Genetic Algorithms in the Brill Tagger

Moving towards language independence

Johannes Bjerva

Department of Linguistics
Thesis (15 ECTS credits)
Degree of Master of Arts in Computational Linguistics (1 year, 60 ECTS credits)
Spring 2013
Supervisors: Kristina Nilsson Björkenstam and Robert Östling
Examiner: Henrik Liljegren

Stockholms universitet

Genetic Algorithms in the Brill Tagger

Abstract

The viability of using rule-based systems for part-of-speech tagging was revitalised when a simple rule-based tagger was presented by Brill (1992). This tagger is based on an algorithm which automatically derives transformation rules from a corpus, using an error-driven approach. In addition to performing on par with state of the art stochastic systems for part-of-speech tagging, it has the advantage that the automatically derived rules can be presented in a human-readable format.

In spite of its strengths, the Brill tagger is quite language dependent, and performs much better on languages similar to English than on languages with richer morphology. This issue is addressed in this paper through defining rule templates automatically with a search that is optimised using Genetic Algorithms. This allows the Brill GA-tagger to search a large search space for templates which in turn generate rules which are appropriate for various target languages, which has the added advantage of removing the need for researchers to define rule templates manually.

The Brill GA-tagger performs significantly better ($p < 0.001$) than the standard Brill tagger on all 9 target languages (Chinese, Japanese, Turkish, Slovene, Portuguese, English, Dutch, Swedish and Icelandic), with an error rate reduction of between 2% – 15% for each language.

Keywords

Genetic Algorithms, Language Independent Part-of-Speech Tagging, Transformation-Based Learning

Sammendrag

Da Brill (1992) presenterte sin enkle regelbaserte ordklasse-tagger ble det igjen aktuelt å bruke regelbaserte system for tagging av ordklasser. Taggerens grunnlag er en algoritme som automatisk lærer seg transformasjonsregler fra et korpus. I tillegg til at taggeren yter like bra som moderne stokastiske metoder for ordklasse-tagging har Brill-taggeren den fordelen at reglene den lærer seg kan presenteres i et format som lett kan oppfattes av mennesker.

Til tross for sine styrker er Brill-taggeren relativt språkavhengig ettersom den fungerer mye bedre for språk som ligner engelsk enn språk med rikere morfologi. Denne oppgaven forsøker å løse dette problemet gjennom å definere regelmaler automatisk med et søk som er optimert med Genetiske Algoritmer. Dette lar Brill GA-taggeren søke gjennom et mye større område enn den ellers kunne ha gjort etter maler som i sin tur genererer regler som er tilpasset målspråket, hvilket også har fordelen at forskere ikke trenger å definere regelmaler manuelt.

Brill GA-taggeren yter signifikant bedre ($p < 0.001$) enn Brill-taggeren på alle 9 målspråk (Kinesisk, Japansk, Tyrkisk, Slovensk, Portugisisk, Engelsk, Nederlandsk, Svensk og Islandsk), med en feilprosent som er mellom 2% og 15% lavere i alle språk.

Emneord

Genetiske Algoritmer, Språkuavhengig Ordklasstagging, Transformasjonsbasert Innlæring

Sammanfattning

När Brill (1992) presenterade sin enkla regelbaserade ordklasstaggare blev det återigen aktuellt att använda regelbaserade system för taggning av ordklasser. Taggaren är baserad på en algoritm som automatiskt lär sig transformationsregler från en korpus. Bortsett från att taggaren fungerar lika bra som moderna stokastiska metoder för ordklasstaggning har den också fördelen att reglerna som den lär sig kan presenteras i ett format som lätt kan läsas av människor.

Trots sina styrkor är Brill-taggeren relativt språkberoende i och med att den fungerar mycket bättre för språk som liknar engelska än för språk med rikare morfologi. Den här uppsatsen försöker att lösa detta problem genom att definiera regelmallar automatiskt med en sökning som är optimerad med Genetiska Algoritmer. Detta gör att Brill GA-taggaren kan söka genom ett mycket större område än den annars skulle ha kunnat göra efter mallar som i sin tur genererar regler som är anpassade för målspråket. Detta har också fördelen att forskare inte behöver definiera regelmallar manuellt.

Brill GA-taggeren får signifikant bättre träffsäkerhet ($p < 0.001$) än Brill-taggeren på alla 9 målspråken (Kinesiska, Japanska, Turkiska, Slovenska, Portugisiska, Engelska, Nederländska, Svenska och Isländska), med en felprocent som är mellan 2% och 15% lägre för alla språk.

Nyckelord

Genetiska Algoritmer, Språkoberoende Ordklasstaggning, Transformationsbaserad Inlärning

Table of Contents

List of Figures

List of Tables

1 Introduction

Part-of-Speech (PoS) tagging is the task of labelling every word in a sequence of words with a tag indicating what lexical syntactic category it assumes in the given sequence. Having tools which can successfully perform this task is crucial for higher-level language processing, such as machine translation and syntactic parsing. In order for such systems to be truly useful, they need to be applicable for a variety of languages. However, even though many state of the art PoS taggers claim to be language-independent, this is rarely the case. Bender (2009) points out that models such as n-gram models, although remarkably effective for languages similar to English, come with a *hidden language dependency*. This is due to the fact that languages with a more flexible word order than English, or with more complex morphology, suffer from data-sparseness when using language models based on n-grams.

The main topic of this paper is the transformation- and rule-based tagger proposed by Brill (1992). The Brill tagger in its original implementation is also quite language specific, as it does not perform well on e.g. inflectional languages. Previous work has shown that the Brill tagger can be made suitable for such languages, although this does not make the Brill tagger truly *language independent* (as defined by Bender (2009)), as parameters need to be adjusted in order for the tagger to be successful for different languages.

The aim of this thesis is to investigate whether an automatic search for the rule templates on which a Brill tagger relies can increase its language independence. This search will be made feasible by using Genetic Algorithms as an optimisation method. Corpora in 9 languages which differ in terms of morphological complexity and syntactical structure (Chinese, Japanese, Turkish, Slovene, Portuguese, English, Dutch, Swedish and Icelandic) are used for evaluation, so as to assert that the Brill GA-tagger is fairly language independent. In addition to providing a solution to this issue, the resulting tagger will be open-sourced under the MIT License.[1]

2 Background

The background material of this thesis covers four main areas. In the first section, the Brill tagger is outlined. In the second section, an introduction to Finite-State Automata and Finite-State Transducers is given. In the third section, Genetic Algorithms are presented. The fourth section contains an overview of how these methods have been applied to the Brill tagger previously. Following these sections, a summary of the aims of this work is given.

2.1 The Brill Tagger

PoS tagging can be successfully carried out with methods such as Hidden Markov Models (Cutting et al., 1992; Kupiec, 1992), Decision Trees (Schmid, 1994), rule-based methods (Brill, 1994, 1995; Loftsson, 2007), Maximum Entropy methods (Ratnaparkhi, 1996; Zhao et al., 2007), Neural Networks (Marques and Lopes, 2001), Conditional Random Fields (Lafferty et al., 2001), Averaged Perceptrons (Collins, 2002), Support Vector Machines (Mayfield et al., 2003) and Bilingual Graph-based Projections (Das and Petrov, 2011). The main topic of this paper is the transformation- and rule-based tagger proposed by Brill (1992). Although the prevailing methods used for PoS tagging were, and still are, stochastic in nature, Brill shows that his simple transformation-based system can perform on par with such PoS taggers for English. Brill's tagger achieved an error rate of approximately 5% for English, while more recent systems according to Manning (2011) generally achieve error rates in the neighbourhood of 3% for English.

[1] The MIT License template: http://opensource.org/licenses/MIT

The Brill tagger is fundamentally different from stochastic taggers, and to some extent also from older rule-based methods, in that it through an error-driven search *automatically* obtains a list of transformation rules. These are used to assign PoS tags to a given sequence of words, through transforming a given tag to a different tag in a specific context. The procedure of learning such rules is commonly referred to as Transformation Based Learning (TBL). In contrast, stochastic methods such as those based on Hidden Markov Models might amass a collection of conditional probabilities derived from n-grams of tags (e.g. $P(tag_c|tag_{ab})$). Although both simple and more sophisticated stochastic taggers can reach very high accuracies when assigning PoS tags, they lack an advantage that rule-based taggers possess, as stochastic taggers do not contain any explicit human-readable rules, but merely something akin to one or more massive probability matrices. Rule-based taggers, on the other hand, can easily present the rules they use in the tagging process in a comprehensible format. This transparency is of additional value for languages for which resources are sparse, as this allows for a more straight-forward analysis of the rules obtained. An outline of the Brill tagger's training phase can be seen in Figure 1.

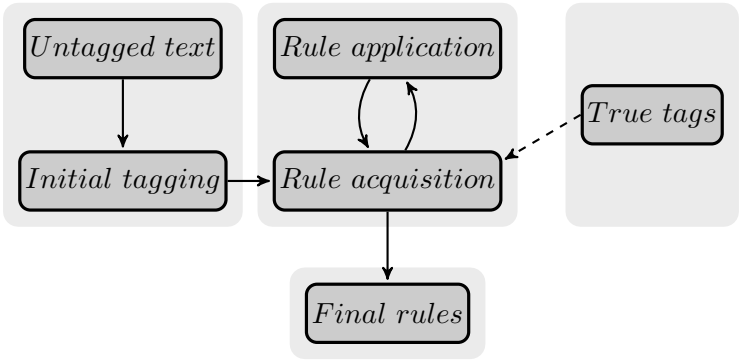

Figure 1: Outline of the Brill tagger's training phase

2.1.1 Initial Tagging

In the Brill tagger's initial tagging phase words are assigned PoS tags based on non-contextual features. First, each word is assigned the most frequent tag of that word in the training material, as shown in the following examples.[1]

(1)	Every	minute	counts
	DT	NN	VBZ
(2)	Every	minute	detail
	DT	*NN	NN

Since the tagger always chooses the most common tag to a word, this leads to errors of the type that can be seen in the example above, where *minute* has been erroneously tagged as a noun in (2), where it should be an adjective.

Words that are not found in the training material are handled separately, and can be assigned tags depending on either manually defined or automatically derived features of words. For instance, words could be tagged depending on their suffixes (or other language-dependent tell-tale signs). Words not fitting any category after this process are assigned the generally most frequent tag in the training material (this would normally mean assigning the tag *NN* to such words when tagging Swedish or English). Since no contextual information is used in this stage, many words are likely to be tagged incorrectly (Brill, 1992).

[1]The PoS tag set used here is taken from the Penn Treebank (Marcus et al., 1993)

2.1.2 Transformation-Based Tagging

After this initial phase, the contextual error-driven tagger is applied. This tagger attempts to apply transformation rules in order to reduce the amount of tagging errors. Considering that the rules correct the mistakes made by the initial tagging, they are commonly referred to as *patches*. These rules are obtained automatically, and are made to fit one of several predetermined context-dependent rule templates. The following rule templates are used by Brill (1992):

Change **A** to **B** when:

i. Tag at relative position +/- 1 is **C**
ii. Tag at relative position +/- 2 is **C**
iii. One of the two following / preceding tags is **C**
iv. One of the three following / preceding tags is **C**
v. Preceding tag is **C**, and following tag is **D**
vi. Preceding or following two tags are **C** and **D**
vii. Current word is / is not capitalised
viii. Previous word is / is not capitalised

This list should in no way be seen as being complete, considering that any imaginable template can be used, be it simple or complicated. Adding more rule templates can only be beneficial for the tagger's accuracy (although detrimental to its training time). This is due to the fact that templates not resulting in any improvement will not yield rules with sufficiently high scores to be utilised. Furthermore, the risk of overtraining is fairly low in systems utilising TBL, such as the Brill tagger (Ramshaw and Marcus, 1994). Although the increased training time caused by adding more templates is an obstacle, a potential solution to this will be presented in Section 2.3.

When deciding which rules to add to the list of patches, a list of tagging errors is compiled, consisting of error triplets of the form $\langle tag_a, tag_b, n\rangle$ where n is the number of times a word is mistagged with tag_a when the correct tag is tag_b. For each such error triplet and each patch matching the given templates, the error reduction (new errors caused subtracted from old errors corrected) of the patch is calculated. The patch with the highest score is added to the patch list and applied to the training data. This process is repeated until a given accuracy threshold is reached, or no further rules with positive scores can be learned. Note that a patch changing tag_a to tag_b is only applied if the word in question is tagged at least once with tag_b in the training data (Brill, 1992).

2.1.3 Performance of the Brill Tagger

In its original form, the Brill tagger performs rather slowly when tagging new texts, being outperformed by far by conventional stochastic taggers. According to Roche and Schabes (1995), this can be explained by its inherent local non-determinism, which in part is caused by that rules might in some situations render each other useless.

(3) $A \rightarrow B$ IF NEXTTAG $= C$
(4) $B \rightarrow A$ IF PREVTAG $= C$

The example rules show (3) a rule that changes tag A to B if the next tag is C, and (4) a rule that changes tag B to A if the previous tag was C. If these rules are applied in order to the tag sequence CAC, the tags assigned will end up back at square one (CAC -> CBC -> CAC). Additionally, (3) requires the tagger to look ahead one step in the sentence it is tagging, when considering whether or not to apply the rule. This behaviour is the cause of local non-determinism in Brill's tagger, which is the main reason behind the excessively time-consuming tagging process, where the tagger needs at most RKn steps to tag a sequence of length n, using R rules, and requiring K words of context (Roche and Schabes, 1995).

2.2 Finite-State Automata

Finite-state automata (FSA) and finite-state transducers (FST) are two concepts which are necessary in order to understand the implementation by Roche and Schabes (1995). Hence, a sufficient introduction to such devices and the conventions of notations used in this paper will be given. Note that although the work of Moore (1956), Mealy (1955), Salomaa (1973) and Schützenberger (1977) gave rise to both the FSAs presented in this section and the FSTs presented in Section 2.2.2, the definitions used are mainly borrowed from Roche and Schabes (1995).

