Finding Minimal Perfect Hash Functions

Kevin Karplus*
Gary Haggard$

September 1984 TR 84-637

Computer Science Department
Cornell University
Ithaca, NY 14853

* Computer Science Department
  and School of Electrical Engineering,
  Cornell University

$ Computer Science Department
University of Maine at Orono
(work done while visiting Cornell)
ABSTRACT

A heuristic is given for finding minimal perfect hash functions without extensive searching. The procedure is to construct a set of graph (or hypergraph) models for the dictionary, then choose one of the models for use in constructing the minimal perfect hashing function. The construction of this function relies on a backtracking algorithm for numbering the vertices of the graph. Careful selection of the graph model limits the time spent searching. Good results have been obtained for dictionaries of up to 181 words. Using the same techniques, non-minimal perfect hash functions have been found for sets of up to 667 words.

Introduction

A hash function is a mapping from a set of keys $K$ to $n$ consecutive integers (usually $[0, n - 1]$ or $[1, n]$). If the mapping is one-to-one, the function is a perfect hash function. If the mapping is one-to-one and onto ($|K| = n$), the mapping is a minimal perfect hash function. In this paper, we only examine cases where the keys are words (that is, short sequences of alphabetic characters), but the techniques can be applied to other kinds of keys as well.

Hash functions are used for fast searching of sets of keys. Hashing algorithms provide constant time searches on the average, but $O(|K|)$ worst-case behavior. Using a perfect hash function allows constant-time performance in the worst case. However, perfect hash functions can only be found when the set of keys is fixed.

Minimal perfect hash functions offer a slight storage savings over arbitrary perfect hash functions, and make the search more interesting. The techniques given in this paper can be used to generate both minimal and non-minimal perfect hash functions.
Previous results

Ordinary hash functions are cheap to compute, and families of good hash functions have been described in the literature [CW]. A tempting technique is to try arbitrary hash functions until a perfect one has been found. This approach was investigated in [S] for non-minimal perfect hash functions, but only small sets were considered. If we map \( m \) keys to \( n \) integers, there are \( n^m \) hash functions, of which only \( n!/(n-m)! \) are perfect. The table below gives the density of perfect hash functions for small values of \( m \) and \( n \). Even for fairly small sets of words, perfect hash functions are rare, and minimal perfect hash functions are very rare indeed. Finding perfect hash functions by generating random good hash functions is only feasible for very small sets, or for moderately small sets and much larger hash ranges.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( m )</th>
<th>( 10 )</th>
<th>( 20 )</th>
<th>( 30 )</th>
<th>( 40 )</th>
<th>( 50 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>65.5 ( 10^{-3} )</td>
<td>23.2 ( 10^{-3} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>293. ( 10^{-3} )</td>
<td>3.05 ( 10^{-3} )</td>
<td>195. ( 10^{-9} )</td>
<td>67.5 ( 10^{-18} )</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>30</td>
<td>452. ( 10^{-3} )</td>
<td>27.9 ( 10^{-3} )</td>
<td>142. ( 10^{-6} )</td>
<td>25.6 ( 10^{-9} )</td>
<td>28.4 ( 10^{-15} )</td>
</tr>
<tr>
<td>80</td>
<td>40</td>
<td>556. ( 10^{-3} )</td>
<td>74.6 ( 10^{-3} )</td>
<td>1.90 ( 10^{-3} )</td>
<td>6.60 ( 10^{-6} )</td>
<td>1.89 ( 10^{-9} )</td>
</tr>
<tr>
<td>100</td>
<td>50</td>
<td>628. ( 10^{-3} )</td>
<td>130. ( 10^{-3} )</td>
<td>7.79 ( 10^{-3} )</td>
<td>112. ( 10^{-6} )</td>
<td>307. ( 10^{-9} )</td>
</tr>
<tr>
<td>120</td>
<td>60</td>
<td>680. ( 10^{-3} )</td>
<td>187. ( 10^{-3} )</td>
<td>18.97 ( 10^{-3} )</td>
<td>636. ( 10^{-6} )</td>
<td>6.14 ( 10^{-6} )</td>
</tr>
</tbody>
</table>

In [C] a method for finding minimal perfect hash functions was presented. Only functions of the following form were considered:

\[ h(\text{key}) = g(\text{first letter}) + g(\text{last letter}) + \text{length(\text{key})} \]

where \( g(\text{letter}) \) is computed by table lookup. The technique consisted of ordering the words of the dictionary, then doing a backtracking search to assign appropriate values to the letters. The method works well for small dictionaries, but fails for larger ones.

In this paper, we will investigate a family of functions which includes the form used in [C] as a special case. Minimal perfect hash functions can be found in reasonable amounts of time for much larger sets of words than previously reported.

Word graphs (and hypergraphs)

The first and last letters of a word are not always a good choice for hashing functions. In [C], the set of PASCAL pre-defined identifiers was used as an example only after eliminating ODD, since the pair ODD and ORD could not be distinguished. For three letter month abbreviations, the second and third
letter were used, to distinguish JAN and JUN. With a general set of words, it is not clear \textit{a priori} how to pick a pair of letter positions to use for generating a perfect hash function. We consider several candidate letter positions, and choose the smallest $k$-tuple of positions likely to yield a perfect hash function. Upper and lower case letters are considered equivalent, and all other characters (such as apostrophes and hyphens) are removed from the word before hashing. The hash functions themselves are of the form:

$$h(\text{word}) = g_1(\text{letter}) + \ldots + g_k(\text{letter}) + \text{length(\text{word})}$$

where each function $g_i$ is a table lookup for a different letter position. The class of hash functions is reduced to the hash functions generated in [C] if $k$ is two, if the positions chosen are the first and last letters, and if the two functions $g_1$ and $g_2$ are identical.

In the program implementing the search for perfect hash functions, positions of letters in a word of length $L$ are numbered from 0 to $L-1$. The candidate positions tried by the program are generated by the following functions: $0$, $(L-1) \mod 2$, $(L-1) \mod 3$, $(L-1) \mod 4$, $(L-1) \mod 5$, $1$, $2$, $3$, $4$, $L-2$, $L-3$, $L-4$, $L-5$, $2 \mod L$, $3 \mod L$, $4 \mod L$, $5 \mod L$, $6 \mod L$, $7 \mod L$, $8 \mod L$, $L/2$, $L/3$, $2L/3$, $L/4$, and $3L/4$, where $x \mod y$ means the remainder when $x$ is divided by $y$, and all the divisions truncate non-integers. If the computed position is less than zero, then zero is used, and if it is $L$ or larger, $L-1$ is used.

When the dictionary has only a few different lengths of words, then several of the functions for generating candidate positions generate the same positions. In such a case, only the first of the equivalent functions is used.

Definition: Let $K$ be a set of words, $w$ be a member of $K$, and $w_p$ be the letter at position $p$. The word (hyper-)graph $G(p_1, \ldots, p_k)$ over $K$ is the $k$-partite hypergraph formed as follows:

The vertex set $V$ is the disjoint union of $V_{pi}$, where $V_p = \{\text{letters in words of } K \text{ occurring at position } p\}$.

Each (hyper-)edge is labeled with a word of $K$, has $k$ vertices, and has $w_{pi}$ as its vertex in $V_{pi}$.

