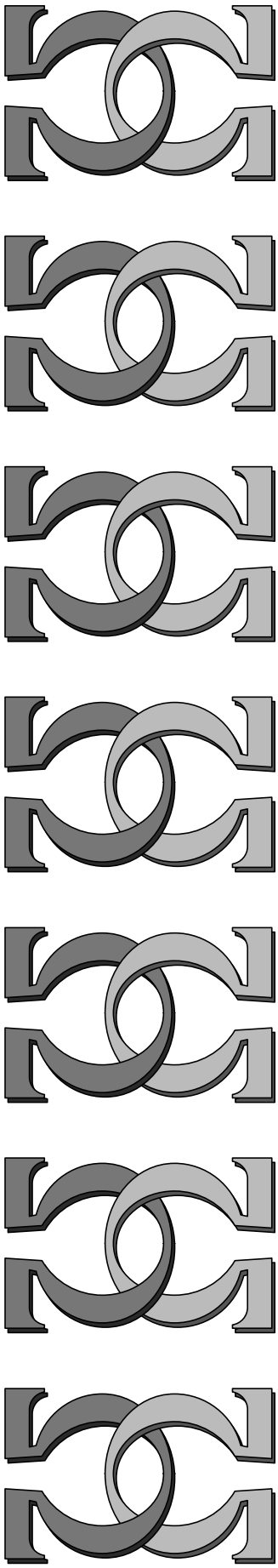

CDMTCS Research Report Series

Matching T-Codes to a Source

Ulrich Günther
Department of Computer Science
University of Auckland

CDMTCS-153
April 2001

Centre for Discrete Mathematics and
Theoretical Computer Science

Matching T-Codes to a Source

Ulrich Günther
Department of Computer Science, The University of Auckland
ulrich@cs.auckland.ac.nz

May 30, 2001

Abstract

A classic area of application for variable-length codes is data compression. This paper looks at a class of variable-length codes, the T-codes, that have been noted for their self-synchronisation properties. While the coding efficiency of T-codes is less than or equal to that of Huffman codes, no simple algorithm for the construction of T-codes from a given set of source symbol probabilities has been developed so far. To date, the only approach to finding the most efficient T-code set for a given source seems to be an exhaustive search, which to date has been too complex to be of much practical value. This paper describes some shortcuts for this search and presents recent improvements that yield a much lower computational complexity.

1 Introduction

This paper assumes the following situation: an information source emits a finite number N_s of distinct source symbols σ_i, where $i = 1 \ldots N_s$, in a continuous stream. $P(\sigma_i)$ denotes the probability that the next source symbol emitted by the source is σ_i. Each σ_i is assigned a unique variable-length codeword $x(\sigma_i)$ from a prefix-free code set $C \in S^+$, where S is an alphabet and S^+ denotes the set of all finite, non-empty strings formed over that alphabet.

The expected redundancy r of this encoding is given by the difference between the weighted mean of the codeword lengths of the encoded source and the source's entropy (cf. Shannon [1]):

$$r = \sum_i P(\sigma_i) \left[|x(\sigma_i)| + \log_{\#S} P(\sigma_i) \right] \qquad (1)$$

We further assume that $P(\sigma_i)$ is independent of previously emitted symbols — an assumption that does not hold in many practical cases. Still, in many cases this presents a fair approximation and significant compression gains may be achieved by minimising r.

Huffman [2] introduced his now famous algorithm for the construction of a prefix-free variable-length code with minimal redundancy, assuming such a simple source coding model. His algorithm has found many practical applications. It is used in the `pack` command under UNIX and for the compression of DCT coefficients in the popular JPEG image compression algorithm.

The Huffman algorithm starts with given source probabilities $P(\sigma_i)$ and yields a variable-length code set over S with minimal r. This code set is generally not the only possible one that minimises r. One of the reasons for this is that r does not depend on the code itself, but merely on the code length distribution, i.e., the histogram function of codeword lengths. This is a result of the $|x(\sigma_i)|$ term in Equation (1). Codes that share the same code length distribution are hence equivalent in the sense of Equation (1).

Capocelli, Giancarlo, and Taneja [5] presented bounds on the redundancy of Huffman codes. Similar work was also undertaken by Gallager [3] and Johnsen [4].

Another class of variable-length codes, the T-codes, are of particular interest under the aspect of code self-synchronization.

2 T-Codes

Generalized T-codes [7, 8, 13] are a recursively constructed class of variable-length codes.

Depending on the source symbol probabilities, a Huffman code may also be a T-Code. In fact, any T-Code set could theoretically have been constructed as a result of a Huffman code construction algorithm.

However, rather than letting the source symbol probabilities determine the length of the codewords and the structure of the decoding tree, T-Codes are constructed with no regard to symbol probabilities. Their construction focuses instead on a recursive tree structure.

A T-Code set is constructed as follows:

1. Start with the alphabet S. Every finite alphabet is a (trivial) T-Code set by default, with the letters being primitive codewords.
2. Given a T-Code set, another T-Code set may be derived from it by a process called "T-augmentation". This involves picking a codeword from the existing T-Code set, which is called the "T-prefix", and a positive integer called the "T-expansion parameter". Any T-Code set may thus be derived from an alphabet in a series of n T-augmentations using a series of T-prefixes $p_1, p_2, \ldots, p_n$ and a series of T-expansion parameters $k_1, k_2, \ldots, k_n$ (now also called "copy factors"[14]). The resulting set is denoted $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ and is said to be a T-Code set at "T-augmentation level" n.

