Heuristic Hyper-minimization of Finite State Lexicons

Senka Drobac*, Krister Lindén*, Tommi A Pirinen†, Miikka Silfverberg*

*University of Helsinki
Department of Modern Languages, PO Box 24
†University of Helsinki
Department of Speech Sciences, PO Box 9
senka.drobac, krister.linden, tommi.pirinen, miikka.silfverberg@helsinki.fi

Abstract

Flag diacritics, which are special multi-character symbols executed at runtime, enable optimising finite-state networks by combining identical sub-graphs of its transition graph. Traditionally, the feature has required linguists to devise the optimisations to the graph by hand alongside the morphological description. In this paper, we present a novel method for discovering flag positions in morphological lexicons automatically, based on the morpheme structure implicit in the language description. With this approach, we have gained significant decrease in the size of finite-state networks while maintaining reasonable application speed. The algorithm can be applied to any language description, where the biggest achievements are expected in large and complex morphologies. The most noticeable reduction in size we got with a morphological transducer for Greenlandic, whose original size is on average about 15 times larger than other morphologies. With the presented hyper-minimization method, the transducer is reduced to 10,1% of the original size, with lookup speed decreased only by 9,5%.

Keywords: hyper-minimization, lexicon, FST

1. Introduction

Finite-state transducers are an established way of encoding morphological analysers for natural languages. Nevertheless, full-scale morphological analysers can often grow to be too large for use cases like spell checkers, speech processing and shallow parsing, which should have a moderate memory footprint. Large transducers can be optimised by preserving the sub-lexicon structure with special symbols called flag diacritics, which prevents combinatorial blow ups in determinization.

Until now, applying flag diacritics has required a linguist to provide the lexicon compiler with their positions. However, there are two major problems with this kind of approach: Firstly, linguists often do not have a very good understanding of the structure of the finite-state networks built from lexicographical-morphological descriptions; Secondly, the addition of flag diacritics to these descriptions makes them unreadable and unmanageable since the amount of non-linguistic data in the linguistic description increases.

One of the reasons why flag diacritics have been so cumbersome from the linguist's point of view, is their two-fold nature. On the one hand, they are there to optimise the finite-state automaton structure, e.g. in (Karttunen, 2006). On the other hand, they are the primary method of describing non-contiguous morphological constraints (Beesley, 1998). If they are spuriously applied to restrict separated morphotactic dependencies, the effect on optimisation is at best haphazard, and the resulting description may be neither linguistically motivated nor maintainable from a computational view-point.

This article seeks to address problems associated with flag diacritics by using an algorithm for inducing flag positions from the linguistic morpheme structure, implicitly present in lexical descriptions.

2. Background

Finite state morphology (Beesley and Karttunen, 2003) is the state-of-the-art in writing morphological analysers for natural languages of a whole range of typologically varying morphological features. The finite-state approach is built around two practical concepts: constructing lexicographical descriptions of the language using a tool called lexc and expressing morphophonological variations as regular expression rules. In this paper, we study the use of the lexicographical structure as framed by lexc.

Lexc supports a simple right-linear morphosyntactic grammar formalism. In linguistic terms, this means approximately the following: we have collections of lexicons, which are lists of morphemes. Each morpheme in a lexicon has continuation lexicons, which in turn determine the set of morphemes that can succeed the morpheme.

Consider for example Finnish morphology. Nominal inflection can be constructed neatly from left to right. In Figure 3, there is a lexc representation of the Finnish words *talo* 'house', *asu* 'clothing' and *kärry* 'cart', and nominal suffixes *n* (singular genitive), *lle* (singular allative) and *ksi* (singular translative). Derivation of word-forms starts from the `Root` lexicon. Each of the nouns in the root set of morphemes continues rightwards to the `NOUNCASES` set of morphemes, and each case morpheme continues towards the special `#` lexicon signifying the end of a word-form.

Finnish was used as an example in Karttunen's paper on flag diacritics on optimisation (2006). In that

