertices.

In Go, it is *illegal* to make a move which *repeats* a position which has previously occurred in the game. In a typical situation, there are 100 positions that have previously occurred, and one is considering 100 possible moves in each position and looking several ply deep, say considering 10000 possible 4-move sequences. Naturally one only wants to consider move-sequences consisting only of legal moves. The naive approach to checking move legality would thus involve about $4 \times 10000 \times 100 \times 361 \approx 1.5 \times 10^9$ steps, i.e. by comparing the positions gotten after every candidate move, to each position in the list of previously occurring positions, 1 trit at a time. Naturally, this labor may be reduced. By lexicographically pre-sorting the previously-occurred positions (but not the few new ones which arise during the move sequence itself!), we reduce the work by a factor of perhaps 30. Usually we realize that no repetition has occurred after only examining a third of the position, so cut by another factor of 3, so the total work is really only about 1.5×10^7 steps. The hash function approach which we will now recommend, however, will cut the work by an *additional* factor of perhaps 50. This is because the work to find the hash function of a position, given that we know the last move and we know the hash function of the predecessor position, is usually only 1 step. Looking this hash function up in a table to see if it has previ-

*NEC Research Institute, 4 Independence Way Princeton NJ 08540

ously occurred, takes only one step, too. However this hashing method will, with very small probability, sometimes mistakenly regard a legal move as illegal. By the use of several "independent" hash functions of our type, this mistake probability may be made very low without unduly increasing the space required for the hash tables. (Such mistakes will never cause us to make an illegal move, but could detrimentally affect the quality of our play. If such mistakes only occur for 1 move out of every 10^{12} considered, the effect on the quality of the one's play will be negligible.)

1.2 Application 2: Combinatorial optimization

Many intractible combinatorial optimization problems may be phrased as a problem of finding an "optimal" n-bit binary word. For example, we mention the problem of "optimal graph bipartition." In this problem we have a n-vertex graph G, where $n = 2m$, and we wish to label each vertex with either a 1 or a 0, in such a way that the total number of 1's is m (balanced cut), and the "cost" of the cut, i.e. the total number of edges of G which join an 0-vertex to a 1-vertex, is minimum possible. This problem is known to be NP-complete [4].

One will observe that in this example, the legal words are not *all* n-bit words, but in fact are the n-bit words of *constant weight* m, which is a situation which often arises and which we will later address.

A heuristic approach, to find not necessarily optimal, but often good, solutions to such problems is *local search*. In this approach, one starts with some legal n-bit word, often generated at random, and then examines the Hamming ball of the $\binom{n}{k}$ n-bit words within Hamming distance k of that word, where k is some small number. If we find a legal word in this ball with cost less than or equal to the present word, we make it the present word and continue on. One continues until either one gets stuck (all the words in the Hamming ball have higher cost than I do) – which usually happens rather quickly – or one becomes fatigued.

The main disadvantage of this approach is precisely its virtue: one tends to get stuck in local optima which are not anywhere near as good as the global optimum. Two fixes have been proposed to reduce this disadvantage. The first fix is to search repeatedly from many different random initial starts. This fix introduces a new disadvantage: if there is some connected subset of the space of legal words, all of which have small cost, then one would prefer to stay in this subset, rather than wasting time repeatedly descending into it from random starts. The second fix is to deform the Hamming ball, searching deeper in some (believed to be promising) "directions" than in others. This idea can help, but of course the essential problem remains unaffected.

The method of "simulated annealing" is the same as local search, except that it occasionally allows "uphill" moves, where the probability that such a move is permitted is small if the contemplated cost increase is large compared to a "temperature" parameter, which is gradually decreased ("cooled") as the method progresses. Simulated annealing thus overcomes the two disadvantages of repeated local search: the uphill moves allow one to climb out of local optima, and the gradual cooling allows one to gradually restrict oneself to promising lower and lower cost subsets of the space, if they exist. Unfortunately simulated annealing has a new third disadvantage: the sequence of words generated will typically involve a great number of repetitions of the same word. Although one can get unstuck from local optima, it may take a very long time involving a great number of repeated failed attempts to climb out of a local basin before finally succeeding.

We are now going to propose a new method which overcomes all three disadvantages. It is simply this.

