Hyph-bench: Benchmark Dataset of Hyphenated Words for Generating Hyphenation Patterns

Ondřej Metelka and Petr Sojka

Faculty of Informatics, Masaryk University, Brno, Czech Republic
514098@mail.muni.cz, sojka@fi.muni.cz

Abstract. The hyphenation algorithm, based on hyphenation patterns and developed primarily for TEX, is used almost exclusively by typesetting systems, web browsers, and other applications that require breaking text lines. The essence of minimal pattern generation lies in the NP-complete task of optimizing size and coverage in pattern generation from a word list of hyphenated words, while maintaining near 100% accuracy.
The problem of optimising pattern generation has already been studied for several Slavic languages; however, the heuristic setting of parameters for the pattern generation process is based on the quality and quantity of the hyphenated word list.

We have designed and collected the datasets of a hyphenated word list of Slavic and non-Slavic languages with the primary goal of benchmarking and optimization of pattern generation by `patgen`. We have set and computed baselines for included languages. We have prepared a benchmark dataset with baselines that often beat the currently widely used patterns in precision and/or recall. We are paving the road towards more accurate, smaller patterns with better and consistent coverage of word hyphenation in several Slavic and non-Slavic languages. The dataset also enables experiments in generating segmentation or hyphenations for language families, or even the universal, syllabic hyphenation patterns.

Keywords: hyphenation, effectiveness, hyphenation patterns, `patgen`, pattern generation, benchmark dataset, hyphenated wordlists, supervised learning

"Imitating the Grand Wizard, I invite you to Go Forth and make masterpieces of hyphenation patterns..." – Yannis Haralambous

1 Introduction and Related Work

Quality hyphenation matters. It allows for tighter, more uniform word spacing. It creates a "tighter," more visually pleasing rag. It is the only way to make text readable and aesthetically pleasing in narrow columns of newspapers, magazines, and mobile web browsers. It ensures correct word recognition (`ther-apist` instead of `the-rapist`).

A. Horák, P. Rychlý, A. Rambousek (eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2025, pp. 173–185, 2025.

In the domain of word hyphenation/segmentation, approaches have evolved from rule-based systems and pattern matching to statistical, machine learning methods, including neural. We categorize them chronologically as follows:

Pattern-Based Generation (The Baseline) The *de facto* standard for typographic hyphenation is the algorithm developed by Liang for the TEX typesetting system [7]. This approach utilizes the `patgen` program to generate a set of weighted hyphenation patterns from a list of hyphenated words. The algorithm employs a dynamic packed trie data structure to efficiently store patterns and identify hyphenation points based on local context windows. It operates on a multilevel system where patterns can either enable or inhibit hyphenation based on weighted values. This method is renowned for its exceptional space efficiency and high coverage.

Finite State Methods Bouma [3] explores the formalization of hyphenation using Finite State Transducers (FSTs). Unlike the specialized packed trie implementation found in TEX, this approach utilizes standard finite state calculus, where hyphenation patterns and rules are expressed as regular relations. This method demonstrates that hyphenation can be effectively modeled using deterministic automata, providing a rigorous mathematical framework that allows for the seamless composition of hyphenation with other finite-state morphological processes while maintaining accuracy comparable to traditional pattern-based systems.

Neural Sequence Labeling In the broader context of Natural Language Processing, hyphenation can be modeled as a character-level sequence labeling task (e.g., assigning tags like `B-reak` or `No-break`). State-of-the-art architectures for such tasks typically employ end-to-end deep learning models. Ma and Hovy [10] introduced a widely adopted architecture combining Bidirectional LSTMs (to capture left and right context) with Conditional Random Fields (CRF) to ensure valid label transitions. This represents a methodological shift from finite-state logic to probabilistic graphical models.

Language-Agnostic Neural Syllabification Krantz et al. [5] propose a language-agnostic approach to the problem, framing syllabification as a sequence labeling task utilizing Bidirectional Long Short-Term Memory (Bi-LSTM) networks. By treating hyphenation points as classification targets between characters, they demonstrate that high-accuracy segmentation can be achieved directly from raw text data without reliance on language-specific rules or manual feature engineering. This contrasts with the `patgen` approach by prioritizing generalization capabilities through deep learning over the storage efficiency of packed tries.