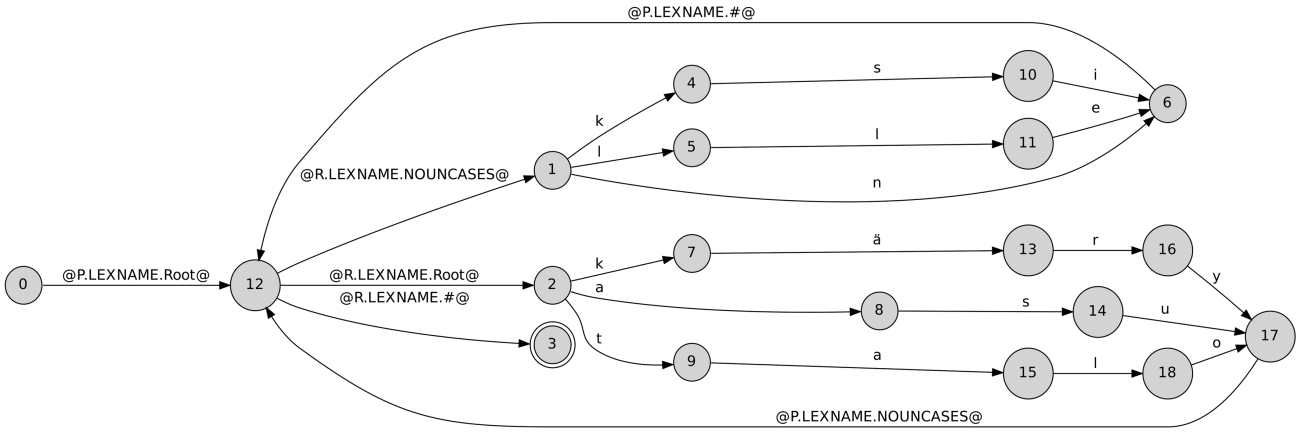

Figure 1: Simplified part of Finnish lexc grammar description with automatic flags

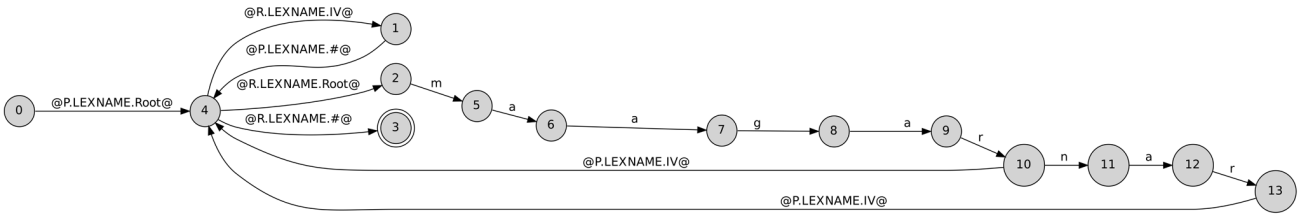

Figure 2: Simplified part of Greenlandic lexc grammar description with automatic flags

```
LEXICON Root
talo NOUNCASES ;
asu NOUNCASES ;
kärry NOUNCASES ;

LEXICON NOUNCASES
n # ;
lle # ;
ksi # ;
```

Figure 3: Simplified part of Finnish lexc grammar description

paper, he showed that the optimisation quality of wisely selected flag diacritics can be substantial; from a 20,498 state automaton to a 1,946 state one. The article describes Finnish numerals, which have the feature of requiring agreeing inflection in free compounding. This can be achieved by allowing all compounds and restricting the combinations by flags, instead of by lexicon structure. Unfortunately, the article does not show examples or re-producible description of the lexicographical data, but according to our experience there are no available morphologies that show similar compression quality, so it can be considered towards the upper bounds of what such compression can achieve.

3. Flag diacritics

Flag diacritics are special multi-character symbols which are interpreted during runtime. They can be used to optimize large transducers to couple entrance points of the sub-graphs with the correct exit points. Their special syntax is: `@operator.feature.value@`, where `operator` is one of the available operators (P, U, R, D, N, C), `feature` is the name of a feature set by the user and `value` can be any value held in a feature, also provisionally defined. For additional information on the semantics of flag diacritic operators, see Beesley and Karttunen (2003).

In this paper, we will use only two types of flag diacritics: positive setting (`@P.feature.value@`) and require test (`@R.feature.value@`). While positive setting flag only sets the feature to its value, the require test flag invokes testing whether the feature is set to the designated value. For example, `@P.LEXNAME.Root@` will set feature `LEXNAME` to value `Root`. If later in the path there is an R flag that requires test `@R.LEXNAME.Root@`, the invoked test will succeed and that path will be considered valid.

4. Methods

Our algorithm is based on the idea that adjacent morph combinatorics can be expressed with finite-state flags like this:

Every rightward continuation is replaced with a positive setting flag with feature called `LEXNAME` and a value corresponding to the continuation lexicon. For example, the lexicon in Figure 3 has one continuation lexicon: `NOUNCASES`, which is represented using the positive setting `@P.LEXNAME.NOUNCASES@@`. Correspondingly, a require test `@R.LEXNAME.NOUNCASES@` is inserted in the beginning of the `NOUNCASES` continuation lexicon.