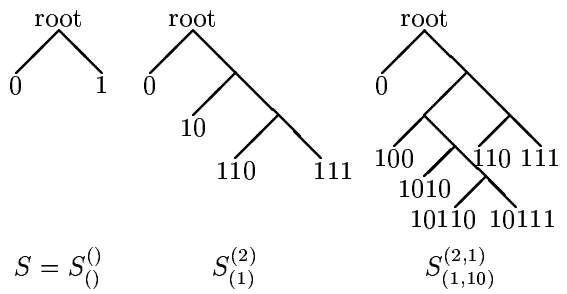

Figure 1: T-augmentation as a copy-and-append process of decoding trees

The T-augmentation itself is performed according to the following equation:

$$S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})} = \bigcup_{i=0}^{k_{n+1}} \{p^i_{n+1} s | s \in S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)} \backslash \{p_{n+1}\}\} \cup \{p^{k_{n+1}}_{n+1}\} \quad (2)$$

where $S^{()}_{()} = S$.

That is:

1. select a T-prefix p_{n+1} from the existing set $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$, which has been derived from an alphabet S by $n \geq 0$ T-augmentations.
2. select a T-expansion parameter $k_{n+1} \geq 1$.
3. make k_{n+1} additional copies of the decoding tree.
4. Successively concatenate the new copies and the original of the tree via the T-prefix chosen, i.e., the root of the first copy attaches to p_{n+1}, that of the second copy to $p_{n+1}p_{n+1}$, etc.

Figure 2 depicts the construction of the T-code set $S^{(2,1)}_{(1,10)}$. In the first T-augmentation in this example, two new copies of the tree for S are made, and the two new copies are linked to each other and to the original tree via the respective leaf nodes corresponding to the

codeword 1 in the original and the first of the two new copies. The second T-augmentation links a new copy of the tree for $S_{(1)}^{(2)}$ to the original tree via the leaf node 10.

Note that the above construction pays no attention to coding efficiency, although the choice of T-prefixes and T-expansion parameters permits the construction of a wide range of sets with varying code length distributions.

3 T-Codes vs. Huffman Codes

Whereas the Huffman algorithm guarantees minimal r and is constrained only by $P(\sigma_i)$, the T-augmentation algorithm makes no obvious provision for $P(\sigma_i)$ to influence the construction process and gives no guarantee of a minimal r [13]. However, it does guarantee good synchronisation performance of the resulting code set. Thus, there is often a trade-off between self-synchronisation and efficiency when a T-code set is used for source coding.

If one wishes to use T-codes in a compression application in order to exploit its synchronisation properties, one cannot use the Huffman algorithm in most cases. Unfortunately, there is no known simple algorithm for T-codes that permits the selection of the T-code set with the lowest possible redundancy.

This poses the question as to how we may find a T-code set with minimal r for a given source.

For many practical applications it may be sufficient to simply soften the criterion somewhat and look for a T-code set with a "small" r, rather than the smallest r achievable.

Mark Titchener [15] suggested such an approach. It exploits the fact that the largest efficiency gains are made with the shortest codewords. He proposes to encode the source as a Huffman code to obtain an optimal codeword length distribution. Starting with the shortest codewords in that distribution, one must then try to construct a T-code set that matches it closely.

This may be well justified in a number of practical situations, e.g., where only approximate source statistics are known and the additional inefficiency would be small compared to the errors in the symbol probabilities.

In another approach, Gavin Higgie presented a database of "best T-codes" in 1996 [9]. While it does address the issue of coding efficiency, it restricts itself to *simple* T-codes, i.e., T-code sets for which all T-expansion parameters are set to 1. However, the generalized T-codes used here give rise to a much greater range of code length distributions. Hence, they can often provide a more efficient encoding for a given source.

To date, the only strategy for finding a T-code set minimal r that has been identified is to perform an exhaustive search of all feasible code length distributions. This approach was first presented by the author in [8] and [13]. This paper presents a much improved version of the algorithm in [13].

4 The Search Algorithm

The search algorithm presented here operates mainly as a recursive "divide-and-conquer" algorithm. This section discusses the general structure of the algorithm, whereas the next section takes a detailed look at the techniques used.

The basic idea behind the algorithm is the following: each potential T-code set may be regarded as a node in a tree. The tree's root is the alphabet that the sets in the tree are based on. Each branch in the tree represents a particular T-augmentation of the T-code set represented by the node the branch originates from. The T-augmentation that the branch represents uses a distinct pair of T-prefix and T-expansion parameter. Finding the most efficient (i.e., lowest redundancy) set in the tree requires us, in principle, to traverse the tree exhaustively, i.e., we must visit each node, calculate the redundancy of the as-

sociated T-code set with respect to the given source, and keep a record of the set with the best redundancy that we have found in the process.

This is only feasible if we have criteria that limit the number of nodes in the tree, which would otherwise be infinite. We will discuss these criteria in the next section.

