USING N-GRAM STATISTICS FOR "CRYPTIC" LANGUAGE WORD SEGMENTATION, AUTO-COMPLETION AND AUTO-CORRECTION

Author: Milos Colic, student ID: 814817

Dipartimento di Elettronica, Informazione e Bioingegneria

Facolta di Ingegneria MI – Ingegneria Informatica MI

Politecnico di Milano

Supervisor: Matteo Matteucci

Co-advisor: Simon Martin

This thesis is submitted for the Master of Science degree

September 2015

ABSTRACT

This thesis aims to deal with the issues of auto-completing and auto-correcting commands formed using the syntax of an artificial language called “Cryptic”. This particular language is a proprietary language defined and put into production by the multinational company Amadeus SAS France. Despite the fact that “Cryptic” is an artificial language, the techniques used in natural language processing are used trough out this thesis.

Data available in Amadeus “Cryptic” logs is composed of unsegmented commands; therefore no notion of words was present in the command logs. Taking this as starting point reasonable approach is to build n-gram statistics at the symbol level, which will provide estimators on relations between symbols. On the other hand, one important attribute of this system is that each command is of a finite length. This fact implies that every command starts with a special symbol Ш. Cyrillic symbols are used to denote any special character that supplements “Cryptic” alphabet. Implicit starting symbol simplifies the model into a tree structure, instead of a graph, and the root of this structure is a node with label Ш. By combining B+ trees, FP trees, and hash-based indexes we have obtained a hybrid tree structure that facilitates various kinds of fast searches that are needed in the algorithms used in the process of building the system.

Based on this tree structure a majority vote procedure is used to segment commands into words and to extract a dictionary of “Cryptic” language in an automated way. Majority voting bases its decision on various segmentation indicators such as Shanon’s mutual information function, entropy value of an n-gram and probability variance per symbol pair. Once a dictionary is formed statistics based on a notion of language words can be produced and used to detect errors; after errors have been detected corrected output is produced.

Validation of a subsystem dedicated for command segmentation is computed on the corpus of English language, because the corpus of “Cryptic” commands, which are properly segmented and labeled, does not exist. Results obtained by performing validation on English language corpus are taken as proof of concept. Any decrease of precision in segmentation on “Cryptic” commands (which unfortunately at the moment cannot be measured) is assigned to the larger amount of ‘randomness’ in the language creation. “Cryptic” language was incrementally created by various teams over a period of more than 20 years and lacks any formal standardization, a major part of the innate

language randomness comes from this fact. Validation on auto-complete and auto-correct capabilities is done on the portion of the data logs that is not used in the training phase.

ACKNOWLEDGEMENTS

I would like to give my thanks to Amadeus SAS, the department in Nice, for providing the resources needed for performing this research. Especially I am thankful to Simon Martin, Leonardo Freitas Gomes and Anthony Hock-Koon for their help during my internship in Amadeus. On the side of Politecnico di Milano I owe a great gratitude to my tutor on this thesis Matteo Matteucci, his guidance was of great value to my work.

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

- Ш – Command start symbol/word
- Ф – Word delimiter symbol
- Ж – Command end symbol/word
- З – Error symbol
- MCM – Markov Chain Model
- HMM – Hidden Markov Model
- CHT – Character level n-gram tree
- WRDT – Word level n-gram tree
- CMDT – Command level n-gram tree
- EP – Error prefix
- ES – Error suffix
- DNA – Deoxyribonucleic acid

1 INTRODUCTION

1.1 “Cryptic” language

“Cryptic” language is a language designed and implemented by Amadeus SAS engineering division. Over more than 20 years, different teams have implemented various commands that have been incrementally added to the syntax of the “Cryptic” language. The language itself was conceptualized as a minimalistic language that is mainly comprised out of abbreviations for words from the English language. One example of such word is ‘an’ which stands for ‘availability numerical’, another example would be ‘ce’ that stands for ‘car equipment’. From these two examples, one would deduce that all words in “Cryptic” are comprised from starting letters of individual words from English language description. However, even that notion is not homogenous in these two simple cases, in the first example the order of words is inverted w.r.t. what we would encounter in actual English language description. Due to high heterogeneity of cultures that comprise teams, geographical and temporal displacement of different teams, word definitions are as well extremely heterogeneous. If we observe word ‘dnn’ which stands for ‘decode hotel rate’ it becomes clear that some words are created as meaningful abbreviations. Others, however, were selected just because this precise short string was not taken at the moment of implementation of particular system functionality.

To understand why such words were introduced and why English language (or any other natural language) words were not used, one should focus on the throughput of Amadeus system. Amadeus designs and implements large-scale distributed systems used by travel agencies, airlines and end customers. This system provides functionalities for searching and booking tickets and reservations in all segments of the travel industry,

such as airline and train tickets, car rental, hotel bookings and perform various other travel-related activities. Amadeus system at the peak time has over 200.000 transactions per seconds and over 15.000 agents spread over the world using its functionalities. Through evolution, Amadeus system has kept as part of its architecture various legacy subsystems. One of such legacy subsystems is communication system based on “Cryptic” language and restructuring this part of Amadeus system would be extremely costly and time-consuming. At the moment of creation, communication subsystem was confronted with the issue of limited available network bandwidth since the most common internet connection medium at that particular time was the dial-up modem. This fact imposed a requirement to reduce the length of every command as much as possible. Since every command has a strictly defined list of needed arguments to execute correctly, the only possible approach in compressing the commands is to reduce the length of the words. In practice, what can be done is that instead of ‘Belgrade’ a code substitution ‘beg’ is used. This way variable power compression is performed. To understand what is meant by variable power consider the following example. Word ‘Belgrade’ is replaced by ‘beg’ and we have compression of approximately166%. On the other hand, word ‘Nice’ is replaced by ‘nce’ and we have only 25% compression. There are as well cases that have 0% reduction like word ‘Nis’, but these are few in numbers.

This optimization even though improves compression of commands and uses more optimally system’s bandwidth it includes some new variability and randomness into the syntax of the words. These particular words are standardized since they were firstly used in airport management, and they always provide unified length representation (3 character long code) of any airport in the world. The only case in which standardization falls apart is when the city has more airports, and then we have a city code that maps all involved airports, and each airport has its additional code.

All these issues arise from just one particular command denoted as ‘an’, and if these issues were not enough, there are additional variability with each command. Since every command can be used in more than one context, the number of arguments may vary depending on the usage. This attribute of “Cryptic” language highly resembles how humans form sentences in natural languages, although no formal grammar is presented. How the system in practice deals with each particular command is defined statically, every team has developed a parser for a specific command they have implemented. What is worth mentioning is that it is almost impossible to collect all these parsers and

use them in this thesis. Collecting all the parsers would imply a high cost of implementation and the increase of precision would not correspond proportionally to invested time and money. That is why in this thesis focus is put on a generic solution that, based on statistical and probabilistic qualities of the observed data, segments commands into words.

