# A string-to-graph constructive alignment algorithm for discrete and probabilistic language modeling

**Andrei Shcherbakov** **Ekaterina Vylomova**
The University of Melbourne
Parkville, Victoria 3010, Australia
andreas@softwareengineer.pro evylomova@gmail.com

## Abstract

We propose a novel algorithm of graph-to-string alignment that constructs automata from a sample of string sequences. While being a variant of global edit distance, the algorithm also employs graph construction operations in order to form an optimal alignment. It allows dynamic insertion of edges and nodes driven by optimization of the best string-to-graph alignment score. The algorithm may be used both to derive discrete non-deterministic acceptor (or a regular expression) and to build a probabilistic generative model of an input language. An outstanding ability to produce accurate approximate models from extremely sparse training sets constitutes the main advantage of the technique.

## 1 Introduction

Tasks such as text generation or language modelling might require approximate string matching. Often the models are trained on a set of matching samples (sometimes augmented with mismatching, or negative, samples). In most cases, they rely on extrinsic metrics (n-gram similarity (Kondrak, 2005), perplexity (Jelinek et al., 1977), Kolmogorov (Li and Vitányi, 2013) or cognitive (Rogers et al., 2013) complexities, Levenshtein distance (Levenshtein, 1966)). An objective function may not directly correspond to any formally proven or intuitively observable reasonable goal. In current paper, we attempt to combine structure awareness of graph-based models with simplicity of similarity-based approaches. We train a graph-based model that minimizes modified recurrent edit distance[1] between each training sample and the path in the graph closest to it. As a graph-based model we propose to use a weighted non-deterministic finite state acceptor (WFSA) to enable a compact representation of learned sequences as well as to allow alternations and repetitions of substrings. The approach we consider in the paper is essentially based on the hypothesis that sequences that are likely to appear in a language normally form dense clusters around some basic patterns. In other words, the sequences are characterized by relatively low edit distance to *some* cluster centroid. The proposed technique handles both branching and looping of subsequences in a consistent way. As a result, pattern generalization may be carried out without any extra pre- or post-processing.[2]

## 2 Architecture

We first provide a set of positive (and, optionally, negative) string samples in order to build a weighted acceptor (WFSA). The training procedure runs as follows. Initially, the acceptor only contains unconnected start and finish (accepting) nodes. At each training iteration we compute the best possible alignments between the acceptor and a given sample string using Smith–Waterman (Smith et al., 1981) alignment search procedure with the following major differences. First, it allows virtually any automaton edge (even non-existent ones) to be considered for alignment paths.[3] In such a way, it enables new candidate edges that should be built in order to improve fit to a given sample. Second, it speculates on prospective new states of acceptor, one per sample string character. Each string sample $y$ is processed in the following two phases during training.

**Phase 1 - counting alignment scores and building a temporary transducer.** We construct a ma-

[1] Here "recurrent" means that a path may contain loops.

[2] The code is available at http://regexus.com/jumpalign , https://github.com/andreas-softwareengineer-pro/jumpalign

[3] Contrary to it, Smith–Waterman algorithm effectively considers both string operands as unmodifiable linear-shaped automata

