Petr Sojka; Ondřej Sojka

The Unreasonable Effectiveness of Pattern Generation

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 29 (2019), No. 1-4, 73–86

Persistent URL: `http://dml.cz/dmlcz/150175`

The Unreasonable Effectiveness of Pattern Generation

Petr Sojka, Ondřej Sojka

Languages are constantly evolving, and so are their hyphenation rules and needs. The effectiveness and utility of TEX's hyphenation have been proven by its usage in almost all typesetting systems in use today. The current Czech hyphenation patterns were generated in 1995, and no hyphenated word database was freely available.

We have developed a new Czech word database and have used the `patgen` program to generate new effective Czech hyphenation patterns efficiently and evaluated their generalization qualities. We have achieved full coverage on the training dataset of 3,000,000 words, and developed a validation procedure of new patterns for Czech based on the testing database of 105,000 words approved by the Czech Academy of Science linguists.

Our pattern generation case study exemplifies a practical solution to the widespread dictionary problem. The study has proven the versatility, effectiveness, and extensibility of Liang's approach to hyphenation developed for TEX. The unreasonable effectiveness of the pattern technology has led to applications that are and will be used, even more widely now, nearly 40 years after its inception.

Keywords: `patgen`, hyphenation patterns, unreasonable effectiveness, Czech

...the best approach appears to be to embrace the complexity of the domain and address it by harnessing the power of data: if other humans engage in the tasks and generate large amounts of unlabeled, noisy data, new algorithms can be used to build high-quality models from the data. (Peter Norwig, [1])

Introduction

In their famous essays, Wigner [2], Hamming [3] and Norwig [1] consider mathematical and data-driven approaches to be miraculously, unreasonably effective. One of the very first mathematically founded approaches that harnessed the power of data was Franklin Liang's language-independent solution for TEX's hyphenation algorithm [4], and his program `patgen` for a generation of hyphenation patterns from a word list.

Dictionary problem The task at hand was a *dictionary problem.* A dictionary is a database of records; in each record, we distinguish the key part (the word) and the data part (its division). Given an already hyphenated word list of a language, a set of *patterns* is magically generated. Hyphenation patterns are much smaller than the original word list and typically encode almost all hyphenation points in the input list without mistakes. Liang's pattern approach thus could be viewed as an efficient lossy, ideally lossless, *compression* of the hyphenated dictionary with a compression ratio of several orders of magnitude.

It has been proven [5, chapter 2] that the optimization problem of exact lossless pattern minimization is non-polynomial by reduction to the minimum set cover problem.

Generated patterns have minimal length, e.g., shortest context possible, which results in their *generalization* properties. Patterns could hyphenate words not seen during learning: yet another miracle of the generated patterns.

Pattern preparation In the 36 years of `patgen` use, there have been hundreds of hyphenation patterns created, either by hand or *generated* by the program `patgen`, or by the combination of both methods [6]. The advantage of *pattern generation* is that one can fine-tune pattern qualities for specific usage. Having an open-source and maintained word list adds another layer of flexibility and usability to the deployment of patterns. This approach is already set up for German variants and spellings [7] and was an inspiration for doing the same for the Czech language.

In this paper, we report on the development of the new Czech word list with a free license and complementary sets of hyphenation patterns. We describe the iterative process of initial word list preparation, word form collection, estimation of pattern generation parameters, and novel applications of the technology.

Hyphenation is neither anarchy nor the sole province of pedants and pedagogues. Used in moderation, it can make a printed page more visually pleasing. If used indiscriminately, it can have the opposite effect, either putting the reader off or causing unnecessary distraction. (Major Keary)

Initial word list preparation

As a rule of thumb, the development of a large new hyphenated word list starts with a small dataset. The experience and outputs from this initial phase, e.g., hyphenation patterns, are then applied to the larger and larger lists.