To illustrate everything previously said let's observe the following command:

$$an \uparrow par \uparrow nce$$

Symbol ↑ corresponds to the implied separator, this separator is not typed, but it could be present as ' ' if the user chooses to do so. As it can be noticed this is another source uncertainty in data, in some cases (very rarely) we have spaces between words and even in these case spaces are not necessarily placed in all respected positions. Examples of 'correct' input for the previous command are following:

$$anparnce$$

$$an\ par\ nce$$

$$an\ parnce$$

$$anpar\ nce$$

The fact that we can both encounter implicit and explicit word separators lead to the assumption that no word separators are ever present. In the case word separators are present they are removed as a part of data pre-processing. Up until now we have not given any syntactic meaning to the command above. Actual meaning behind string 'an↑par↑nce' is numerical availability (an) for a flight from Paris (par) to Nice (nce). Now it becomes logical that a user might want to specify a date in this format leading to 'an↑11↑aug↑par↑nce', it is noticeable that in this case number of words have changed from 3 to 5. In addition to this option, there are many others like specifying round trip, querying for a specific seat class, seat number, luggage, and other options.

Another interesting attribute of "Cryptic" language is the fact that there is strict split between functionality carrier words and argument words. This fact can be related to natural languages where we observe words with different semantic roles, such as nouns, verbs, articles, and other word types. Although one crucial difference needs to be stated, in "Cryptic" significant number of arguments practically appearing in logs are statistically unimportant. Put into simple terms, many arguments that we encounter in actual system usage are one time words. One time word should be understood as words

similar to passenger name (to some extent), passenger age (more notable since it changes over time), most notably flight numbers. These words are statistically irrelevant since it is highly unlikely that we can predict such words. We can predict one time words only in cases when an agent had sold a significant number of tickets to the same person, but this rarely happens. On the other hand, functionality carrier words are statistically important words. Functionality carrier words refer to command identifiers. Through them, we can predict some arguments such as airports, in some cases dates, commands that follow this particular command, and other words that have strong relations. It is reasonable to observe that certain regions observe traffic flow peak in a certain period of the year. This fact can induce possibility of predicting dates, even though they are one time words. For example, Nice airport has considerably more traffic during summer period due to the vicinity of touristic attractions. It is reasonable to predict that weekend dates in the month of August of the current year are very probable as command arguments if the end destination is Nice. Why functionality carrier words allow sequences of commands to be predictable comes from the fact that the “Cryptic” language is a preorder language. Functionality carrier word is always the first word of the command, and all following words are the arguments. It can be noticed that if we observe functionality carrier words in context free manner (without any arguments) we can spot frequent patterns of commands and suggest to the user which could be the following command.

This section of the thesis was intended just as an illustration of complexity and variability of “Cryptic” as a language. It also casts light on the reasons for including unsupervised word segmentation into the study; this is because no dictionary is present. In addition, no formal grammar available nor labeled and segmented train data corpus from which words can be learned based on Zipf’s law. The only possible approach is to use unsupervised word segmentation that relies on probabilistic measures on symbol level. This particular topic will be covered with more care in Chapter 3.

1.2 Auto-completion and auto-correction as engineering techniques

Though its development Internet became a tool used to distribute information. From this fact, we understand that average user of the Internet should be allowed to query data available on the Internet as quickly and flexible as possible. All major web search engines provide auto-completion and auto-correction functionalities which

facilitate more flexible and user-friendly environment. A necessity of simplifying this querying process emerged from the ever-increasing amount of information available on the Internet and the fact that typographical errors in the process of querying Internet data increase the difficulty in obtaining desired data.

Every time average user of any major web search engine (e.g. Google) provides as query string something that is highly unlikely it encounter the following message 'Did you mean *' (* denotes the corrected input string); this is nothing more than auto-correction functionality. Another example of autocorrect functionality is the one an average user of smart phone encounters every day. While using the messaging service on any new generation smartphone, (of course unless the user disables such functionality) experiences help from the application that corrects typographical errors. These two techniques may seem different to some extent. Web browser strategy is more lethargic than smartphone strategy; it waits for the user to press 'Enter' and only then it provides the corrected input. On the other had smartphone strategy automatically corrects the input while the user is typing, this strategy is more aggressive in the implementation of auto-correction. Previously explained differences do not affect the fact that both techniques follow the same background logic based on hidden Markov models [4] and n-gram statistic concepts [17]. Both hidden Markov models and n-gram statistics will be covered with more care in Chapter 2. In the context of this observation, it is enough to conceptualize hidden Markov models and n-gram statistics as "happens before" relations between words of one language. Take as an example the following sentence:

$$\text{“}High\ school\ education\ system\ *\text{”}$$

In this particular sentence * denotes the rest of user's input and it is of no significance to the discussion, it is just placed here to complete the context. An n-gram "happens before" relation is observed between words 'high', 'school', 'education', and 'system'. This relation will describe how likely is that if we observe word 'high' the next word will be 'school'. Based on the first word 'high', the system should be able to predict that one of the probable next words is 'school' and not 'shoe' and correct the user's input.

Auto-complete functionality relies on the same concepts as auto-correct functionality does, implementation is slightly different since in this case system instead of correcting the user it suggests to the user what to type next. Put into the frame of the previous example, the system should be able to predict the word to follow 'high school'

could be ‘education’. Of course suggestions and corrections highly depend on the data used to learn these “happens before” relations, more on this topic will be presented in Chapter 2.

Due to observing compression of words which could often confuse end users, Amadeus “Cryptic” language would greatly benefit from auto-complete and auto-correct functionalities. These modules will supplement interaction between the user and the command line terminal used to issue “Cryptic” commands. As previously discussed the “Cryptic” language is in some of its attributes similar to natural languages, but in some other attributes it differs from them. Due to the highly expressed randomness in the “Cryptic” language dictionary it can be noticed that symbol/character level predictions would provide low-quality output. Words such as ‘pd’ and ‘pf’ are words with only one letter difference. The system might discriminate one word over the other just because ‘pd’ has larger frequency than ‘pf’; regardless of possible high order relation among other words. This kind of probabilistic ‘discrimination’ can be avoided if the prediction is observed on the word level, and the fact that the “Cryptic” is pre-order language will tune prediction toward more reliable estimates. Adding more context in predicting input increases stability of the solution, but, on the other hand, it increases complexity as well. More on the actual implementation of auto-complete and auto-correct functionalities in the “Cryptic” language will be presented in Chapters 4 and 5 respectively.

1.3 Word segmentation as engineering technique

Word segmentation refers to the method that automatically adds word separator symbols (usually space in natural language) to unsegmented data. Currently in the state of the art systems, there are various techniques, which differ on the fact if there is segmented train data corpus available or not. Also, these methods differ on the criteria on which the position of word separator is determined.