HEURISTIC SEARCH ALGORITHM. Explore the Hamming ball consisting of the words at Hamming distance $\leq k$ from the current word. If you find a legal word with smaller cost, and which has not been explored before, then make it the current word and continue on. (In one "eager" version of the algorithm, one goes to the first such word found. In the "demanding" version one goes to the least cost such word in the Hamming ball, breaking ties in some manner if necessary.) Otherwise [uphill step] make the current word the least-cost legal one in the Hamming ball, which has not previously been visited, and continue on.

It is still possible for this method to become stuck, but this can only happen if all the legal words in the Hamming k-ball surrounding the present word have already been visited. This is quite unlikely, and even if it does occur, then (at least in the demanding variant) one will be guaranteed to have found, previously, the least-cost legal word within Hamming distance $\leq 2k$ of the current one. (And even if one *is* at a $2k$-optima, it is still probable that the algorithm will not get stuck.) Thus this method is much more immune to becoming stuck in local optima than naive local search, but it never revisits a word, unlike simulated annealing. By using the hash functions described in this paper, we may determine whether a word has been visited previously, in only $O(1)$ time with a very high correctness probability, and one may compute the hash function of the current word, no matter how long it is, in $O(1)$ time. Again, a low probability of error (very occasionally excluding some word which should not have been excluded) will not be a practical disadvantage. $O(1)$ words of storage will typically need to be allocated to "remember" each new word we visit, if efficent hashing schemes (a subject which will not be addressed in this paper) are used.

This algorithm is related to another idea called "TABU search" [5]. The TABU algorithm maintains a "TABU list" of the most recent T transitions one has made. T is some small number such as 9. One continually modifies the current word to make it as small cost as possible (within the local neighborhood) *except* that remaining at the current location or making the inverse of any transition on the TABU list, is forbidden. Among other things, this prevents infinite cycles of period 2 in the sequence of visited words. However, TABU can allow infinite cycles of length 3. (Consider the three transitions corresponding to XORing the current word with 101_2, 011_2, and 110_2.) Another flaw in TABU is the fact that a forbidden move from the TABU list could in fact lead directly to the never-previously encountered global optimum! The present algorithm has neither of these flaws: No state-space cycles (of any period) can occur, and (except with very low probability) no move is forbidden unless it would lead to a cycle. In my opinion TABU has only one advantage over the present algorithm: it has bounded memory consumption. The present algorithm keeps storing more and more stuff in hash tables (typically 1 more word per state-word explored).

1.3 The point

The two properties of our hash function that were essential in making the two applications (we just discussed) work, were (1) the fact that we could update the hash function in $O(1)$ time, and (2) we made probabilistic claims ("high correctness probability").

The $O(1)$ update time will be shown next section. We will never really provide any rigorous justification for the probabilistic claims (but see section 2.1). Considering that the applications we've discussed are wholy deterministic, perhaps such a justification cannot make much sense. However, we will now provide a nonrigorous plausibility argument. (Also in any particular application, one may verify these claims by statistical experiments.)

If it were true that the hash functions we will present behaved like "independent random functions," and if it were true that the words we use in our hash tables and hash table lookups were independently randomly chosen from a uniform distribution, then certainly these claims would be justified. In fact, as soon as any reasonable reader sees our hash functions (next section), he will believe that they look very independent and are quite pseudorandom. However, it is certainly *not* the case that the words (in the two applications above) look like independent choices from a uniform distribution. They in fact form trails and clusters: if a word occurs, it is much more likely than usual that another word will occur at small Hamming distance from the present word. To overcome this deviation from uniform i.i.d.-ness, note property 3 mentioned in the abstract; we will show specifically how to design hash functions such that any two words at small Hamming distance have *different* hash values. Thus our hash functions will in fact perform *better* than random functions, for words close together under Hamming distance, but they will behave similarly to *independent* random functions for words at larger Hamming distances (see section 2.1).

2 Hash functions based on Prime numbers.

A hash function $h(x)$ will be said to be "k-locally collisionless" if it obeys property 3 of the abstract.

The hash functions we discuss are not new, but their local collisionlessness properties have not been observed in the hash function literature. Some ideas similar to this property *have* been used before in the combinatorics literature [8] [6].

Specifically, we recommend the following *linear modular* function: Regard the binary (or ternary) word as a number N in base B ($B = 2$ or 3 in the binary and ternary cases). The hash function is

$$h(x) = c \sum_{i=0}^{n-1} g^i d_i \bmod p \qquad (1)$$

where d_i is the ith digit of the word, c is some constant, typically of size $c \approx 2p/3$, p is some suitable *prime* number, and g is some suitable number. It is often convenient to choose p so that $g = B$ is a generator modulo p. In this case $h(x) = cN \bmod p$.