One simplification, however, is so important that it deserves an immediate mention: Since the code length distribution determines the redundancy, we can simply use codeword length distributions as nodes rather than full T-code sets. As a result, we must also prescribe codeword lengths to use as T-prefixes for each T-augmentation, rather than actual codewords. As there is often more than one codeword of a given length in a T-code set, several sets resulting from T-augmentations with these codewords map to the same distribution. This reduces the number of nodes in the tree significantly, especially since the savings accumulate over several T-augmentations.

The basic algorithm is depicted as a flow chart in Figure 2.

It is a "breadth-first" algorithm, i.e., it traverses all distributions at a given T-augmentation level first before it continues investigating the distributions at the next higher level. In principle, it works as follows:

1. Initialisation: sets up global parameters such as the variables for recording the best code length distribution δ_b and best redundancy $r(\delta_b) = \infty$ found so far. It also sets the "seed" code length distribution δ_S of the alphabet (i.e., $\#S$ codewords of length 1, zero codewords for all other lengths) and uses it as the base distribution for the recursive generation and investigation of the T-augmented distributions. The distribution δ_S and all other code length distributions used in the algorithm also include T-prefix length and T-expansion factor vectors P and K, which are empty in the case of δ_S. δ_S is then added as the first element to a previously empty queue.

2. Recursive matching. While the queue is non-

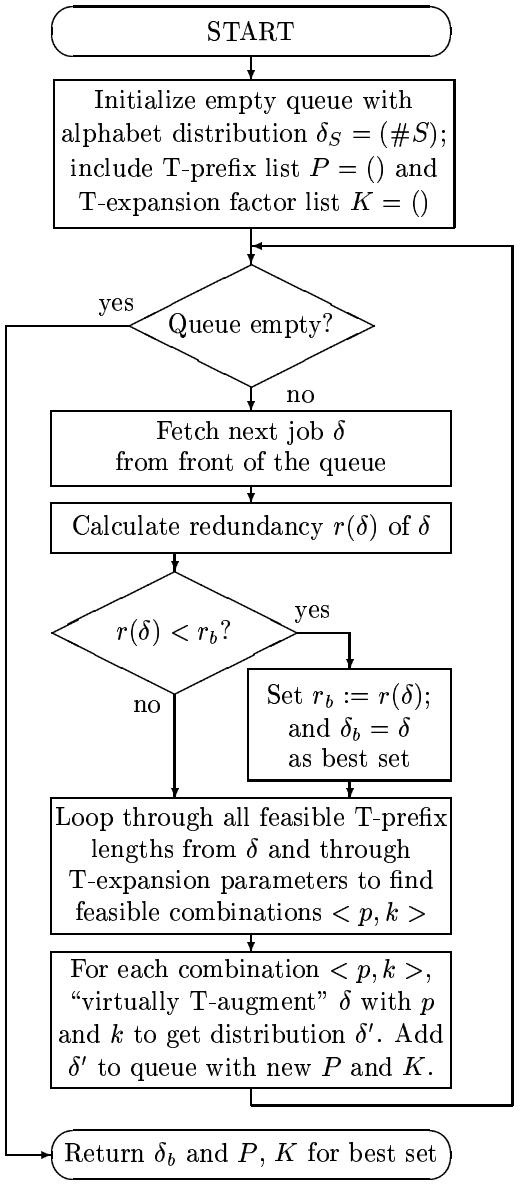

Figure 2: Outline of the breadth-first matching algorithm

empty, the algorithm fetches the first element, which we call δ, from the front of the queue. δ includes the codeword length distribution as well as a T-prefix length vector P and a T-expansion parameter vector K that describe how the distribution was arrived at. The procedure then calculates the redundancy $r(\delta)$ associated with the element's base distribution δ. If $\#\delta < N_s$, we may define $r(\delta) := \infty$. If the redundancy of δ, $r(\delta)$, is smaller than the best redundancy $r(\delta_b)$ found so far, then δ is the best match for the source to date and we need to update the global records on $r(\delta_b)$ and the best redundancy set.

The algorithm must now investigate whether a T-augmented version of δ could yield an encoding with lower redundancy. Starting with the shortest available T-prefix length in δ, the procedure loops through all combinations of available T-prefix lengths p and T-expansion parameters k that meet the feasibility criteria that we have yet to discuss (see below). For each feasible combination $< p, k >$, it creates a new code length distribution δ' corresponding to a T-augmentation from the base distribution δ, with new T-prefix length and T-expansion parameter vectors (P, p) and (K, k). In order to adhere to the breadth-first paradigm, δ' is added as a new element to the back of the queue. Any distributions thus obtained are used by the procedure as the base distributions for recursive calls to itself. The algorithm then proceeds to fetch the next element from the front of the queue until the queue is empty.

3. At the end of the algorithm's run, all feasible distributions have been generated, and the best achievable redundancy is known. A set of T-prefix lengths and T-expansion parameters for a code set with this redundancy is also returned.

Why breadth-first? The above algorithm was implemented initially both as a depth-first algorithm [13], and as a breadth-first algorithm. In the depth-first algorithm, any new candidate distribution resulting from δ is investigated immediately rather than added to the end of a queue. Only once its branch of the tree has been exhaustively investigated, execution returns to the base distribution and the next T-prefix length/T-expansion parameter combination for it is investigated. In its raw version, this approach seemed to outperform the breadth-first algorithm described above. However, recent improvements to the latter have led to a breadth-first implementation that is significantly faster than the depth-first algorithm in most cases. We will return to that topic later in this paper.