In case a segmented train data corpus is available, extraction of language dictionary (in some bibliography also referred as lexicon) becomes straightforward. Words are groups of symbols between two separator symbols as well as the first and the last words which are marked by sentence separator symbols on one of their sides. Observe the following example:

Ш$Mary\Phi and\Phi Jane\Phi like\Phi to\Phi eat\Phi croissants\Phi for\Phi breakfast$Ж

In the previous example symbols Ш, Ф, and Ж denote beginning of the sentence, word separator, and ending of the sentence respectively. As have been established earlier these symbols can be freely chosen symbols that are outside of the language alphabet. In the following sections, alphabet set will be denoted as A_l and set of supplementing symbols will be denoted as S_l. From the provided sentence it is easy to extract set of words, any group of symbols that falls between two symbols Ф, or between Ш and Ф, or between Ф and Ж are considered a word of the language's dictionary. After the notion of dictionary has been established we can define notion of word frequencies in the train data corpus. Number of occurrences of a particular word is said to be the frequency value of this word. During the word extraction phase of word segmentation, all words frequencies are computed; every time a word is detected to be a duplicate of previously encountered word this word's frequency value is incremented. As have been described in [1], words in a natural train data tend to follow Zipf's law or extended Zipf's law defined latter by Mandelbrot. Zipf's law states that we can order words in natural language data by their frequency values and based on this ordering we can define following probability distribution mass function:

$$f(r) \sim \frac{1}{r^{\alpha}}. \quad [1.1]$$

In the equation [1.1], variable α is a parameter of the distribution, r represents the rank of the word in the data w.r.t its frequency and $f(r)$ represents probability distribution of the word's rank in the data. The parameter α usually takes empirical value close to 1 for natural languages [6]. Why this probability distribution mass function is defined as proportional to the fractional value on the right is because the actually value of probability mass function is normalized by factor $N_n$which represent harmonic series sum of order n, which is computed in accordance with following formula:

$$N_n = \sum_{k=1}^{n} \frac{1}{k}. \quad [1.2]$$

On the other hand Mandelbrot's extension of Zipf's law is a simple modification of the previously stated law by a distribution parameter β which empirically tends to the value of natural number e. Mandelbrot's extension is described by a following equation:

$$f(r) \sim \frac{1}{(r+\beta)^{\alpha}}. \quad [1.3]$$

It can be easily noticed that Zipf's law is a restriction of Mandelbrot's law by simply fixing value of β to 0. Word segmentation based on previously segmented train data can use either two of these laws to define value function of a particular word (also referred in bibliography as token) found to be a part of a sentence. Parameters α and β are learned for train data, which should be properly segmented in order to compute frequencies of each word. Once words are ranked w.r.t. their respective frequencies in the train data either Zipf's or Mandelbrot's values of $f(r)$ are computed for each word. Based on these notions segmentation algorithm tries to maximize total value of $f(r)$ for an unsegmented sentence. Total value of the sentence is obtained as a summation of $f(r)$ values of words obtained by placing k word separators in the sentence.

Another class of problem is word segmentation when no lexicon or segmented train data is presented. This type of segmentation problems occupies much attention in Asia's academic circles that are concentrating on natural language processing. This information has its roots in the fact that many of the Asian languages avoid using word separators, implying that sentences in these languages are just long strings of symbols. In these languages, words are determined by context implicitly and not by space positions like in the English language (any many other languages). It becomes clear that most of the training data available in languages such as Japanese, Chinese, Korean, and other Asian languages are all unsegmented. When automated language translations between Asian languages and English language were designed and proposed many word segmentation techniques based on unsegmented data, have arisen as side results. This attribute of Asian languages has made them similar to "Cryptic" language and techniques applicable to Asian languages could as well be applied in the case of "Cryptic".

Most of the techniques, in this case, are based on the notion of mutual information function. As described in [2], mutual information function is defined by the following equation:

$$I(X,Y) = \sum_{x \epsilon X} \sum_{y \epsilon Y} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right). \quad [1.4]$$

What is of particular interest in the word segmentation techniques is the value of information function (as presented in [3]) when x and y have particular values, here denoted as A and B (in general represent a pair of concrete values whose mutual information is of interest). This value is usually denoted as $mi(x = A, y = B)$ and

represents the ratio between joint probability of observing symbols A and B together in the data and probability of independence assumption between symbols A and B. Practically speaking we are not interested in relation between two positions in the string which correspond to variables x and y, instead we are interested in relation between two particular symbols.

$$mi(x = A, y = B) = log\left(\frac{p(x = A, y = B)}{p(x = A)p(y = B)}\right). \quad [1.5]$$

When $mi(x = A, y = B)$ is a lot smaller than 0 we have indication of symbols A and B tending not to appear together and in the case of $mi(x = A, y = B)$ being a lot larger than 0 we have indication of A and B usually appearing together, in the zone close to 0 decision is not established. These observations come from the fact that independence assumption between probability of A and probability of B is only satisfied when $mi(x = A, y = B)$ is equal to 0, in other two cases we values that indicate dependency between A and B. In case of positive values of $mi(x = A, y = B)$ we have indication that joint probability of A and B is larger than product of marginal probabilities of A and B which implies A and B tend to appear together in the data. On the other hand if we obtain negative values of $mi(x = A, y = B)$ we have opposite indications, meaning that A and B tend to appear separately. When $mi(x = A, y = B)$ is equal to 0 we can only claim that independence assumption is satisfied but we cannot claim that A and B tend to appear together nor that they tend to appear separately. It is clear that this value can be useful in deciding where to place a word separator in a sentence to achieve word segmentation. More on this topic will be presented in Chapter 3.

2 N-GRAM STATISTICS

2.1 Markov chain models

To be able to achieve fully understanding of n-gram statistic, firstly some attention should be given to Markov chain models (later denoted as MCMs) and Hidden Markov models. Markov chain models are nothing more than probabilistic final state automatons (as it has been explained in [5]). These models are characterized by a set of states and set of probabilistic arcs that correspond to state transition function. Usually, MCMs are illustrated by a graph representation or by a matrix representation. These two representations are equivalent, although humans are more accustomed to graph representation since it seems to be a more intuitive choice of visualization.

A model of a probabilistic process can be considered as MCM only if it satisfies certain attributes. As have been previously stated MCMs are defined by set of states, this set can be denoted as $S = \{s_1, s_2, \dots, s_n\}$, a MCM starts in a state $s_i \epsilon S$. States in MCM are connected with arcs whose weights are probabilities of transition corresponding to an arc being taken. For example if arc between states s_i and s_j has a weight of 0.3 it means that process corresponding to this particular MCM will change its state from s_i to s_j with probability of 0.3. Occurrence of a state transition is called a step of MCM. Another important attribute of MCMs is that in addition to weights arcs are described with a label as well. Arc labels correspond to the output of the model. What is important to be stated is that if we denote as $A_{o;s_i}$ the set of outgoing arcs corresponding to a state s_i then following conditions must hold:

$$\forall x, y \epsilon A_{o;s_i}, label(x) \neq label(y), \quad [2.1]$$

$$\sum_{x \epsilon A_{o;s_i}} p(x) = 1. \quad [2.2]$$

The first condition is referring to the fact that labels of state's outgoing arcs must be unique, or said in a different manner, two arcs that share a source cannot have the same label. This condition ensures the deterministic behavior of the MCMs. The second condition is referring to the fact that all probability mass must be divided among arcs, and no probability mass should be left unassociated with an arc. In some cases, we can encounter visualizations in which a portion of probability mass is missing. These examples are valid MCMs only if this part of probability mass is associated to implicit self-arc, meaning that the missing probability is linked with the chance of process not changing the state. These cases are quite rare.