Subword Segmentation by Transformers While distinct from typographic hyphenation (which focuses on line breaking), subword segmentation is the dominant paradigm in modern NLP for training Large Language Models. Sennrich et al. [13] introduced Byte-Pair Encoding (BPE) for neural machine translation, which iteratively merges frequent character pairs to form subword units. Franek [4] measures trade-offs between finite-state and LLM-based approaches.

The most widespread method for word-level hyphenation in software systems, such as TEX or in web browsers, is pattern-based hyphenation, due to its high accuracy and generalization, small size, and potential applicability to any language with a hyphenated word list. To conduct a rigorous comparison of segmentation approaches, one requires ground truth data for evaluation.

The repository that provides hyphenation patterns for TEX Live distribution and many other document processing systems is `hyph-utf8`. To date, it supports hyphenation patterns for 90 languages. These patterns are loaded into the LATEX format file, enabling TEX engines quickly load patterns for 85 languages in packed tries to perform correct, language-specific hyphenation. Even though the best results for ever-developing languages and their new words are obtained by patterns generated by `patgen` from the large-scale word list of hyphenated words, a substantial portion of the 90 languages in `hyph-utf8` rely on manually created rules.

To make masterpieces of hyphenation patterns, one needs to develop a big, consistently hyphenated word list together with heuristically optimized parameters for the `patgen` generation. While pattern generation is unreasonably efficient [20,16], minimalisation of pattern size is known to be NP-complete [18, Theorem 2.1].

Recent studies [16] show that high-quality patterns could be generated even for sets of languages [17], and word list development may be transferred from related languages [15]. This triggers the pressing need for an available repository of word lists to ease the development of new word lists and their `patgen` generation parameters.

This paper describes the first version of hyph-bench, a benchmark dataset of hyphenated words for evaluation of hyphenation approaches. We also report on baselines based on the generation of near-optimal hyphenation patterns. We start with the description and design of the datasets in Section 2. We proceed with the pattern generation process principles in Section 3. We design an evaluation of our benchmark dataset and metrics used in Section 4. Our results are finally described in Section 5. We conclude with the summary and future work in Section 6.

> "If you want to use TeX for languages other than English, you must prepare a dictionary of hyphenation exceptions and/or patterns. Such dictionaries should be compiled by experts, and they should be paid for their expertise." – Donald Ervin Knuth

2 Hyphenated Word Lists for Pattern Generation

In this section, we present the initial version 1.0 of hyph-bench dataset collection and describe the preprocessing that has been performed. We then discuss the differences and similarities between the datasets. Finally, we elaborate on inconsistencies that occur in the data.

Table 1: Dataset hyph-bench origin: data origin, licenses, and ambiguities. The two rightmost columns show the difference in the number of ambiguous hyphenations before and after preprocessing.

Language/Name	Origin	Licence	Amb. before	Amb. after
cs/cshyphen_cstenten	[22]	CC BY-NC-SA 3.0	9607	278
cs/cshyphen_ujc	[20]	MIT	884	17
cs/wiktionary	[23,24]	CC BY-SA 4.0	46	30
cssk/cshyphen	[22]	MIT	N/A	N/A
de/wiktionary	[23,24]	CC BY-SA 4.0	741	398
de/wortliste	[6]	MIT	0	0
el/wiktionary	[23,24]	CC BY-SA 4.0	87	54
es/wiktionary	[23,24]	CC BY-SA 4.0	15	14
is/hyphenation-is	[12]	CC BY 4.0	0	0
it/wiktionary	[23,24]	CC BY-SA 4.0	108	33
ms/wiktionary	[23,24]	CC BY-SA 4.0	3	2
nl/wiktionary	[23,24]	CC BY-SA 4.0	361	243
pl/wiktionary	[23,24]	CC BY-SA 4.0	117	111
pt/wiktionary	[23,24]	CC BY-SA 4.0	30	22
ru/wiktionary	[23,24]	CC BY-SA 4.0	204	175
th/orchid	[19]	CC BY-SA 4.0	4976	153
tr/wiktionary	[23,24]	CC BY-SA 4.0	41	25
uk/wiktionary	[15]	CC BY-SA 4.0	14	1

Overview The data were collected from several sources and in different formats. The dataset has been designed for maximal diversity, serving the purpose of optimizing the pattern generation process. Table 1 summarizes all the datasets included in the collection by the time this article was published. The language of each dataset in the benchmark is identified by its two-letter code, as specified in the ISO 639-1 standard. The only exception is *cssk*, denoting the Czechoslovak word list, which is the concatenation of the codes *cs* and *sk*. The majority of the data was collected from the Wiktionary [23] dump previously processed by the Wiktextract tool [24]. Multiple languages are represented, including Slavic ones – Czech, Polish, Ukrainian, and Russian. Another part of the benchmark comprises Czech and Czechoslovak word lists used for creating joint Czechoslovak hyphenation patterns [20,22]. The rest of the collection contains word lists for German [6], Icelandic [12], and Thai [19]. We highlight that languages with non-Latin scripts are also represented, such as Modern Greek, Russian, Ukrainian, and Thai.

