Masaryk University in Brno
Faculty of Informatics

Competing Patterns in Language Engineering and Computer Typesetting

Petr Sojka

Ph.D. Dissertation

Supervisor: Karel Pala

Brno, January 2005

Competing Patterns in Language Engineering and Computer Typesetting

A Dissertation
Presented to the Faculty of Informatics
of Masaryk University in Brno
in Partial Fullfillment of the Requirements
for the Ph.D. Degree

by
Petr Sojka
January 2005

Abstract

The goal of this dissertation is to explore models, methods and methodologies for machine learning of the compact and effective storage of empirical data in the areas of language engineering and computer typesetting, with a focus on the massive exception handling.

Research has focused on the pattern-driven approach. The whole methodology of so called *competing patterns* capable of handling exceptions to be found so widely in natural language data and computer typesetting, is further developed. Competing patterns can store *context dependent* information and can be learnt from data, or written by experts, or combined together.

In the first part of the thesis, the theory of competing patterns is built; competing patterns are defined, cornerstones of methodology based on stratified sampling, bootstrapping and problem modeling by competing patterns are described. Segmentation problems (hyphenation) and problems of disambiguation of tagged data in corpus linguistics are used as examples when developing formal model of the competing patterns method.

The second part consist of a series of seven published papers that describe problems addressed by the proposed methods: applications of competing patterns and related learning methods in areas of hyphenation, hyphenation of compound words and, for example, the segmentation of Thai texts.

Key Words

competing patterns • context-sensitive patterns • machine learning • natural language engineering • hyphenation • segmentation • Thai text segmentation • disambiguation • formal concept analysis • part of speech tagging

Declaration

I declare that this dissertation thesis is the result of my work carried out at the Faculty of Informatics, Masaryk University in Brno, Czech Republic. The work described here, is to the best of my knowledge, all my own, except where clear reference is made to the work of others. The material contained herein has not been submitted in whole or in part for a degree, diploma, professional or any other qualification at this or another university.

Petr Sojka

Acknowledgments

Feeling gratitude and not expressing it
is like wrapping a present and not giving it.
— William Arthur Ward

This thesis would not be the same without the support of many people that deserve my thanks. It is my pleasure to thank them all.

Jiří Hořejš taught me the basics of Computer Science and helped me with my first steps in the field. Under his guidance I learnt my first research skills.

The work on the thesis has been supervised by Karel Pala, saving me from dead-ends and narrowing my concentration of the research topics. I would like to express my gratitude for his continuous encouragement during the long period of doing research presented in this thesis. I have got many words of encouragement from my colleagues: Jozef Gruska, Ivan Kopeček, Jaroslav Král, Jiří Sochor and Pavel Zezula, to mention only some of them.

David Antoš, Pavel Ševeček and Pavel Smrž worked with me on the problems presented in the second part of this thesis and provided valuable feedback and discussions.

The thesis writing has included other tasks: the draft has been proofread by Libor Škarvada and Tomáš Hála and corrections to my English prose were done by Silvie Bilková, Michael L. G. Hill and James Thomas.

Finally, I owe my deepest gratitude to my family for their emotional support and patience, especially during the final stages of the thesis preparation under the pressure of the deadline, when they tolerated me working long hours in order to achieve completion of this work.

Contents

List of Tables

List of Figures

Terms and Notation

Symbol	Meaning
Σ	an arbitrary finite non-empty set of symbols
ε	an empty string
Σ^*	the set of all finite sequences of the elements of Σ including ε
Σ^+	the set of all finite sequences of the elements of Σ (without ε)
string	an element of Σ^*
$\|s\|$	the length of a string s
$\|\Sigma\|$	the cardinality of a set Σ
$\mathbb{N}$	the set of all positive integers
$\mathbb{R}$	the set of all real numbers
$\{a_1, a_2, \ldots, a_n\}$	the enumeration of elements of a finite set
$W \setminus C$	set difference $\{x \mid x \in W \wedge x \notin C\}$
$\cdot$ (centered dot)	concatenation of symbols
. (dot symbol)	*begin of word marker* and *end of word marker*

PART I

THE THEORY OF COMPETING PATTERNS

Chapter 1

Introduction

"Everything is a symbol, and symbols can be combined to form *patterns*. Patterns are beautiful and revelatory of larger truths. These are the central ideas in the thinking of Kurt Gödel, M. C. Escher, and Johann Sebastian Bach, perhaps the three greatest minds of the past quarter-millennium." (Hofstadter, 1979) Recognition of patterns is considered as the central issue in intelligence. Artificial intelligence needs *statistical emergence* (Hofstadter, 1983): for real semantics, symbols must be *decomposable,* complex, autonomous — active. The proper handling of various types of patterns is essential in the many levels of *natural language processing* (NLP) and other fields.

The problems of computerized natural language processing and computer typesetting have been tackled by both linguists and computer scientists for decades. As the available computing power steadily grows, new approaches recently deemed impossible are becoming reality — example being *corpus linguistics* (Armstrong et al., 1999; Young and Bloothooft, 1997; Boguraev and Pustejovsky, 1996). The so-called *empirical approaches* are used for machine learning of language phenomena: from huge language data (corpora, wordlists), language models and patterns are learnt by sophisticated algorithms through *machine learning* techniques. As an example of this shift, successful unsupervised learning of natural language morphology from language word lists has been reported recently (Goldsmith, 2001). These merely statistical approaches work quite well for many tasks in the area of computational linguistics, and quickly reach above 90% efficiency in tasks such as part of speech tagging, sentence segmentation, speech recognition or probabilistic parsing. The main drawback of a solely statistical approach is that the results of learning methods are usually not understandable by expert linguists, as the language models are hidden in weights of synapses of neural nets or in zillions of probabilities or conditioned grammar rules. It appears that going the "last mile", increasing the remaining few percent is not feasible, and ways to cover the remaining exceptions are being sought.

Conversely, a rule-based approach, such as, when the results of the learning process are human-understandable *rules* or *patterns*, allows for the merging of hand-crafted and machine learnt knowledge. It is becoming clear that a close cooperation between computer scientists and linguists is necessary (Brill et al., 1998)—both sides need each other. Neither rigorous computational analysis and formal models nor linguistic introspection and language models should be absent in successful approaches.

Patterns can be identified as a set of objects that share some common property (a formal definition is to be found in Chapter 2 on page 7). During the emergence of patterns covering the rules in data, some *exceptions* may occur. Remaining errors and exceptions covered in the first level can be viewed again as set of objects and described by *inhibiting patterns*. The next layer of *covering patterns* may describe the patterns in the data not handled by previous rules, and so on. By this process, knowledge from the data can be learnt, either by an automatic procedure, or by information fusion from different sources.

There is plethora of methods of machine learning (Michalski et al., 1983a; Michalski et al., 1983b; Mitchell, 1997), data mining and knowledge management (Shi et al., 2004), structural pattern recognition (Schlesinger and Hlaváč, 2002), exploratory data analysis (Hájek and Havránek, 1978; Hájek et al., 2004) and association rule mining (Zhang and Zhang, 2002). However, up to now, we are not aware of an systematic attempt made to deal with the large-scale exception handling that is so widespread across linguistic data in machine learning methods and data mining. This work is one of the first attempts to formalize and fully employ the theory of competing patterns for the utilization of language data in the areas of natural language processing and computer typesetting.

1.1 Thesis Organization

The thesis is organized into two parts. As the research has been carried over last decade and results have already been published in scientific journals and conference proceedings, the main part of the thesis is the collection of seven published papers in Part II on page 27. To make the thesis focused on one topic, we are not including any of the many other papers published by the author in the last decade: (Géczy et al., 1993; Thành et al., 1996; Sojka, 1998a; Sojka, 1998b; Došlá et al., 1999; Došlá et al., 2002; Sojka, 2003a; Sojka, 2003b; Sojka, 2003c; Holeček and Sojka, 2004; Nevěřilová and Sojka, 2005), among others.

The first part consists of several chapters that describe the methodology of competing patterns. Pattern theory is formalized in Chapter 2 on page 7. An overview of the methodology of pattern development is given in Chapter 3 on page 13. A list of applications in the areas of computer typesetting and language engineering is discussed in Chapter 4 on page 17. The closing chapter of part one sums up author's contributions of this thesis and discusses the direction of further research and the employment of results.

References are listed at the end of each chapter; a collected bibliography is at the end on page 121.

References

Susan Armstrong, Kenneth Church, Pierre Isabelle, Sandra Manzi, Evelyne Tzoukermann, and David Yarowsky, editors. 1999. *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Publishers Group.

Branimir Boguraev and James Pustejovsky. 1996. *Corpus Processing for Lexical Acquisition*. MIT Press.

Eric Brill, Radu Florian, John C. Henderson, and Lidia Mangu. 1998. Beyond N-Gram: Can Linguistic Sophistication Improve Language Modeling? In *Proceedings of the ACL '98*.

Zuzana Došlá, Roman Plch, and Petr Sojka. 1999. Matematická analýza s programem Maple: 1. Diferenciální počet funkcí více proměnných (Mathematical Analysis with Program Maple: 1. Differential Calculus). CD-ROM, `http://www.math.muni.cz/~plch/mapm/`, December.

Zuzana Došlá, Roman Plch, and Petr Sojka. 2002. Matematická analýza s programem Maple: 2. Nekonečné řady (Mathematical Analysis with Maple: 2. Infinite Series). CD-ROM, `http://www.math.muni.cz/~plch/nkpm/`, December.

Peter Géczy, Petr Sojka, and Jan Blatný. 1993. Robustness and Generalization of Multilayer Neural Networks. In Igor Mokriš, editor, *Proceedings of the International Conference Image Processing and Neural Networks, Liptovský Mikuláš, 1993*, pages 163–170, Liptovský Mikuláš. Military Technical University in Liptovský Mikuláš, Slovak Electrotechnical Society of Military Technical University.

John Goldsmith. 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics*, 27(2):153–198, June.

Petr Hájek and Tomáš Havránek. 1978. *Mechanising hypothesis formation — Mathematical foundations for a general theory*. Springer-Verlag.

Petr Hájek, Jan Rauch, David Coufal, and Tomáš Feglar. 2004. The GUHA Method, Data Preprocessing and Mining. In *Database Support for Data Mining Applications*, volume LNAI 2862, pages 135–153.

Douglas R. Hofstadter. 1979. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books.

Douglas R. Hofstadter. 1983. Artificial intelligence: Subcognition as computation.

Jan Holeček and Petr Sojka. 2004. Animations in a pdfTEX-generated PDF. In Apostolos Syropoulos, Karl Berry, Yannis Haralambous, Baden Hughes, Steven Peter, and John Plaice, editors, *TEX, XML, and Digital Typography*, volume 3130 of *Lecture Notes in Computer Science*, pages 179–191, Berlin, Heidelberg, August. Springer-Verlag.

Ryszard Spencer Michalski, Jaime Guillermo Carbonell, and Tom Michael Mitchell. 1983a. *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto.

Ryszard Spencer Michalski, Jaime Guillermo Carbonell, and Tom Michael Mitchell. 1983b. *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann Publishers, Inc., Los Altos, California.

Tom Michael Mitchell. 1997. *Machine Learning*. McGraw-Hill.

Zuzana Nevěřilová and Petr Sojka. 2005. XML-Based Flexible Visualisation of Networks: Visual Browser. Submitted.

Michail I. Schlesinger and Václav Hlaváč. 2002. *Ten Lectures on Statistical and Structural Pattern Recognition*. Kluwer Academic Publishers, Dordrecht, The Netherlands, May.

Yong Shi, Weixuan Xu, and Zhengxin Chen, editors. 2004. *Data Mining and Knowledge Management: Chinese Academy of Sciences Symposium CASDMKM*, volume LNCS 3327 of *Lecture Notes in Computer Science*. Springer-Verlag, July.

Petr Sojka. 1998a. An Experience from a Digitization Project. *Cahiers GUTenberg*, (28–29):276–281, March.

Petr Sojka. 1998b. Publishing Encyclopaedia with Acrobat using TEX. In *Towards the Information-Rich Society. Proceedings of the ICCC/IFIP conference Electronic publishing '98*, pages 217–222, Budapest, Hungary, April. ICCC Press.

Petr Sojka. 2003a. Animations in PDF. In *Proceedings of the 8th SIGCSE Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2003*, page 263, Thessaloniki. Association of Computing Machinery.

Petr Sojka. 2003b. Interactive Teaching Materials in PDF using JavaScript. In *Proceedings of the 8th SIGCSE Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2003*, page 275, Thessaloniki. Association of Computing Machinery.

Petr Sojka. 2003c. Rapid Evaluation using Multiple Choice Tests and TEX. In *Proceedings of the 8th SIGCSE Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2003*, page 265, Thessaloniki. Association of Computing Machinery.

Hàn Thế Thành, Petr Sojka, and Jiří Zlatuška. 1996. TEX2PDF — Acrobatics with an Alternative to DVI Format. *TUGboat*, 17(3):244–251.

Steve Young and Gerrit Bloothooft, editors. 1997. *Corpus-Based Methods in Language and Speech Processing*. Kluwer Academic Publishers Group, Dordrecht.

Chengqi Zhang and Shichao Zhang. 2002. *Association Rule Mining*, volume LNAI 2307 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Chapter 2

Formalization of Competing Patterns

A theory of patterns can be developed using several formalisms. The choice of representation crucially determines the performance of a pattern mining process from data, and the analysis of information-theoretic properties of patterns.

In this chapter we build a competing patterns theory formally. Other approaches, as formalisms of Grenander's *general pattern theory* (Grenander, 1993), or *formal concept analysis* (Carpineto and Romano, 2004) are worth studying, too.

2.1 Basic Notions

The two fundamental problems are pattern definition and pattern recognition/generation from input data. There are many ways of formalizing patterns — sets of objects sharing some recognizable properties (attributes, structure, ...).

Definition 2.1 (pattern). By alphabet we mean a finite, nonempty set. Let us have two disjoint alphabets Σ (the alphabet of *terminals*, called also called *characters* or *literals*) and V (the alphabet of *variables*). *Patterns* are words over the free monoid $\langle(\Sigma \cup V)^*, \varepsilon, \cdot\rangle$. The length $|\varepsilon|$ of an empty word ε is zero. Patterns having only terminals are called *terminal patterns* or *literal patterns*. The length of a literal pattern p, denoted by $|p|$, is the number of literals in it. The *language* $L(\alpha)$ *defined by a pattern* α consists of all words obtained from α by leaving the terminals unchanged and substituting a terminal word for each variable v. The substitution in our case has to be *uniform*: different occurences of v are replaced by the same terminal word. If the substitution always replaces variables by a *nonempty* word, such language L_{NE} is *non-erasing*, and such pattern is called *NE-pattern*. Similarly, we define an *erasing* language L_E as a language generated by an *E-pattern* such that substitution of variable v by empty word ε is allowed.

The pattern *SVOMPT* for English sentences where the variables denote Subject, Verb, Object, Mood, Place, Time may serve as an example of E-pattern. A useful task is to infer a pattern common to all input words in a given sample by the process of *inductive inference*. It has been shown by Jiang et al. (1995) that the *inclusion problem* is undecidable for both erasing and non-erasing pattern languages. It is easy to show that the decidability of the *equivalence problem* for non-erasing languages is trivial. The decidability status of the equivalence problem for E-patterns remains open. These results show that trying to infer language description in the form of a set of patterns (or the whole grammar) automatically is very difficult task.

We focus our attention in the further study to literal patterns only.

Definition 2.2 (classifying pattern). Let A be alphabet, let $\langle A, \leq \rangle$ be a partially ordered system, $\leq$ be a *lattice order* (every finite non-empty subset of A has lower and upper bound). Let . be a distinguished symbol in $\Sigma' = \Sigma \cup \{.\}$ that denotes the beginning and the end of word — *begin of word marker* and *end of word marker*. *Classifying patterns* are the words over $\Sigma' \cup V \cup A$ such that dot symbol is allowed only at the beginning or end of patterns.

Terminal patterns are "context-free" and apply anywhere in the classified word. It is important to distinguish patterns applicable at the beginning and end of word by the dot symbol in a pattern.[1] Classifying patterns allow us to build *tagging hierarchies* on patterns.

Definition 2.3 (word classification, competing word patterns).
Let P be a set of patterns over $\Sigma' \cup V \cup A$ (*competing patterns*, *pattern set*). Let $w = w_1 w_2 \ldots w_n$ be a word to be classified with P. Classification $\mathit{classify}(w, P) = a_0 w_1 a_1 w_1 \ldots w_n a_n$ of w with respect to P is computed from a pattern set P by a competing procedure: all patterns whose projection to Σ match a substring of w are collected. a_i is supremum of all values between characters w_i and w_{i+1} in matched patterns. $\mathit{classify}(w, P)$ is also called the *winning pattern*.

It is worth noting that the classification procedure can be implemented very efficiently even for large pattern bases. Its effectiveness depends on the data structure where the patterns are stored. When an indexed trie is used, the classification of a word can be realized in linear time with respect to the word length $|w|$ and does not depend on $|P|$.

Our motivation for studying of competing patterns was the word division (hyphenation) problem. It is related to a dictionary problem — the problem of effective storage of a huge word list. An enumerated list of Czech words may have well above 6,000,000 words. Storage of such a large

1. It is itmrnopt to dgtusisinh ptatren apcbliplae at the bngninieg and end of wrod by the dot sobmyl in a ptarten.

table even using hashing requires considerable space. Another idea is to use finite-state methods — finite-state automata and transducers. It has been shown that decomposition of the problem by using *local grammars* (Gross, 1997) or building cascades of finite state machines (Hobbs et al., 1997) is a tractable, even though very time-consuming task. The main problem with these approaches is that they do not generalize well — they do not perform well on unseen words. A structural decomposition of W into patterns is the key idea here, and brings better generalization qualities:

Definition 2.4 (word division problem). Let W be a set of words over $\Sigma \cup \{0,1\}$ such that placing 1 between two letters in w denotes the possibility of word division at that point (placing 0 or nothing means the opposite). We want to find pattern set P such that winning patterns $\mathit{classify}(w,P)$ encode the same information as w. In this case we say that P or $\mathit{classify}(w,P)$ *covers* w.

An example of competing patterns for the hyphenation problem is shown in Figure 10.1 on page 86.

We want to find a pattern set that is minimal in size and maximal in performance; we have to define these performance measures.

Definition 2.5 (precision, recall, F-score). Let $W = (\Sigma \cup \{0,1\})^*$, and P a set of patterns over $\Sigma' \cup \mathbb{N}$. Let $\mathit{good}(w,P)$ is the number of word divisions where $\mathit{classify}(w,P)$ covers w, $\mathit{good}(W,P) = \sum_{w \in W} \mathit{good}(w,P)$. $\mathit{bad}(w,P)$ is the number of word divisions where $\mathit{classify}(w,P)$ classifies word division that is not in w, $\mathit{bad}(W,P) = \sum_{w \in W} \mathit{bad}(w,P)$. $\mathit{missed}(w,P)$ is the number of word divisions where $\mathit{classify}(w,P)$ fails to classify word division that is in w, $\mathit{missed}(W,P) = \sum_{w \in W} \mathit{missed}(w,P)$. The definition of the measures is then as follows:

$$\mathit{precision}(W,P) = \frac{\mathit{good}(W,P)}{\mathit{good}(W,P) + \mathit{bad}(W,P)} \tag{2.1}$$

$$\mathit{recall}(W,P) = \frac{\mathit{good}(W,P)}{\mathit{good}(W,P) + \mathit{missed}(W,P)} \tag{2.2}$$

The precision and recall scores can be combined into a single measure, known as the *F-score* (Manning and Schütze, 1999):

Definition 2.6 (F-score).

$$F(W,P) = \frac{2 \times \mathit{precision}(W,P) \times \mathit{recall}(W,P)}{\mathit{precision}(W,P) + \mathit{recall}(W,P)} \tag{2.3}$$

An F-score reaches its maximum when both precision and recall is maximal; in the case $F(W,P) = 1$ all information about word division is compressed into the pattern base P.

Definition 2.7 (lossless compression, cross validation). If $F(W, P) = 1$ we say that we *losslessly compressed* W into P. We can test performance of P on an unseen word list W' to measure the generalization properties of pattern set P — in the machine learning community, the term *cross validation* is used.

Here we see one advantage of the pattern approach. In the case where we have solved the hyphenation problem by storing all the words with the division point in a hash table or using a finite state transducer, we do not know how to segment new, unseen words. On the other hand, pattern P trained on W can perform well on unseen words (typically new long words or compounds) — as in patterns the *rules* are generalized.

There are many pattern sets P that losslessly compress (cover) W; one straightforward solution is having just one pattern for every word $w \in W$ by putting dot symbol around the word with division points marked by 1. Such a pattern set P is a *feasible solution*. But we want to obtain minimal pattern set. Minimality can be measured by the number of patterns, by the number of characters in patterns, or by the space the patterns occupy when stored in some data structure. Even if we take the simplest measure by counting the patterns, and try to find a minimal set of patterns that cover W, we will show how hard the task is. To formulate it more precisely, we need to define:

Definition 2.8 (minimum set cover problem). An instance of set cover problem is finite set X and a family $\mathcal{F}$ of subsets of X, such that $X = \bigcup_{S \in \mathcal{F}} S$. The problem is to find a set $C \subseteq \mathcal{F}$ of *minimal size* which covers X, i.e. $X = \bigcup_{S \in C} S$.

The minimum set cover problem (MSCP) is known to be in the class of NPO problems (optimization problems analogical to NP decision problems), (Ausiello et al., 1999). A variant of MSCP, in which the subsets have positive weights and the objective is to minimize the sum of the weights in a set cover, is also NPO. Weighted version of minimum set cover problem is approximable within $1 + ln|X|$ as shown by Chvátal (1979).

Theorem 2.1 (pattern minimization problems). *Let W be a set of words with one division only. Problem of finding minimal number of patterns P that losslessly compress W is equivalent to the (weighted) minimum set cover problem.*

Proof. For every subset $C \in W$ there exists at least one feasible solution P_C such that P_C covers C and does not cover any word in $W \setminus C$, e.g., pattern set $\{.c. \mid c \in C\}$. Between all such feasible solutions we choose a canonical representative P'_C — a set which is smallest by some measure (e.g., number of patterns, or number of characters in the pattern set). We now have a one to one correspondence between all pattern sets that cover exactly C represented by P'_C and C. Thus we showed that a pattern coverage minimization problem

is equivalent to the weighted minimum set cover (Chvátal, 1979) in NPO class. □

We have shown that even a pattern covering problem without competition is already NPO. When trying to cover W by competing patterns, complicated interactions may arise — we need some approximation of the optimal solution.

Liang's main concern in the pattern covering problem was the size of the patterns stored in a packed trie (indexed trie with packing the different families of the trie into a single large array — see Table 11.1 on page 100) in computer memory. He discusses NP-completeness of finding a minimum size trie (Liang, 1983, page 25) by pointing to the problem transformation from graph coloring by Pfleger (1973).

Competing patterns extend the power of finite state transducer somewhat like adding the "not" operator to regular expressions.

Methods for the induction of covering patterns from W are needed.

Attempts to catch the regularities in empirical data (W in our case) can be traced back to the 1960s, when Chytil and Hájek started to generate unary hypotheses on finite models using the GUHA method (Hájek and Havránek, 1978).

Definition 2.9 (matrix representation of the data). Let us have $m \times n$ matrix $W = w_{ij}$ of data that describe m objects with n binary attributes $P_1, P_2, \ldots, P_n$ (unary predicates). Either P_j or $\neg P_j$ holds. *Elementary conjunction* is a conjuction of literals P_j, $1 \leq j \leq n$, where every predicate appears once at most. Similarly, *Elementary disjunction* is a disjunction of literals P_j with the same condition. We say that the object i fulfills elementary conjunction Φ if the formula exactly describes the attributes in line i of W. We say that Φ holds for W if Φ holds for all objects (lines in W). We say that formula Φ is *p-truth* if Φ holds for at least $100p\%$ of objects, $p \in \mathbb{R}, 0 < p \leq 1$.

We immediately see that we can represent our hyphenation problem by a matrix W: the attribute in column j, P_j tells whether a word can be divided (true or 1) or not (false or 0).

GUHA method searches for such elementary conjunctions A (*antecedents*) and elementary disjunctions S (*succedents*) with no common predicates, such that implication $A \rightarrow S$ is p-truth; it searches for hypotheses with highest p to detect dependencies in data. Observational language in this case is *propositional logic*. There are many general approaches using first-order predicate calculus or even higher formalisms (Lloyd, 2003), but these are not necessary for our task.

Definition 2.10 (p-truth pattern α). Let us have m hyphenated words represented in matrix W as in Definition 2.9. We say that pattern α is *p-truth pattern* if it covers at least $100p\%$ of applicable word segmentation points.

The greedy approach for pattern search consists in collecting p-truth patterns with the highest p of the shortest length. Short patterns give a high generalization and good minimalization of space for pattern storage. But during its generation some heuristics have to be used, as maximal coverage of covering patterns does not imply good performace in the succeeding phases of pattern generation (of inhibiting patterns). Further discussion on pattern preparation follows in Section 3.3 on page 14.

References

Giorgio Ausiello, Giorgio Gambosi, Pierluigi Crescenzi, and Viggo Kann. 1999. *Complexity and Approximation*. Springer-Verlag.

Claudio Carpineto and Giovanni Romano. 2004. *Concept Data Analysis: Theory and Applications*. Wiley, July.

Václav Chvátal. 1979. A Greedy Heuristic for the Set Covering Problem. *Mathematics of Operations Research*, 4:233–235.

Ulf Grenander. 1993. *General Pattern Theory*. Clarendorn Press, Oxford.

Maurice Gross. 1997. The Construction of Local Grammars. pages 329–354.

Petr Hájek and Tomáš Havránek. 1978. *Mechanising hypothesis formation — Mathematical foundations for a general theory*. Springer-Verlag.

Jerry R. Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. 1997. FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. pages 383–406.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Department of Computer Science, Stanford University, August.

John W. Lloyd. 2003. Learning Comprehensible Theories from Structured Data. In S. Mendelson and A.J. Smola, editors, *Advanced Lectures on Machine Learning, LNAI 2600*, pages 203–225.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Chapter 3

Methodology of Competing Patterns

The idea of competing patterns is taken from the method developed by Liang (1983) for his English hyphenation algorithm. It has been shown by extensive studies (Sojka and Ševeček, 1995; Sojka, 1995; Sojka, 1999) that the method scales well and that parameters of the pattern generator — PATGEN program (Liang and Breitenlohner, 1999) — could be fine-tuned so that virtually all hyphenation points are covered, leading to about 99.9% efficiency.

The methodology consists of several parts:

stratification – for repetitive pattern generation, it is practical to have a stratified word list with 'information bearing' samples only;

bootstrapping – input data (word list with marked hyphenation points) preparation;

goal-driven threshold setting heuristics – the quality of generated patterns depends on many parameters that have to be set in advance;

data filtering by threshold setting heuristics – we can filter out 'dangerous' data — data that are hard to learn for manual inspection.

3.1 Stratification

Word lists from which patterns are generated may be rather big. A full list of Czech word forms has about 6,000,000 entries when generated by the Czech morphological analyzer `ajka`. It may be even more than that for other tasks with huge input data collections such as POS tagging, or Thai text segmentation (see Chapter 12 on page 105). Context necessary for ambiguity resolution is often repeated several times — a word list may be stratified. Stratification means that from 'equivalent' words only one or small number of representatives are chosen for the pattern generation process.

With the stratification procedure described in Section 6.3.2 we have downsampled 3,300,000 Czech word forms to a word list of 372,562 word

forms (samples) for PATGEN input. The same approach was used also for Slovak with the results are in Table 6.4 on page 37.

Stratified sampling is less important when we insist on lossless compression, or when we have enough computing power for pattern generation.

3.2 Bootstrapping

The preparation of data for machine learning is often a time-consuming task and for extremely large data sets, a technique called *bootstrapping* is used. It was used for tagging the ORCHID corpus (Section 12.3 on page 110), and for tagging word divisions it is also usefull. The idea is to tag only small initial data set (word list), and generate patterns from this input. Then, these bootstrap patterns are used for the automatic tagging of a bigger input list, and checked before the next pattern generation phase.

Bootstrapping may bring errors especially with overlapping prefixes (ne-, nej-, po-, pod-). It is worth the trouble marking these points separately, e.g., with the help of a morphological analyzer. For detailed description of this technique, refer to Section 9.4 on page 73.

3.3 Pattern Generation

Pattern generation processes are driven by several threshold parameters whose settings are essential for the quality and properties (precision and recall) of generated patterns. Our experience shows that parameter setting not only depends on the requested pattern behaviour but to a certain extent on the problem at hand. Parameter setting has to be tuned for every pattern generation project.

PATGEN runs at various levels. At every level, a new set of patterns is generated. It starts with short patterns (counting frequencies of substrings of a given length), generating longer ones in the next level as 'exceptions', and making 'exceptions of exceptions' in the next level, etc. With this model, we can learn exact dependencies between contexts of hyphenation points in words that are used in a much wider context than can standard (bi|tri)gram or other statistical methods taken into consideration — there are examples when the segmentation decision depends on the word segment that is six characters away.

There is no known algorithm that helps with setting of the parameters of the learning process. Liang's original patterns (`hyphen.tex`) that are in every TEX distribution as a default patterns for (American) English are very

inefficient and have very low recall. They cover only 89.3% (Liang, 1983, page 37) — of very small word list (Webster's Pocket Dictionary) of 49,858 words. The threshold used in pattern generation were not tuned at all, and better choices can lead to smaller pattern size and higher (actually complete) coverage of hyphenation points in an input word list.

3.4 Pattern Parameter Setting Optimization

Our extensive experience shows that parameter setting is highly language dependent — it differs when generating patterns for Thai segmentation (Chapter 12 on page 105) for Czech and Slovak hyphenations (Chapter 6 on page 27 or German compounds (Chapter 7 on page 46). Scannell (2003) reports that using this methodology he generated a new pattern for Irish that does not produce any hyphen points which are not in the database and miss just 10 out of 314,639 hyphen points. This is consistent with our findings that the methodology is usable as very effective lossless compression algorithm, and there is the power of competing patterns to cover *all* exceptions from data.

We may experiment with parameter setting so that generated patterns are *nearly* lossless. Words that were not covered in this phase are in some way different than the rest. This difference may well be right, but usually show an input data tagging error. We suggest manually checking this small set of words esspecially when developing and marking new word lists from scratch.

3.5 Layering of Disambiguation Patterns

There can be a different version of the input data (different variants of segmentation, tagging), with different patterns. As competing patterns are decomposable into layers, we can "plug-in" patterns developed by experts on the problem and merge or compare them with those generated. We can let the patterns "compete" — or adjust them so that, for example, expert knowledge takes preference over generated patterns, or we can take the expert patterns as initial set of patterns and generate the patterns to cover the rest of the input data. Tables 6.1 and 6.2 show that hyphenation patterns were often done by hand, or by a combination of hand crafted and generated patterns. Having several layers of expert patterns, we can easily set up their priorities by changing the classification numbers in the patterns. This priority handling is necessary in most information fusion tasks.

The potential of the methodology was mostly tested on the segmentation problems described in the following chapter.

References

Franklin M. Liang and Peter Breitenlohner. 1999. PATtern GENeration program for the TEX82 hyphenator. Electronic documentation of PATGEN program version 2.3 from web2c distribution on CTAN.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Department of Computer Science, Stanford University, August.

Petr Sojka and Pavel Ševeček. 1995. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Petr Sojka. 1999. Hyphenation on Demand. *TUGboat*, 20(3):241–247.

Chapter 4

Applications of Competing Patterns

The methodology of competing patterns was used on a wide spectrum of tasks, and for several tasks we have designed or suggested its application. In Section 4.1 we list applications in computer typeseting, and in Section 4.2 on the following page tasks solvable by our methodology in the area of language engineering are discussed.

4.1 Competing Patterns in Computer Typesetting

Pattern technique can be used in every task, where some *context-dependent* action should be taken during typesetting. There are many examples of application in the computer typesetting area:

Hyphenation for Czech and Slovak. New hyphenation patterns and process of their development is described in Chapter 6 on page 27 and (Sojka, 2004).