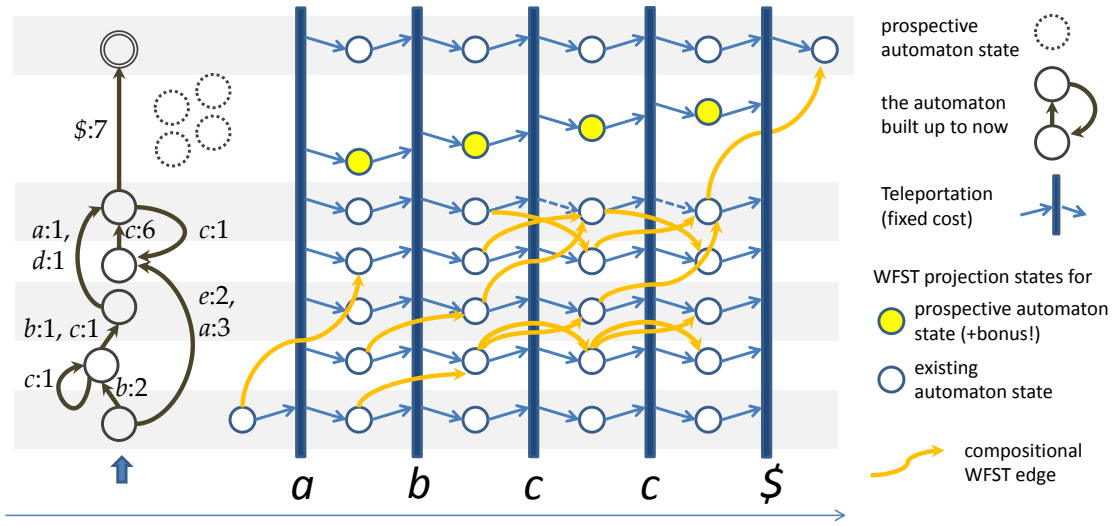



Figure 1: A single acceptor-to-string alignment step.

trix of best partial alignment scores $S_{i,j}$, where $n$ is number of existing WFSA states, $i \in \{0..n + |y|\}$ is existing or prospective state of acceptor, and $j \in \{1..|y|\}$ is sample string position. For each $j$ we consider all $n$ existing automaton states augmented with one new (prospective) state "number" $n + j$. In order to assign candidate scores for $S_{i,j}$ cell, we transfer scores from $S_{*,j-1}$ cells in two ways: (1) using existing WFSA edges that match the $y_j$ character,[4] applying respective edge weights as increments; or (2) using virtual "teleport" edges that might be added between any pair of states, with a fixed teleport penalty $T$:[5]

$$S_{i,j} = \max \begin{cases} \max_k S_{k,j-1} - T \\ \max_{k:\exists k \xrightarrow{y_j} i} (S_{k,j-1} + w(k \xrightarrow{y_j} i)) \end{cases} \quad (1)$$

where $y_j$ is the $j^{th}$ character of the sample string $y$; $k \xrightarrow{y_j} i$ is WFSA edge from state $k$ to state $i$ labelled with $y_j$ character; $w(k \xrightarrow{y_j} i)$ is its weight.

Indeed, we build a transient weighted finite-state transducer (WFST) which translates the acceptor into the sample string. Each transducer state $s_{i,j}$ maps into the respective $S_{i,j}$ score. In order to reduce complexity, we upper bound the number of incoming edges to each WFST state by a constant hyperparameter $K$, only creating WFST edges that yield best candidate scores to a given $S_{i,j}$ according to Eq. 1. Other potential incoming edges are ignored, if any. In Fig. 1, WFST edges that would be built at infinitely great $K$ but rejected at $K = 2$ as not yielding competitive scores, are given in dashed line.

For each $j$ we create a new candidate transducer state $s_{n+j,j}$ that is only "accessible" from $s_{*,j-1}$ by teleportation and apply a fixed bonus $B$ to its score: $S_{n+j,j} = \max_k S_{k,j-1} + B$. Such a state (shown in yellow color in Fig. 1) may further be either mapped to a newly created state of the acceptor or deleted if it fails to improve the resulting alignment score. The bonus encourages initial endorsement of newly hypothesized states, which otherwise wouldn't have competitive scores.