Preprocessing Since the Wiktionary data were downloaded as raw JSON files, these had to be parsed first. Each line of the raw data corresponds to one Wiktionary entry and may contain hyphenation information if the entry contains it and the Wiktionary processing tool [24] was able to capture it. After extracting this information, we use regular expressions to get hyphenated words in the desired format.

In case of the *cssk/cshyphen* dataset, each hyphenated word was provided with a weight corresponding to its priority. We used these weights as factors to replicate each entry in the final dataset accordingly.

To enable the use of the `patgen` pattern generation program with the benchmark, we had to perform a few additional tasks. Firstly, characters reserved for internal computations (numbers, '.', and '*') had to be removed. Secondly, `patgen` cannot process words of length greater than 50. This is a problem, particularly in the *th/orchid* dataset, although other datasets also contained too-long entries. We decided to split the lines in the Thai dataset along some hyphenation points (as the lines are actually multi-word sequences), but exclude the troublesome words in other cases, though it is a shame not to have `do-nau-dampf-schiff-fahrts-elek-t-ri-zi-tä-ten-haupt-betriebs-werk-bau-un-ter-be-am-ten-ge-sell-schaft` in the data.

To achieve better benchmark quality, we started to address ambiguity and inconsistencies in the data. There were lines with different hyphenation points listed for the same word, especially in the *wiktionary* datasets. This poses a practical problem, as generating patterns with 100% precision is infeasible in such conditions. In rare cases, the true hyphenation depends on semantics, which is impossible to determine from an isolated word (`rec-ord` versus `re-cord`), so we left them both. In most cases, the reason is human error: if hyphenation points of one case are a proper subset of another one, e.g., `di-verse` is a subset of `di-ver-se`, merge these into the more inclusive one – `di-ver-se`. If hyphenations of the same words are not proper subsets, we keep both for further consistency cleaning and acknowledge that precision will be affected.

We manually checked the disambiguation output of *cs/wiktionary* and consulted the official guide for hyphenation of the Czech language [2]. We found that a higher permissiveness strategy caused changing, e.g., `špa-něl-ský` to `špa-něl-s-ký`. While setting `\righthyphenmin` to 3 might be the preferred setting for some Czech documents with wide columns, added hyphenation points may be filtered out by setting `\righthyphenmin` to 3 at the document level during typesetting, not causing any harm. On the contrary, our previous experiments showed [16] that consistent marking of syllables near the word borders *do improve* pattern quality and generalization, and the patterns are smaller because of the smaller need to cover exceptions near the word borders. Changes in datasets before and after preprocessing are shown in Table 1. We did not disambiguate the weighted Czechoslovak dataset, as it contains duplicate records by design.

Statistics and Issues in the Data Table 2 shows basic statistics for comparisons within the collection. There is significant variability between datasets in terms of their size, as well as average line lengths and density of hyphenation marks. For instance, the reader may notice a similarity between Germanic languages (such as German and Dutch, which have longer words, thus allowing more hyphenation points). Another observation is that some Slavic datasets (*cs/cshyphen_** and *uk/wiktionary*) have nearly identical values for per-line statistics. Thai deviates from others by order of magnitude because lines in *th/orchid* corpus are actually whole paragraphs. Since there are no explicit spaces or punctuation in

Table 2: Dataset collection overview: full size in kBytes for UTF-8 encoded files (Size), average line length in characters ($\overline{\#\text{Lines}}$), average number of hyphenation points per line ($\overline{\#\text{Hyphs}}$), `\lefthyphenmin` and `\righthyphenmin` parameters (LRhyp), and the number of distinct lowercase characters (#Chars) occurring.