Additionally, every morphological description starts

with `Root`, which is represented using the pair of flags `@P.LEXNAME.Root@@R.LEXNAME.Root@` and ends with `#`, which is represented using the pair of flags `@P.LEXNAME.#@@R.LEXNAME.#@`.

The transducer built from the morphological description in Figure 3 is shown in Figure 1.

4.1. Composition

Lexicons that contain flag diacritics can be composed with other transducers which also contain flag diacritics without worrying about flag collisions. This is achieved by renaming flag diacritics in both argument transducers in such a way that collisions become impossible and then inserting flag diacritics freely from each argument to the other.

Consider for example the composition of a lexicon L with a rule R. If both transducers contain flag diacritics for feature `FEATURE`, then *all features* F are renamed `F1` in L and `F2` in R. [1] All flag diacritics (like `@P.F.True@`) are renamed correspondingly (to `@P.F1.True@` in L and `@P.F2.True@` in R) and a new lexicon L' and rule R' are created by inserting freely all flag diacritics from R to L and from L to R, respectively.

The transducers L' and R' can be composed and it is easy to prove that the result satisfies the property that, if flag diacritics are compiled out, then the resulting transducer without flag diacritics will accept exactly the same strings as the composition of the transducers that are obtained by compiling out flag diacritics from the original lexicon L and the original rule R.

4.2. Pruning invalid paths

When a lexicon is compiled into a transducer, a single state becomes the point where all continuation lexicons begin with a require test and end with positive settings. We call this state a flag diacritic hub. The initial hub is shown in Figure 2 in state 4. Various `P` flags are transitions leading to the state and matching `R` flags are transitions leading out of the state. When lexical transducers with those flag-diacritic hubs are composed with grammar rule transducers, unnecessary paths may occur. They are the result of combining non-matching `P` and `R` flags. Since an `R` invokes a require test, a path will not be valid unless a matching `P` flag feature was previously set. Therefore, such paths do not change the language, but increase the transducer size.

In order to remove all those paths, path pruning is needed. Checking that in every hub state for every outgoing `R` flag there is a matching incoming `P` flag. In case the `P` flag is not preceding the `R` flag, the whole path is proclaimed invalid and removed from the transducer.

```
Step 1:

P.LEXNAME.sublex_i -> JOINER.sublex_i
R.LEXNAME.sublex_i -> JOINER.sublex_i

Step 2:
    transducer - filter
        .o.
    P.LEXNAME.Root Σ*
        .o.
   Σ* R.LEXNAME.#

Where transducer is the lexicon composed with
rules (with ? as any symbol and Σ* as
universal language) and filter is:

Σ* [(?-JOINER.sublex_i) | ?:?]
    JOINER.sublex_i
[(?-JOINER.sublex_i) | ?:?] Σ*

Step 3:
    JOINER.sublex_i -> epsilon
```

Figure 4: Removal of flag diacritics

Operation	Name
a b	concatenation
a | b	disjunction
a : b	cross product
a .o. b	composition
*****	Kleene star

Table 1: List of operators

4.3. Removing flag diacritics

Since real-world morphological descriptions may contain empty, or nearly empty, continuation lexicons, inserting flags in those cases only increase the size of the transducer, without gaining any benefits. Therefore, those continuation lexicons need to be recognized and corresponding flag diacritics removed. Additionally, rule composition with a flagged lexicon usually results in dramatical size increase. In order to reduce the final transducer size, it is important to recognize which flag diacritics should be removed from the lexicon before composing it with rules.

Removal of flag diacritics is usually done with the command `remove-flag-FEATURE`, which removes all flags with the given `FEATURE`. However, our flags all have the same feature, called `LEXNAME`, and values corresponding to sub-lexicon names. Therefore, if we want to remove just flags for a certain sub-lexicon `SUBLEX`, we use the algorithm shown in Figure 4, with operators explained in Table 1:

[1] It is not sufficient to rename only flag diacritics with common features, because that might clash with existing feature names.

```
LEXICON Root
r1 A;
r2 A;
r3 B;

LEXICON A
a1  A;
a2  B;

LEXICON B
b2 #;
```

Figure 5: Example of flag removal measurement T

First, `P.LEXNAME.SUBLEX` and `R.LEXNAME.SUBLEX` transitions are substituted with an arbitrary special symbol, ie. `$JOINER.SUBLEX$` in the entire transducer. Then, from the original transducer we subtract all the paths that do not have two identical joiners next to each other. We also filter all paths that do not start with `P.LEXNAME.Root` and end with `R.LEXNAME.#` and finally substitute all `$JOINER.SUBLEX$` transitions with epsilon transition.