**Phase 2 - endorsement.** After calculation of all $S_{i,j}$ scores, we trace paths of alignment from $S_{accepting,|y|}$ back to $S_{start,0}$. We *endorse* all WFSA edges laying at WFST-to-WFSA projections of those optimal and near-optimal paths. *Endorsement* here means increasing weight of an existing or prospective WFSA edge $k \xrightarrow{y_j} i$ by an amount of $\phi(k \xrightarrow{y_j} i)$, which is *endorsement flow* trough the WFST edge $s_{k,j} \mapsto s_{i,j-1}$ that maps back to $k \xrightarrow{y_j} i$ at some $j$. If multiple WFST edges map to a single $k \xrightarrow{y_j} i$ (in a case of looped path), the latter receives multiple endorsements. If an edge ought to be endorsed doesn't yet exist (which happens in case of teleportation) then we create it just in time; such a procedure lets the WFSA grow. Endorsement flow $\phi$ is calculated by summing outgoing edge flows at each WFST state and then distributing through the incoming edges. The final (accepting) state is assumed to receive constant endorsement flow $R$ from outside

[4] We track a modifiable $character \mapsto weight$ map for every edge

[5] Technically, for the sake of efficiency, one may handle the matrix as a sparse one, assuming the default score of any missing $(i, j)$ element to be $\max_k S_{k,j-1} - T$.

which is then distributed to all states using a dynamic programming procedure. $R$ plays part of learning rate. Choosing a reasonably low value for it ($0.1B$) helps one to alleviate bias caused by a particular order of sample learning.

The distribution of flow over incoming edges of a given state $s$ is determined by the softmax:[6]

$$\phi(e) = \frac{\exp(-L(e))}{\sum_{v \in V_s} \exp(-L(v))} \sum_{u \in U_s} \phi(u) \quad (2)$$

where $\{V_s\}$ and $\{U_s\}$ are sets of incoming and outgoing edges, respectively, for some state $s$; $e \in V_s$; $L(e)$ is *edge loss* which is calculated as difference between the respective $S_{i,j}$ score for $s$ and a candidate score brought to it by $e$ edge.

For negative training samples, we apply a similar procedure (*disendorsement*) with negative amounts of flow. The endorsement/disendorsement procedure described above keeps the sum of incoming edge weights and the sum of outgoing edge weights equal for any given WFSA state. Besides other benefits, that fact enables simple realization of unbiased sampling procedure, just as easy as random weighed selection of an edge to proceed with.

## 3 Experiments

### 3.1 Learning a WFSA from positive samples

The approach demonstrates ability to recognize basic branching and recurrent patterns in string sample sets that can be used to construct regular expressions. Fig. 2(a) illustrates an acceptor for the following set:

```
12345 , 1232345 , 123232345 ,
abfg , abcdefg , abcdecdecdefg ,
123456, 123232323232323456
```

### 3.2 Contrastive learning of WFSA

We experimented with training acceptors on both positive and negative examples. At every epoch, for each sample, we applied a respective positive or negative endorsement to the edges. Although we started with a *symmetrical* approach by endorsing proportionally to the score differences, it made any graph prone to any reshaping necessary for better fitting to a training set. Therefore, we found *asymmetrical* approach to be significantly more efficient. For false positive matches we randomly set a negative score to some of edges constituting the best path of acceptor-to-sample alignment. This leads to a permanent denial of those edges, and alternative edges will be considered in further learning process to achieve a high enough score for positive samples. Our experiments showed that the acceptor easily learns combinatorial rules like one that requires it to accept "`aaabbb`, `aaaccc`, `dddccc`" but to reject `dddbbb`". For most cases, it achieves a 100% accuracy in less than 20 epochs.