Language/Name	Size	#Lines	$\overline{\#\text{Lines}}$	$\overline{\#\text{Hyphs}}$	LRhyp	#Chars
cs/cshyphen_cstenten	8,202.6	597,170	11.83	2.79	1 2	48
cs/cshyphen_ujc	1,377.1	104,277	11.48	2.59	1 1	48
cs/wiktionary	632.0	53,962	9.95	2.16	1 1	48
cssk/cshyphen	10,646.4	835,145	11.10	2.54	1 1	50
de/wiktionary	12,821.6	807,813	15.01	2.96	1 1	86
de/wortliste	10,035.4	604,970	15.78	2.99	2 2	47
el/wiktionary	603.9	28,208	11.90	2.87	1 1	52
es/wiktionary	11,753.4	816,723	13.36	3.27	1 1	50
is/hyphenation-is	3,037.8	218,308	12.12	1.91	1 1	45
it/wiktionary	981.1	77,351	11.96	2.90	1 1	39
ms/wiktionary	13.6	1,247	8.52	1.81	1 1	69
nl/wiktionary	8,124.5	528,598	14.73	2.82	1 1	68
pl/wiktionary	1,713.6	141,916	10.75	2.20	1 1	52
pt/wiktionary	541.6	44,321	11.17	2.74	1 1	40
ru/wiktionary	3,795.6	152,608	13.10	2.74	1 1	51
th/orchid	3,567.1	34,868	40.69	6.02	1 1	95
tr/wiktionary	187.2	15,619	10.41	2.37	1 1	42
uk/wiktionary	351.6	15,701	11.79	2.65	1 1	41

Thai transliteration (claimed by [1]), the hyphenation task shifts to identifying word and sentence boundaries within a paragraph.

It is important to say that the quality of the data varies across datasets. On the one hand, there are maintained and possibly manually disambiguated word lists, such as *de/wortliste* [6]. On the other hand, there are no guarantees of quality in *wiktionary* datasets, as anyone can contribute to a Wiktionary entry, and the entries are usually not subject to professional checks. It is also the reason why there are so many distinct characters present in some datasets, such as in *de/wiktionary*. This may also be the reason why there are so few hyphenation points in Russian. Nevertheless, that is technically not a problem when the data is used as a benchmark and may even contribute to generalization evaluation if used with other datasets.

"Math is sometimes called the science of patterns." – Ronald Graham

3 Pattern Generation Basics

Along with collecting data for the benchmark, we also compiled a baseline for evaluating the quality of patterns generated by the `patgen` program [7,9]. In

this section, we provide a brief overview of the pattern generation process with `patgen` and present the metrics used for evaluation.

Pattern Generation with `patgen` The `patgen` pattern generation tool [8] is part of the TEX Live distribution. It creates patterns from hyphenated word lists that are subsequently used in the TEX82 hyphenation algorithm. For efficient lookup, the algorithm uses a dynamic packed trie data structure.

A single run of the program consists of several phases. First, the *translate file* is read if present, and the characters (byte sequences) it contains are mapped to `patgen`'s internal codes. Then the input patterns are loaded. Subsequently, the algorithm iterates over the word list and decides which of the candidate patterns to include in the output collection. Finally, the selection is evaluated on the input word list to obtain values *good* (hyphenation point correctly identified), *bad* (hyphenation point identified on a non-hyphenated position of a word), and *missed* (hyphenation point not identified, but a word is hyphenated on that position). Speaking in statistical terms, these correspond to true positives, false positives, and false negatives, respectively.

The quality of patterns generated by `patgen` is influenced by the choice of algorithm parameters. We provide an overview in Table 3. It is possible to control the minimal distance of hyphenation points from the edge of the word (`\lefthyphenmin`, `\righthyphenmin`) and lengths of the patterns explored in the current run (*pat_start*, *pat_finish*). The power and the meaning of the generated patterns can be modified by specifying the hyphenation level (*hyph_start*, *hyph_finish*): patterns on higher levels overwrite those on lower levels during hyphenation; odd levels create hyphenating patterns, while even levels create inhibiting patterns (no hyphenation allowed). The parameters *good_wt*, *bad_wt*, and *thresh* play a key role in pattern selection. For a pattern to be accepted, the following inequality must hold [9]:

$$\alpha * good\_wt - \beta * bad\_wt \geq thresh,$$

where α denotes the number of positive examples (word split correctly by hyphenating pattern / previously wrong hyphenation fixed by inhibiting pattern), and β the number of negative examples (word split incorrectly by hyphenating pattern / previously correct hyphenation broken by inhibiting pattern) in the word list.