Hyphenation of compound words. The plausibility of the approach was shown for German in (Sojka, 1995).

Context-dependent ligatures and discretionary insertions. A typesetting engine should not use ligatures in some cases, e.g., on compound word boundaries, and insert or change characters during the hyphenation of some characters in German. Chapter 7 on page 46 discusses these applications in detail.

Fraktur long s versus short s. In the Gothic letter-type there are two types of s-es, a long one and the normal one. The actual usage depends on the word's morphology. This is another typical context-dependent auto-tagging procedure implementable by contextual patterns.

End of sentence recognition. To typeset a different width space at the end of a sentence automatically, one has to filter out abbreviations that do not normally appear at the end of a sentence. A hard, but managable task for competing patterns.

Context dependent spell checking. Storing a big word list in a packed digital tree is feasible and gives results comparable to spelling checkers like ispell. We think that even context-dependent spell checking can be taught by patterns.

Thai segmentation. There is no explicit word/sentence boundary, punctuation and inflexion in Thai text. This information, implicitly tagged by spaces and punctuation marks in most languages, is missing in standard Thai text transliteration. It is, however, needed, during typesetting for line-breaking. It was shown in Chapter 12 on page 105 that the pattern-based technology outperforms the currently used probabilistic trigram model (Sornlertlamvanich et al., 1999).

Arabic letter hamza. Typesetting systems for Arabic scripts need to have built-in logic for choosing one of five possible appearances of the letter hamza, depending on the context. This process can easily be learned as pattern-driven task.

Greek or Czech diacritics. In (Haralambous and Plaice, 2001, page 153), there is an algorithm — full of exceptions and context dependent actions — for the process of adding proper accents in Greek texts. Most parts of it can easily be described as a sequence of pattern-triggered actions and thus be implemented by the pattern technique.

Similarly, there are many Czech texts written without diacritics from the times when email mailers only supported seven-bit ASCII, which wait to be converted into a proper form. Even for this task patterns could be trained.

4.2 Competing Patterns in Natural Language Processing

One's ability to produce and recognize grammatical utterances is not based on the notions of statistical approximation and the like.
— Chomsky (1957)

Corpora based natural language processing has become a mainstream research topic (Halteren, 1999; Armstrong et al., 1999) in the area of *language engineering* in the last decade. The power of today's computer chips allows the storage and searching of the whole of Shakespeare's works within a microsecond. The crucial problem is to identify and capture the right language patterns for particular tasks for various language models. The ideal solution is to learn language models from language corpora (Čermák, 1998; Francis and Kučera, 1982; Pala et al., 1997).

In spite of the wide availability of more powerful (context free, mildly context sensitive, or even Turing equivalent) formalisms, the bulk of the

applied work on language and sub-language modeling (*Natural Language Engineering*) is still performed by various *Finite-State Methods* (based on *Finite-State Automata* or *Finite-State Transducers*) (Karttunen et al., 1996; Mohri, 1997; Roche and Schabes, 1995; Mohri, 1996; Kornai, 1999).

While it is certainly true that the mathematical theory of (weighted) regular sets and relations is mature, the same cannot be said of the algorithmic aspects of the subject. As the size of machines grows, we can discern two complementary trends: on the one hand, the search for more efficient algorithms continues, and on the other, techniques leveraging the already remarkable efficiency and scalability of finite state techniques begin to appear (Kornai, 1999). Building such finite state language models from scratch is a very time-consuming task.

4.2.1 Pattern Mining

An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil's behavior.
— Turing (1950)

Statistical techniques (*hidden Markov models*, *N-gram models*) for language learning (for an overview see (Manning and Schütze, 1999)) are frequently used, especially in speech analysis. These models attempt to capture constraints of language by conditioning the probability of a word on a small fixed number of predecessors. The inadequacy of Markov models has been stated by Chomsky (1956). Nevertheless, there are several attempts to use the Viterbi algorithm (Viterbi, 1967) for POS tagging tasks. These methods require large corpora (sparseness problem) to achieve adequate results.

In his thesis, Brill (1993) developed a rule-based approach for learning local dependencies in language data (part of speech tagging). His tagger has three parts, each of which is inferred from a training corpus: a lexical tagger, unknown word tagger and contextual tagger, which is the core of the tagger. Contextual rule *templates* are shown in Table 4.1 on the following page.

Templates are instantiated based on tag statistics (280 contextual rules were obtained). Problems with guessing the right templates for particular problem are partially overcome by unsupervised learning of the rule-based part (Brill and Pop, 1999). Word precision of POS tagging in English is reported up to 96%, but it is a believed that with more linguistic sophistication, the results may even be improved (Brill et al., 1998).

Table 4.1: Rule templates in Brill's POS tagger.

A B PREVTAG C	A to B if previous tag is C
A B PREV1OR2OR3TAG C	A to B if previous 1/2/3 tag is C
A B PREV1OR2TAG C	A to B if previous one or two tags is C
A B NEXTTAG C	A to B if next tag is C
A B SURROUNDTAG C D	A to B if surrounding tags are C and D
A B NEXTBIGRAM C D	A to B if next bigram tag is C D
A B PREVBIGRAM C D	A to B if previous bigram tag is C D

4.2.2 Hyphenation versus Morphological Segmentation

Hyphenation rules are in fact set by conventions. German publishers usually respect DUDEN (1991), British use conventions published by Allen (1990), Americans has their Webster's dictionary (Gove and Webster, 2002).

Conventions differ even for the same language, as an example from English shows. British use etymological roots of a word when choosing hyphenation points while Americans follow pragmatically syllabic segmentation. British even differentiate between several levels of desirability of hyphenation patterns.

Morphological, or rather morphosyntactic segmentation is a task of cutting a word into several segments (prefix, stem, intersegment, ending), according to the word morphology. The relation between hyphenation and morphological segmentation is not obvious. Karlsson (1993, page 95) reports that in Finnish there is high congruence between morphological and syllable boundaries in compounds. Thus a simple CV-rule for hyphenation before consonant-vowel pair has recall of 95.7% and precision of 98.7%.

Morphological analyzer `ajka` (Sedláček, 1999) uses a specific algorithm for morphological segmentation. Nevertheless, at least for stem/intersegment parts, competing patterns can be generated to solve this task. These patterns can be helpful when a new word is about to be segmented, for classifying morphological paradigms. Experiments we did with Czech morphological segmentation are described in Section 10.4 on page 87.

4.2.3 Segmentation in Speech Processing

During two fundamental tasks of speech processing, speech recognition and speech synthesis, segmentation of speech and text for synthesis has to be done. As speech systems and corpora are available, finding a corpus for training competing patterns for the segmentation tasks should not be a problem.

4.2.4 Encoding of Patterns for Part of Speech Tagging

A high quality part-of-speech tagger is the foundation stone of any NLP system. As hand made tagging is erroneous and costly, a lot of effort has been devoted to developing methods of POS tagging — statistical taggers and rule-based taggers being the mainstream methods used so far. Although their efficiency approaches 96–97%, it still leads to about one tagging error per sentence. Statistical taggers fail on exceptions, rule-based taggers are very hard to develop and maintain as natural language evolves in time.

Given a sentence $w_1\ w_2\ \ldots\ w_n$ an ambiguous tagger gives various possible tags: $p_{11}\ \ldots p_{1a_1}$ for the first word, $p_{21}\ \ldots p_{2a_2}$ for the second, etc. Writing output as $(p_{11}\ \ldots p_{1a_1})(p_{11}\ \ldots p_{2a_2})\ldots(p_{n1}\ \ldots p_{na_n})$ the task is to choose the right POS (p_{ij}) for every i. Taking the tag set used in the Brown University Standard Corpus of Present-day American English (Francis and Kučera, 1982) for the sentence "The representative put chairs on the table." we get the output

```
. AT ( NN - JJ ) ( NN VBD - ) (NNS - VBZ) IN AT NN .
```

Hyphenation markers immediately after the right POS tags show solutions for training. Such 'word lists' (for each sentence from a training corpus we get one 'hyphenated' word) could be used by PATGEN for disambiguation patterns generation. Sentence borders are explicitly coded.

4.2.5 Results

We have tried several ways of encoding of morphosyntactic disambiguation problem by patterns (Macháček, 2003). Results show that the method is applicable for the partial disambiguation and comparable with other approaches (inductive logic programming, error driven transformation-based). It is especially useful for deciding "hard" problems, where statistical techniques fail and for disambiguation where wider context has to be taken into account. Patterns were able to disambiguate about 30% of all ambiguities with error rate less than 0.25%. Disambiguation coverage failure often shows errors in manual tagging and after fixing these errors in a training corpus, disambiguation results are better than those obtained by other state-of-the-art methods. Details of experiments done are to be found in (Macháček, 2003).

References

Susan Armstrong, Kenneth Church, Pierre Isabelle, Sandra Manzi, Evelyne Tzoukermann, and David Yarowsky, editors. 1999. *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Publishers Group.

Eric Brill and Mihai Pop. 1999. Unsupervised learning of disambiguation rules for part-of-speech tagging. (Armstrong et al., 1999), pages 27–42.

Eric Brill, Radu Florian, John C. Henderson, and Lidia Mangu. 1998. Beyond N-Gram: Can Linguistic Sophistication Improve Language Modeling? In *Proceedings of the ACL '98*.

František Čermák. 1998. Czech National Corpus: Its Character, Goal and Background. In Petr Sojka, Václav Matoušek, Karel Pala, and Ivan Kopeček, editors, *Text, Speech and Dialogue*, pages 9–14, Brno, Czech Republic, September. Masaryk University Press.

Nelson W. Francis and Henry Kučera. 1982. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin.

Philip Babcock Gove and Merriam Webster. 2002. *Webster's Third New International Dictionary of the English language Unabridged*. Merriam-Webster Inc., Springfield, Massachusetts, U.S.A, January.

Hans van Halteren, editor. 1999. *Syntactic Wordclass Tagging*. Kluwer Academic Publishers Group.

Yannis Haralambous and John Plaice. 2001. Traitement automatique des langues et composition sous Omega. *Cahiers GUTenberg*, (39–40):139–166, May.

Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. 1996. Regular Expressions for Language Engineering. *Natural Language Engineering*, 2(4):305–328.

András Kornai. 1999. *Extended Finite State Models of Language*. Cambridge University Press.

David Macháček. 2003. Přebíjející vzory ve zpracování přirozeného jazyka (Competing Patterns in Natural Language Processing). Master's thesis, Masaryk University in Brno, Faculty of Informatics, Brno, Czech Republic.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Mehryar Mohri. 1996. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2:61–80. Originally appeared in 1994 as Technical Report, Institut Gaspard Monge, Paris.

Mehryar Mohri. 1997. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, June.

Karel Pala, Pavel Rychlý, and Pavel Smrž. 1997. DESAM — Annotated Corpus for Czech. pages 523–530, Milovy, November. Springer-Verlag.

Emmanuel Roche and Yves Schabes. 1995. Deterministic Part-of-Speech Tagging. *Computational Linguistics*, 21(2):227–253.

Radek Sedláček. 1999. Morphological Analyzer of Czech (in Czech). Master's thesis, Masaryk University in Brno, April.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Petr Sojka. 2004. Slovenské vzory dělení: čas pro změnu? (Slovak Hyphenation Patterns: A Time for Change?). *CSTUG Bulletin*, 14(3–4):183–189.

Virach Sornlertlamvanich, Thatsanee Charoenporn, and Hitoshi Isahara. 1999. Building a Thai Part-Of-Speech Tagged Corpus. *The Journal of the Acoustical Society of Japan (E)*, 20(3):140–189, May.

Andrew J. Viterbi. 1967. Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267, April.

Chapter 5

Conclusions

In Section 5.1 we give an overview of the main contributions of our work. We conclude with an outline of possible directions for future work.

5.1 Author's Contributions

A dissertation should make a contribution to world knowledge. It is hoped that this applies to this work as well. Let us sum up the results.

Formal definition of competing patterns. We have described and developed a new approach to language engineering based on the theory of covering and inhibiting patterns.

New approaches to competing pattern generation. We have verified plausibility and usefulness of bootstrapping and stratification techniques for machine learning techniques of pattern generation process. We have related our new techniques to those used so far — with the new approach, the results improve significantly.

Properties of pattern generation process. We have shown that reaching size-optimality of pattern generation process is an NPO problem; however, it is possible to achieve full data recall and precision on the given data with the heuristics presented.

New approach to Thai text segmentation problem. In Chapter 12 on page 105 (Sojka and Antoš, 2003) we have shown that an algorithm using competing patterns learnt from segmented Thai text returns better results than current methods for this task.

Thai segmentation patterns. New patterns for Thai segmentation problem were generated from data in the ORCHID corpus.

New Czech and Slovak hyphenation patterns. The new hyphenation patterns for Czech and Slovak give much better performace than the previous ones, and are in practical use in distributions of text processing

systems ranging from TEX, SCRIBUS, OPENOFFICE.ORG to Microsoft Word.

New patterns for specific tasks. Patterns for specific tasks demanded in the areas of computer typesetting and NLP were developed — phonetic hyphenation, universal syllabic hyphenation, and the possibility of using context-sensitive patterns for disambiguation tasks were shown.

Foundation for new pattern generation algorithms. Redesign of a program for pattern generation in of OPATGEN in an object oriented manner allows easy experiment with new pattern generation heuristics.

Usage of the methodology for partial morphological disambiguation. We have shown that the methodology can be used for partial disambiguation tasks. Experiments showed performance for the partial morphological disambiguation of Czech.

5.2 Mission Statement and Future Work

This work investigated ways how to losslessly compress empirical data (represented as huge binary table for example) into order of magnitude smaller structured patterns in such a way, that information can be located in linear time with respect to the pattern length (irrespectively of the number of patterns). The author hopes that applications of this methodology and techniques developed will be used not only for better hyphenation and segmentation, but for various problems and tasks in the area of natural language engineering, namely speech processing, phonology, word sense disambiguation, etc. The first indications of this hope being realized are the new Irish hyphenation patterns (Scannell, 2003), as well as expression of interest on the usage of the methodology for text segmentation in machine translation software (by Telelingua Software) or by native Thai speakers (by NECTEC). Further applications may have big impact — as is shown, default hyphenation patterns for English that are kept for backward compatibility only cover less than 80% of hyphenation patterns. We showed that it is possible to achieve 100%, making the documents shorter and thus saving forests of trees.

References

Kevin Patrick Scannell. 2003. Hyphenation patterns for minority languages. *TUGboat*, 24(2):236–239.

Petr Sojka and David Antoš. 2003. Context Sensitive Pattern Based Segmentation: A Thai Challenge. pages 65–72, Budapest, April.

PART II

PAPERS

Chapter 6

Competing Patterns for Czech and Slovak Hyphenation

In this chapter we describe basic techniques of pattern-driven approach to the hyphenation problem. Results presented here were joint work with Pavel Ševeček; the paper was written by the Petr Sojka based on word lists and experiments realized by Pavel Ševeček. My proportion on the results presented here is agreed to be 65%.

The first version of this paper was published at the EuroTEX conference (Sojka and Ševeček, 1994):

> Petr Sojka and Pavel Ševeček. 1994. Hyphenation in TEX — Quo Vadis? In Włodek Bzyl and Tomek Przechlewski, editors, *Proceedings of the 9th European TEX Conference, Gdańsk, 1994*, pages 59–68, September.

The paper was requested to be reprinted in (Sojka and Ševeček, 1995a) and final version was accepted for the journal publication (Sojka and Ševeček, 1995b):

> Petr Sojka and Pavel Ševeček. 1995a. Hyphenation in TEX — Quo Vadis? In Michel Goossens, editor, *Proceedings of the TEX Users Group 16th Annual Meeting, St. Petersburg, 1995*, pages 280–289, Portland, Oregon, U.S.A. TEX Users Group.
>
> Petr Sojka and Pavel Ševeček. 1995b. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

This chapter contains the final version with minor corrections. The paper has been cited several times, e.g., in (Beccari et al., 1995; Scannell, 2003) and in on-line discussions about solving hyphenation in current XSL formatting objects (XSL-FO) processors.

Petr Sojka, Pavel Ševeček

Hyphenation in TEX — Quo Vadis?

Abstract: A significant *progress* has been made in the hyphenation ability of TEX since its first version in 1978. However, in *practice*, we still face problems in many languages such as Czech, German, Swedish etc. when trying to adopt local typesetting industry standards.
In this paper we discuss problems of hyphenation in multilingual documents in general and explain *principles* used in TEX engine for hyphenation of words. We show how we have made Czech and Slovak hyphenation patterns and we describe our results achieved using the PATGEN program for hyphenation pattern generation. We show that hyphenation of compound words may be partially solved even within the scope of TEX82. We discuss possible enhancements of the process of hyphenation pattern generation and describe features that might be reasonable to think about to be incorporated in OMEGA or another successor to TEX82.

Key Words: hyphenation • PATGEN • compound words • pattern generation • hyphenation exceptions • ε-TEX • $\mathcal{NTS}$ • OMEGA

6.1 Motivation

Go forth and make masterpieces of hyphenation patterns . . .
— Haralambous (1994)

Editors' and publishers' typographical requirements for camera-ready prepared documents are growing. To meet some of their requirements in TEX, especially when typesetting in narrow columns, one needs perfect hyphenation patterns in order to find almost all permissible hyphenation points.

When making Czech hyphenation patterns and typesetting multilingual documents we encountered some problems related to achieving quality hyphenation and decent-looking documents within TEX. This work has led us to ideas about possible remedies and future extensions in a successor to TEX.

This chapter consists of three parts. In the first part we summarize the developments that have been made on the issue since TEX's conception. In the second one, we describe our attempts to create Czech and Slovak

hyphenation patterns and summarize hints and suggestions for PATGEN users. In the third part we discuss possible improvements that might take place in a TEX successor (OMEGA, ε-TEX or New Typesetting System ($\mathcal{NTS}$)).

6.2 Hyphenation Development

Let us review the developments in hyphenation in TEX that have been made so far.

6.2.1 English

In TEX78 a rule-driven algorithm for English was built-in by Liang and Knuth. Their algorithm found 40% of the allowable hyphens, with about 1% error (Liang, 1981). Although authors claimed that these results are "quite good", Liang continued working on the generalization of the idea of rules expressed by hyphenating and inhibiting patterns. In his dissertation (Liang, 1983) he describes a method, which is used in TEX82, based on the generalization of the prefix, suffix and the vowel-consonant-consonant-vowel rules. He wrote (in WEB) the PATGEN program (Liang and Breitenlohner, 1991) to automate the process of pattern generation from a set of already hyphenated words. He started with the 1966 edition of Webster's Pocket Dictionary that included hyphenated words and inflections (about 50,000 entries in total). In the early stages, testing the algorithm on a 115,000 word dictionary from the publisher, 10,000 errors in words not occurring in the pocket dictionary were found. "Most of these were specialized technical terms that we decided not to worry about, but a few hundred were embarrassing enough that we decided to add them to the word list." (Liang, 1983, p. 30). He reports the following figures: 89.3% permissible hyphens found in the *input* word list with 4447 patterns with 14 exceptions.

Liang's method is described by Knuth (1986, Appendix H) and was later adopted in many programs such as TROFF (Emerson and Paulsell, 1987) and LOUT, and in localizations of today's WYSIWYG DTP systems such as SCRIBUS, OPENOFFICE.ORG, QuarkXPress, or programs as etc. Although specialized dictionaries such as (Allen, 1990) by Oxford University Press separate possible word division points into at least two categories (preferred and less recommended), we have not seen any program that incorporates the possibility of taking into account these classes of hyphenation points so far.

6.2.2 Those Other Languages

... patterns are supposed to be prepared
by experts who are paid well for their expertise.
(Knuth, 1986, p. 453, 8th printing)

The first version of TEX82 allowed only one set of patterns to be loaded at a time. Thus it was not possible to typeset multilingual documents with correct hyphenation in all languages and this limitation was quite unsatisfactory. Already in 1985, two attempts to solve the problem were made:

Multilingual TêX: Extensions, most of which afterwards Knuth adopted in TEX 3.x were suggested and implemented by Ferguson (1985). A new primitive `\language`[1] was introduced for switching between several sets of `\patterns` and hyphenation exceptions. A new `\charsubdef` primitive is still used in today's 8-bit implementations of TEX. Full details can be found in (Ferguson, 1988).

ISITEX: Barth and Nirschl (1985) presented an approach to achieving decent hyphenation in German texts under the name SITEX, or in its interactive version under the name ISITEX. Their method, (available as a change file for UNIXTEX from `http://www.apm.tuwien.ac.at/`) has been used in Germany for years and is being improved (Barth and Steiner, 1992; Barth et al., 1993; Steiner, 1995). This approach has not been accepted for inclusion in $\mathcal{NTS}$ (NTS-L, 1992 1995).

SITEX (ISITEX for the interactive version) introduces a new primitive `\nebenpenalty` which allows differentiation between main (compound word boundaries) and secondary (word stem) hyphenation points.

A new notation for hyphenation patterns is introduced and a hyphenation algorithm for German is hardwired into the program. The tables for the algorithm, file `sihyphen.tex` (60 kB) are written manually and can be simply edited and enriched. However, no provision for the generation of these patterns from a word list (such as the PATGEN program) is offered.

During the last 15 years almost every year there appeared a paper in TUGboat reporting new patterns for some language (see Table 6.1 on page 32). Another couple of hyphenation patterns, fonts and preprocessors is available in ScholarTEX[2] (Haralambous, 1991).

Although Donald Knuth introduced the new primitives `\language` and `\setlanguage` for switching between several sets of hyphenation patterns in

1. A rather misleading name, as it deals with only one particular feature of a language — hyphenation — which feature is of only limited interests to linguists.
2. ScholarTEX is a registered trademark of Yannis Haralambous.

TEX 3.0, there are indications that not all of the related problems have been solved and further investigations are necessary (Fanton, 1991).

Proposals on how to customize TEX for a new language were suggested by Partl (1990). New tools to simplify the generation of 8-bit (virtual) fonts were designed — `fontinst` (Jeffrey, 1993) and `accents` (Zlatuška, 1991). A macro package for simple language switching babel (Braams, 1991a; Braams, 1991b; Braams, 1993) was produced to simplify typesetting of multilingual documents. An international version of the `Makeindex` program was written (Schrod, 1991). The DC fonts (Ferguson, 1990; Haralambous, 1992a; Haralambous, 1993a), designed to permit hyphenation in many languages, are now being widely distributed, forced by the new LATEX wave. Compliance with the suggestions of the working group TWGMLC[3] (Haralambous, 1992a) could help too (naming conventions for hyphenation files, etc.). Multilingual document aspects of typesetting are being collected in the scope of LATEX3 project in (Gaulle, 1994), where a nice collection of language-related TEX primitives can be found, together with definitions of the terminology used.

6.2.3 Exception Logs

> *If any computer center decides to preload different exceptions from those in plain TEX (i.e., in the file `HYPHEN.TEX`), the changed exceptions should not under any circumstances be put into `HYPHEN.TEX` or `PLAIN.TEX`. All local changes should go into a separate file, so that TEX will still produce identical results on all machines. In fact, I recommend not preloading those changes, but rather assuming that individual users will have their own favorite collection of updates to the standard format files.*
> — Knuth (1983)

The exception log and corrections for US English hyphenation were reported several times by Thulin (1987), Beeton (1989) and Kuiken (1990) as shown in Table 6.3 on page 34. These listings are published in accordance with Knuth's wish in (Knuth, 1983). Only words with *wrongly* placed hyphenation points are listed, not those where TEX finds only a subset of possible breakpoints.

This shows that significant care and effort is still needed and *is being gradually spent* on the checking of hyphenation points during proof-reading and that the standard US patterns are not sufficient to satisfy current needs. Additional sets of patterns (two versions — `ushyphen.add` and `ushyphen.max`) were generated by Kuiken (1990) to cover the exceptions by additional patterns and these add-on files are available on CTAN[4]. *But*, by adding one of these files to the end of the `\patterns` command in `hyphen.tex`, in order to

3. TEXnical Working Group on Multiple Language Coordination
4. Comprehensive TEX Archive Network

Table 6.1: Hyphenation patterns for TEX with PATGEN statistics for various languages.

language	trie	ops	done by	# patt	size	author (& reference)
BG (Bulgarian)	688	56	hand	263	1,672	Ognyan Tonev/90
CA (Catalan)	661	11	hand	826	6,136	Goncal Badenes, Francina Turon/91
CY (Welsh)	8,552	143	PATGEN	6,728	43,162	Yannis Haralambous, (Haralambous, 1993b)
CZ_1 (Czech)	3,676	90	hand	4,479	25,710	Ladislav Lhotka/91, (Lhotka, 1991)
CZ_2	5,302	67	PATGEN	4,196	23,474	Pavel Ševeček/94, (Sojka and Ševeček, 1994)
DE_{min} (German)	6,099	170	PATGEN	4,066	25,660	Norbert Schwarz/88
DE_{max}	9,980	255	PATGEN	7,007	45,720	Norbert Schwarz/88
DE (v3.1)	8,375	207	PATGEN	5,719	39,251	Norbert Schwarz, Bernd Raichle/94, (Schulze, 1984; Partl, 1988; Breitenlohner, 1988; Obermiller, 1991; Kopka, 1991)
DK (Danish)	1,815	60	PATGEN	1,145	6,411	Frank Jensen/92
EL (Mod. Greek)	1,278	23	hand	1,616	8,786	Yannis Haralambous/92
EO (Esperanto)	4,895	143	PATGEN	4,118	23,224	Derk Ederveen/93
ES (Spanish)	1,106	29	hand	578	4,609	Francesc Carmona/93
ET (Estonian)	2,054	45	PATGEN	1,267	7,976	Enn Saar/92
FI (Finnish)	583	27	hand	232	1,342	Kauko Saarinen/92, (Saarinen, 1988)
FR (French)	1,634	86	comb.	917	30,022	Jacques Désarménien, Daniel Flipo, Bernard Gaulle et al./84–94, (Désarménien, 1984)
Ancient Greek			hand			Yannis Haralambous/92, (Haralambous, 1992b)
HR (Croatian)	1,471	46	hand	916	7,250	Cvetana Krstev/93
HY (Armenian)						Yannis Haralambous (in ScholarTEX)
IS (Icelandic)	5,477	145	PATGEN	4,187	29,919	Jorgen Pind/87
IT (Italian)	1,327	15	hand	729	4,255	Salvatore Filippone/92, (Canzii et al., 1984)

Table 6.2: Hyphenation patterns for TEX with PATGEN statistics for various languages (continued).

language	trie	ops	done by	# patt	size	author (& reference)
IT (Italian)	529	37	hand	210	2,571	Claudio Beccari/93, (Beccari, 1992)
Latin			hand			Y. Haralambous/92, (Haralambous, 1992b)
Modern Latin			hand			Claudio Beccari/92, (Beccari, 1992)
LT (Lithuanian)	2,169	77	PATGEN	1,546	9,639	Vitautas Statulevicius & Y. Haralambous/92
NL_1 (Dutch)	7,824	124	PATGEN	6,105	37,997	CELEX/89
NL_2	10,338	187	PATGEN	7,928	50,969	CELEX/89
NL_3	520	24	hand	326	1,732	Peter Vanroose
NO (Norwegian)	3,669	220	PATGEN	2,371	15,589	Ivar Aavatsmark/92
PL (Polish)	4,954	194	hand	4,053	28,907	Hanna Kołodziejska/94, (Kołodziejska, 1987; Kołodziejska, 1988)
PT (Portuguese)	374	10	hand	126	534	Pedro J. de Rezende, (Rezende, 1987)
RU (Russian)	4,599	92	hand	4,121	29,272	Dimitri Vulis, (Vulis, 1989; Malyshev et al., 1991a; Malyshev et al., 1991b; Samarin and Urvantsev, 1991)
SK (Slovak)	3,600	248	hand	2,569	22,628	Jana Chlebíková/92, (Chlebíková, 1991)
SK	7,606	78	PATGEN	6,137	35,623	Pavel Ševeček/94, (Sojka and Ševeček, 1994)
SR (Serbian)	1,475	40	hand	896	6,890	Cvetana Krstev/89, (Krstev, 1991)
SV (Swedish)	5,269	125	PATGEN	3,733	23,821	Jan Michael Rynning/91
TR (Turkish)	678	16	hand	1,834	9,580	Pierre A. MacKay/88, (MacKay, 1988)
UK (UK English)	10,995	224	PATGEN	8,527	54,769	Dominik Wujastyk/93
US (US English)	6,075	181	PATGEN	4,447	27,302	Frank Liang/82, (Liang, 1983)
US	6,661	229	PATGEN	4,810	30,141	Gerald D.C. Kuiken/90, (Kuiken, 1990)

Table 6.3: A growing number of exceptions for `hyphen.tex`.

# of exceptions	where reported
14	(Liang, 1983)
24	(Beeton, 1984, TUGboat 5, no. 1)
88	(Beeton, 1985, TUGboat 6, no. 3)
127	(Beeton, 1986, TUGboat 7, no. 3)
129	(Thulin, 1987, TUGboat 8, no. 1)
501	(Beeton, 1989, TUGboat 10, no. 3)
543	(Beeton, 1992, TUGboat 13, no. 4)

overcome huge exception lists that should be loaded with every document, one loses the compatibility between different installations and acts against Knuth's wishes.

6.2.4 The Need to Regenerate US English Patterns

`! TeX capacity exceeded, sorry [exception dictionary=307.]`
— Donald E. Knuth

So, to follow Knuth's rules, every document should start with loading the exception file — for this, one has to increase the size of exception buffer in TEX82 (in words) from 307 to at least 607 (as it is now usual in UNIXTEX, emTEX and other installations). However, this is barely sufficient for the current English exception file (remember one has to add words in all possible inflexions), for inflexional languages (such as Czech, where from one stem there are about 20 different suffixes) it is unusable.

Maybe it is time to regenerate the patterns from a bigger (say, 200,000 entries) word list once again from the scratch?[5] Imagine the day when you know that TEX will find 99.99% of hyphens contained in your copy of Webster, so you will not have to go through a list of exceptions and a couple of dictionaries to check hyphenation points in your document! For backward compatibility one has to save every document together with the patterns and exceptions used anyway.[6]

5. Otherwise in 2050 there will have to be an extra issue of TUGboat devoted to the publication of exceptions to `hyphen.tex`.

6. A search on `CTAN` via `quote site index` command shows five files of *different* lengths with the name `hyphen.tex`. (And Knuth and Liang's `hyphen.tex` can be found there under four different names — `hyphen.tex`, `ushyph1.tex`, `ushyphen.std`, `ushyphen.tex` — which leads to the total confusion!)

6.3 Making Czech and Slovak Hyphenation Patterns with PATGEN

A program should do one thing, and do it well. — Ken Thompson

The first Czech patterns were made in 1988 by Petr Novák using PATGEN from a list of 170,000 word forms. Because of errors in his word list, and only partially optimized PATGEN parameter settings, the patterns were good but not perfect.

The patterns were not publicly available, so the second attempt was done by hand by Lhotka (1991) just as MacKay (1988) did for Turkish. Because of lots of exceptions to the 'rules', their usage was not quite comfortable either.

As Novák's list of words had lately been made public, we started compiling a bigger word list from various sources using the old patterns for bootstrapping. We have learnt a lot from the experience described by Rynning (1991) and Haralambous (1993b) and in a tutorial (Haralambous, 1994).

6.3.1 Czech Hyphenation Rules

Czech hyphenation rules are described in Martincová et al. (1993, p. 53–54) and in a special book (Haller, 1956) where a list of exceptions was published. Briefly, we have syllabic hyphenation with morphological 'etymological' exceptions. Hyphenation is preferred between a prefix and the stem, and on the boundary of compound words. Things become complicated when:

1. The word evolved in such a way that although historically it was built from a prefix plus the stem of another word, today it is perceived as a new word stem. As an example may serve the word `ro-zu-mět` — "to understand" (syllabic division) against `roz-u-mět` (`roz` is the prefix and `umět` means "to know").
2. There is no agreement on word hyphenation — e.g., the *current* rules for word `sestra` — "sister" allow one to hyphenate `se-stra`, `ses-tra` and `sest-ra`.
3. Word stem hyphenation points change when a suffix is added — e.g., `hrad` — "castle" cannot be hyphenated, but with a suffix it can — `hra-du`.
4. Compound words, e.g. `tři-a-třiceti-letý` — "33 years old", are taken into account. Czech has a lot of compound words, but not to such an extent as German has.
5. The hyphenation of a word depends on the semantics: `nar-val` "narwhal" and `na-rval` "plucked".