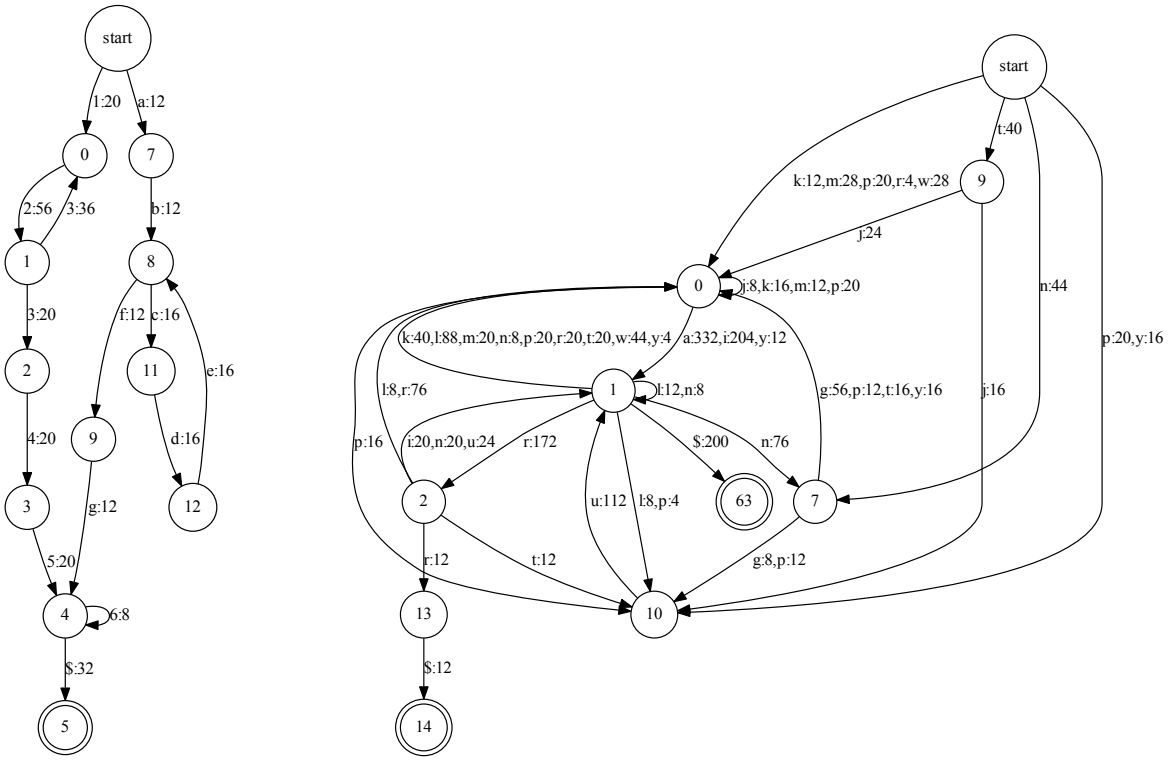
### 3.3 Probabilistic language modeling

We created WFSAs on the basis of small samples of vocabularies of the English (Germanic, IE) and the Kukatja (Pama-Nyungan, Australian Aboriginal) languages.[7] For the latter, we reused the linguistic resources introduced in Shcherbakov et al. (2016). We generated "novel" possible words using the acceptors and measured precision of the outputs, i.e. percentage of predicted strings that were observed in the corresponding language dictionaries. To generate a word, we traversed a path in the weighted acceptor from its accepting state back to the initial state, choosing an incoming edge and a character to generate at each state. In our experiments, the likelihood of test vocabulary word production was comparable with the one obtained in modified Kneser-Ney approach (Heafield et al., 2013) if the training corpus size is large enough. However, for very low-resource settings ($10 \ldots 500$ training words) the alignment-based approach outperforms n-gram approaches (at the best fixed $n$) in $\sim 1.2+$ times. Fig. 2(b) shows a graph trained for a random 300 words sampled from Kukatja dictionary. Although the automaton is quite simple, as many as 10% of its output words hit the remaining known Kukatja vocabulary that contains $\sim 9,000$ words.[8] A larger automaton may be built if we increase $L$ hyperparameter, but this choice does not lead to precision increase. Interestingly, a model trained on the English poem "Humpty Dumpty" (only 18 words!) predicts real English words with a precision above 18% which is 1.5x greater than the best result achieved by

[6] Minor terms below a threshold are ignored in the implementation in order to avoid excessive growth of the WFSA and performance loss.