> "Measurement is the first step that leads to control and eventually to improvement. If you can't measure something, you can't understand it." — H. James Harrington

4 Evaluation and Metrics

In this section, we define the metrics we do use for the evaluation, the term ***parameter profiles***, and propose a method for evaluating them on our benchmark. We also present example profiles that can serve as baselines for advanced parameter search techniques.

Table 3: Overview of `patgen` parameters

Name	Value range	Description
`\lefthyphenmin`	1 – 99	Length of word prefix without hyphenation
`\righthyphenmin`	1 – 99	Length of word suffix without hyphenation
hyph_start	1 – 9	Initial hyphenation level
hyph_finish	1 – 9	Final hyphenation level
pat_start	1 – 15	Minimal pattern length in given level
pat_finish	1 – 15	Maximal pattern length in given level
good_wt	integer	Weight of examples in favor of the pattern
bad_wt	integer	Weight of examples against the pattern
thresh	integer	Evaluation threshold

Metrics For evaluation, we decided to use the following metrics:

Precision P**:** the ratio of correct hyphenations out of all identified hyphenations, defined as $P = \frac{good}{good+bad}$. The values *good*, resp. *bad*, and *missed* are the amounts of correctly resp. incorrectly identified hyphenation points. This is the primary metric, as we do not want errors to appear in the automatically typeset documents.

Recall R**:** the ratio of correctly identified hyphenations out of all hyphenations in the data, defined as $R = \frac{good}{good+missed}$. This is the secondary metric, as allowing more hyphenation points may result in more uniform interword spacing and economic savings due to shorter documents.

F-score F**:** weighted harmonic mean of P and R. [11] We prioritize P much more than R and use $F_{1/7}$-score, defined as $F_{1/7} = \frac{(1+1/7)^2 * P * R}{(1/7)^2 * P + R}$. Based on

Generalization: To assess the performance of the created patterns on unseen data, a 10-fold cross-validation is performed: We generate patterns from a train split of the data, measure their $F_{1/7}$ on the unseen test split, and average the values across the folds.

Size: Tertially, the size (and speed) of the generated patterns matter too. The number of final patterns and their development as the number of patterns generated on a specific hyphenation level are reported. Alternatively, the number of trie nodes (*ops* reported by `patgen` at the end of generation) or an average of trie depth could be used, too.

We are emphasizing precision over recall in $F_{1/7}$, because a bad hyphenation point is much worse than a hyphenation point that was not covered by the patterns, and this metric has already been used in prior work [16].

Parameter Profiles The quality of patterns depends on several adjustable parameters shown in Table 3. To compare different strategies of pattern generation, we define ***parameter profile*** as a set of parameter values that fully determine the `patgen` run on given data. In practice, this means specifying pattern length

Table 4: Parameter profiles values. Parameters *pat_start* and *pat_finish* are combined into column "Lengths" and *good_wt*, *bad_wt*, and *thresh* into column "Params".

Profile	Level	Lengths	Params
base	1	2 4	1 1 1
	2	3 5	1 2 1
	3	4 7	1 3 1
	4	5 9	1 4 1
cshyphen	1	1 3	1 5 1
	2	1 3	1 5 1
	3	2 6	1 3 1
	4	2 7	1 3 1
wortliste	1	1 3	2 3 1
	2	2 4	1 5 1
	3	3 5	1 6 1
	4	4 6	1 7 1

Profile	Level	Lengths	Params
wortliste8	1	1 3	2 3 1
	2	2 4	1 5 1
	3	3 5	1 6 1
	4	4 6	1 7 1
	5	5 12	1 8 1
	6	6 12	1 9 1
	7	7 12	1 9 1
	8	8 12	1 9 1

window (*pat_start*, *pat_finish*) and weights (*good_wt*, *bad_wt*, *thresh*) for every hyphenation level.

Having a parameter profile, we can feed it to the `patgen` program and evaluate it on our benchmark. To assess the generalization properties of the created patterns, a 10-fold cross-validation was also performed. We generate patterns from a train split of the data, measure their performance on the unseen test split, and average the values across the folds.

Baseline Profiles We evaluated three parameter profiles: our own *base*, one adapted from earlier works [22, Table 2, *correct optimized* profile] *cshyphen*, and two from [6] *wortliste8*, and *wortliste* that contains the first 4 levels of the earlier. We summarize their values in Table 4.