These rules make it difficult to create patterns that describe all these exceptions and exceptions to exceptions. As we had at hand a word list with lists

of allowable prefixes and suffixes, together with preliminary patterns to hyphenate word stems for bootstrapping, we decided to generate a hyphenated list of Czech words for PATGEN.

6.3.2 Stratified Sampling

> *A large body of information can be comprehended reasonably well by studying more or less random portions of the data. The technical term for this approach is* stratified sampling.
> — Knuth (1991, p. 3)

Czech is a highly inflexional language; on average 20–30 inflexions can be derived from one word stem by changing the suffix added and one can multiply it almost twice, as negation can be formed from many words (adjectives, verbs) by prefixing `ne`. Thus from a 170,000 stem word list about 3,300,000 word forms may be generated. Generating patterns from such a list would be very impractical. Because the suffixes are often the same or similar, we generated a word list by means of the following rules:

- We add only every 7th (actually 17th worked as well) derived word form from the full list to the PATGEN input list, with the following exceptions:
  1. Every stem must be accompanied by at least one derived form.
  2. Every derived form with overlapping prefixes has to be present in the PATGEN input list as well.
  3. Only one word with prefixes `ne` (by which one can form negation to almost every word) and `nej` (by which one creates superlatives) is included.
  4. The hand-made list of exceptions (about 10,000 words) from Haller (1956) and other sources are always included.

With this procedure we have 372 562 Czech words to work with PATGEN. The same approach was used also for Slovak. The results are in Table 6.4 on the following page.

Samples of PATGEN statistics are presented in Tables 6.5, 6.6 and 6.7. These tables show that by twiddling with PATGEN parameters one may generate various versions of patterns. Usually the size of patterns and percentage of bad hyphenations are the minimization criteria, but maximization of percentage of good (found) hyphenations and other strategies might be chosen. Comparing this approach to the use of feedforward neural networks to learn hyphenation rules (Smrž and Sojka, 1997), discrete patterns win from almost all perspectives.

Table 6.4: PATGEN statistics for Czech and Slovak.

# of words	# of hyphenation points Correct	Wrong	Missed
Czech			
372,562	1,019,686 (98.26%)	39 (0.01%)	18,086 (1.74%)
Slovak			
333,139	1,025,450 (98.53%)	34 (0.01%)	15,273 (1.47%)

6.3.3 Compound Words

Hints for hyphenation are most often needed at the word boundaries of compound words. — Saarinen (1988, p. 191)

As an experiment we took our (rather huge) word list of Czech words in which hyphenation was marked only on prefix and compound word boundaries.

Table 6.5: Standard Czech hyphenation with Liang's parameters for English.

level	length	param	% correct	% wrong	# patterns	size
1	2–3	1 2 20	96.95	14.97	+ 855	
2	3–4	2 1 8	94.33	0.47	+1,706	
3	4–5	1 4 7	98.28	0.56	+1,033	
4	5–6	3 2 1	98.22	0.01	+2,028	32 kB

The PATGEN program was able to produce hyphenation patterns for this list successfully. The number of patterns was rather large, but feasible (25–84 kB, depending on parameters). From a 380,698 item word list the patterns found 307,470 of the hyphenation points correctly, 5,040 points were hyphenated wrongly (exceptions), and 4,680 hyphenation points were

Table 6.6: Standard Czech hyphenation with improved pattern size strategy (cf. Table 6.4).

level	length	param	% correct	% wrong	# patterns	size
1	1–3	1 2 20	97.41	23.23	+ 605	
2	2–4	2 1 8	85.98	0.31	+ 904	
3	3–5	1 4 7	98.40	0.78	+1,267	
4	4–6	3 2 1	98.26	0.01	+1,665	23 kB

Table 6.7: Standard Czech hyphenation with improved recall (percentage of hyphenation points covered) strategy.

level	length	param	% correct	% wrong	# patterns	size
1	1–3	1 5 1	95.43	6.84	+2,261	
2	1–3	1 5 1	95.84	1.17	+1,051	
3	2–5	1 3 1	99.69	1.24	+3,255	
4	2–5	1 3 1	99.63	0.09	+1,672	40 kB

Table 6.8: Czech hyphenation of composed words with Liang's parameters but allowing patterns of length one in level one.

level	length	param	% correct	% wrong	# patterns	size
1	1–3	1 2 20	72.97	14.32	+ 300	
2	2–4	2 1 8	69.32	3.09	+ 450	
3	3–5	1 4 7	84.09	4.02	+ 870	
4	4–6	3 2 1	82.61	0.33	+2,625	25 kB

missing. Some of hyphenation points in the input word list might be wrong, as the database we used is only preliminary. Due to our experience with the standard hyphenation list, after correction of errors (wrongly marked hyphenation points, typos) PATGEN can generalize *substantially* better and the size of the list of patterns is reduced significantly.

To test the possibility of creating patterns for compound words in detail, we generated a word list of more than 100,000 words with 101,687 hyphenation points marked. The list included both compound words and simple ones too.

The results of some of the runs are shown in Tables 6.8, 6.9 and 6.10.

Table 6.9: Czech hyphenation of composed words with slightly modified parameters (percentage of correct slightly optimized).

level	length	param	% correct	% wrong	# patterns	size
1	1–3	1 2 20	72.97	14.32	+ 300	
2	2–4	2 1 8	69.32	3.09	+ 450	
3	3–5	1 4 3	90.82	4.24	+3,014	
4	4–6	3 2 1	89.07	0.36	+2,770	40 kB

Table 6.10: Czech hyphenation of composed words with other parameters of generation (percentage of correct optimized, but percentage of wrong and size increased).

level	length	param	% correct	% wrong	# patterns	size
1	1–3	1 5 1	64.35	5.34	+1,415	
2	2–4	1 5 1	67.10	1.88	+1,261	
3	3–5	1 3 1	97.94	5.39	+8,239	
4	4–6	1 3 1	97.91	1.14	+2,882	84 kB

6.3.4 Generalization

Just for curiosity we tried patterns for different languages on our Czech PATGEN input word list — see Table 6.11 on the next page. There are interesting speculations about these numbers — e.g., trying Slovak patterns on the Czech word list, one finds more than 90% of hyphenation points. On the contrary, probably because of non-syllabic principles and different rules for pronunciation, UK English rules are totally different — only 19% of Czech words are hyphenated correctly by UK patterns. Surprisingly, Swedish, Finnish and Dutch (NE_3) patterns make fewer wrong hyphenations than the Czech old hyphenation patterns. The difference between Dutch patterns made by hand (NE_3) based on the syllabic principle) and those made by PATGEN (NE_1, NE_2) may be caused by the fact that general syllabic hyphenation is relatively good for languages in which the hyphenation is based on syllabic principles. Having hyphenated word lists of different languages, it might be interesting to measure the 'syllabic principles of hyphenation' of different languages on a universal syllabic hyphenation.

As hyphenation in most languages is based on syllabic principles, it is worth trying to create universal syllabic hyphenation and only learn the difference (exceptions) from this universal hyphenation. Let us try to summarize what we think that should be done in the future.

Table 6.11: PATGEN-like statistics for using various language patterns on a Czech hyphenated word list.

Language	Correct	Wrong	Missed
CZ (Sev)	98.26%	0.01%	1.74%
NE_3	57.38%	4.11%	42.62%
SV	57.10%	5.32%	42.90%
FI	52.67%	5.40%	47.32%
CZ (Lho)	93.39%	5.89%	6.61%
SK	90.77%	7.28%	9.23%
US	31.84%	9.58%	68.16%
IT	49.27%	9.88%	50.73%
NO	51.61%	11.32%	48.39%
FR	59.07%	11.54%	40.93%
NE_1	59.14%	11.59%	41.86%
NE_2	58.80%	11.99%	41.20%
UK	18.84%	12.19%	81.16%
DE_{min}	58.62%	12.50%	41.38%
DE_{max}	58.56%	12.70%	41.44%
PL*	69.00%	12.96%	31.00%
PL	68.06%	13.12%	31.94%
DE (v.3.1)	58.84%	13.86%	41.16%

* with transformed patterns — accented letters substituted by non-accented ones

6.4 Future Work

I hope TEX82 will remain stable at least until I finish Volume 7 of The Art of Computer Programming. — Knuth (1989a, p. 625)

6.4.1 Compound Word Hyphenation Support in a Successor to TEX

Good typography therefore is a silent art; not its presence but rather its absence is noticeable. — Mittelbach and Rowley (1992b)

It seems feasible to incorporate separate compound word hyphenation patterns in ε-TEX.

These experiments, discussed in Section 6.3.3 on page 37 show that, even with the current TEX, only doubling the patterns for a language with compounds might allow, e.g., switching between standard hyphenation in narrow columns and compound-word only hyphenation in wide columns.

With a simple change in the program, one may achieve additional flexibility in hyphenation:

New registers `\leftcompoundhyphenmin` and `\rightcompoundhyphenmin` may be helpful for filtering unneeded hyphenation near compound word borders and `\compoundwordhyphenpenalty` might set a penalty (usually much lower than `\hyphenpenalty`) for breaks on compound word boundaries. In this case `\compoundwordchar` character (i.e., the compound work mark in the DC fonts) could be *automatically* inserted there to prevent ligatures going over a compound word boundary.

Another minor addition might be added too, e.g., ε-TEX: in the old version of MLTEX a flag `\dischyph` was implemented, indicating whether to hyphenate words with discretionaries (i.e. embedded hyphens) or not.

6.4.2 Pattern Generalization

Apart from PATGEN extensions according to character clustering, which are orthogonal, we are thinking of the following generalization. Currently, there are only two classes of inter-letter state: an odd or even number that carries information whether to hyphenate or not. The natural generalization would be having n classes. Interletter numbers in patterns would code these classes in such a way that number m between letters will mean that this position belongs to the class number $k \equiv m \pmod{n}$ — when numbering classes from zero. The case $n = 2$ is the current situation, so `\pattern[2]` might mean classical Liang's patterns. Another class might be a prefix boundary, a compound word boundary or whatever else might possibly be useful for the hyphenation algorithm to be aware of the word (discretionary being another possibility).

An application for English is straightforward too. Our approach will allow one to distinguish "preferred" and "less recommended" classes of hyphenation points as published in (Allen, 1990).

In German, one may make other classes (and patterns), e.g., classes for different discretionary breaks.

6.4.3 Suggestions for ε-TEX

> *Please correct if you have* a hyphenated word *at the bottom of a right-hand page.*
> — AMS (1993)

A possible direction was shown by Plaice (1993) and in (Haralambous and Plaice, 1994; Plaice, 1994). With suggested clustering of letters and enriched PATGEN (Liang and Breitenlohner, 1991) one could achieve context-dependent discretionaries and thus solve the `c-k` → `k-k`-like problems in German.

Taylor (1992, p. 249) mentions a possible definition of

`\brokenpenalty` = `\ifrecto` 500`\else` 200`\fi`.

If the output routine could communicate with the parameter-breaking algorithm, word breaks crossing page boundaries could be eliminated.

6.5 Conclusions

Therefore it still is not the right moment to manufacture TEX on a chip.
— Knuth (1989a, p. 641)

In this chapter we presented an overview on the topic of hyphenation in TEX and our results based on experience with Czech and Slovak. We conclude that the current possibilities of TEX are far from perfect and might be improved either in the scope of TEX82 (creation of better hyphenation patterns for various languages by PATGEN), ε-TEX (e.g., duplication of hyphenation mechanism for compound words), OMEGA or $\mathcal{NTS}$ (special capabilities for context-dependent discretionaries).

References

R. E. Allen. 1990. *The Oxford Spelling Dictionary*, volume II of *The Oxford Library of English Usage*. Oxford University Press.

Wilhelm Barth and Helmut Steiner. 1992. Deutsche Silbentrennung für TEX 3.1 (German hyphenation for TEX 3.1). *Die TEXnische Komödie*, (Heft 1). Journal of DANTE (Deutschsprachige Anwendervereinigung TEX e.V.); Group of German-speaking TEX Users.

Wilhelm Barth, Helmut Steiner, and H. Herbeck. 1993. ISITEX Interaktive Silbentrennung für die deutsche Sprache unter TEX 3.14 und 3.141 unter UNIX (Interactive hyphenation for German and TEX 3.14 and 3.141 under UNIX). Electronic documentation of ISITEX from `http://www.apm.tuwien.ac.at/`, August.

Claudio Beccari, Radu Oprea, and Elena Tulei. 1995. How to make a foreign language pattern file: Romanian. *TUGboat*, 16(1):30–41.

Claudio Beccari. 1992. Computer Aided Hyphenation for Italian and Modern Latin. *TUGboat*, 13(1):23–33, April.

Barbara Beeton. 1984. Hyphenation exception log. *TUGboat*, 5(1):15, May.

Barbara Beeton. 1985. Hyphenation exception log. *TUGboat*, 6(3):121, November.

Barbara Beeton. 1986. Hyphenation exception log. *TUGboat*, 7(3):146–147, October.

Barbara Beeton. 1989. Hyphenation exception log. *TUGboat*, 10(3):336–341, November.

Barbara Beeton. 1992. Hyphenation exception log. *TUGboat*, 13(4):452–457, December.

Johannes Braams. 1991a. Babel, a multilingual style-option system for use with LATEX's standard document styles. *TUGboat*, 12(2):291–301, June.

Johannes Braams. 1991b. Babel, a multilingual style-option system. *Cahiers GUTenberg*, 10–11:71–72, September.

Johannes Braams. 1993. An update on the babel system. *TUGboat*, 14(1):60–62, April.

Peter Breitenlohner. 1988. German TEX, a next step. *TUGboat*, 9(2):183–185, August.

G. Canzii, F. Genolini, and Dario Lucarella. 1984. Hyphenation of Italian words. *TUGboat*, 5(1):14, May.

Janka Chlebíková. 1991. Ako rozdeliť (slovo) Československo (How to Hyphenate (word) Czechoslovakia). *ČSTUG Bulletin*, 1(4):10–13, April.

Jacques Désarménien. 1984. How to run TEX in a French environment: Hyphenation, fonts, typography. *TUGboat*, 5(2):91, November.

Sandra L. Emerson and Karen Paulsell. 1987. *troff Typesetting for* UNIX™ *Systems*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

M. Fanton. 1991. TEX: les limites du multilinguisme. *Cahiers GUTenberg*, 10–11:73–80, September.

Michael J. Ferguson. 1988. TEX is Multilingual. In Thiele (Thiele, 1988), pages 179–189.

Michael J. Ferguson. 1990. Fontes latines européennes et TEX 3.0. *Cahiers GUTenberg*, 7:29–32, November.

Bernard Gaulle. 1994. Requirements in multilingual environments. in electronic form (version 1.02) on CTAN as file `vt15d02.tex`, March.

Michel Goossens, editor. 1994. *Proceedings of the TEX Users Group 15th Annual Meeting, Santa Barbara, 1994*, Portland, Oregon, U.S.A. TEX Users Group.

Jiří Haller. 1956. *Jak se dělí slova (How the Words Get Hyphenated)*. Státní pedagogické nakladatelství Praha.

Yannis Haralambous and John Plaice. 1994. First applications of Ω: Greek, Arabic, Khmer, Poetica, ISO-10646/Unicode, etc. In Goossens (Goossens, 1994), pages 256–264.

Yannis Haralambous. 1991. ScholarTEX. *Cahiers GUTenberg*, 10–11:69–70, septembre.

Yannis Haralambous. 1992a. TEX Conventions Concerning Languages. *TEX and TUG News*, 1(4):3–10.

Yannis Haralambous. 1992b. Hyphenation patterns for ancient Greek and Latin. *TUGboat*, 13(4):457–469, December.

Yannis Haralambous. 1993a. DC fonts — questions and answers. *TEX and TUG News*, 2(1):10–12.

Yannis Haralambous. 1993b. Using `PATGEN` to Create Welsh Patterns. Submitted to TUGboat, July.

Yannis Haralambous. 1994. A Small Tutorial on the Multilingual Features of PATGEN2. In electronic form, available from CTAN as `info/patgen2.tutorial`, January.

Alan Jeffrey. 1993. A PostScript font installation package written in TEX. *TUGboat*, 14(3):285–292, October.

Donald E. Knuth. 1983. A note on hyphenation. *TUGboat*, 4(2):64, September.

Donald E. Knuth. 1986. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1988. The Errors of TEX. Technical Report STAN-CS-88-1223, Department of Computer Science, Stanford University, September.

Hanna Kołodziejska. 1987. Dzielenie wyrazów polskich w systemie TEX. Technical Report 165, Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego.

Hanna Kołodziejska. 1988. Le traitement des textes polonais avec le logiciel TEX. *Cahiers GUTenberg*, (0):3–10, April.

Helmut Kopka. 1991. *LATEX—Erweiterungsmöglichkeiten mit einer Einführung in METAFONT*. Addison-Wesley Verlag, Bonn, Germany, second edition.

Cvetana Krstev. 1991. Serbo-Croatian hyphenation: a TEX point of view. *TUGboat*, 12(2):215–223, June.

Gerard D.C. Kuiken. 1990. Additional Hyphenation Patterns. *TUGboat*, 11(1):24–25, April.

Ladislav Lhotka. 1991. České dělení pro TEX (Czech Hyphenation for TEX). *CSTUG Bulletin*, (4):8–9, April.

Franklin M. Liang and Peter Breitenlohner. 1991. `PATtern GENeration` program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.0 from UNIXTEX distribution at `ftp://ftp.cs.umb.edu`, November.

Franklin M. Liang. 1981. TEX and hyphenation. *TUGboat*, 2(2):19–20, July.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Department of Computer Science, Stanford University, August.

Pierre A. MacKay. 1988. Turkish hyphenations for TEX. *TUGboat*, 9(1):12–14, April.

Basil Malyshev, Alexander Samarin, and Dimitri Vulis. 1991a. Russian TEX. *Cahiers GUTenberg*, 10–11:1–6, September.

Basil Malyshev, Alexander Samarin, and Dimitri Vulis. 1991b. Russian TEX. *TUGboat*, 12(2):212–214, June.

Frank Mittelbach and Chris Rowley. 1992. The future of high quality typesetting: structure and design. In Jiří Zlatuška, editor, *Proceedings of the 7th European TEX Conference, Prague, 1992*, page 255, Brno, September. Masarykova Universita.

NTS-L. 1992–1995. New typesetting system discussion list. Archived in CTAN/digests/nts-l/.

Walter Obermiller. 1991. TEX in Germany. *TUGboat*, 12(2):211–212, June.

Hubert Partl. 1988. German TEX. *TUGboat*, 9(1):70–72, April.

John Plaice. 1994. Progress in the Omega Project. In Goossens (Goossens, 1994), pages 190–193.

Pedro de Rezende. 1987. Portuguese hyphenation table for TEX. *TUGboat*, 8(2):102–102, July.

Kauko Saarinen. 1988. Experiences with TEX in Finland. In Thiele (Thiele, 1988), pages 189–194.

Alexander Samarin and A. Urvantsev. 1991. CyrTUG, le monde TEX en cyrillique. *Cahiers GUTenberg*, 12:71–74, December.

Kevin Patrick Scannell. 2003. Hyphenation patterns for minority languages. *TUGboat*, 24(2):236–239.

Joachim Schrod. 1991. An International Version of MakeIndex. *Cahiers GUTenberg*, 10–11:81–90, September.

Bernd Schulze. 1984. German hyphenation and *Umlaut*s in TEX. *TUGboat*, 5(2):103, November.

Pavel Smrž and Petr Sojka. 1997. Word Hy-phen-a-tion by Neural Networks. *Neural Network World*, 7:687–695.

Petr Sojka and Pavel Ševeček. 1994. Hyphenation in TEX — Quo Vadis? In Włodek Bzyl and Tomek Przechlewski, editors, *Proceedings of the 9th European TEX Conference, Gdańsk, 1994*, pages 59–68, September.

Wilhelm Steiner. 1995. *Automatische Silbentrennung durch Wortbildungsanalyse*. Ph.D. thesis, Technisch-Naturwissenschaftliche Fakultät.

Christina Thiele, editor. 1988. *Proceedings of the TEX Users Group 9th Annual Meeting, Montréal, 1988*, Portland, Oregon, U.S.A. TEX Users Group.

Anders Thulin. 1987. More hyphenation exceptions. *TUGboat*, 8(1):76–76, April.

Dimitri Vulis. 1989. Notes on Russian TEX. *TUGboat*, 10(3):332–336, November.

Jiří Zlatuška. 1991. Automatic generation of virtual fonts with accented letters for TEX. *Cahiers GUTenberg*, 10–11:57–68, September.

Chapter 7

Compound Word Hyphenation

The paper presented in this chapter was published in (Sojka, 1995a) and in the TUG '95 conference preprint Proceedings. Final version appeared in the journal (Sojka, 1995b):

> Petr Sojka. 1995a. Notes on Compound Word Hyphenation in TEX. Technical Report FIMU-RS-95-04, Masaryk University in Brno, Faculty of Informatics, August.
>
> Petr Sojka. 1995b. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

This paper was awarded when presented at the TUG 1995 conference in Florida, by nomination of Donald E. Knuth, who attended the conference, and heard my presentation.

This chapter contains the final version with minor corrections. The paper has been cited several times, e.g. in (Kodydek and Schönhacker, 2003; Scannell, 2003) and others.

Compound Word Hyphenation

Petr Sojka

Abstract: The problems of the automatic compound word and discretionary hyphenation in TEX are discussed. These hyphenation points have had to be marked manually in the TEX source file so far. Several methods how to tackle these problems are observed. The results obtained from experiments with German word list are discussed.

7.1 Motivation

. . . problems [with hyphenation] have more or less disappeared, and I have learnt that this is only because, nowadays, every *hyphenation in the newspaper is* manually checked *by human proof-readers.*
— Jarnefors (1995)

In (Sojka and Ševeček, 1994) a case study of problems related to achieving quality hyphenation in TEX was presented with emphasis on the pattern generation for inflexional languages like Czech. It was shown that many issues can be handled within the frame of the good old TEX. Nevertheless, some of them definitely not because TEX was not originally designed as a universal tool for typesetting of all kinds of publications in all languages, but as a personal productivity tool for typesetting of The Art of Computer Programming (Knuth, 1973a) in American English. This was the initial motivation.

In this chapter we continue elaborating these issues, with the emphasis on the hyphenation problems in the presence of long compound words in Germanic (and Slavic) languages.

7.2 Problems

7.2.1 Compounds

The main problem with automatic hyphenation was nicely expressed in ISO-10646 electronic discussion list by Jarnefors (1995):

> "The leading Swedish daily newspaper Dagens Nyheter had severe problems with sometimes occurring incorrect hyphenations a couple of years ago. It (and its computerized typesetting) was

> during a period the object of much amusement, ridicule and irritation from its readers. These problems have more or less disappeared, and I've learnt that this is only because, nowadays, *every* hyphenation in the newspaper is manually checked by human proof-readers. Because of the higher frequency of long words in Swedish compared to e.g., English or French, around a third of all lines in a typical newspaper article (with approximately 30 characters per line) end with a hyphenated word.
> The hyphenation problems in Swedish have to do with the high frequency of compound words (the Swedish vocabulary cannot be enumerated: new compounds are easily created by anyone) and the rule that a compound word shall always be hyphenated between the constituent word parts, to ease the flow of reading."

For instance, in German and Czech there are no hyphens in compound words, you take the first word, rarely a fill-char and the second word. In some languages, compounds are built with hyphens. With this construction, it is easy to break at the end of line and to spell-check. However, in most of the languages compound word boundaries cannot be deducted from syntax only.

7.2.2 Dependency of Hyphenation Points on Semantics

In some cases, even the context of the sentence is needed in order to be able to decide on the hyphenation point. A collection of examples for several languages follows:

Czech *nar*|*val* 'narwhal' and *na*|*rval* 'gathered by tearing, plucked'; *pod*|*robit* 'subjugate, to bring under one's domination' and *po*|*drobit* 'to crumble'; *o*|*blít* 'to vomit up' and *ob*|*lít* 'to pour around'

Danish *træ*|*kvinden* 'the wood lady' and *træk*|*vinden* 'the draught'; *ku*|*plet* 'verse' and *kup*|*let* 'domed'

Dutch *kwart*|*slagen* 'quarter turns' and *kwarts*|*lagen* 'quartz layers'; *go*|*spel* 'the game of Go' and *gos*|*pel* 'certain type of music'; *rots*|*tempel* 'rock temple' and *rot*|*stempel* 'damned stamp'; *dij*|*kramp* 'cramp in the thighs' and *dijk*|*ramp* 'dike catastrophe'; *ver*|*ste* 'farthest' and *vers*|*te* 'most fresh'.

English *rec*|*ord* (noun) *re*|*cord* (verb) or even *record* (adjective).

German *Staub*|*ecken* 'dusty eck' and *Stau*|*becken* 'traffic jam in the valley'; *Wach*|*stube* 'guard room' and *Wachs*|*tube* 'wax tube'; *Bet*|*tuch* and *Bett*|*tuch*.

Fortunately, number of these homonyms is far below 1% and for such a small number of possible hyphenation points it is not worth to do full semantical analysis.

7.2.3 Exceptions

Some hyphenation points are forbidden because of unwanted connotations the new parts of the word may have:
Czech *kni|hovna, sere|náda, tlu|močení, se|kunda*
English *the|rapists, anal|ysis*
German *Spargel|der, beste|hende, Gehörner|ven, bein|halten, Stiefel|tern*

7.2.4 Discretionary Hyphenation Points

1. `\discretionary{`*xx*`}{`*x*`}{`*xx*`}` (in German, *x* is a consonant `f`, `l`, `m`, `n`, `p`, `r` or `t`)
   Now, let us consider the situation that the first word ends with a double consonant and the second word starts with the same consonant. If the second letter of the second word is a consonant, nothing changes — *Sauerstoff* + *Flasche* composes to *Sauerstoffflasche*. If the second letter of the second word is a vowel, the three consonants will be reduced to two — *Schiff* + *Fahrt* composes to *Schiffahrt*. One can find meaning-dependent discretionaries: *Bett|tuch* 'sheet' vs. *Bet|tuch* 'prayer shawl'.
2. `\discretionary{k}{k}{ck}` (German)
   This discretionary (as most of the others) has the rationale in the fact that pronunciation of `c` depends on the following letter (as in other languages). If hyphen occurs just after the letter `c`, the reading is slowed down because the reader does not know how to pronounce it and the eye has a long way to the beginning of the next line.
   Even here the hyphenation can depend on the word meaning: word *Druckerzeugnis* is hyphenated *Druck|erzeugnis* in case of 'printed matter' or *Druk|kerzeugnis* when speaking about 'certificate for a printer'.[1]
3. `\discretionary{a}{}{aa}` (Dutch)
   There is another type of discretionary in which a character is deleted in case hyphenation occurs — word *omaatje* becomes *oma|tje* when hyphenated.
4. `\discretionary{é}{}{ee}` (Dutch)
   Apart from character deletion another change may occur: *cafeetje* becomes *café-tje* when hyphenated.
5. `\discretionary{l}{l}{l·l}` (Catalan)

1. The German speaking countries are in the process of introducing new rules for hyphenation, in which `ck` is not any more allowed to be hyphenated. With the new rules, an old way which was introduced in 1902 — e.g. hyphenation of *Zuk|ker* 'sugar' might change to *Zu|cker* in the future norm.

In Catalan the word paral·lel is broken as paral|lel, intel·ligencia as intel|ligencia. l·l is considered as one character (trigraph). With this hyphenation it changes to another two characters.

7.2.5 Language Evolution

Another complication is the fact that a language is not fixed, non-evolving entity, but it changes, sometimes quite rapidly. New words, especially compounds, are being adopted every day. An example of an adaptation of a language to the technology — the typewriter and telegraphy in this case — may serve different spelling allowed for umlauted characters ä, ö, ü and ß in German (ae, oe, ue, ss). Some compounds are becoming to be percepted as base words. Thus the idea of fixing hyphenation algorithm/patterns once and forever is not a clever one.[2] A solution may consist in a relatively easy generation of algorithm or patterns from the updated dictionary or description of changes.

7.3 Solutions

7.3.1 Compounds

It is obvious that we need to take the burden of the manual markup of compound word borders from the writer and leave it to the machine (typesetting system). The proper solution of this problem is a language module for every language, with the ability of creating new words by composition from others. This module, based on the morphology of a language, is needed, e.g., in a spellchecker for a given language anyway. Most probably, such language modules will become a part of the language support of operating systems in near future. Such dynamic libraries will be shared among software applications. Building such a module, however, is not a trivial task, because only some of the compounds are meaningful words.

Looking for a temporary TEX patch that will help the current TEX users, especially those writing in Germanic and Slavic languages, the following algorithm may be used (compare with (Sojka and Ševeček, 1994)):

1. For a particular language a special word list is created, which contains all word forms, but only compound word borders are marked there.
2. Hyphenation patterns from this word list are created by PATGEN (Liang and Breitenlohner, 1991).

2. When storing document for later retypesetting with TEX we also have to save the hyphenation patterns.

Table 7.1: Example of a discretionary hyphenation table for German.

pre break text 1	post break text 2	no break text 3	left context 4	right context 5	discretionary character 6	example 7
k	k	ck	c	k	c_1	Drucker
ek	k	äck	äc	k	c_2	Bäcker
ff	f	f	f	f	c_3	Schiffahrt
ll	l	l	l	l	c_4	Rolladen
mm	m	m	m	m	c_5	Programmeister
nn	n	n	n	n	c_6	Brennessel
pp	p	p	p	p	c_7	Stoppunkt
rr	r	r	r	r	c_8	Herraum
tt	t	t	t	t	c_9	Balettheater

3. A special pass in the paragraph breaking algorithm of TEX (for the detailed description consult (Knuth and Plass, 1981; Knuth, 1986a; Knuth, 1986b)) is added after the first (no hyphenation trial) pass. Words are hyphenated using the compound word patterns. Then, an extra penalty `\compoundwordhyphenpenalty` is associated with these hyphenation points.
4. If `\tolerance` has not been met by now, further hyphenation points are added using the 'standard' patterns. These new hyphenation points have associated `\hyphenpenalty`, allowing differentiation between the two types of hyphenation points.
5. Hyphenation points 'near' the word borders (specified by `\leftdiscretionaryhyphenmin` and `\rightdiscretionaryhyphenmin` are suppressed (removed).
6. The algorithm now continues with the 'old' second and eventually the third (`\emergencystretch`) passes.
7. `\compoundwordchar` (e.g., as in Cork-coded fonts `\char'027`) is included at the compound word breakpoint in order to prevent ligatures spanning over the word borders šéflékař 'chief doctor' versus šéflékař which is wrong due to fl ligature).

7.3.2 Discretionary Hyphenation Points

Manual insertion of discretionary points is tedious and it is usually forgotten[3], leading to typographic errors.

3. How many of you, TEX users, remember to type `eigh\discretionary{t}{t}{t}een` instead of just `eighteen`?

One solution is the following one. For every language, a table of possible discretionaries is created. For a German example see Table 7.3.1 on the previous page. Words with these discretionaries are added to the word list with the "discretionary character" inserted between the "left context" and the "right context". From such an extended word list the patterns are generated.