[7] Kukatja is an Australian Aboriginal language spoken by about 300 people in Western Desert, Australia.

[8] Since the real vocabulary size of Kukatja is not known, we may reasonably expect that the real precision is greater, even exceeding one measured for English.




(a) Example in Subsection 3.1

(b) 90 words of Kukatja (paths with low weight edges were pruned)

Figure 2: Examples of the learned WFSA.

the modified Kneser-Ney model. Finally, since finite state machines are well-known in the domain of morphology modeling, we additionally explored this direction. We used short (100..1000 words) samples of CELEX English morphology database (Baayen et al., 1995) to train an acceptor. We observed that common prefixed and suffixes got aligned and represented compactly in a resulting automaton structure. However, overfitting to stems significantly affects its overall generalization potential that results is perplexity values comparable to ones achieved by n-gram models (Table 1).

| Resource (language) | Test set | | | Train set | |
|---|---|---|---|---|---|
| | K=1 | K=2 | 2gr. | K=1 | 2gr. |
| Kukatja | 1.79 | 1.77 | 1.66 | 1.9 | 1.63 |
| English | 2.54 | 2.46 | 2.32 | 2.4 | 2.13 |
| CELEX | 2.69 | 2.66 | 2.55 | 2.28 | 2.18 |

Table 1: Per-character perplexity (natural logarithms) for acceptors trained on 300-word random subsamples at different settings of $K$ hyperparameter. Figures for a bi-gram model ("2gr.") are given for comparison.

### 3.4 Learning software control flow graphs

Reverse engineering of a software control flow graph (CFG) based on samples of execution traces is another area where the proposed constructive alignment algorithm performs well. Experiments with sample programs demonstrated its ability to produce accurate estimations of CFG suitable for analysis and verification purposes.

Consider a simple abstract program example given in Figure 3. Suppose that each abstract action writes a respective character to a log upon execution. In such a way, every program run produces an execution trace string. For instance, given $num = 1, c_1 = c_2 = c_3 = false$ it will produce a `bcfefq` trace. An automaton learnt from a collection of traces obtained at different inputs is shown in Fig. 3. The result correctly back engineers the program control flow graph. It may be noted that procedure call context was properly preserved. Indeed, two different edges labelled 'f' were produced for procedure `f` calls originating from different points of code.

Speaking more generally, it may concern dis-

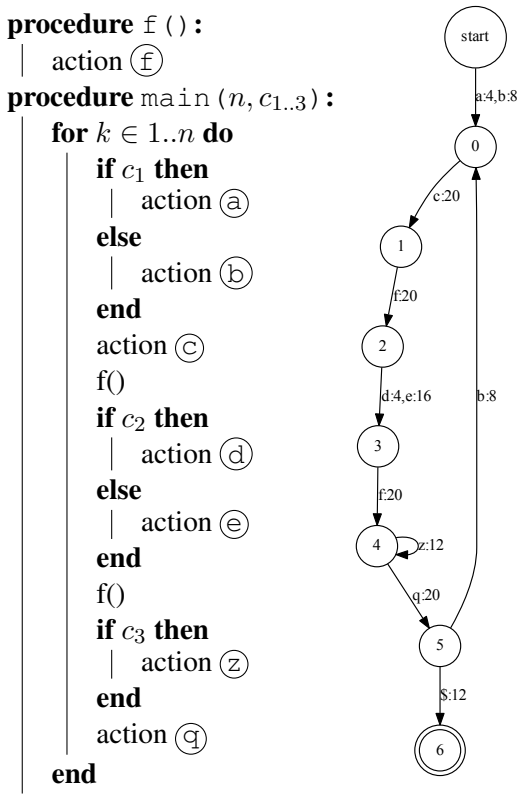



Figure 3: A control flow graph inferred from a code sample execution traces.

covering latent states of transition graph for various natural and artificial processes (process mining).