"Above all else, show the data." – Edward R. Tufte

5 Results

In this section, we summarize and discuss the experiments conducted with the data and parameter profiles. We first present a comprehensive evaluation of the parameter profiles we used, evaluating both generalization and space efficiency perspectives. We then examine a selected run of `patgen` with the *base* parameter profile over the Czech Wiktionary dataset.

Parameter Profiles Cross-Validation In Table 5 we list the results of our experiments run over all dataset-profile combinations. High $F_{1/7}$-score values in most cases indicate fairly robust behavior of the program on unseen data, although the *base* and *wortliste* profiles seem to create notably less generalizing

Table 5: Cross-validation results overview: $F_{1/7}$-score (rounded to 4 decimal points) and the number of nodes in the pattern-storing trie (rounded to the lowest higher integer) for *base*, *cshyphen*, *wortliste*, and *wortliste8* parameter profiles, averaged over the 10 cross-validated folds

Language/Name	*base*		*cshyphen*		*wortliste*		*wortliste8*	
	$F_{1/7}$	trie size	$F_{1/7}$	trie size	$F_{1/7}$	trie size	$F_{1/7}$	trie size
cs/cshyphen_cstenten	0.9266	14630	0.9444	12052	0.9243	12121	0.9243	13008
cs/cshyphen_ujc	0.9803	19339	0.9830	17046	0.9803	14736	0.9794	20761
cs/wiktionary	0.9712	11275	0.9759	10643	0.9744	8330	0.9739	9960
cssk/cshyphen	0.9981	27653	0.9986	25302	0.9981	21685	0.9982	23690
de/wiktionary	0.9982	37751	0.9983	32394	0.9977	29449	0.9983	37648
de/wortliste	0.9000	54035	0.9775	47544	0.8985	43214	0.9007	55424
el/wiktionary	0.9846	4232	0.9875	3602	0.9866	3223	0.9864	3722
es/wiktionary	0.9997	3299	0.9998	2023	0.9998	1784	0.9998	2129
is/hyphenation-is	0.9804	30244	0.9839	32284	0.9802	28664	0.9797	31229
it/wiktionary	0.9963	4026	0.9970	3593	0.9968	2875	0.9964	4114
ms/wiktionary	0.9039	991	0.9296	796	0.9244	721	0.9236	733
nl/wiktionary	0.9967	33889	0.9972	30859	0.9963	27514	0.9967	33954
pl/wiktionary	0.9904	10058	0.9913	8564	0.9904	7262	0.9901	9845
pt/wiktionary	0.9955	2449	0.9969	1761	0.9970	1422	0.9969	1768
ru/wiktionary	0.9192	68380	0.9270	53820	0.9212	42164	0.9173	92188
th/orchid	0.9569	32666	0.9622	28795	0.9488	24258	0.9551	38587
tr/wiktionary	0.9870	2217	0.9920	1406	0.9916	1273	0.9911	1407
uk/wiktionary	0.9517	5458	0.9602	4741	0.9564	4272	0.9552	4880

patterns of *de/wortliste* dataset than *cshyphen*. We noticed that for datasets with values of `\lefthyphenmin` and `\righthyphenmin` different from 1 (namely *cs/cshyphen_cstenten* and *de/wortliste*), the generalization is much worse

With the pattern trie size, we measure the conciseness of generated patterns. It takes into account both the number of patterns and their length. In this field, *cshyphen* is superior to the other two profiles and beats them in many cases by order of magnitude. There is an exception to this claim in the Thai dataset, where *base* profile is far better. This may be due to the different nature of the dataset, as it does not contain isolated words but sentences.

It is also worth noting that *wortliste8*, although generating larger tries in 8 levels, stays on par in terms of generalization with *wortliste*, using only the first 4 levels of *wortliste8*. This shows that four levels of exceptions may be enough for the hyphenation of natural languages and that larger patterns only catch errors in the data and/or infrequent exceptions from base patterns found on lower levels: their influence on the *F*-score is negligible, given the consistent data.

Parameter Profile Example: Czech Wiktionary To delve deeper into understanding the pattern learning process, we ran `patgen` on the *cs/wiktionary* dataset with the *base* profile and monitored the evolution of the chosen metrics over the run. Table 6 summarizes performance statistics after running each

Table 6: Example run of *base* profile on *cs/wiktionary* dataset. Parameters *pat_start* and *pat_finish* are combined into column "Lengths" and *good_wt*, *bad_wt*, and *thresh* into column "Params".