Different p's yield different, and "independent," hash functions. Only certain *good* choices of p are going to yield local collisionlessness. We will clarify this in the next few subsections. Properties 1 and 2 of the abstract, however, are easy:

To compute the hash function in n steps, simply set h equal to 1, and then for $i = n-1 \ldots 0$ do $h \leftarrow gh + d_i \bmod p$, then multiply by $c \bmod p$. To compute the change in the hash function when the ith bit of the word is changed, in 1 step, we use a pretabulated table of $cg^i \bmod p$, for $i = 0 \ldots n-1$. The new hash function is the old one, plus or minus cg^i, modulo p, in the binary case $B = 2$. If $B > 2$, then the correction is $cg^i \Delta_i$ where Δ_i is the change in the ith digit.

2.1 "Independent" hash functions?

Two hash functions of our type based on *different* primes p are "independent" in the following sense.

If $h_p(x) - h_p(y)$, which is (by modular linearity) the same as $h_p(z)$, $z = x - y \neq 0$ takes on some value (in particular, the value 0, indicating the [false] equality of x and y), then it is still as unlikely as ever that $h_q(z)$, where q is some other prime, will take on that value.

This is because of the Chinese Remainder Theorem. This theorem, specialized to our problem, states that there is exactly one integer N in the range $1, 2, \ldots, \prod_{i=1}^{k} p_i$ which is congruent to $a_i \bmod p_i$, for any particular values a_i, $i = 1 \ldots k$, where the p_i's are distinct primes.

2.2 Which primes to use.

We now explain how to choose p and g to obtain locally collisionless hash functions. The choice of c is irrelevant to local collisionlessness; its presence merely allows the hash functions to behave more "independently randomly" than if $c = 1$ were used (in which case there would be strong correlated dependence on the first few least significant digits of the word).

If g is nonzero modulo p and $p > B$, then mg^i, $i = 0 \ldots p-1$, $m = \pm 1, \ldots, \pm(B-1)$ are all *nonzero* modulo p. Hence $h(x)$ will be 1-locally collisionless, for *any* $B = 2, 3, \ldots$.

If we choose an odd prime p with $p > 2n$ and such that g is a *generator* modulo p, then $\pm g^i$, $i = 0 \ldots n-1$, $n \le (p-1)/2$, are also all *unequal* modulo p, so that with $B = 2$, we will always get 2-local collisionlessness. With $B \ge 3$, suppose we use L_1*-distance*, in which two words are distance 1 apart if they differ in a single digit and the difference is 1. Thus base-B words at L_1-distance 2 apart either differ by 2 in a single digit, or else by 1 in two digits, and the difference in the hash function is thus $2 \times g^i$ or else $\pm g^i \pm g^j$, $i \ne j$, independent $\pm$'s. In either case, this is nonzero, so we have 2-local collisionlessness for any $B \ge 3$, using L_1-distance. (Incidentally, in the application to the game of Go, if black is -1, white is $+1$, and uncolored is 0 – regarding ternary digits of N as being selected from the set $\{-1, 0, 1\}$ – then transitions from ∓ 1 to ± 1 in a single digit during a Go game are essentially impossible,[1] so ternary L_1-distance actually is the right metric to use. This metric also seems of more interest than the usual Hamming metric when transmitting ternary information on a noisy electrical channel with signals of $\{-1, 0, 1\}$ volts.) Here it is not absolutely necessary that g be a generator mod p, it will suffice for the $1 \pm g^i$ mod p, $i = 1 \ldots n-1$, to be nonzero.

If we are only concerned with the binary case $B = 2$ and with binary words of *constant weight*, then odd primes p with $p > n$ and g a generator mod p (or less strongly, such that $1 - g^i$ mod p, $i = 1 \ldots n-1$ are nonzero) suffice to get 3-local collisionlessness, because $k = 3$ is implied by $k = 2$, since only even distances can occur in the constant weight case.