FSAs are essentially a formalism for modelling regular languages, equivalent to regular expressions (regex); any regex can be represented by an FSA and vice-versa (Kleene, 1956). An FSA can be represented by a directed graph, in which *nodes* represent *states* and *arcs* represent *transitions* (see Figure 2). In this paper arrows are used to indicate arcs, with their associated symbols shown above each arc. States are represented as grey circles, and are numbered using $q_{a...z}$. States drawn with a surrounding concentric circle indicate end states (such as state q_f in Figure 2). Unless otherwise stated, q_a denotes the initial state of the automaton.

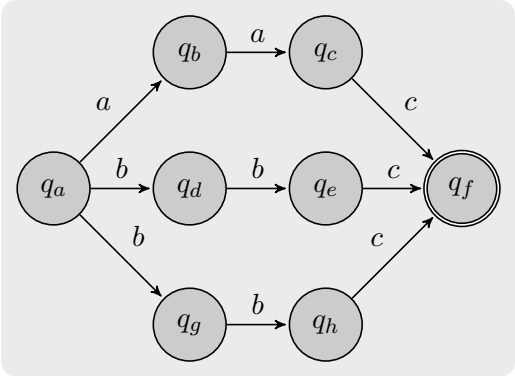

Figure 2: A simple finite-state automaton

An FSA can be used to accept or generate strings by following arcs between nodes until a node representing an end state is reached. As shown in Table 1, an FSA can be formally defined as a 5-tuple $\langle Q, \Sigma, q_0, F, \delta(q,i)\rangle$ (Roche and Schabes, 1995).

Table 1: Definition of a Finite-State Automaton

$Q = q_0 q_1 q_2 ... q_{N-1}$	a finite set of N states
Σ	a finite input alphabet of symbols
$q_0 \in Q$	the initial state
$F \subseteq Q$	the set of final states
$\delta(q,i)$	the transition function or transition matrix between states. given a state $q \in Q$ and an input symbol $i \in \Sigma$, $\delta(q,i)$ returns a new state $q' \in Q$. i.e. δ maps from $Q \times \Sigma$ to Q.

Certain properties found in FSAs provide them with their efficiency and flexibility. One such property is the fact that FSAs have been proven to have closure under several important set-operators, namely union ($\cup$), intersection ($\cap$), Kleene star (Σ^*), concatenation ($\cdot$) and complementation (Roche and Schabes, 1997). In contrast, other formalisms such as CFGs are not closed under complementation or intersection. As will be shown in Section 2.2.4, intersection and concatenation in particular are essential to the implementation of the tagger presented in this paper.

2.2.1 Determinising Finite-State Automata

A strength of FSAs is that for every non-deterministic FSA (NFA), there is an equivalent deterministic FSA (called DFA) (Jurafsky and Martin, 2009, p. 72). In a DFA, no combination of a state and input symbol can lead to more than one next state – or formally, if the automaton is in a state $q \in Q$, and the input read is a, then $\delta(q, a)$ uniquely determines the state q' to which the automaton traverses (Roche and Schabes, 1997). One possible DFA of the NFA in Figure 2 can be seen in Figure 3 (note that the states q_g and q_h have been removed).

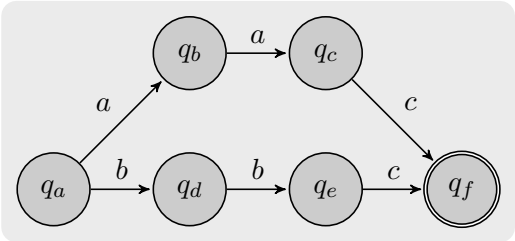

Figure 3: A possible deterministic version of the FSA in Figure 2

The efficiency of DFAs is a result of that following a single path through a DFA is computationally inexpensive, as the time needed to recognise a string is linearly proportional to its length (Roche and Schabes, 1995). This efficiency is especially clear when compared to the alternative of traversing all possible paths through an NFA, backtracking when necessary, until an end state is reached. In the latter case of an NFA, the time needed to recognise a string is dominated by the total amount of states and possible paths (Mohri, 1997).

Finally, FSAs can be turned into a minimal form, containing the smallest possible amount of states, although equivalent in every other way (Hopcroft, 1971; Hopcroft et al., 1979). This allows FSAs to be stored in a compact format, which is a further advantage of the implementation presented later in this paper. A minimal form of the DFA from Figure 3 can be seen in Figure 4.

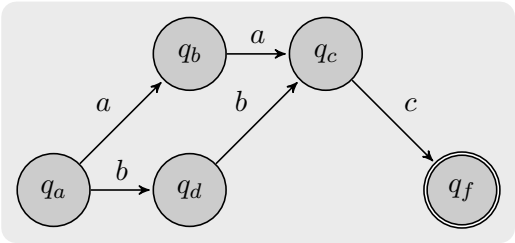

Figure 4: A minimal form of the FSA in Figure 3

2.2.2 Finite-State Transducers

An FST is similar to an FSA in most ways. However, rather than labelling arcs with a single symbol, arcs in an FST are labelled with two symbols, essentially mapping from one symbol to another (Jurafsky and Martin, 2009, p. 91). A simple example of an FST is shown in Figure 5.

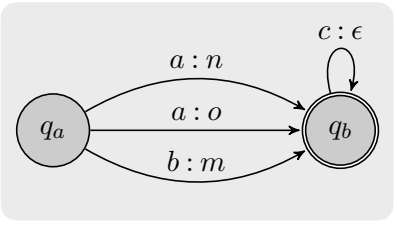

Figure 5: A simple finite-state transducer

Similarly to an FSA, an FST can also be seen as a machine which either generates or recognises an input string – albeit generating or recognising pairs of strings, rather than a single string. In addition to these properties, an FST can be seen as translating from one input string to an output string, or as a set relater, computing relations between sets (Jurafsky and Martin, 2009, p. 91). Furthermore, FSTs share the strength found in FSAs obtained from the various operators under which they have closure (Roche and Schabes, 1997). As shown in Table 2, an FST can be formally defined as a 6-tuple $\langle Q, \Sigma_1, \Sigma_2, q_0, F, \delta(q,i)\rangle$ (Roche and Schabes, 1995).

Table 2: Definition of a Finite-State Transducer

$Q = q_0 q_1 q_2 ... q_{N-1}$	a finite set of N states
Σ_1	a finite input alphabet of symbols
Σ_2	a finite output alphabet of symbols
$q_0 \in Q$	the initial state
$F \subseteq Q$	the set of final states
$\delta(q,i)$	the transition function between states given a state $q \in Q$ and an input symbol $i \in \Sigma_1^*$ $\delta(q,i)$ returns a set of states $Q' \in Q$.
$\sigma(q,i)$	the output function giving the set of output strings for each state and input given a state $q \in Q$ and an input symbol $i \in \Sigma_1^*$, $\sigma(q,i)$ returns a set of output strings with each string $s \in \Sigma_2^*$.

When computing the output of a transducer T_1, given an input sequence, one possible procedure entails traversing T_1 over all possible states until an end state is reached, backtracking when no transition from a state matches the current input symbol. Given the FST in Figure 6 and the input sequence a, b, one would first have to try and fail with the transition from q_a to q_b, backtracking back to q_a since the state q_b has no outgoing transitions matching the next symbol of the input sequence, before continuing on to the successful transitioning from q_a through q_d, and finally ending up at q_e.

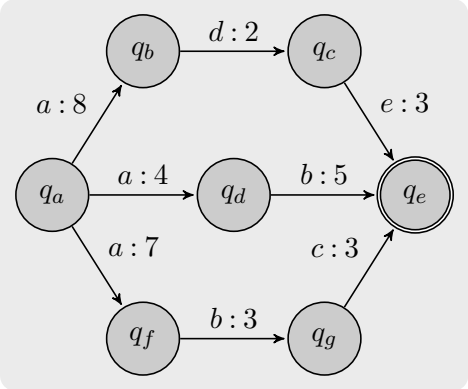

Figure 6: A more complex finite-state transducer

Another way of computing this output is based on seeing the input string as an FSA A_1 (see top left of Figure 7). Computing $A_1 \cap T_1$ thus results in a more compact representation, not including the paths that do not match A_1 (see bottom of Figure 7). However, the resulting FST needs to be pruned of nodes that do not lead to an end-state (see Figure 8), which is an operation of similar complexity to the aforementioned backtracking (Roche and Schabes, 1995). If, however, the FST

is determinised prior to this, this procedure can be carried out more efficiently, as no nodes need to be pruned off the resulting tree after calculating the intersection (Roche and Schabes, 1995).

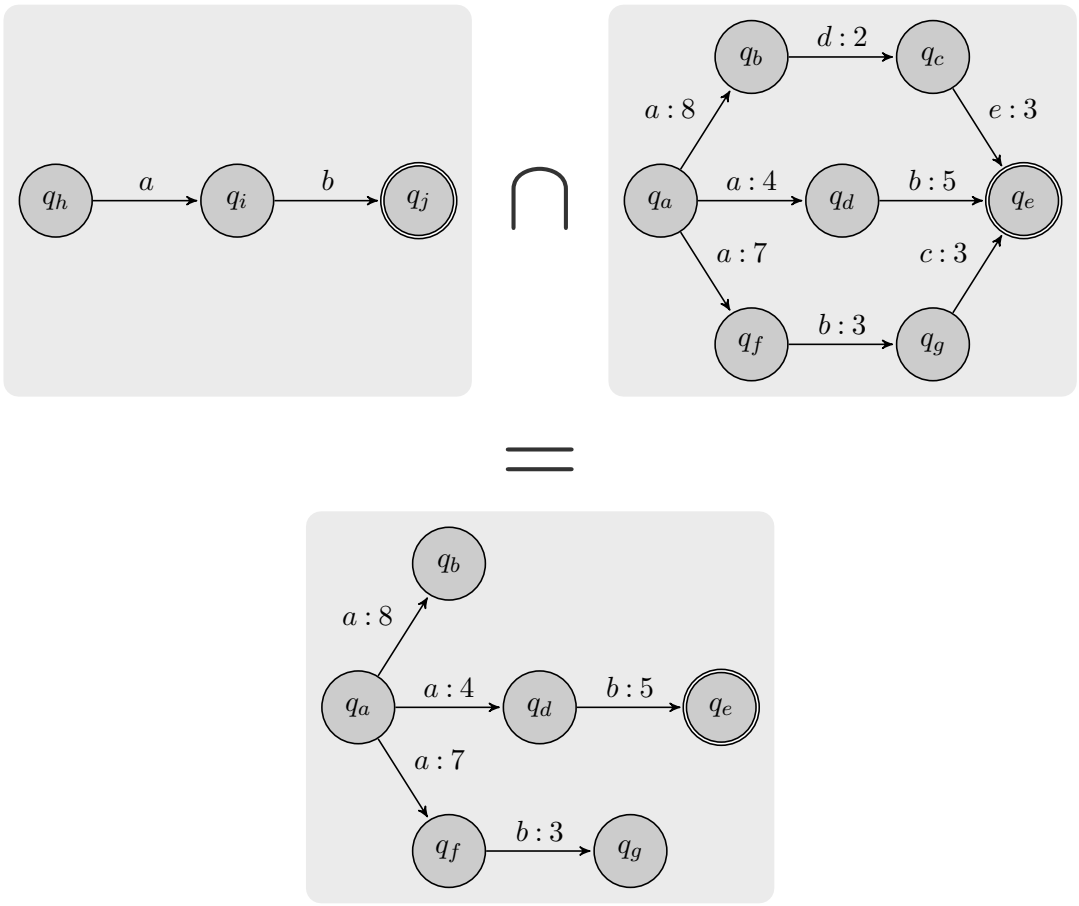

Figure 7: Intersection of an FSA and FST yielding a new FST

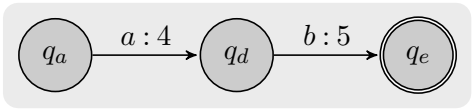

Figure 8: FST of Figure 7 pruned.

2.2.3 Sequential and p-Subsequential Transducers

Sequential transducers are transducers which are deterministic on their input (Mohri, 1997). In such transducers, the input of the outgoing arcs from any given state do not overlap. The output of a sequential transducer, however, may be non-deterministic. Such transducers are computationally interesting since their performance is only dependent on the size of the given input, and not the size of the transducer itself (Mohri, 1997). Using a sequential transducer simply entails following the unique path in which the input symbols of the arcs match the symbols of the given input string, while writing the output labels as they are uncovered. Provided the cost of writing these output labels does not depend on the label's length, the time complexity of this is $O(n)$ where n is the length of the provided input string.

Table 3: Definition of a Sequential Transducer

Q, Σ_1, Σ_2, q_0, F	defined as for FSTs
$\otimes$	the deterministic state transition function mapping $Q \times \Sigma_1$ to Q
$*$	the deterministic emission function mapping $Q \times \Sigma_1$ to Σ_2^*

The concept of sequential transducers can be developed further by adding in an optional concatenation of additional output strings at final states (Schützenberger, 1977, as cited in Mohri (1997)). That is to say, after a string has passed through a transducer, some other output string may be concatenated at the end of the obtained output string. Transducers which behave in this way are known as subsequential transducers (Mohri, 1997). Ambiguities encountered in natural language, such as ambiguity of lexemes, grammars and pronunciation dictionaries, cannot be taken into account with sequential transducers (Mohri, 1997). Extending the sub sequential transducers to p-subsequential transducers is a way of solving this issue (Mohri, 1994). According to Mohri (1997, p. 272), '[...] one cannot find any reasonable case in language in which the number of ambiguities would be infinite, [...] p-subsequential transducers seem to be sufficient for describing linguistic ambiguities'. A p-subsequential transducer is similar to a subsequential transducer in most respects, differing only in that p strings are concatenated onto the standard output, as opposed to one string. That is to say, a p-subsequential transducer where $p = 1$ is indeed a subsequential transducer.