Level	Patterns	#Good	#Bad	#Missed	Lengths	Params
1	890	114,443	47,169	1,900	2 4	1 1 1
2	2,637	113,597	1,511	2,746	3 5	1 2 1
3	1,490	115,888	1,365	455	4 7	1 3 1
4	914	115,939	260	404	5 9	1 4 1

hyphenation level and lists recorded values of *good*, *bad*, *missed* hyphenations, and the number of patterns generated at each level. The reader may observe that even the first-level patterns cover the majority of hyphenation points in the data, although they identify a large number of incorrect ones. The problem seems to be assessed immediately in the second level, where the number of *bad* hyphenations drops significantly, but the generated inhibiting patterns erase some of the correct hyphenations as well (*good* drops and *missed* increases). In further levels, both *bad* and *missed* decrease simultaneously. The amount of new patterns increases at first, but starts dropping steadily in later levels.

"If I waited for perfection...I would never write a word." – Margaret Atwood

6 Conclusion and Future Work

We have prepared a diverse dataset hyph-bench of word lists for generating hyphenation patterns in thirteen languages. We have defined an evaluation framework to compare different algorithms and heuristics for generating hyphenation patterns. We have computed and reported baseline results for several settings. The reported dataset and experiment scripts are available in the repository `https://github.com/tondach01/hyph-bench`.

We hope that hyph-bench will stimulate the research and development of new word lists, pattern generation strategies, and optimized hyphenation patterns, especially for the languages that still do not have word lists, and for the generation of universal patterns for sets of languages that share the phonetic rules of syllabification and hyphenation, as, e.g., the set of Slavic languages. [17].

In future work, to achieve the above vision, the steps are:

word list development Word lists could be extended from available sources with the techniques of bootstrapping [22,21] and transfer learning from [15].

dynamic adaptive profile development Pattern generation could be improved with the implemented heuristics as Bayes optimization suggested in [15].

pattern generation for big alphabets Universal hyphenation and segmentation preparation [14] will need a new version of the `patgen` program [17] capable of working with alphabets of Unicode size.

The benchmark is about to be deposited on the LINDAT repository `https://lindat.cz`.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Antoš, D.: PATLIB, Pattern Manipulation Library. Master's thesis (2002), master's Thesis, Masaryk University, Brno, Faculty of Informatics (advisor: Petr Sojka), https://is.muni.cz/auth/th/c4rby/?lang=en
2. Ústav pro jazyk český AV ČR: Internetová jazyková příručka (2008–2025), https://prirucka.ujc.cas.cz/, the work uses data provided by the research infrastructure LINDAT/CLARIAH-CZ (https://lindat.cz) supported by the Ministry of Education, Youth, and Sport of the Czech Republic (project no. LM2023062).
3. Bouma, G.: Finite State Methods for Hyphenation. Natural Language Engineering **9**(1), 5–20 (2003). https://doi.org/10.1017/S1351324903003073
4. Franěk, J.: Transformer-Based Word Hyphenation. Master's thesis, Masaryk University, Brno, Faculty of Informatics (advisor: Petr Sojka) (2025), https://is.muni.cz/th/wbgkw/
5. Krantz, J., Dulin, M., Palma, P.D.: Language-Agnostic Syllabification with Neural Sequence Labeling. In: Wani, M.A., Khoshgoftaar, T.M., Wang, D., Wang, H., Seliya, N. (eds.) 18th IEEE International Conference On Machine Learning And Applications, ICMLA 2019, Boca Raton, FL, USA, December 16–19, 2019. pp. 804–810. IEEE (2019). https://doi.org/10.1109/ICMLA.2019.00141
6. Lemberg, W.: A database of German words with hyphenation information (2025), https://repo.or.cz/wortliste.git, wordlist version with last change of Sep 20th, 2025
7. Liang, F.M.: Word Hy-phen-a-tion by Com-put-er. Ph.D. thesis, Department of Computer Science, Stanford University (Aug 1983)
8. Liang, F.M.: Word Hy-phen-a-tion by Com-put-er. Ph.D. thesis, Dept. of Computer Science, Stanford University (Aug 1983), https://tug.org/docs/liang/liang-thesis.pdf
9. Liang, F.M., Breitenlohner, P.: PATtern GENeration program for the TEX82 hyphenator (2020), https://tug.ctan.org/info/knuth-pdf/other/patgen.pdf
10. Ma, X., Hovy, E.: End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In: Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers). pp. 1064–1074. ACL (2016). https://doi.org/10.18653/v1/P16-1101
11. Rijsbergen, C.J.V.: Information Retrieval. Butterworths (1979)
12. Rúnarsson, K., Jónsson, B., Gíslason, M.: Icelandic hyphenation dictionary (2.0) (2020), http://hdl.handle.net/20.500.12537/86, CLARIN-IS
13. Sennrich, R., Haddow, B., Birch, A.: Neural Machine Translation of Rare Words with Subword Units. In: Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers). pp. 1715–1725. ACL (2016). https://doi.org/10.18653/v1/P16-1162
14. Shao, Y., Hardmeier, C., Nivre, J.: Universal Word Segmentation: Implementation and Interpretation. Transactions of the Association for Computational Linguistics **6**, 421–435 (2018). https://doi.org/10.1162/tacl_a_00033
15. Sojka, O.: Transfer Learning of Slavic Syllabification for Hyphenation Patterns (Feb 2025), Bachelor Thesis, Masaryk University, Brno, Faculty of Informatics (advisor: Pavel Šmerk, https://is.muni.cz/th/s17d6/?lang=en