Graham and Sloane [6] had observed that the function $\sum_{i=0}^{n-1} i d_i \bmod n$ was a suitable hash function in this case. This function is of course related to ours by a permutation. (The fact that we have forbidden transitions with $k = 1$ allows us to have a 0 coefficient when $i = 0$ and still obtain 2-local collisionlessness.) In fact, Graham and Sloane's function has a slightly smaller range (n values) than ours (p values, where p is a prime which will be slightly larger than n, the best case being $p = n+1$). This fact makes Graham and Sloane's function better for some purposes of combinatorial argument, but it is worse than our functions as hash functions, since by choice of different p we can find different hash functions which behave like highly "independent random" functions. Meanwhile, Graham and Sloane's functions with different moduli behave highly dependently.

Now, suppose in addition to these constraints on p, we also require that the numbers $1 \pm g^i \pm g^j$, where the $\pm$'s are independent and $1 \le i < j \le n$, be *nonzero.* (This property of p may be confirmed, if it is true, in $O(n \log n)$ steps, by computing and sorting g^i mod p.) In that case, by using the fact that numbers modulo p form a field, we see that automatically it will be the case that the numbers $g^q \pm g^r \pm g^s$, q, r, s unequal, will be nonzero, so that we obtain 3-local collisionlessness with $B = 2$.

Similarly, if we also require that the numbers $1 \pm g^i \pm g^j \pm g^\ell$, where the $\pm$'s are independent and $1 \le i < j < \ell \le n$, be nonzero, we get 4-local collisionlessness with $B = 2$. If we also require that the numbers $1 \pm g^i \pm g^j \pm g^\ell \pm g^\mu$, where the $\pm$'s are independent and $1 \le i < j < \ell < \mu \le n$, be nonzero, we get 5-local collisionlessness with $B = 2$. These properties of p may be confirmed, if they are true, in $O(n^2 \log n)$ steps by computing and sorting the $g^i \pm g^j$. And so on.

It is important to note that the *field* property of numbers modulo p is giving us one extra Hamming distance for free, which is why this hash function is better than some other one, and also allows us to find suitable hash functions and to check that they work, in a lot less work than one might (naively) have thought.

Returning again to the *constant weight binary* case, we observe that we get 5-local collisionlessness if g is a generator and the numbers g^i mod p, $i = 0 \ldots n-1$, form an "S_2-set." That is, if the sums $g^i + g^j$, $0 \le i < j < n$, are disjoint mod p. Or equivalently, if the numbers $1 + g^a$ mod p are disjoint from the numbers $g^b + g^c$ mod p, $1 \le a < n$, $1 \le b < c < n$. (A table of some of the best known small S_2-sets is given in [2].)

Similarly one gets 7-local collisionlessness in the constant weight binary case with if g is a generator and the numbers g^i mod p, $i = 0 \ldots n-1$, form an "S_3-set." That is, if the sums $g^i + g^j + g^\ell$, $0 \le i < j < \ell < n$, are disjoint mod p.

Finally, let us consider the ternary case with usual Hamming distance. The difference in the hash function between two words at Hamming distance 1 is either g^i or $2 \times g^i$; always nonzero. The difference in the hash function between two words at Hamming distance 2 is $\pm g^i \pm g^j$, $\pm 2g^i \pm g^j$, or $\pm 2g^i \pm 2g^j$, $i \ne j$, independent $\pm$'s. If we choose an odd prime p with $p > 2n$ and such that g is a generator modulo p, then $\pm g^i$, $i = 0 \ldots n-1$, $n \le (p-1)/2$, are also all *unequal* modulo p, so that the first and third of these cases cannot be zero. The nonvanishing of the middle case may also be assured by choosing an odd prime p with $p > 4n$, such that g is a generator modulo p, and such that if $\pm 2 = g^\ell$ mod p, then $|\ell| \ge n$. In that case we will achieve 2-local collisionlessness.

2.3 Applications to coding theory.

Let the coding-theory function $A(n, d)$ denote the largest number of n-bit binary words which can exist, such that any pair of them have Hamming distance $\ge d$. Similarly let $A_B(n, d)$, $B \ge 3$, denote the largest number of n-digit radix-B words which can exist, such that any pair of them have Hamming distance $\ge d$.