Bootstrapping idea As word lists of a well-established language are sizeable and manual creation of a huge hyphenated word list is tedious work, we used the bootstrapping technique. We illustrate the process of initial word list preparation in the diagram in Figure 1 on the facing page. We have obtained a hyphenated

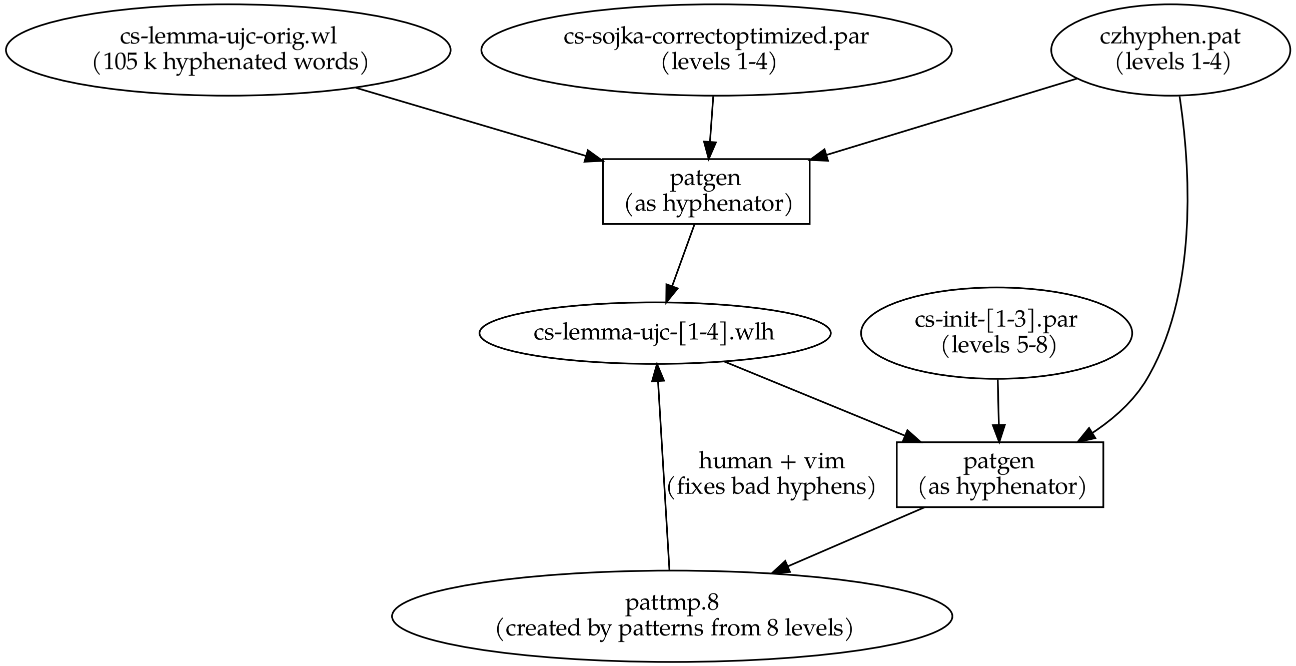

Figure 1: Life cycle of initial word list preparation, illustrated with the development of 105 k Czech consistently hyphenated words. `czhyphen.pat` represents the original Czech hyphenation patterns from [8] and `cs-sojka-correctoptimized.par` are correct optimized `patgen` parameters from the same paper. `cs-init-[1-3].par` are custom parameters that trade off bad hyphens (which have to be manually checked) for missed hyphens. Information on which hyphenations `patgen` missed, and where it wrongly inserted a hyphen is sourced from `pattmp`.

word list with 105,244 words from the Czech Academy of Sciences, Institute of the Czech Language (ÚJČ). Upon closer inspection, we discovered many problems with the data, probably stemming from the fact that it has been crafted by multiple linguists over many years. The few hyphenation rules [9] that are in the Czech language are not applied consistently. The borderline cases were typically between syllabic (ro-zum) and etymological variants (roz-um) of hyphenation, or the way of handling words borrowed from German or English into Czech. There are sporadic examples of words, where correct syllabification depends on the semantics of the word: *narval* and *oblít* are two examples of them in Czech. These are preferably not to be hyphenated, to stay on the safe side.