The hyphenation algorithm of TEX (Knuth, 1986a, parts 38 – 43, sections 813 – 965) has to be extended. Roughly speaking:

1. As a first step, "normal" hyphenation points in the word in question are found.
2. The discretionary exception table is looked up (similar to the `\hyphenation` list of exception). If the word is found there, the discretionary is inserted and the algorithm ends, otherwise it is continued by step 3.
3. The discretionary table is looked up and at the hyphenation points that match "left and right context" strings (columns 4 and 5 in Table 7.3.1 on the preceding page), the "discretionary character" (column 6) is inserted. Such a word is hyphenated once again to check whether this discretionary really applies at this position. If so, the corresponding discretionary (columns 1 – 3 of the Table 7.3.1 on the previous page) is automatically inserted.
4. "Normal" hyphenation points, which appear 'near' to "discretionary" hyphenation points (within the 'window' specified by the values of counters `\leftdiscretionaryhyphenmin` and `\rightdiscretionary-hyphenmin`), are removed.

This approach takes the advantage of the data structure used for storing the information about the hyphenation points. The patterns are stored using the *trie* data structure (Knuth, 1973b, pp. 481 – 505). This data structure implicitly allows an effective prefix compression, and in the case of *packed tries* (Liang, 1983) even suffix compression by suffix sharing. Because of that, the increase in the size of the patterns is negligible, as the patterns doublets share both prefix and suffix parts in the trie.

Also, the look up time in the trie is linear with respect to the word length of hyphenated words. The time needed for looking up in the trie for the second time is thus acceptable — it is only performed sometimes — when the context of a hyphenation point is matched in the discretionary table.

The algorithm is backward compatible in the sense that if discretionary table is not present for the current language, nothing changes with respect to the standard TEX behavior.

7.3.3 Exceptions

The exceptions can be reasonably handled by the patterns. Although the generation of patterns for languages with lots of exceptions may lead to complex patterns, it is much better to regenerate the patterns with the exceptions than maintain huge lists of exceptions and to slow down the processing considerably.

Because regenerating of patterns is not always possible, to enable enrichment of the knowledge of discretionary hyphenation points compiled into the patterns, it is wise to introduce a new `\discretionaryhyphenation` for this purpose.

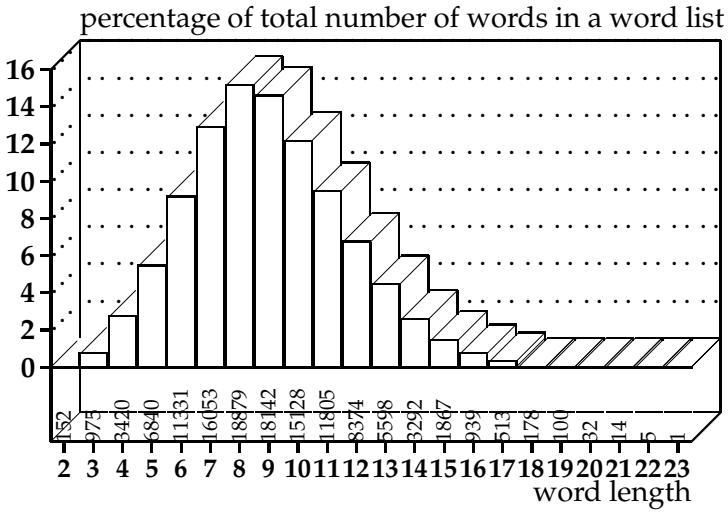

Figure 7.1: English word list statistics: US English word list (123 664 words), average word length 8.93 characters.

7.4 Experiments

We had several databases of words available for experiments. For inflexional languages (Czech, German), they were based on morphology, for English it was just a list of word forms. We did our PATGEN experiments with the German word list generated from the full word list by our stratified sampling technique very similar to that described in (Sojka and Ševeček, 1994, page 63) for Czech. We took German because the problems there are the most serious. Simple statistics show how the languages differ.

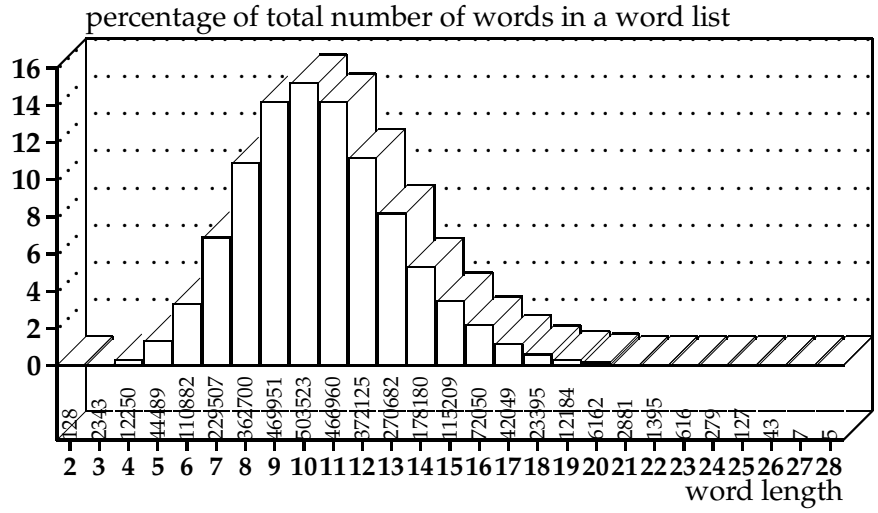

Figure 7.2: Czech word list statistics: Czech word list (3,300,122 words), average word length 10.55 characters.

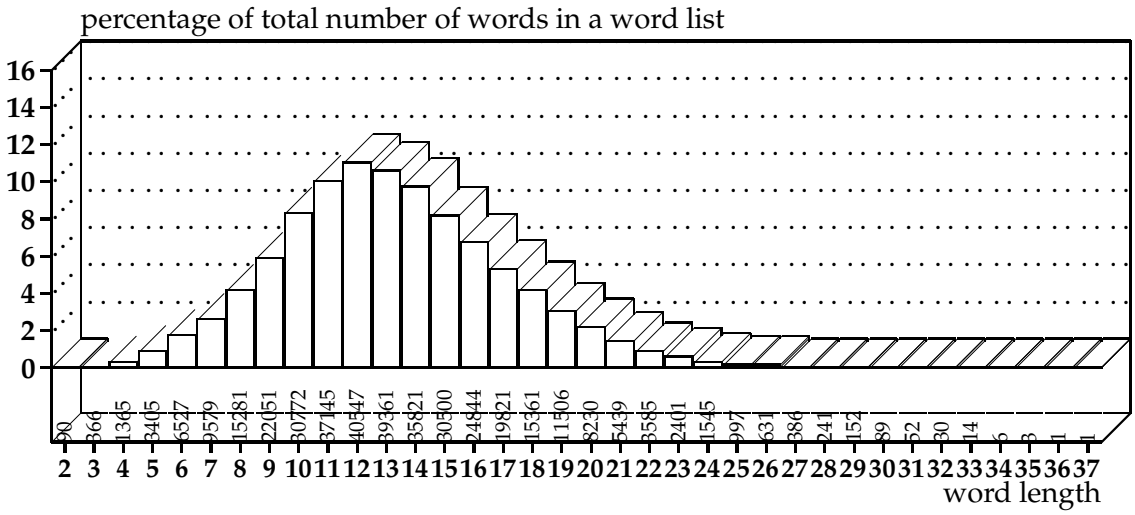

Figure 7.3: German word list statistics: German word list (368,152 words), average word length 13.24 characters

7.4.1 Non-Uniformity of Languages

In Tables 7.1, 7.2 and 7.3 there are histograms of word lengths in our databases. Although it is clear that shorter words are more frequent then the long ones, we see that in German the average word is much longer than in English and also in Czech. It is interesting to compare the total number of words. As Czech is very inflexional language, from about 170,000 word stems we got more than 3,300,000 word forms. One can compare that with the best English dictionaries and spellers, which do not have more than 200,000 word forms. The ratio of

the total number of word forms to the number of word stems for German is about 3 (we have about 120,000 word stems), but for Czech it is almost 20.

The average word length depends on the word list chosen, but in general our results are commensurable with the result published for Welsh (Haralambous, 1993) — 9.71 characters per word, but the words like *Llanfairpwllgwyngyllgogerychwryndrobwllllantysiliogogogoch* were not taken into account there.

Table 7.2: German compound word hyphenation with pattern size optimized strategy (cf. Table 6.4 on page 37).

level	length	param	% correct	% wrong	# patterns	statistics
1	1–3	1 2 20	62.41	13.38	+ 472	good=134279
2	2–4	2 1 8	52.89	2.53	+ 712	bad=676
3	3–5	1 4 7	87.11	4.05	+2,951	missed=22636
4	4–6	3 2 1	85.57	0.43	+1,506	patterns size=33.6 kB

Table 7.3: German compound word hyphenation with different (percentage of correct optimised) strategy.

level	length	param	% correct	% wrong	# patterns	statistics
1	1–3	1 2 20	62.41	13.38	+ 472	good=143478
2	2–4	2 1 8	52.89	2.53	+ 712	bad=698
3	3–5	1 4 3	93.06	4.23	+6,612	missed=13437
4	4–6	3 2 1	91.44	0.44	+1,586	patterns size=56.5 kB

Table 7.4: German compound word hyphenation covering even more break points.

level	length	param	% correct	% wrong	# patterns	statistics
1	1–3	1 3 1	60.43	9.87	+4,819	good=149502
2	1–4	1 3 2	60.24	4.21	+1,714	bad=888
3	3–6	1 2 1	98.76	10.82	+1,939	missed=7413
4	3–7	1 1 1	95.28	0.57	+ 353	patterns size=70.2 kB

7.4.2 Compounds in German

In the word list, only the compound word borders and prefixes were marked. This led to about 150,000 positions in our German word list. The words without any breaks of this kind were not removed. The results of PATGEN runs applied to this word list are summarized in Tables 7.2 and 7.3. The

Table 7.5: Standard German hyphenation pattern generation with slightly improved (size) Liang's parameters.

level	length	param	% correct	% wrong	# patterns	statistics
1	1–3	1 2 20	94.25	23.72	+449	good=485590
2	2–4	2 1 8	82.66	0.56	+1,183	bad=48
3	3–5	1 4 7	98.59	1.08	+1,737	missed=8047
4	4–6	3 2 1	98.37	0.01	+1,333	patterns size=25.2 kB

Table 7.6: German hyphenation pattern generation using a word list with discretionary points added (the same parameters as in Table 7.5).

level	length	param	% correct	% wrong	# patterns	statistics
1	1–3	1 2 20	93.90	23.40	+ 456	good=492366
2	2–4	2 1 8	82.48	0.55	+1,182	bad=60
3	3–5	1 4 7	98.60	1.13	+1,760	missed=8155
4	4–6	3 2 1	98.37	0.01	+1,388	patterns size=25.6 kB

efficiency achieved (about 90% breaks covered) is quite sufficient, as a 'normal' hyphenation pass follows and the error when a hyphenation point is classified as 'normal' instead of 'compound' reflects only a different penalty associated with this break. At the expense of pattern size we can do even better (see Table 7.4 on the preceding page).

7.4.3 Discretionary Hyphenation Points

In our German word list, we had 1,626 words with the `c-k` discretionary and 42 words with the discretionary hyphenation of type *x*-*x*, where *x* is a consonant — see Table 7.3.1 on page 51, (Raichle, 1995) or (DUDEN, 1991) for a list of possible discretionaries in German.

Then we created doublets of these words with these discretionaries by inserting the discretionary character (column 6) at the hyphenation position and added them to our word list. Then we applied PATGEN at this new word list. Results can be compared in Tables 7.5 and 7.6. The difference in the pattern size is small as expected — the size of the pattern file increased by less than 0.4 kB, which makes a difference in the trie structure of about 100 bytes only.

7.5 Conclusions

We claim that the integration of language modules with build-in knowledge about a particular language is a must in today's top-rated systems for publishing. We suggested extensions of hyphenation algorithms of TEX that may help

with hyphenation especially in the Germanic languages with high frequency of compound words and discretionary hyphenation. Suggested extensions are possible with limited changes to TEX — The Program (Knuth, 1986a). Their implementation in any conservative successor to TEX will be rather straightforward and when agreed on their usefulness they will be implemented as independent change files in the future. It remains to be decided on the primitives our approach needs.

7.6 Summary

Some computerized typesetting methods in frequent use today may render a conservative approach to word division impractical. Compromise may therefore be necessary pending the development of more sophisticated technology.
— Anonymous (1993, Section 6.43)

We have outlined some of the possibilities offered by TEX and PATGEN for the development of customized hyphenation patterns. We have suggested bootstrapping and iterative techniques to facilitate pattern development. In addition we are suggesting wider employment of PATGEN and preparation of hyphenated word lists and modules of patterns for easy preparation of hyphenation patterns on demand in today's age of digital typography (Knuth, 1999).

References

DUDEN. 1991. *Duden Band 1 — Rechtschreibung der Deutschen Sprache*. Dudenverlag, 20., neu bearbeitete und erweiterte Auflage edition.

Yannis Haralambous. 1993. Using `PATGEN` to Create Welsh Patterns. Submitted to TUGboat, July.

Donald E. Knuth and Michael F. Plass. 1981. Breaking Paragraphs into Lines. *Software—Practice and Experience*, 11(11):1119–1184, November.

Donald E. Knuth. 1973a. *Fundamental Algorithms*. The Art of Computer Programming. Addison-Wesley, Reading, Massachusetts, second edition.

Donald E. Knuth. 1973b. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1986a. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1986b. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1999. *Digital Typography*. CSLI Lecture Notes 78. Center for the Study of Language and Information, Stanford, California.

Gabriele Kodydek and Martin Schönhacker. 2003. Si3Trenn and Si3Silb: Using the SiSiSi Word Analysis System for Pre-Hyphenation and Syllable Counting in German Documents. Lecture Notes in Artificial Intelligence LNCS/LNAI 2807, pages 66–73, České Budějovice, Czech Republic, September. Springer-Verlag.

Franklin M. Liang and Peter Breitenlohner. 1991. `PAT`tern `GEN`eration program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.0 from UNIXTEX distribution at `ftp://ftp.cs.umb.edu`, November.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Department of Computer Science, Stanford University, August.

Bernd Raichle. 1995. Kurzbeschreibung – `german.sty` (version 2.5), April. Available from CTAN archives.

Kevin Patrick Scannell. 2003. Hyphenation patterns for minority languages. *TUGboat*, 24(2):236–239.

Petr Sojka and Pavel Ševeček. 1994. Hyphenation in TEX — Quo Vadis? In Włodek Bzyl and Tomek Przechlewski, editors, *Proceedings of the 9th European TEX Conference, Gdańsk, 1994*, pages 59–68, September.

Chapter 8

Word Hy-phen-a-tion by Neural Networks

A comparison of competing patterns approach with feedforward neural networks to solve the hyphenation problem is presented in this chapter. It was drafted as a technical report (Smrž and Sojka, 1996) as an outcome of Master's thesis of Pavel Smrž supervised by the latter author. The final version has been published in the journal Neural Network World (Smrž and Sojka, 1997):

> Pavel Smrž and Petr Sojka. 1996. Word Hy-phen-a-tion by Neural Networks. Technical Report FIMU-RS-96-04, Masaryk University in Brno, Faculty of Informatics, August.
>
> Pavel Smrž and Petr Sojka. 1997. Word Hy-phen-a-tion by Neural Networks. *Neural Network World*, 7:687–695.

This journal version appears here with only minimal changes. Authors agree that their proportion on the results is the same.

Word Hy-phen-a-tion by Neural Networks

Pavel Smrž and Petr Sojka

Abstract: We are discussing our experiments we made to learn a feed-forward neural network for the task of finding valid hyphenation points in all words of a given language. Multilayer neural networks were successfully used to solve this difficult problem. The structure of the network used is given, together with a discussion about training sets, influence of input coding and results of experiments done for the Czech language. We end up with pros and cons of the tested approach — hybrid architecture suitable for a multilingual system.

Key Words: neural networks • hyphenation • back propagation • generalization • typesetting • multilayer perceptron

8.1 Introduction

> *The invention of the alphabet was one of the greatest advances in the history of civilization. However, the ancient Phœnicians probably did not anticipate the fact that, centuries later, the problem of word hyphenation would become a major headache for computer typesetters all over the world.* — Liang (1983, page 39)

The problem of finding all valid hyphenation points in all words of a given language has been tackled for decades. Most of the approaches used so far are deterministic. A rule-driven hyphenation algorithm for English was implemented in TEX78 (Liang, 1981). The method was improved by Liang (1983) for use in TEX82 (Knuth, 1986). It is based on the generalization of the prefix, suffix and vowel-consonant-consonant-vowel rules. The program `PATGEN` (Liang and Breitenlohner, 1991) enables the process of pattern generation from a set of already hyphenated words to be automated. This algorithm or its derivatives are used in many DTP systems like `troff` (Emerson and Paulsell, 1987), `Lout`, QuarkXpress, 3B2, Scribus and many others today.

Liang's algorithm performs well for nonflexional languages with a small number of compounds like English but there is still lack of good methods for other languages, especially for inflexional languages (all Slavonic languages, Dutch, German etc). Sojka and Ševeček (Sojka and Ševeček, 1995; Sojka, 1995) state that in Czech, on average, 20–30 different word forms — inflexions — can be derived from one word stem. This number can be almost

doubled if negatives are formed from many words (adjectives, verbs, adverbs, some nouns) by adding the prefix `ne`. Thus, from a 170,000 stem word list about 5,000,000 inflexions may be generated in Czech.

For multilingual documents usually several separate algorithms for every language used are needed, even if the languages are only dialects, leading to high computer memory demands. Typesetting in narrow columns brings the necessity to find very high percentage of all valid hyphenation points.

From the DTP world and prominent publishers another need is being heard of: several classes of hyphenation points are called for, to make a distinction, e.g., between valid and not recommended, but possible one, as published in (Allen, 1990).

This leads to the stochastic approaches rather than deterministic ones — Brunak and Lautrup (1990) show that a neural network is likely to be a way leading quickly to the working solution.

8.2 Hyphenation Problems with Neural Networks

8.2.1 Hyphenation of Czech Words

We performed our experiments with multilayer neural networks trained on hyphenation for Czech language. The problem of word hyphenation in Czech is rather complex. Hyphenation rules for Czech language are described in (Hlavsa and others, 1993) and (Haller, 1956). In (Haller, 1956) also a list of exceptions is given including about 10,000 words. Czech language has syllabic hyphenation with "etymological" exceptions. Hyphenation is preferred between a prefix and the stem and on the boundary of compound words.

8.2.2 Neural Net Architecture

The architecture of the networks used in the experiments is similar to that of NETtalk (Sejnowski and Rosenberg, 1987) — multilayer feedforward nets. We use usual notation here: 7-30-1 means NN topology with 7 neurons in the input layer, 30 neurons in the middle layer and 1 neuron in the output layer. Layers are fully interconnected.

The input of our network is a series of seven consecutive letters from one of the training words. The central letter in this sequence is the "current" one for which the output is to be produced. Three letters on either side of this central letter provide the context that helps to determine the hyphenation point. Individual words are moved through the input window so that each

letter in the word with the exception of the last two is "seen" in the central position. Blanks are added before and after the word as needed.

One type of tested networks uses unary encoding. For each of the seven letter positions in the input, the network has a set of 43 input units: one for each of 41 letters in Czech, one for letters from other languages, and one for blank. Thus, there are $43 \times 7 = 301$ input units. Another tested type uses real numbers rather than binary ones for encoding the input. The letters are coded in the form of numbers from the set $\{0.02, 0.04, \ldots, 0.98\}$. The exact coding of a particular letter will be given later.

The networks have one or two output neurons. In the first case, the meaning of the output value is 0 for 'do not hyphenate' and 1 for 'insert hyphenation point'. In the second case the output 0 1 means 'hyphenate', 1 0 'do not hyphenate'.

8.2.3 Training Sets Used

We had a set of 169,888 hyphenated Czech words to experiment with. The problem with this set was a considerably large number of errors. There are two types of errors. The first type is probably the worse one — the hyphen is placed in the position where the word cannot be hyphenated. In the case of errors of the second type the algorithm is not able to find an allowed hyphenation point. These errors are a big complication of typesetting in narrow columns.

8.3 Empirical Results — Brute Force Trial

For the first group of experiments the networks with the topologies 301-30-1 and 301-100-1 were employed. In both cases each layer was completely interconnected with the next one. The training set was divided into 170 parts each containing 1000 words. Totally, 1,581,183 training patterns were generated.

The network was trained with each part of the training set. The training was carried out until the network error dropped below the value of 0.1 or until 100 cycles were reached. The learning rate was initially set to the value of 0.7. Then it was stepwise decreased in each training by 0.1 to the final value of 0.3 which was used for the rest of learning.

Naturally, this learning process was extremely time consuming. The training of the network 301-30-1 took about 17 days of user time on Sun SparcStation 10 not taking into account the time for generating patterns. For the training of the network 301-100-1 the supercomputer Silicon Graphics POWER Challenge L was used. Despite its computing power the training took about 18 days of user time.

The results of the experiments described above are summarized in Tables 8.1 and 8.2. Although in the latter case the network contained more than three-fold number of connections, the number of wrong patterns was almost the same. It is obvious that, in this case, the performance of the network cannot be significantly improved by increasing the number of hidden layer neurons only.

Table 8.1: Results of learning of the network 301-30-1 with 1,581,183 training patterns.

```
STATISTICS   (1581183 patterns)
wrong: 3.43% (  54282 patterns)
right:96.57% (1526901 patterns)
```

Table 8.2: Results of learning of the network 301-100-1 with 1,581,183 training patterns.

```
STATISTICS   (1581183 patterns)
wrong: 3.26% (  51599 patterns)
right:96.74% (1529584 patterns)
```

As stated earlier, the biggest problem with the training of word hyphenation is to obtain a good training set. It seems to be unrealistic to avoid all errors but it is necessary to try to find patterns with the least number of wrongly hyphenated words and with the maximum of correctly marked hyphenation points. The file of 169,888 hyphenated words contained many errors. Therefore, when the network was tested using this file, some correctly hyphenated words were considered erroneous by the system. Remaining errors mainly occurred in words which belong to the exceptions from hyphenation rules, especially in words adapted from foreign languages.

To test if a network can even learn all the exceptions from hyphenation rules, all 54,282 training patterns wrongly hyphenated by the network 301-30-1 were used as one big training set and presented to another network of the type 301-30-1. Learning rate decreased stepwise from the value 0.8 to 0.2. After 100 cycles the network learnt all but 4 training patterns which is an excellent result.

8.4 The Influence of Input Coding: Use of Real Numbers

All the following experiments were carried out with the training set containing 78,809 hyphenated words beginning with the letter `m`. The number of errors in these data was very low. The number of errors of the first type was negligible and the relative number of errors of the second type was smaller too. A subset of 1,000 words in which the errors were corrected was used for most of shorter experiments. They were performed with the networks with seven input layer neurons for seven consecutive letters. Each letter was coded as a real number.

␣ 0.02	a 0.04	á 0.06	b 0.08	c 0.10	č 0.12	d 0.14
ď 0.16	e 0.18	é 0.20	ě 0.22	f 0.24	g 0.26	h 0.28
i 0.30	í 0.32	j 0.34	k 0.36	l 0.38	m 0.40	n 0.42
ň 0.44	o 0.46	ó 0.48	p 0.50	q 0.52	r 0.54	ř 0.56
s 0.58	š 0.60	t 0.62	ť 0.64	u 0.66	ú 0.68	ů 0.70
v 0.72	w 0.74	x 0.76	y 0.78	ý 0.80	z 0.82	ž 0.84

Table 8.3: Coding of letters according to the alphabet.

In the beginning, the codes of letters were assigned according to the alphabet (see Table 8.3).

The results of experiments with the network 7-30-9-2 are summarized in Table 8.4. It is obvious that these results are not satisfactory as the network error is too high. The network wrongly hyphenated even words often used with simple syllabic hyphenation. It did not recognize the rules of making syllables, did not take into account which letters are vowels and which consonants. Therefore, this approach proved to be inapplicable due to poor generalization achieved.

Table 8.4: Results of experiments with the network 7-30-9-2 and coding according to Table 8.3.

```
STATISTICS          (8411 patterns)
wrong   : 14.12 %   (1188 patterns)
right   : 85.88 %   (7223 patterns)
```

In order to improve results, learning with another input coding of letters was used. It is shown in Table 8.5 on the following page. All vowels in Czech were coded as small numbers in the range of $\langle 0.02, 0.28 \rangle$, all consonants except `r` and `l` as numbers from $\langle 0.50, 0.98 \rangle$. Letters `r` and `l` are consonants but can make syllables in Czech. Therefore, they were coded using numbers 0.40 and 0.42 separately from the other consonants. The blank was coded as 0.34, i.e. as a number between code numbers of vowels and consonants.

Table 8.5: Alternative coding of letters.

a 0.02	á 0.04	e 0.06	é 0.08	ě 0.10	i 0.12	í 0.14
o 0.16	ó 0.18	u 0.20	ú 0.22	ů 0.24	y 0.26	ý 0.28
		␣ 0.34			r 0.40	l 0.42
			b 0.50	c 0.52	č 0.54	d 0.56
ď 0.58	f 0.60	g 0.62	h 0.64	j 0.66	k 0.68	m 0.70
n 0.72	ň 0.74	p 0.76	q 0.78	ř 0.80	s 0.82	š 0.84
t 0.86	ť 0.88	v 0.90	w 0.92	x 0.94	z 0.96	ž 0.98

The results of the experiments with the network 7-30-9-2 and the coding described above are shown in Table 8.6. The comparison of results with both coding alternatives is given in Figure 8.1 on the following page.

Table 8.6: Results of experiments with the network 7-30-9-2 and coding according to Table 8.5.

```
STATISTICS          (8411 patterns)
wrong   :  3.41 %   ( 287 patterns)
right   : 96.59 %   (8124 patterns)
```

Information about the type of a letter (consonant or vowel) helped the network to generalize. The different coding of letters `r` and `l` also improved learning. The results with this network could be probably further improved by another sorting of input letter codes. Many errors were caused by a special nature of joined letters `c` and `h`. In Czech they both are used together as a two-character symbol of one sound and, in fact, they form a sort of a single letter. Thus, `c` and `h` cannot be separated by a hyphen. A solution of this problem may be to code joined `c` and `h` by a special number which would differ from the codes of both single letters.

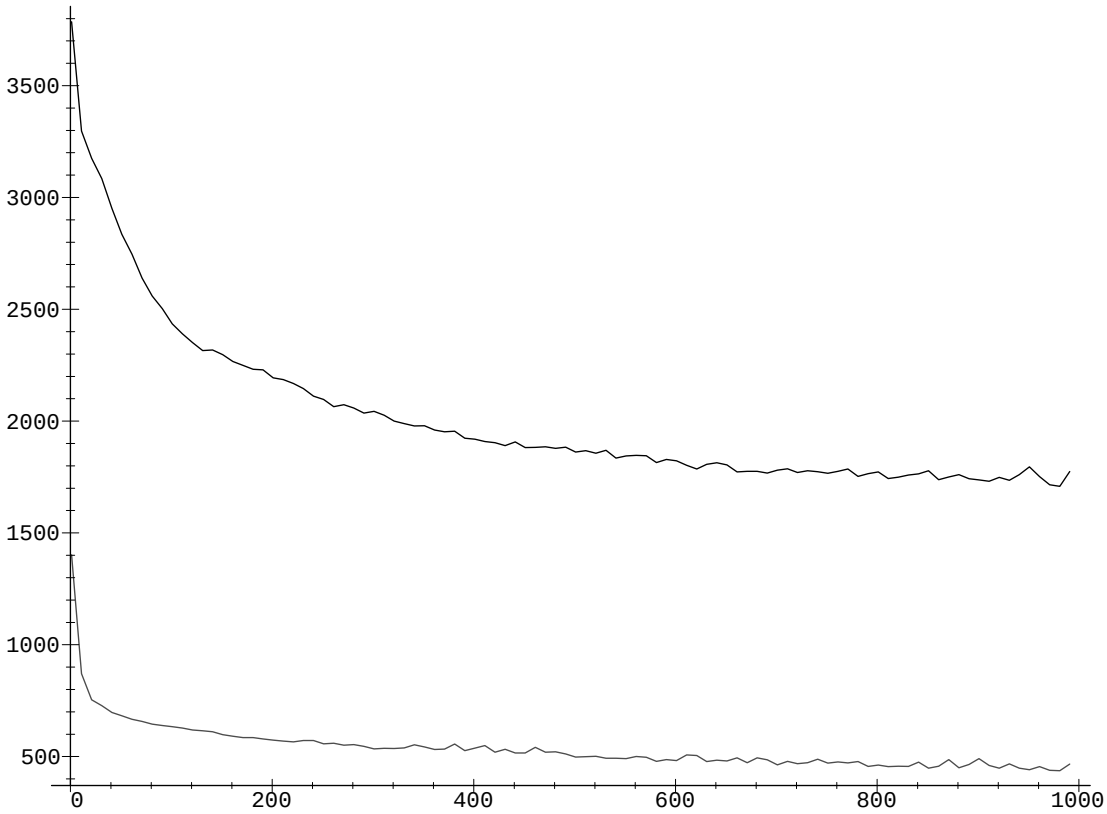

Figure 8.1: Comparison of the results of experiments with the network 7-30-9-2 using both coding alternatives. Upper curve: Coding according to Table 8.3 on page 64. Lower curve: Coding according to Table 8.5 on the preceding page.

8.5 Comparison of Various Topologies

The next series of experiments was designed to compare the abilities of networks with different topology. Networks 301-30-1, 301-60-1, 7-30-1, 7-30-9-2, and 7-60-2 were compared. In the case of networks with seven input neurons the alternative coding was used (as described in Table 8.5 on the previous page). A detailed description of results can be found in (Smrž, 1995).

No significant difference in learning performance was observed between the networks with topologies 301-30-1 and 301-60-1. A similar result was obtained earlier using the networks 301-30-1 and 301-100-1 and the other training set — see Section 8.3 on page 62.

The results obtained with the network 301-30-1 are distinctly better than those with networks consisting of a smaller number of neurons and synapses for which different coding was necessary. On the other hand, this network needs much more memory for weight storage.

Learning and generalization performance of a network is significantly influenced not only by the number of hidden layer neurons but also by the network topology (Bugmann et al., 1992). Using the network 7-30-9-2 with two hidden layers, better results were obtained though the total number of neurons and connections was smaller than that of the network 7-60-2 with only one hidden layer.

Next, the generalization ability of the networks 7-30-9-2 and 301-30-1 was studied. The networks were trained with the subset of 1,000 words. Then the whole set of 78,809 words was used for testing. The results are given in Tables 8.7 and 8.8. The percentage of wrongly hyphenated words can be considered very low if the number of errors in the set used for testing is taken into account.

Table 8.7: Generalization ability of the network 7-30-9-2.

```
STATISTICS    (648928 patterns)
wrong:  4.91% ( 31886 patterns)
right: 95.09% (647042 patterns)
```

Table 8.8: Generalization ability of the network 301-30-1.

```
STATISTICS    (648928 patterns)
wrong:  2.78% ( 18057 patterns)
right: 97.22% (630871 patterns)
```

To sum up our observation: to train a neural network to perform hyphenation in a language one should:

1. Create a training set with proofread hyphenated words without errors with all exceptions from generalizable rules.
2. Make a clever ordering of letters in the given language reflecting "similarity"/"exchangeability" of letters.
3. Use a topology with two hidden layers might be cheaper in memory consumption but learning is then harder.

The learning process itself can be used for finding errors in training data (proofreading of words that were not learnt). This gradual bootstrapping process may lead to a perfect network.

8.6 Syllabic Hyphenation

Finally, it was tested how well a network would perform if only the type of letters (consonants or vowels) was given. The network 7-30-1 was used. Consonants were coded as `0`, vowels as `1` and blank as `0.5`. The results of these experiments are given in Table 8.9 on the next page. The results clearly show that the syllabic hyphenation plays a dominant rôle in Czech language. However, as the error was about 6%, it was obvious that if only the syllabic hy-

phenation was included in the algorithm, the results would be unsatisfactory for everyday use.

Table 8.9: Results of experiments with the network 7-30-1 and consonant/vowel coding.