The \textit{dimension} of the word hypergraph is the number of vertices in each hyperedge. A hypergraph of dimension two is an ordinary graph, and hypergraphs of degree zero and one are possible.
A hash function can be defined by labeling each vertex with a number, the hash value for a word being the sum of the labels on the vertices of the hyperedge plus the length of the word. The functions $g_i$ (letter) are undefined for letters not in the graph. Arbitrarily assigning them a large value makes it probable that words not in the dictionary have hash values outside the image of $K$.

As an example of a word graph, consider $K$={JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC). Here is the graph $G(1,2)$ (position numbering starts with 0, so $G(1,2)$ uses the second and third letters):

![Graph](image)

A word graph can have several edges with the same pairs of vertices. For example, in $G(0,2)$ for the month names there are two edges J-N, corresponding to the words JAN and JUN. A perfect hash function is only possible in this scheme if all edges with the same vertices correspond to words of different lengths.

The steps in finding a minimal perfect hash function are: 1) generate a collection of word graphs (or hypergraphs), 2) choose one which looks promising, and 3) assign values to the vertices of the graph in such a ways as to assign distinct values to each edge.

Edges incident on vertices of degree one can be assigned any desired hash value, since the vertex can be assigned a value independent of any other vertex value. Thus the vertex assignment problem can be simplified somewhat by deleting all edges containing a vertex unique to that edge. The reduced graph may have new vertices of degree one, allowing more edges to be deleted. Repeating this process eventually results in a graph with no vertices of degree one. The edges removed are referred to as tree edges, and are assigned hash values in reverse order of their removal, after all the edges in the reduced hash graph have been assigned values.

In $G(1,2)$ for month name abbreviations, all the edges are tree edges, so a perfect hash function can be found immediately. The tree edges could be
removed in the order: OCT, NOV, DEC, SEP, FEB, AUG, JUL, MAY, APR, MAR, JUN, JAN. Arbitrarily choosing JAN=1, FEB=2, ..., DEC=12, we can assign vertex values as follows:

<table>
<thead>
<tr>
<th>$V_1$</th>
<th>$V_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=2</td>
<td>N=0</td>
</tr>
<tr>
<td>U=3</td>
<td></td>
</tr>
<tr>
<td>R=2</td>
<td></td>
</tr>
<tr>
<td>P=1</td>
<td></td>
</tr>
<tr>
<td>Y=4</td>
<td></td>
</tr>
<tr>
<td>L=1</td>
<td></td>
</tr>
<tr>
<td>G=2</td>
<td></td>
</tr>
<tr>
<td>E=1</td>
<td>B=0</td>
</tr>
<tr>
<td>P=7</td>
<td></td>
</tr>
<tr>
<td>C=10</td>
<td></td>
</tr>
<tr>
<td>O=8</td>
<td>V=0</td>
</tr>
<tr>
<td>C=7</td>
<td>T=0</td>
</tr>
</tbody>
</table>
Choosing a word graph

The most important criterion for choosing a word graph is that it must allow a perfect hash function. A necessary, but not sufficient, condition is that all edges with the same vertices correspond to words of different lengths. Since tree edges make finding a minimal perfect hash function much easier, an obvious second criterion for a good word graph is that it maximize the number of tree edges.

Vertex value assignments are easier if the effects of an assignment are fairly local in the graph. The program does not check connectivity of the graph, but tries to maximize the number of vertices remaining after tree edges have been removed. By maximizing the number of tree edges and the number of vertices remaining, the total number of vertices is kept high and the average degree low.

There is only one possible graph with dimension zero. It can form a perfect hash function only if all the word lengths are different, and a minimal perfect hash function only if the lengths are consecutive. It is very rare that a minimal perfect hash function can be found so easily.

Each candidate position function generates a graph of dimension one. The program generates all these graphs, counting their vertices and tree-edges. In the examples tried, only the 22 word random set (r022) had a graph of dimension one in which all words mapped to distinct edges.

The program could continue generating all graphs of dimension 2, 3, and so on until a graph in which all words are distinct edges is found. Unfortunately, the process of building a graph and counting the tree-edges is fairly expensive. A heuristic was introduced to generate only graphs that are likely to have all edges distinct and to have many tree-edges. The best few (four in the program) position tuples of dimension $k-1$ are kept, and all possible extensions of them to $k$ tuples are tried. The "best" tuple is defined as the weighted sum of the total number of edges, the number of tree-edges, and the number of vertices.

Nine experiments were run to try tuning the weights. The program tried to find hash functions for seventeen sets of words (through r181 in the table below). In every case a graph which distinguished the words was found, but the search for a hash function was interrupted if it took too long. Some larger sets were also tried, and graphs were found, but no minimal perfect
hash functions. A weighting was considered better for a particular dictionary if it generated a different graph and took less total time in the search for a hash function. For successful searches, most of the time was spent building graphs and counting tree-edges. For unsuccessful searches, most of the time was spent trying to find a vertex value assignment.

<table>
<thead>
<tr>
<th>Edge weight</th>
<th>Tree-edge weight</th>
<th>Vertex weight</th>
<th>comparison with default</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
<td>(this is the default)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>(ibm &amp; r139 worse)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>(r066, r124, &amp; r162 worse)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(r115, r162 better. r124, r181 worse)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(most sets worse)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(most sets worse)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>(r066 better. r162 worse)</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1</td>
<td>=</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>(r124 better. ibm, pasc-all worse)</td>
</tr>
</tbody>
</table>

From these experiments, it seems that the number of tree-edges is the best predictor of which tuples to expand, and that the total number of edges is more important than the number of vertices. Below is a more detailed table of the graphs chosen for the default weighting, including some larger sets that were not used in the tuning. The table gives: the name of the set, the number of words in the set, the number of tree edges found in the hypergraph chosen, the number of vertices left after removing the tree edges, the excess space allowed for the hash function (and the amount actually used) expressed as a fraction of the number of words, and the position functions used to generate the hypergraph.