It is impractical to try to manually find inconsistencies and systemic errors, even in a relatively short word list like this. We slightly modified and extended the process suggested in [10, page 242]: We used `patgen` and the current Czech patterns to hyphenate the word list and manually checked only the 25,813 words, where the proposed hyphenation points differed from the official (were bad or missed), creating a new word list `cs-lemma-ujc-1.wlh` [11] in the process.

However, we are erroneous humans making mistakes. To find these mistakes, we have used `patgen` to generate the four additional levels of hyphenation patterns on top of the current patterns from the checked word list. We have also adjusted the parameters (see `cs-init-[1-3].par` [11]) used for generation of the four additional levels to trade off bad hyphens (which have to be manually checked) for missed ones. We have then used these patterns, with eight levels in total, to hyphenate the checked word list and manually rechecked the wrongly hyphenated points (dots in `patgen` output), with missed hyphenation points (implicitly marked as the hyphen sign in hyphenated word list). We have repeated this process three times, iterating on `cs-lemma-ujc-[2-4].wlh`. Word list number four is used for the generation of bootstrapping patterns and final pattern validation.

Word list preparation and design

Any live language continually changes, and Czech is no exception. Many new Czech words now come from other languages, mostly from English. It presents a challenge for the patterns; they must not only correctly hyphenate Czech words according to Czech syllabic boundaries, but foreign words must be hyphenated correctly too, according to their new Czech syllabic pronunciation [12]. To have the patterns keep up with language evolution, we must maintain not only the patterns but also a hyphenation word list. In this section, we detail how we have built such a word list.

csTenTen corpus We have first obtained a word list with frequencies, generated from the Czech Web Corpus of TenTen family (csTenTen) [13]. We then filtered this word list to include only words that appear more than ten times in two crawls [14] made in years 2012 and 2017. We ended up with a word list containing 922,216 words, a non-negligible fraction of which are misspellings and jargon.

Word list cleanup We have then cleaned this word list by using the Czech morphological analyzer `majka` [15] to remove all words not known to it. We removed 370,291 typos, misspellings, and similar atypical lexemes, and we kept only 551,925 frequently occurring valid words in the dataset.

Word list expansion The morphological analyzer `majka` [15] also allows us to expand words into all their inflected forms. We chose not to use the expansion feature of `majka`, because the word list would grow to 3,779,379 (almost a fourfold increase) and csTenTen already contains most of the commonly used types of inflections. It would also distort which hyphenation `patgen` gives the most weight to. We tried supplying logarithms of word frequencies from csTenTen to the word list, so more weight could be given to patterns that cover the most common words. It did not significantly improve validation scores in our case, as one can see in Table 3 on page 81. We think that this is partly because `patgen` is limited to one digit of frequency per word and partly because the validation score (computed from error rate on `ujc` word list) does not capture real-world usage.

We expanded the word list with `majka` by adding 54,569 lemmas (base forms) that were present in the word list, but not in their base form. It increased the word list size to 606,494 words.

We list the statistics of word lists used in pattern generation in Table 1.

Table 1: Czech word list shortcut names and statistics

shortcut	word list description	count
`ujc`	checked word list for validation	105,244
`all`	all frequent word forms from web known to `majka` plus all lemmas known to `majka`	606,494
`allflex`	previous plus all word forms generated by majka	2,100,581
`allflexjargon`	previous plus all non-standard and jargon word forms	3,779,379
`biggest`	tokens that are present in the csTenTen more than 10 times	3,918,054

Maintenance The German *wortliste* [7] project served as inspiration for our open word list format, detailed in the `README.md` [11].