Using the hash function $\sum_{i=0}^{n-1} (i+1) d_i \bmod 2n$ and noting that the set of binary n-bit words x having any particular *constant* value of $h(x)$ must have minimal dis-

[1] The Go problem composers among our readers may wish to consider the question of whether it *is* possible. A solution is given at the end of the paper!

tance ≥ 3, we see that a distance-3 binary length-n code must exist with cardinality at least as large as the average size of these sets, namely

$$A(n,3) \geq 2^n/(2n). \tag{2}$$

The sphere-volume bound is

$$A(n,3) \leq 2^n/(n+1), \tag{3}$$

so we have come (asymptotically) within a factor of 2 of the upper bound. This result is not new, since a bound which is always as strong, and usually stronger, namely

$$A(n,3) \geq 2^{n-r}, \tag{4}$$

where r is the least integer with $2^r > n$, arises from considering the shortened binary Hamming codes [7]. This bound (EQ 4) is in fact *tight* if $2^r - n \leq 4$, as was shown by Best and Brouwer [1]. For the values of n (powers of 2) for which the present bound (EQ 2) equals the bound (EQ 4), in fact (EQ 4) can be improved by use of the "$u|u+v$" construction ([7] 2.9), so our result does not yield any records at all.

Nevertheless, let us persevere. We are going to prove the bound

$$A_B(n,3) \geq \lceil B^n/p \rceil, \tag{5}$$

where p is the least prime with $p > 2(B-1)n$ which obeys a certain property (EQ 12) described below. It is extremely plausible that $p < 2(B-1)n + o(n)$ for any fixed value of B as $n \to \infty$, although *this* is a number theoretic conjecture (see below) which we do not know how to prove. One may compare (EQ 5) with the sphere-volume upper bound

$$A_B(n,3) \leq \frac{B^n}{(B-1)n+1}. \tag{6}$$

If the conjecture is true, then the lower bound (EQ 5) comes asymptotically within a factor of 2 of the upper bound (EQ 6) as $n \to \infty$ with B fixed. It would also exceed the competing lower bound

$$A_B(n,3) \geq B^{n-r}, \tag{7}$$

where r is the least integer such that $B^r > (B-1)n$, arising from shortened radix-B Hamming codes, at an infinite number of n for each $B \geq 3$. In particular for all sufficiently large n of the form $(3^z+1)/2$ when $B=3$. (It should be pointed out that, until the present paper, *no* ternary distance-3 code was known which was better than a shortened Hamming code.) Also (EQ 7) is only valid if B is a prime power, but our bound is valid for all $B \geq 2$. Finally, we should mention the competing *Gilbert bound* [7]

$$A_B(n,3) \geq B^{n-\mu}, \tag{8}$$

where μ is the least integer such that

$$B^\mu \geq (B-1)(n-1)+1, \tag{9}$$

which is of course weaker than the bound from shortened Hamming codes, but has the advantage that it works for all $B \geq 2$, prime power or not.

The property that p and g must have for (EQ 5) to hold, is as follows:

$$\pm g^i a \pm g^j b \neq 0 \bmod p, \\ 0 \leq a,b < B, \quad ab \neq 0, \quad 0 \leq i < j < n. \tag{10}$$

This property of course assures 2-local collisionlessness. This is implied by the following property, which is easier to verify:

$$\pm a \pm g^j b \neq 0 \bmod p, \quad 0 \leq a,b < B, \quad ab \neq 0, \quad 1 \leq j < n. \tag{11}$$

This in turn will hold if the indices (discrete logarithms to base g mod p) of a, $\ell(a)$, differ by at least n for all $a = 1,2,\ldots,B-1$. In order for this to happen with g a generator mod p, it must be that $p > 2(B-1)n$. In fact, the following conjecture is extremely plausible: **Conjecture:** *There exists a suitable prime p with $p = 2(B-1)n + o(n)$.*

We now offer a plausibility argument to support this conjecture. A pair (p,g) will certainly be suitable if g is a generator such that for some choice of m_i, and some choice of ν, $0 < \nu < 1$, $0 \leq m_i < n^\nu$, and some choice of $\pm$ signs S_i, and some choice of a permutation ω of $\{1,2,\ldots,B-1\}$, we have

$$g^{in+m_i} = S_i \omega_i \bmod p, \quad i = 1,2,\ldots,B-1. \tag{12}$$

Suppose $p-1 = 2q$, q prime. Most number theorists believe that the gaps between such p's are $o(n)$.[2] Now, for such a p, the number of generators g modulo p is $p/2-1$. We have to choose g and m_i to satisfy $B-1$ congruences