To put more light on the MCMs let's consider the following example. A man tosses a fair coin, fair meaning that the probability of getting head is equal to the probability of getting tail, and thus both probabilities are equal to 0.5. Let's denote this coin as C_1. Based on result of the toss he will write a particular letter on a sheet of paper. When toss results in heads the output will be letter 'a' and in case of tails letter 'b'. After fist toss, coin C_1 is replaced with a biased coin C_2, a toss of this coin will result in head in 75% of tosses. The process is repeated with coin $C_2$and after output is produced, either 'a' or 'b' depending on the result of the toss, coins are switched again after result of the toss is observed. A man will play this game indefinitely.

Let's denote as s_1 a state of the world in which a man M is tossing coin $C_1$and as s_2 a state of the world in which he is tossing coin C_2. Transitions between these two states are given by combining probability of an outcome, and the letter M is written on a piece of paper. This is a simple toy example, but it can serve in illustrative purposes. Visualization of this MCM is presented in Figure 2.1. This example can be extended and modified to describe a user writing sentences in the "Cryptic" language. If instead of a coin toss M is tossing a die with n sides, every side of this die corresponds to one word for from the "Cryptic" language lexicon. At this particular point of discussion, let assume all words in dictionary are equally probable, and that all valid commands are composed of 3 words. Let's restrict this system further so more compact example is obtained. For each word position a designated die is used, thus in this experiment we can observe 3 different dies. First die can assume values from set $W_1 = \{an, ig, md\}$,

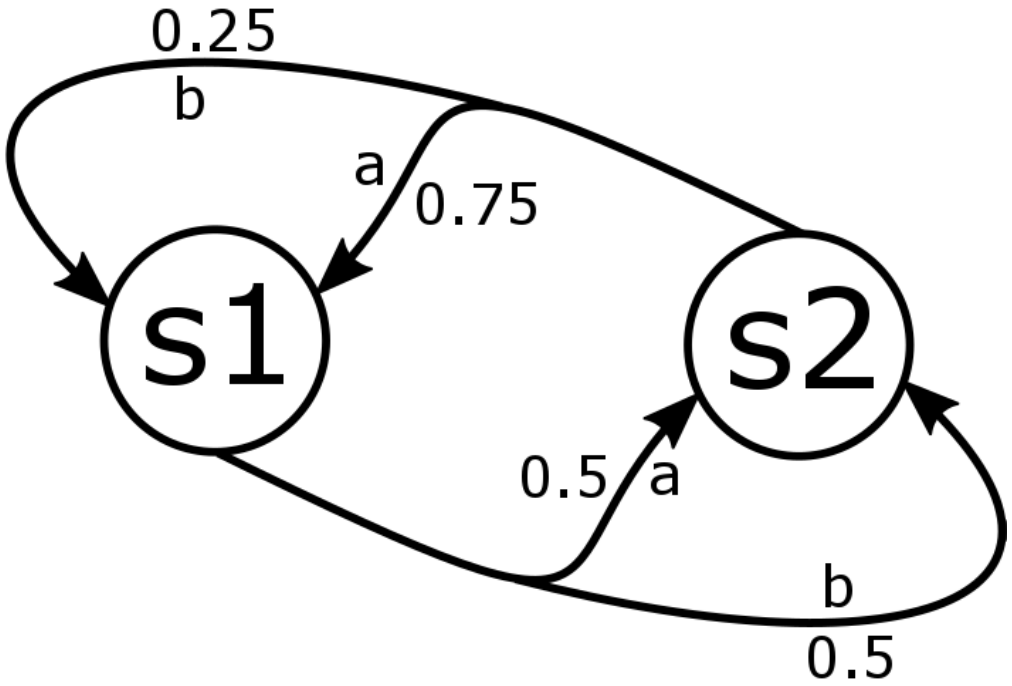

Figure 2.1: Markov Chain Model example of tossing two different coins

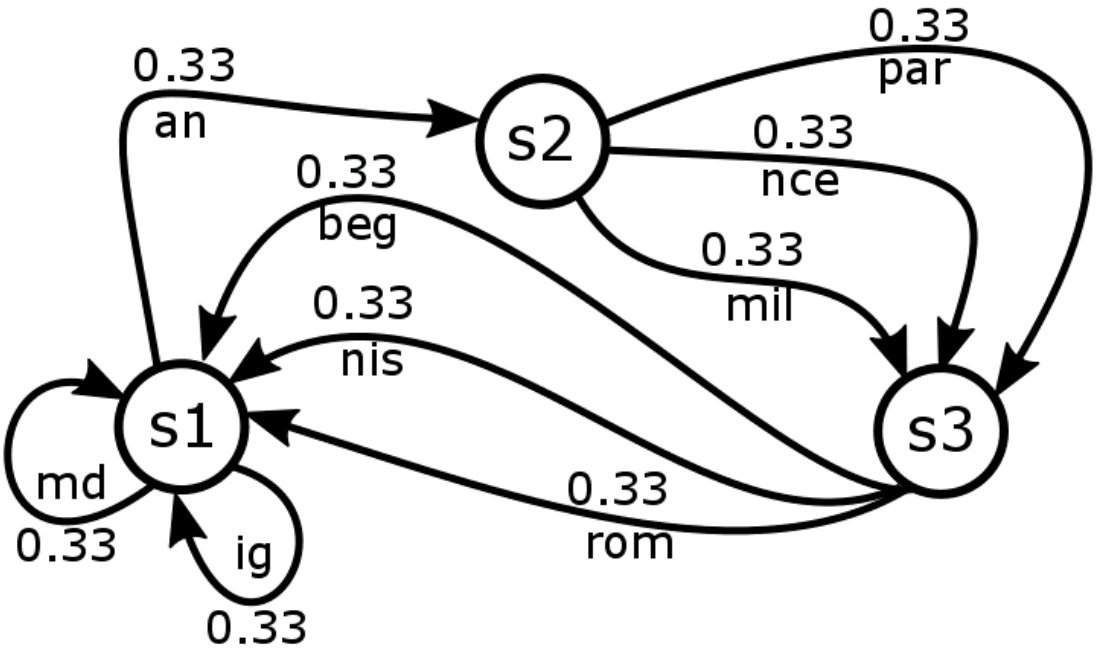

Figure 2.2: Markov Chain Model of limited version of the "Cryptic" system

second die can assume values from set $W_2 = \{par, nce, mil\}$ and last die can assume values from set $W_3 = \{beg, nis, rom\}$. In addition let's assume dies are fair, or more precisely every value is present on two different sides of six-side die and at every throw each side is equiprobable. This particular model is very limited and can produce insignificantly small subset of the "Cryptic" commands, even so this model can be used as point of intuition behind decision to correlate the "Cryptic" system with MCMs in this thesis. In Figure 2.2 a visualization of this model is presented.