STATISTICS		(8411 patterns)
wrong	: 6.08 %	(511 patterns)
right	: 93.92 %	(7900 patterns)

8.7 Discussion

The results obtained for Czech hyphenation are close to those for "classical" approach showed in (Sojka and Ševeček, 1995; Sojka, 1995). Testing the "syllabic hyphenation neural network" on "close" languages (e.g., syllabic ones), preprocessed for accents, gives similar results. This fact allows to build a modular hybrid system, in which separate neural networks will be trained to cover "close" languages, and hyphenation of words not covered by them will be stored in the exception tries in the PATGEN fashion. Such a system is able not only perform well if properly tuned up — in addition — it can be trained to give a measure of suitability of hyphenation points found for the [DTP] system.

8.8 Conclusion and Acknowledgments

We showed that solving the word hyphenation problem with neural networks is possible and that generalization abilities of neural networks allow to build a working system for the given task. Combining with exception lists, we can build a quality system which is able to store hyphenation points for several languages with moderate memory needs.

We acknowledge the possibility to use computer facilities of Supercomputing Center Brno.

References

R. E. Allen. 1990. *The Oxford Spelling Dictionary*, volume II of *The Oxford Library of English Usage*. Oxford University Press.

Guido Bugmann, Petr Sojka, Michael Reiss, Mark Plumbley, and John G. Taylor. 1992. Direct Approaches to Improving the Robustness of Multilayer Neural Networks. pages 1063–1066, Brighton, UK, September. Elsevier Science Publishers B.V.

Sandra L. Emerson and Karen Paulsell. 1987. *troff Typesetting for* UNIX™ *Systems*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Jiří Haller. 1956. *Jak se dělí slova (How the Words Get Hyphenated)*. Státní pedagogické nakladatelství Praha.

Zdeněk Hlavsa et al. 1993. *Pravidla českého pravopisu (The Rules of the Czech Spelling)*. Academia Praha.

Donald E. Knuth. 1986. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Franklin M. Liang and Peter Breitenlohner. 1991. `PAT`tern `GEN`eration program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.0 from UNIXTEX distribution at `ftp://ftp.cs.umb.edu`, November.

Franklin M. Liang. 1981. TEX and hyphenation. *TUGboat*, 2(2):19–20, July.

Terry J. Sejnowski and C. R. Rosenberg. 1987. Parallel Networks that Learn to Pronounce English Text. *Complex Systems*, 1:145–168.

Pavel Smrž. 1995. Learning Algorithms of Neural Networks. Master's thesis, Masaryk University in Brno, April.

Petr Sojka and Pavel Ševeček. 1995. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Chapter 9

Hyphenation on Demand

In this chapter it is shown that the technique of competing patterns can be tailored to the specific problems in the area of computer typesetting.

The first version of this paper was presented at the TUG 1999 conference in Vancouver (Sojka, 1999a) and final version appeared as the journal publication (Sojka, 1999b):

> Petr Sojka. 1999a. Hyphenation on Demand. pages 1085–1091, Vancouver, August. The University of British Columbia.
>
> Petr Sojka. 1999b. Hyphenation on Demand. *TUGboat*, 20(3):241–247.
>
> This chapter contains the final version with minor corrections.

Hyphenation on Demand

Petr Sojka

Abstract: The need to fully automate the batch typesetting process has increased with the use of TEX as the engine for high-volume and on-the-fly typeset documents which, in turn, has lead to the need for programmable hyphenation and line--breaking of the highest quality.
An overview of approaches to building *custom* hyphenation patterns is provided, along with examples. A methodology of the process is given, combining different approaches: one based on morphology and hand-made patterns, and one based on word lists and the PATGEN program. The method is aimed at modular, easily maintainable, efficient, and portable hyphenation. The bag of tricks used in the process to develop custom hyphenation is described.

9.1 Motivation

In principle, whether to hyphenate or not is a style question and CSS [Cascading Style Sheets] should develop properties to control hyphenation. In practice, however, for most languages there is no algorithm or dictionary that gives all (and only) correct word breaks, so some help from the author may occasionally be needed. — Bos (1999)

Separation of content and presentation in today's open information management style in the sense of SGML/XML (Goldfarb, 1990; Megginson, 1998) is a challenge for TEX as a batch typesetting tool. The attempts to bring TEX's engine to untangle presentation problems in the WWW arena are numerous (Sutor and Díaz, 1998; Skoupý, 1998).

One bottleneck in the high-volume quality publishing is the proofreading stage — line-breaking and hyphenation handling that need to be tuned fine to the layout of a particular publication. Tight deadlines in paper-based document production and high-volume electronic publishing put additional demands for better automation of the typesetting process. The need for multiple presentations of the same data (e.g., for paper and screen) adds another dimension to the problem. Problems with hyphenation are often one of the most

difficult. As most TEX users are perfectionists, fixing and tuning hyphenation for every presentation is a tedious, time-consuming task.

Several issues related to hyphenation in TEX were discussed in (Sojka and Ševeček, 1995; Sojka, 1995). On the ground that we were involved in typesetting tens of thousands of TEX pages of multilingual documents (mostly dictionaries), we want to point out several methods suitable for the development of hyphenation patterns.

9.2 Pattern Generation

There is no place in the world that is linguistically homogeneous, despite the claims of the nationalists around the world. — Plaice (1998)

Liang (1983) in his thesis written under Knuth's supervision, developed a general method to solve the hyphenation problem that was adopted in TEX82 (Knuth, 1986, App. H). He wrote the PATGEN program (Liang and Breitenlohner, 1999), which takes

- a list of already hyphenated words (if any),
- a set of patterns (if any) that describes "rules",
- a list of parameters for the pattern generation process,
- a character code translation file — added in PATGEN 2.1; for details see Haralambous (1994),

and generates

- an enriched set of patterns that "covers" all hyphenation points in the given input word list,
- a word list hyphenated with the enriched set of patterns (optional).

The patterns are loaded into TEX's memory and stored in a data structure (cf. Knuth (1986a, parts 40 – 43)), which is also efficient for retrieval — a variant of *trie* memory (cf. Knuth (1998, pp. 492 – 512)). This data structure allows to search a hyphenation pattern in linear time with respect to the pattern length. The algorithm using a trie that "outputs" possible hyphenation positions may be formally viewed as the finite automaton with output (Mealy automaton or transducer).

9.3 Pattern Development

. . . problems [with hyphenation] have more or less disappeared,
and I've learnt that this is only because, nowadays, every *hyphenation*
in the newspaper is manually checked *by human proof-readers.*
— Jarnefors (1995)

Studying patterns that are available for various languages shows that PATGEN has only been used for about half of the hyphenation pattern files on CTAN (cf. Tables 6.1 and 6.2).

There are two approaches to hyphenation pattern development, depending on user preferences. Single authors using TEX as an authoring tool want to minimize system changes and want TEX to behave as a fixed point so that retypesetting of old articles is easily done, thanks to backwards compatibility. For such users, one set of patterns that is fixed once and for all might be sufficient.

On the other hand, for publishers and corporate users with high--volume output, it is more efficient to make a long-term investment into development of hyphenation patterns for particular purposes. I remember one TEX user saying that my suggestion to enhance standard hyphenation patterns with custom-made ones to allow better hyphenation of chemical formulæ would save his employer thousands of pounds per year. Of course, with this approach, one has to archive *full* sources for every publication, together with hyphenation patterns and exceptions.

One of the possible reasons PATGEN has not been used more extensively may be the high investment needed to create hyphenated lists of words, or better, a morphological database of a given language.

9.4 Pattern Bootstrapping and Iterative Development

The road to wisdom? Well it's plain and simple to express:
Err and err and err again but less and less and less. — Hein (1966)

When developing new patterns, it is good to iterate the following bootstrapping technique. It could avoid the tedious task of manually marking hyphenation points in huge lists of words:

1. write down the most obvious initial patterns, if any, and/or collect "the closest" ones (e.g., consonant-vowel rules),
2. extract a small word list for the given language,
3. hyphenate the current word list with current set patterns,
4. check all hyphenated words and correct them; in the case of errors return to step 3,

5. collect a bigger word list,
6. use the previously generated set of patterns to hyphenate the words in this bigger list,
7. check hyphenated words, and if there are no errors, move to step 9,
8. correct the word list and return to step 6,
9. generate final patterns with PATGEN with parameters fitted the particular purpose (tuned for space or yielding efficiency),
10. merge/combine new patterns with other modules of patterns to fit the particular publishing project.

To find an initial set of patterns, some basic rules of hyphenation in the specific language should be known. The language can be grouped into one of two categories: those that derive hyphenation points according to morphology (etymology) and those that derive hyphenation according to pronunciation — "syllable-based" (syllabic) hyphenation. For the first group of languages, one should start with patterns for most frequent prefixes and suffixes and endings. For syllable-based hyphenation, patterns based on sequences of consonants and vowels might be used (cf. Anonymous (1993, Section 6.44), and Haralambous (1999)) as first approximation of hyphenation patterns.

As using TEX itself for hyphenation of word lists and development of patterns may be preferred to other possibilities, we will start with this portable solution, using hyphenation of phonetic transcriptions as an example of a syllable-based "language".

Let us start with some plain TEX code to define consonant-vowel (CV) patterns:

```
% ... loading plain.tex
%     without hyphen.tex patterns ...
\patterns{cv1cv cv2c1c ccv1c cccv1c
ccccv1c cccccv1c v2v1 v2v2v1 v2v2v2v1
...
}
```

There is a way to typeset words together with their hyphenation points in TEX; the code from Olšák (1997, with minor modifications) looks like this:

```
\def\showhyphenspar{\begingroup
 \overfullrule=0pt \parindent0pt
 \hbadness=10000 \tt
 \def\par{\setparams\endgraf\composelines}%
 \setbox0=\vbox\bgroup
            \noindent\hskip0pt\relax}

\def\setparams{\leftskip=0pt
```

```
 \rightskip=0pt plus 1fil
 \linepenalty1000 \pretolerance=-1
 \hyphenpenalty=-10000}

\def\composelines{%
 \global\setbox1=\hbox{}%
 \loop
   \setbox0=\lastbox \unskip \unpenalty
   \ifhbox0 %
     \global\setbox1=\hbox{%
       \unhbox0\unskip\hskip0pt\unhbox1}%
 \repeat
 \egroup % close \setbox0=\vbox
 \exhyphenpenalty=10000%
 \emergencystretch=4em%
 \unhbox1\endgraf
 \endgroup}
```

Now, we will typeset our word list in the typewriter font without ligatures. To use the CV patterns defined above we need to map word characters properly:

```
% vowels maping
\lccode‘\a=‘v  \lccode‘\e=‘v
\lccode‘\i=‘v  \lccode‘\o=‘v
...
% consonants
\lccode‘\b=‘c  \lccode‘\c=‘c
\lccode‘\d=‘c  \lccode‘\f=‘c
...
\raggedbottom \nopagenumbers
\showhyphenspar
The need to fully automate the
batch typesetting process increases
with the use of word in wordlist
...
\par\bye
```

Finally, extracting hyphenated words from `dvi` the file via the `dvitype` program, we get our word list hyphenated by our simple CV patterns.

Another way to get the initial word list hyphenated is to use PATGEN with initial patterns and no new level, letting PATGEN hyphenate the word list that was input. PERL or RUBY addicts may want to use hyphenation modules by Pazdziora (2005) or Ziegler (2005) for the task.

Once the job of proofreading of the word list is finished, we can generate new patterns and collect other words in the language. Using new patterns on the new collection will show the efficiency of the process.

Fine tuning of patterns may be iterated, once PATGEN parameters are set, so that nearly 100% coverage of hyphenation points is achieved in every iteration. The setting of such PATGEN parameters may be difficult to find on the first attempt. Setting of these parameters is discussed in (Sojka and Ševeček, 1995).

9.5 Modularity of Patterns

It is tractable for some languages to create patterns by hand, simply by writing patterns according to the rules for a given language. This approach is, however, doomed to failure for complex languages with several levels of exceptions. Nevertheless, there are special cases in which we may build pattern modules and concatenate patterns to achieve special purpose behavior. This applies especially when additional characters (not handled when patterns have been built originally) may occur in words that we still want to hyphenate.

Patterns generated by Raichle (1997) may serve as an example that can be used with any fonts in standard LaTeX eight-bit T1 font encoding, to allow hyphenation after an explicit hyphen. Similar pattern modules can be written for words or chemical formulæ that contain braces and parentheses. These can be combined with "standard" patterns in the needed encodings. Some problems might be caused by the fact that TeX does not allow metrics to be defined for `\lefthyphenmin` and `\righthyphenmin` properly — we might want to say that ligatures, for instance, only counted as a single letter. We may want that some characters should not affect hyphenation at all (e.g., parentheses in words like *colo(u)r*). We must wait until some naming mechanisms for output glyphs (characters) is adopted by the TeX community for handling these issues.

Adding a new primitive for the hyphenmin code — let's call it `\hccode`, a calque on `\lccode` — would cause similar problems: changing it in the mid-paragraph would have unpredictable results.[1]

It is advisable to create modules or libraries of special-purpose hyphenation patterns, such as the ones mentioned above, to ease the task of pattern development. These patterns might be written in such a way to be easily adaptable for use with core patterns of a different language.

1. ε-TeX v2 has a new feature to fix the `\lccode` values during the pattern reading phase.

9.6 Common Patterns for More Languages

Having large hyphenated word lists of several languages the possibility then exists to make multilingual or special-purpose patterns from collections of words by using PATGEN. Joining word lists and generating patterns for particular publications on demand is especially useful when the word databases are structured and split into sublists of personal names, geographic names, abbreviations, etc. These patterns are requested when typesetting material in which language switching is not properly done (e.g., on the WWW).

Czech and Slovak are very closely related languages. Although they do not share exactly the same alphabet, rules for hyphenation are similar. That has led us to the idea of making one set of hyphenation patterns to work for both languages, saving space in a format file that supports both. In the Czech/Slovak standard TEX distribution there is support for different font encodings. For every encoding, hyphenation patterns have to be loaded as there is no character remapping on the level of trie possible. Such Czechoslovak patterns would save patterns for each encoding in use.

It should be mentioned that this approach cannot be taken for any set of languages as there may be, in general, identical words that hyphenate differently in different languages; thus, simply merging word lists to feed PATGEN is not sufficient without degrading the performance of patterns by forbidding hyphenation in these conflicting words (e.g., re-cord vs. rec-ord).

9.7 Phonetic Hyphenation

As an example of custom-made hyphenation patterns, the patterns for hyphenating phonetic (IPA) transcription are described in this section. Dictionaries use this extensively — see Figure 9.1 on the following page,[2] taken from Kirsteinová.

The steps used to develop the hyphenation patterns for this dictionary were similar to those described in Section 9.4 on page 73:

1. write down the most obvious (syllabic) patterns,
2. extract all phonetic words from available texts,
3. hyphenate this word list with the initial set of patterns,
4. check and correct all hyphenated words,
5. generate final quality patterns.

In bigger publishing projects efforts like this pay off very quickly.

2. IPA font used is TechPhonetic downloadable from `http://www.sil.org/ftp/PUB/SOFTWARE/WIN/FONTS/`.

akkompagnement *sb* [ækʌmpænjə-ˈmaŋ] *-et, -er* hudební doprovod *m*
alimentationsbidrag *sb* [ælimɛntæ-ˈšoːʼnsˌbiˌdraːʼw] *-et,* - alimenty *pl*, příspěvek *m* na výživné dítěte
befolknings‖eksplosion *sb* [beˈfʌlʼg-neŋs-] *-en, -er* populační exploze *f* ■ **-tilvækst** *-en, -er* přírůstek *m* obyvatelstva ■ **-tæthed** *-en, -er* hustota *f* obyvatelstva
bemærkelsesværdig *adj* [beˈmæʀ-gəlsəsˌvæʀʼdi] *-t, -e* pozoruhodný
beslutningsdygtig *adj* [beˈslud-neŋsˌdøgdi] *-t, -e* schopný rozhodovat; **den lovgivende forsamling var ~** zákonodárné shromáždění bylo schopné se usnášet

Figure 9.1: Example of phonetic hyphenation in (Kirsteinová and Borg, 1999).

9.8 Hyphenation for an Etymological Dictionary

In some publications as in (Rejzek, 2001), a different problem can arise: the possibility of having more than 256 characters used within a single paragraph. This problem cannot be, in general, easily solved[3] within the frame of TEX82. For this purpose we thus tried OMEGA, the typesetting system by Plaice and Haralambous (Plaice, 1994; Haralambous and Plaice, 1994; Plaice and Haralambous, 1996; Haralambous, 1996; Haralambous and Plaice, 1998). One has to create special virtual fonts (e.g., by using the fontinst package) on top of the OMEGA ones, in order to typeset it — see Figure 9.2 on the following page.

3. One could try to reencode all fonts used in parallel in some paragraph so that they share the same `\lccode` mappings, but this exercise would have to be done for each multilingual-intensive publication, again and again.

naivní ‘prostoduchý, dětinský’, *naivita, naivka.* Přes něm. *naiv* z fr. *nai:f* tv., původně ‘přirozený, opravdový’ z lat. *nātīvus* ‘přirozený’ od *nātus,* což je příč. trp. od *nāscī* ‘rodit se’. Srov. ↑*nacionále.*

náruživý ‘vášnivý, silně zaujatý’, *náruživost.* Jen č. Souvisí s č.st. *oruží* ‘zbroj, zbraň, náčiní’ (všesl.). Psl. kořen **-rǫg- (B1, B7)* nejspíš souvisí s lit. *įrangùs* ‘prudký’ (HK), *rángtis* ‘spěchat’, *ren~gtis* ‘chystat se’, *rangà* ‘přístroj, nástroj’, dále asi se stŕhn. *ranc* ‘rychlý, vířivý pohyb’, něm. *renken* ‘pohybovat se krouživě sem tam’, angl. *wrench* ‘trhnout, vykroutit’, vše by to bylo od ie. **u̯reng-* ‘kroutit, ohýbat’ a vzdáleně příbuzné by bylo ↓*vrhat.*

nebozez ‘vrták na dřevo’, *nebozízek.* Hl. *njeboz,* sln. *nabôzec.* Přejato z germ., forma by ukazovala až na germ. **naba--gaiza* před změnou *-z->-r- (A5, B1),* která už je provedena v sthn. *nabagēr* (něm. *Näber* tv.). První část germ. slova odpovídá něm. *Nabe* (viz ↑*náboj*), druhá něm.st. *Ger* z gót. **gaiza-* ‘něco špičatého’.

Figure 9.2: Using OMEGA to typeset paragraphs in which words from languages with more than 256 different characters may appear and be hyphenated in parallel.

9.9 More Hyphenation Classes

But at least I can point out a minor weakness of TEX's algorithm: all possible hyphenations have the same penalty. This might be ok for English, but for languages like German that have a lot of composite words there should be the ability to assign lower penalties between parts of a composite i.e. Um-brechen should be favored against Umbre-chen. — Hars (1999)

Some suggestions on handling multiple hyphenation classes were suggested by Sojka (1995). A prototype implementation of ε-TEX and PATGEN has recently been done (Classen, 1998). For a wider adoption of such improvements

availability of large word lists and development of new patterns is crucial. Many of the methods mentioned above could be used to develop such multi-class/multi-purpose patterns. (Allen, 1990) contains such a word list, which shows that some publishers do pay attention to line-breaking details.

9.10 Speed Considerations

Even though hyphenation searches using a trie data structure are fast, searching for unnecessary hyphenation points is a waste of time. It is advisable to tell TEX where words shouldn't be hyphenated. Comparing several possibilities for suppressing hyphenation, the option of setting `\lefthyphenmin` to 65 is slightly faster than switching to `\language`, which has no patterns. These solutions outperform the `\hyphenpenalty 10000` solution by a fair amount (Arsenau, 1994).

9.11 Reuse of Patterns

Sometimes we need the same patterns with different `\lefthyphenmin` and `\righthyphenmin` parameters. The suggested approach is not to limit hyphens close to word boundaries during the pattern generation phase but to use TEX's `\setlanguage` primitive. This can be done to achieve special hyphenation handling for the last word in a paragraph (e.g., a higher `\righthyphenmin`) given proper markup by a preprocessing filter. For example:

```
\newcount\tmpcount
\def\lastwordinpar#1{%
\tmpcount=\righthyphenmin \righthyphenmin=5
\setlanguage\language #1
\expandafter\righthyphenmin\the\tmpcount
\setlanguage\language}

\showhyphens{demand}
\lastwordinpar{demand\showhyphens{demand}}
\bye
```

9.12 Future Work

If you find that you're spending almost all your time on practice, start turning some attention to theoretical things; it will improve your practice.
— Knuth (1989b)

It seems inevitable that embedding of language-specific support modules will be necessary for *the* typesetting system in the future. These demands must not only be applied to hyphenation but also to spelling or even grammar checkers. As even people using WYSIWYG systems may use tools that help to visualize possible typos (in color, etc.) on the fly, the computing power of today's machines is surely sufficient to do the same in batch processing with even better results.

The idea of using patterns to capture mappings specific for particular languages or dialect modules can be further generalized for different purposes and mappings. The use of the theory of finite-state transducers (Mohri, 1996; Mohri, 1997; Roche and Schabes, 1997; Roche and Schabes, 1995) to implement other classes of language modules looks promising.

Applicability of outlined pattern-driven approach remains to be shown in other areas of NLP and LE e.g., for syllabification, a word sense or semantic disambiguation, simply speaking, in any area based on finite-state models of any kind.

References

R. E. Allen. 1990. *The Oxford Spelling Dictionary*, volume II of *The Oxford Library of English Usage*. Oxford University Press.

Donald Arsenau. 1994. Benchmarking paragraphs without hyphenation. Posting to the Usenet group `news:comp.text.tex` on December 13.

Matthias Classen. 1998. An extension of TEX's hyphenation algorithm. `ftp://peano.mathematik.uni-freiburg.de/pub/etex/hyphenation/`.

Charles F. Goldfarb. 1990. *The SGML handbook*. Clarendon Press, Oxford.

Yannis Haralambous and John Plaice. 1994. First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer. *TUGboat*, 15(3):344–352, September.

Yannis Haralambous and John Plaice. 1998. The Design and Use of a Multiple-Alphabet Font with Ω. In Roger D. Hersch, Jacques André, and Heather Brown, editors, *Lecture Notes in Computer Science 1375*, pages 126–137. Springer-Verlag, April.

Yannis Haralambous. 1996. ΩTimes and ΩHelvetica fonts under development: Step One. *TUGboat*, 17(2):126–146, June.

Blanka Kirsteinová and Blanka Borg. 1999. *Dánsko-český slovník, Dansk-Tjekkisk Ordbog [Danish-Czech Dictionary]*. LEDA, Prague, Czech Republic.

Donald E. Knuth. 1986. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Franklin M. Liang and Peter Breitenlohner. 1999. PATtern GENeration program for the TEX82 hyphenator. Electronic documentation of PATGEN program version 2.3 from web2c distribution on CTAN.

David Megginson. 1998. *Structuring XML Documents*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Mehryar Mohri. 1996. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2:61–80. Originally appeared in 1994 as Technical Report, Institut Gaspard Monge, Paris.

Mehryar Mohri. 1997. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, June.

John Plaice and Yannis Haralambous. 1996. The latest developments in Ω. *TUGboat*, 17(2):181–183, June.

John Plaice. 1994. Progress in the Omega project. *TUGboat*, 15(3):320–324, September.

Jan Rejzek. 2001. *Etymologický slovník českého jazyka (Ethymological Dictionary of Czech Language)*. LEDA, Prague, Czech Republic.

Emmanuel Roche and Yves Schabes. 1995. Deterministic Part-of-Speech Tagging. *Computational Linguistics*, 21(2):227–253.

Emmanuel Roche and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press.

Karel Skoupý. 1998. NTS: a New Typesetting System. *TUGboat*, 18(3):318–322.

Petr Sojka and Pavel Ševeček. 1995. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Robert S. Sutor and Angel L. Díaz. 1998. IBM techplorer: Scientific Publishing for the Internet. *Cahiers GUTenberg*, (28–29):295–308, March.

Chapter 10

Competing Patterns for Language Engineering

Methodology for handling and storing language data is described in a paper presented at the TSD 2000 conference in Brno (Sojka, 2000):

> Petr Sojka. 2000. Competing Patterns for Language Engineering. In Petr Sojka, Ivan Kopeček, and Karel Pala, editors, *Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000*, Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, pages 157–162, Brno, Czech Republic, September. Springer-Verlag.
>
> This chapter contains slightly modified version of the above paper.

Competing Patterns for Language Engineering

Petr Sojka

Abstract: In this paper we describe a method of effective handling of linguistic data by means of a *covering and inhibiting patterns* — patterns that "compete" with each other. A methodology of developing such patterns is outlined. Applications in the areas of morphology, hyphenation and part-of-speech tagging are shown. This pattern-driven approach to language engineering allows the combination of linguist expertise with the data learned from corpora — layering of knowledge. Searching for information in pattern database (dictionary problem) is blindingly fast — linear with respect to the length of searching word as with other finite-state approaches.

We anticipate that experimental computer science will continue and develop, and that, through experimentation, we will enjoy a renaissance in interaction of theory and practice.
— Introduction to special issue of TCS (Salomaa et al., 2000, page 3)

10.1 Introduction

There is a need to store empirical language data in almost all areas on natural language engineering (LE). Finite-state methods (Roche and Schabes, 1997; Kornai, 1999; Mohri, 1996; Mohri, 1997; Karttunen et al., 1996) have found their revival in the last decade. The theory of finite-state automata (FSA) and transducers (FST) is a well developed part of theoretical computer science (for an overview, see e.g. (Gruska, 1997; Büchi, 1989)). As the finite-state machines (FSM) tend to grow with increased demand for quality of language processing, more attention is being given to the efficiency of the handling of FSM (Mohri, 2000; Câmpeanu et al., 1998). The size of some FSM used in natural language processing exceeds ten million states (e.g., weighted finite automata and transducers for speech recognition). The practical need to reduce the size of these data structures without losing their expressiveness and excellent time complexity of operations on them is driving new research activities — trend of experimental Computer Science is seen. Several FSM-based software tools (Mohri et al., 1998; Silberztein, 2000; Watson, 1999; Hobbs et al., 1997) have already been implemented for LE.

In this paper we explore a method of FSM decomposition that allows a significant size reduction of FSM — the idea of storing empirical data using *competing patterns*. The data structure *trie* and pattern technique have been developed by Liang (1983) for hyphenation algorithm in TEX (Knuth, 1986, Appendix H). We defined several kinds of patterns and our extensive experiments showed that the method is applicable in several areas of language engineering. *Bootstrapping* and *stratification* techniques allow us to speed up the development of such machines — space savings and time of development savings are enormous.

This paper is organized as follows. In Section 10.2, we give basic definitions and a short overview of known results. Section 10.3 on page 87 discusses pattern development methodology. Section 10.4 on page 87 describes in detail applications in the area of Czech morphology. An overview of results for hyphenation and compound word division is given in Section 10.5 on page 88. Possible applications like part-of-speech (POS) tagging are described in Section 10.6 on page 88.

10.2 Patterns

We start by formally introducing different kinds of patterns and basic notions (for a detailed discussion and examples, see e.g. (Büchi, 1989; Jiang et al., 1995)).

Definition 10.1 (pattern). Let us have two disjoint alphabets Σ (the alphabet of *terminals*) and V (the alphabet of *variables*). *Patterns* are words over the free monoid $\langle(\Sigma \cup V)^*, \varepsilon, \cdot\rangle$. Length of the empty word ε, $|\varepsilon|$, is zero. Patterns consisting of terminals only called *terminal patterns*. The *language* $L(\alpha)$ *defined by pattern* α consists of all words obtained from α by leaving the terminals unchanged and substituting a terminal word for each variable v. The substitution in our case has to be *uniform*: different occurences of v are replaced by the same terminal word. If the substitution replaces variables always by *nonempty* word, such language L_{NE} is *non-erasing*, and such pattern is called *NE-pattern*. Similarly, we define *erasing* language L_E as a language generated by *E-pattern* such that substitution of variable v by empty word ε is allowed.

As an example of E-pattern may serve pattern *SVOMPT* for English sentences where the variables denote Subject, Verb, Object, Manner, Place, Time. An obvious useful task is to infer a pattern common to all input words in a given sample by the process of *inductive inference*. It has been shown in (Jiang et al., 1995) that *inclusion problem* is undecidable for both erasing and non-erasing pattern languages. It is easy to show that the decidability

```
   h y p h e n a t i o n
l1          1n a
l1              1t i o
l2           n2a t
l2                2i o
l2       h e2n
l3.h y3p h
l4       h e n a4
l5       h e n5a t
  .h0y3p0h0e2n5a4t2i0o0n.
   h y-p h e n-a t i o n
```

In this example $\langle A, \leq \rangle$ is $\mathbb{N}$ (natural numbers). There are 5 pattern levels — `l1`...`l5`. Patterns in odd levels are *covering*, in an even levels *inhibiting*. Winner pattern is `.h0y3p0h0e2n5a4t2i0o0n.`. Pattern `h e n5a t` wins over `n2a t`, thus hyphenation is possible.

Figure 10.1: Competing patterns and pattern levels.

of *equivalence problem* for non-erasing languages is trivial. The decidability status of the equivalence problem for E-patterns remains open. These results show that trying to infer language description in the form of set of patterns (or the whole grammar) automatically is very difficult task. It has been shown that decomposition of the problem by using *local grammars* (Gross, 1997) or building cascades of FSM (Hobbs et al., 1997) is a tractable, but very time-consuming task. Methods for the induction of patterns (from corpora) are needed.

Definition 10.2 (classifying pattern). Let $\langle A, \leq \rangle$ be a partially ordered system, $\leq$ be a *lattice order* (every finite non-empty subset of A has lower and upper bound). Let . be a distinguished symbol in $\Sigma' = \Sigma \cup \{.\}$ that denotes the beginning and the end of word — *begin of word marker* and *end of word marker*. *Classifying patterns* are the words over $\Sigma' \cup V \cup A$ such that dot symbol is allowed at the beginning or end of patterns.

Terminal patterns are "context-free" in that, they apply anywhere in the classified word — dot symbol in a pattern specifies pattern at the beginning and end of a word. Classifying patterns allows us to build *tagging hierarchies* on patterns.

Definition 10.3 (word classification, competing word patterns). Let P be a set of patterns over $\Sigma' \cup V \cup A$ (*competing patterns*, *pattern base*). Let $w = w_1 w_2 \ldots w_n$ be a word to classify with P. Classification $\mathit{classify}(w, P) = a_0 w_1 a_1 w_1 \ldots w_n a_n$ of w with respect to P is computed from pattern base P by competition. All patterns whose projection to Σ match a substring of w are collected. a_i is supremum of all values between characters w_i and w_{i+1} in matched patterns. $\mathit{classify}(w, P)$ is also called a *winning pattern*.

An example of competing patterns is shown in Figure 10.1. Competing pattern sets can be used on all levels of natural language processing — covering structures used in morphology, their exploration is seen on both

syntax (parsing) and semantic (WSD) levels. Competing patterns extend the power of FST somewhat like adding the complement operator with respect to A. Ideally, instead of storing full FST, we make patterns that embody the same information in an even more compact manner. Collecting patterns matching a given word can be done in linear time, using a trie data structure for pattern storage.

10.3 Methodology

10.3.1 Pattern Generation

There are several pattern generation strategies that allow the choice between size-optimal or coverage-optimal patterns (Sojka and Ševeček, 1995) with the PATGEN program (Liang and Breitenlohner, 1999). A generation process can be parametrised by several parameters whose tuning strategies are beyond the scope of this paper; see (Sojka and Ševeček, 1995; Sojka, 1995) for details. Parameters could be tuned so that virtually all hyphenation points are covered, leading to about 99.9 % efficiency, and size is not far from optimum. Further investigation and research is necessary to find sufficient conditions for finding optimal results.

10.3.2 Stratification Technique

As word lists from which patterns are generated are rather big (5,000,000 for Czech morphology or hyphenation, even more for other tasks such as POS tagging), they may be stratified. Stratification means that from 'equivalent' words only one or a small number of representatives are chosen for the pattern generation process.