<table>
<thead>
<tr>
<th>set</th>
<th>words</th>
<th>tree</th>
<th>other</th>
<th>excess</th>
<th>position functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>edges</td>
<td>vertices</td>
<td>excess</td>
<td>position functions</td>
<td></td>
</tr>
<tr>
<td>r022</td>
<td>22</td>
<td>11&lt;50%</td>
<td>5</td>
<td>0.00&lt;0.00</td>
<td>2L/3</td>
</tr>
<tr>
<td>c</td>
<td>28</td>
<td>22&lt;79%</td>
<td>6</td>
<td>0.00&lt;0.00</td>
<td>4</td>
</tr>
<tr>
<td>freq</td>
<td>29</td>
<td>16&lt;55%</td>
<td>9</td>
<td>0.00&lt;0.00</td>
<td>4 mod L</td>
</tr>
<tr>
<td>pasc</td>
<td>36</td>
<td>23&lt;54%</td>
<td>10</td>
<td>0.00&lt;0.00</td>
<td>(L-1) mod 4</td>
</tr>
<tr>
<td>pasc-id</td>
<td>40</td>
<td>28&lt;70%</td>
<td>8</td>
<td>0.00&lt;0.00</td>
<td>(L-1) mod 2</td>
</tr>
<tr>
<td>r066</td>
<td>66</td>
<td>19&lt;29%</td>
<td>25</td>
<td>0.00&lt;0.00</td>
<td>3</td>
</tr>
<tr>
<td>pasc-all</td>
<td>76</td>
<td>29&lt;38%</td>
<td>31</td>
<td>0.00&lt;0.00</td>
<td>(L-1) mod 5</td>
</tr>
<tr>
<td>ibm</td>
<td>86</td>
<td>15&lt;17%</td>
<td>33</td>
<td>0.00&lt;0.00</td>
<td>L/2</td>
</tr>
<tr>
<td>r069</td>
<td>89</td>
<td>10&lt;11%</td>
<td>33</td>
<td>0.00&lt;0.00</td>
<td>L-1</td>
</tr>
<tr>
<td>r092</td>
<td>92</td>
<td>24&lt;26%</td>
<td>40</td>
<td>0.00&lt;0.00</td>
<td>L-1</td>
</tr>
<tr>
<td>r105</td>
<td>105</td>
<td>12&lt;11%</td>
<td>35</td>
<td>0.00&lt;0.00</td>
<td>L-1</td>
</tr>
<tr>
<td>r115</td>
<td>115</td>
<td>8&lt;7%</td>
<td>38</td>
<td>0.00&lt;0.00</td>
<td>L-1</td>
</tr>
<tr>
<td>r117</td>
<td>117</td>
<td>21&lt;16%</td>
<td>44</td>
<td>0.00&lt;0.00</td>
<td>L-1</td>
</tr>
<tr>
<td>r124</td>
<td>124</td>
<td>14&lt;11%</td>
<td>51</td>
<td>0.00&lt;0.00</td>
<td>L-1</td>
</tr>
<tr>
<td>set</td>
<td>words</td>
<td>tree</td>
<td>other</td>
<td>excess</td>
<td>position functions</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>edges  vertices space</td>
</tr>
<tr>
<td>r139</td>
<td>139 13(6%)</td>
<td>51</td>
<td>0.00(0.00)</td>
<td>0</td>
<td>L-1 L-3</td>
</tr>
<tr>
<td>r162</td>
<td>162 14(6%)</td>
<td>55</td>
<td>0.00(0.00)</td>
<td>(L-1) mod 3</td>
<td>L/4 3L/4</td>
</tr>
<tr>
<td>r181</td>
<td>181 11(6%)</td>
<td>58</td>
<td>0.00(0.00)</td>
<td>L-1</td>
<td>(L-1) mod 2 3</td>
</tr>
<tr>
<td>r192</td>
<td>192 9(5%)</td>
<td>62</td>
<td>0.00(0.00)</td>
<td>(L-1) mod 2 3</td>
<td>L-4</td>
</tr>
<tr>
<td>r192</td>
<td></td>
<td></td>
<td>0.10( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r192</td>
<td></td>
<td></td>
<td>0.20(0.19)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r192</td>
<td></td>
<td></td>
<td>0.30(0.23)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r192</td>
<td></td>
<td></td>
<td>0.40(0.396)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r236</td>
<td>236 9(4%)</td>
<td>63</td>
<td>1.50(0.22)</td>
<td>0</td>
<td>2 3L/4</td>
</tr>
<tr>
<td>r373</td>
<td>373 13(3%)</td>
<td>65</td>
<td>0.70(0.79)</td>
<td>(L-1) mod 3</td>
<td>(L-1) mod 5</td>
</tr>
<tr>
<td>r373</td>
<td></td>
<td></td>
<td>1.50(0.79)</td>
<td>L-2</td>
<td>L-5</td>
</tr>
<tr>
<td>r495</td>
<td>495 8(2%)</td>
<td>88</td>
<td>2.00(0.49)</td>
<td>/</td>
<td>0 L-1 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\ 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r667</td>
<td>667 9(1%)</td>
<td>117</td>
<td>2.00(0.94)</td>
<td>/</td>
<td>0 L-1 4 mod L</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\ L/2</td>
<td></td>
<td>L/3</td>
</tr>
</tbody>
</table>

One larger set (r992) was tried, but no hypergraphs of dimension 5 or less were found with distinct edges for all the words of the set. Because of some poor early decisions in implementation, the program would require major restructuring to handle hypergraphs of dimension six or more.

For small sets, two position functions usually suffice to form graphs with no duplicate edges. However, note that the set pasc-all, with only 76 words, needs three position functions to avoid duplication. In fact, the seven words (CASE, ELSE, PAGE, READ, REAL, TRUE, TYPE) contained in pasc-all require three position functions by themselves. (Consider all possible pairs of positions for four-letter words: 0&1, 0&2, 1&2 READ=REAL; 0&3 TRUE=TYPE; 1&3 CASE=PAGE; 2&3 CASE=ELSE.) Some of the larger sets (for which no minimal perfect hash function has yet been found) apparently require four, five, or more position functions to avoid duplicate edges.

**Assigning vertex values**

After the position functions have been chosen and the word hypergraph built, values have to be assigned to all vertices. The tree-edges can be removed and the associated vertices assigned values in reverse order after the rest of the vertices have values. For the main body of the graph, the vertex assignment proceeds as follows:

1) choose a vertex
2) if there is no legal assignment, backtrack and change a previous choice
3) otherwise, assign the smallest legal value to the vertex (or 0, if the vertex value is unconstrained)
4) go back to 1 until all vertices assigned.
The word hypergraph is stored redundantly, so that the edges incident on a particular vertex or the vertices in a particular edge can be quickly found. A number of auxiliary data structures are used to speed up the vertex-value assignment. For example, keeping the set of unused edge values makes it easier to determine what the legal values for a vertex are. Also, keeping a stack of the value assignments makes backtracking easier.

The simplest scheme would be to have a fixed ordering for the vertices, and to do simple backtracking (undoing the most recent choice) when a conflict is found. This scheme worked well for the small assignments of [C], but can take a very long time on larger, more complicated graphs. Various enhancements were tried to speed up the vertex-assignment search. They can be divided into two classes: vertex-choice heuristics and backtracking shortcuts.

The vertex-choice heuristics attempt to speed things up by choosing the most difficult vertices first. Not only does this reduce backtracking by making conflicts appear sooner, it also reduces the number of conflicts, since the difficult vertices are assigned while there are still many unused edge values. The backtracking shortcuts attempt to speed up the search by eliminating combinations of assignments that can not work. For example, if two edges incident on the current vertex have the same edge value, only changes to one of the other vertices of the edges will break the conflict. A simple backtracking scheme could waste a long time undoing and retrying more recent assignments that can't affect the conflict.
**Vertex-choice heuristics**

The vertex-choice heuristics used are fairly simple. First, find the set of edges with the fewest unassigned vertices (not counting edges with all vertices assigned values). Call this set $E_{\text{min}}$. Next, choose the best vertex in one of these edges. "Best" can be defined in one of three ways:

- Most) choose the vertex with the most edges in $E_{\text{min}}$.
- Widest) choose the vertex with the widest range of partial sums for edges in $E_{\text{min}}$.
- Most & Widest) of the vertices with the most edges in $E_{\text{min}}$ choose the vertex with the widest range of partial sums.