Usually, MCMs are represented by directed graphs; it is worth mentioning that cycles can be encountered in these graphs. Since valid commands in the "Cryptic" syntax are all of finite length representations that include cycles are highly undesirable. Outputs produced by cycles in MCM's graph could be of infinite leght or of finite length that is not valid in the context of observed language. Let's assume that we are observing a language that only allows words 'abc' and 'abca' (further denoted as L_1). Underlying MCM model (further denoted as M_1) is represented by a graph composed of three states, one corresponding to each of allowed symbols 'a', 'b', and 'c'. In scope of M_1, states form a cycle $s_a \rightarrow s_b \rightarrow s_c \rightarrow s_a$. This cycle implies that M_1 produces outputs such as 'abc','abca','abcab', up to 'abcabc…', where '…' denote infinite length string. Cosequently, strings produced by M_1 need to be tested w.r.t. constraints of the language L_1. It is clear, following the starting assumption, that only 'abc' and 'abca' satisfy costraints imposed by L_1. Following this fact it would much more desirable to convert M_1 into M_2 which would assume shape of a tree structure. Model M_2 will hold information that only words 'abc' and 'abca' are satisfying L_1's constraints, this information is held in two possible paths in the model M_2 represented by $s_s \rightarrow s_{a1} \rightarrow s_b \rightarrow s_c \rightarrow s_e$ and $s_s \rightarrow s_{a1} \rightarrow s_b \rightarrow s_c \rightarrow s_{a2} \rightarrow s_e$. We have included new states that denote start and end of the word and we have split state s_a into states s_{a1} and s_{a2} to avoid cycles.

Graph representation of MCM that corresponds to the "Cryptic" user can be broken into a tree representation following the logic in the example of the language L_1. Tree representation avoids the issue of cyclic behavior, and it is a subclass of directed acyclic graphs. In addition directed acyclic graphs are a subclass of directed graphs, ergo decision to use trees might be considered as imposing new additional constraints on MCM definition. Translation of previously described reduced the "Cryptic" model to tree representation is presented in Figure 2.3. It is worth stating that this procedure could be understood as decompression of directed graph into a tree. Tree representation, in general, has more states than corresponding directed graph, but on the other hand cycles are avoided. Repeating sequences are always of a finite length ergo cycles can be replaced by finite length sequence of state sequences. For example if we have observed that graph contains cycle $s_1 \rightarrow s_2 \rightarrow s_1$ and we know that this sequence can appear only 3 times in a row due to the "Cryptic" syntax constraints, this cycle will be translated into a following sequence $s_{11} \rightarrow s_{21} \rightarrow s_{12} \rightarrow s_{22} \rightarrow s_{13} \rightarrow s_{23}$. Another reason for moving from graph to tree representation is the fact that all commands start with

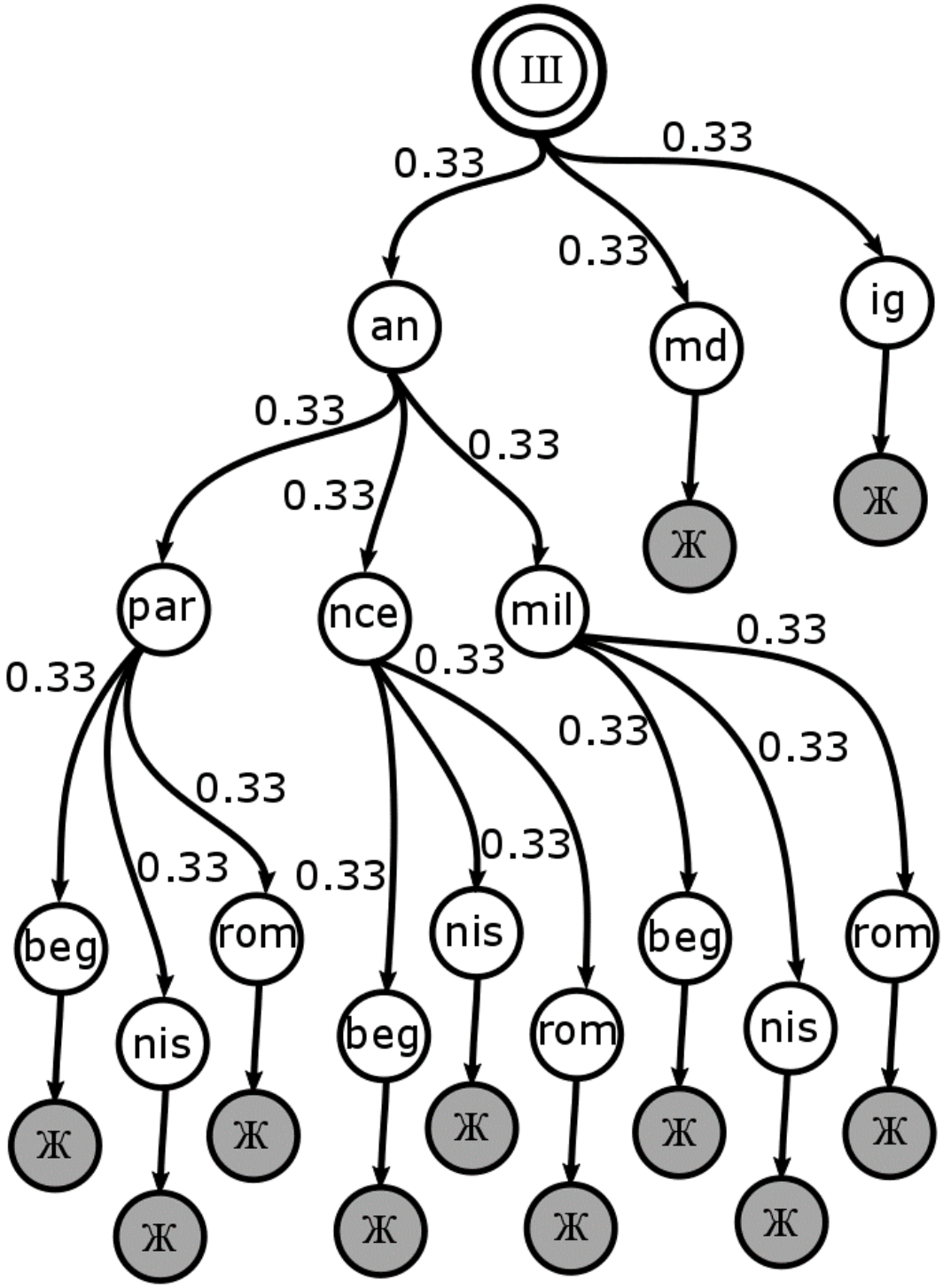

Figure 2.3: Tree representation of MCM presented in Figure 2.2

implicit symbol Ш and end with symbol Ж. In order to better illustrate transformation of MCM graph into a tree representation, let's consider the self loop with label 'ig' in Figure 2.2. In process of translating MCM graph presented in Figure 2.2 into tree representation presented in Figure 2.3 state denoted as s_1 has been split into three new states denoted as s_{an}, s_{md}, and s_{ig}. In addition two new states $s_{Ш}$ and $s_{Ж}$ have been added to represent existance of command start and end symbols. In the frame of model