5 Feasibility Criteria and Simplifications

This section discusses the techniques that are used to ensure that: (a) the search algorithm terminates, (b) the search of multiple equivalent code length distributions is avoided, (c) the code length distributions are restricted to codeword lengths of interest, and (d) that code length distributions that cannot yield a minimum redundancy are avoided, if possible.

5.1 Virtual T-Augmentation

As mentioned before, rather than searching for a "best set", we may restrict ourselves to searching for the best distribution and a "recipe" that permits the construction of a set which features that distribution. For this purpose, the algorithm uses a technique called "virtual T-augmentation".

Let δ_C denote the code length distribution function of a code C, such that $\delta_C(l)$ denotes the number of codewords of length l in the code set C, and define $\delta_C(l) = 0$ for $l \leq 0$. Then

$$\delta_S(l) = \begin{cases} \#S & \text{if } l = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

and

$$\delta_{S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}}(l) = \begin{cases} \delta_{S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}}(l) - 1 & \text{if } l = |p_{n+1}| \\ \sum_{k'_{n+1}=0}^{k_{n+1}} \delta_{S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}}(l - k'_{n+1}|p_{n+1}|) & \\ \qquad \text{otherwise} & \end{cases} \quad (4)$$

The last equation defines the *virtual T-augmentation*. Note that only the length of the T-prefixes plays a role here — if $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ contains more than one codeword of the intended T-prefix length $|p_{n+1}|$, then $\delta_{S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}}(l)$ does *not* depend on *which* of these is chosen as the T-prefix in a corresponding "real" T-augmentation. On the other hand, the choice of T-expansion parameters has a significant influence on $\delta_{S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}}(l)$.

Virtual T-augmentation permits us to manipulate T-code set code length distributions rather than full sets, which is exactly what is required if we wish to minimise the redundancy r.

5.2 Avoiding Multiple Equivalent T-Prescriptions

As mentioned before, two code sets will have the same redundancy r if they have the same code length distribution. Hence, it is sufficient to generate the code length distribution for just one such set to obtain r for all sets having the same distribution. In particular, two T-code sets share the same distribution if they have been generated by equivalent sets of T-prefixes and T-expansion parameters. These sets of T-prefixes and T-expansion parameters are called T-prescriptions [6, 12, 13]. If we use different T-prescriptions for the same set, we are also generating the same distribution for the same set.

An obvious way of preventing this duplication of distributions is to specify that all set T-prescriptions must be in their anti-canonical form [6, 12, 13], i.e., to demand that the T-expansion parameters used must be one less than a prime number. Since the number of primes in any larger finite interval of $\mathbb{N}$ is significantly less than the number of natural numbers in that interval, this constraint saves a significant amount of computation.

5.3 Non-Decreasing T-Prefix Lengths

A further way of eliminating duplicate code length distributions is to use T-prefixes in ascending order of length:

Consider the form of codewords in $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ as presented in Theorem 4.2.3 in [13],

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} k'_0, \quad (5)$$

and the form of the associated pseudo-T codewords from Theorem 6.1.2 in [13]:

$$x_\phi = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \ldots p_1^{k'_1} \lambda, \quad \text{with } 0 \le k'_i \le k_i \text{ for } i = 1, \ldots, n. \quad (6)$$

Given a fixed set of T-prefix lengths and T-expansion parameters for these two equations, neither of the code length distributions that these two equations give rise to is changed by "swapping" the order of any two adjacent substrings of the form $p_{m-1}^{k'_{m-1}}$ and $p_m^{k'_m}$ for $2 \le m \le n$. I.e., for "adjacent" T-augmentations,

$$\delta_{S^{(k_1,k_2,\ldots,k_{n-1},k_n)}_{(p_1,p_2,\ldots,p_{n-1},p_n)}} = \delta_{S^{(k_1,k_2,\ldots,k_n,k_{n-1})}_{(p_1,p_2,\ldots,p_n,p_{n-1})}}, \quad (7)$$

provided that both $S^{(k_1,k_2,\ldots,k_{n-1},k_n)}_{(p_1,p_2,\ldots,p_{n-1},p_n)}$ and $S^{(k_1,k_2,\ldots,k_n,k_{n-1})}_{(p_1,p_2,\ldots,p_n,p_{n-1})}$ exist.

Here, we have two cases to consider:

- $|p_n| \le |p_{n-1}|$. In this case, $p_n \in S^{(k_1,k_2,\ldots,k_{n-2})}_{(p_1,p_2,\ldots,p_{n-2})}$ and both sets exist. If we demand that T-prefixes should be used in ascending order of length for all virtual T-augmentations, we generate the distribution for the second set and hence, by equivalence, cover the first set, too.