Table 4: Definition of a Subsequential Transducer

Q, Σ_1, Σ_2, q_0, F	defined as for FSTs
$\otimes$	the deterministic state transition function mapping $Q \times \Sigma_1$ to Q
$*$	the deterministic emission function mapping $Q \times \Sigma_1$ to Σ_2^*
ρ	the final emission function mapping F to Σ_2^*

The advantage of this type of transducer lies in their efficiency, which is caused by the determinism found in their input. As with DFAs, this determinism allows them to be traversed in a time proportional to the length of the string to be processed. Furthermore, there are efficient algorithms for both determinising (Mohri, 1997) and minimising (Mohri, 2000) such devices.

2.2.4 Encoding the Brill Tagger as a Deterministic Finite-State Transducer

A conventional Brill tagger can be viewed as an NFA, which operates in RKn steps, where R is the amount of rules used by the tagger, K is the length of the context span required by the tagger and n is the length of the input sequence (Roche and Schabes, 1995). Implementing the tagger as a DFT allows the tagger to operate in n steps, thus dramatically increasing the performance to a level faster than some conventional HMM taggers (Roche and Schabes, 1995). This transformation operates in four steps: representing the rules as FSTs, transforming these FSTs into *local extensions*, combining these transducers into one transducer, and finally determinising the resulting FST. The following example rule will be used to illustrate parts of this transformation.

(5) $A \rightarrow B$ IF PREVTAG $= C$

Turning this rule into an FST would yield the result shown in Figure 9. As noted by Roche and Schabes (1995), this representation is inefficient since the transducer needs to be applied at every position of the given input sequence; using this FST to tag a corpus of length n would require at most rsn steps, where r is the number of rules and s is the number of states in each rule's transducer.

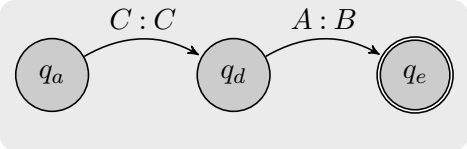

Figure 9: Representation of rule (3) as an FST

The second step is to transform the FSTs obtained from the first step so that they can be applied globally to the entire input sequence in one pass (Roche and Schabes, 1995). This transformation needs to be performed for each rule represented by an FST. If we have a transformation function f_1 which transforms e.g. n to m, the goal is to extend this to another function f_2 which allows this transformation to be applied multiple times over an input sequence. That is to say, a function which would apply this transformation to each place in a given sequence where the symbols match those of the transducer's input symbols. Such a function is known as the *local extension* of a transducer (Roche, 1993). The local extension of the FST from Figure 9 is shown in Figure 10. As noted by Roche and Schabes (1995), these transducers still need to be applied one after another, thus requiring rn steps to tag an input sequence, where n is the length of the sequence and r is the number of rules to apply.

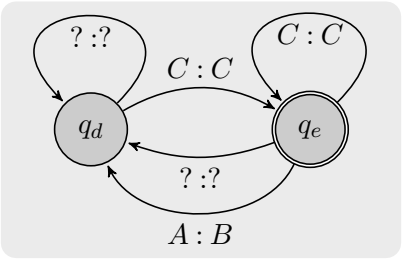

Figure 10: Local extension of the FST from Figure 9

The third step consists of performing the combination of these transducers, as mentioned in Section 2.2, through the application of the formal composition operation (denoted by $\circ$). This formalisation and the algorithm for computing this composition are presented and examined in more detail by Elgot and Mezei (1965).

The final step is the determinisation of this relatively complex FST consisting of the combination of all of the local extensions of the rules obtained by the tagger. Although such determinisation is not possible for all FSTs, Roche and Schabes (1995) present a proof which shows that the FSTs generated from rules used in a Brill tagger are always determinisable. This transformation results in a representation of the Brill tagger which can tag a given sequence in n steps, where n is the length of this sequence.

One major weakness does however remain after applying this procedure. The representation of Brill rules as FSTs using the method outlined by Roche and Schabes (1995) is not applicable to lexical rules, as the input and output tapes read by the device consist solely of PoS tags. This also means that rules containing more specific morphological changes are incompatible with this implementation. The importance of such rules is mentioned in Section 2.5.2, and a suggestion for how such features can be included is presented in Section 7.6.1.

2.3 Genetic Algorithms

The term *Genetic Algorithms* (GA) is used to refer to a family of computational models first investigated by Holland (1975), inspired by I. Rechenberg's ideas outlined a decade earlier in his *Evolution Strategies* (cf. Rechenberg (1994)). As the name implies, this type of model attempts to mimic natural evolution, by incorporating elements such as simulated natural selection (i.e. some semblance of *survival of the fittest*) as well as crossover and mutation (Poli et al., 2008). GAs have been found to be astonishingly efficient and useful tools for various search and optimisation problems within fields such as physics, bioinformatics and financial mathematics.

GAs can be seen as simply being a metaheuristic approach to refine a given search space. This allows GAs to find solutions much more efficiently than exhaustive search methods which might require the entirety of such a space to be investigated. This strength draws from the fact that samples from such a space are initially drawn at random in GAs. Next, the samples yielding the highest fitness are chosen, and are allowed to recombine with each other and produce offspring for the next search iteration. This procedure will efficiently pick samples from a multi-dimensional search space while eliminating areas of the search space which do not yield a high fitness, and simultaneously being highly resistant to becoming stuck in local optima (Goldberg, 1989; Sivanandam and Deepa, 2008; Whitley, 1994). It is, however, important to note that a GA will not necessarily find the global optimum, but rather an approximation or an *acceptably good* solution (Goldberg and Holland, 1988; Holland, 1975; Whitley, 1994).

2.3.1 The Canonical Genetic Algorithm

Before delving into various alternate implementations of GAs, it can be useful to investigate the *canonical genetic algorithm*, as detailed by Holland (1975). Briefly put, the flow of execution in the canonical genetic algorithm can be seen as a two-stage process, starting with the *current population* (which in the algorithm's first iteration is also the *initial population*). Selection is applied to this current population in order to create an *intermediate population*. Crossover and mutation are then applied to this intermediate population in order to create the *next population* (see Figure 11).

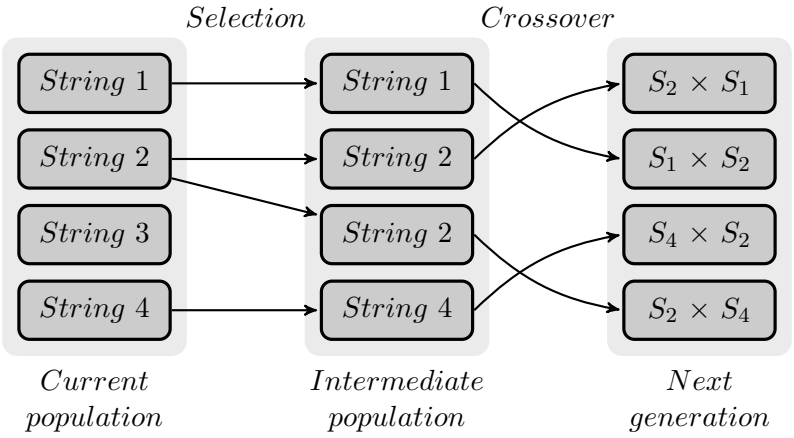

Figure 11: Outline of the canonical genetic algorithm. The crossover operation is indicated by $\times$.

The first step when implementing this canonical genetic algorithm is to generate an initial population. Each individual is represented by a binary string (Whitley, 1994), which should have as short a length as the problem permits. In the case of transformation templates, each bit might denote whether or not a PoS tag in a given position is interesting. After this initial population has been created, each string is *evaluated* and assigned a *fitness* score. As defined in the canonical genetic algorithm, *fitness* is $f_i/\bar{f}$, where $\bar{f}$ is the average evaluation of all strings in the current population and f_i is the evaluation of the given i_{th} string (Whitley, 1994). The fitness value of a

given string can also be assigned in other ways, such as tournament selection or simply the string's rank in the current population (see Section 2.3.2 for more details). Fitness is thus a measure of how useful a certain individual is, compared to the rest of the current population. *Evaluation*, on the other hand, is the process of calculating how useful a certain individual is on a whole. That is to say, evaluation provides a measure of performance with respect to the problem at hand, such as how many tagging errors might be corrected by implementing a certain rule.

When representing each individual as a binary string, the initial population can easily be constructed by simply generating the predefined amount of individuals, with each bit in every individual's bit string set at random. This population then assumes the place of the current population. Next, the fitness of each string in the current population is calculated. Selection is carried out based on each individual's fitness value. In the canonical genetic algorithm, the probability that strings in the current population are copied and placed into the intermediate generation is proportional to their fitness. There are, however, many variations on how to perform selection, some of which are detailed in Section 2.3.2.

After selection has been carried out, the next generation can be constructed. This is done by applying crossover operations to pairs of individuals from the intermediate population. The simplest form of crossover is called *one-point crossover*. This process consists of selecting a single crossover-point for each pair of individuals. Each pair's offspring is a result of combining the information *before* the crossover-point from one individual, with the information *after* the crossover-point from the other, and vice versa. After this recombination, mutation can be applied. This is normally done with a very low probability, typically around 1% per individual, and normally consists of flipping a random bit of an individual (Sivanandam and Deepa, 2008; Whitley, 1994).

After the process of selection, recombination and mutation is complete, the next population can be evaluated. The process of evaluation, selection, recombination and mutation forms one *generation* in the execution of a genetic algorithm.

2.3.2 Variations on Selection

There are numerous ways in which selection can be applied, referred to as *selection schemes*. The three most commonly used selection schemes are: *ranking selection*, *proportionate reproduction* and *tournament selection* (Goldberg and Deb, 1991). Miller and Goldberg (1995) list some criteria for an ideal selection scheme, including that it should be simple to implement, efficient on both parallel and non-parallel architectures, as well as that the *selection pressure* applied from a scheme should be easily adjustable. Selection pressure denotes the degree to which more fit individuals in a population are favoured. It is this force which allows GAs to improve population fitness by producing further generations. It is also this force which is largely responsible for the convergence rate[1] of a GA, as higher selection pressure will lead to higher convergence rates.

Perhaps the simplest of the selection schemes listed above is *ranking selection*. This scheme consists of ordering each individual in the current population by fitness and picking the n best individuals (see e.g. Baker (1985); Whitley (1989)). This simplicity does, however, come with a severe drawback. Even though such rank-based methods do guarantee that the most fit individuals in each generation will survive and generate offspring for the next generation, they do poorly in terms of supplying each new generation with the variation provided by including weaker individuals (Whitley, 1994). In other words, purely rank-based methods are prone to becoming stuck in local optima.

A strictly better way of applying selection, called *stochastic universal sampling*, is outlined by Whitley (1994). This scheme falls into the category of *proportionate reproduction* as listed above. In short, it consists of *simultaneously* picking N individuals from the population with a probability proportional to their fitness values in an unbiased manner (Baker, 1985). For instance,

[1]The *convergence rate* is essentially the time it takes for a GA to reach an acceptable solution from which no further significant improvements can be made.

given two individuals a and b with fitness 0.3 and 0.6 respectively, individual b is twice as likely to be selected using this scheme. Although proportionate reproduction is generally quite successful as a selection scheme, it suffers from being significantly slower than the other schemes discussed in this section (Goldberg and Deb, 1991).

Tournament selection is a highly popular selection scheme for GAs, largely due to its simplicity, adaptability for parallel architectures as well as the simplicity with which the selection pressure can be manipulated (see e.g. Goldberg (1989); Goldberg and Deb (1991); Sivanandam and Deepa (2008); Whitley (1989)) – a perfect match for the criteria outlined by Miller and Goldberg (1995). Tournament selection essentially entails holding tournaments each consisting of n competitors. Within each tournament, the winner is the individual with the highest fitness of all the n competitors. A total of $\frac{N}{n}$ tournaments are held, with N being the total amount of individuals. The winners are then inserted into the intermediate generation, prior to applying crossover operations and creating the next generation. Since this intermediate generation consists of tournament winners, it will have a higher average fitness than the current population's average fitness. The resulting increased selection pressure drives the GA to improve average fitness over each generation. This selection pressure can easily be manipulated by altering the tournament size n. Increasing the tournament size will lead to higher selection pressure, since the winner from a larger tournament generally will have a higher fitness value than the winner of a smaller tournament (Goldberg and Deb, 1991).

2.3.3 Variations on Crossover Operations

As with selection schemes, there are some different ways in which crossover operations can be applied. The simplest of which is called *one-point crossover*. Consider the two following individuals:

1 (a) $\langle 0,1,0,0,0,1,1,1\rangle$
1 (b) $\langle x,y,x,y,y,y,x,x\rangle$

Applying one-point crossover to the individuals 1 (a) and 1 (b) consists of two steps. First, a point is randomly selected at which each individual will be split into two. Then the left hand side of one individual is joined with the right hand side of the other individual, and vice versa. In the following example, the individuals have been split along the middle, resulting in the two individuals 1 (c) and 1(d).

1 (c) $\langle 0,1,0,0,y,y,x,x\rangle$
1 (d) $\langle x,y,x,y,0,1,1,1\rangle$

Two-point crossover is an alternative method outlined in Whitley (1994). Analogously to one-point crossover, two-point crossover consists of first selecting two crossover points at random which determine the points at which two individuals will be split and rejoined.

These two types of crossover can be considered as forming a ring (DeJong, 1975), where the first and last bits of an individual are adjacent. In this perspective, one-point crossover is simply a possible outcome of two-point crossover, where one of the crossover points lies between the first and last bits of an individual (see Figure 12 for an illustration).