$$G \times \ell(ni+m_i) = a_i \bmod (p-1), \quad i = 1,2,\ldots,B-1. \tag{13}$$

where $G = \ell(g)$, $\ell()$ represents the discrete logarithm to base g modulo p and the a_i's arise as discrete logs of $\pm\omega_i$. The cardinality of the set of G's satisfying one of these congruences is thus $\approx n^\nu/2$, and the cardinality of the set of G's satisfying all $B-1$ of these congruences (assuming the very plausible conjecture that the solutions of the congruences behave like "independent random" sets) is $\approx (1-n^{\nu-1})^{B-1} n/2$, which is of order n as $n \to \infty$ if $1 > (B-1)(1-\nu)$. Thus almost certainly, many suitable generators g will exist modulo any prime p with $p > 2(B-1)(n+n^\nu)$, if long as ν obeys $1 > (B-1)(1-\nu)$, $0 < \nu < 1$. (Of course, some of the sets do not behave "independently randomly," for example since $2 \times 2 = 4$, but such dependencies help, rather than hinder, our argument.) End of plausibility argument.

We also point out that the conjecture is very well supported by the numerical evidence. For example in the ternary case $B=3$, we have

[2] In fact $O((logn)^4)$, or $O(n^c)$ where $c = 1/2+\epsilon$, or or $O(n^c)$ where $1/2 < c < 1$, depending on how strong a conjecture that number theorist wants to make that day.

n= 2	p= 11	g= 3	p/n=5.500
n= 3	p= 13	g= 3	p/n=4.333
n= 4	p= 19	g= 4	p/n=4.750
n= 5	p= 23	g= 4	p/n=4.600
n= 7	p= 29	g= 4	p/n=4.143
n= 9	p= 37	g= 3	p/n=4.111
n= 11	p= 47	g= 4	p/n=4.273
n= 13	p= 53	g= 4	p/n=4.077

n= 14	p= 59	g= 4	p/n=4.214
n= 15	p= 61	g= 4	p/n=4.067
n= 16	p= 67	g= 4	p/n=4.187
n= 17	p= 71	g= 4	p/n=4.176
n= 19	p= 79	g= 4	p/n=4.158
n= 20	p= 83	g= 4	p/n=4.150

n= 43	p= 173	g= 4	p/n=4.023
n= 57	p= 229	g= 3	p/n=4.018
n= 60	p= 251	g= 30	p/n=4.183

n= 122	p= 491	g= 4	p/n=4.025
n= 180	p= 727	g= 174	p/n=4.039
n= 183	p= 733	g= 3	p/n=4.005

n= 366	p= 1471	g= 265	p/n=4.019
n= 543	p= 2179	g= 21	p/n=4.013
n= 551	p= 2207	g= 4	p/n=4.005

n= 1097	p= 4391	g= 4	p/n=4.003
n= 1637	p= 6551	g= 4	p/n=4.002
n= 1640	p= 6571	g= 80	p/n=4.007

n= 3281	p= 13127	g= 4	p/n=4.001
n= 4915	p= 19661	g= 4	p/n=4.000
n= 4923	p= 19699	g= 824	p/n=4.001

with the ratio p/n unquestionably approaching 4. The cases n = 14-19, n = 42-57, n = 122-180, n = 365-543, n = 1094-1637, n = 3281-4915, each lead to new record ternary distance-3 codes with n trits/word and $\geq 3^n/p$ words. The first few of these are

$$A_3(14,3) \geq 81068, \quad A_3(15,3) \geq 235228, \tag{14}$$
$$A_3(16,3) \geq 642489, \quad A_3(17,3) \geq 1818876, \tag{15}$$
$$A_3(18,3) \geq 4904057, \quad A_3(19,3) \geq 14712171. \tag{16}$$

2.4 Tables

In this section we tabulate a few examples of k-locally collisionless hash functions, so that a programmer can avoid the effort of searching for suitable p and g himself (in the few small instances that we have listed, anyway!) and merely use the tabulated values. We have not bothered to tabulate cases with $k \leq 2$ since they are so easy to find.

The $k = 3$ case for unconstrained binary words:

n= 10	p= 89	g= 44	p/n^1.5=2.814
n= 20	p= 239	g= 13	p/n^1.5=2.672
n= 30	p= 433	g=103	p/n^1.5=2.635
n= 54	p= 881	g=440	p/n^1.5=2.220
n= 100	p= 2719	g=240	p/n^1.5=2.719
n= 200	p= 6793	g= 50	p/n^1.5=2.402
n= 493	p= 29581	g= 29	p/n^1.5=2.702
n= 545	p= 33791	g= 85	p/n^1.5=2.656
n= 1000	p= 120121	g= 34	p/n^1.5=3.799