- $|p_n| > |p_{n-1}|$. In this case, the latter set *may* not exist: if $p_n \notin S^{(k_1,k_2,\ldots,k_{n-2})}_{(p_1,p_2,\ldots,p_{n-2})}$, then p_n must be generated as a result of the $(n-1)$'th T-augmentation and hence $|p_n| > |p_{n-1}|$. If we demand non-decreasing T-prefix lengths for all virtual T-augmentations, we will generate the distribution for the first set. Whether the second set exists or not is thus unimportant as its possible code length distribution is covered by default. □

¿From the above we may conclude that it is safe to require that the T-prefix length should be non-decreasing for all successive virtual T-augmentations.

5.4 Assignment of Codewords

When calculating the redundancy of a T-code set, one must of course assign source symbols to codewords. This is obviously done in rank order of probability, i.e., the shortest codewords in the T-code set get assigned to the highest probabilities such that

$$P(\sigma_i) > P(\sigma_{i'}) \Rightarrow |x(\sigma_i)| \leq |x(\sigma_{i'})|. \qquad (8)$$

This implies that some of the longer codewords in the T-code set may be unassigned. In turn, this yields a feasibility criterion for the choice of T-prefix lengths and T-expansion parameters.

5.5 Feasibility of a Virtual T-Augmentation

Presume that the present distribution $\delta_{S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}}$ contains a sufficient number of codewords to enable us to encode the source. Let L and ℓ be the length of the longest and shortest assigned codewords in the present distribution. Further let $|p_{n+1}|$ be the T-prefix length, and k_{n+1} the T-expansion parameter under consideration. Then we can require that

$$k_{n+1}|p_{n+1}| + \ell < L. \qquad (9)$$

The left hand side of this inequality is the length of the shortest codeword that would be newly created in this (virtual) T-augmentation. If this length is not shorter than the longest codeword assigned so far, then no source symbol can be assigned a shorter codeword in the new distribution and hence r will not decrease. Nor would any of the newly created codewords in this distribution serve as sensible T-prefixes. Consequently, it does not make sense to further investigate this distribution.

As any codeword that is used as a T-prefix for a virtual T-augmentation is no longer available in the T-augmented set, the criterion may be tightened further: we may require that at any new T-augmentation produce at least two new codewords shorter than L — one to compensate for the loss of the T-prefix codeword, and one to achieve more efficient encoding as above. Hence, we define ℓ_2 as the length for which there are at least two codewords with length smaller than or equal to ℓ_2. The previous equation thus becomes:

$$k_{n+1}|p_{n+1}| + \ell_2 < L. \qquad (10)$$

Note that this requirement puts a bound on the number of sets that need to be searched and hence guarantees that the search algorithm will terminate.

5.6 Redundancy Criterion

The feasibility criteria presented so far are independent of the probabilities of occurrence of the source symbols that we wish to encode. Given a certain number of source symbols and using only the feasibility criteria above, the search algorithm would thus always search the same number of distributions. The criterion introduced here is used to determine the feasibility of distributions on the basis of the source symbol probabilities:

Presume that we have established an upper bound for the redundancy of the most efficient T-code set, e.g., from a set distribution whose redundancy we have previously calculated. As we calculate the re-

dundancy for a newly generated distribution, we may without loss of generality add the $P(\sigma_i)|x(\sigma_i)|$ in order of decreasing $P(\sigma_i)$ and watch the sum's value after each addition. Once the sum exceeds the previously established bound, we know that the present set is not a contender for the most efficient encoding. This saves some work.

To improve on the previous bound, a T-augmentation leading to a more efficient set must change the code length distribution for codeword lengths up to the length at which the initial set's redundancy summation exceeded the bound. This requires a T-prefix of less than that length. Invoking the results on the length order of T-prefixes from above, we obtain a replacement value for L in Equation (10) for further T-augmentations with the present set distribution as the base distribution.

5.7 Applying the Redundancy Criterion at Both Ends of the Queue

This technique only applies to the breadth-first search, but it is quite powerful. In a virtual T-augmentation, the code length distribution does not change for lengths below $|p_{n+1}|$, and the symbol that would have been encoded with p_{n+1} will now have to be encoded by a longer codeword. Thus, the portion of the redundancy attributable to the codewords of lengths up to and including $|p_{n+1}|$ represents a residual redundancy that does not disappear for any distributions derived from it (due to the non-decreasing order of the T-prefix lengths used).

If this residual redundancy is larger than the best redundancy found so far, then there is no point in investigating the distribution any further. If we add a distribution to the queue, it has to be feasible at the time we add it. However, by the time it arrives at the other end of the queue, the best redundancy may have improved, and a previously feasible distribution may no longer be feasible. In practice, it pays to add the residual redundancy at the time of queueing to the object queued. This enables a quick check at the time of dequeueing to discard such "expired" distributions.

5.8 Maximum Feasible Codeword Length

Another significant saving can be made if we acknowledge that codewords above a certain length are simply of no interest. Given a coding alphabet with $\#S$ symbols, we know that any complete code set C over S with a maximal codeword length of $|\hat{x}|$ must have a minimum number of codewords:

$$\#C \geq (|\hat{x}| - 1)(\#S - 1) + \#S. \tag{11}$$

If we require exactly $N_s = \#C$ codewords, we obtain a bound on the maximum codeword length possible:

$$L_m = |\hat{x}| = \left\lceil \frac{N_s - 1}{\#S - 1} \right\rceil, \tag{12}$$

where $\lceil q \rceil$ denotes the smallest integer greater than or equal to q. Given N_s, we may choose $k = L_m - 1$ and any $p \in S$ to obtain a T-code set $S_{(p)}^{(k)}$ such that the longest codewords in $S_{(p)}^{(k)}$ are of length L_m. Since $S_{(p)}^{(k)}$ has a sufficient number of codewords to encode the source, we may ask whether it is possible to encode the source more efficiently with a T-code set that requires us to assign codewords longer than L_m.