4.4. Choosing flag-diacritics for removal

We have experimented with removing different flags and flags combinations from the lexical transducer to get an optimal transducer size after the composition with rules, although the flag removal usually increases lexical transducer size. For the Greenlandic lexicon, we have counted for each sub-lexicon how many morphs there are (w), how many unique continuations come from it (c) and how many times it was mentioned as a continuation in other sub-lexicons (m). In Figure 5 is an example lexicon with three sub-lexicons: `Root`, `A` and `B`. Sub-lexicon `Root` has three morphs `r1`, `r2` and `r3`. From `Root` there are two unique continuations `A` and `B` and the sub-lexicon itself wasn't mentioned anywhere as a continuation in the entire lexicon.
Similarly, sub-lexicon `A` has 2 morphs, two continuations and was mentioned totally 3 times in the lexicon - two times as continuations from the `Root` sub-lexicon and once from itself. Sub-lexicon `B` has only one word, one continuation and was mentioned 2 times.
After that, we calculated T which is product of those 3 counts:

$$T = w * c * m \quad (1)$$

The complete calculation for this small example lexicon is shown in Table 2.
After calculating T for all the sub-lexicons in the Greenlandic lexicon and sorting values, we got data that seems to fit an exponential function. We experimented removing one flag diacritic at a time and the sizes we got for the lexicon transducer itself and the one with grammar rules composed is shown in Figure 6. On the x-axis are sub-lexicons, while the y-axis shows sizes in megabytes and the right y-axis the T-measure.

Lexicon	w	c	m	T
Root	3	2	0	6
A	2	2	3	12
B	1	1	2	2

Table 2: Counting number of morphs (w), continuations (c) and mentions (m) in each sub-lexicon

The dotted line shows sizes of the lexicon transducer with flag diacritics removed for one sub-lexicon at a time. Each new removal is done to the previous result transducer. The solid dot line shows sizes of the same lexicon transducers with the rules composed to them. The dashed line shows the T measure (with values on the right y-axis).
It is interesting to see that the relations of the T measure and transducer sizes after flag removal are somewhat proportional. Additionally, the point where the transducer size with the applied rules is the smallest is just after the place where the T measure starts to grow rapidly.
In Table 3, we show sizes of language transducers composed with grammar rules. First, there are sizes of original transducers, then transducers compiled with our new method that inserts all flag diacritics and finally flagged transducers after removing some of the flags, as described in Section 4.3.

5. Data

We measure the success of our algorithm using real-world, large-scale language descriptions. For this purpose we have acquired freely available, open source language descriptions from the language repository of the University of Tromsa (Moshagen et al., 2013).[2] The languages selected are Greenlandic (kal), North Saami (sme), Erzya (myv), Finnish (fin) and Lule Sami (smj). All operations with transducers were performed using Helsinki Finite State Technology tools (Lindén et al., 2011).

6. Discussion

The results of this study show that large-scale language descriptions can be compiled into smaller transducers using automatically inserted flags. The effect is especially pronounced for language descriptions which repeat morphemes in many different places, like the morphological analyzer for Greenlandic. Since flag diacritics themselves take space in the transducer graph, this method did not offer improvements for descriptions where the original transducer was small.
While requiring `R` flag diacritics will always occur only once for every continuation lexicon, the data shows that, for certain right continuations, P flag diacritics occur hundreds of times. This happens every time when in the same sublexicon there are words which have the same beginning. For example, Figure 2 shows how two morphemes `maagar` and `maagarnar` have the