The case $k = 4$ for unconstrained binary words:

n= 10	p= 179	g= 89	p/n^2=1.790
n= 19	p= 739	g= 299	p/n^2=2.047
n= 32	p= 3257	g= 781	p/n^2=3.181
n= 40	p= 6323	g= 562	p/n^2=3.952
n= 59	p= 18583	g= 400	p/n^2=5.338
n= 70	p= 27737	g= 927	p/n^2=5.661
n= 91	p= 51593	g= 550	p/n^2=6.230
n= 100	p= 84521	g= 614	p/n^2=8.452
n= 102	p= 86627	g= 270	p/n^2=8.326
n= 116	p= 92569	g= 824	p/n^2=6.879

The case $k = 5$ for unconstrained binary words:

n= 10	p= 617	g= 245	p/n^2.5=1.951
n= 20	p= 6907	g= 841	p/n^2.5=3.861
n= 29	p= 25579	g=1017	p/n^2.5=5.648
n= 34	p= 48757	g= 594	p/n^2.5=7.233
n= 37	p= 72767	g= 659	p/n^2.5=8.738
n= 42	p= 99529	g= 869	p/n^2.5=8.706
n= 46	p= 117577	g= 596	p/n^2.5=8.193

The case of "S2 sets:" $k = 5$ for constant weight binary words:

n= 10	p= 103	g= 51	p/n^2=1.030
n= 20	p= 659	g= 71	p/n^2=1.647
n= 29	p= 1601	g= 471	p/n^2=1.904
n= 49	p= 3823	g= 49	p/n^2=1.592
n= 65	p= 12037	g= 170	p/n^2=2.849
n= 87	p= 26449	g= 174	p/n^2=3.494
n= 111	p= 38629	g= 179	p/n^2=3.135
n= 115	p= 52901	g= 171	p/n^2=4.000
n= 165	p= 111871	g= 205	p/n^2=4.109

The experimental evidence in these tables, as well as much more extensive tables not printed here, seems to support the assertion that these hash functions are only "a constant factor worse than perfect."

That is: *any* k-locally collisionless hash function must assume at least H possible values, where H is the minimum number of colors needed to color the graph of radix-B length-n words where two words (vertices) are joined by an edge if their Hamming distance is $\leq k$. Just because of the presence of cliques, one can easily see that $H = \Omega(n^{k/2})$ for any fixed $k > 1$, if k is even. In fact, it seems, experimentally, based on the tables above, that we may *succeed* in finding k-locally collisionaless hash functions with $p = O(n^{k/2})$, or at least, with $p = O(n^{k/2} f(n))$ where $f(n)$ is certainly a slowly growing function, if it is not bounded above by a constant[3].

[3]A solution to the Go problem in a previous footnote. (Eric B.

References

[1] M.R. Best and A.E. Brouwer: *Triply shortened binary Hamming code is optimal,* Discrete Math. 17 (1977) 235-245

[2] A.E. Brouwer, J.B. Shearer, N.J.A. Sloane, W.D. Smith: *A new table of constant weight codes,* IEEE Trans. Info. Theory IT-36,6 (Nov 1990) 1334-1380

[3] H.C. Ferreira, J.F. Hope, A.L. Nel: *On ternary error-correcting line codes,* IEEE Trans. Communications, 37,5 (May 1989) 510-515

[4] M.R. Garey and D.S. Johnson: *NP completeness,* Freeman 1977

[5] F. Glover: *TABU search,* Oper. Research Soc. of America J. Comput. 1 (1989) 190-206

[6] R.L. Graham and N.J.A. Sloane: *Lower bounds for constant weight codes,* IEEE Trans. Info. Theory IT-26 (1980) 37-43

[7] F.J. MacWilliams and N.J.A. Sloane: *Theory of Error correcting codes,* North-Holland, 1977.

[8] James Singer: *A theorem in finite projective geometry and some applications to number theory,* Trans. AMS 43 (1938) 377-385.

Baum.) Start with white stone at corner (A1) and board blank. While white continually passes (if you don't like passes, I think you can generalize easily) black fills in the whole board except another corner (A19), capturing the white stone en route and playing in A1 himself. Then white plays in A19 (capturing all the black men) and proceeds to fill in the whole board except A1, whereupon Black plays in A1. (This is possibly the worst played Go game ever conceived of...) Baum points out that the same ploy will suffice to interconvert a very wide class of Go positions into each other.