Of the three, "Most & Widest" works best. This vertex ordering scheme works well for hypergraphs of dimension one and two, but not as well for higher dimensions. A more sophisticated vertex-choice scheme may be needed to handle dictionaries of more than one or two hundred words efficiently.

**Backtracking heuristics**

The backtracking heuristics are more complicated than the vertex-choice ones. Successful hash function searches backtrack rarely, and spend more time building graphs and counting tree edges than doing the vertex assignment. Failing hash function searches spend almost all their time attempting vertex-value assignments. More intelligent backtracking heuristics could increase the size of dictionary that the program could handle.

It is difficult to report on the exact state of the backtracking heuristics, since they were written incrementally. After each modification, traces were done to look for patterns in the backtracking, and changes made to reduce the most obvious wasted effort. Changes that made matters worse were either undone or fixed with an additional heuristic. Since the heuristics interact, it is difficult to assign credit to any particular heuristic.

Currently, the only edges that affect the vertex value are those which have only the chosen vertex unassigned. We call these the *almost completed edges*. If there are no such edges, the vertex is assigned an arbitrary value and another vertex is chosen. The program offers the choice of assigning either zero or the smallest available edge weight divided by the number of vertices still unassigned in the edge to each such free vertex. Although
different hash functions are found in each case, the search time is not affected much by this option.

There are three different conflicts that can cause backtracking:
- **edge conflict** two different almost completed edges have the same partial sum.
- **too big** the range of partial sums for almost completed edges is larger than the range of unused edge values.
- **no fit** every vertex assignment will make the almost completed edges have values that are unused by other edges.

Each type of conflict triggers a different backtracking method.

For edge conflicts, vertex values are unassigned (and popped off the backtrack stack) until the partial sums of the edges in conflict differ. It is not enough to pop vertices until a vertex of one of the edges is removed, as that vertex may be included in both edges, so changing its value would not remove the conflict. The program proceeds forward again, trying to find a larger value for the last vertex popped off the stack.

For "too-big" conflicts, vertices are popped until
1) the last vertex popped is in the almost completed edge with the smallest partial sum, but not in the one with the highest partial sum,
   or 2) a vertex assignment can be made so the partial sums of the almost completed edges will fit.

In the second case, one extra vertex needs to be popped to prevent the search from entering an infinite loop. The loop does occur sometimes if this precaution is not taken. What happens is that two sets of edges are alternately put into the same space, and undoing one set of assignments leaves just enough room to do the other set, but not to do both. The extra vertex popping ensures progress, but is risky, in that a solution to the vertex assignment problem may be skipped. This is not very dangerous, since solutions have to be fairly common for the search technique to have any chance of being efficient.

The vertex that triggered the "too-big" conflict and other popped vertices of the almost completed edge with the highest partial sum are added to a set of vertices to try immediately. When there are vertices in that set, the vertex choice heuristics are disabled and a vertex is taken from the set instead. This shortcut reduces the backtracking enormously, since extraneous assignments are not tried until the known conflict has been resolved.
For "no-fit" conflicts, vertices are popped until some vertex of an almost completed edge has been popped. A slight change was added to speed the search. The backtracking is not stopped by vertices of the almost completed edge with the highest partial sum. Since reassigning them raises the partial sum of the highest edge, it is unlikely to result in the conflict being resolved. Before this change was made, several "no-fit" conflicts occurred together, followed by a "too-big" conflict when the partial sums got far enough apart. The conflict-triggering vertex and any popped vertices of the highest edge are added to the set of vertices to try next.

Optionally, the backtracking from a "no-fit" conflict can be stopped when an assignment is possible for the vertex triggering the conflict. This option was not turned on for the tests reported here, as preliminary tests showed that it increased rather than reduced the backtracking.

Here is a summary of the types of conflicts for each set (using the position functions of the earlier table):

<table>
<thead>
<tr>
<th>word set</th>
<th>vertices changed</th>
<th>edge conflicts</th>
<th>too-big conflicts</th>
<th>no-fit conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>r022</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>freq</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pasc</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pasc-id</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r066</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pasc-all</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibm</td>
<td>40</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>r089</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r092</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r105</td>
<td>59</td>
<td>3</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>r115</td>
<td>106</td>
<td>3</td>
<td>1</td>
<td>36</td>
</tr>
<tr>
<td>r117</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>r124</td>
<td>221</td>
<td>10</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>r139</td>
<td>137</td>
<td>20</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>r162</td>
<td>93</td>
<td>1</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>r181</td>
<td>223</td>
<td>8</td>
<td>2</td>
<td>104</td>
</tr>
<tr>
<td>r192</td>
<td>394</td>
<td>12</td>
<td>4</td>
<td>167</td>
</tr>
<tr>
<td>r192</td>
<td>751</td>
<td>30</td>
<td>6</td>
<td>313</td>
</tr>
<tr>
<td>r192</td>
<td>180</td>
<td>5</td>
<td>1</td>
<td>107</td>
</tr>
<tr>
<td>r192</td>
<td>58</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>word set</td>
<td>vertices</td>
<td>edge changed</td>
<td>edge conflicts</td>
<td>too-big conflicts</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>--------------</td>
<td>----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>r192</td>
<td>.4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>r236</td>
<td>1.5</td>
<td>44</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>r373</td>
<td>.7</td>
<td>292</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>r373</td>
<td>1.5</td>
<td>92</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>r495</td>
<td>2.0</td>
<td>36</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>r667</td>
<td>2.0</td>
<td>372</td>
<td>39</td>
<td>0</td>
</tr>
</tbody>
</table>

It is interesting that it is much easier to find perfect hash functions for the sets handled in [C] with the new method. The month names example is all tree-edges, so can be trivially done by hand. Two sets (pasc and pasc-id) require no backtracking, and freq only backtracks once, so all three could have been done by hand. As a matter of record, both freq and pasc were successfully done by hand to determine the effectiveness of the tree-edge removal before the search program was written.

**Timing analysis**

The vertex-assignment search is potentially an exponential search. It usually finishes quickly for small sets, and may run for long times without finishing on slightly larger ones. The worst-case running time is too large to be practical, and average-case complexity is difficult to determine without a model of the data. We have to be content with empirical timing results.

The searches that were allowed to run to completion spent most of their time building word graphs and counting tree-edges in the search for an acceptable word graph. Building a word graph and counting the tree-edges should take time roughly proportional to the number of words in the set. The number of word graphs built depends on the dimension of the final hypergraph (the number of position functions needed to get a graph with all edges distinct). The dimension increases with the size of the word set, but we have not determined the exact relationship.

The search program was written in C and run on a VAX 11/780 under UNIX 4.2bsd. (The machine has a floating point accelerator, but floating point was not used in the program.) A log-log plot of the CPU time spent finding a perfect hash function versus the size of the set shows little scatter. Doing linear regression on the logarithms gives us the following empirical estimate of performance:

\[
\text{CPU seconds} = .059 \text{ words}^{1.5}
\]
Future work

Non-minimal perfect hash functions for large word sets are fairly easily found. It would be interesting to analyze the dependence of the hypergraph dimension on the size of the word set, and the amount of extra room needed to find a perfect hash function quickly.