Theorem 5.1 (Maximum Codeword Length) *There exists no source with N_s symbols such that the T-code set with the lowest redundancy in an encoding of that source requires the assignment of codewords longer than L_m.*

A formal proof of this theorem is an open problem. However, it is easy to give a "handwaving argument" for why the theorem is sensible: consider $S_{(p)}^{(k)}$ as above. $S_{(p)}^{(k)}$ is not only a T-code set, but also the possible outcome of a Huffman code construction process. Of all Huffman codes possible for a source with

N_s characters, $S_{(p)}^{(k)}$ is the one with the longest possible codewords. Since the Huffman code construction yields a minimum redundancy code for a given source, all possible Huffman codes for the source

- yield the same or a lower redundancy than $S_{(p)}^{(k)}$, and
- use only codewords up to length L_m.

As $S_{(p)}^{(k)}$ is a T-code set, it sets a bound on the redundancy for the set we wish to find. The redundancy criterion we have introduced above may be applied here in a similar way — any improvement in the redundancy that could be achieved requires an increase in the number of codewords that are shorter than L_m. This then enables "codeword assignment swapping", i.e., the codeword(s) that are used as T-prefix(es) disappear, but their loss must be outweighed by a gain from the additional codewords created, i.e., it must be possible for source symbols with previously longer assignments to "move up" the tree. Since the longest possible assignment in a completely filled tree is of length L_m, any "better" tree cannot have any longer codewords assigned.

This leaves only those sets with codewords up to length L_m or less to consider.

□

Thus we may assume that the virtual T-augmentations in our algorithm do not need to keep track of codeword lengths larger than L_m.

Furthermore, it means that a virtual T-augmentation is not feasible unless

$$k|p| + \ell_2 < L_m. \tag{13}$$

This complements the already established rule for L: we now have a feasibility criterion that also works for sets that are too small to encode the source. For larger sets, we may simply replace L_m by L.

It should be noted that the other feasibility criteria mentioned in this chapter ensure on their own that the algorithm will terminate. However, in practice, the main benefit of the theorem is that it permits some savings to be made with respect to memory requirements and computation time, as the size of the distributions is limited.

5.9 Dropping the Logarithms

If we take a closer look at Equation (1), we notice that the term with the logarithms of the source symbol probabilities is constant. We may hence drop it when comparing the redundancies of two sets.

6 Performance of the Search Algorithm

The algorithm presented above is still expensive in both execution time and memory requirement (queue). Both depend primarily on the number of set distributions that have to be searched. As a general rule, this increases with N_s. However, due to the redundancy bound feasibility criterion, it also has a strong dependence on the source symbol probabilities. Moreover, there is a dependence on the shape/skew of the source probability distribution. As a rule of thumb, the less skewed the distribution, the more sets need to be searched. This is mainly a result of the redundancy criterion — highly skewed sources are governed by the probabilities for the most frequent codewords, which ensures a high residual redundancy at small T-prefix lengths. Consequently, sources with equiprobable symbols take longest to match.

Figure 3 shows the number of sets searched for N_s source symbol probabilities, using a highly skewed distribution. It also shows the CPU time taken by a 667 MHz Compaq AlphaServer DS20E for which the algorithm was implemented in C as a CMEX function for MATLAB. It is evident from the spread of the data that the source probability distribution is itself the dominant factor in determining the execution time of the algorithm. For such highly skewed

distributions, the new algorithm seems to search only $O(N_s^2)$ distributions. Given that it takes $O(N_s)$ steps to investigate a distribution, the computational complexity of the algorithm is $O(N_s^3)$ for highly skewed source probability distributions. This compares favourably to the algorithm in [13], where even the number of distributions searched grew more than exponentially with N_s. For $N_s = 80$, the number of distributions searched is now typically three orders of magnitude below that of the algorithm in [13].

Figure 4 shows the number of sets searched for N_s source symbol probabilities, this time for both highly skewed distributions such as in Figure 3 and for equiprobable symbols. It is evident that the complexity of the search is much higher for the latter. This comes as no surprise since equiprobable symbols minimze the residual redundancy in each case, thus blunting the redundancy criterion.

7 Breadth-First Now Wins

The introduction of queue-front vetting represents an improvement to the breadth-first algorithm that renders it, on average, faster than the depth-first algorithm presented in [13]. This is consistent with the observation that the important decisions about coding efficieny are made during the first few T-augmentations, which allow the feasibility criteria to be tightened early on in the search. An exception are distribtions for which the symbols are either equiprobable or almost equiprobable – depth-first searches fewer sets than breadth-first here, although the margin for distributions up to 40 source symbols seems to be comparatively small (breadth-first searches about 13% more sets at 40 symbols). In the high-skew case, breadth-first wins by a factor of typically 2 or 3 for $N_s = 40$ and by over an order of magnitude for $N_s = 100$.