10.3.3 Bootstrapping Technique

Developing patterns is usually an iterative process. One starts with hand-written patterns, uses them on input word list, sees the results, makes the correction, generates new patterns, etc. This technique succeeded in accelerating the pattern development process by the order of magnitude. We usually do not start from scratch, but use some previously collected data (e.g., word list).

10.4 Application to Czech Morphology

For the information extraction, information retrieval systems, indexing and similar tasks we need information on many kinds of sub-word divisions:

dividing a word into atoms (cutting of prefixes, compound words recognition etc.) (Kodydek, 2000). We have created several competing pattern sets using the word database of Czech morphological analyser `ajka` (Sedláček, 1999). We have taken a word list of 564,974 Czech words (6.6 MB) with marked prefix segmentation and added 51,816 similar ones (starting with the same letters, but morphologically different). We were able to build patterns that were able to perform prefix segmentation on 100% of words of our input word list and 98% of words in our test set.

In comparison to naïve storage of word segmentation, there is a compression ratio of several order of magnitude higher. Even compared to storage of FSM using suffix compression in a trie, patterns compacted in trie data structure gives a tenfold space reduction, still with linear access time.

10.5 Application to Hyphenation and Compound Words

We have used the pattern technique to cover Czech and German hyphenation points and compound word borders. From a Czech word list (372,562 words, approx. 4 MB), we were able to create 8,239 patterns (40 kB) that cover 99.63% hyphenation points. From a German word list (368,152 words, 5 MB), we were able to create 4,702 patterns (25.2 kB) that cover 98.37% hyphenation points.

Covering compound word hyphenation was more difficult, as longer patterns are needed. With slightly different parameters of pattern generation, we were able to create patterns for German compound words with 8,825 patterns (70.2 kB) with 95.28% coverage. Higher coverage is at the expense of pattern size growth.

For details of hyphenation pattern generation for compound words in Czech and German using PATGEN, see (Sojka and Ševeček, 1995; Sojka, 1995; Sojka, 1999).

10.6 Outline of an Application to Part-of-Speech Tagging

Two mainstream approaches are being used for the POS task: *linguistic*, based on hand-coded linguistic rules (constraint grammars) (Karlsson et al., 1995; Oliva et al., 2000) and *machine learning* (statistical, transformation-based) approaches, based on learning the language model from corpora. Their combination is probably what is needed — the 'built-in' linguistic knowledge should be communicated to and take preference over, for example, statistical knowledge acquired during learning. We hope that ordered and competing patterns will be a viable unifying carrier of information that will allow combination of both approaches.

Finite-state cascaded methods have already been applied to the POS task (Abney, 1997). Let us outline one possible approach. Given sentence $w_1\ w_2 \ldots\ w_n$, an ambiguous tagger gives various possible tags: $p_{11} \ldots\ p_{1a_1}$ for the first word, $p_{21} \ldots\ p_{2a_2}$ for the second, etc. Writing output as $(p_{11} \ldots p_{1a_1})(p_{11} \ldots\ p_{2a_2}) \ldots (p_{n1} \ldots\ p_{na_n})$, the task is to choose the right POS p_{ij} for every w_i. Taking the tag set from Brown corpus (the Brown University Standard Corpus of Present-day American English) (Francis and Kučera, 1982) for the sentence "The representative put chairs on the table.", we get the output

```
. AT  ( NN - JJ )    ( NN  VBD - ) ( NNS - VBZ ) IN AT  NN    .
  The representative put           chairs        on the table .
```

Hyphenation markers immediately after POS tag show good solutions for training. Such 'word lists' (for each sentence from training corpus we get one 'hyphenated word') are used by PATGEN for disambiguation patterns generation. Sentence borders are explicitly coded. This or similar notation can be used for both formulation of ambiguous tagging decision strategies of variable context length by linguists. In comparison to classical constraint grammars, our experience shows that the obligation to write only rules which are true in any context is very difficult and only a few linguists are able to do so. Having pattern/rule levels/hierarchy helps to develop disambiguation strategies more easily and quickly. Generalisation for patterns over tree hierarchies is worked on.

10.7 Conclusion

We have shown effective methods for empirical language data storage and handling by means of competing patterns. Our *pattern-driven approach* to language engineering has been tested in several areas — hyphenation and morphology using prototype solution — programs PATGEN and TEX and their algorithms and data structures were used. Searching the pattern database is blindingly fast (linear with respect to the text length). It remains to be shown, that this approach is applicable and useful in areas as phonology, syllabification, speech segmentation, word sense or semantic disambiguation and speech processing. We believe that the pattern-driven approach will be explored in NLP systems for various classification tasks soon.

Acknowledgement

The author would like to thank reviewers for their suggestions to improve wording of the paper, and Radek Sedláček for providing a prefixed word list for experiments described in Section 10.4 on page 87.

References

Steven Paul Abney. 1997. Part-of-Speech Tagging and Partial Parsing. pages 118–136, Dordrecht. Kluwer Academic Publishers Group.

J. Richard Büchi. 1989. *Towards a Theory of Formal Expressions*. Springer-Verlag, New York, U.S.A.

Cezar Câmpeanu, Nicolae Sânteau, and Sheng Yu. 1998. Minimal cover-automata for finite languages. In Jean-Marc Champarnaud, Denis Maurel, and Djelloul Ziadi, editors, *Lecture Notes in Computer Science 1660*, pages 43–56, Berlin, Heidelberg. Springer-Verlag.

Nelson W. Francis and Henry Kučera. 1982. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin.

Maurice Gross. 1997. The Construction of Local Grammars. (Roche and Schabes, 1997), pages 329–354.

Jozef Gruska. 1997. *Foundations of Computing*. International Thomson Computer Press.

Jerry R. Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. 1997. FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. (Roche and Schabes, 1997), pages 383–406.

Tao Jiang, Arto Salomaa, Kai Salomaa, and Sheng Yu. 1995. Decision problems for patterns. *Journal of Computer and Systems Sciences*, 50(1):53–63.

Fred Karlsson, A. Voutilainen, J. Heikkilä, and A. Antilla. 1995. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin.

Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. 1996. Regular Expressions for Language Engineering. *Natural Language Engineering*, 2(4):305–328.

Donald E. Knuth. 1986. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Gabriele Kodydek. 2000. A Word Analysis System for German Hyphenation, Full Text Search, and Spell Checking, with Regard to the Latest Reform of German Orthography. In Sojka et al. (Sojka et al., 2000), pages 39–44.

András Kornai. 1999. *Extended Finite State Models of Language*. Cambridge University Press.

Franklin M. Liang and Peter Breitenlohner. 1999. `PATtern GENeration` program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.3 from web2c distribution on CTAN.

Mehryar Mohri, Fernando C. N. Pereira, and Michael D. Riley. 1998. FSM Library — General-purpose finite-state machine software tools. `http://www.research.att.com/sw/tools/fsm/`.

Mehryar Mohri. 1996. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2:61–80.

Originally appeared in 1994 as Technical Report, Institut Gaspard Monge, Paris.

Mehryar Mohri. 1997. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, June.

Mehryar Mohri. 2000. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, March.

Karel Oliva, Milena Hnátková, Vladimír Petkevič, and Pavel Květoň. 2000. The Linguistic Basis of a Rule-Based Tagger of Czech. In Sojka et al. (Sojka et al., 2000), pages 3–8.

Emmanuel Roche and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press.

Kai Salomaa, Derick Wood, and Sheng Yu, editors. 2000. *Implementing Automata*, volume 231.

Radek Sedláček. 1999. Morphological Analyzer of Czech (in Czech). Master's thesis, Masaryk University in Brno, April.

Max Silberztein. 2000. INTEX: an FST toolbox. *Theoretical Computer Science*, 234:33–46.

Petr Sojka and Pavel Ševeček. 1995. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

Petr Sojka, Ivan Kopeček, and Karel Pala, editors. 2000. *Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000*, Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, Brno, Czech Republic, September. Springer-Verlag.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Petr Sojka. 1999. Hyphenation on Demand. *TUGboat*, 20(3):241–247.

Bruce W. Watson. 1999. Implementing and using finite automata toolkits. (Kornai, 1999), pages 19–36.

Chapter 11

Pattern Generation Revisited

Problems of machine learning of competing patterns are tackled in this chapter. The method of generating competing patterns are revisited.

The results were presented at the EuroTEX 2001 conference in Kerkrade (Antoš and Sojka, 2001):

> David Antoš and Petr Sojka. 2001. Pattern Generation Revisited. In Simon Pepping, editor, *Proceedings of the 16th European TEX Conference, Kerkrade, 2001*, pages 7–17, Kerkrade, The Netherlands, September. NTG.

This chapter contains a slightly modified version of the above paper. The present author's proportion on the presented results is 50%.

Pattern Generation Revisited

David Antoš, Petr Sojka

Abstract: The PATGEN program, being nearly twenty years old, doesn't suit today's needs:

- it is nearly impossible to make changes, as the program is highly optimized (like TEX),
- it is limited to eight-bit encodings,
- it uses static data structures,
- reuse of the pattern technique and packed trie data structure for problems other than hyphenation (context dependent ligature handling, spell checking, Thai syllabification, etc) is cumbersome.

Those and other reasons explained further in the paper led us to the decision to reimplement PATGEN from scratch in an object-oriented manner (like NTS–New Typesetting System reimplementation of TEX) and to create the PATtern LIBrary PATLIB and the (hyphenation) pattern generator based on it.

We argue that this general approach allows the code to be used in many applications in computer typesetting area, in addition to those of pattern recognition, which include various natural language processing, optical character recognition, and others.

Key Words: patterns • UNICODE • hyphenation • tagging • transformation • OMEGA • PATGEN • PATLIB • reimplementation • templates • C++

11.1 Introduction

The ultimate goal of mathematics is to eliminate all need for intelligent thought.
— Graham, Knuth, Patashnik (Graham et al., 1989, page 56)

The ultimate goal of a typesetting engine is to automate as much as possible of what is needed for a given design, allowing the author to concentrate on the content of the text. The author maps her/his thoughts in *linear* writing, a sequence of *symbols*. Symbols (characters, words or even sentences) can be combined into *patterns* (of characters, words or sentences). Patterns describe "higher rules" and dependencies between symbols, depending on *context*.

The technique of covering and inhibiting patterns used in the PATGEN program (Liang and Breitenlohner, 1999) is highly effective and powerful. The

pattern technique is an effective way to extract information out of large data files and to recognize the structures again. It is used in TEX as an elegant and language-independent solution for high-quality word hyphenation. This effective approach found its place in many other typesetting systems including the commercial ones. We think this method should be studied well, as many other applications are possible, in addition to those in the field of typesetting and natural language processing.

The generation of hyphenation patterns using the PATGEN program does not satisfy today's needs. Many generalizations are needed for wider use. The OMEGA system (Haralambous and Plaice, 1997; Plaice and Haralambous, 1996) was introduced. One of its goals is to make direct typesetting of texts in UNICODE possible, hence enabling the hyphenation of languages with more than 256 characters. An example of such a language is Greek, where 345 different combinations of Greek letters with accents, breathings, syllable lengths and the subscript iota are needed (Haralambous and Plaice, 1994). Therefore, OMEGA needs a generator capable of handling general/universal hyphenation patterns. Those new possibilities and needs in computer typesetting, together with the detailed analysis described below, led us to revise the usage of pattern recognition and to design new software to meet these goals.

The organization of the paper is as follows. The next section (page 95) defines the patterns, using a standard example of hyphenation. Then an overview (Section 11.3 on page 96) of the process of pattern generation is given. The following Section 11.4 on page 97 describes one possible use for patterns and is followed by a Section 11.5 on page 98, in which the limitations for exploiting the current version of PATGEN are argued.

The second part of this paper starts with a Section 11.6 on page 98 which describes the design of the new software library for pattern handling. Then packed digital trees, the basic data structure used in PATLIB, are presented (Section 11.7 on page 99). Some thoughts about implementing the translation/tagging process using pattern based techniques are summarized in the Section 11.8 on page 101. The final Section 11.9 on page 103 contains a summary and suggestions for future work.

11.2 Patterns

> *Middle English* patron *'something serving as a model', from Old French. The change in sense is from the idea of a patron giving an example to be copied. Metathesis in the second syllable occurred in the 16th century. By 1700* patron *ceased to be used on things, and the two forms became differentiated in sense.*
> — Origin of word *pattern*: (Hanks, 1998)

Patterns are used to recognize "points of interest" in data. A point of interest may be the inter-character position where hyphenation is allowed, or the border between water and forest on a landscape photograph, or something similar. The pattern is a sub-word of a given word set and the information of the points of interest is written between its symbols.

There are two possible values for this information. One value indicates the point of interest *is* here, the other indicates the point of interest *is not* here. Natural numbers are the typical representation of that knowledge; odd for yes, even for no. So we have *covering* and *inhibiting* patterns. Special symbols are often used, for example a dot for the word boundary.

Below we show a couple of hyphenation patterns, chosen out of the English hyphenating pattern file. For the purpose of the following example, we deal with a small subset of the real set of patterns. Note that the dot represents a word boundary.

```
.li4g    .lig5a    3ture    1ga    2gam
```

Using the patterns goes as follows. All patterns matching any sub-word of the word to be hyphenated are selected. Using the above subset of patterns with the word "ligature" we get:

```
. l i g a t u r e .
. l i4g
. l i g5a
        3t u r e
    1g a
```

The pattern `2gam` matches no sub-word of "ligature". The patterns *compete* and the endresult is the maximum for inter-character positions of all matching patterns, in our example we get:

```
. l0i4g5a3t0u0r0e .
```

According to the above we may hyphenate `lig-a-ture`.

To sum up: with a "clever" set of patterns, we are able to store a mapping from sequences of tokens (words) to an output domain — sequence of boolean values —, in our case positions of hyphenation points. To put it in another way: tokens (characters) emit output, depending on the context.

For a detailed introduction to TEX's hyphenation algorithms see (Knuth, 1986, Appendix H). We now need to know how patterns are generated to understand why things are done this way.

11.3 Pattern Generation

An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil's behavior.
— Turing (1950)

We want to cover as much hyphenation points as possible, with no errors, and generate a *minimal* set of competing patterns. Even when giving up the minimality requirement, we may get surprisingly good results compressing the *input data* information into a pattern set iteratively. Let us now describe the generating process.

We need a large input data file with marked points of interest. Hyphenating words, we use a large dictionary with allowed hyphenation points. Now we repeat going through the data file in several *levels*. We generate covering patterns in odd levels and inhibiting ones in even levels.

We have a rule how to choose *pattern candidates* at each level. In our case it may be "an at most k characters long substring of the word containing the hyphenation point". We choose pattern candidates and store them into a suitable data structure. Not all candidates are good patterns, so we need a *pattern choosing rule*. Usually we remember the number of times when the candidate helps and spoils finding a correct hyphenation point. We always test new candidates according to all patterns selected so-far. We are interested in the functionality of the whole set. The pattern choosing rule may be a linear function over the number of good/bad word efficiency of the candidate compared to a threshold. This heuristic is used in PATGEN, but other heuristics may lead to better (e.g., with respect to space) pattern sets with the same functionality. The candidates marked as good by the previous process are included into the set of patterns. The pattern set still makes mistakes. We continue generating another level, an even level this time, when we create inhibiting patterns. The next level will be covering and so on. A candidate at a certain level is good if it repairs errors made by previous levels.

This is also the way how PATGEN works. A PATGEN user has no chance to change the candidate and/or pattern choosing rules, which are similar to the ones previously described. Hyphenating patterns for TEX were created for several dozens of languages (Sojka and Ševeček, 1995), usually created from a list with already hyphenated words. There are languages where the patterns were created by hand, either entirely, or in part.

How successful is this technique? The natural language dictionary has several megabytes of data. Out of such dictionary patterns having only tens of kilobytes may be prepared, covering more than 98% of the hyphenation points with an error rate of less than 0.1%. Experiments show that four or five levels are enough to reach those parameters. By using various strategies of setting linear threshold parameters we may optimize the patterns to the size, the covering ratio and/or errors (Sojka, 1995). As not many lists with hyphenated words are publicly available for a serious research on pattern generation heuristics, we think that most available patterns are suboptimal. For more information on pattern generation using PATGEN have a look at tutorial (Haralambous, 1994).

11.4 Tagging with Patterns

The solution of the hyphenation problem and the techniques involved were studied extensively (Sojka and Ševeček, 1995) and together with long-lasting usage in TEX and other typesetting systems, their advantages have been verified. The application of the techniques of bootstrapping and stratification (Sojka, 1995; Sojka, 1999) made them even more attractive. However, to the best of our knowledge, nobody has so far suggested and used a context dependent task for the resolution of other context dependent tasks.

We may look at the hyphenation problem as at the problem of *tagging* the possible hyphenation positions in finite sequences of characters called words. On a different level of abstraction, the recognition of sentence borders is nothing more than "tagging" the beginnings and ends of sentences in sequences of words.

Yet another example: in high-quality typography, it is often necessary to decide, whether a given sequence of characters is to be typeset as a ligature (such as ij, fi, fl) and not as separate characters (ij, fi, fl). This ambiguity has to be resolved by tagging of appropriate occurrences, depending on the context: ligatures are, e.g., forbidden on compound word boundaries.

All these tasks (and many others, see page 101) are "isomorphic" — the same techniques developed and used for hyphenation may be used here as well. The key issue in applicability of the techniques for the variety of context-dependent tagging tasks is understanding and effective implementation of the pattern generation process. The current implementation of PATGEN is not up to these possible new uses.

11.5 PATGEN Limitations

What man wants is simply independent choice,
whatever that independence may cost and wherever it may lead.
— Fedor Dostoevsky, *Notes from Underground* (1864)

The PATGEN program has several serious restrictions. It is a monolithic structured code, which, although very well documented (documented PASCAL, WEB), is very difficult to change. PATGEN is also "too optimized", necessary to make it possible to run in the core of the PDP-10, so understanding the code is not easy. In this sense PATGEN is very similar to TEX itself. The data structures are tightly bound to the stored information: high-level operations are performed on the data structures directly without any levels of abstraction.

The data structures of PATGEN are hardwired for eight-bit characters. Modification to handle more characters — full UNICODE — is not straightforward. The maximum number of PATGEN levels is nine. When generating patterns, you can collect candidates of the same length at the same time only. The data structures are static, running out of memory requires the user to change constants in the source code and recompile the program.

Of course PATGEN may be used to generate patterns on other phenomenons besides word hyphenation, but only if you transform the problem into hyphenation. This might be non-trivial and moreover, it's feasible only for problems with small alphabets, fewer than approximately 240 symbols (PATGEN uses some ASCII characters for special and output purposes).

11.6 PATLIB

My library was dukedom large enough.
— Shakespeare, The Tempest (1611), act 1, sc. 2, l. 109

We decided to generalize PATGEN and to implement the PATtern LIBrary PATLIB for general pattern manipulation. We hope that this will make the techniques easily accessible. A UNICODE word hyphenation pattern generator is the testbed application.

For portability and efficiency reasons we chose C++ as the implementation language. The C++ code is embedded in CWEB to keep the code documented as much as possible. Moreover, the code "patterns" called *templates* in C++ allow us to postpone the precise type specification to higher levels of development which turned out to be a big advantage during the step-wise analysis. We do hope that templates increase flexibility of the library.

The PATLIB library consists of two levels, the finite language store (which is a finite automaton with output restricted to finite languages, im-

plemented using packed trie) and the pattern manipulator. The language store handles only basic operations over words, such as inserting, deleting, getting iteratively the whole stored language and similar low-level operations. The output of a word is an object in general, so is the input alphabet.

The pattern manipulator handles patterns, it means words with multiple positioned outputs. We also prepared a mechanism to handle numbers of good and bad counts for pattern candidates.

The manipulator and the language store work with objects in general, nevertheless to keep efficiency reasonable we suggest to use numbers as internal representation for the external alphabet. Even if the external alphabet is UNICODE, not all UNICODE characters are really used in one input data file. So we can collect the set of all used characters and build a bijection between the alphabet and the internal representation by numbers $\{1, \ldots, n\}$, where all the numbers are really used.

We separated the semantics from the representation. We don't have to care what the application symbols are. An application using this library may implement any strategy for the generation of patterns.

Of course, we have to pay for more generality and flexibility with performance loss. As an example, the output of a pattern in PATGEN is a pointer to a hash table containing pairs $\langle \text{level\_number}, \text{position} \rangle$, we must have an object with a copy constructor.

11.7 Packed digital tree (trie)

The trie data structure we use to store patterns is well known — the idea of representing a family of strings by an abstract concept of a trie is credited to Axel Thue and his paper *Skrifter ugivne af Videnskabs-Selkabet i Christiania* from 1912 published in Mathematisk-Naturvidenskabelig Klasse, No. 1. Its practically usable variant — being described only seldom in programming books — is much less known.

A trie is usually presented and described as in (Knuth, 1998) as an m-ary tree. Its nodes are m-ary vectors indexed by a sorted alphabet. A node in depth l from the root corresponds to the prefix of length l. Finding a word in a trie starts at the root node. We take the next symbol of the word, let it be k. Then the k-th member of the node points to the lower level node, which corresponds to the unread rest of the word. If the word is not in the trie, we get the longest prefix.

Trie containing the words ba, bb, bbb, and da over the alphabet $\{a, b, c, d\}$ is shown in Figure 11.1 on the next page. Underlining indicates the end of a word.

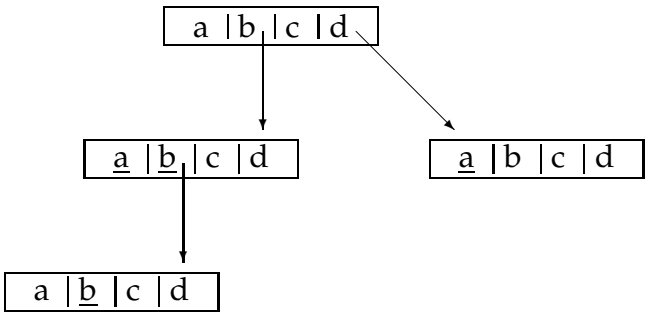

Figure 11.1: Trie — an example.

It is not difficult to implement this data structure. Nodes may be put into a linear array one by one, pointers going to the start of the next nodes. But this approach wastes memory, especially if the words are long and the nodes sparse. Using dynamic memory does not change this property.

The advantage of a trie is that the time needed for the look-up and insertion of a word is *linear to the length of the word*, this means the time needed does *not* depend on the number of words stored.

The need for memory may be reduced (paying with a small amount of time), as shown by Liang (1983). In practical applications the nodes are sparse, hence we want to store them *mixed into one another* into a linear array. One node uses the fields which are left empty by another node.

When working with this structure, we must have a way to decide which field belongs to a certain node. This may be done with a little trick. To each field we add information about which alphabet symbol is related to the array position. Moreover, two nodes must never start at the same position of the array. We must add one bit of information if the position is a *base position* and when inserting, we never pack two nodes at the same base position.

Table 11.1: Packed trie.

Index	1	2	3	4	5	6	7	8	9
Character		a	b	c	d	a	b	b	a
Pointer			5		8		6		
Base position?	Y				Y	Y		Y	
End of word?						Y	Y	Y	Y

In Table 11.1 the same language as previously used is stored. The trie starts on position 1, this is the base position. The trie root is treated specially for implementation reasons, it is always fully stored in the array, even if there

are no words starting with the appropriate character. Only the pointer is null in that case.

We assert numerical values to the alphabet symbols: $a = 1, b = 2, c = 3, d = 4$. How do we distinguish the fields belonging to a node? Let the node start at base position z. We go through positions $z + a, \ldots, z + d$ and check where the condition "the character on position $z + i$ is i" holds. For the root this is always true. In the root, there is a pointer under character b (on position 3). It points to base position 5. Moreover, the root says we have a word starting with d. Let us go through the positions belonging to base position 5, this means relating to prefix b. They are:

- position 6, this should be related to a, this holds, the pointer is null, the end-of-word flag is true, hence ba belongs to the language and any other word starting with ba does not.
- position 7, which is related to b, so the position belongs to the node, the position is end-of-word, therefore bb belongs to the language and there are words starting with bb continuing as said by the node on base position 6.
- positions 8 and 9 should belong to the characters c and d, this is not the case, these positions do not belong to the current node.

The reader may easily check that Table 11.1 on the previous page contains the same language as shown in Figure 11.1 on the preceding page. Sixteen fields are needed to store the language naïvely, we need nine when packing. The ratio is not representative, it depends on the language stored.

The trie nodes may be packed using the first-fit algorithm. This means when packing a node, we find the first position where it can be done, where we do not interfere with existing nodes and we do not use the same base position. We can speed up the process using the following heuristics. If the node we want to store is filled less than a threshold, we don't loose time finding an appropriate position but store it at the end of the array. Otherwise we use the first-fit method as described. Our experience shows that array usage much better than 95% may be obtained without significant loss of speed.

11.8 Pattern Translation Processes

If all you have is a hammer, everything looks like a nail. — popular aphorism

Let us review several tasks related to computer typesetting, in order to see whether they could be implemented as Pattern Translation Processes (PTP), implemented using PATLIB. Most of them are currently being tackled via *external* ΩTPs in OMEGA (Haralambous and Plaice, 2001).

Hyphenation of compound words The plausibility of the approach was shown for German in (Sojka, 1995).

Context-dependent ligatures In addition to the already mentioned ligatures at the compound word boundaries, another example exists:

Fraktur long s versus short s In the Gothic letter-type there are two types of s-es, a long one and the normal one. The actual usage depends on the word morphology. Another typical context-dependent auto-tagging procedure implementable by PTP.

End of sentence recognition To typeset a different width space at the end of a sentence automatically, one has to filter out abbreviations that do not normally appear at the end of a sentence. A hard, but doable task for PTP.

Spell checking Storing a big word list in a packed digital tree is feasible and gives results comparable to spelling checkers like ispell. For languages with inflection, however, several hierarchical PTP's are needed for better performance. We are afraid that PTP's cannot beat specialized fine-tuned morphological analyzers, though.

Thai segmentation There is no explicit word/sentence boundary, punctuation and inflexion in Thai text. This information, implicitly tagged by spaces and punctuation marks in most languages, is missing in standard Thai text transliteration. It is, however, needed, during typesetting for line-breaking. It has yet to be shown that the pattern-based technology is at least comparable to the currently used probabilistic trigram model (Sornlertlamvanich et al., 1999).

Arabic letter hamza Typesetting systems for Arabic scripts need to have built-in logic for choosing one of five possible appearances of the letter hamza, depending on the context. This process can easily be formulated as a PTP.

Greek accents In (Haralambous and Plaice, 2001, page 153) there is an algorithm — full of exceptions and context dependent actions — for the process of adding proper accents in Greek texts. Most parts of it can easily be described as a sequence of pattern-triggered actions and thus be implemented as a PTP.

Similarly, there are many Czech texts written without diacritics from the times when email mailers only supported seven-bit ASCII, which wait to be converted into a proper form. Even for this task PTP's could be trained.

We believe that PTP implementation based on PATLIB could become the common ground for most, if not all, ΩTP's. Hooking and piping various PTP's in OMEGA may lead to uniform, highly effective (all those mapping are *linear* with respect to the length of the text) document processing. Compared

to external ΩTP's, PTP implementation would win in speed. To some extent, we think that a new version of PATGEN based on PATLIB will not only be independent of language (for hyphenation), but of application, too.

11.9 Summary and Future Work

Write once, use everywhere. — paraphrase of a well known slogan

We have discussed the motivation for developing a new library for the handling and generation of patterns, and we have presented its design and first version. We argue that the pattern-based techniques have a rich future in many application areas and hope for PATLIB to be playing a rôle there.

Readers are invited to download the latest version of PATLIB and the PATGEN reimplementation at `http://www.fi.muni.cz/~xantos/patlib/`.

References

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. 1989. *Concrete Mathematics*. Addison-Wesley, Reading, MA, USA.

Patrick Hanks, editor. 1998. *The New Oxford Dictionary of English*. Oxford University Press, Oxford.

Yannis Haralambous and John Plaice. 1994. First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer. *TUGboat*, 15(3):344–352, September.

Yannis Haralambous and John Plaice. 1997. Methods for Processing Languages with Omega. In *Proceedings of the Second International Symposium on Multilingual Information Processing, Tsukuba, Japan*. Available as `http://omega.enstb.org/yannis/pdf/tsukuba-methods97.pdf`.

Yannis Haralambous and John Plaice. 2001. Traitement automatique des langues et composition sous Omega. *Cahiers GUTenberg*, (39–40):139–166, May.

Yannis Haralambous. 1994. A Small Tutorial on the Multilingual Features of PATGEN2. In electronic form, available from CTAN as `info/patgen2.tutorial`, January.

Donald E. Knuth. 1986. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1998. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, third edition.

Franklin M. Liang and Peter Breitenlohner. 1999. `PATtern GENeration` program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.3 from web2c distribution on CTAN.

John Plaice and Yannis Haralambous. 1996. The latest developments in Ω. *TUGboat*, 17(2):181–183, June.

Petr Sojka and Pavel Ševeček. 1995. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Petr Sojka. 1999. Hyphenation on Demand. *TUGboat*, 20(3):241–247.

Virach Sornlertlamvanich, Thatsanee Charoenporn, and Hitoshi Isahara. 1999. Building a Thai Part-Of-Speech Tagged Corpus. *The Journal of the Acoustical Society of Japan (E)*, 20(3):140–189, May.

Chapter 12

Context Sensitive Pattern Based Segmentation: A Thai Challenge

Segmentation of stream of Thai characters is the main problem any NLP or typesetting system faces before further processing. This chapter shows how the problem of segmentation of Thai text has been successfully solved by the developed techniques of competing patterns, beating techniques used so far.

The results were presented by the first author at the EACL 2003 conference in Budapest (Sojka and Antoš, 2003):

> Petr Sojka and David Antoš. 2003. Context Sensitive Pattern Based Segmentation: A Thai Challenge. pages 65–72, Budapest, April.

This chapter contains an extended version of the above paper. The present author proportion on the results presented is 75%.

Context Sensitive Pattern Based Segmentation: A Thai Challenge

Petr Sojka, David Antoš

Abstract: A Thai written text is a string of symbols *without* an explicit word boundary markup. A method for a development of a segmentation tool from a corpus of an already segmented text is described. The methodology is based on the technology of *competing patterns*. A new UNICODE pattern generation program, OPATGEN, is used for the learning phase. We show feasibility of our methodology by generating patterns for Thai segmentation from already segmented text of the Thai corpus ORCHID: the segmentation algorithm quickly reaches F-score of 93%. Finally, we enumerate possible new applications based on the pattern technique, and conclude with the suggestion of a general Pattern Translation Process. The technology is general and can be used for any other segmentation tasks as phonetic, morphologic segmentation, word hyphenation, sentence segmentation and text topic segmentation for any language.

12.1 Motivation and Problem Description

From Latin segmentum, *from* secare *'to cut' (as term in geometry).*
— Origin of word *segmentation* (Hanks, 1998)

Many natural language processing applications need to cut strings of letters, words or sentences into segments: phonetic, morphologic segmentation, word hyphenation, word phrase and sentence segmentation may serve as examples of this *segmentation task*. In Thai, Japanese, Korean and Chinese languages, where there are no explicit word boundaries in written texts, performing character stream segmentation is the first crucial step in the natural language processing of written texts. An elegant way of solving this task is to learn the segmentation from an already segmented corpus by a supervised machine learning technique.

12.1.1 The Thai Segmentation Problem

A Thai paragraph is a string of symbols (44 consonants, 28 vowels). There are neither explicit syllable, word and sentence boundaries, nor punctuation

in Thai text streams. For a lexical, semantic analysis or typesetting, the first crucial step is to find syllable, word and sentence boundaries. The Thai typesetting engine has to be able to segment the text in order to break lines automatically, too. Similarly, tools are needed to insert the `<wbr>` HTML tag automatically for the web browser rendering engine. A good word segmentation is a prerequisite for any Thai text processing including Part-of-Speech (POS) tagging (Murata et al., 2002).

