UNRAVEL—A Decipherment Toolkit

Malte Nuhn and **Julian Schamper** and **Hermann Ney**
Human Language Technology and Pattern Recognition
Computer Science Department, RWTH Aachen University, Aachen, Germany
<surname>@cs.rwth-aachen.de

Abstract

In this paper we present the UNRAVEL toolkit: It implements many of the recently published works on decipherment, including decipherment for deterministic ciphers like e.g. the ZODIAC-408 cipher and Part two of the BEALE ciphers, as well as decipherment of probabilistic ciphers and unsupervised training for machine translation. It also includes data and example configuration files so that the previously published experiments are easy to reproduce.

1 Introduction

The idea of applying decipherment techniques to the problem of machine translation has driven research on decipherment in the recent time. Even though the theoretical knowledge has been published in the form of papers there has not been any release of software until now. This made it very difficult to follow upon the recent research and to contribute new ideas. With this publication we want to share our implementation of two important decipherment algorithms: Beam search for deterministic substitution ciphers and beamed EM training for probabilistic ciphers. It is clear that the field of decipherment is still under heavy research and that the true value of this release does not lie in the current implementations themselves, but rather in the opportunity for other researchers to contribute their ideas to the field.

2 Overview

Enciphering a plaintext into a ciphertext can be done using a myriad of encipherment methods. Each of these methods needs its own customized tools and tweaks in order to be deciphered automatically. The goal of UNRAVEL is not to provide a solver for every single encipherment method, but rather to provide reusable tools that can be applied to unsupervised learning for machine translation.

UNRAVEL contains two tools: DET-UNRAVEL for decipherment of deterministic ciphers, and EM-UNRAVEL for EM decipherment for probabilistic substitution ciphers and simple machine translation tasks. A comparison of both tools is given in Table 1.

The code base is implemented in C++11 and uses many publicly available libraries: The GOOGLE-GLOG logging library is used for all logging purposes, the GOOGLE-GFLAGS library is used for providing command line flags, and the GOOGLETEST library is used for unit testing and consistency checks throughout the code base.

Classes for compressed I/O, access to OpenFST (Allauzen et al., 2007), access to KENLM(Heafield, 2011), representing mappings, n-gram counts, vocabularies, lexicons, etc. are shared across the code base.

For building we use the GNU build system. UNRAVEL can be compiled using GCC, ICC, and CLANG on various Linux distributions and on MacOS X. Scripts to download and compile necessary libraries are also included: This makes it easy to install UNRAVEL and its dependencies in different computing environments.

Also, configuration- and data files (if possible from a license point of view) for various experiments (see Section 4.2 and Section 5.2) are distributed. Amongst others this includes setups for the ZODIAC-408 and Part two of the BEALE ciphers (deterministic ciphers), as well as the OPUS corpus and the VERBMOBIL corpus (probabilistic cipher/machine translation).

3 Related Work

We list the most important publications that lead to the implementation of UNRAVEL: Regarding DET-

UNRAVEL, the following literature is relevant: Hart (1994) presents a tree search algorithm for simple substitution ciphers with known word segmentations. The idea of performing a tree search and looking for mappings fulfilling consistency constraints was later adopted to n-gram based decipherment in an A* search approach presented by Corlett and Penn (2010). DET-UNRAVEL implements the beam search approach presented by Nuhn et al. (2013) together with the refinements presented in (Nuhn et al., 2014). The Bayesian approach presented by Ravi and Knight (2011a) to break the ZODIAC-408 cipher is not implemented, but configuration and data to solve the ZODIAC-408 cipher with DET-UNRAVEL is included. Also it is worth noting that Hauer et al. (2014) provided further work towards homophonic decipherment that is not included in UNRAVEL.

The EM training for the decipherment of probabilistic substitution ciphers, as first described by Lee (2002) is implemented in EM-UNRAVEL together with various improvements and extensions: The beam- and preselection search approximations presented by Nuhn and Ney (2014), the context vector based candidate induction presented by Nuhn et al. (2012), as well as training of the simplified machine translation model presented by Ravi and Knight (2011b).

4 Deterministic Ciphers: DET-UNRAVEL

Given an input sequence f_1^N with tokens f_n from a vocabulary V_f and a language model of a target language $p(e_1^N)$ with the target tokens from a target vocabulary V_e, the task is to find a mapping function $\phi : V_f \rightarrow V_e$ so that the language model probability of the decipherment $p(\phi(f_1)\phi(f_2)\ldots\phi(f_N))$ is maximized.