Although a detailed description of *the schema theorem* (Holland, 1975) is beyond the scope of this thesis, it should be noted that the nature of one-point crossover leads to the position of the bits in each individual being highly important in determining whether or not those bits will remain together after applying crossover. That is to say, adjacent bits are much less likely to be separated by one-point crossover ($p = \frac{1}{L}$ where L is the length of the string) than bits at the end-points of the string ($p = 1$).

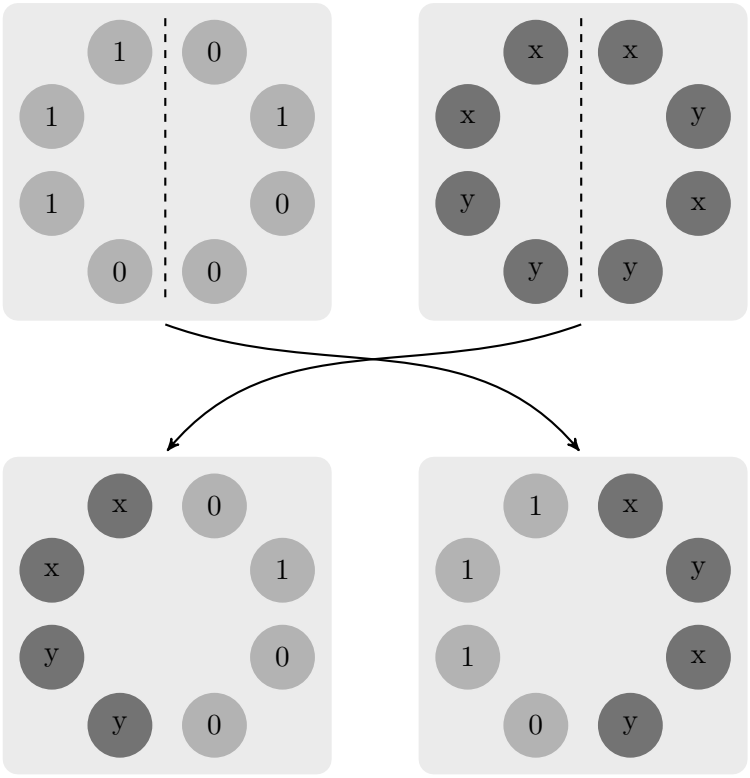

Figure 12: Two-point crossover equivalent to the one-point crossover of Example 1. The dashed lines indicate the points at which the individuals are split. The arrows represent the crossover operation.

2.3.4 Elitism

Elitism is a possible feature to include in a GA. It essentially entails leaving the best individual(s) of a population unchanged, so as not to waste potentially useful solutions to the problem at hand. In other words, the best individuals are allowed to pass along their traits directly to the next generation. This is useful since certain individuals might contain information which is more crucial in order to solve the given problem than others (Reed et al., 2001). If this information is thrown away during crossover, they might not have the chance to reappear except for through mutation, thus lowering the success of the GA as a whole. Furthermore, including elitism provides a way of improving the performance of a GA through increasing the convergence rate (Reed et al., 2001). Elitism might be particularly useful within transformation-based learning, considering that loss of one or more successful transformation rules might be severely detrimental to the performance of the system.

2.4 Parallel Genetic Algorithms

Natural populations are inherently parallel, with several millions of individuals cohabiting and periodically exchanging genetic material. This inherent parallelism provides part of the motivation behind using genetic algorithms as a optimisation tool (Whitley, 1994). In this section, two manners in which this inherent parallelism can be exploited will be presented. First, a simple parallel GA using global populations is outlined. Next, an *Island Model* in which separate subpopulations exist in different threads is presented.

2.4.1 Parallel Genetic Algorithms I - Global Populations

A GA can be implemented in parallel without having to stray far from the canonical GA (see Section 2.3.1). By using tournament selection (see Section 2.3.2), the addition of parallelism becomes quite trivial. If we have a population size N which is evenly divisible with the tournament size n, we can easily distribute these tournaments over s threads where $s = \frac{N}{n}$. Each thread hosts independent tournaments by randomly sampling individuals from the current population, and keeps track of the winners of its tournaments. With the resulting winners residing within each thread, crossover and evaluation can now occur in parallel before this process is repeated. This process is illustrated in Figure 13.

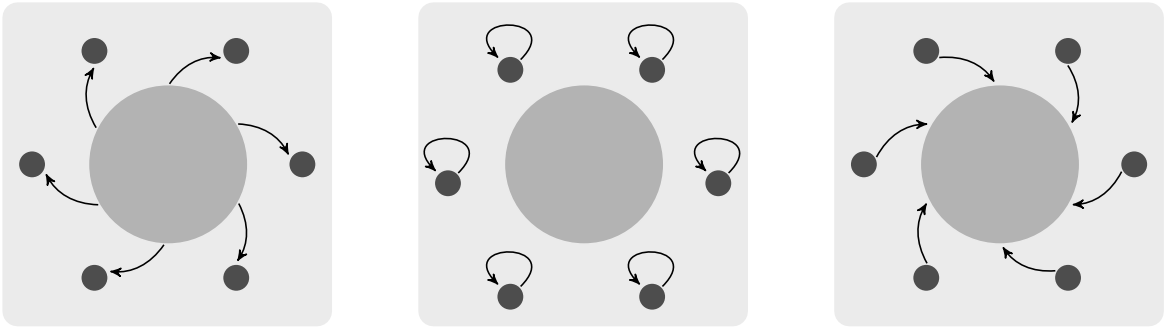

Figure 13: An example of a GA using Global Populations. The large grey circle represents the Global Population of individuals. The small surrounding circles represent threads.

In the leftmost segment of Figure 13, each thread randomly samples n individuals from the global population where n is the tournament size. In the middle segment, fitness evaluation and crossover operations are performed, as well as the actual tournament selection procedure – all in parallel. In the final segment, the resulting individuals are resubmitted to the global population, before this procedure is repeated.

2.4.2 Parallel Genetic Algorithms II - Island Models

Using an *Island Model* as a tool for parallelising a GA allows for dividing up the search into larger units. This is particularly advantageous if we wish to use a limited amount of threads due to, for instance, a limited amount of available processor cores. For example, we might be limited to 24 cores, whereas our population might consist of 2400 strings. We could divide this total population into subpopulations of 100 strings each, allowing each of these subpopulations to execute as a GA. Every few generations, the subpopulations exchange a few strings with each other as genetic material is exchanged between remote populations. This *migration* is what lets subpopulations share genetic material, thus providing them with access to more genetic diversity (Gorges-Schleuter, 1991; Starkweather et al., 1991; Tanese, 1989; Whitley and Starkweather, 1990).

Figure 14 depicts an example in which we have 6 subpopulations, each residing on a separate island. Each island will execute a separate GA, starting with its own randomised population. The genetic drift[1] and sampling error caused by this initial randomisation in each population means that at any given time, we will have 6 slightly different populations residing on each island. When *migration* is introduced (as depicted by the arrows), the Island Model allows for the exploitation of these differences. That is to say, the variation caused by the differing initial populations as well as genetic drift, creates a source of genetic diversity. Each island containing its own subpopulation has access to this diversity through periodic migration.

[1]Genetic drift denotes the alteration of genetic material and fitness between different populations across generations.

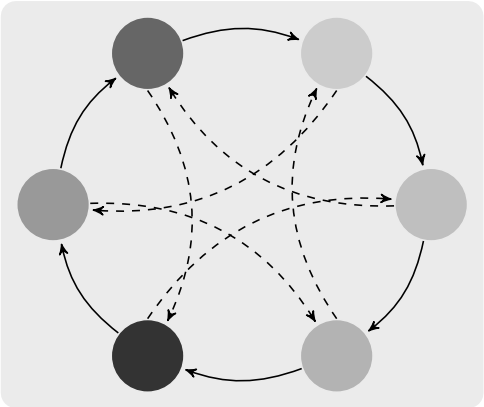

Figure 14: An example of an Island Model Genetic Algorithm. The colouring of the islands represent the similarity of the genetic material as the islands exchange genetic material. The arrows represent migration, with dashed arrows representing long-range migration.

2.5 Previous Improvements on the Brill tagger

Numerous previous modifications and improvements have been made for the Brill tagger. These include general improvements on the tagger's accuracy, time consumption as well as modifications to make the tagger more accurate for specific languages. This section presents an overview of the modifications most influential to this paper.

2.5.1 Improving Training Times in TBL-based Systems

Ramshaw and Marcus (1994) present an approach which greatly reduces the training time of the Brill algorithm, by making the time-consuming updating step more efficient. Since this update step is normally applied for each new rule, a large proportion of the training time is spent in this phase. The method presented in their paper involves having each rule store pointers to the samples in the corpus to which they apply. These samples in turn, have pointers to the rules which apply to them. When the system has access to these pointer lists, the update process can be performed quite efficiently. This is owed to the fact that the system can identify positions where the given rule would be applied and update the scores of the rules which are affected by the changes caused by this application (Ramshaw and Marcus, 1994).

Ngai and Florian (2001) present an approach which builds on the one presented by Ramshaw and Marcus (1994). When investigating a rule b which is applied on the corpus S, the goal is to identify the set of rules r which are affected by the application of this rule, in either of the rule's score sets $G(r)$ or $B(r)$. $G(r)$ denotes the samples on which the given rule applies and changes them to the correct classification, and $B(r)$ the samples on which the given rule applies and changes them to an incorrect classification. This identification process is complicated by the fact that the tags assigned to words are not independent in PoS tagging (Ngai and Florian, 2001), as the tag assigned to a word depends on the tags of the preceding and succeeding tags.

Making the assumption that samples in fact are independent only marginally affects actual tagging accuracy, as shown in n implementation by Hepple (2000) in what is referred to as an *Independence and Commitment* (IC) system. In an IC system, rules are assumed to be independent of each other, which has the advantage that the scores of rules do not need to be evaluated more than once. Additionally, once a rule has assigned a PoS tag to a word, this tag is committed, so that the tag can no longer be changed. This approach performs significantly faster than the implementation presented by Ngai and Florian (2001), at the cost of marginally worse performance. The lower accuracy achieved with this system can be explained by that the tagger is unable to learn new rules to correct the mistakes made by previous rules.

2.5.2 Adaptation to Specific Target Languages

Previous research has been successful in adapting the tagger to other languages, such as Polish (Acedański, 2010; Acedański and Gołuchowski, 2009), Hungarian (Megyesi, 1999), Arabic (Freeman, 2001) and Swedish (Prütz, 2002). On a general level, these improvements consist of manually altering the templates used in the tagger in order to match properties of the language at hand.

Acedański (2010) added the feature of *generalised transformation templates*, which allows for rules which only stipulate the change of a part of a complex morphosyntactically annotated tag (e.g. changing from *singular* to *plural*, rather than changing the entire tag from *NN* to *JJ*). More complex *lexical transformation templates* allowing the tagger to match prefixes and suffixes in order to determine the appropriate tag have also been found to be successful (Acedański, 2010; Megyesi, 1999). These additions were found to substantially improve the performance of the Brill tagger (Acedański, 2010; Acedański and Gołuchowski, 2009).

Although previous work has been successful in improving the performance of the Brill tagger for specific target languages, there is to the best of our knowledge no research which has attempted to improve the tagger's multi-lingual applicability in a more general manner.

2.5.3 Searching for Rules with Genetic Algorithms

A paper by Wilson and Heywood (2005) deals with an attempt at implementing the Brill tagger using GAs. Their implementation essentially entails evolving individuals consisting of nearly 400 rules, from which they randomly select 4 rules and calculate scores on a randomly selected portion of their test material. This approach turns out to not be particularly fruitful, as the tagger achieves an average accuracy of 89.8% for English text, while the original Brill tagger achieved an accuracy of 94.9% for English text. This can be explained by the fact that the heuristic approach used in a GA, in which several candidate individuals are automatically generated and improved upon over the course of several generations, does not seem to make much sense within the framework of transformation rules. For instance, a successful rule for English might alter *NN* to *AB* if the previous word was *to*, while another might alter *NN* to *VB* if the next tag is *AB*. Performing crossover on this would simply entail a composition of the two rules, resulting in a rule altering *NN* to *VB* if the previous word was *to* and the next tag is *AB*. Although this more complex rule does appear to make sense, it can be considered to be too specific, thus losing out on many potential corrections from the first more general rule. Although the approach used by Wilson and Heywood (2005) was not successful, this paper employs GAs in an entirely different way.

2.6 Aims of this work

This thesis aims to provide answers to the following questions.

1. Can the Brill tagger be made more language independent by defining rule templates automatically by using Genetic Algorithms?

2. Bender (2009) remarks that many studies make uncorroborated claims of language independence. To what extent can the results obtained here be generalised for other languages?

3. Does the assumption of rule independence made by Hepple (2000) extend to the complex and general rules used in the Brill GA-tagger?

3 Data

The Brill tagger and the Brill GA-tagger were developed and evaluated using 9 languages: Chinese, Japanese, Turkish, Slovene, Portuguese, English, Dutch, Swedish and Icelandic. These languages were chosen as they represent languages with varying degrees of morphological complexity and structure. Table 5 contains details of the data used for each language and the size of the tag sets used. Segmented corpora were used for training and evaluation for languages where words are not delimited by whitespace. For each corpus, the number of PoS tags listed includes all PoS tags as well as all combinations of morphological features that are used. Note that although some of the corpora are parsed, the only information used is the PoS and morphological tags for each word, as well as the word itself.

