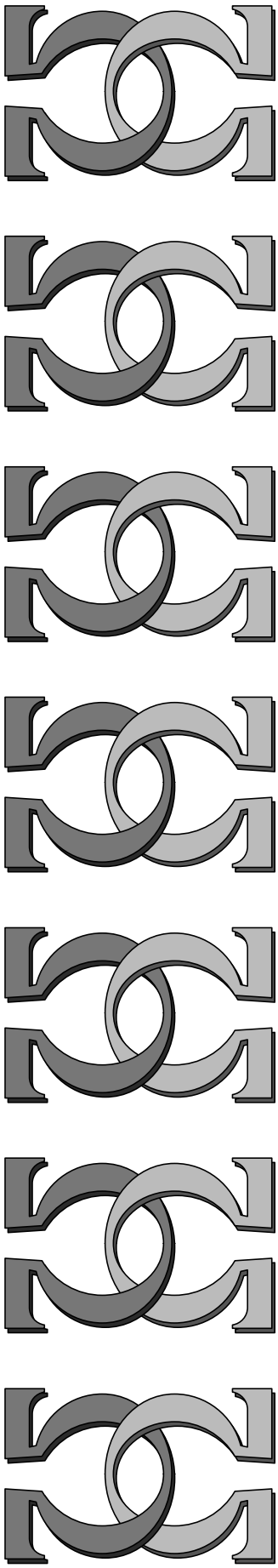


CDMTCS
Research
Report
Series

T-Complexity and T-Information Theory – an Executive Summary

Ulrich Günther
Department of Computer Science
University of Auckland

CDMTCS-149
February 2001

Centre for Discrete Mathematics and
Theoretical Computer Science

T-Complexity and T-Information Theory – an Executive Summary

Ulrich Günther
Department of Computer Science
The University of Auckland

February 19, 2001

Abstract

This paper describes the derivation of the T-Complexity and T-Information Theory from the decomposition of finite strings, based on the duality of strings and variable-length T-Codes. It further outlines its similarity to the string parsing algorithm by Lempel and Ziv. It is intended as a summary of work published mainly by Titchener and Nicolescu.

1 A brief Introduction to T-Codes

This paper first gives an introduction to T-Codes and their construction technique, as this is fundamental for the understanding of the T-decomposition algorithm that underpins the T-complexity measure.

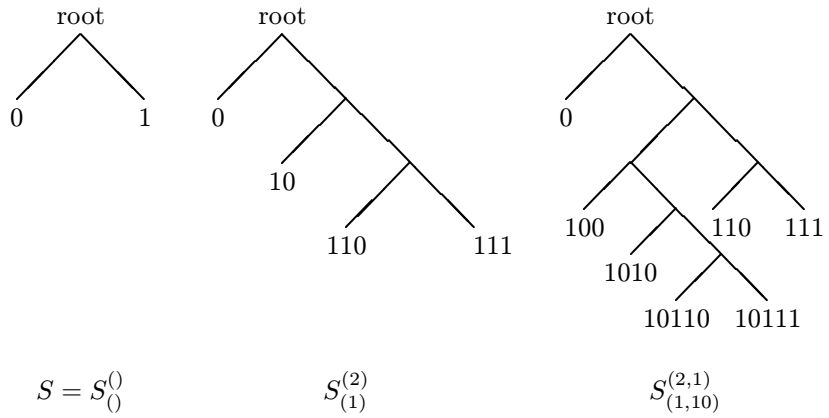
T-Codes [3, 4, 11] are similar to Huffman codes in that they are codes with variable-length codewords. Depending on the source symbol probabilities, a Huffman code may also be a T-Code. In fact, any T-Code set could theoretically have been constructed as a result of a Huffman code construction algorithm.

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 a finite alphabet S (e.g., the binary alphabet, where $S = \{0, 1\}$). 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



.

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

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.

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

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

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 1 depicts the construction of the T-Code set $S^{(2,1)}_{(1,10)}$. In the first T-augmentation in this example, the tree for S is copied three times and the three copies are linked to each other via the respective leaf nodes corresponding

to the codeword 1. The second T-augmentation links two copies of the tree for $S^{(2)}_{(1)}$ via the leaf node 10.

Another example, using codeword lists rather than trees, is shown in Table 1. By T-augmenting over and over again, we can generate sets of arbitrary size.

By choosing the T-prefixes and the T-expansion parameters wisely, a T-Code tree may be constructed to suit a particular source. However, this is not of prime relevance here.