presented in Figure 2.2 self-loop with label ‘ig’ had a meaning of user executing a one word command ‘ig’. In the tree model sequence $s_1 \rightarrow s_1$ has been replaced by $s_Ш \rightarrow s_{ig} \rightarrow s_Ж$. Once a state $s_Ж$ is reached user’s input is accepted and the process of accepting user’s commands is restarted. These symbols will be made explicit during the preprocess phase of the algorithm used to create tree data structure. This tree data structure is latter used to make predictions. In addition k-ary tree data structures have search complexity of $O(\log n)$, like it has been presented in [16], where n represents number of nodes in the tree and not the fan-out factor. This is highly pessimistic estimate and due to the fact that we use this structure in slightly modified way, complexity is reduced. Complexity can be further improved by adding hash maps and by sorting children of every node. More detailed analysis on the creation of tree representation will be presented in Chapter 4.

2.2 Hidden Markov models

Hidden Markov models, later referred as HMM, are models of MCM stochastic processes for which only the output of the process is observable, and the underlying structure of the process is hidden. HMM process can be understood as a black box that provides outputs at a given sample rate. As stated in [4] these processes can be observed trough sequences of outputs and from these sequences estimators can be extracted.

To cast more light on the topic of HMMs, let’s again consider example of a man M tossing two coins C_1 and C_2, full description of this example is provided in Section 2.1. This example can be extended by adding an observer O and placing a curtain between M and O. O cannot see what M is doing, only information available to O is what he hears M is saying. In this version of the experiment M is saying out loud letters ‘a’ and ‘b’ depending on the result of the toss. From O’s perspective process behind the curtain could equally likely involve M taking a ball from an urn and saying the label on the ball. This uncertainty of actual physical implementation of the process is of no significance, what is important is the underlying MCM and probabilistic transitions between MCM’s states. After spending statistically significant period of time observing the output of HMM presented in the previous example, O can built estimate of the HMM. In the frame of this discussion O will probably end up with the model of a biased coin toss with probabilities $p(a) = 0.625$ and $p(b) = 0.375$. What is considered to be statistically significant period of time is any time period in which enough observations are performed. Depending on the required precision of the estimate it

could be 100, 1000, 1000000 or any other number of tosses. What is also worth mentioning is that O should reserve a small portion of observations as test data to be able to compute how precise the estimate of HMM he produced is actually.

To place HMMs in the context of the “Cryptic” language consider the second example from the previous section, the example of a limited version of the “Cryptic” user. For a detailed description refer to Section 2.1. This example can be modified in the same way the coin toss example was, by placing an observer O and adding a curtain between M and O. For visualization of this modification consult Figure 2.4. The ongoing logic is identical as in previous case, M will speak words based on the result of a die toss, and O will be able to hear him. As stated in the coin toss example, after a statistically significant period of time, O will be able to construct estimates of ongoing HMM. Assumptions made about the actual physical embodiment of the process might not be correct but if the probabilistic attributes of such estimator tend to actual probabilistic characteristics of the process this misconception is of no significance. From HMM perspective, it is of no difference if M is tossing a die or he is taking balls from urns as long as decision states and transition probabilities are the same.

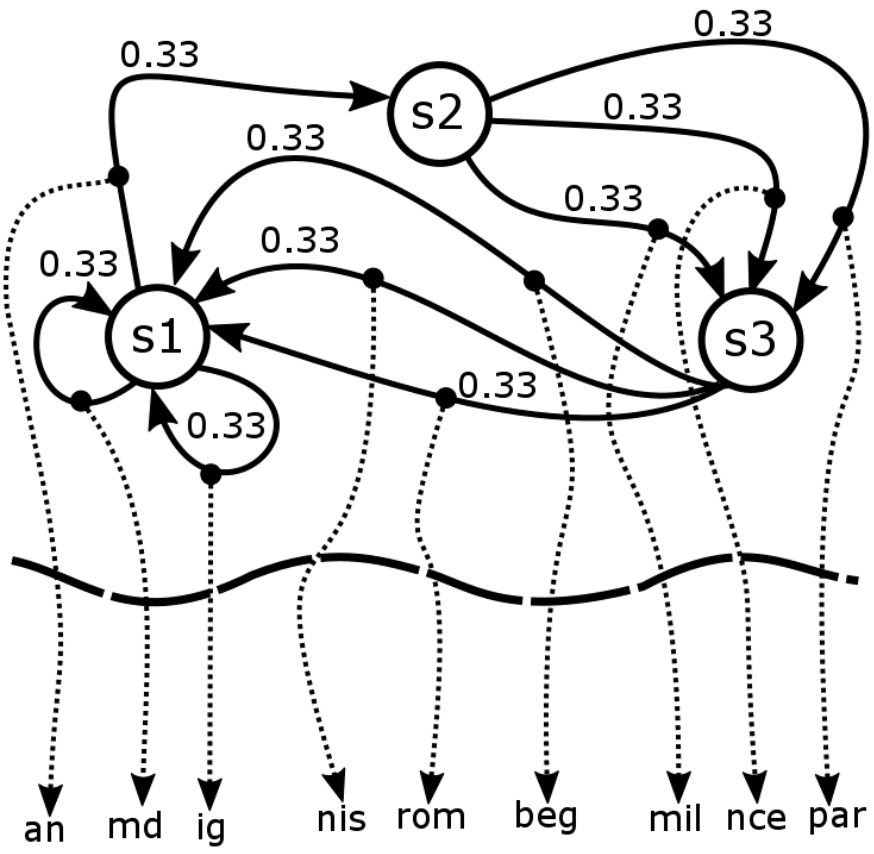

Figure 2.4: Hidden Markov Model of limited version of the “Cryptic” System

2.3 N-gram statistics

As described in [17] n-gram statistics is a model of natural languages that captures "happens before" relation among words of a language. This model can be considered as a special class of HMMs. To create a proper idea of n-gram best starting point would be the unigram statistics model. Unigram model is represented by a state set $S = \{s\}$ comprising of only one state. The state transitions are all self-arcs and their weights correspond to probability of particular outcome with no change in this probability regardless of history of previous observations. If we reflect back on the coin toss example and the approximation made by O that M is tossing a single biased coin with outcome probabilities $p(a) = 0.625$ and $p(b) = 0.375$, this approximation is actually unigram model of ongoing HMM. In this case we only have one state and two self-transitions. In this particular case it might be hard to notice difference between results obtained by actual process and by unigram estimate.