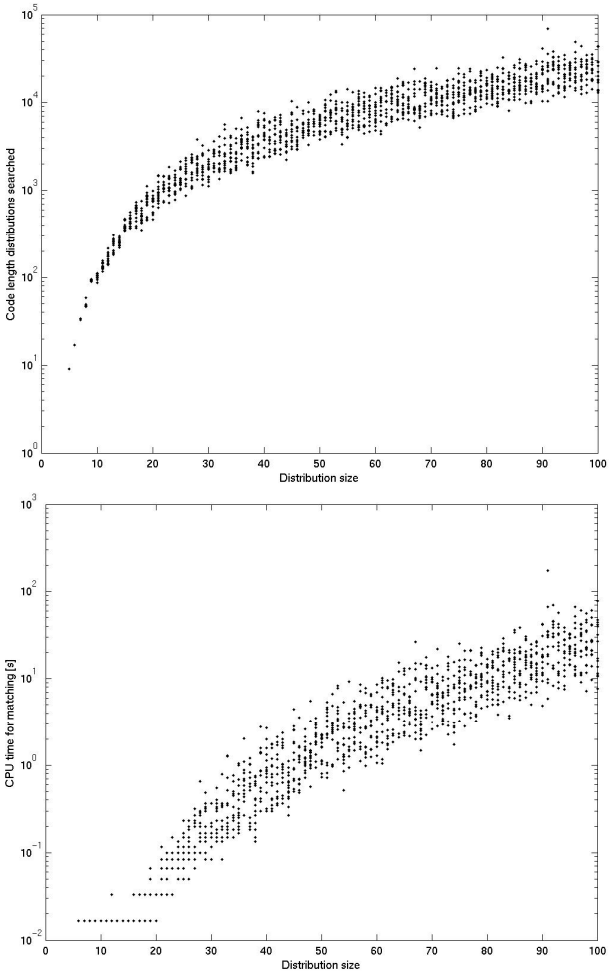

Figure 3: The number of code length distributions that have to be searched, and hence the execution time of the matching algorithm presented in this paper, depend primarily on the number of source symbols, N_s, and their associated probabilities of occurrence. In each of the above plots, fifteen skewed probability distributions were randomly generated for each N_s and used as input to the matching algorithm (assuming a binary alphabet). Each marker in the top plot shows the number of codelength distributions searched for a particular probability distribution. The bottom plot shows the CPU times taken. It is evident that the source probability distribution itself has a large influence on the number of sets that need to be searched.

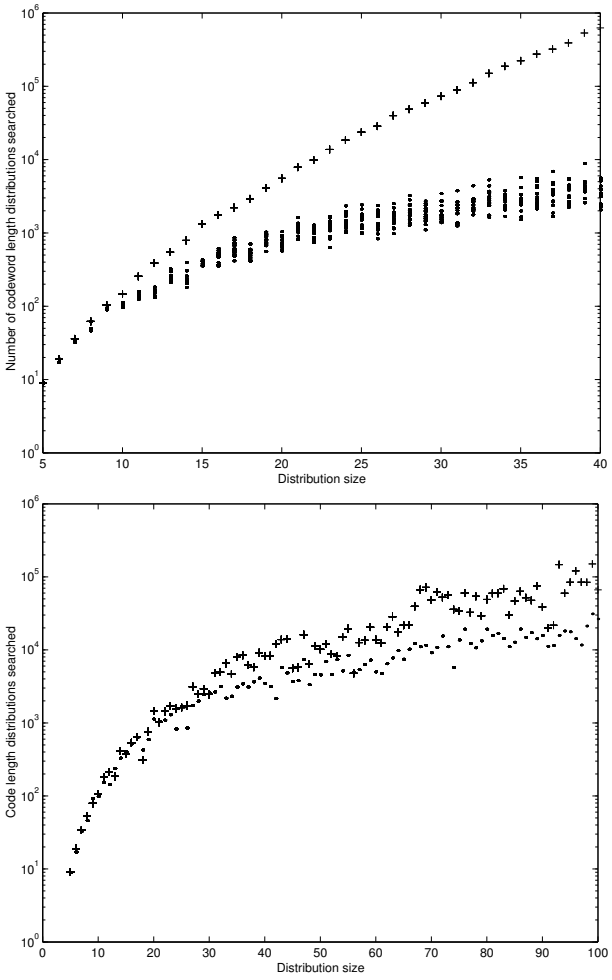

Figure 4: The number of sets that need to be searched to find the best match for a given source size N_s depends strongly on the source probabilities. For the top plot above, 15 probability distributions of size $N_s = 40$ were randomly generated with a high skew (lowest probability of occurrence was about ten orders of magnitude below the highest). These are represented by dots. The crosses represent the case of equiprobable symbols, for which the highest number of sets are searched. Bottom: The breadth-first algorithm (dots) now generally outperforms the depth-first algorithm (crosses) for source probability distributions with high skews.

8 Discussion

When we wish to find the most efficient T-code set for a given set of source symbol probabilities, the constraints set by the T-augmentation construction prevent us from using the well-known Huffman code construction algorithm. To date, the only exact solution to this problem appears to be an exhaustive search of all feasible T-code code length distributions.