16. Sojka, O., Sojka, P.: Towards Perfection of Machine Learning of Competing Patterns: The Use Case of Czechoslovak Patterns Development. In: Horák, A., Rychlý, P., Rambousek, A. (eds.) Proceedings of the 17th Workshop on Recent Advances in Slavonic Natural Language Processing, RASLAN 2023, Kouty nad Desnou, Czech Republic, December 8–10, 2023. pp. 113–120. Tribun EU, Brno (2023), https://www.fi.muni.cz/usr/sojka/papers/sojka-sojka-raslan-2023.pdf
17. Sojka, O., Sojka, P., Máca, J.: A roadmap for universal syllabic segmentation. TUGboat **44**(2) (2023), https://doi.org/10.47397/tb/44-2/tb137sojka-syllabic
18. Sojka, P.: Competing Patterns in Language Engineering and Computer Typesetting. Dissertation theses, Masaryk University, Faculty of Informatics, Brno (Jan 2005), https://www.fi.muni.cz/usr/sojka/papers/sojka-thesis.pdf, https://is.muni.cz/publication/837861/en
19. Sojka, P., Antoš, D.: Context Sensitive Pattern Based Segmentation: A Thai Challenge. In: Hall, P., Rao, D.D. (eds.) Proceedings of EACL 2003 Workshop on Computational Linguistics for South Asian Languages – Expanding Synergies with Europe. pp. 65–72. Budapest (Apr 2003), https://www.fi.muni.cz/usr/sojka/papers/thaieacl.pdf
20. Sojka, P., Sojka, O.: The Unreasonable Effectiveness of Pattern Generation. TUGboat **40**(2), 187–193 (2019), https://tug.org/TUGboat/tb40-2/tb125sojka-patgen.pdf
21. Sojka, P., Sojka, O.: Towards Universal Hyphenation Patterns. In: Horák, A., Rychlý, P., Rambousek, A. (eds.) Proceedings of Recent Advances in Slavonic Natural Language Processing—RASLAN 2019. pp. 63–68. Tribun EU, Karlova Studánka, Czech Republic (2019), https://nlp.fi.muni.cz/raslan/2019/paper13-sojka.pdf, https://is.muni.cz/publication/1585259/?lang=en
22. Sojka, P., Sojka, O.: New Czechoslovak Hyphenation Patterns, Word Lists, and Workflow. TUGboat **42**(2) (2021), https://doi.org/10.47397/tb/42-2/tb131sojka-czech
23. Wiktionary contributors: Wiktionary, https://www.wiktionary.org/
24. Ylonen, T.: Wiktextract: Wiktionary as Machine-Readable Structured Data. In: Calzolari, N., Béchet, F., Blache, P., Choukri, K., Cieri, C., Declerck, T., Goggi, S., Isahara, H., Maegaard, B., Mariani, J., Mazo, H., Odijk, J., Piperidis, S. (eds.) Proceedings of the Thirteenth Language Resources and Evaluation Conference. pp. 1317–1325. European Language Resources Association, Marseille, France (Jun 2022), https://aclanthology.org/2022.lrec-1.140/