It is worth noting here that some T-Code sets may be constructed with more than one set of T-prefixes and T-expansion parameters. E.g., the set $S^{(3)}_{(0)}$ is the same as the set $S^{(1,1)}_{(0,00)}$. A set of T-prefixes and T-expansion parameters used in the construction of a T-Code set is called a "T-prescription". As the example illustrates, a T-Code set may have more than one T-prescription. However, all T-prescriptions for a given T-Code set can be derived from each other with ease. For the purposes of this paper we shall assume that, if several T-prescriptions exist, we will always refer to the one T-prescription for which the T-expansion parameters are maximised (anti-canonical T-prescription). For a detailed discussion of this topic see [2] or [8]. A less rigorous version of the proof may also be found in [11].

2 The Significance of the Longest Codewords

Consider the two longest codewords in the sets of the examples in Table 1 and Figure 1. For each set, they differ by exactly one letter, the last one. In these binary sets, there are exactly two longest codewords per set. More generally, the number of longest codewords equals the cardinality of the alphabet. Furthermore, the longest codewords contain – in reverse succession – all the T-prefixes that were used in the construction of the set. The length of the runs of the T-prefixes in the longest codewords equals the T-expansion parameter (in the T-prescription for which the T-expansion parameter is maximised).

As it turns out, it is possible to derive each T-Code set from either of its longest codewords. The algorithm for this is described in the next section. For the moment, let us simply presume that it exists.

Furthermore, given an arbitrary finite string over an arbitrary finite alphabet, it is always possible to find a T-Code set for which this string is one of its longest codewords. This set is unique, i.e., there is no other T-Code set for which the same string is also one of the longest codewords. For a proof of this theorem see [2] or [8].

Since the longest codewords in a T-Code set are identical except for the last letter, we may regard their common part as an identifier for the T-Code set.

Note that this duality between strings and T-Code sets permits us to think of the T-Code set construction algorithm not only as a code construction algorithm, *but also as a string construction (production) algorithm.* The T-augmentations are the steps in this algorithm.

	T-augmentation level			
n	0	1	2	3
k_n	n/a	1	1	3
set	S	$S^{(1)}_{(1)}$	$S^{(1,1)}_{(1,10)}$	$S^{(1,1,3)}_{(1,10,0)}$
	0	0	0	$\not\emptyset$
	1	$\not 1$	–	–
		10	$\not 1\not\emptyset$	–
		11	11	11
			100	100
			–	–
			1010	1010
			1011	1011
				$\not\emptyset\not\emptyset$
				–
				–
				011
				0100
				–
				01010
				01011
				$\not\emptyset\not\emptyset\not\emptyset$
				–
				–
				0011
				00100
				–
				001010
				001011
				0000
				–
				–
				00011
				000100
				–
				0001010
				0001011

Table 1: T-augmentation from the binary alphabet S via the intermediate T-Code sets $S^{(1)}_{(1)}$ and $S^{(1,1)}_{(1,10)}$ to the final set $S^{(1,1,3)}_{(1,10,0)}$. The columns show the codewords in the respective T-Code sets. The "deleted" strings are the now internal nodes of the new decoding tree.

If we consider the string production aspect, we can create (or lengthen) a string as follows:

1. take an existing string (which may consist of just one letter) and consider the T-Code set $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ for which it is one of the longest codewords.

2. pick a codeword p_{n+1} from $S^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$ and append k_{n+1} copies of it to the left of the string. The new string is now one of the longest codewords from $S^{(k_1,k_2,\ldots,k_{n+1})}_{(p_1,p_2,\ldots,p_{n+1})}$.

If we repeat this as often as we desire, we can generate arbitrarily long strings.

Consider now the choice of T-prefix codeword and T-expansion parameter: if we choose a long p_{n+1} to append to the left, we create a longer resulting string than by choosing a short codeword. However, the number of steps needed to create the string remains constant - and the patterns that the "extra" bit of string includes are indeed all patterns we have already seen as T-prefixes in the initial set (or, in other words, as substrings in the original string).

Similarly, by choosing a large k_{n+1}, one does not add much extra information, but merely repeats an already occurring pattern. Note that there is a similar theme of construction steps and focus on recurring patterns in the LZ algorithm [1].

Before we take this theme further, however, we need to discuss how an existing string can be parsed to yield the associated T-Code set.

The next section describes how one arrives at the T-augmentation construction recipe for a string, i.e., at the T-prescription for the corresponding T-Code set. The parsing algorithm used to obtain it is called "T-decomposition".