The number of distributions that need to be searched for this purpose may be restricted by the use of a number of feasibility criteria in a branch-and-bound algorithm [13]. The algorithm's output are the T-expansion parameters and T-prefix lengths of a T-code set with minimal redundancy. This leaves a choice when it comes to picking the T-prefixes. It is thus possible to select a set on the basis of other properties, such as a minimal expected synchronisation delay.

The introduction of additional feasibility criteria has now led to the algorithm presented above, which yields a substantial improvement on the algorithm presented by the author in [13] for high-skew sources.

While these recent improvements give the algorithm practical value, its computational complexity of at best $O(N_s^3)$ still compares unfavourably to the Huffman algorithm, which is of order $O({N_s}^2)$.

Further improvements to the method presented are conceivable. For example, one approach could be in the form of new feasibility criteria – the author doubts that he has exhausted all possibilities. Testing for feasibility criteria carries a cost which is multiplied by the number of distributions that need to be searched. Additional criteria are more likely to be of benefit if they lead to a "pruning" of the search tree close to the root, or if they rule out a large proportion of sets that would otherwise be searched. Cost and benefit have to be weighed against each other. Since we now seem to be dealing with smaller search trees, the total cost of additional tests has fallen — but the benefits may decrease, too.

Another improvement could derive from the parallelisation of the algorithm. Given a base distribution, the recursive calls for different virtual T-augmentations may be processed in parallel, on a multi-processor machine or over a distributed network with a good load balancing scheme. The gains here might extend beyond those due to the extra processing power available: the initial shape of most T-code code length distributions is formed during the first few T-augmentations, and these short codewords have the most significant influence on the code's redundancy. If these short distributions could be processed in parallel, updated (i.e., lower) bounds for the branch-and-bound could become available earlier and might save parallel searches from unnecessary recursions.

For some practical implementations, the algorithm suggested by Titchener may also prove to be the most economic. This could be the case especially when large sources are involved, the encoding speed is paramount, and the efficiency is only of secondary importance.

9 Acknowledgements

My thanks go to Mark Titchener for his interest in the matching problem, to Aaron Gulliver for suggesting the breadth-first approach in the first place, and to Wei Ching Tham, Wang Hongqiang, and Todd K. Moon for prompting me to revisit the problem.

References

[1] C. E. Shannon: *A Mathematical Theory of Communications*. Bell Systems Technical Journal, 27:379, July 1948

[2] D. Huffman: *A Method for the Construction of Minimum Redundancy Codes*. Proc. Inst. Radio Eng., 40:1098-1101, September 1952

[3] R. G. Gallager: *Variations on a Theme by Huffman*. IEEE Trans. Inform. Theory, 24(6):668-674, November 1978

[4] O. Johnsen: *On the Redundancy of Binary Huffman Codes*. IEEE Trans. Inform. Theory, 26(2):220-222, 1980.

[5] R. M. Capocelli, R. Giancarlo, and I. J. Taneja: *Bounds on the Redundancy of Huffman Codes*. IEEE Trans. Inform. Theory, 32(6):854-857, November 1986

[6] R. Nicolescu: *Uniqueness Theorems for T-Codes*. Technical Report. Tamaki Report Series no.9, The University of Auckland, 1995.

[7] M. R. Titchener: *Generalized T-Codes: an Extended Construction Algorithm for Self-Synchronizing Variable-Length Codes*, IEE Proceedings – Computers and Digital Techniques, 143(3), June 1996, pp. 122-128.

[8] U. Guenther: *Data Compression and Serial Communication with Generalized T-Codes*, Journal of Universal Computer Science, V. 2, N 11, 1996, pp. 769-795. `http://www.iicm.edu/jucs_2_11`

[9] G. R. Higgie: *Database of best T-codes*, IEE Proceedings – Computers and Digital Techniques, Volume 143(4), July 1996, pp. 213 -218

[10] U. Guenther, P. Hertling, R. Nicolescu, and M. R. Titchener: *Representing Variable-Length Codes in Fixed-Length T-Depletion Format in Encoders and Decoders*, CDMTCS Research Report no.44, Centre of Discrete Mathematics and Theoretical Computer Science, The University of Auckland, August 1997. `http://www.cs.auckland.ac.nz/research/CDMTCS/docs/pubs.html`.

[11] U. Guenther, P. Hertling, R. Nicolescu, and M. R. Titchener: *Representing Variable-Length Codes in Fixed-Length T-Depletion Format in Encoders and Decoders*, Journal of Universal Computer Science, 3(11), November 1997

http://www.iicm.edu/jucs_3_11, pp. 1207–1225.

[12] R. Nicolescu and M. R. Titchener, *Uniqueness Theorems for T-Codes*, Romanian Journal of Information Science and Technology, 1(3), March 1998, pp. 243–258.

[13] U. Guenther: *Robust Source Coding with Generalized T-Codes*. PhD Thesis, The University of Auckland, 1998. http://www.tcs.auckland.ac.nz/~phd.pdf

[14] W. Ebeling, R. Steuer, and M. R. Titchener: *Partition-Based Entropies of Deterministic and Stochastic Maps*, accepted for publication in *Stochastics and Dynamics*.

[15] M. R. Titchener: *verbal communication*