Table 5: Corpus overview

Language	**Tokens**	**Types**	**Tags**
Chinese	$\sim 10^6$	$\sim 6 \times 10^4$	38
Japanese	$\sim 2 \times 10^5$	$\sim 3 \times 10^3$	91
Turkish	$\sim 5 \times 10^4$	$\sim 2 \times 10^4$	997
Slovene	$\sim 3 \times 10^4$	$\sim 7 \times 10^3$	728
Portuguese	$\sim 2 \times 10^5$	$\sim 3 \times 10^4$	874
English	$\sim 10^6$	$\sim 5 \times 10^4$	45
Dutch	$\sim 2 \times 10^5$	$\sim 3 \times 10^4$	732
Swedish	$\sim 10^6$	$\sim 10^5$	153
Icelandic	$\sim 10^6$	$\sim 7 \times 10^4$	492

3.1 Chinese

Version 7.0 of the Chinese Treebank[1] is used in this study. It consists of roughly 1.2 million words, and is annotated using a tag set consisting of 38 tags, similar to those used in the Penn Treebank (Xue et al., 2005).

3.2 Japanese

The Tübingen Treebank of Spoken Japanese[2] is used for the Japanese tagging in this study. It contains approximately 160,000 tokens of spontaneous speech from native speakers of Japanese (Kawata and Bartels, 2000). It is manually annotated, and its tag set contains 91 tags.

3.3 Turkish

The METU-sabanci Turkish Treebank[3] is used for the Turkish tagging in this study. It is a semi-manually morphologically and syntactically annotated treebank corpus (Oflazer et al., 2003). It consists of approximately 50,000 tokens, and its tag set includes 997 tags.

[1] Chinese Treebank: http://www.cis.upenn.edu/~chinese/ctb.html
[2] Tübingen Treebank: http://www.sfs.uni-tuebingen.de/en/ascl/resources/corpora/tueba-js.html
[3] METU-Sabanci: http://ii2.metu.edu.tr/content/treebank

3.4 Slovene

The Slovene Dependency Treebank[1] is used in this study. It is a small syntactically annotated corpus, consisting of approximately 30,000 words gathered from the Slovene translation of Orwell's novel *1984* (Džeroski et al., 2006). Its tag set consists of 728 tags.

3.5 Portuguese

The Floresta sintá(c)tica treebank[2] is used for the Portuguese tagging in this study. It consists of approximately 200,000 tokens gathered from the European Portuguese newspaper Público (Afonso et al., 2002). Its tag set contains 874 tags.

3.6 English

The Wall Street Journal section of the Penn Treebank[3] is used in this study. It consists of roughly 1.2 million words, and is annotated manually using a tag set consisting of 45 tags (Marcus et al., 1993).

3.7 Dutch

The Alpino Treebank[4] is used for the Dutch tagging in this study. It is a syntactically annotated corpus consisting of approximately 150,000 words gathered from a section of the Eindhoven corpus consisting of newspapers (Van der Beek et al., 2002). The tag set of the corpus consists of 732 tags.

3.8 Swedish

Version 3.0 of the Stockholm Umeå Corpus[5] is used for the Swedish tagging in this study. It is a balanced and manually annotated corpus consisting of roughly 1.2 million words. The SUC tag set consists of 22 PoS tags (Källgren, 2006), which in combination with the morphosyntactic features yields a total of 153 unique tags.

3.9 Icelandic

The Icelandic Parsed Historical Corpus[6] is used for the Icelandic tagging in this study. It is a historical corpus consisting of roughly 1.1 million words from mainly narrative and religious texts (Rögnvaldsson et al., 2011), distributed over periods of time ranging from the 12th to the 21st century. Its tag set consists of 492 tags.

[1]Slovene Dependency Treebank: http://nl.ijs.si/sdt/
[2]Floresta sintá(c)tica treebank: http://www.linguateca.pt/floresta/info_floresta_English.html
[3]Penn Treebank: http://www.cis.upenn.edu/~treebank/
[4]Alpino Treebank: http://www.let.rug.nl/vannoord/trees/
[5]Stockholm Umeå Corpus: http://spraakbanken.gu.se/eng/resources/suc
[6]Icelandic Parsed Historical Corpus: http://www.linguist.is/icelandic_treebank

4 Implementation

The Brill tagger implemented in this paper borrows traits from several other papers. First and foremost, it is based on the original implementation by Brill (1992). This implementation lays the foundation for the evaluations presented in this paper, where it serves as a baseline. Further improvements are implemented based on traits borrowed from Ngai and Florian (2001), Hepple (2000) and Roche and Schabes (1995). This also serves as a foundation for the Brill GA-tagger. The rule templates used by the Brill GA-tagger are not defined manually, but derived automatically through the use of a Genetic Algorithm, in an attempt to make the tagger less language specific. The GA implementation used here mostly follows the canonical genetic algorithm. Note that no improvements were made on the initial tagging performed by either tagger, as the object of interest is not tagging accuracy *per se*, but rather accuracy *improvement* as a result of using GAs. That is to say, the initial tagging consists of labelling each word with the Part-of-Speech it is most frequently labelled with in the training material for the given language. Considering that rules that correct a very low amount of errors are prone to being a result of overtraining, a cut-off frequency of $5 \times 10^{-6}\%$ (5 corrections per 1 million words) was used. That is to say, any rules correcting fewer errors than this were not considered.

4.1 Improvements on Training Time

The training time of systems based on transformation-based learning can in general be described as being sub-optimal. The Brill tagger is no exception to this portrayal. Consequently, part of the methodology employed here revolves around improving upon this factor.

The training time was improved partially by implementing the rule evaluation procedure in parallel. Seeing as no data dependencies exist during the evaluation of the set of rules which match a template, this implementation is fairly trivial. In this implementation, the parallelisation simply consists of distributing the templates evenly over each available processor core. Each core can then handle the evaluation of the subset of rules which it is allocated. Provided that each template takes an equal amount of time to evaluate, this should result in a nearly linear speed-up for each additional available core.

Additionally, Ngai and Florian (2001) suggest a method in which the frequent redundancy when evaluating rules is addressed. In the Brill tagger's original implementation, each iteration of the rule learning process goes through each potential new rule in order. This leads to a large amount of rules being re-evaluated when their scores have not changed. Consider a rule $\langle tag_x, tag_y, c \rangle$ where tag_x is the tag to change from, tag_y the tag to change to and c the given context in which to apply the rule. It is clear that the only rules which need to be re-evaluated after such a rule has been applied, are the rules in which x or z correspond to the latest rule's x or y tags.

4.2 Improvements on Tagging Time

As previously noted, the tagging process of the Brill tagger is a major concern in terms of time consumption. Due to this, the improvements detailed by Roche and Schabes (1995) were adapted and implemented. That is to say, the rules were represented as a DFT, following the steps detailed in Section 2.2.4. As previously mentioned, however, this implementation does not extend to rules that require more than one feature, so it was not included in the final version of either tagger.

4.3 Using Genetic Algorithms to Search for Rule Templates

In an attempt to improve the accuracy and multi-lingual applicability of the Brill GA-tagger, rule templates were automatically searched for using a search optimised with a GA.

$$\begin{array}{cc} PoS & \begin{bmatrix} 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \\ Mult. & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ Lex. & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad \begin{array}{cc} & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad \begin{array}{cc} & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \end{bmatrix} \end{array}$$

$$\begin{array}{cc} PoS & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ Mult. & \begin{bmatrix} 0 & \mathbf{1} & 0 & 0 \end{bmatrix} \\ Lex. & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad \begin{array}{cc} & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \end{bmatrix} \end{array} \quad \begin{array}{cc} & \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

(a) PoS tag (-1) and multiple tag (+2)

(b) Multiple tag (-2) and lexical entity (+3)

(c) Lexical entity (-2) and PoS tag (+1)

Figure 15: Examples of the representation of individuals in the GA

4.3.1 Representation of Individuals

As detailed previously, rules in the Brill tagger can be described as having the form $\langle tag_x, tag_y, c \rangle$, where tag_x is the tag to change from, tag_y is the tag to change to and c is the context surrounding tag_x. Out of these parameters, c is by far the most complex as it can contain $k^{|c|}$ different values, where k is the number of PoS tags in the tag set and words in the lexicon. Seeing as tag_x and tag_y are both mandatory features in any rule and are gathered from a pre-compiled list of error triplets, the definition of a template is solely contingent on c. In this study, templates encode the obligatory presence of a certain PoS tag, the obligatory presence of one out of multiple PoS tags as well as the obligatory presence of a lexical entry.

The obligatory presence of one out of multiple tags constitutes a novel addition to the Brill tagger, which to the best of our knowledge has not been used previously. This should allow the Brill GA-tagger to generate more general rules than the Brill tagger. More importantly, this should allow more specific rules to achieve high enough scores to be used, by incorporating an element of generality. For instance, for some languages one may want to change a noun to a verb if the preceding tag is either an adverb or an infinitive marker, and the next word is noun. This feature will help such rules being learned, when either of the rules of which it consists might not receive high enough scores in isolation.

Individuals are implemented as two $n \times m$ bit matrices, where n is defined as above and m is the desired maximum context span (here, $m = 4$). These two matrices represent the contexts before and after tag_x. In each matrix, a zero indicates that the tag found in this position is to be specified in rules using this template, whereas a one indicates that the tag or lexeme found in this position is to be ignored and left unspecified. The columns of each matrix denote the feature in which we are interested. Figure 15 depicts examples of such matrices, with Figure 15a showing a template concerned with the PoS tag at position -1 and one out of several tags at position +2, Figure 15b showing a template concerned with one out of several tag at position -2 and the lexical entity at position +3, and Figure 15c showing a template concerned with the lexical entity at position -2 and the PoS tag at position +1. In this implementation we have 2^{24} possible rule templates.

4.3.2 Fitness Calculation

The fitness value of each individual needs to be calculated once per generation. In the Brill GA-tagger, this consists of generating a set $\mathbb{A}$ consisting of all possible c parameters which match the individual's template.

The tag_x and tag_y parameters are obtained from a pre-compiled list of error triplets $\langle tag_x, tag_y, n \rangle$ where tag_x is the occurring tag, tag_y the correct tag, and n the amount of times this error occurs.

These are combined with the c parameters to form full rules. These rules are then evaluated using the training material. Scores are calculated as $r - w$ where r is the amount of tags corrected by the rule, and w is the amount of tags which were made incorrect by the rule. The fitness value of the individual is the mean score of all the rules formed from each $c \in \mathbb{A}$.

4.3.3 Parameter Settings

The selection procedure used in this paper is *tournament selection*, as discussed in Section 2.3.2. Although many different selection schemes exist, tournament selection can often yield better results than e.g. strictly ranked selection (Miller and Goldberg, 1995). In its simplest form, it essentially consists of randomly selecting pairs of individuals from the current population to compete with each other – the winner of the tournament is simply the individual with the higher fitness value (see Section 2.3.2 for more details). This selection scheme was chosen as it both performs well in terms of time consumption and in terms of ensuring that each generation contains a wide variety of genetic material.

Crossover was implemented as one-point crossover, as discussed in Section 2.3.3, by selecting a point at random for each pair of individuals and combining the first part of the first individual's bit vector with the second part of the second individual's bit vector and vice versa. One-point crossover was chosen over two-point crossover partially because of its simplicity and that the differences in performance between different types of crossover do not affect the outcome of GAs to a particularly large extent. Mutation was implemented as the chance of a randomly occurring bit-flip in an individual's bit matrix with a low probability ($p = 0.01$ per individual per generation), which is in line with many other GA implementations.

Elitism consists of allowing an individual to continue on to the next generation without altering its genetic material (see Section 2.3.4 for more details). Other work has applied this factor by allowing very few individuals per generation to be affected by elitism (Reed et al., 2001). This implementation applies elitism at a quite high rate, by always letting the 10 most fit individuals remain unchanged per generation. This was applied due to the fact that the problem of template evolution can be seen as a process which generates two distinct types of templates. On the one hand we have simple templates which to some extent resemble the ones used by Brill (1992). These templates will often achieve very high fitness levels, which can be explained by the fact that they are highly simplistic and thus also quite general, leading to many potential tagging errors which can be corrected through utilising these templates. More specific templates, on the other hand, might only be able to generate a handful of successful rules, leading to fairly low fitness values being assigned to them. It ought to be clear that both types of templates are necessary for the Brill GA-tagger to be successful. Hence, using a high degree of elitism allows the tagger to save the highly fit simple templates while simultaneously evolving more complex templates for more specific tagging errors.

4.4 Parallelisation

The tagger's training procedure was implemented in parallel with two main purposes in mind – training speed and improving the GA. In order to improve upon the training speed, each template evaluation was carried out by dividing the templates evenly over the available cores. Considering that evaluating a template simply consists of generating all rules which fit the template and calculating the scores of these rules across the training material, parallelisation of this process is trivial. In this implementation, each template was simply allocated to an available core to carry out the evaluation. In an attempt to improve the templates through increased genetic diversity, the GA was implemented as an Island Model, as detailed in Section 2.4.2.

5 Evaluation

This section consists of an overview of the experimental setup used to evaluate the taggers. A Most Common Tag baseline is added to more clearly illustrate the difference in difficulty of applying PoS tagging to the different target languages.

5.1 Most Common Tag Baseline

Perhaps the simplest approach to PoS tagging is to assign to each word the PoS tag with which it occurs most frequently in the given training material. This naïve approach constitutes what will be referred to as the *most common tag baseline*. Considering that this is the procedure which the Brill tagger uses in its initial tagging state, this serves as a worst case scenario in terms of tagging accuracy. Furthermore, this baseline could to some extent illustrate how challenging it is to apply PoS tagging to a given language. For instance, a language which has no ambiguity in terms of which PoS should be assigned to a word would constitute a trivial case and have a baseline at 100%. A more challenging language, on the other hand, might have a baseline at approximately 60%.

5.2 Brill Baseline

The Brill tagger (Brill, 1992) was used as a baseline for the experiments. This tagger was evaluated using 10-fold cross validation. The corpus was split into 10 approximately equally sized parts. For each fold k, the tagger was tested on part k and trained on the remaining 9 parts. This procedure was repeated for each language.