12.1.2 Existing Approaches to Thai Segmentation

There is a program SWATH (Smart Word Analysis for THai) with three implemented dictionary based algorithms (longest matching, maximal matching, bigram model). It is used by the Thai Wordbreak Insertion service `http://ntl.nectec.or.th/services/www/thaiwordbreak.html` at NECTEC, the Thai National Electronics and Computer Technology Center. These methods have limited performance because of problems with handling unknown words. There are other approaches based on the probabilistic language modeling (Sornlertlamvanich, 1998; Sukhahuta and Smith, 2001), logically combined neural networks (Ma et al., 1996), or machine learning with C4.5 learning algorithm (Sornlertlamvanich et al., 2000).

Mamoru and Satoshi (2001) reported that their Thai syllable recognizer, in which knowledge rules based on heuristics derived from the analysis of unsuccessful cases were adapted, gave a ratio of segmentation of 93.9% in terms of sentences for the input of Thai text. The Thai text used was *Kot Mai Tra Sarm Duang* (Law of Three Seals), and had 20,631 sentences (Jaruskulchai, 1998, Chapter 3).

The feature based approach using RIPPER and Winnow learning algorithms is described in (Meknavin et al., 1997). Aroonmanakun (2002) recently reported the approach based on a trigram model of syllables and syllable merging, with very high precision and recall. His Thai word segmentation online service on `http://www.arts.chula.ac.th/~ling/wordseg/` is performed using maximum collocation approach.

All these attempts show the need and importance of highly efficient and quality solution of the Thai word segmentation problem.

12.2 Patterns

Patterns are used to recognize "points of interest" (segment boundaries) in data. The pattern is a sub-word of a given word set and the information of the points of interest is written between its symbols.

There are two possible values for this information. The first value indicates the point of interest *is* here, the latter indicates the point of interest *is not* here. Natural numbers are the typical representation of that knowledge; odd for yes, even for no. So we have *covering* and *inhibiting* patterns. Special symbols are often used, e.g., a dot for the word boundary. Patterns are as short as possible to store the information: the context of the variable length is modeled by this approach.

12.2.1 Competing Patterns

More formally, let us have an alphabet Σ. *Patterns* are words over the free monoid $\langle \Sigma^*, \varepsilon, \cdot \rangle$ where ε is the empty word, and $\cdot$ (centered dot) is the operation of *concatenation*. Let $\langle A, \leq \rangle$ be a partially ordered system, $\leq$ be a *lattice order* (every finite non-empty subset of A has lower and upper bound). Let . be a distinguished symbol (dot) in $\Sigma' = \Sigma \cup \{.\}$ that denotes the beginning and the end of a word: *begin of word marker* and *end of word marker*. *Classifying patterns* are the words over $\Sigma' \cup A$ such that the dot symbol is allowed at the beginning or end of patterns only.

Let P be a set of patterns over $\Sigma' \cup A$ (*competing patterns*, *pattern base*). Let $w = w_1 w_2 \ldots w_n$ be a word to classify with P. Classification $\mathit{classify}(w, P) = a_0 w_1 a_1 w_1 \ldots w_n a_n$ of w with respect to P is computed from the pattern base P by the following *competition*. All patterns whose projection to Σ match a substring of w are collected. a_i is the supremum of all values between characters w_i and w_{i+1} in matched patterns. $\mathit{classify}(w, P)$ is also called the *winning pattern*. The Winning pattern holds the definitive information (hyphenation, segmentation) about w with respect to the pattern base P. There can be several pattern bases for the same w that "compete" as well.

12.2.2 Example

An example of competing patterns used for hyphenation of English words is shown in Figure 12.1 on the next page. In this example the ordered system $\langle A, \leq \rangle$ used for classification of candidates for hyphenation border is $\mathbb{N}$ (natural numbers). There are five pattern levels used in the example — Level `1`...Level `5`. There were eight patterns that matched the input (`1na`, `1tio`,...). Patterns in odd levels are *covering*, in an even levels *inhibiting*. Inhibiting patterns specify the exceptions with respect to the knowledge acquired in lower levels. The winner pattern is `.h0y3p0h0e2n5a4t2i0o0n.`: the pattern `h e n5a t` *wins* over `n2a t`, thus possible segmentations are `hy-phen-ation`.

Competing pattern sets can be used on all levels of natural language processing — covering structures used in morphology, their exploration is seen on both syntax (parsing) and semantic (word sense discrimination) levels.

```
        h y p h e n a t i o n
Level 1          1n a
Level 1              1t i o
Level 2           n2a t
Level 2                2i o
Level 2       h e2n
Level 3 .h y3p h
Level 4       h e n a4
Level 5       h e n5a t
       .h0y3p0h0e2n5a4t2i0o0n.
        h y-p h e n-a t i o n
```

Figure 12.1: Competing patterns and pattern levels for segmentation of English word `hyphenation`.

For detailed definitions and more examples see (Liang, 1983; Sojka, 2000).

12.2.3 Comparison with Finite-State Approaches

Competing patterns technology can be viewed as one of the finite-state approaches, with their pros and cons. Competing patterns extend the power of Finite-State Transducers (FST) similarly as addition of the complement operator with respect to Σ does. Ideally, instead of storing full FST, we make patterns that embody the same information in even more compact manner. Collecting patterns matching a given word can be done in linear time, using the trie data structure for pattern storage.

It was shown that decomposition of the problem by using *local grammars* (Gross, 1997) or building cascades of Finite-State Machines (Hobbs et al., 1997) is a tractable, but a very time-consuming task. Supervised learning methods for the induction of patterns from segmented text are needed.

12.2.4 Pattern Generation — Programs PATGEN and OPATGEN

Liang (1983) wrote a pattern generation program PATGEN for the generation of hyphenation patterns from the list of already hyphenated words. The method for generation of patterns is not only independent of a language for which (hyphenation) patterns are generated, but of an application domain, too. PATGEN was used for the preparation of quality hyphenation patterns for several dozens of languages (Sojka and Ševeček, 1995). A new enriched UNICODE version of PATGEN called OPATGEN, was developed at Masaryk University in Brno (Antoš and Sojka, 2001). The program opens new possibilities for pattern generation and new applications. The only thing that must be done to create

patterns is to map the problem in mind to the alphabet used by OPATGEN (UNICODE). OPATGEN is based on separate PATLIB (Antoš, 2002) library, so even making a new special purpose frontend for a new application should be straightforward.

12.3 Thai Texts in ORCHID Corpus

Literally 'free'. — Origin of word *Thai*: (Hanks, 1998)

There is a freely available corpus of already segmented Thai texts called ORCHID (Sornlertlamvanich et al., 1997). Parts of speech are tagged, too, using the bootstrapping technique, manual editing and proofreading. There are 9,967 paragraphs in the corpus (6 MB in TIS-620 encoding).

Even native Thai speakers do not agree on the definition of the main notion — Thai word (Jaruskulchai, 1998). Problems appear whether a "compound word" should be considered as single entity or not. We have based our machine learning experiments purely on the data available in the ORCHID corpus, showing the power of the machine learning technique. We cannot comment on the quality of the corpus tagging, as we are not Thai native speakers.

The corpus consists of articles. Every article has headers containing meta-information, usually in Thai and English, followed by the text, consisting of paragraphs. Paragraphs are numbered and tagged with `#Pn` marks. Paragraphs contain sentences. The sentences are tagged with `#n`. Each sentence appears twice, first untagged. The second occurrence is tagged with part-of-speech tags. Each word is followed by the tag, e.g., `/VACT` for the active verb.

12.3.1 Corpus Preprocessing

In order to create patterns recognizing Thai word boundaries we had to preprocess the corpus. We used a simple Perl script. The word boundaries are marked in the second occurrences of sentences in the corpus. Therefore, we cut out only the marked parts. The "points of interest" should be denoted with '-' sign for OPATGEN. We substituted all part-of-speech tags with the minus signs. There are also text entities marked with single angle bracket tags, e.g., `<space>`. All of them act as word separators in the corpus and we also substituted them with our word boundary mark. That is also what we did with numbers, we silently removed them as there is no reason to encounter them into patterns. When applying patterns, numbers are trivially word boundaries.

Finally, we joined the whole paragraphs up. The places between sentences are also word boundaries. We decided not to join larger portions of the text (like several paragraphs or even articles) as we did not want the words OPATGEN had to deal with to be longer than hundreds of symbols. It would

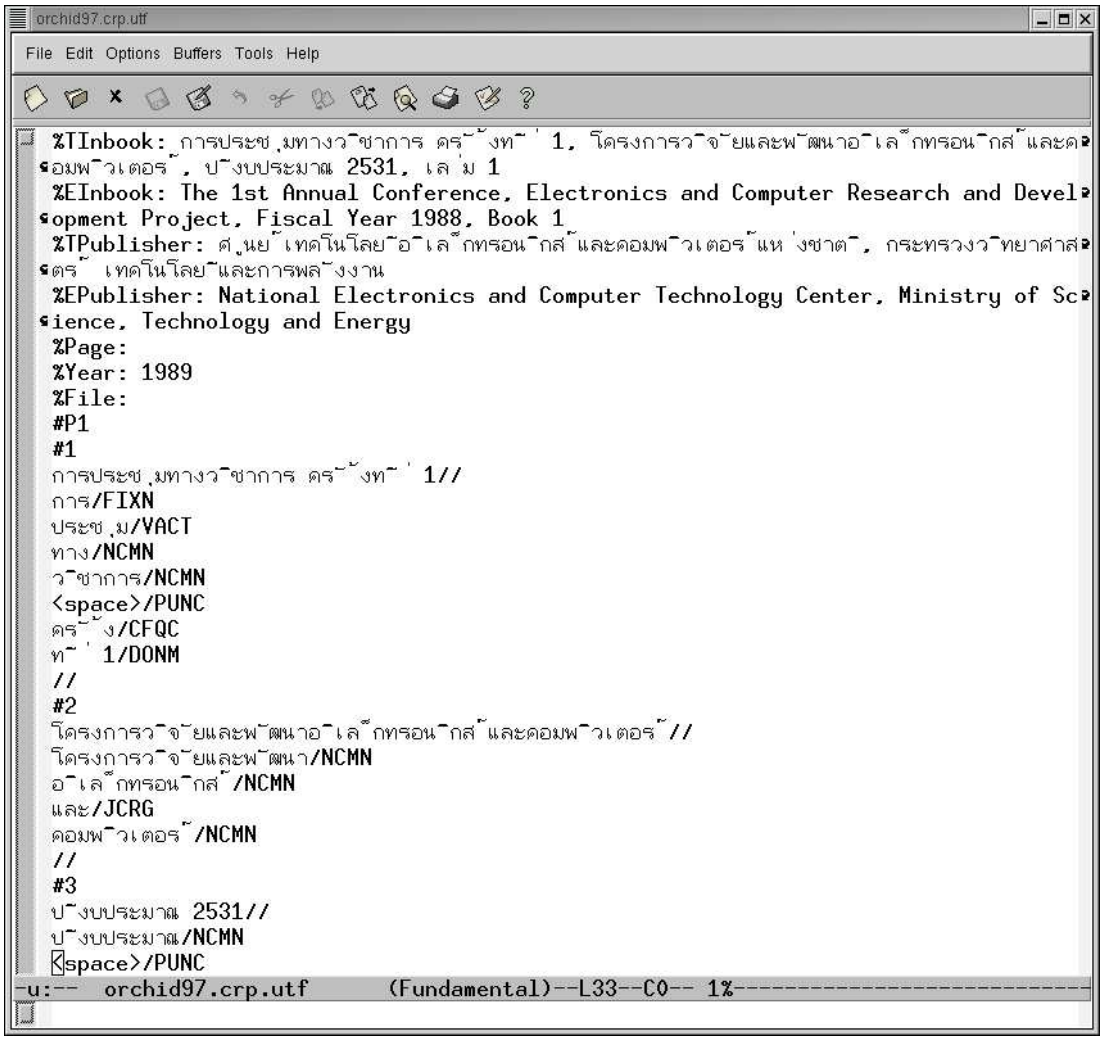

Figure 12.2: ORCHID loaded into Emacs.

slow the pattern generation down and we would add only a bit of information, only the word boundaries that appear between words finishing and starting a paragraph. The preprocessed starting paragraphs from ORCHID (input for OPATGEN) look like this:

-การ-ประชุม-ทาง-วิชาการ-ครั้ง-ที่-โครงการวิจัยและพัฒนา-อิเล็กทรอนิกส์-และ-คอมพิวเตอร์-ปีงบประมาณ-เล่ม-

-วัน-ที่-สิงหาคม-ห้องประชุม-ชั้น-

-สาร-

-ฯพณฯ-รัฐมนตรีว่าการ-กระทรวงวิทยาศาสตร์-เทคโนโลยีและการพลังงาน-

-ประเทศไทย-ได้-มี-การ-ปรับเปลี่ยน-โครงสร้าง-ใน-การ-พัฒนา-เศรษฐกิจ-ของ-ประเทศ-จาก-ประเทศ-เกษตรกรรม-ไปสู่-ความ-เป็น-ประเทศอุตสาหกรรม-มาก-ยิ่งขึ้น-ใน-การ-ดำเนินการ-เพื่อให้-บรรลุ-วัตถุประสงค์-ดังกล่าว-จะ-ต้อง-อาศัย-ปัจจัยพื้นฐาน-หลาย-ประการ-ใน-การ-เป็น-ตัวเร่ง-และ-เป็น-ฐาน-เช่น-การ-พัฒนา-เทคโนโลยี-ที่-ใช้-ใน-การ-ผลิต-ของ-ภาคอุตสาหกรรม-กระทรวงวิทยาศาสตร์-เทคโนโลยีและการพลังงาน-จึง-ได้-ให้ความสำคัญ-เป็น-ลำดับ-สูง-ใน-การ-พัฒนา-อุตสาหกรรม-อิเล็กทรอนิกส์-และ-คอมพิวเตอร์-ซึ่ง-อุตสาหกรรม-นี้-จะ-มี-บทบาท-ที่-สำคัญ-มาก-ใน-ภาคอุตสาหกรรม-โดย-เป็น-ปัจจัยพื้นฐาน-หรือ-ส่วนประกอบ-ที่-สำคัญ-ของ-การ-ผลิต-ผลิตภัณฑ์อุตสาหกรรม-แทบ-ทุก-สาขา-

. . .

12.4 Methodology

Problems worthy of attack prove their worth by hitting back.
— Piet Hein: Grooks

An important question is what kind of evaluation measures is most appropriate to compare the segmentation proposed by automated tools with the correct segmentations in the test set. A widely used evaluation scheme is the *PARSEVAL scheme*, based on the notions of *precision* and *recall*.

12.4.1 Evaluation Measures

The definition of the measures for our application is as follows:

$$\text{Precision} = \frac{\#\text{ found well}}{\#\text{ found well} + \#\text{ bad}} \tag{12.1}$$

$$\text{Recall} = \frac{\#\text{ found well}}{\#\text{ found well} + \#\text{ missed}} \tag{12.2}$$

A segment is correct if both the start and the end of the segment is correctly predicted.

The precision and recall scores are combined into a single measure, known as the *F-score* (Manning and Schütze, 1999):

$$\text{F-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12.3}$$

Other possibilities of an evaluation metric for segmentation are P_k metric (Beeferman et al., 1997; Beeferman et al., 1999) or WindowDiff (Pevzner and Hearst, 2002). We do not use them, as they are appropriate for the topic

text segmentation, where small errors in positions of segment cuts are acceptable.

Table 12.1: Results of Thai segmentation pattern generation (6,000 paragraphs from ORCHID).

level	length	param	% correct	% wrong	# patterns	UTF-8 size (kB)
1	1–5	1 6 1	97.98	4.87	+12907	130
2	2–6	1 1 1	96.83	0.69	+ 2,091	156
3	3–11	1 3 1	99.58	0.82	+ 2,578	204
4	4–12	4 1 1	97.83	0.03	+ 685	217
5	9–19	1 3 1	99.58	0.15	+ 1,689	270
6	10–20	1 1 1	99.56	0.04	+ 119	274

12.4.2 Experiments

We have divided the corpus into the training set (3/5) and the test set (2/5) and used the training set (6,000 paragraphs) for pattern generation. Ideally, we strive for smallest patterns solving the task with the highest F-score as possible. As a general procedure how to achieve this goal is not known, parameters for the generation were chosen after some trial and error (one has to find good-working thresholds for adding new patterns). Using the knowledge about threshold parameters used for the generation of hyphenation patterns we quickly reached 100% precision (patterns were able to cover segmentation in training set without errors).

Table 12.2: Results of Thai segmentation pattern generation (8,000 paragraphs from ORCHID).

level	length	param	% correct	% wrong	# patterns	UTF-8 size (kB)
1	1–5	1 6 1	97.92	4.86	+15,443	161
2	2–6	1 1 1	96.53	0.65	+ 2,596	196
3	3–11	1 3 1	99.57	0.79	+ 3,448	267
4	4–12	4 1 1	97.87	0.03	+ 953	286
5	9–19	1 3 1	99.68	0.12	+ 2,468	364
6	10–20	1 1 1	99.67	0.04	+ 129	368

Parameters used for pattern generation are shown in Table 12.1. In the second column, there are lengths of pattern candidates. A generation process

Table 12.3: Precision, recall, and F-score on unseen text.

trained on # para	good	bad	missed	precision	recall	F-score
4,000	139,788	11,231	15,529	92.56%	90.00%	91.26
6,000	98,243	7,951	9,432	92.51%	91.24%	91.87
8,000	46,361	3,358	3,703	93.25%	92.60%	92.92

can be parametrized by several parameters whose tuning strategies are beyond the scope of this paper; see (Sojka and Ševeček, 1995; Sojka, 1995) for details. Setting of the thresholds could be tuned so that virtually all hyphenation points are covered.

As there are quite long words in Thai (10 to 20-syllable word is not an exception), to achieve 100% precision, we may probably need patterns as long as 20 characters to model long distance dependencies. This increases the time of pattern generation, but not above the achievable level (it took half a day on Pentium 4 class PC).

The 'param' column contains the pattern choosing the rule weights. The percentages show the behavior of the patterns on the corpus during generation. Finally, there is the number of patterns added in particular level and the pattern size in kilobytes (coded in UTF-8 encoding). It is seen that most of the work is done by short patterns.

Next, we increased the training set to 8,000 paragraphs. Results are shown in Table 12.2 on the previous page. Both precision and recall slightly increased with bigger training sets.

The behavior of the patterns on data they were generated from does not show how they act on previously unseen data (generalization abilities). Therefore we tested performance on the test set (3,967 paragraphs). The obtained recall is above 90%. With the bigger training corpus we do get better performance measures as shown in Table 12.3. From the main results given in this table it follows that the the ORCHID Corpus is quite small for our task: given the bigger training corpus one would have even better performance.

Resulting 19424 patterns look like this:

.o1m .p1me .pre1p .s1f .s2mo .s1mp .st1in .x1p .x1y .ก1ก .ก1ม
.ก1ห .การจ่าย3 .การ5พัฒนา .การพัฒนาระบบ5 .การพัฒนาโปรแกรม5ส .การรับ4
.การ1ว .การ5ศึกษา .การออกแบบ5 .การออกแบบและ5 .การออกแบบและพัฒนา5
.การ5เริ่ม .ก่อนที่3 .คณะ5กรรมการนโยบาย. .คณะกรรมการ5บริหาร.
.คณะกรรมการอำนวยการ6 .คณะผู้5ทำ5 .คณะ3วิศ .คณะ5อนุกรรมการท .คำ3ก
.คำ1ภ .ง1ก .งาน1ค .งาน1ฐ . . .

To sum up the properties of pattern technique, even with small data like ORCHID Corpus we have got 1:20 compression of the information stored

and hidden there. The patterns can be trained to 100% precision on the training data and to making essentially no error (one can always add the pattern for the whole paragraph). One can balance tradeoff between recall and precision measured on testing data. Moreover, the application of patterns is very efficient. The speed of the segmentation is *linear* with respect to the length of the word we apply them on: in our case with the length of the paragraph. It makes them one of the first choices in cases where the processing speed is important. The speed of the segmentation using developed Thai patterns is at the range of 10,000 words per second (wps) on a Pentium 4 class PC. Memory consumption using compact digital trie implementation used in TEX for this performance is much below 0.3 MB.

The generation process may be optimized with respect to the resulting pattern size; the tradeoff among size, covering ratio, and error is adjustable. Nevertheless, good patterns may be small in size and therefore applicable for handhelds, mobile phones and other small equipment. There is no reason for SMS's to have awfully broken words on the display of a cellular phone.

Creating patterns is possible due to availability of large tagged corpora. The technique of competing pattern generation might be useful for corpora builders as well. The thresholds set for pattern generation can be tuned up in such a way that highly improbable (bad?) segmentation points are not learnt. This way, the pattern generation process may serve as a filter selecting possible errors in input corpus tagging. These errors are a traditional nightmare for anybody who deals with large experimental data. Creating patterns for a phenomenon appearing in the corpus thus may help to clean the errors when the error list reported by the generator is checked manually. The size of the error list may be tuned by the number of levels and by the setting the thresholds appropriately.

12.5 Data-Driven Approach Based on Competing Patterns

If all you have is a hammer, everything looks like a nail.
— Abraham Maslow

Let us comment on the technology of competing patterns from different points of view. The application of the techniques of bootstrapping and stratification (Sojka, 1995; Sojka, 1999) made it even more attractive.

12.5.1 Pattern Translation Processes

A process based on competing patterns that adds markup to the string of symbols is called *Pattern Translation Process*, PTP, (Antoš and Sojka, 2001). In the terminology of automata theory, it is a special type of the finite

state transducer. With this finite state approach (Roche and Schabes, 1997), quite powerful engines could be designed, with an exceptional speed: time complexity of the PTP implementation based on digital tries is *linear* with respect to the input length (length of input sentence). Putting PTP's in a cascade, we still stay in linear time. In addition to PATLIB, there are quite efficient digital trie publicly available implementations as JUDY (Silverstein, 2002). Such PTP implementations are very memory efficient.

Although many natural language special purpose tools are being developed, their implementation using competing patterns technology with bootstrapping, stratification and pattern generation techniques (Sojka and Ševeček, 1995; Sojka, 1995; Sojka, 1999) is possible. We believe that in addition to the one of the hardest problems — Thai segmentation — many other NLP problems can be solved by our competing pattern data-driven approach. Let us add couple of notes about applications in the Computer Typesetting area.

12.5.2 Applications in Computer Typesetting

A good list of tough problems in the area of the computer typesetting, most of which are tractable by OPATGEN, is presented in (Haralambous and Plaice, 2001). A new typesetting system OMEGA (Haralambous and Plaice, 1997), gradually developed from the well known TEX typesetting system, is designed to be able to typeset a text in all languages of the world. To solve typesetting problems that are not supported by the OMEGA engine itself external special purpose programs outside of OMEGA are invoked as so called external OTP's (OMEGA Translation Processes).

When we analyze most of the problems and application of computer typesetting described in (Haralambous and Plaice, 2001), we see that most of them could be formulated as a string rewriting of regular languages with a varying context length. They can be seen as translation processes that typically add information to a token (character or word) stream.

In the typesetting engine application, the main idea is the usage of pattern recognition in the middle of the digestive process of a typesetting engine. A cascade of PTP's is able to efficiently solve the hardest problems known so far, in linear time, given the sets of competing patterns.

The process should take part just after macro expansion phase. The patterns are recognized and the token list is changed appropriately on places denoted by the set of patterns. Note that mere pre-processing is not strong enough to simulate such a process due to macroexpansion, allowing generation of visible material.

The system OMEGA allows similar kind of processing, OMEGA Translation Processes (OTP's). There are two kind of OTP's, internal and external. Internal processes allow regular expression substitutions. External

ones use external programs to change the stream of tokens and are of course more general than PTP's. The question is why to implement a more specialized process. There are several good arguments to do so.

- Internal implementation of pattern translation would be significantly quicker than calling external programs (note the ideas of incorporating METAPOST into [Ex]TEX).
- In any case, the pattern recognizing machinery is needed to provide word hyphenation.
- Having a component for pattern recognition, it would be easy to provide Hyphenation on Demand, it means loading the hyphenation pattern on the fly, not only when generating format files.
- We think that using UN*X-like filters with text interfaces is too big overhead for natural binary data like token lists.
- Constructing relatively small components from which the whole system is built is a good software engineering practice.

We think of a typesetting engine created out of relatively independent components with precisely defined interfaces. The components can be connected with regard to user's wishes. If the user wants to use pattern translation, he/she would use the component. Similarly, if someone extensively uses embedded graphics, he would use METAPOST component. The idea is an exact analogy of modular operating system architecture with small kernel that turns out to be stable, maintainable and efficient.

12.6 Conclusion and Future Work

We are all apprentices in a craft where no-one ever becomes a master.
— Ernest Hemingway

We have shown the feasibility of technology of competing patterns to tackle the Thai word segmentation problem.

To evaluate next steps of the technology — bootstrapping and stratification techniques — we are looking for (native Thai) partners to pursue further research. Improving consistency of tagging of the corpus will even improve the system performance. Application for Thai sentence segmentation problem (Charoenpornsawat and Sornlertlamvanich, 2001) is straightforward, too, but a bigger corpus is needed for learning.

Having a tagged corpus freely available, one may try an easier task of not only word segmentation, but syllable segmentation. This may be needed for typesetting engine to use, due to long Thai words. The most promising approach thus seems to be using competing patterns for syllable segmentation,

and then parse the text upwards, merging syllables into words and words into sentences.

Other general questions remain open. How to set the OPATGEN parameters to get space-minimal, level-minimal, highest-precision, highest recall patterns given the data? We are still looking for a rigorous theory for setting the parameters of the pattern generation process. We also think of an automated pattern generation, performing the parameter setting using an expert system or statistical methods.

We have also spent some effort on developing better generation strategies. The implicit strategy used by OPATGEN is basically brute-force testing of all reasonable pattern candidates. It is not straightforward how to optimize the process, but using bigram or trigram statistics of wordlist is an idea worth trying.

The choice of best fitting data structure for patterns needs further investigation, even though keeping the set of patterns is a general dictionary problem, studied for years by computer scientists. There are other approaches than those used in TEX, PATGEN and OPATGEN, namely packed dynamic tries LC-tries (Nilsson and Karlsson, 1999) and new digital trie library implementations like JUDY (Silverstein, 2002).

Also the idea of a hardware accelerated pattern generation engine (for generation only) might become a solution for people who use time of their CPU's with pattern generation.

Acknowledgment

Support of the grant CEZ:J07/98:143300003 is acknowledged.

References

David Antoš and Petr Sojka. 2001. Pattern Generation Revisited. In Simon Pepping, editor, *Proceedings of the 16th European TEX Conference, Kerkrade, 2001*, pages 7–17, Kerkrade, The Netherlands, September. NTG.

David Antoš. 2002. PATLIB, Pattern Manipulation Library. `http://www.fi.muni.cz/~xantos/patlib/`.

Douglas Beeferman, Adam Berger, and John Lafferty. 1997. Text segmentation using exponential models. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing*, pages 35–46, Providence, RI.

Douglas Beeferman, Adam Berger, and John Lafferty. 1999. Statistical Models of Text Segmentation. *Machine Learning*, 34(1–3):177–210.

Paisarn Charoenpornsawat and Virach Sornlertlamvanich. 2001. Automatic Sentence Break Disambiguation for Thai. In *Proceedings of ICCPOL 2001*, pages 231–235, May.

Maurice Gross. 1997. The Construction of Local Grammars. (Roche and Schabes, 1997), pages 329–354.

Patrick Hanks, editor. 1998. *The New Oxford Dictionary of English*. Oxford University Press, Oxford.

Yannis Haralambous and John Plaice. 1997. Methods for Processing Languages with Omega. In *Proceedings of the Second International Symposium on Multilingual Information Processing, Tsukuba, Japan*. Available as `http://omega.enstb.org/yannis/pdf/tsukuba-methods97.pdf`.

Yannis Haralambous and John Plaice. 2001. Traitement automatique des langues et composition sous Omega. *Cahiers GUTenberg*, (39–40):139–166, May.

Jerry R. Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. 1997. FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. (Roche and Schabes, 1997), pages 383–406.

Chuleerat Jaruskulchai. 1998. *Automatic Indexing for Thai Text Retrieval*. Ph.D. thesis, School of Engineering and Applied Science, George Washington University, August.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Department of Computer Science, Stanford University, August.

Qing Ma, Hitoshi Isahara, and Hiromi Ozaku. 1996. Automatic part-of-speech tagging of thai corpus neural networks. In *Lecture Notes in Computer Science 1112*, pages 275–280. Springer-Verlag.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Surapant Meknavin, Paisarn Charoenpornsawat, and Boonserm Kijsirikul. 1997. Feature-based Thai Word Segmentation. In *Proceedings of the Natural Language Processing Pacific Rim Symposium (NLPRS 1997)*, pages 41–46.

Masaki Murata, Qing Ma, and Hitoshi Isahara. 2002. Comparision of Three Machine-Learning Methods for Thai Part-of-Speech Tagging. *ACM Transactions on Asian Language Information Processing*, 1(2):145–158.

Stefan Nilsson and Gunnar Karlsson. 1999. IP-Address Lookup Using LC-Tries. *IEEE Journal on Selected Areas in Communications*, 17(6):1083–1092.

Lev Pevzner and Marti A. Hearst. 2002. A Critique and Improvement of an Evaluation Metric for Text Segmentation. *Computational Linguistics*, 28(1):19–36.

Emmanuel Roche and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press.

Alan Silverstein. 2002. Judy IV Shop Manual. `http://judy.sourceforge.net/application/shop_interm.pdf`.

Petr Sojka and Pavel Ševeček. 1995. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Petr Sojka. 1999. Hyphenation on Demand. *TUGboat*, 20(3):241–247.

Petr Sojka. 2000. Competing Patterns for Language Engineering. In Petr Sojka, Ivan Kopeček, and Karel Pala, editors, *Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000*, Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, pages 157–162, Brno, Czech Republic, September. Springer-Verlag.

Virach Sornlertlamvanich, Thatsanee Charoenporn, and Hitoshi Isahara. 1997. ORCHID: Thai Part-Of-Speech Tagged Corpus. Technical Report TR-NECTEC-1997-001, Thai National Electronics and Computer Technology Center, Thailand, December. `http://www.links.nectec.or.th/`.

Virach Sornlertlamvanich, Tanapong Potipiti, and Thatsanee Charoenporn. 2000. Automatic Corpus-Based Thai Word Extraction with the C4.5 Learning Algorithm. In *COLING*, pages 802–807. Morgan Kaufmann.

Virach Sornlertlamvanich. 1998. *Probabilistic Language Modeling for Generalized LR Parsing*. Ph.D. thesis, Department of Computer Science, Tokyo Institute of Technology, September.

Rattasit Sukhahuta and Dan Smith. 2001. Information Extraction Strategies for Thai Documents. *International Journal of Computer Processing of Oriental Languages (IJCPOL)*, 14(2):153–172.

Bibliography

Steven Paul Abney. 1997. Part-of-Speech Tagging and Partial Parsing. In Young and Bloothooft (Young and Bloothooft, 1997), pages 118–136.

R. E. Allen. 1990. *The Oxford Spelling Dictionary*, volume II of *The Oxford Library of English Usage*. Oxford University Press.

AMS. 1993. Instructions for Author-Prepared Books.

Anonymous. 1993. *The Chicago Manual of Style*. University of Chicago Press, fourteenth edition.

David Antoš and Petr Sojka. 2001. Pattern Generation Revisited. In Simon Pepping, editor, *Proceedings of the 16th European TEX Conference, Kerkrade, 2001*, pages 7–17, Kerkrade, The Netherlands, September. NTG.

David Antoš. 2002. PATLIB, Pattern Manipulation Library. `http://www.fi.muni.cz/~xantos/patlib/`.

Susan Armstrong, Kenneth Church, Pierre Isabelle, Sandra Manzi, Evelyne Tzoukermann, and David Yarowsky, editors. 1999. *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Publishers Group.

Wirote Aroonmanakun. 2002. Collocation and Thai Word Segmentation. In *Proceedings of SNLP-Oriental COCOSDA 2002*, pages 68–75.

Donald Arsenau. 1994. Benchmarking paragraphs without hyphenation. Posting to the Usenet group `news:comp.text.tex` on December 13.