DET-UNRAVEL solves this optimization problem using the beam search approach presented by Nuhn et al. (2013): The main idea is to structure all partial ϕs into a search tree: If a cipher contains $|V_f|$ unique symbols, then the search tree is of height $|V_f|$. At each level a decision about the n-th symbol is made. The leaves of the tree form full hypotheses. Instead of traversing the whole search tree, beam search traverses the tree top to bottom and only keeps the most promising candidates at each level. Table 2 shows the important parameters of the algorithm.

4.1 Implementation Details

During search, our implementation keeps track of all partial hypotheses in two arrays H_s and H_t. We use two different data structures for the hypotheses in H_s and the hypotheses in H_t: H_s contains the full information of the current partial mapping ϕ. The candidates in the array H_t are generated by augmenting hypotheses from the array H_s by just one additional mapping decision $f \rightarrow e$ and thus we use a different data structure for these hypotheses: They contain the current mapping decision $f \rightarrow e$ and a pointer to the parent node in H_s. This saves memory in comparison to storing the complete mapping at every point in time and is faster than storing the mapping as a tree, which would have to be traversed for every score estimation.

The fact that only one additional decision is made during the expansion process is also used when calculating the scores for the new hypothesis: Only the additional terms of the final score for the current partial hypothesis ϕ are added to the predecessor score (i.e. the scheme is $score_{new} = score_{old} + \delta$, where $score_{old}$ is independent of the current decision $f \rightarrow e$).

The now scored hypotheses in H_t (our implementation also includes the improved rest cost es-

Aspect	Deterministic Ciphers: DET-UNRAVEL	Probabilistic Ciphers: EM-UNRAVEL
Search Space	Mappings ϕ	Substitution tables $\{p(f\|e)\}$
Training	Beam search over all ϕ. The order in which the decisions for $\phi(f)$ for each f are made is based on the extension order.	EM-training: In the E-step use beam search to obtain the most probable decipherments e_1^I for a given ciphertext sequence f_1^J. Update $\{p(f\|e)\}$ in M-step.
Decoding	Apply ϕ to cipher text.	Viterbi decoding using final $\{p(f\|e)\}$.
Experiments	ZODIAC-408, pt. two of BEALE ciphers	OPUS, VERBMOBIL

Table 1: Comparison of DET-UNRAVEL and EM-UNRAVEL.

timation as described in (Nuhn et al., 2014)) are pruned using different pruning strategies: Threshold pruning—given the best hypothesis, add a threshold score and prune the hypotheses with scores lower than best hypothesis plus this threshold score—and histogram pruning—which only keeps the best B_{histo} hypothesis at every level of the search tree. Further, the surviving hypotheses are checked whether they fulfill certain constraints $C(\phi)$ like e.g. enforcing 1-to-1 mappings during search.

Those hypotheses in H_t that survived the pruning step and the constraints check are converted to full hypotheses so that they can be stored in H_s. Then, the search continues with the next cardinality.

The order in which decisions about the symbols $f \in V_f$ are made during search (called *extension order*) can be computed using different strategies: We implement a simple *frequency sorting* heuristic, as well as a more advanced strategy that uses *beam search* to find an improved enumeration of $f \in V_f$, as presented in (Nuhn et al., 2014).

Our implementation expands the partial hypotheses in H_s in parallel: The implementation has been tested with up to 128 threads (on a 128 core machine) with parallelization overhead of less than 20%.

4.2 Experiments

The configurations for decoding the ZODIAC-408 cipher as well as Part two of the BEALE ciphers are almost identical: For both setups we use an 8-gram character language model trained on a subset of the English Gigaword corpus (Parker et al., 2011). We obtain n-gram counts (order 2 to 8) from the input ciphers and pass these to DET-UNRAVEL. In both cases we use the improved heuristic together with the improved extension order as presented in (Nuhn et al., 2014).

For the ZODIAC-408, using a beam size $B_{hist} = 26$ yields 52 out of 54 correct mappings. For the Part two of the BEALE ciphers a much larger beam size of $B_{hist} = 10$M yields 157 correct mappings out of 185, resulting in an error rate on the string of 762 symbols is 5.4 %.