5.3 Tagging with Genetic Algorithms

The Brill GA-tagger was implemented using GAs as detailed in Section 4.3 in order to automatically learn the templates used for the tagger. Evaluation was done in the same way as for the Brill tagger, using 10-fold cross validation. As with the Brill tagger, this procedure was repeated for each language.

5.4 Assumed Rule Independence

As noted by Hepple (2000), the cases in which rules interact with each other are quite rare in the Brill tagger. The advantage of assuming complete rule independence is that the training procedure can be carried out much faster than even the implementation by Ngai and Florian (2001), as the score of each rule only needs to be calculated once. In order to investigate the effect of this in the case of the Brill GA-tagger, each evaluation procedure was carried out twice: once with assumed rule independence, and once without.

6 Results

The results from the evaluation are detailed in three sections. The first section lists the tagging error rates obtained from the taggers. The second section contains information about the rules learned by the taggers. The third section includes the results from the implementation of the optimisation techniques by Roche and Schabes (1995), Hepple (2000) and Ngai and Florian (2001).

6.1 Tagging Accuracy

A comparison of the taggers' accuracies **without** assumed rule independence is shown in Table 6. The Brill GA-tagger significantly outperforms the Brill tagger for all languages in this condition (two tailed McNemar's test (see McNemar (1947)), $p < 0.001$). Table 7 contains the corresponding information **with** assumed rule independence. The Brill GA-tagger performs significantly worse than the Brill tagger in this condition (two tailed McNemar's test, $p < 0.001$). The GA implementation using an Island Model resulted in an accuracy equal to the Brill GA-tagger.

Table 6: Tagger comparison **without** assumed rule independence. All numbers denote error rates. Δ denotes the change in error rate from the standard Brill tagger to the GA tagger.

Language	**Most common tag**	**Brill tagger**	**Brill GA-tagger**	Δ[1]
Chinese	15.9%	13.8%	12.8%	7.3%
Japanese	7.6%	7.1%	6.9%	2.8%
Turkish	43.9%	42.3%	39.9%	5.7%
Slovene	28.4%	27.0%	26.1%	3.3%
Portuguese	32.6%	29.8%	29.0%	2.7%
English	8.8%	6.6%	6.4%	2.6%
Dutch	18.6%	15.9%	15.4%	3.0%
Swedish	17.3%	16.0%	15.5%	3.1%
Icelandic	23.3%	21.5%	18.2%	15.3%

Table 7: Tagger comparison **with** assumed rule independence.

Language	**Most common tag**	**Brill tagger**	**Brill GA-tagger**	Δ[1]
Chinese	15.9%	15.1%	15.3%	−1.3%
Japanese	7.6%	7.2%	7.5%	−4.2%
Turkish	43.9%	42.5%	43.0%	−1.2%
Slovene	28.4%	28.1%	28.3%	−0.7%
Portuguese	32.6%	31.2%	31.4%	−0.6%
English	8.8%	7.9%	8.1%	−2.5%
Dutch	18.6%	17.3%	17.4%	−0.6%
Swedish	17.3%	16.4%	16.6%	−1.2%
Icelandic	23.3%	22.8%	23.1%	−1.3%

[1] All changes in error rate are significant (two tailed McNemar's test, $p < 0.001$)

6.2 Transformation Rules

Figure 16 illustrates the FST representation of the two English rules learned by the Brill GA-tagger which are shown in Table 8. Examples of some of the templates learned by the GA are shown in Appendix 2.

The Brill tagger first learns the rules which are the most efficient for the given problem (see Section 2.1.2 for more details). In other words, the first rules learned are the ones that correct the largest amount of errors, whereas rules learned later on typically only correct a handful of errors. Figure 17 contains examples from Chinese, Japanese, Turkish and Icelandic, showing that the Brill GA-tagger is able to learn rules which outperform the rules learned by the standard Brill tagger when rule independence is not assumed. For each language, the amount of rules corrected tapers off relatively quickly.

The most efficient rules for each language learned by the Brill tagger and Brill GA-tagger respectively are shown in Table 8. For each language, the Brill GA-tagger is able to learn more general rules at first, which score higher than the more or less equivalent rules learned by the Brill tagger.

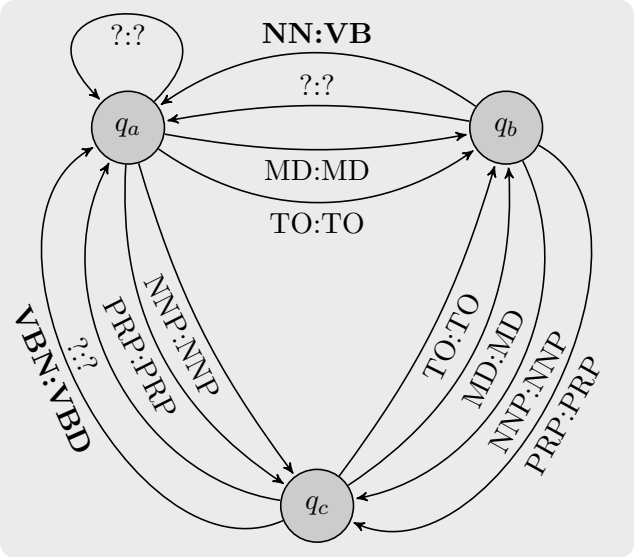

Figure 16: Two English rules represented as FSTs. Transitions which change a PoS tag are **bold**.

6.3 Implementation Efficiency

The tagging procedure was sped up by implementing the rules as a DFT. This implementation sped up the tagging process by a factor of 20. However, this implementation was not included in the evaluated taggers, considering that the procedure used to represent rules as a DFT does not extend to rules where more than one feature is necessary. A suggestion for how this might be done is presented in Section 7.6.1.

The training procedure was sped up in two different ways. First, parts of the implementation by Ngai and Florian (2001) were tested, which yielded a speed-up of approximately a factor of 10. Next, the improvements by Hepple (2000) were tested, yielding a speed-up of a factor of approximately 300.

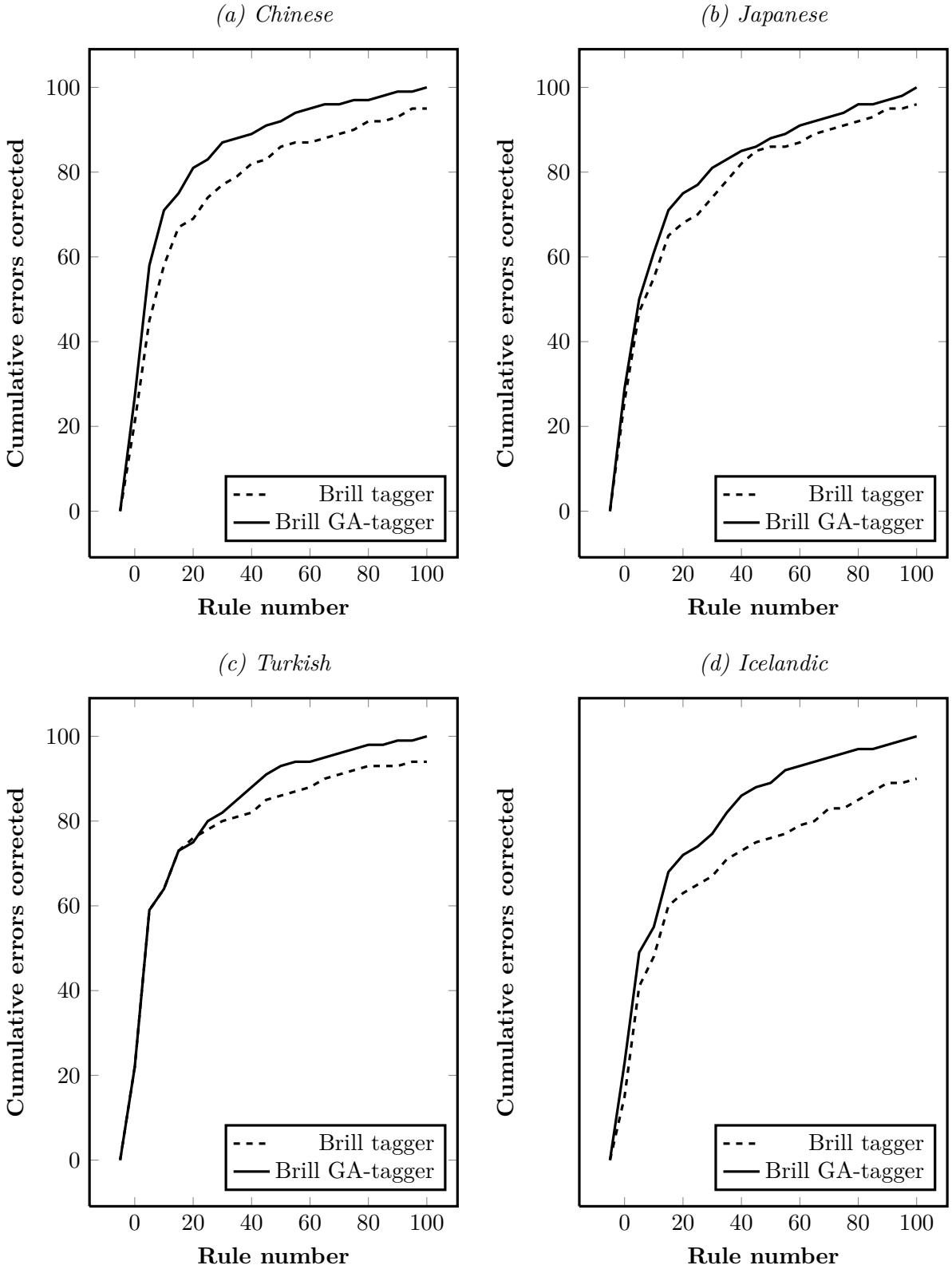

Figure 17: Cumulative errors corrected for the first 100 rules from Chinese (a), Japanese (b), Turkish (c) and Icelandic (d) when not assuming rule independence. Dashed lines represent the Brill tagger and whole lines represent the Brill GA-tagger. The x-axis depicts each rule and the y-axis represents the cumulative errors corrected for each language. The y-axis is normalised to the sum of all error corrections by the Brill GA-tagger.

Table 8: Rules learned by the taggers in different conditions. Italic text indicates lexical entities. Items within curly brackets indicate optional tags. The notation used for each PoS tag is taken from the corpus to which the rule applies.

Language	**Tagger**	**From**	**To**	**Left ctxt**	**Right ctxt**
Chinese	GA	VV	NN	–, –, {DEG,VE}	–
	GA	NN	VV	–, –, {DEV,AD}	–
	Brill	NN	VV	–, –, AD	–
	Brill	DT	PN	–	VC, –, –
Japanese	GA	Vfin	VSfin	–, –, {VN,Pfoc}	–
	GA	N	V	–	P, –, –
	Brill	Vfin	VSfin	–, –, VN	–
	Brill	Vfin	VSfin	–, –, Pfoc	–
Turkish	GA	Adj	Adv	–, –, {N sg., N pl}	–
	GA	Adj	Adv	–	Card, –, –
	Brill	Adj	Adv	–, –, Det	–
	Brill	Adj	Adv	–	Card, –, –
Slovene	GA	Adp. Acc.	Adp. Loc.	–	Noun Loc., –, –
	GA	Adp. Loc.	Adp. Acc.	–	Noun Acc., –, –
	Brill	Conj.	Part.	–, –, Conj.	–
	Brill	Adp. Acc.	Adp. Loc.	–	Noun Loc., –, –
Portuguese	GA	A Mas	A Fem	–, –, N Fem	–
	GA	prp sam	prp	–	{N Fem, N Mas}, –, –
	Brill	A Mas	A Fem	–, –, N Fem	–
	Brill	prp sam	prp	–	N Fem, –, –
English	GA	NN	VB	–, –, {TO, MD}	–
	GA	VBN	VBD	–, –, {NNP PRP}	–
	Brill	NN	VB	–, –, *to*	–
	Brill	VBP	VB	–, –, MD	–
Dutch	GA	Adj. adv	Adj. attr	–, –, Art.	N, –, –
	GA	Pron. z	Pron. a	–	N, –, –
	Brill	Adj. adv	Adj. attr	–	N, –, –
	Brill	Pron. z	Pron. a	–	N, –, –
Swedish	GA	JJ PLU	JJ SIN	–, –, JJ SIN	NN SIN, –, –
	GA	JJ SIN	JJ PLU	–, –, JJ PLU	NN PLU, –, –
	Brill	JJ PLU	JJ SIN	–	NN SIN, –, –
	Brill	JJ SIN	JJ PLU	–	NN PLU, –, –
Icelandic	GA	D-A	D-N	–, –, {NS-N, N-N}	–
	GA	N-N	N-A	–, –, {VB, NPR-A}	–
	Brill	D-A	D-N	–, –, N-N	–
	Brill	N-N	N-A	–, –, NPR-A	–

7 Discussion

This section begins with a discussion of the data sets used and the challenges posed by some of the corpora. Following this, the implementation method is discussed, after which a brief overview of the validity of this study is given. The results obtained in this work are then examined more carefully. After this discussion, a novel way of implementing feature-rich rules as an FSTs is suggested. Finally, an overview of the relevance of this thesis as well as some suggestions for future work is presented.

7.1 Discussion of Data

As remarked by Bender (2009), the majority of NLP studies only take the trouble to evaluate their work on one or two languages, with less than 7% of papers accepted to the ACL 2008[1] conference considering 3 or more languages. In spite of this, a large proportion of the papers using one or two languages claimed that their approach would be cross-lingually applicable. This constitutes an interesting issue, considering that approaches only tested on one or two languages are liable to contain elements which are tailored for these languages. Adding in the fact that over 71% of the papers accepted to ACL 2008 solely dealt with Germanic languages further emphasises the one-sidedness of this common procedure. The taggers implemented in this study were evaluated on several languages from different language families in order to avoid this type of bias. Furthermore, the languages were selected so as to include languages with varying degrees of morphological complexity and structure. This allows the results found here to be generalised further than what would have been the case if only one or a few languages were used. In the case of the Brill tagger, this is particularly interesting as previous studies have found the Brill tagger to be somewhat lacking in performance for languages with more morphological complexity than English (Acedański, 2010; Acedański and Gołuchowski, 2009; Megyesi, 1999).