Since the searching algorithm currently used is potentially exponential, it would be satisfying to know that there is no fast algorithm for finding minimal perfect hash functions. We conjecture that finding minimal perfect hash functions for sets of \( n \) words is NP-complete, but finding perfect hash functions with enough extra space can be done in polynomial time.

It might be worthwhile to investigate other classes of functions. As an example, word length could be just another dimension to the word hypergraph, and we could look for hash functions of the form

\[
    h(\text{word}) = g_0(\text{length}) + g_1(\text{letter}) + \ldots + g_k(\text{letter}).
\]
References


Appendix: word sets

The sets of words used for hashing are listed here. The sets freq, pasc, and 
pasc-id are from [C] (except that ODD and ORD are both in pasc-id). The 
set pasc-all is the union of pasc and pasc-id. The set c is the keywords 
and pre-declared identifiers of the c programming language. The set ibm 
is a subset of the IBM-360 instruction set. Those sets named r<number> 
were chosen randomly from a large spelling list. No distinction is made 
between upper- and lower-case letters. Apostrophes, hyphens, and other 
non-alphabetic characters are ignored.

c.words

auto break case char continue default do double else entry extern float for goto if int long register return 
short sizeof static struct switch typedef union unsigned while

freq.words

a and are as at be but by for from had have he her his i in is it not of on or the this was which with

ibm.words

a ah al alr ar bal balr bc bcr bcstr bxh bxle c ch cl cle clic clcr cr cvb cvd d dr ex hio ic i la lcr lh ln lnr lpr 
lpsw lr ltr m mh mr mvc mvi mvn mvo mvz n nc nl nr o oc oi or pack s sh sio sli sls slida slidsl sil silr spm sr 
ssr ssrs srsl ssld ssldsl ssldslr ssm sr st stc sth stm svm tch tie tm tr trt ts unpk x xc xi xr

pasc-all.words

abs and arctan array begin boolean case char chr const cos dispose div do downto else end eof eoln exp 
false file for function get goto if in input integer label in maxint mod new nil not odd of or ord otherwise 
output pack packed page pred procedure program put read readln real record repeat reset rewrite round 
set sin sgr sqrt succ text then to true trunc type unpack until var while with writeln

pasc-id.words

abs arctan boolean char chr cos dispose eof eoln exp false get integer in maxint new odd ord output 
pack page pred put read readln real record repeat rewrite round sin sgr sqrt succ text true trunc unpack write 
writeln

pasc.words

and array begin case const div do downto else end file for function goto if in label mod nil not of or 
otherwise packed procedure program record repeat set then to type until var while with
activate ambush Armata backpack berne beryllium bestubble bloom bosom bridegroom Brisbane bryozoa burp byway Byzantine cabal calla cape caught cleanse clinician congressman constrain cutback defrost desist dint distraught dragon Edwin elfin equatorial exclusion fourfold glove grasp groundwork harvestman hull humidiﬁy idiomatizable impracticable incise independent inhibit inhibitory interception invent Julie kilojoule Lin megalomaniac memory Messrs meticulous midget midshipmen Montclair mouthful Muong Nepal nephew Namey nosebag nuptial object onogeny optometrist organic pacemaker paddle pass Paulette paymaster perish ping Piraeus plenty politico postage pouch precaution quadrangular rally renounce Ross rubbbery scudding shah sibling snippet socioeconomic stamen stamen Suez sulphur Sumerian sunﬁsh sunken switch Tallahassee tensional they threesome tycoon vanity vermin Viennese Vogel Waterbury whiten wishbone Zachary zesty

r117.words

6th abuse accent accidental acetone adaptation adaptive additive admitted adulterous Ampex anastomosis astigmatic barnstorm being Beloit Bloch Bolivia Boris BP breadboard callus certify chard Christianson climatology cog company coney conquer cormorant Cyprus dapper Detroit discordant discrete dishwasher disruption endorse ensign enslave ethylene familial farther ﬁnance ﬂathead foyjer fruition generic Glasgow grindstone guardian gunﬂare hardboard hardscrabble height Hitler hooves housebroken hydrolysis inbum inactivet inlet insight instantiation leaden Lear Loeb logjam Malaysia Middlesex mimesis mischievous misshapen mnemonic Mohr moneymake motley ND neater neuritis not ordeal palmetto pander phantasy precipice presence proprioeception quintet ray Rayleigh reside rubbing ruckus sanction saucepan Savonarola scientiﬁc scription sheath skeptic sketchbook soggy Spanish steak taxpayer Thebes Thorstein thou torque wadi waken weighty whereby yeah zigzagging

r124.words

AC Addison aster backplate bedtime Belshazzar Berenices Boeing Britain bum Camille Carbondale chronicle Cloth cofactor Coffman concierge concretion connector corny crazy dame daybreak deal delicatessen delusive depreciable diary drippy economic Ellison emotional encyclical explicable farmland Faustus FCC FDA falt ferry gadget gale Guinea Hadrian Hague hailstone Haines Hamal house hurl hydropnusse imaginary impertinent inalienable Indonesian infect inﬂict insist intact intervention is Jansenist Kaskasia ken lap lengthy lever lookup madman malnourished Marlowe McLean Monticello Negroid Nevins newsmen nipple notch obsolete outrageous Pabst Paul pavane phenolic pity pocketbook pop pope predicament prefect presentation puberty pure python quiver rebelled roughshod seafare shipboard Sproul stimulate Stonehenge stopgap stylish summary suture sweater teeth through totalitarian turbine urethane Velasquez vitamin Vladimir volunteer vomit wahn Walton whole widget Woodbury worrisome yarmulke

r139.words

adulate alphabetic anybody Appleton archangel Arkansas Atkins bebop benthic Blackburn blacksmith blutwurst bubble bustard cant cede chalcedony chipmunk commemorate corrigible cozen critique cure cutaneous datum declamatory declimate densitometric desiccate diameter discernible disquisition dollily downspout drawback droop duke dusk earthmoving ecumenic embassy enlarge expeditious fake faulty Feldman ﬁnesseg ﬁrst Fitzgerald flung Gauguin get given glue grand grieve hamburger Hebraic Helmut hipping Howell lichen neum Ignoble Illinois imperturbable installation IRS job jog ketchup kiss Lakehurst landau Lillian Loire malaise malformed mascara megahertz measurable mode Mollie morgen nailed natural Nielsen Nero orate ordnance palate pampa Peter pharmacologist pistaccio pledge Pocono prompt prophesy punish quipping radium relinquish repairmen resonant reversion Rosalie Rwanda sack serum showcase showy sickle slogan Socrates Solon specie spheroideal squashed squirehood stereography stock stylish Sudanese suffragette sulky superb supping surpass tensor Tientsin tomato TOPLAS touch transit Turing upon worship yellowish
r162.words