3 T-Decomposition

Suppose that, for a given string x and a letter a from the alphabet S, we want to find the T-Code set for which xa is one of the longest codewords. Consider the following algorithm:

1. Set $m = 0$.

2. Decode xa as a string of codewords from $S^{(k_1,k_2,\ldots,k_m)}_{(p_1,p_2,\ldots,p_m)}$.

3. If xa decoded into a single codeword from $S^{(k_1,k_2,\ldots,k_m)}_{(p_1,p_2,\ldots,p_m)}$, set $n = m$ and finish.

4. Otherwise, set the T-prefix p_{m+1} to be the second-to-last codeword in the decoding over $S^{(k_1,k_2,\ldots,k_m)}_{(p_1,p_2,\ldots,p_m)}$.

5. Count the number of adjacent copies of p_{m+1} that immediately precede the second-to-last codeword. Add 1 to this number, and define it to be the T-expansion parameter k_{m+1}.

6. T-augment with p_{m+1} and k_{m+1}.

7. Increment m by 1 and goto step 2 above.

Example: let $x = 011000101010$ and $a = 0$, and let $xa = 0110001010100$ be the longest codeword in some T-Code set. Decoded over $S = \{0, 1\}$, we obtain the following codeword boundaries indicated by a dot:

$$xa = 0.1.1.0.0.0.1.0.1.0.1.0.0.$$

from which we identify $p_1 = 0$ and $k_1 = 1$. Decoded over $S^{(1)}_{(0)}$ we obtain

$$xa = 01.1.00.01.01.01.00.$$

i.e., $p_2 = 01$ and $k_2 = 3$. Hence, decoded over $S^{(1,3)}_{(0,01)}$, we get

$$xa = 011.00.01010100.$$

such that $p_3 = 00$, $k_3 = 1$, and $p_4 = 011$ with $k_4 = 1$. The reader may wish to verify that $xa = 0110001010100$ is indeed one of the longest codewords of $S^{(1,3,1,1)}_{(0,01,00,011)}$.

One can also regard this mechanism as a succesive elimination of lower-level codeword boundaries, as shown in the following graphic for a different string:

S	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	
$S^{(1)}_{(1)}$	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	
$S^{(1,1)}_{(1,10)}$	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	
$S^{(1,1,3)}_{(1,10,0)}$	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	
$S^{(1,1,3,1)}_{(1,10,0,11)}$	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	
$S^{(1,1,3,1,1)}_{(1,10,0,11,1010)}$	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	
$S^{(1,1,3,1,1,1)}_{(1,10,0,11,1010,1010011)}$	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	
$S^{(1,1,3,1,1,1,2)}_{(1,10,0,11,1010,1010011,100)}$	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	1	1	1	1	

N.B.: The above algorithm has been simplified to run more efficiently. The reason for this lies in the well-explained self-synchronisation of T-Codes. A T-Code decoder will not only inherently self-synchronize: Provided that there are no errors in the string that is being decoded, it can also tell whether it has achieved synchronisation with respect to the T-Code set it uses for decoding. Hence we usually do not *have* to decode the entire string in each pass — the decoding of a

suffix of the string is all we need, provided it enables the decoder to synchronize fully *before* the decoder hits the run of the T-prefixes for the next T-augmentation.

A more detailed treatment of T-decomposition using the current notation may be found in [11]. As mentioned above, the original proof may be found in [2, 8].

4 T-Complexity

When Lempel and Ziv [1] proposed their production complexity, they recognised that the number of parsing steps would give a meaningful measure of string complexity.

Titchener pursued a similar thought and proposed a "T-complexity" measure $C_T(xa)$ as follows [10, 7, 9, 13]:

$$C_T(xa) = \sum_{i=1}^{n} \log_2(k_i + 1) \tag{2}$$

where the k_i are the T-expansion parameters found in the decomposition of xa. The units of C_T are effective T-augmentation steps, or *taugs*.

This measure was published in [7] and has since discussed in several other papers by Titchener [9, 10, 13] and in a paper by Titchener, Fenwick, and Chen [12]. A further paper by Ebeling, Steuer, and Titchener [14] is in print.

5 Bounds on the T-Complexity

Titchener further contemplated on the question which strings would deliver maximal/minimal T-complexity for a given length.

For a given string length L, minimal T-complexity is delivered by strings that contain $L - 1$ copies of a single letter, repeated over their entire length, followed by an arbitrary letter. If L is the length of the string xa, then we have a single step and $C_T(xa) = \log_2 L$.

Maximal T-complexity is delivered, among others, by strings that satisfy both of the following two conditions:

1. A T-prescription can be found for the string such that all T-expansion parameters in the T-prescription are equal to one.