Although it would have been interesting to test the taggers on Hungarian and Polish, so as to better compare the results obtained here with Megyesi (1999) and Acedański and Gołuchowski (2009), it was not possible within the scope of this study to obtain access to PoS tagged corpora for either of these languages. Even so, the languages used here represent a fairly large and varied sample (at the very least from an NLP perspective). It should therefore be fair to assume that the results obtained here can be replicated for a large variety of languages, and at the very least for languages similar to the ones used in this work.

The corpora used in this work consist mostly of similar types of data, gathered from various types of printed works as detailed in Section 3. This reflects what can be considered as common practice in the field of NLP, where the vast majority of studies are evaluated on well-established corpora such as the Penn Treebank. Two of the corpora used differ from this norm, however, as they consist of historical texts (Icelandic) and spoken language (Japanese).

7.1.1 Historical Corpora

The Icelandic Parsed Historical Corpus (IcePaHc) contains texts dating from the 12th century to the early 21st century. Evaluating PoS taggers on this corpus is particularly interesting, as the texts represent centuries of grammatical change. It could thus be considered to be a particularly challenging case for PoS tagging. Positive results when tagging this type of material might be some indication of the robustness of a PoS tagger. This is especially true if the tagger is always trained and tested on different periods of time, as is the case in this study.

7.1.2 Corpora of Spoken Language

The Tübingen Treebank of Spoken Japanese also represents a special case in this study. Spoken language can be considered as a particularly challenging case for PoS tagging, partially due to

[1] ACL 2008: The 46th Annual meeting of the Association for Computational Linguistics

factors such as disfluencies which may alter the local context on which most PoS taggers rely, the Brill tagger included (see Heeman and Allen (1999) for more details on this subject). In cases where disfluencies are tagged, this may end up being an advantage, however, as filled pauses might occur in quite systematic contexts and act as, e.g., sentence delimiters. However, disfluencies such as false starts and repairs are most likely not be particularly helpful. Including a corpus containing spoken language in this study could be considered a further strength in terms of the variation in the data sets employed.

7.1.3 Small Corpora

Some of the corpora used in this study were relatively small, namely the Slovene, Dutch, Portuguese, Japanese and Turkish corpora, consisting of fewer than 10^5 tokens each. The case of using small corpora poses an interesting challenge, considering that a small corpus is highly likely to not contain a sufficient amount of generalisable tagging errors. In other words, training on a small corpus poses a high risk in terms of overfitting, even for the Brill tagger. For instance, a fairly successful rule learned by the Brill tagger for English is to change *IN* to *RB* if the next word is tagged *JJ* and the word after that is *as*. Intuitively, this might seem like an odd rule. However, this is exactly the type of rule that is necessary to correct mistakes in the tagging of a fairly frequent idiom, namely *as good as*. The most common tag for *as* in the Penn Treebank is *IN*, which is the correct tag for the second *as*. The first *as*, however, should be tagged as *RB*. After the initial tagging both would be tagged as *IN*. Using a somewhat complex template, however, the tagger might learn to change *IN* to *RB* if the following word was tagged *JJ* and the next word after that is *as*. The tagger would thus have learned to successfully tag this phrase. A small corpus might only contain a handful of occurrences of this or other idioms, thus leading to this quite efficient rule receiving a low score and potentially not being learned at all. Furthermore, a small and unbalanced corpus such as the Slovene corpus, consisting of parts of an Orwell novel, is likely to contain very specific cases which do not generalise well to different styles or genres. If, for instance, Orwell consistently finished most sentences with a noun, a somewhat odd rule changing *VB* to *NN* if the next word is a full stop might receive a high enough score to be learned, thus tagging the final word in *"I want to fish."* as *NN*.

Using small corpora is a quite relevant issue. Many NLP tasks, such as PoS tagging and syntactic parsing, can be carried out with a high degree of success for languages for which large corpora are available. For under-resourced languages, however, state of the art methods which often rely heavily on these large data sets are not particularly successful. Neither the Brill tagger nor the Brill GA-tagger achieve high accuracies for these small corpora. This is caused in part by the corpus size itself, but can also be explained by the fact that the languages which have small corpora in this study are relatively morphologically complex. However, the nature of rule-based tagging allows for more transparency than statistical PoS taggers do. It may thus be more worthwhile to use the Brill GA-tagger for languages for which resources are sparse, seeing as the rules generated by the tagger can be examined with far more ease than, e.g., massive matrices containing transmission and emission probabilities. It is not entirely unthinkable that examining rules in this manner might lead to some sort of increased understanding of the language in question, or at the very least give the investigator clues as to what is or is not working.

7.2 Discussion of Implementation

The implementation of the taggers used in this paper relies mainly on three areas. First and foremost, the tagger outlined by Brill (1992) served as a baseline. Secondly it relies on the optimisations to the tagger's training procedure by Ngai and Florian (2001) and Hepple (2000). Finally, the Brill GA-tagger includes the novel addition of searching for rule templates automatically using a search optimised by a GA.

7.2.1 Application of Genetic Algorithms

The automatic search for rule templates may not have been the most optimal area to apply the search optimisation. Even with the flexible templates used in the Brill GA-tagger, the total amount of templates is merely 2^{24}, while the potential rules for a simple template is closer to 492^{24} in the case of Icelandic where the tag set contains 492 tags (in reality this number is even higher, as the number of tokens which could be used in lexical tags is much higher than the number of PoS tags). It might then be considered as more useful to optimise the search for the most beneficial rules using a GA, rather than whittling down the already low number of templates. That is not to say that the use of a GA was not successful, as this allowed the Brill GA-tagger to find more and better rule templates than what is used in the Brill tagger.

Additionally, as detailed in Section 2.3, previous work has resulted in an immense amount of potential variations and optimisations which can be made to GAs. Hence, it is far from clear that the results obtained from the application of a GA to the Brill tagger used in this paper is the most optimal. For instance, only the island model variation of parallel GAs was examined, although several others exist. Furthermore, crossover was restricted to solely using one-point crossover, and mutation was consistently locked to the low probability of $p = 0.01$ per individual per generation.

7.2.2 Training Optimisation

A further area in which improvements could have been made in the methodology is in the optimisation of the training phase. The implementation used here borrowed traits from Ngai and Florian (2001), which lead to a speed-up by a factor of 10. However, due to time constraints this was not fully optimised. Implementing this fully would most likely speed up the training phase by approximately a factor of 100, as reported by Ngai and Florian (2001).

7.3 Discussion of Evaluation

The evaluation procedure used in this study is one that is commonly applied when evaluating PoS taggers. The use of k-fold cross-validation generally provides a good indication of how well a tagger performs, while avoiding any potential influence from testing the tagger on a particularly challenging or simple part of the given corpus. Adding in the fact that the taggers were evaluated on a relatively large variety of languages and types of texts, it should be fair to consider the validity of this study as sufficiently high.

7.4 Discussion of Results

The Brill GA-tagger introduced in this work was able to assign correct PoS tags to words significantly better than the Brill tagger. This improvement was consistent across all 9 languages used in this study, with the Brill GA-tagger achieving a significant error rate reduction (two tailed McNemar's test, $p < 0.001$) of approximately 2% – 15% for each language. The exception to this was the experiment with assumed rule independence, in which the Brill tagger outperformed the Brill GA-tagger. The implementation using an Island Model GA yielded no further improvements.

7.4.1 Cross-lingual Performance

The sample of languages used in this work can be considered to provide a sufficiently large foundation for the purposes of evaluating the Brill GA-tagger's cross-lingual performance. As expected, the Brill GA-tagger learns to use templates which are well suited for the languages at hand. That is to say, for languages for which a certain type of templates do not make much sense, such templates are not used. For instance, lexical templates are not learned at all for Japanese, which has relatively free word order. In other words, the tagger learns that it does not make much sense to use a template which requires some sort of consistent and systematic lexical ordering for Japanese. For Chinese, on the other hand, the lexical templates are used extensively, as several PoS tags can be accurately determined from their lexical context.

7.4.2 Assumed Rule Independence

Hepple (2000) found that assuming rule independence only marginally worsened the accuracy of the Brill tagger (see Section 2.5.1). The results found in this study largely confirm this finding. However, this assumption affects the Brill tagger and Brill GA-tagger to different extents. As shown in Table 7, the Brill GA-tagger performs significantly worse than the Brill tagger when this assumption is made. A closer inspection of the rules reveals that this outcome could have been expected. This is due to the fact that the taggers learn the most effective rules first, and the fact that the Brill GA-tagger has access to quite general templates. In the Brill GA-tagger, the first rules to be learned are very general and affect a very large number of tags. This then leads to a greater number of tagging mistakes, which are now committed and unchangeable (see Section 2.5.1 for details of the IC procedure). The Brill tagger, on the other hand, can be described as being more careful in its approach, as such over-generalisations are avoided. It should be noted that the Brill GA-tagger can easily correct these mistakes from the more general rules, when *not* assuming rule independence.

7.4.3 Rule Comparison

A closer look at the rules learned by the Brill GA-tagger and Brill tagger reveal some interesting results. For one, the Brill GA-tagger is able to find rules which are much more effective than the Brill tagger. This can be explained by the fact that the Brill GA-tagger can create rules which are more general than the rules generated from the standard tagger. This in turn is caused by the templates with optional tags. Take, for instance, the Brill GA-tagger's Icelandic rule changing *D-A* to *D-N* when the preceding tag is either *NS-N* or *N-N* (Table 8). This rule corrects more mistakes than the Brill tagger's equivalent Icelandic rule which performs this transformation *only* when the preceding tag is *N-N*. That is not to say that the Brill tagger would not learn the this transformation when the preceding tag is *NS-N*, provided that the errors corrected from carrying out this transformation are sufficiently numerous. However, this means that the Brill tagger will require more rules in its patch list to achieve performance equivalent to the novel tagger. Furthermore, this requires the cut-off frequency at which no further rules are learned to be sufficiently low for the rule to be learned at all.

Additionally, the Brill GA-tagger continues to find rules with sufficiently high scores when the baseline tagger is unable to. In other words, the Brill tagger can not find rules with scores higher than the cut-off frequency, even though the Brill GA-tagger is able to. This can be explained by the fact that that the automatically defined templates allow for a much higher complexity than the standard Brill templates. For instance, the Brill GA-templates allow for tags to be contingent on e.g. the next tag and the word following that tag. Although the resulting rule from such a template would rarely have any effect it can be useful for set phrases or idioms, such as *as good as*, as mentioned previously.

7.4.4 Automatically Obtained Templates

A closer inspection of the templates learned (see Appendix 2) reveals that they are in general quite simple and also quite local. The *simplicity* of the templates is found in that they generally contain 3 or fewer active positions in their bit matrix representations. Although this is far less complex than what the model used allows for, 3 active positions is more complex than what is specified by the standard Brill templates. That is to say, the most complex templates are not used, which can be explained by the fact that the positions in a corpus where a rule can be applied decreases quickly as the complexity of the template increases – a form of *data sparsity.* This means that even though a search optimised with a GA can be used as a tool to generate complex templates which fit some pattern in the target language, these templates run the risk of receiving low fitness values if the patterns they match are too infrequent. However, using a sufficiently large corpus in the training material might lead to these templates being learned and used.

Locality is the second feature most templates have in common. The activated positions in the bit matrices which represent the rule templates are generally close to the focus word. This indicates that it is perhaps not necessary to encode as large a context as is done in this study, where words as far as 4 positions away from the tag to be changed were considered.

7.5 Relevance and Potential Applications

This thesis has resulted in what should be fair to consider a somewhat unique and useful tagger. The Brill GA-tagger has been shown to be relatively language-independent, as its accuracy is better than that of the Brill tagger for all languages used in this study. However, the tagger's accuracy does not compare well to other state of the art PoS taggers for the languages tested (cf. Täckström (2013, p. 170)). Even so, the tagger should hopefully be of use in cases where a language-independent tagger is needed. One such case could be to use the Brill GA-tagger as the final tagger in a cascaded tagger, in which several taggers are successively applied. For instance, a stochastic PoS tagger might make quite systematic errors which the Brill GA-tagger could easily find and correct, thus boosting the performance of the cascaded system as a whole.

Although research on the Brill tagger was the focus of a substantial amount of research in the decade after its creation, the interest in this topic appears to have died out in recent years. Even so, this work does hold some relevance to the field of NLP as it stands today, considering that the use of this rule-based system allows for easier inspection of the rules generated, which could be of particular interest for under-resourced languages.

7.6 Suggestions for Future Work

There are several ways for future work to improve upon the Brill GA-tagger presented in this paper, the most important of which are listed here.

7.6.1 Representing Feature-Rich Rules as Finite-State Transducers

Figure 16 showed an FST encoding two simple English rules. For rules such as these, a transformation into an FST can be carried out through following the steps presented by Roche and Schabes (1995). Due to the fact that rules which rely on some sort of lexical information or more complex morphological information are not supported in this procedure, this transformation could not be fully implemented in the Brill GA-tagger. A suggestion for how this might be done in future work is presented here, seeing as an actual implementation was beyond the scope of this study.

As mentioned in Section 2.2.3, the limitation of the implementation presented by Roche and Schabes (1995) can be explained by the fact that the FST they describe reads *one* input tape consisting of PoS tags and writes *one* modified output tape containing the PoS tags as corrected by the Brill tagger's rules. This approach is acceptable when considering the output tape of the FST. As for the input tape, however, this approach is insufficient when the tagger includes rules which require more than one feature. For instance, if a rule using lexical contextual information is added to a set of rules using PoS tags, there is no clear way of representing this rule. After all, the FST is completely oblivious as to which word each PoS tag in the input actually belongs to.