abrasion agenda Aiken airmail Alexandra Allison allocable aloha amen American amply angry are Artie auction authoritative Baltimore barycentric benign bloodroot booklet bridgehead bull cacophonist cameldown catchup chandelier Christmas chub circumvent cocktail Coleman comedy composition Conklin cowardice cramming criss cycle deadline default deluge depositor detergent diacritical Dido docksides drieb Elysee etymology explicit extramarital Faraday fight finale formal francium Ginn glue glycerin Gould Gunderson hereafter hereof hypothesis immunization insuppressible inbreed incubi infelicity infinitum instead interception intricacy jazz Jew jolt judicature Kaplan Karen Lancashire Laocoön Lisa Lucille Lyra magnetite Magruder Mendelssohn Mephistopheles midge MIT molal monarch neophyte nightgown novel obtain offer offset passion Pax philodendron physiotherapist pickaxe pizza pope poseur pray precipitous predominate probe prudential psychosomatic puddle qualify quarrymen ragging repelled repetitive retain Rhodes ripe sabotage saucy sawbuck serious set shortage slugging snowball spatium sprain squishy strip sulfite Susanne swerve tabernacle tagging tao tend theoretician throttling thyroidal timbre tonsil trafficking transosceanic urea valid vandal venial verboose virtuoso Waltham waveform whence Wirth woebegone yah yogi yon

r181.words

abstinent acc climate acquaintance airy Alec allergic Allis along also Amazon anatomy any apathetic apparatus armoire arthritis asthma Augustan axon bar behavioral beneficiary bookcase cant capillary centrist cesium Cessna characteristic check check checkup checkpoint chink circumlocution Clarence clasp Cochran Cohen compliant composure corruptible cortex Coulter county courtesy creosote defy degas deify dichondra digging dispensate dispersive domain dome Eleazar elicit Ellison enfranchise entire entirety equine essence esteem eternity extirpate fecund fickle floor football fraternity fraught funk Galatia Geneva giggle grabbing graceful graduate Graff guru haunt heard hector honeycomb Illinois imitable impenetrable imponderable incarnate inequitable interferometric intravenous inversion jocular language larceny laurel Lexington Lois Lorraine malnourished McGowan memorabilia Merck meridional midget minuet Moen monument mosaic Nazareth Neff nodule oratory orb paperback pare Pegasus permalloy pharmacology phenolic phthalate powdery preface proviso Prussia rang rash rationale reduct rejoinder Rembrandt restive retrieve rifleman risky Romano round salmon Sandburg sausage schoolroom scud sharpshoot skied Slavic sobriquet somehow steadfast Stephenson stockade stratify strobe suction supine swabbing swift syrinx tab tackle that thing throwback toodle trajectory trip ultraconservative usurer Vandenberg vanguard verify vintage wavefront weight whack Whalen Wilbur Winchester Winnipeg Yugoslavia
192.words

academy adjourn Adkins aforesaid ahead alluvial amidst anagram arcane ashy assiduous audience Augusta aversion avoid axis babble bail Bayport bezel bias biochemic bitwise bluebook bouquet breakfast broaden c cameo camp Canadian captor Carbone chinch codpiece colony colossal conciliatory configuration confirmatory conservatory Corvus cotman Cuzbertson cuain Darlene decontrol define degree diatom diesel doorman Doris du dun dupoly emphases enigmatic entomology Ethan Eurasia evangel Faustian fluoride foist fusty glow godmother goldfish gorge grade grade haggard hair Hayward Hecuba hemming heroin hold hypocrisy impetuous imposture independent indistinct indoctrinate instantiation intelligible intermittent intricacy jockstrap Josephson judicious Justine Kentucky kola Kowalski lacing lace lee lexical logging Lowell Madeleine malice mammoth manslaughter Marlowe martyrdom melanin middleweight mocassin mode mortify nevertheless Newman Newtonian NIMH novelty nuclide oblivious octahedron Olson Oneida onset oodles optimist optometric oratorical pacify packet paraxial parsley patrol permitted perpendicular peticoat phenol Poland politician porterhouse post Prado prescott probe Pullman punish reach regale Regis relay rest roach saltwater San scaffold Schoenberg seclusion seditious serene serviette sheath shipboard stop soccer sollicit spaghetti speakeasy star stepchild sticktight stolen swishy symbol Talishassee tangent tangy tantrum Thai Thelma trammel treacle triumphal two TWX unanimity USA vicelike Victorian Voss waggle weatherstripping wonder Yale

236.words

Adelia Adkins advert again agglutinatin ague Armstrong arsenic attainer attendee snow beard becket bedraggle belt benefice Bernie Bertrand bethought bid binocular bivouac bran bravery breastplate caliper capacitor captious chalcedony chemisorb circulatory classificatory clerk conglobule coastal coerce collocation confederate contextual contort coppery cotman cosy cue curbside curia cyclist dactyl dark day decouple despotic dreadnought drench droop droplet ducat dully Edna encephalitis enqueue enslave Eocene epistle epitaxy escapee Eurydice excelling extensor farewell Farrell faun feasible ferric figure final float fluff foohardy forthsome forthright forum foxhole foxy fraught furrow gaff Gil Gino glacier glide gopher great grubby handyman Hansel Harmon havoc heel Helmholtz hemorrhoid hereabout Herbimer Hieronymus Hillcrest ho Holm Holocene hornmouth hornwort hospital Hubbell buckster Huggins iconoclast impute incalculable indefinable indeterminacy Indies inescapable inhale inspire interlude irresolvable jam Josiah kernel Khartoum kilobuck kneel knowhow kudzu length lengthary letterman listen locomotive lulu Luxembourg Mackey maid malformation manifestation Marsha marshall McKenzie merganser midas modicum mugging nab neath nimble Northampton note nucleoli Numerische osmosis ovenbird pact path peed penultimate philosophic Phoenicia plagioclase Polaris prefatory prescription prolapse pursue quadrennial quandary quicken radiophysics rancorous rapacious religion revving Rio rouse Rowena satanic sedan Sepoy serene severity Sherlock showman siderite signet Silverman snarl snippy snob Sparkman sporadic squirm syringe stanch stontium suey sunbonnet sunburn Susan swag swank swan symposia synchrony tamarack tassel teaspoon tenon tentative text thermonuclear thesis threat trident troposphere tutu Ulysses uncle upset veracity Vivaldi Vulcan wanton warehouseman whirlpool wisecrack Wolff yen yeshiva