## 4 Discussion

The constructive alignment algorithm demonstrated its ability to efficiently solve various kinds of tasks when training sets are too small to employ other options. This fact supports the hypothesis that most of the acceptable sequences are likely to congest within low edit distances from a relatively sparse variety of alterable common patterns.

The method also demonstrated its advantages over n-gram language modeling for "dense" training resources, still its higher complexity may challenge the advantages as a kind of "light" learning algorithms. The approach seem to be promising for regular expression construction since it addresses repetitions and branching in an intuitive, uniform way of alignment alternative trade-off.

## 5 Related work

Tomita (1982) made an early successful attempt to address automaton construction from sets of positive and negative samples. Their approach employs a global optimization over edge mutations. Dupont (1994) proposed a further adaptation of genetic algorithms to the task of automata creation. Formalization of a regular grammatical inference task was also a major contribution. Brauer et al. (2011) employed a feature-based approach which constructed a query answer that had the shortest possible description length. Bui and Zeng-Treitler (2014) applied the Smith–Waterman algorithm (Smith et al., 1981) over sequences of keys to align input positive samples. A dedicated "primary" sample string was aligned to all other strings and then a branching graph was produced. The authors proposed to use a machine learning classifier to derive optimally performing sequences of keys from given samples. Recently Shcherbakov (2016) proposed to build a regular expression acceptor by incrementally aligning input samples to a topologically sorted representation of the automaton. It essentially did fuzzy alignment of characters dynamically reducing character sets to character classes at corresponding score penalties.

Finally, Fernau (2009) used a simplified blockwise alignment procedure and a dedicated generalization for the loop creation. The construction process combines *generating* prospective patterns with *inferring* them from input samples.

## 6 Conclusion

We proposed an approach to language modeling that constructs an acceptor driven by optimization of minimum edit distance between a learning sample and a path in the automaton. We carried out a series of preliminary experiments demonstrating its effectiveness in four major areas of the algorithm's potential application, particularly, for low-resource task settings.

## References

R Harald Baayen, Richard Piepenbrock, and Leon Gulikers. 1995. The celex lexical database (release 2). *Distributed by the Linguistic Data Consortium, University of Pennsylvania*.

Falk Brauer, Robert Rieger, Adrian Mocan, and Wojciech M Barczynski. 2011. Enabling information extraction by inference of regular expressions from sample entities. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1285–1294. ACM.

Duy Duc An Bui and Qing Zeng-Treitler. 2014. Learning regular expressions for clinical text classification. *Journal of the American Medical Informatics Association*, 21(5):850–857.

Pierre Dupont. 1994. Regular grammatical inference from positive and negative samples by genetic search: the gig method. In *International Colloquium on Grammatical Inference*, pages 236–245. Springer.

Henning Fernau. 2009. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521–541.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H Clark, and Philipp Koehn. 2013. Scalable modified kneser-ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 690–696.

Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. 1977. Perplexitya measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63.

Grzegorz Kondrak. 2005. N-gram similarity and distance. In *International symposium on string processing and information retrieval*, pages 115–126. Springer.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Ming Li and Paul Vitányi. 2013. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In *Formal grammar*, pages 90–108. Springer.

Andrei Shcherbakov. 2016. A branching alignment-based synthesis of regular expressions. In *AIST (Supplement)*, pages 315–328.

Andrei Shcherbakov, Ekaterina Vylomova, and Nick Thieberger. 2016. Phonotactic modeling of extremely low resource languages. In *Proceedings of the Australasian Language Technology Association Workshop 2016*, pages 84–93.

Temple F Smith, Michael S Waterman, et al. 1981. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197.

Masaru Tomita. 1982. Learning of construction of finite automata from examples using hill-climbing. rr: Regular set recognizer. Technical report, Carnegie-Mellon University Pittsburgh, PA, Dept of Computer Science.