5 Probabilistic Ciphers: EM-UNRAVEL

For probabilistic ciphers, the goal is to find a probabilistic substitution table $\{p(f|e)\}$ with normalization constraint $\forall_e \sum_f p(f|e) = 1$. Learning this table is done iteratively using the EM algorithm (Dempster et al., 1977).

Each iteration consists of two steps: Hypothesis generation (E-Step) and retraining the table $\{p(f|e)\}$ using the posterior probability $p_j(e|f_1^J)$ that *any* translation e_1^I of f_1^J has the word e aligned to the source word f_j (M-Step).

From a higher level view, EM-UNRAVEL can be seen as a specialized word based MT decoder that can efficiently generate and organize *all* possible translations in the E-step, and efficiently retrain the model $\{p(f|e)\}$ on *all* these hypotheses in the M-step.

5.1 Implementation Details

In contrast to DET-UNRAVEL, EM-UNRAVEL processes the input corpus sentence by sentence. For each sentence, we build hypotheses e_1^I from left to right, one word at a time:

First, the empty hypothesis is added to a set of currently active partial hypotheses. Then, for each partial hypothesis, a new source word is chosen such that local reordering constraints are fulfilled. For this, a coverage vector (which encodes the words that have already been translated) has to be updated for each hypothesis. Once the current source word to be translated next has been chosen, hypotheses for all possible translations of this source word are generated and scored. After having processed the entire set of partial hypotheses, the set of newly generated hypotheses is

Name	Description
Pruning	
B_{hist}	Histogram pruning. Only the best B_{hist} hypotheses are kept.
B_{thres}	Threshold pruning. Hypotheses with scores S worse than $S_{best} + B_{thres}$, where S_{best} is the score of the best hyptohesis, are pruned.
Constraints	
$C(\phi)$	Substitution constraint. Hypotheses not fulfilling the constraint $C(\phi)$ are discarded from search.
Extension Order	
V_{ext}	Extension order. Enumeration of the vocabulary V_f in which the search tree over all ϕ is visited.
B_{hist}^{ext}	Histogram Pruning for extension order search.
W_n^{ext}	Weight for $n-$gram language model lookahead score.

Table 2: Important parameters of DET-UNRAVEL.

pruned: Here, the partial hypotheses are organized and pruned with respect to their cardinality. For each cardinality, we keep the B_{histo} best scoring hypotheses.

Similarly to DET-UNRAVEL, the previously described expansion and pruning step is implemented using two arrays H_s and H_t. However, in EM-UNRAVEL the partial hypotheses in H_s and H_t use the same data structures since—in contrast to DET-UNRAVEL—recombination of hypotheses is possible.

In the case of large vocabularies it is not feasible to keep track of *all* possible substitutions for a given source word. This step can also be approximated using the preselection technique by Nuhn and Ney (2014): Instead of adding hypotheses for all possible target words, only a small subset of possible successor hypotheses is generated: These are based on the current source word that is to be translated, as well as the current language model state.

Once the search is completed we compute posteriors on the resulting word graph and accumulate those across all sentences in the corpus. Having finished one pass over the corpus, the accumulated posteriors are used to re-estimate $\{p(e|f)\}$ and the next iteration of the EM algorithm begins. Also, with every new parameter table $\{p(e|f)\}$, the Viterbi decoding of the source corpus is computed.

Name	**Description**
Pruning	
B_{hist}	Histogram pruning. Only the best B_{hist} hypotheses are kept.
Preselection Search	
B^{lex}_{cand}	Lexical candidates. Try only the best B^{lex}_{cand} substitutions e for each word f based on $p(f|e)$
B^{LM}_{cand}	LM candidates. Try only the best B^{LM}_{hist} successor words e with respect to the previous hypothesis' LM state.
Translation Model	
W_{jump}	Jump width. Maximum jump size allowed in local reordering.
C_{jump}	Jump cost. Cost for non-monotonic transitions.
C_{ins}	Insertion cost. Cost for insertions of words.
M_{ins}	Maximum number of insertions per sentence.
C_{del}	Deletion cost. Cost for deletions of words.
M_{del}	Maximum number of of deletions per sentence.
Other	
λ_{lex}	Lexical smoothing parameter.
N_{ctx}	Number of candidate translations allowed in lexicon generation in context vector step.

Table 3: Important parameters of EM-UNRAVEL.

While full EM training is feasible and gives good results for the OPUS corpus, Nuhn et al. (2012) suggest to include a context vector step in between EM iterations for large vocabulary tasks.