373.words
abhorrent accelerometer adobe advisory Afghanistan aficionado agouti allegation alphabetic amble Americana amoebae Andean animism annual appoint Aristotele armillaria arose assimilate assumption attrition aural australite automata backplane Basel beavering beechwood Bengal Bertrand Bees Bhutan biceps biennial Bingham birdbath birthplace bivariate Bizet blast Blinn bluefish bodybuilder borrow boyish Bradshaw brassiere broadloom business butchery caller cardiovascular Carey cassette chauff chalice challenge cherubim chortle chromosphere classy clink clubroom conglobate coax cockleshell coffeeshop collegiate communion compassionate compendium compress confluent connoté consequential consist convalescence corny corralled corrugate crabapple cram crankshaft credulity cursory cut dashboard davit daze Decca decertify deputy desert detente Devon devour Dickinson directrix disembovel dispel divine Dnieper dominion dole dramaturgy drug duet echo Emily equinoctial estoppel exasperate exhibit expound exterior facilitate fence fete fever flange flannel flashy floodlight foe foolproof fore foulmouth furl furtive future gagging galley Galgargantuan gaunt Genevieve Gillespie glacia glamour gnaw godlike governance grabbing granola Grecian grill Gwyn gypping h Hagen handmade heap hector heigh herb Heuser Howard Hoyt Hubbard humane Hurd ignorant improvise inactivate incontrovertible incumbent indium inefficient Ingram intimacy irredeemable isochronous jarring judicable Judy Kevin kidnapping Kinney landau Langmuir laurel lexical libertine limpid load log lopseed lucre Lura lurk lush lysergic manageable Manley marshall masseur Mathewson Matthews member message michigan microscopy millennium Missoula molybdenite monarchic monel monkeyflower Monticello morgen Morocco morrow mythic nadir naiad narrate Natchez nautical nepenthe newsmen NH Nobel Northrop nostril obdurately obstrepe offshore opiate opt optoacoustic opulent outlandish ow paid palazzi parquet partook peep peltry pendant penetrable pervasion petition phenomenal pillory pimple pinpoint pizzicato planet polemic portage posseman postdoctoral precinct predilect prefab prefix preview progression prohibit proposition proteolysis psychophysical purple quadratic quadruple quaff quill rankle recent recappable red referent Rembrandt Remington representative retail rhythmic ribonucleic Rigel ring rob rockaway rocky round saddle sandhill Sanford Sanhedrin sappy satiate Savonarola Savoyard schematic schoolwork screed scribble seizure service Sharpe shed shield shoeface shed SIAM Sicilian sincere sinful sinistral sial Sistine Smithsonian solicitude somatic sorghum soundproof Spanish sparge spectrometer spectrometric spent spiderwort Spiegel staid Stanton steady Sterno stockroom street stumpsage stumpily stunning subpoena suggestible sunset surface suspend suspenso swimming talon Teledyne template tenspot terse Theodore therapist Theseus tor tory treatise treble trial trite trunk TTY tub tundra tweed twirl unanimous USGS vagabond vicelike Victorian Wehr wheelchair wildlife Williams Winslow Zomba

---

r495.words
r667.words

abalone abdominal aberrate abysmal Achilles acid adherent adjourn administer admix
adventitious adversary aerogene agilean ago ahem alb allergy amalgam amorphous ampersand
analogy Annale annum anorhistic apostate apparel appellant approbation ardent argillaceous
aristocratic arraign Ashmolean assert asseverate Atwood autonomic autumnal Azerbaijan
azimuth azure badge Baku balene Bambi barber Barclay barony bed begging beginning belmen
belt benzene beret betel beware bezei Bhutan biometry bishop bitch bittersweet blade blockage
blockhouse boa boatmen boon bootstrap Bosch botheromey boycott Brest Brian briny Bristol
brothel bryophyta bulb bureaucracy buried bursty butch butene cachalot Cady calumniate
Calvary Cambridge canal canvass capella carcinogen career carouse Carruthers centerpiece
centroid cerium cerulean chafe chance chancy chaste chaste cheap Chen choose choppy choreography
chrysanthemum churchgoer churchyard circlet citrate Clarence Claudia clearheaded cliche
Clifford closet cloth cloture cobble coccidiosis Colgate colicky collapse combatted comedy compile
compent comptee comprehensible compression concur condemn Conestoga conferring conforml
congest conscription consent conservatism consist convey Conway cookbook copolymer cornua
Cornelius Coronado corontine Corvus cosmopolitan covenant cover crackle crafty Cramer cranial
crematory crew cricket cried croak crupper cubic cultivable CUNY curlique Cushing cushion
cutworm Cynthia Dadaist dasi dateline Davies decelerate deconvolution deficit demise demon
Denmark depressed desecrate desiderata desideratum detector derrtert deus dismal d'oeuvre
dogberry Dolan dolphin Domesday dossier dynast earwig eat ebony editor eerie el
electrocardiogram Eliash ellipsoid energy enquiry enrich enthrone esquire Ephesian
establish estate Etruscan eventful excommunicate exculpatory exemplar exhaustion exhilarate
expenditure extension extensor extricable eyeball FAA face fag fairy fanciful fare fast feather
fidelity ferroelectric fetter fettle fibrin fickle filler fireplace fireproof fiscal fixative flak flake
flam flapping flaxen Fleming fondle fox Frenchmen frilly frozen fugitive fuse gel genealogy
gentlemen geopolitic gestalt Gilmore glad glassware gleam glycerol godparent Godwin grandfather
graphic grease greatcoat gripe Guam gunning Gutenberg hangar Hanoi hardtop hardworking hatch
heady Hercules hermetic hesitiate Hewitt hexameter Hibbard horseman Huber Hugo hundred
hydrochloric hydrochloride hypothesis ibex identical ignomamus imbroglio impetuous implicit
inasmuch incalculable incensuous incidental indigestible indiscreet indwell inequity inevitable
inexorable inherit inquest inquiry insufficient intemperance interior interrogatory invariable
involutorial ionosphere irradiate Irving Joan jolly Jr ja Judith junta kaolin Katherine Kauffman
keep Kelsey Kelvin Kendall key Kimberly knives knowledgeable Knowiton kuquat land Lange
Larsen layoff left Len Leon Leroy Leanto libertarian libretto lignum limbo liquor lobo locust
linenloth Loire lord Louvre lucrative luke luminosity madrigal mailman Malagasy manometric
manslaughter manumission Marie Marrietta martini mastermind materiel maximum maxwell
menace Merle Mervin mesa metallurgy mien Millikan milord minuet modulate momentum Monroe
mosaic mossy motel mountain multiplicity Mumford myofibril naive Nanette Nanking Nash Navajo
nay nearest neon nervous nettlestone neuralgia neuropsychiatric Niagara nips nickel nigh nirvana
NM northwestern novitate numerous occult o'clock officeholder OK optoacoustic ordeal O'Shea
oxen p paintbrush palliate pang pantheist parenthesis Parisian passport path patrol patty pecan
pedestrian peepy Pegasus pelvis pepperoni Perle permission persist peripatetic pertain
philosophy phraseology pilewort ping pion placental pleural polygonous porridge portend Powell
praseodymium preliminary prince prizewinning productivity profound prolongate propagate
prospector protege prothonyatory prune psychosis puckish Pulitzer purgatory puritanitc pursuit
quaint queen quick quit rabbet racy radiography radius radix raj Ramsey ransom ravel realtor rectangular reduce repentant reside restraint retrogress revival revival Ripley Rosenberg roughcast roundhouse saga sagacious Saigon salesperson salvo samovar Sargent scheme Schuykill sclerosis scopic scoria script scrutiny scull scurry sedan Sepoy Seton seventieth Severn shipyard shish ahod showroom sideways sienna signpost silhouette silt sink sinister sire Sistine skill skyrocket slavish sleepy sludge smaller Smithsonian snapdragon Sofia solvent somathe southernmost spanel spate Spaulding specie splanchy spurious stain staminate staple stasis stature stave stem Stephensohn sternum Stevens stringent striven strum stuff stupor Sturbridge subject subliminal succubus Suffolk suffa sundered suppose surgery swath swathes sweater sweat swell swelter swerve tachometer tact taffy tanager Taos tardy tassels teaspoonful tecum teen Telefunken teleprompter ten Terre textual Thayer the thyrotoxic tipping titular tolerant Toni tonsillitis transgress transpacific treasury trellis tribunal trichloroethane trinket triumph trivium tunnel twisty typesetting ubiquitous Ullman van variac verify vermilion vesicular vexatious victorious viewpoint villa volatilize waken waken washbasin washbowl Watson wax Webster whale whereon whet wig wildfire willful wingtip Winnipeg winnow wrongdoing wrongful yowl yearbook Yeats yet zeroth zeta Zoroaster