A suggestion which should prove to be sufficient is to have several input tapes, similarly to an n-tape FSA (see Elgot and Mezei (1963))[1]. The number of input tapes used should be equal to the number of features used by the tagger. For instance, when using PoS tags and lexical contexts, the number of input tapes should be 2. However, considering that such devices may not be determinable, each separate input tape should be used to form a separate FST. Each of these FSTs can then be applied in succession to the input sequence. This should allow the tagger to

[1] An n-tape FSA is defined as an FSA (see Section 2.2), with the addition of a *tape selector function*. This function specifies which tape the FSA should read the next input symbol from.

tag a given input sequence in an efficient manner, with a time complexity of $O(kn)$ where k is the amount of features and n the length of the input sequence.

7.6.2 Improved Initial Tagging

An important step towards improving the overall performance of the tagger is to improve the initial tagging. As detailed in Section 2.1.1, the Brill tagger first assigns to each word a tag which is determined from non-contextual information. In this work, this simply consisted of assigning to each word the PoS tag with which it most frequently occurred in the training material. Although this is sufficient for the purposes of investigating whether the application of GAs improves the tagger's accuracy, it is somewhat sub-optimal if this is considered from a real-world perspective. Future research should improve upon this, through for instance using morphological analysis. Such tools are often strictly language-dependent, however, so in order to keep to the language-independent vein of the Brill GA-tagger, any improvements to the initial tagging should also be as language-independent as possible.

7.6.3 Optimisation Techniques

The search for rule templates used in this paper was optimised using a GA. There are two main ways in which this could be developed further. The first of these is to simply experiment more with the GA setup. Seeing as GAs have a large amount of parameters which can be adjusted, it is quite likely that fine-tuning the parameters used in this study could lead to a further improved PoS tagger.

Additionally, it may be interesting to see a comparative analysis of different search optimisation techniques for the Brill GA-tagger. As mentioned in Section 2.3, a GA generally arrives at what is an *acceptably good* solution to a problem. It is not unthinkable that other types of search optimisation techniques might perform better than GAs, potentially leading to improvements to this tagger – in terms of time consumption, tagging accuracy or both.

7.6.4 Complex Templates

The templates used in this work encoded one or more out of three features: the presence of a PoS tag, the presence of a lexical item or the presence of one out of several PoS tags. These templates could be extended further to include other features. This should further boost the tagger's performance, considering that work by both Megyesi (1999) and Acedański and Gołuchowski (2009) has shown that such improvements are helpful for certain languages. For instance, allowing partial features is suitable for languages with more inflectional morphology, where one might want to alter only a morphosyntactical marker of a tag, such as changing a *singular* tag to a *plural* tag. The current implementation only implicitly uses this information, as the tagger considers e.g. *NN SIN* to be a completely different tag than *NN PLU*. This could be improved upon through handling this in a more informed manner.

It should be noted that including the possibility of having more complex templates can only be beneficial to the tagger's performance, as the rules generated from these templates will only be utilised if they correct a sufficient amount of mistakes. For instance, the lexical feature in the templates used in this work does not make much sense for languages with relatively free word order such as Japanese – this simply leads to that no rules containing this feature are used for such languages.

7.6.5 Larger Language Sample

Although the sample of languages used in this work was relatively substantial, it would be interesting to see how well the Brill GA-tagger performs on an even larger variety of languages. Additionally, training and evaluating the tagger on larger data sets than the ones used for some of the languages here would likely yield better results.

8 Conclusions

The following three questions were posed in Section 2.6.

1) Can the Brill tagger be made more language independent by defining rule templates automatically by using Genetic Algorithms?

The Brill GA-tagger developed in this work achieved significantly higher accuracy than the Brill tagger. This study has thus shown that using Genetic Algorithms when searching for rule templates can improve the performance of the Brill tagger for many languages. However, the increase in performance is relatively similar for each language.

2) To what extent can the results obtained here be generalised for other languages?

The tagger was evaluated using 9 languages (Chinese, Japanese, Turkish, Slovene, Portuguese, English, Dutch, Swedish or Icelandic). The results should therefore be generalizable across several languages, and at the very least for languages similar to the ones evaluated.

3) Does the assumption of rule independence made by Hepple (2000) extend to the complex and general rules used in the Brill GA-tagger?

Assuming rule independence resulted in far worse performance. This can be explained by the fact that the Brill GA-tagger first learns more general rules than the Brill tagger. These rules cause more errors which it is unable to fix, because of the assumed rule independence.

Bibliography

Acedański, S. (2010). A morphosyntactic Brill Tagger for inflectional languages. In *Advances in Natural Language Processing*, pages 3–14. Springer.

Acedański, S. and Gołuchowski, K. (2009). A morphosyntactic rule-based Brill tagger for Polish. In *Proceedings of Intelligent Information Systems*, pages 67–76.

Afonso, S., Bick, E., Haber, R., and Santos, D. (2002). Floresta sintá(c)tica: a treebank for Portuguese. In *Proceedings of LREC*, volume 2002.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111. L. Erlbaum Associates Inc.

Bender, E. M. (2009). Linguistically naïve != language independent: Why NLP needs linguistic typology. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 26–32. Association for Computational Linguistics.

Brill, E. (1992). A Simple Rule-Based Part-of-Speech Tagger. In *Proceedings of the workshop on Speech and Natural Language*, pages 112–116. Association for Computational Linguistics.

Brill, E. (1994). A Report of Recent Progress in Transformation-Based Error-Driven Learning. In *Proceedings of the workshop on Human Language Technology*, pages 256–261. Association for Computational Linguistics.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational linguistics*, 21(4):543–565.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 133–140. Association for Computational Linguistics.

Das, D. and Petrov, S. (2011). Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 600–609.

DeJong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor.

Džeroski, S., Erjavec, T., Ledinek, N., Pajas, P., Žabokrtsky, Z., and Žele, A. (2006). Towards a Slovene dependency treebank. In *Proc. of the Fifth Intern. Conf. on Language Resources and Evaluation (LREC)*.

Elgot, C. C. and Mezei, J. E. (1963). Two-sided finite-state transductions. In *Switching Circuit Theory and Logical Design*, pages 17–22.

Elgot, C. C. and Mezei, J. E. (1965). On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68.

Freeman, A. (2001). Brill's POS tagger and a morphology parser for Arabic. In *Proceedings of ACL Workshop on Arabic Language Processing*.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

Goldberg, D. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996.

Goldberg, D. and Holland, J. (1988). Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99.

Gorges-Schleuter, M. (1991). Explicit parallelism of genetic algorithms through population structures. In *Parallel Problem Solving from Nature*, pages 150–159. Springer.

Heeman, P. A. and Allen, J. F. (1999). Speech repairs, intonational phrases, and discourse markers: modeling speakers' utterances in spoken dialogue. *Computational Linguistics*, 25(4):527–571.

Hepple, M. (2000). Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 278–277. Association for Computational Linguistics.

Holland, J. (1975). Adaptation in natural and artificial systems. *Ann Arbor, MI*, 1(97):5.

Hopcroft, J. (1971). An n log n algorithm for minimizing states in a finite automaton. Technical report, DTIC Document.

Hopcroft, J., Motwani, R., and Ullman, J. (1979). *Introduction to automata theory, languages, and computation*, volume 2. Addison-wesley Reading, MA.

Jurafsky, D. and Martin, J. (2009). *Speech and Language Processing*. Pearson Education Inc., 2nd edition.

Källgren, G. (2006). Documentation of the Stockholm Umeå Corpus. In Gustafson-Čapková, S. and Hartmann, B., editors, *Manual of the Stockholm Umeå Corpus version 2.0*, pages 5–85. Department of Linguistics, Stockholm University.

Kawata, Y. and Bartels, J. (2000). *Stylebook for the Japanese treebank in VERBMOBIL*.

Kleene, S. (1956). Representation of events in nerve nets and finite automata. In Shannon, C. and McCarthy, J., editors, *Automata Studies*, pages 3–41. Princeton University Press.

Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech & Language*, 6(3):225–242.

Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning 2001*, pages 282–289.

Loftsson, H. (2007). Tagging Icelandic text using a linguistic and a statistical tagger. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 105–108. Association for Computational Linguistics.

Manning, C. D. (2011). Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *Computational Linguistics and Intelligent Text Processing*, pages 171–189. Springer.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330.

Marques, N. C. and Lopes, G. P. (2001). Tagging with small training corpora. In *Advances in Intelligent Data Analysis*, pages 63–72. Springer.

Mayfield, J., McNamee, P., Piatko, C., and Pearce, C. (2003). Lattice-based tagging using support vector machines. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 303–308. ACM.

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079.

Megyesi, B. (1999). Improving Brill's POS tagger for an agglutinative language. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 275–284. Citeseer.

Miller, B. and Goldberg, D. (1995). Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Urbana*, 51:61801.

Mohri, M. (1994). Compact representations by finite-state transducers. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 204–209. Association for Computational Linguistics.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311.

Mohri, M. (2000). Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1):177–201.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153.

Ngai, G. and Florian, R. (2001). Transformation-based learning in the fast lane. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8. Association for Computational Linguistics.

Oflazer, K., Say, B., Hakkani-Tür, D. Z., and Tür, G. (2003). Building a Turkish Treebank. In Abeille, A., editor, *Building and Exploiting Syntactically annotated Corpora*, pages 261–277. Kluwer Academic Publishers.

Poli, R., Langdon, W. W. B., and McPhee, N. F. (2008). *Field Guide to Genetic Programming*. Lulu Enterprises Uk Limited.

Prütz, K. (2002). Part-of-speech tagging for Swedish. *Language and Computers*, 43(1):201–206.

Ramshaw, L. and Marcus, M. (1994). Exploring the statistical derivation of transformational rule sequences for part-of-speech tagging. In *Proceedings of the ACL Workshop on Combining Symbolic and Statistical Approaches to Language*, pages 128–135.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in natural language processing*, volume 1, pages 133–142. Philadelphia, PA.

Rechenberg, I. (1994). Evolution strategy. *Computational Intelligence: Imitating Life*, 1.

Reed, P. M., Minsker, B. S., and Goldberg, D. E. (2001). The practitioner's role in competent search and optimization using genetic algorithms. In *World Water and Environmental Resources Congress, Washington, DC*.

Roche, E. (1993). *Analyse syntaxique transformationnelle du français par transducteurs et lexique-grammaire*. PhD thesis, Université Paris 7.

Roche, E. and Schabes, Y. (1995). Deterministic Part-of-Speech Tagging with Finite-State Transducers. *Journal of the Association for Computational Linguistics*, 21(2):227–253.

Roche, E. and Schabes, Y. (1997). *Finite-state language processing*. MIT press.

Rögnvaldsson, E., Ingason, A. K., and Sigurðsson, E. F. (2011). Coping with variation in the Icelandic Diachronic Treebank. *Oslo Studies in Language*, 3(2).

Salomaa, A. (1973). *Formal Languages*. Academic Press, University of Michigan.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of international conference on new methods in language processing*, volume 12, pages 44–49. Manchester, UK.

Schützenberger, M. P. (1977). Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4:47–57.

Sivanandam, S. and Deepa, S. (2008). *Introduction to Genetic Algorithms*. Springer Berlin Heidelberg.

Starkweather, T., Whitley, D., and Mathias, K. (1991). Optimization using distributed genetic algorithms. In *Parallel problem solving from nature*, pages 176–185. Springer.

Täckström, O. (2013). *Predicting Linguistic Structure with Incomplete and Cross-Lingual Supervision*. PhD thesis, Uppsala University.

Tanese, R. (1989). Distributed Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 434–439. Morgan Kaufmann Publishers Inc.

Van der Beek, L., Bouma, G., Malouf, R., and Van Noord, G. (2002). The Alpino dependency treebank. *Language and Computers*, 45(1):8–22.

Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the third international conference on Genetic algorithms*, volume 1, pages 116–121.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85.

Whitley, D. and Starkweather, T. (1990). Genitor II: A distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 2(3):189–214.

Wilson, G. and Heywood, M. (2005). Use of a genetic algorithm in brill's transformation-based part-of-speech tagger. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 2067–2073. ACM.

Xue, N., Xia, F., Chiou, F.-D., and Palmer, M. (2005). The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207.

Zhao, W., Zhao, F., and Li, W. (2007). A new method of the automatically marked Chinese part of speech based on Gaussian prior smoothing maximum entropy model. In *Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on*, volume 3, pages 447–453. IEEE.

Appendix - Automatically obtained templates

$$\begin{matrix} PoS \\ Mult. \\ Lex. \end{matrix}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{matrix} PoS \\ Mult. \\ Lex. \end{matrix}\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) PoS tag (+1)

$$\begin{bmatrix} 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \end{bmatrix}$$

(b) PoS tag (-1) and Lexical entity (+1)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) Optional (-1) and PoS (+1)

$$\begin{matrix} PoS \\ Mult.. \\ Lex. \end{matrix}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{matrix} PoS \\ Mult. \\ Lex. \end{matrix}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \end{bmatrix}$$

(d) Lexical entity (+1)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(e) Lexical entity (-1)

$$\begin{bmatrix} 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(f) PoS tag (-1)

$$\begin{matrix} PoS \\ Mult. \\ Lex. \end{matrix}\begin{bmatrix} 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{matrix} PoS \\ Mult. \\ Lex. \end{matrix}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(g) PoS tag (-1) and multiple tag (+2)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(h) Multiple tag (-2)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(i) PoS tag (+1)

Figure 18: Some of the templates learned by the tagger for English (a, b, c), Chinese (d, e, f) and Japanese (g, h, i)

Stockholms universitet/Stockholm University
SE-106 91 Stockholm
Telefon 08 - 16 20 00
www.su.se

Stockholms universitet