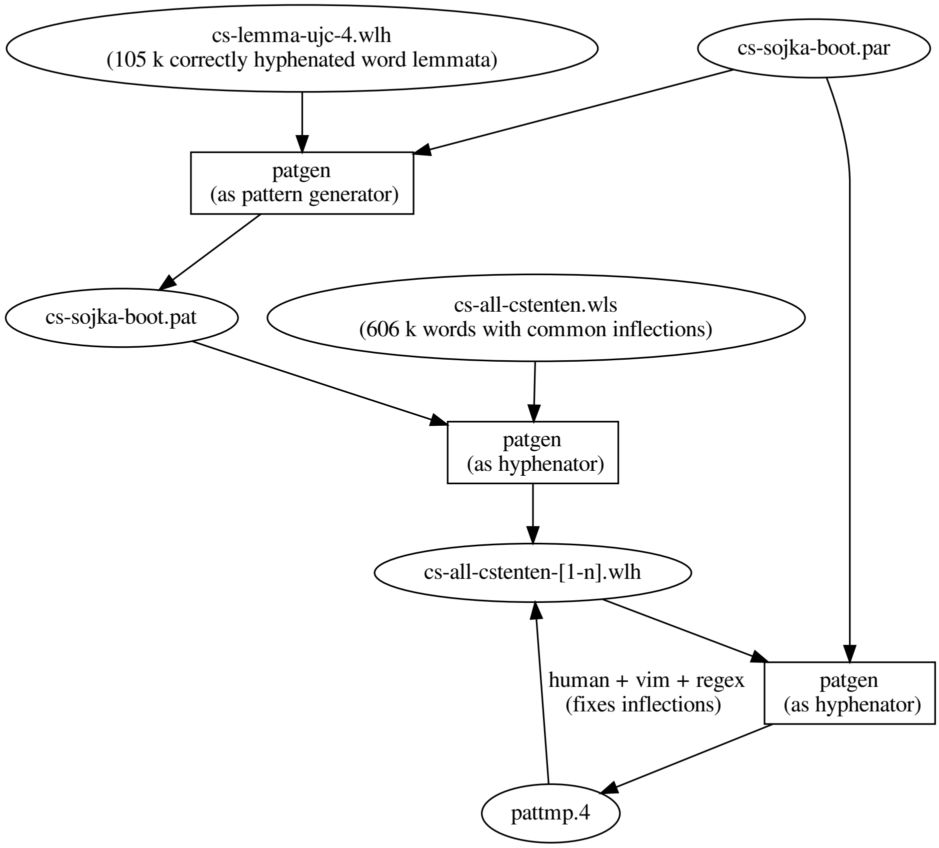

Figure 2: How we bootstrapped hyphenation of the big word list by training patterns (`cs-sojka-boot.pat`) on the small word list and applying them to the big one. `cs-sojka-boot.par` are `patgen` parameters that are designed to generate many patterns but still retain their generalization properties. `pattmp` highlights which hyphenation points in the source file the new pattern level missed, which were correctly covered and where they wrongly put a hyphen.

One must regard the hyphen as a blemish to be
avoided wherever possible. (Winston Churchill)

Bootstrapping — iterative development of hyphens in the big word list

It would be tedious to hyphenate such a big word list by hand manually, so we train patterns on a small list and apply them to the big word list, as illustrated in Figure 2 on the preceding page. Then, we train patterns on the (now hyphenated) big word list and have `patgen` show what it would have hyphenated differently. With this approach, we cherry-pick inconsistencies in the word list.

Since the big word list contains not only lemmas of words, but also characteristic inflections, we use regular expressions to add hyphens around them and fix inconsistencies. We keep iterating on this, as shown in Figure 2 on the facing page, until the patterns, generated with `cs-init-[1-3].par` [11], achieve nearly perfect coverage.

The resulting patterns hyphenate according to the standard Czech hyphenation rule: hyphenation is allowed whenever it does not change the pronunciation of the word. Thanks to the effectiveness of pattern generation, this works not only for Czech words but also for foreign (Latin, French, German, English) ones.

Hyphens, like cats, are capable of arousing tenderness or shudders. (Pamela Frankau)

Pattern generation

The last Czech hyphenation patterns were generated in 1995 [8], and are in use not only in TeX but also in other widespread typesetting systems. For conservative users, there is no strong incentive for change, because the error rate is relatively low (the first version of the validation set measured an error rate around 4%), and coverage is relatively high (the first version of the validation set measured around 7% missed hyphenation points).