r992.words

aborning abrasion abscissae accipiter accord accrue acme acquiscent addend Addison additive adject ado advantageous advisee aerogene aeronautic affluence agnomen alle Albania Alex Alexis aliphatic allyl aloha although alway ambiguity Americana ammo amort Angeline anionic anorexia ante Antioch anyway apostolic appall arbitrage Arden Argentina aspheric assault assiduity assign astonish atonal attitude audio audiotape austere auto automate automatic avenge award awful awkward ax eye Aylesbury babyhood Bach backside bacterium ballot Bambi banana berry barbudo bark Barnard barricade baseplate basophilic bathrobe bayonet bazaar beacon beauteous Becker bedtime beget beggary begotten belate below bemuse beneath benefactor benthic Benz Bergland Berlin bestseller bicarbonate bicker Birmingham bite bled blest blonde Blum blury Boca bong bopping born botanic bothersome boulevard boutique Boylston breadfruit breakfast breastplate breech bricklayer Bridgewater bristle broken Bromley bronzey broughaha Bryan bulk bunkmate Burnside burnt bursty Byzantine cabbage caddis Calhoun calico Callahan calm Cambodia Cameroun candy capella captious Caputo caravan Carbone cardiology careful Carlo carload carney Casanova catatonic catfish Cayuga cell centennial Cetus Chad chairperson chamber chance chap checkout chemistry cherub chicanery Chinatown chuckle chuff Churchill cinematic Citroen cityscape clamp claustrophobia clean clergymen clink clockwork clothesbrush Clyde coed Cohn cohort colossalus commend commentary commissariat committing commodore compel compress compression compressive concerti configuration Conrail consent consistent contradistinct contrast cookbook Cornelia correled corrupt corruptible counteract coverlet crap crash crate creek creating creek crucify culinary culprit culture cumbersome cunning curio cyycd Cygnus dachshund darn deafen Deanna debarring debugging debutante decedent deerskin defensive defocus dfet demagogue demean demon demonstrable denial dentistry descent descriptor designate Desmond deviate devil Diego diffraction diopter diorama disastrous disciplinary discipline discretionary disembowel disparage ditch Dixon dodge doe doleful Doris Downing dream driven drood drosophila drudgery dulcet duplex dustbin ear eardrum Econometrica effete egregious Eileen ejector Ektachrome electoral Ellis elin embattle embouchure emolument employer encryption enter entertain envenom envision epithet Epsom erase Erasmus erotica errancy escapade espionage Evans Evanston evergreen evoke excommunicate executive exhort exogamous exposist extolling
extort Fabian Fairfax faithful falcon Farber farce Farley Farmington fasciculate faucet feminine finish firebug firework fizzle flammable Flanagan flinty fluncty flood Floyd focus folio Fomalhaut foolhardy forborne Foss frankfurter fraternity freedom fresco frivolous frout frugal front frontal frontiersman frothy frozen fuel funnel gable Gail Galtreath gallinule gallow gander Gannett gaur gear gent Gettysburg gibby Gifford Gilligan gird given gladdy Glidden glimmer glitch glossed glottal glove goggle Goliath Goodman gorilla Gorton gradient grain grammatical graven grilled gristmill Guam guard guess guilty Gullah gunpowder Gurka Haag habitual halfhearted handcapping handset happenstance harbinger hardtop Harlan Havana hazard heady hearken hearth heathenish Heine helpmate Henri hexafluoride hey hippy hire historic hitch hitting holt Honduras honeymoon horoscope Howitz horseback hostage how humble Humboldt humerus hundrefold hurray hyaline hypothetic IBM Icelandic icy idol idyllic ignorant impume inapt inaugural inborn incalculable incceptor income incompressible inconceivable incorrect increase incredulity incredulous incriminate incubus inductee industrial inept inerence infantryman infarct infect infelicitous infer ingather inglorious ingot inhabitant innovate inset insidious insolvable inspiration intellect interject interrupt interruptible intestate introductory invariable involute irksome irrefutable isochronal isoclinal isotherm item janissary jelly jockstrap Johanson journalistic Julio justify kaleidoscope Kamchatka Kathleen Katmandu Kelsey keynote kilobuck Kipling kite knowhow knowledgeable kulak Kur Kuro lace lance Laocoön larcey Larsen lean leap legislature leitmotif leitmotiv lest Leviticus libel libido licensible lifelike Lindsay lithology lithospheric liveth locust lodgepole Loomis loot Lottes lousy lowdown luminance lurid lustful lutetium macro madcap madstone magazine Malabar malaprop malfunction malpractice mandarin mane mantelepiece Margery masculine masonry masquerade masterpiece Mathews mature McCann McCullough McDougall McGuire Merriam merrymake mesa meteor mettle Middleton midweek might mike militate milk minaret miniature minor Mira misery moccasin mockery modicum Moiseyev moisture moment monastic monies Montrachet morn Morristown motivate mountainside mousy mugho multiplication mustang mustn't mutant muskle mystify NAACP naiad natty nausea nave NCR nearby necropolis neurasthenic Newbold Newell newscast newareel nightdress ninefold nitpick nodding Noel noisemaker Noll nominee nonogenarian nook Northumberland noticeable nova novitiate nu nuisance nutrient obese obliterate ocident octogenarian octopus octocon Oedipus old onerous ordinance Orestes oriole orthicon orthography oscilloscope ozone pack pal pamphlet panning parochial Parsifal participle particular passe patchy patchcheck PBS peaky peak pedantry pee pellet pennroyal pep perpendicular persecutory perspicuous Perth pet Peterson phenomenon philodendron phonetic pianist Pickford Pierson pigtail pinscher piracy pitchstone placeable plagued planetaria plaque Pleiades plethora pliant plod poach pokercase polar Polaris pollution ponderous pony portulaca possession practical preface preclude presumption previous proclamation proffer promote pronunciation prophesy propyl protagonist Proust proverbial psychoanalytic psyllium pupil puppyish purr pursue pushbutton pyxene Pythagorean quadrant qualified query question queue quirky quixotic radices raffish rat ravel receive reciprocate reclassify redemptive refinery refractory refutation regime remnant renewal repellant replica reprisal requisite rescind resilient respiration retrieval reverberate revere rheumatism rid Ridgway rig rigid riotous rivulet Roberts rollback Rome round root roughen round roundtable rudimentary Runge Russia rye sail sandpiple sandpiper satin satiric satisfactory Saturnalia saute scabbard scandal scare scarlet scary scenario scenery scheme schoolbook schooner scope scoreboard scrim scum scurrilous seam seashore seater seeable sera Seton Severn Seville shedding shimmering shiplap shipwreck shopping shortstop show showy shrapnel SIAM sidewise silk siltstone Sims sinewy sinus skew skittle