Let's observe another example, this time we will consider an example from natural language process world. Assume man M, again behind the closed curtain who is reading Shakespeare one letter at the time, and an observer O, who is able to listen what M is saying. Let's assume O is not understanding the language in which M is reading. The previous assumption restricts O from remembering that he/she might have read this particular book at one point of his/her life. Now assume the only action O can take is to count how many times each specific letter M has spoken, and to count the total of all letters spoken by M during the observations. Using these values O can extract unigram estimate of ongoing HMM process.

The first step taken by O after completing the observation phase is to produce maximum likelihood unigram estimates. These estimates are produced by simply applying following formula:

$$q_{ml}(x) = \frac{count(x)}{count(*)}. \quad [2.3]$$

In Formula 2.3 q_{ml} is the maximum likelihood n-gram estimators, n-gram estimators estimate probability of an event x occurring, count(x) is the number of observations in which event x occurred and count(*) is total number of all observations. If O applies previous formula to all the letters encountered in the observation phase, he will obtain enough information to create the unigram model of the ongoing HMM. This model is provided in Figure 2.5.

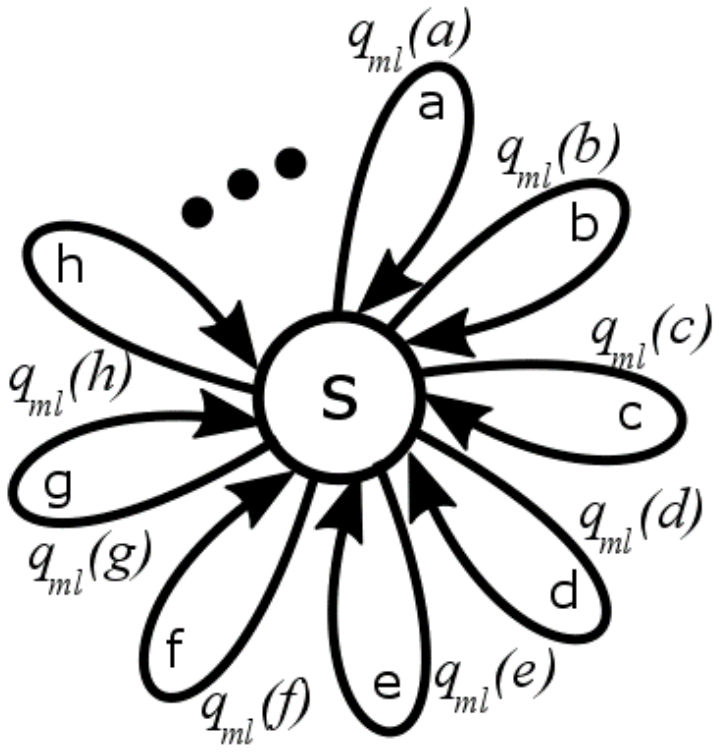

Figure 2.5: Unigram model of a man reading Shakespeare

Let's consider now how O could improve his estimator. One intuitive approach would be to assume that two successive letters are somehow correlated. Instead of just counting how many times each character occurred O should as well count how many times each pair of letters occurred. This strategy leads to the definition of bigram statistics. Bigrams are computed by applying the following formula:

$$q_{ml}(x|y) = \frac{count(yx)}{count(y)}. \quad [2.4]$$

Formula [2.4] can be understood as a fraction of cases in which given event y event x is observed. The logic behind this extension is quite straightforward. If we take for example that observation at step k is letter 'a' it is almost impossible to encounter another 'a' at step k+1, since this is the case in most of the natural languages. There are some cases (such as 'Aaron', 'bazaar', and 'laager') where we can encounter sequence 'aa', but these words appear rarely which, in fact, makes it easier to predict them. This fact implies that what we expect next letter to be is governed by what we observe at the current moment. A visualization of bigram model is presented in Figure 2.6. We can go one step further and define the 3-gram, the 4-gram, and generic case of the n-gram statistics. In case of the bigram statiscs we are interested in predicting joined appearance

of a pair of symbols, while in the case of the 3-gram statistics we are interested in predicting joined appearance of a group of three symbols. In general the n-gram statistics is computing the maximum likelihood probability estimate of a group of *n* symbols appearing together.

One could notice that this extension will lead to state space explosion since moving away from unigram to bigram model will result in increasing the number of states from one to the size of the language alphabet. The number of states increases further in case of 3-gram, 4-gram, and in general n-gram statistics. The worst case expected number of states for language alphabet of size *m* and length of n-gram statiscs equal to *n* is $O(m^n)$, which implies that we can expect large number of states for even realatively small lengths of n-gram. It comes as no surprise that most of the state of the art natural language processing systems restrict the length of n-gram to a value close to five, as it has been noted in [14-20]. Even so in the frame of this thesis discussion will not be restricted on 4-grams and 5-grams, but kept focusing on generic n-gram models.

As can be noticed in the definition of bigram, the recursive relation can be observed between bigram and unigram models. This recursive relation is maintained for any two n-gram and (n-1)-gram models. We define n-gram estimator with the following formula:

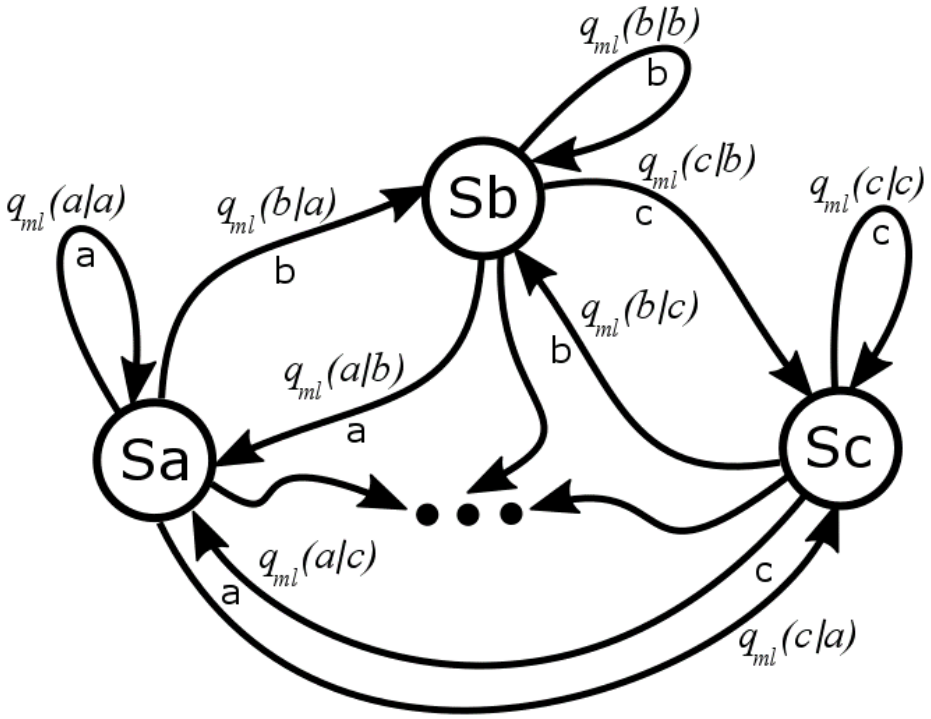

Figure 2.6: Bigram model of a man reading Shakespear

$$q_{ml}(x|Y) = \frac{count(seq(Y)x)}{count(seq(Y))}. \quad [2.5]$$

Where *Y* is a set of *n-1* predecessors of *x*, and *seq(Y)* denotes temporally ordered sequence of such predecessors. To notice the recursive relation let's consider (n-1)-gram estimator based on *Y*:

$$q_{ml}(y_{n-1}|(Y \backslash y_{n-1})) = \frac{count(seq(Y))}{count(seq(Y \backslash y_{n-1}))}. \quad [2.6]$$