Pattern generation from 3,000,000 words does not take hours as it did two decades ago, but seconds, even on commodity hardware, which allows for rapid development of “home-made” patterns.

In each training pass, `patgen` generates all possible substrings of every word in its dictionary with length higher than or equal to the parameter `pat_start` but lower than `pat_finish`. Then, for each generated pattern, it counts how many hyphenation points would be correctly covered by the pattern and how many hyphenation points would the patterns wrongly identify. Statistics of Good, Bad and Missed represent the count of points, where the patterns correctly identified a hyphen, where the patterns expected a hyphen but there was none, and the count of hyphenation points that the patterns did not identify.

We have developed a Python wrapper for `patgen` that we use in Jupyter notebooks. It allows rapid iteration, and easy sharing of results — see Table 2 and `demo.ipynb` [11].

Had Liang in 1983 had the same ease of changing `patgen` parameters, running it, and seeing the results in 60 seconds, he would have inevitably generated higher than 89% coverage while staying within the limit of 5,000 patterns [4, page 37].

Table 2: Outputs from running `patgen` in our Jupyter notebook with two different parameter sets. The first parameter set is from the German Trennmuster project [7] and generates 7,291 patterns, 40 kB. The second one from [8] generates shorter and smaller patterns — 4,774 patterns, 25 kB.

Level	Patterns	Good	Bad	Missed	Lengths
1	750	1,683,529	525,670	0	1 5
2	3,178	1,628,874	38	54,655	2 6
3	2,548	1,683,528	9,931	1	3 7
4	1,382	1,683,287	0	242	4 8
5	92	1,683,528	0	1	5 9
6	0	1,683,528	0	1	6 10
7	1	1,683,529	0	0	7 11

Level	Patterns	Good	Bad	Missed	Lengths
1	1,608	1,655,968	131,481	27,561	1 3
2	1,562	1,651,840	2,533	31,689	1 3
3	2,102	1,683,528	2,584	1	2 5
4	166	1,683,135	6	394	2 5

It has also become common to use a validation dataset to ensure generalization abilities. Our usage of a validation dataset has proven useful. Table 3 shows that if we were to use the *correct optimized* parameters from [8] that have been in use for Czech, we would overfit on the training dataset and perform *worse* than their *size optimized* counterparts. The validation word list has to be carefully checked with linguists from UJČ for consistency to minimize the generalization error. Most of the current errors stem from foreign words used in Czech texts.

When the validation word list is added to training, then patterns could be developed to serve as a lossless compression of word list dataset and thus maximize

Table 3: Effectiveness and effectivity of pattern generation on Czech word lists. Comparison of validation scores of patterns trained on various word list and parameter combinations. Percentages sum up to more than 100%, because they are calculated by dividing the count of Good, Bad or Missed hyphens by the total word count.

Word list	Params	Good %	Bad %	Missed %	Size	Patterns	Time (s)
`all`	correctopt [8]	99.76	2.94	0.24	30 kB	5,593	58.13
	sizeopt [8]	98.95	2.80	1.05	19 kB	3,816	59.46
	german [7]	99.74	2.21	0.26	51 kB	8,991	201.9
weighted `all`	correctopt [8]	99.76	2.94	0.24	30 kB	5,590	59.23
	sizeopt [8]	98.95	2.80	1.05	20 kB	3,821	58.74
	german [7]	99.74	2.21	0.26	51 kB	8,978	207.35
`allflex`	correctopt [8]	99.46	4.02	0.54	28 kB	5,387	212.55
	sizeopt [8]	99.26	3.72	0.74	29 kB	5,537	212.59
	german [7]	99.42	3.35	0.58	49 kB	8,663	1,035.16
`allflexjargon`	correctopt [8]	99.47	4.08	0.53	29 kB	5,612	365.96
	sizeopt [8]	99.31	3.78	0.69	31 kB	5,938	369.92
	german [7]	99.43	3.36	0.57	53 kB	9,308	1,786.4

the effectiveness of the pattern technology. See results with parameter set *german* for an example of almost lossless compression.