2. All of the T-prefixes in that T-prescription are at most as long as the shortest codeword from the T-Code set that the string represents,

This ensures a maximum number of T-augmentations for a given length of string. Note that these two conditions together are sufficient, but not necessary: If we calculate the T-complexity of all strings of length L, then there there must obviously always be at least one string whose T-complexity is maximal. However, there are certain lengths for which no strings exist over a particular alphabet that satisfy the two conditions above (e.g., there are binary strings

with $L = 5$ and $L = 7$ that satisfy them, but no such strings exist for $L = 6$ — one would need to T-augment with a T-prefix of length 3 while there is still a codeword of length 2 left in the set).

The construction rule for strings with maximal T-complexity demands that, for a given alphabet, there must be an upper bound for C_T. Titchener found by experimentation that the logarithmic integral $li(L \ln \#S) = \int_0^{L \ln \#S} \frac{dq}{\ln q}$ seems to provide such an upper bound, asymptotically. Moreover, he found that the maximal T-complexity appears to converge rapidly towards this bound for strings that are only a few dozen letters long. A proof of this theorem has yet to be given — it is currently based on experimental evidence only.

Experimental evidence also shows the following:

- If all strings of a given length L are analyzed, they show on average a high but sub-maximal C_T. This is easily explained in that "random" strings will, during their production in a random process, pick short T-prefixes with a high likelihood — but that likelihood is not equal to one, and occasionally a longer T-prefix is picked. Most of the strings of a given length L have T-complexities that fall within a very narrow band. As L increases, the distribution of C_T values for strings of length L seems to become increasingly peaked and *seems to drop away from the maximum value of C_T for that length.*
- Strings that would generally be regarded as being non-random (e.g., representations of rational numbers) fall outside of this peak. Irrational numbers such as π, $\sqrt{2}$ etc. or strings produced by natural random processes such as radioactive decay seem to fall inside the band.

6 T-Information and T-Entropy

For practical purposes, i.e., comparisons between strings of different length and their substrings, a nonlinear concave function such as the logarithmic integral is a bit unwieldy.

One can argue that a string with maximal C_T that is T-augmented a number of times, each time with one of the – respectively – shortest T-prefixes available, has information added to it at a high and – approximately – constant rate over its length.

Applying the inverse logarithmic integral to C_T thus results in "linear-looking" curves for both strings with maximal C_T and for strings that fall within the narrow band mentioned above.

Titchener recognized this [13] and thus defined an information measure I_T as:

$$I_T = \mathrm{li}^{-1}(C_T) \tag{3}$$

where li^{-1} is the inverse logarithmic integral.

He further defined a T-entropy as $H_T = \Delta I_T / \Delta L$, i.e., the rate of change of I_T along the string. Furthermore, he defined $\bar{H}_T = I_T / L$ as the average T-entropy.

7 Conclusions

The T-complexity definition seems reasonable given a similar approach to string complexity by Lempel and Ziv [1]. From this, the derivation of T-information and T-entropy also seem to be reasonable steps to take.

Experimental evidence suggests that they produce "meaningful" results. For example, recent results by Ebeling, Steuer, and Titchener show that T-entropy and the Kolmogorov-Sinai entropy seem to be closely related in nonlinear (symbolic) dynamics [14].

To demonstrate this, a web site has been set up which allows online upload and analysis of files representing strings:

`http://www.tcs.auckland.ac.nz/~ulrich/cgi-bin/staff/tcomplexityplot9.cgi`

Another web site, exploring the possibility of an application in similarity searching, has also been set up:

`http://www.tcs.auckland.ac.nz/~ulrich/cgi-bin/similarity.cgi`

References

[1] A. Lempel and J. Ziv: *On the Complexity of Finite Sequences*. IEEE Trans. Inform. Theory", 22(1), January 1976, pp. 75-81.

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

[3] 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.

[4] 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`

[5] 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`.

[6] 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, pp. 1207–1225. `http://www.iicm.edu/jucs_3_11`.

[7] M. R. Titchener: *A Deterministic Theory of Complexity, Information and Entropy*, *IEEE Information Theory Workshop*, February 1998, San Diego.

[8] 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.

[9] M. R. Titchener, *A novel deterministic approach to evaluating the entropy of language texts*, *Third International Conference on Information Theoretic Approaches to Logic, Language and Computation*, June 16-19, 1998, Hsitou, Taiwan.

[10] M. R. Titchener, *Deterministic computation of string complexity, information and entropy*, *International Symposium on Information Theory*, August 16-21, 1998, MIT, Boston.

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

[12] M. R. Titchener, P. M. Fenwick, and M. C. Chen: *Towards a Calibrated Corpus for Compression Testing*, Data Compression Conference, DCC-99, Snowbird, Utah, March 1999.

[13] M. R. Titchener: *A measure of Information*, IEEE Data Compression Conference, Snowbird, Utah, March 2000.

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