Using the Viterbi decoding of the source sequence from the last E-step and the corpus used to train the LM, we create normalized context vectors for each word e and f. The idea is that vectors for words e and f that are translations of each other are similar. For each word $f \in V_f$, a set of candidates $e \in V_e$ can be computed. These candidates are used to initialize a new lexicon, which is further refined using standard EM iterations afterwards.

Both, EM training and the context vector step are implemented in a parallel fashion (running in a single process). Parallelization is done on a sentence level: We successfully used our implementation with up to 128 cores.

5.2 Experiments

We briefly mention experiments on two corpora: The OPUS corpus and the VERBMOBIL corpus.

The OPUS corpus is a subtitle corpus of roughly 100k running words. Here the vocabulary size of the source language (Spanish) is 562 and the target language (English) contains 411 unique words. Using a 3-gram language model UNRAVEL achieves 19.5 % BLEU on this task.

The VERBMOBIL corpus contains roughly 600k running words. The target language vocabulary size is 3,723 (English) and the source language vocabulary size is 5,964 (German). Using a 3-gram language model and the context vector approach, UNRAVEL achieves 15.5 % BLEU.

6 Download and License

UNRAVEL can be downloaded at `www.hltpr.rwth-aachen.de/unravel`. UNRAVEL is distributed under a custom open source license. This includes free usage for noncommercial purposes as long as any changes made to the original software are published under the terms of the same license. The exact formulation is available at the download page for UNRAVEL.

We have chosen to keep this paper independent of actual implementation details such as method- and parameter names. Please consult the `README` files and comments in UNRAVEL's source code for implementation details.

7 Conclusion

UNRAVEL is a flexible and efficient decipherment toolkit that is freely available to the scientific community. It implements algorithms for solving deterministic and probabilistic substitution ciphers.

We hope that this release sparks more interesting research on decipherment and its applications to machine translation.

References

[Allauzen et al.2007] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. Openfst: A general and efficient weighted finite-state transducer library. In Jan Holub and Jan Zdárek, editors, *CIAA*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer.

[Corlett and Penn2010] Eric Corlett and Gerald Penn. 2010. An exact A* method for deciphering letter-substitution ciphers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1040–1047, Uppsala, Sweden, July. The Association for Computer Linguistics.

[Dempster et al.1977] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, 39.

[Hart1994] George W Hart. 1994. To decode short cryptograms. *Communications of the ACM*, 37(9):102–108.

[Hauer et al.2014] Bradley Hauer, Ryan Hayward, and Grzegorz Kondrak. 2014. Solving substitution ciphers with combined language models. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2314–2325. Dublin City University and Association for Computational Linguistics.

[Heafield2011] Kenneth Heafield. 2011. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July. Association for Computational Linguistics.

[Lee2002] Dar-Shyang Lee. 2002. Substitution deciphering based on hmms with applications to compressed document processing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(12):1661–1666.

[Nuhn and Ney2014] Malte Nuhn and Hermann Ney. 2014. Em decipherment for large vocabularies. In *Annual Meeting of the Assoc. for Computational Linguistics*, pages 759–764, Baltimore, MD, USA, June.

[Nuhn et al.2012] Malte Nuhn, Arne Mauser, and Hermann Ney. 2012. Deciphering foreign language by combining language models and context vectors. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 156–164, Jeju, Republic of Korea, July. Association for Computational Linguistics.

[Nuhn et al.2013] Malte Nuhn, Julian Schamper, and Hermann Ney. 2013. Beam search for solving substitution ciphers. In *Annual Meeting of the Assoc. for Computational Linguistics*, pages 1569–1576, Sofia, Bulgaria, August.

[Nuhn et al.2014] Malte Nuhn, Julian Schamper, and Hermann Ney. 2014. Improved decipherment of homophonic ciphers. In *Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, October.

[Parker et al.2011] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English gigaword fifth edition. Linguistic Data Consortium, Philadelphia.

[Ravi and Knight2011a] Sujith Ravi and Kevin Knight. 2011a. Bayesian inference for Zodiac and other homophonic ciphers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 239–247, Portland, Oregon, June. Association for Computational Linguistics.

[Ravi and Knight2011b] Sujith Ravi and Kevin Knight. 2011b. Deciphering foreign language. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 12–21, Portland, Oregon, USA, June. Association for Computational Linguistics.