Life is the hyphen between matter and spirit. (Augustus William Hare)

The unreasonable effectiveness

We were able to solve the dictionary problem for Czech hyphenation effectively.

Space effectiveness From 3,000,000+ hyphenated words stored in approximately 30,000,000 bytes we have produced patterns of size 30,000 bytes, achieving roughly 1000× space *lossless* compression.

Time effectiveness Using the trie data structure for patterns makes the time complexity of accessing the record related to the word, e.g. hyphenation point, very low *constant* time. The constant is related to the depth of the pattern trie data structure, e.g. 5 or 6 in the case of Czech. If the entire pattern trie resides in RAM, the time for finding the patterns for a word is on the scale of tens, at most hundreds, of processor instructions. Word hyphenation throughput is then about 1,000,000 words per second on a modern CPU.

Optimality Even though finding exact space and time-optimal solutions is not feasible, finding an approximate solution close to optimum is possible. Heuristics and insight expressed above, together with interactive fine-tuning of `patgen` parameter options, allows for rapid pattern development.

Automation A close-to-optimal solution to the dictionary problem could be useful not only for Czech hyphenation, but for all other languages [6, 16], and more generally, for other instances of the dictionary problem. Heuristics for thresholding `patgen` pattern generation parameters could be based on a statistical analysis of large input datasets. It could allow the deployment of presented approaches on a much broader problem set and scale. We believe that parameters could be approximated *automatically* from the statistics of the input data.

Pattern generation — in Wigner's words — "has proved accurate beyond all reasonable expectations". Let us paraphrase another one of his quotes:

> The miracle of the appropriateness of the language of ~~mathematics~~ *patterns* for the formulation of the laws of ~~physics~~ *data* is a wonderful gift which we neither understand nor deserve. We should be grateful for it and hope that it will remain valid in future research and that it will extend, for better or for worse, to our pleasure, even though perhaps also to our bafflement, to wide branches of learning.

“We should stop acting as if our goal is to author extremely elegant theories, and instead embrace complexity and make use of the best ally we have: the unreasonable effectiveness of data.” (Peter Norvig, [1])

Conclusion

We have developed a flexible open language–independent system [11] for hyphenation pattern generation. We have demonstrated the effectiveness of this system by updating the old Czech hyphenation patterns [8] and achieving record accuracy. We have also applied recent data and computer science advancements, such as the usage of interactive Jupyter notebooks and a validation dataset to prevent overfitting, to the more than three decades old problem of pattern generation.

Future work

Word lists for other languages The logical next steps will be applying developed techniques to different languages: to Slovak and virtually all others that do not yet have word list hyphenation patterns based on a word list, but for which a word list — either in Sketch Engine or elsewhere — is available.

Stratification Pattern generation could be further sped up by several techniques, such as the stratification of word lists on the level of input, or on the level of counting pro and con examples for including a new pattern or not.

Pattern-encoded spellchecker We have a big dictionary of frequent spelling errors from the csTenTen word list. Nothing prevents us from encoding these into specific patterns or pattern layers with extra levels and using that information during typesetting, e.g. to typeset misspelled words with a red underline in LuaTEX. LuaTEX allows dynamic pattern loading and Lua programming that enables the implementation of this feature, which people are used to having in editors.

Word segmentations Recent progress in machine-learned natural language processing and machine translation builds on subword representations and various types of semantically coherent sentence or word segmentations. As tokenization and segmentation are at the beginning of every natural language processing pipeline, there is a demand for effective and efficient universal segmentation [17]. New neural machine translation systems are capable of open-vocabulary translation by representing rare and unseen words as a sequence of subword units [18, Table 1]. Segmentation is crucial, especially for compositional languages like German, where there are many compounds (mostly out of vocabulary words), and for morphologically rich languages like Hebrew [19] or Arabic, that need to be segmented, represented, and translated.