[2] https://victorio.uit.no/langtech, revision 73836

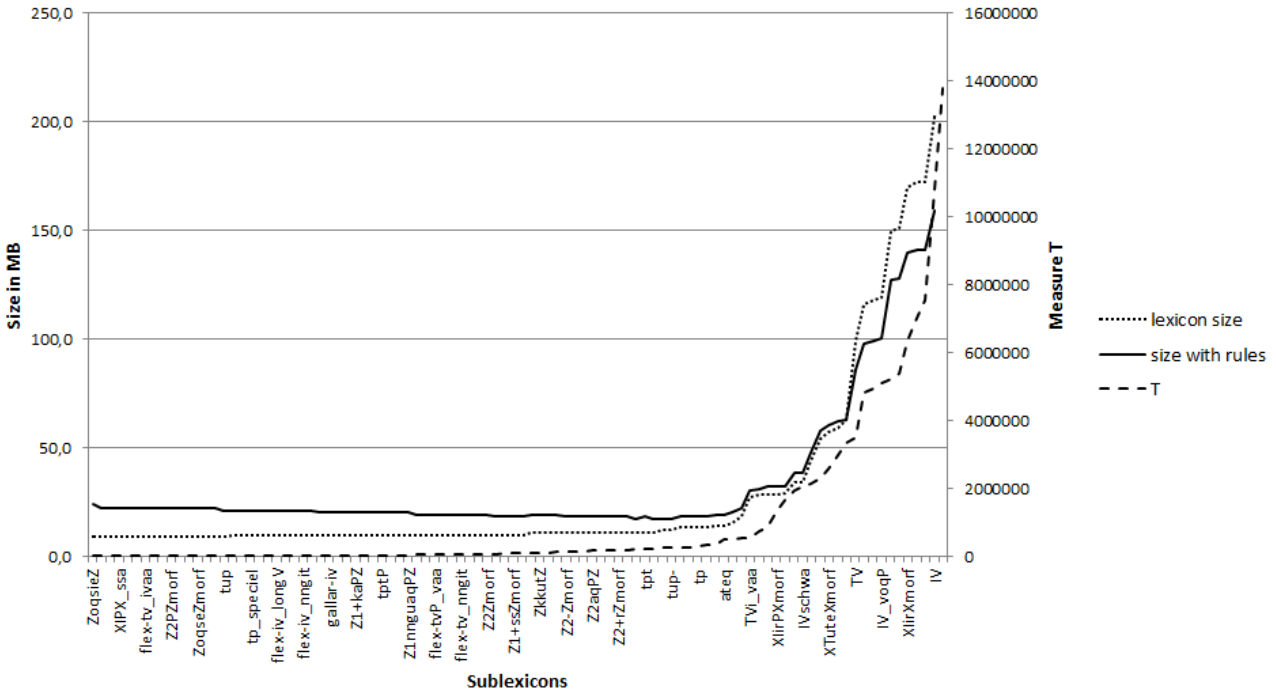

Figure 6: Connection between T measure and transducer sizes with flag-diacritics being removed one by one

Language	Original	With all flags	After optimization	%
Greenlandic	168 M	185 M	17 M	10,1%
North Saami	12 M	14 M	5,7 M	47,5%
Finnish	17 M	17 M	16 M	94,1%
Lule Saami	5 M	19 M	3 M	60,0%
Erzya	3,7 M	16 M	5,3 M	143,2%

Table 3: Sizes of transducers without and with automatic flags (in megabytes); Percentage shows size of the flagged transducer after optimization, in comparison with the original

Language	Original	With flags	%
Greenlandic	2 770 w/s	2 507 w/s	90,5%
North Saami	30 714 w/s	8 775 w/s	28,6%
Finnish	84 415 w/s	27 420 w/s	32,5%

Table 4: Look-up speed of transducers without and with automatic flags; Percentage shows speed of the flagged transducer in comparison with the original

same continuation flag `@P.LEXNAME.IV@`, but they cannot collapse into one path. In future work, it should be checked if removing flags for those paths would further reduce the size of transition graphs.

7. Conclusion

In this article we showed that by using morphologically motivated flags we can dramatically improve the size of large lexical transducers. Automatically inserted flag diacritics can make manual size optimization preformed by linguists unnecessary, which may result in more readable and easier to maintain linguistic descriptions.

8. Acknowledgements

The research leading to these results has received funding from FIN-CLARIN, Langnet and the European Commission's 7th Framework Program under grant agreement n° 238405 (CLARA).

9. References

Beesley, Kenneth R and Karttunen, Lauri. (2003). *Finite state morphology*, volume 18. CSLI publications Stanford.

Beesley, Kenneth R. (1998). Constraining separated morphotactic dependencies in finite-state grammars. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 118–127. Association for Computational Linguistics.

Karttunen, Lauri. (2006). Numbers and finnish numerals. *SKY Journal of Linguistics*, 19:407–421.

Lindén, Krister, Axelson, Erik, Hardwick, Sam, Pirinen, Tommi A, and Silfverberg, Miikka. (2011). Hfst—framework for compiling and applying morphologies. In Mahlow, Cerstin and Piotrowski, Michael, editors, *Systems and Frameworks for Com-*

putational Morphology, volume 100 of *Communications in Computer and Information Science*, pages 67–85. Springer Berlin Heidelberg.

Moshagen, Sjur, Pirinen, Tommi A, and Trosterud, Trond. (2013). Building an open-source development infrastructure for language technology projects. In *Proceedings of Nodalida 2013*.