Giorgio Ausiello, Giorgio Gambosi, Pierluigi Crescenzi, and Viggo Kann. 1999. *Complexity and Approximation*. Springer-Verlag.

Wilhelm Barth and H. Nirschl. 1985. Implementierung eines Verfahrens für die Silbentrennung. Technical Report Bericht Nr. 26, Institut für Praktische Informatik, September.

Wilhelm Barth and Helmut Steiner. 1992. Deutsche Silbentrennung für TEX 3.1 (German hyphenation for TEX 3.1). *Die TEXnische Komödie*, (Heft 1). Journal of DANTE (Deutschsprachige Anwendervereinigung TEX e.V.); Group of German-speaking TEX Users.

Wilhelm Barth, Helmut Steiner, and H. Herbeck. 1993. ISITEX Interaktive Silbentrennung für die deutsche Sprache unter TEX 3.14 und 3.141 unter UNIX (Interactive hyphenation for German and TEX 3.14 and 3.141 under UNIX). Electronic documentation of ISITEX from `http://www.apm.tuwien.ac.at/`, August.

Claudio Beccari, Radu Oprea, and Elena Tulei. 1995. How to make a foreign language pattern file: Romanian. *TUGboat*, 16(1):30–41.

Claudio Beccari. 1992. Computer Aided Hyphenation for Italian and Modern Latin. *TUGboat*, 13(1):23–33, April.

Douglas Beeferman, Adam Berger, and John Lafferty. 1997. Text segmentation using exponential models. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing*, pages 35–46, Providence, RI.

Douglas Beeferman, Adam Berger, and John Lafferty. 1999. Statistical Models of Text Segmentation. *Machine Learning*, 34(1–3):177–210.

Barbara Beeton. 1984. Hyphenation exception log. *TUGboat*, 5(1):15, May.

Barbara Beeton. 1985. Hyphenation exception log. *TUGboat*, 6(3):121, November.

Barbara Beeton. 1986. Hyphenation exception log. *TUGboat*, 7(3):146–147, October.

Barbara Beeton. 1989. Hyphenation exception log. *TUGboat*, 10(3):336–341, November.

Barbara Beeton. 1992. Hyphenation exception log. *TUGboat*, 13(4):452–457, December.

Branimir Boguraev and James Pustejovsky. 1996. *Corpus Processing for Lexical Acquisition*. MIT Press.

Bert Bos. 1999. Internationalization/localization. `http://www.w3.org/International/O-HTML-hyphenation.html`.

Johannes Braams. 1991a. Babel, a multilingual style-option system for use with LaTeX's standard document styles. *TUGboat*, 12(2):291–301, June.

Johannes Braams. 1991b. Babel, a multilingual style-option system. *Cahiers GUTenberg*, 10–11:71–72, September.

Johannes Braams. 1993. An update on the babel system. *TUGboat*, 14(1):60–62, April.

Peter Breitenlohner. 1988. German TeX, a next step. *TUGboat*, 9(2):183–185, August.

Eric Brill and Mihai Pop. 1999. Unsupervised learning of disambiguation rules for part-of-speech tagging. (Armstrong et al., 1999), pages 27–42.

Eric Brill, Radu Florian, John C. Henderson, and Lidia Mangu. 1998. Beyond N-Gram: Can Linguistic Sophistication Improve Language Modeling? In *Proceedings of the ACL '98*.

Eric Brill. 1993. *A Corpus-Based Approach to Language Learning*. Ph.D. thesis, University of Pennsylvania.

S. Brunak and B. Lautrup. 1990. *Neural Networks: Computers with Intuition*. World Scientific, Singapore.

J. Richard Büchi. 1989. *Towards a Theory of Formal Expressions*. Springer-Verlag, New York, U.S.A.

Guido Bugmann, Petr Sojka, Michael Reiss, Mark Plumbley, and John G. Taylor. 1992. Direct Approaches to Improving the Robustness of Multilayer Neural Networks. pages 1063–1066, Brighton, UK, September. Elsevier Science Publishers B.V.

Cezar Câmpeanu, Nicolae Sânteau, and Sheng Yu. 1998. Minimal cover-automata for finite languages. In Jean-Marc Champarnaud, Denis Maurel, and Djelloul Ziadi, editors, *Lecture Notes in Computer Science 1660*, pages 43–56, Berlin, Heidelberg. Springer-Verlag.

G. Canzii, F. Genolini, and Dario Lucarella. 1984. Hyphenation of Italian words. *TUGboat*, 5(1):14, May.

Claudio Carpineto and Giovanni Romano. 2004. *Concept Data Analysis: Theory and Applications*. Wiley, July.

František Čermák. 1998. Czech National Corpus: Its Character, Goal and Background. In Petr Sojka, Václav Matoušek, Karel Pala, and Ivan Kopeček, editors, *Text, Speech and Dialogue*, pages 9–14, Brno, Czech Republic, September. Masaryk University Press.

Paisarn Charoenpornsawat and Virach Sornlertlamvanich. 2001. Automatic Sentence Break Disambiguation for Thai. In *Proceedings of ICCPOL 2001*, pages 231–235, May.

Janka Chlebíková. 1991. Ako rozdeliť (slovo) Československo (How to Hyphenate (word) Czechoslovakia). *CSTUG Bulletin*, 1(4):10–13, April.

Noam Chomsky. 1956. Three Models for the Description of Language. *IEEE Transactions on Information Theory*, IT-2:113–124.

Noam Chomsky. 1957. *Syntactic Structures*. Mouton, The Hague.

Václav Chvátal. 1979. A Greedy Heuristic for the Set Covering Problem. *Mathematics of Operations Research*, 4:233–235.

Matthias Classen. 1998. An extension of TEX's hyphenation algorithm. `ftp://peano.mathematik.uni-freiburg.de/pub/etex/hyphenation/`.

Jacques Désarménien. 1984. How to run TEX in a French environment: Hyphenation, fonts, typography. *TUGboat*, 5(2):91, November.

Zuzana Došlá, Roman Plch, and Petr Sojka. 1999. Matematická analýza s programem Maple: 1. Diferenciální počet funkcí více proměnných (Mathematical Analysis with Program Maple: 1. Differential Calculus). CD-ROM, `http://www.math.muni.cz/~plch/mapm/`, December.

Zuzana Došlá, Roman Plch, and Petr Sojka. 2002. Matematická analýza s programem Maple: 2. Nekonečné řady (Mathematical Analysis with Maple: 2. Infinite Series). CD-ROM, `http://www.math.muni.cz/~plch/nkpm/`, December.

DUDEN. 1991. *Duden Band 1 — Rechtschreibung der Deutschen Sprache*. Dudenverlag, 20., neu bearbeitete und erweiterte Auflage edition.

Sandra L. Emerson and Karen Paulsell. 1987. *troff Typesetting for* UNIX *TM Systems*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

M. Fanton. 1991. TEX: les limites du multilinguisme. *Cahiers GUTenberg*, 10–11:73–80, September.

Michael J. Ferguson. 1985. A multilingual TEX. *TUGboat*, 6(2):57–58, July.

Michael J. Ferguson. 1988. TEX is Multilingual. In Thiele (Thiele, 1988), pages 179–189.

Michael J. Ferguson. 1990. Fontes latines européennes et TEX 3.0. *Cahiers GUTenberg*, 7:29–32, November.

Nelson W. Francis and Henry Kučera. 1982. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin.

Bernard Gaulle. 1994. Requirements in multilingual environments. in electronic form (version 1.02) on CTAN as file `vt15d02.tex`, March.

Peter Géczy, Petr Sojka, and Jan Blatný. 1993. Robustness and Generalization of Multilayer Neural Networks. In Igor Mokriš, editor, *Proceedings of the International Conference Image Processing and Neural Networks, Liptovský Mikuláš, 1993*, pages 163–170, Liptovský Mikuláš. Military Technical University in Liptovský Mikuláš, Slovak Electrotechnical Society of Military Technical University.

Charles F. Goldfarb. 1990. *The SGML handbook*. Clarendon Press, Oxford.

John Goldsmith. 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics*, 27(2):153–198, June.

Michel Goossens, editor. 1994. *Proceedings of the TEX Users Group 15th Annual Meeting, Santa Barbara, 1994*, Portland, Oregon, U.S.A. TEX Users Group.

Philip Babcock Gove and Merriam Webster. 2002. *Webster's Third New International Dictionary of the English language Unabridged*. Merriam-Webster Inc., Springfield, Massachusetts, U.S.A, January.

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. 1989. *Concrete Mathematics*. Addison-Wesley, Reading, MA, USA.

Ulf Grenander. 1993. *General Pattern Theory*. Clarendorn Press, Oxford.

Maurice Gross. 1997. The Construction of Local Grammars. (Roche and Schabes, 1997), pages 329–354.

Jozef Gruska. 1997. *Foundations of Computing*. International Thomson Computer Press.

Petr Hájek and Tomáš Havránek. 1978. *Mechanising hypothesis formation — Mathematical foundations for a general theory*. Springer-Verlag.

Petr Hájek, Jan Rauch, David Coufal, and Tomáš Feglar. 2004. The GUHA Method, Data Preprocessing and Mining. In *Database Support for Data Mining Applications*, volume LNAI 2862, pages 135–153.

Jiří Haller. 1956. *Jak se dělí slova (How the Words Get Hyphenated)*. Státní pedagogické nakladatelství Praha.

Hans van Halteren, editor. 1999. *Syntactic Wordclass Tagging*. Kluwer Academic Publishers Group.

Patrick Hanks, editor. 1998. *The New Oxford Dictionary of English*. Oxford University Press, Oxford.

Yannis Haralambous and John Plaice. 1994a. First applications of Ω: Greek, Arabic, Khmer, Poetica, ISO-10646/Unicode, etc. In Goossens (Goossens, 1994), pages 256–264.

Yannis Haralambous and John Plaice. 1994b. First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer. *TUGboat*, 15(3):344–352, September.

Yannis Haralambous and John Plaice. 1997. Methods for Processing Languages with Omega. In *Proceedings of the Second International Symposium on Multilingual Information Processing, Tsukuba, Japan*. Available as `http://omega.enstb.org/yannis/pdf/tsukuba-methods97.pdf`.

Yannis Haralambous and John Plaice. 1998. The Design and Use of a Multiple-Alphabet Font with Ω. In Roger D. Hersch, Jacques André, and Heather Brown, editors, *Lecture Notes in Computer Science 1375*, pages 126–137. Springer-Verlag, April.

Yannis Haralambous and John Plaice. 2001. Traitement automatique des langues et composition sous Omega. *Cahiers GUTenberg*, (39–40):139–166, May.

Yannis Haralambous. 1991. ScholarTEX. *Cahiers GUTenberg*, 10–11:69–70, septembre.

Yannis Haralambous. 1992a. TEX Conventions Concerning Languages. *TEX and TUG News*, 1(4):3–10.

Yannis Haralambous. 1992b. Hyphenation patterns for ancient Greek and Latin. *TUGboat*, 13(4):457–469, December.

Yannis Haralambous. 1993a. DC fonts — questions and answers. *TEX and TUG News*, 2(1):10–12.

Yannis Haralambous. 1993b. Using `PATGEN` to Create Welsh Patterns. Submitted to TUGboat, July.

Yannis Haralambous. 1994. A Small Tutorial on the Multilingual Features of PATGEN2. In electronic form, available from CTAN as `info/patgen2.tutorial`, January.

Yannis Haralambous. 1996. ΩTimes and ΩHelvetica fonts under development: Step One. *TUGboat*, 17(2):126–146, June.

Yannis Haralambous. 1999. From Unicode to Typography, a Case Study: the Greek Script. Proceedings of 14th International Unicode Conference, available from `http://omega.enstb.org/yannis/pdf/boston99.pdf`, March.

Florian Hars. 1999. Typo-l email discussion list, 4 January.

Piet Hein. 1966. *Grooks*. MIT Press, Cambridge, Massachusetts.

Zdeněk Hlavsa et al. 1993. *Pravidla českého pravopisu (The Rules of the Czech Spelling)*. Academia Praha.

Jerry R. Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. 1997. FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. (Roche and Schabes, 1997), pages 383–406.

Douglas R. Hofstadter. 1979. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books.

Douglas R. Hofstadter. 1983. Artificial intelligence: Subcognition as computation.

Jan Holeček and Petr Sojka. 2004. Animations in a pdfTEX-generated PDF. In Apostolos Syropoulos, Karl Berry, Yannis Haralambous, Baden Hughes, Steven Peter, and John Plaice, editors, *TEX, XML, and Digital Typography*, volume 3130 of *Lecture Notes in Computer Science*, pages 179–191, Berlin, Heidelberg, August. Springer-Verlag.

Olle Jarnefors. 1995. ISO-10646 email discussion list, April.

Chuleerat Jaruskulchai. 1998. *Automatic Indexing for Thai Text Retrieval*. Ph.D. thesis, School of Engineering and Applied Science, George Washington University, August.

Alan Jeffrey. 1993. A PostScript font installation package written in TEX. *TUGboat*, 14(3):285–292, October.

Tao Jiang, Arto Salomaa, Kai Salomaa, and Sheng Yu. 1995. Decision problems for patterns. *Journal of Computer and Systems Sciences*, 50(1):53–63.

Fred Karlsson, A. Voutilainen, J. Heikkilä, and A. Antilla. 1995. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin.

Fred Karlsson. 1993. Automatic Hyphenation of Finnish. Technical Report 13, University of Helsinki, Department of General Linguistics.

Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. 1996. Regular Expressions for Language Engineering. *Natural Language Engineering*, 2(4):305–328.

Blanka Kirsteinová and Blanka Borg. 1999. *Dánsko-český slovník, Dansk-Tjekkisk Ordbog [Danish-Czech Dictionary]*. LEDA, Prague, Czech Republic.

Donald E. Knuth and Michael F. Plass. 1981. Breaking Paragraphs into Lines. *Software—Practice and Experience*, 11(11):1119–1184, November.

Donald E. Knuth. 1973a. *Fundamental Algorithms*. The Art of Computer Programming. Addison-Wesley, Reading, Massachusetts, second edition.

Donald E. Knuth. 1973b. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1983. A note on hyphenation. *TUGboat*, 4(2):64, September.

Donald E. Knuth. 1986a. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1986b. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

Donald E. Knuth. 1988. The Errors of TEX. Technical Report STAN-CS-88-1223, Department of Computer Science, Stanford University, September.

Donald E. Knuth. 1989a. The Errors of TEX. *Software — Practice and Experience*, 19(7):607–685.

Donald E. Knuth. 1989b. Theory and practice. keynote address for the 11th World Computer Congress (Information Processing '89), August.

Donald E. Knuth. 1991. *3:16 Bible texts illuminated*. A-R Editions, Inc.

Donald E. Knuth. 1998. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, third edition.

Donald E. Knuth. 1999. *Digital Typography*. CSLI Lecture Notes 78. Center for the Study of Language and Information, Stanford, California.

Gabriele Kodydek and Martin Schönhacker. 2003. Si3Trenn and Si3Silb: Using the SiSiSi Word Analysis System for Pre-Hyphenation and Syllable Counting in German Documents. Lecture Notes in Artificial Intelligence LNCS/LNAI 2807, pages 66–73, České Budějovice, Czech Republic, September. Springer-Verlag.

Gabriele Kodydek. 2000. A Word Analysis System for German Hyphenation, Full Text Search, and Spell Checking, with Regard to the Latest Reform of German Orthography. In Sojka et al. (Sojka et al., 2000), pages 39–44.

Hanna Kołodziejska. 1987. Dzielenie wyrazów polskich w systemie TEX. Technical Report 165, Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego.

Hanna Kołodziejska. 1988. Le traitement des textes polonais avec le logiciel TEX. *Cahiers GUTenberg*, (0):3–10, April.

Helmut Kopka. 1991. *LATEX—Erweiterungsmöglichkeiten mit einer Einführung in METAFONT*. Addison-Wesley Verlag, Bonn, Germany, second edition.

András Kornai. 1999. *Extended Finite State Models of Language*. Cambridge University Press.

Cvetana Krstev. 1991. Serbo-Croatian hyphenation: a TEX point of view. *TUGboat*, 12(2):215–223, June.

Gerard D.C. Kuiken. 1990. Additional Hyphenation Patterns. *TUGboat*, 11(1):24–25, April.

Ladislav Lhotka. 1991. České dělení pro TEX (Czech Hyphenation for TEX). *CSTUG Bulletin*, (4):8–9, April.

Franklin M. Liang and Peter Breitenlohner. 1991. `PATtern GENeration` program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.0 from UNIXTEX distribution at `ftp://ftp.cs.umb.edu`, November.

Franklin M. Liang and Peter Breitenlohner. 1999. `PATtern GENeration` program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.3 from web2c distribution on CTAN.

Franklin M. Liang. 1981. TEX and hyphenation. *TUGboat*, 2(2):19–20, July.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Department of Computer Science, Stanford University, August.

John W. Lloyd. 2003. Learning Comprehensible Theories from Structured Data. In S. Mendelson and A.J. Smola, editors, *Advanced Lectures on Machine Learning, LNAI 2600*, pages 203–225.

Qing Ma, Hitoshi Isahara, and Hiromi Ozaku. 1996. Automatic part-of-speech tagging of thai corpus neural networks. In *Lecture Notes in Computer Science 1112*, pages 275–280. Springer-Verlag.

David Macháček. 2003. Přebíjející vzory ve zpracování přirozeného jazyka (Competing Patterns in Natural Language Processing). Master's thesis, Masaryk University in Brno, Faculty of Informatics, Brno, Czech Republic.

Pierre A. MacKay. 1988. Turkish hyphenations for TEX. *TUGboat*, 9(1):12–14, April.

Basil Malyshev, Alexander Samarin, and Dimitri Vulis. 1991a. Russian TEX. *Cahiers GUTenberg*, 10–11:1–6, September.

Basil Malyshev, Alexander Samarin, and Dimitri Vulis. 1991b. Russian TEX. *TUGboat*, 12(2):212–214, June.

Shibayama Mamoru and Hoshino Satoshi. 2001. Thai Morphological Analyses Based on the Syllable Formation Rules. *Journal of Information Procesing*, 15(04–007).

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Olga Martincová, Zdeněk Hlavsa, Zdenka Hrušková, Jiřina Hůrková, Jiří Kraus, Alena Polívková, Miloslav Sedláček, Ivana Svobodová, and Věra Vlková. 1993. *Pravidla českého pravopisu (The Rules of the Czech Spelling)*. Pansofia Praha.

David Megginson. 1998. *Structuring XML Documents*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Surapant Meknavin, Paisarn Charoenpornsawat, and Boonserm Kijsirikul. 1997. Feature-based Thai Word Segmentation. In *Proceedings of the Natural Language Processing Pacific Rim Symposium (NLPRS 1997)*, pages 41–46.

Ryszard Spencer Michalski, Jaime Guillermo Carbonell, and Tom Michael Mitchell. 1983a. *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto.

Ryszard Spencer Michalski, Jaime Guillermo Carbonell, and Tom Michael Mitchell. 1983b. *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann Publishers, Inc., Los Altos, California.

Tom Michael Mitchell. 1997. *Machine Learning*. McGraw-Hill.

Frank Mittelbach and Chris Rowley. 1992a. The future of high quality typesetting: structure and design. In Zlatuška (Zlatuška, 1992), page 255.

Frank Mittelbach and Chris Rowley. 1992b. The pursuit of quality — How can automated typesetting achieve the highest standards of craft typography? In C. Vanoirbeek and G. Coray, editors, *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography, Lausanne, Switzerland, 1992*, pages 261–273, New York. Cambridge University Press.

Mehryar Mohri, Fernando C. N. Pereira, and Michael D. Riley. 1998. FSM Library — General-purpose finite-state machine software tools. `http://www.research.att.com/sw/tools/fsm/`.

Mehryar Mohri. 1996. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2:61–80.

Originally appeared in 1994 as Technical Report, Institut Gaspard Monge, Paris.

Mehryar Mohri. 1997. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, June.

Mehryar Mohri. 2000. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, March.

Masaki Murata, Qing Ma, and Hitoshi Isahara. 2002. Comparision of Three Machine-Learning Methods for Thai Part-of-Speech Tagging. *ACM Transactions on Asian Language Information Processing*, 1(2):145–158.

Zuzana Nevěřilová and Petr Sojka. 2005. XML-Based Flexible Visualisation of Networks: Visual Browser. Submitted.

Stefan Nilsson and Gunnar Karlsson. 1999. IP-Address Lookup Using LC-Tries. *IEEE Journal on Selected Areas in Communications*, 17(6):1083–1092.

NTS-L. 1992–1995. New typesetting system discussion list. Archived in CTAN/digests/nts-l/.

Walter Obermiller. 1991. TEX in Germany. *TUGboat*, 12(2):211–212, June.

Karel Oliva, Milena Hnátková, Vladimír Petkevič, and Pavel Květoň. 2000. The Linguistic Basis of a Rule-Based Tagger of Czech. In Sojka et al. (Sojka et al., 2000), pages 3–8.

Petr Olšák. 1997. *TEXbook naruby (in Czech)*. Konvoj, Brno, Czech Republic.

Karel Pala, Pavel Rychlý, and Pavel Smrž. 1997. DESAM — Annotated Corpus for Czech. pages 523–530, Milovy, November. Springer-Verlag.

Hubert Partl. 1988. German TEX. *TUGboat*, 9(1):70–72, April.

Hubert Partl. 1990. How to make TEX and LATEX international. In J. Nadrchal, editor, *Man-Machine Interface in the Scientific Environment. Proceedings of the 8th European Summer School on Computing Techniques in Physics. Skalský Dvur, Czecholsovakia, 19–28 September 1989*, volume 61 of *Computer Physics Communications*, pages 190–200, Amsterdam, The Netherlands. European Summer Schools on Computing Techniques in Physics, North-Holland Publishing Company; Elsevier Science Publishers B. V. In addition to the papers presented at the colloquium, each paper is followed by a summary of the discussion about that paper.

Jan Pazdziora. 2005. `TeX::Hyphen` – hyphenate words using TEX's patterns. `http://search.cpan.org/src/JANPAZ/TeX-Hyphen-0.140/`.

Lev Pevzner and Marti A. Hearst. 2002. A Critique and Improvement of an Evaluation Metric for Text Segmentation. *Computational Linguistics*, 28(1):19–36.

Charles P. Pfleger. 1973. State Reduction in Incompletely Specified Finite-State Machines. *IEEE Trans. Computers C*, 22(4):1099–1102.

John Plaice and Yannis Haralambous. 1996. The latest developments in Ω. *TUGboat*, 17(2):181–183, June.

John Plaice. 1993. Language-dependent ligatures. *TUGboat*, 14(3):271–274, October.

John Plaice. 1994a. Progress in the Omega Project. In Goossens (Goossens, 1994), pages 190–193.

John Plaice. 1994b. Progress in the Omega project. *TUGboat*, 15(3):320–324, September.

John Plaice. 1998. pdftex email discussion list. `http://www.tug.org/archives/pdftex/msg01913.html`, 14 March.

Bernd Raichle. 1995. Kurzbeschreibung – `german.sty` (version 2.5), April. Available from CTAN archives.

Bernd Raichle. 1997. Hyphenation patterns for words containing explicit hyphens. `CTAN/language/hyphenation/hypht1.tex`.

Jan Rejzek. 2001. *Etymologický slovník českého jazyka (Ethymological Dictionary of Czech Language)*. LEDA, Prague, Czech Republic.

Pedro de Rezende. 1987. Portuguese hyphenation table for TEX. *TUGboat*, 8(2):102–102, July.

Emmanuel Roche and Yves Schabes. 1995. Deterministic Part-of-Speech Tagging. *Computational Linguistics*, 21(2):227–253.

Emmanuel Roche and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press.

Jan Michael Rynning. 1991. Swedish Hyphenation for TEX. Received in electronic form from author via email `mailto:jmr@nada.kth.se`, November.

Kauko Saarinen. 1988. Experiences with TEX in Finland. In Thiele (Thiele, 1988), pages 189–194.

Kai Salomaa, Derick Wood, and Sheng Yu, editors. 2000. *Implementing Automata*, volume 231.

Alexander Samarin and A. Urvantsev. 1991. CyrTUG, le monde TEX en cyrillique. *Cahiers GUTenberg*, 12:71–74, December.

Kevin Patrick Scannell. 2003. Hyphenation patterns for minority languages. *TUGboat*, 24(2):236–239.

Michail I. Schlesinger and Václav Hlaváč. 2002. *Ten Lectures on Statistical and Structural Pattern Recognition*. Kluwer Academic Publishers, Dordrecht, The Netherlands, May.

Joachim Schrod. 1991. An International Version of MakeIndex. *Cahiers GUTenberg*, 10–11:81–90, September.

Bernd Schulze. 1984. German hyphenation and *Umlaut*s in TEX. *TUGboat*, 5(2):103, November.

Radek Sedláček. 1999. Morphological Analyzer of Czech (in Czech). Master's thesis, Masaryk University in Brno, April.

Terry J. Sejnowski and C. R. Rosenberg. 1987. Parallel Networks that Learn to Pronounce English Text. *Complex Systems*, 1:145–168.

Yong Shi, Weixuan Xu, and Zhengxin Chen, editors. 2004. *Data Mining and Knowledge Management: Chinese Academy of Sciences Symposium*

CASDMKM, volume LNCS 3327 of *Lecture Notes in Computer Science*. Springer-Verlag, July.

Max Silberztein. 2000. INTEX: an FST toolbox. *Theoretical Computer Science*, 234:33–46.

Alan Silverstein. 2002. Judy IV Shop Manual. `http://judy.sourceforge.net/application/shop_interm.pdf`.

Karel Skoupý. 1998. NTS: a New Typesetting System. *TUGboat*, 18(3):318–322.

Pavel Smrž and Petr Sojka. 1996. Word Hy-phen-a-tion by Neural Networks. Technical Report FIMU-RS-96-04, Masaryk University in Brno, Faculty of Informatics, August.

Pavel Smrž and Petr Sojka. 1997. Word Hy-phen-a-tion by Neural Networks. *Neural Network World*, 7:687–695.

Pavel Smrž. 1995. Learning Algorithms of Neural Networks. Master's thesis, Masaryk University in Brno, April.

Petr Sojka and David Antoš. 2003. Context Sensitive Pattern Based Segmentation: A Thai Challenge. pages 65–72, Budapest, April.

Petr Sojka and Pavel Ševeček. 1994. Hyphenation in TEX — Quo Vadis? In Włodek Bzyl and Tomek Przechlewski, editors, *Proceedings of the 9th European TEX Conference, Gdańsk, 1994*, pages 59–68, September.

Petr Sojka and Pavel Ševeček. 1995a. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289.

Petr Sojka and Pavel Ševeček. 1995b. Hyphenation in TEX — Quo Vadis? In Michel Goossens, editor, *Proceedings of the TEX Users Group 16th Annual Meeting, St. Petersburg, 1995*, pages 280–289, Portland, Oregon, U.S.A. TEX Users Group.

Petr Sojka, Ivan Kopeček, and Karel Pala, editors. 2000. *Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000*, Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, Brno, Czech Republic, September. Springer-Verlag.

Petr Sojka. 1995a. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

Petr Sojka. 1995b. Notes on Compound Word Hyphenation in TEX. Technical Report FIMU-RS-95-04, Masaryk University in Brno, Faculty of Informatics, August.

Petr Sojka. 1998a. An Experience from a Digitization Project. *Cahiers GUTenberg*, (28–29):276–281, March.

Petr Sojka. 1998b. Publishing Encyclopaedia with Acrobat using TEX. In *Towards the Information-Rich Society. Proceedings of the ICCC/IFIP conference Electronic publishing '98*, pages 217–222, Budapest, Hungary, April. ICCC Press.

Petr Sojka. 1999a. Hyphenation on Demand. *TUGboat*, 20(3):241–247.

Petr Sojka. 1999b. Hyphenation on Demand. pages 1085–1091, Vancouver, August. The University of British Columbia.

Petr Sojka. 2000. Competing Patterns for Language Engineering. In Sojka et al. (Sojka et al., 2000), pages 157–162.

Petr Sojka. 2003a. Animations in PDF. In *Proceedings of the 8th SIGCSE Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2003*, page 263, Thessaloniki. Association of Computing Machinery.

Petr Sojka. 2003b. Interactive Teaching Materials in PDF using JavaScript. In *Proceedings of the 8th SIGCSE Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2003*, page 275, Thessaloniki. Association of Computing Machinery.

Petr Sojka. 2003c. Rapid Evaluation using Multiple Choice Tests and TEX. In *Proceedings of the 8th SIGCSE Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2003*, page 265, Thessaloniki. Association of Computing Machinery.

Petr Sojka. 2004. Slovenské vzory dělení: čas pro změnu? (Slovak Hyphenation Patterns: A Time for Change?). *CSTUG Bulletin*, 14(3–4):183–189.

Virach Sornlertlamvanich, Thatsanee Charoenporn, and Hitoshi Isahara. 1997. ORCHID: Thai Part-Of-Speech Tagged Corpus. Technical Report TR-NECTEC-1997-001, Thai National Electronics and Computer Technology Center, Thailand, December. `http://www.links.nectec.or.th/`.

Virach Sornlertlamvanich, Thatsanee Charoenporn, and Hitoshi Isahara. 1999. Building a Thai Part-Of-Speech Tagged Corpus. *The Journal of the Acoustical Society of Japan (E)*, 20(3):140–189, May.

Virach Sornlertlamvanich, Tanapong Potipiti, and Thatsanee Charoenporn. 2000. Automatic Corpus-Based Thai Word Extraction with the C4.5 Learning Algorithm. In *COLING*, pages 802–807. Morgan Kaufmann.

Virach Sornlertlamvanich. 1998. *Probabilistic Language Modeling for Generalized LR Parsing*. Ph.D. thesis, Department of Computer Science, Tokyo Institute of Technology, September.

Wilhelm Steiner. 1995. *Automatische Silbentrennung durch Wortbildungsanalyse*. Ph.D. thesis, Technisch-Naturwissenschaftliche Fakultät.

Rattasit Sukhahuta and Dan Smith. 2001. Information Extraction Strategies for Thai Documents. *International Journal of Computer Processing of Oriental Languages (IJCPOL)*, 14(2):153–172.

Robert S. Sutor and Angel L. Díaz. 1998. IBM techplorer: Scientific Publishing for the Internet. *Cahiers GUTenberg*, (28–29):295–308, March.

Philip Taylor. 1992. The Future of TEX. In Zlatuška (Zlatuška, 1992), pages 235–254.

Hàn Thế Thành, Petr Sojka, and Jiří Zlatuška. 1996. TEX2PDF — Acrobatics with an Alternative to DVI Format. *TUGboat*, 17(3):244–251.

Christina Thiele, editor. 1988. *Proceedings of the TEX Users Group 9th Annual Meeting, Montréal, 1988*, Portland, Oregon, U.S.A. TEX Users Group.

Anders Thulin. 1987. More hyphenation exceptions. *TUGboat*, 8(1):76–76, April.

Alan Turing. 1950. Computing Machinery and Intelligence. *Mind*, (59):433–460.

Andrew J. Viterbi. 1967. Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267, April.

Dimitri Vulis. 1989. Notes on Russian TEX. *TUGboat*, 10(3):332–336, November.

Bruce W. Watson. 1999. Implementing and using finite automata toolkits. (Kornai, 1999), pages 19–36.

Steve Young and Gerrit Bloothooft, editors. 1997. *Corpus-Based Methods in Language and Speech Processing*. Kluwer Academic Publishers Group, Dordrecht.

Chengqi Zhang and Shichao Zhang. 2002. *Association Rule Mining*, volume LNAI 2307 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Austin Ziegler. 2005. `Text::Hyphen` package. `http://rubyforge.org/projects/text-format`.

Jiří Zlatuška. 1991. Automatic generation of virtual fonts with accented letters for TEX. *Cahiers GUTenberg*, 10–11:57–68, September.

Jiří Zlatuška, editor. 1992. *Proceedings of the 7th European TEX Conference, Prague, 1992*, Brno, September. Masarykova Universita.

Author Index

Subject Index