Pattern-based learnable key memories Solutions to variations of the dictionary problem are a hot topic of leading-edge research to design memory data architectures like those used in machine learning of language [20]. Pattern-based memory network architectures could speed up language data access in huge neural networks considerably.

Multilingual hyphenation patterns Given that there are close languages with syllabic-based rules like Czech and Slovak, generating patterns from merged word lists is straightforward. It would save energy on low-resource devices like e-book readers by having them load fewer patterns at a time.

Acknowledgments The authors thank the CSTUG and the TEX Users Group who financially supported the presentation of the project at TUG 2019. We owe our gratitude also to Vít Suchomel of Lexical Computing for word lists from Sketch Engine, to Pavel Šmerk, Frank Liang and Don Knuth for `majka`, `patgen` and TEX, respectively. Thanks go to Vít Novotný and Pavel Šmerk for valuable comments to the paper. Thanks TUG for the permission to reprint the paper from *TUGboat*, with minor extensions, and Don Knuth for discussion after the presentation of the paper at TUG 2019.

References

1. PEREIRA, Fernando; NORVIG, Peter; HALEVY, Alon. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*. 2009, vol. 24, no. 02, s. 8–12. ISSN 1541-1672. Dostupné z DOI: `10.1109/MIS.2009.36`.
2. WIGNER, Eugene P. The Unreasonable Effectiveness of Mathematics in the Natural Sciences. Richard Courant Lecture in Mathematical Sciences delivered at New York University, May 11, 1959. *Communications on Pure and Applied Mathematics*. 1960, vol. 13, no. 1, s. 1–14. Dostupné z DOI: `10.1002/cpa.3160130102`.
3. HAMMING, R. W. The Unreasonable Effectiveness of Mathematics. *The American Mathematical Monthly*. 1980, vol. 87, no. 2, s. 81–90. ISSN 00029890, 19300972. ISSN 00029890, 19300972. Dostupné také z: `https://www.jstor.org/stable/2321982`.
4. LIANG, Franklin M. *Word Hy-phen-a-tion by Com-put-er*. 1983. Dostupné také z: `https://tug.org/docs/liang/`. Disertační práce. Stanford University.

5. SOJKA, Petr. *Competing Patterns in Language Engineering and Computer Typesetting*. 2005. Disertační práce. Faculty of Informatics.
6. REUTENAUER, Arthur; MIKLAVEC, Mojca. *TEX hyphenation patterns* [online]. TUG [cit. 2019-11-14]. Dostupné z: `https://tug.org/tex-hyphen/`.
7. LEMBERG, Werner. *A database of German words with hyphenation information*. Dostupné také z: `https://repo.or.cz/wortliste.git`.
8. SOJKA, Petr; ŠEVEČEK, Pavel. Hyphenation in TEX — Quo Vadis? *TUGboat*. 1995, vol. 16, no. 3, s. 280–289.
9. *Internetová jazyková příručka (Internet Language Reference Book)* [online]. Institute of Czech language, Czech Academy of Sciences [cit. 2019-07-18]. Dostupné z: `http://prirucka.ujc.cas.cz/?id=135`.
10. SOJKA, Petr. Hyphenation on Demand. *TUGboat*. 1999, vol. 20, no. 3, s. 241–247. `https://tug.org/TUGboat/tb20-3/tb64sojka.pdf`.
11. SOJKA, Ondřej; SOJKA, Petr. *cshyphen repository*. Dostupné také z: `https://github.com/tensojka/cshyphen`.
12. SOJKA, Petr. Notes on Compound Word Hyphenation in TEX. *TUGboat*. 1995, vol. 16, no. 3, s. 290–297.
13. JAKUBÍČEK, Milos; KILGARRIFF, Adam; KOVÁŘ, Vojtěch; RYCHLÝ, Pavel; SUCHOMEL, Vít. The TenTen Corpus Family. In: *Proc. of 7th International Corpus Linguistics Conference (CL)*. Lancaster, 2013, s. 125–127.
14. SUCHOMEL, Vít; POMIKÁLEK, Jan. Efficient Web Crawling for Large Text Corpora. In: KILGARRIFF, Adam; SHAROFF, Serge (eds.). *Proc. of the seventh Web as Corpus Workshop (WAC)*. Lyon, 2012, s. 39–43. Dostupné také z: `https://sigwac.org.uk/raw-attachment/wiki/WAC7/wac7-proc.pdf`.
15. ŠMERK, Pavel. Fast Morphological Analysis of Czech. In: SOJKA, Petr; HORÁK, Aleš (eds.). *Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2009*. Karlova Studánka, Czech Republic: Masaryk University, 2009, s. 13–16. ISBN 978-80-210-5048-8. Dostupné také z: `http://nlp.fi.muni.cz/raslan/2009/`.
16. SCANNELL, Kevin Patrick. Hyphenation patterns for minority languages. *TUGboat*. 2003, vol. 24, no. 2, s. 236–239.
17. SHAO, Yan; HARDMEIER, Christian; NIVRE, Joakim. Universal Word Segmentation: Implementation and Interpretation. *Transactions of the Association for Computational Linguistics*. 2018, vol. 6, s. 421–435. Dostupné z DOI: `10.1162/tacl_a_00033`.
18. SENNRICH, Rico; HADDOW, Barry; BIRCH, Alexandra. Neural Machine Translation of Rare Words with Subword Units. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*

(Volume 1: Long Papers). Berlin, Germany: Association for Computational Linguistics, 2016, s. 1715–1725. Dostupné z DOI: `10.18653/v1/P16-1162`.

19. ZELDES, Amir. A Characterwise Windowed Approach to Hebrew Morphological Segmentation. In: *Proc. of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Brussels, Belgium: Association for Computational Linguistics, 2018, s. 101–110. Dostupné z DOI: `10.18653/v1/W18-5811`.
20. LAMPLE, Guillaume; SABLAYROLLES, Alexandre; RANZATO, Marc'Aurelio; DENOYER, Ludovic; JÉGOU, Hervé. *Large Memory Layers with Product Keys* [online]. 2019 [cit. 2019-07-18]. Dostupné z arXiv: `1907.05242 [cs.CL]`.

Nepochopitelná efektivita generování vzorů dělení slov

Jazyky se vyvíjí a spolu s nimi i jejich potřeby a pravidla dělení slov. Mechanismus vzorů dělení slov v TEXu převzala většina dnešních sazebních systémů, což prokazuje jeho efektivitu a užitečnost. Současné vzory dělení slov pro češtinu ale vznikly v roce 1995, kdy ještě neexistovala žádná volně šiřitelná databáze slov.

Vyvinuli jsme novou českou databázi slov, použili jsme program `patgen` k vygenerování nových efektivních vzorů dělení slov pro češtinu a vyhodnotili jsme jejich generalizační schopnosti. Na trénovací datové sadě 3 milionů slov jsme dosáhli plného pokrytí. Dále jsme vyvinuli postup pro validaci nových vzorů dělení slov pro češtinu s využitím databáze 105 tisíc slov schválených lingvisty Akademie věd České republiky.

Naše případová studie generování vzorů dělení slov představuje praktické řešení častého slovníkového problému. Studie dokazuje pružnost, efektivitu a rozšiřitelnost Liangova přístupu k dělení slov vyvinutého pro TEX. Nepochopitelná efektivita mechanismu vzorů dělení slov dala vzniknout aplikacím, které ho využívají i téměř 40 let po jeho vzniku.

Klíčová slova: `patgen`, vzory dělení slov, nepochopitelná efektivita, čeština

Petr Sojka, Faculty of Informatics, Masaryk University, Brno, Czech Republic and CSTUG, sojka@fi.muni.cz
Ondřej Sojka c/o CSTUG, Nejedlého 1, 638 00 Brno, Czech Republic, ondrej.sojka@gmail.com