Together with Formula 2.4 we obtain recursive relation:

$$q_{ml}(x|Y) = \frac{count(seq(Y)x)}{q_{ml}(y_{n-1}|(Y \backslash y_{n-1})) * count(seq(Y \backslash y_{n-1}))}. \quad [2.7]$$

Alternatively, more concisely written:

$$q_{ml}^{n} = \frac{count_n}{q_{ml}^{n-1} * count_{n-2}}. \quad [2.8]$$

This recursive behavior of n-gram estimators fortifies the choice of tree data structure since trees are recursive structures by their nature.

Another important observation to be stated about n-gram statistics is that of preventing over-fitting of train data. Term train data refers to the set of all observations of output obtained from the ongoing HMM process. An additional parameter is added to the definition of n-gram estimator, and it is used to prevent over-fitting. The discount factor beta is a value between 0 and 1, and it is used to reserve some of the probability mass for not observed cases. Not observed cases will not be assigned with probability equal to 0. Newly obtained formula for n-gram estimator is presented below:

$$q_{ml}(x|Y) = \frac{count(seq(Y)x) - \beta}{count(seq(Y))}, \qquad \beta \epsilon (0.0, 1.0), \quad [2.9]$$

$$\alpha(Y) = \sum_{\forall q_{ml}(x|Y) \neq 0} \beta, \quad [2.10]$$

$$\alpha(Y) + \sum_{\forall q_{ml}(x|Y) \neq 0} q_{ml}(x|Y) = 1.0. \quad [2.11]$$

All probability mass discarded by discounting n-gram estimators is added to α value and this way summation of all $q_{ml}(x|Y)$ estimators together with α is equal to *1.0*. This fact will be important latter when notion of entropy is defined and used to determine word separator positions inside of a command. More on this in Chapter 3.

3 WORD SEGMENTATION WITH NO LEXICON

3.1 Word segmentation and Zipf's law

Zipf's law, as already stated in the introduction section, even though it is an empirical law it describes human behavior with a surprising level of generality. This law was defined by Zipf in 1949 in his publication "Human Behaviour and the Principle of Least Effort" and represents mathematical representation of the Principle of Least Effort. This principle states that humans tend to minimize their effort in every action they take. Linguistic communication is as well an action performed by humans, and, therefore, it is expected that this action follows the Principle of Least Effort.

Mathematical formulation of Zipf's law was later extended by Mandelbrot. Mandelbrot's formulation is more general and thus it is used more often:

$$f(r) \sim \frac{1}{(r+\beta)^{\alpha}} \quad [3.1]$$

In formula [3.1] r represents rank of the word w.r.t frequency of the word in the train data while α and β are parameters of the probability distribution. The use of proportionality instead of equality is explained by the fact the usually this law assumes some normalization factor. Most often normalization factor is chosen to be equal to the value of the sum of the harmonic series, presented in Formula 1.2.

As explained in [7] Zipf's law is extensively tested and used in the English

language, in which case it behaves in a highly desirable way. It captures perfectly the fact that humans have a relatively small dictionary of words they use often. In languages that have explicit word separation, one of such is English, Zipf’s law scales well. However in languages that do not observe strict word separation, one of such is Chinese, Zipf’s law observes a decrease in precision.

We have chosen the Chinese language as a reference point as it observes some common characteristics with the “Cryptic” language. The fact that most of the words are comprised of just two or three characters and the fact that words are not separated in both “Cryptic” and Chinese made this correlation as an obvious choice. In [7] authors have conducted an extensive analysis on the applicability of the Zipf’s law in Chinese, and the results were discouraging. Many 3-gram words (three character words) were classified with errors due to the missing word separation. The most frequent 3-grams were not even words in Chinese; this fact pushes the choice in word segmentation away from the use of Zipf’s law.

The previous result could be mitigated using a train corpus that has been previously segmented and from this training data Zipf’s parameters can be extracted, and then Zipf’s distribution can be used to segment the runtime data. This fact does not bring any new quality in the case of “Cryptic” due to lack of any segmented corpus. No “Cryptic” logs that contain segmented data are available, and thus we cannot extract underlying Zipf’s distribution. Facilitating this solution requires major architectural changes in the “Cryptic” system which would be a costly solution, forcing us to search for other possible approaches to the segmentation problem.

3.2 Boundary oriented Segmentation Method

After discarding Zipf’s law as a possible tool for word segmentation, we are moving toward techniques based on mutual information. As already mentioned in the introduction, mutual information is based on the following formula:

$$I(X,Y) = \sum_{x \epsilon X} \sum_{y \epsilon Y} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right). \quad [3.2]$$

Once values of *X* and *Y* are fixed this formula simplifies into:

$$mi(x = A, y = B) = log\left(\frac{p(x = A, y = B)}{p(x = A)p(y = B)}\right). \quad [3.3]$$

This particular formula needs to be slightly modified since we assume spatial ordering of symbols is of great significance; therefore $mi(x = A, y = B)$ will be substituted by

$mi(x = A|\, y = B)$ which is defined by the following formula:

$$mi(x = A|\, y = B) = log\left(\frac{p(x = A|\, y = B)}{p(x = A)p(y = B)}\right) \quad [3.4]$$

In our case probabilities are estimated trough use of n-gram estimators denoted as $q_{ml}(x|Y)$ where x is a symbol whose probability is of interest and *Y* is a sequence of its predecessors. Put more formally *rang(x) = 1* and *rang(Y) = n*, thus order of significance of $q_{ml}(x_k|Y_k)$ is defined as $o\big(q_{ml}(x_k|Y_k)\big) = rang(x_k) + rang(Y_k)$. We value more information provided by a $q_{ml}(x_i|Y_i)$ than the information provided by $q_{ml}\big(x_j\big|Y_j\big)$ if we observe that $o\big(q_{ml}(x_i|Y_i)\big) > o(q_{ml}\big(x_j\big|Y_j\big))$ is satisfied. In addition *Y* is a spatially ordered sequence for which it is true that $Y_i \prec Y_{i+1}$, and where $\prec$ represents relation between two characters in the sequence meaning that the left operand happens immediately before the right operand in the sequence *Y*.

If we combine the definition of $q_{ml}(x|Y)$ with the definition of mutual information we can obtain an extended definition of mutual information adjusted for usage in word segmentation based on n-gram statistics.

$$mi(x = A, y = B) = log\left(\frac{q_{ml}(B[m]|A, B[m]\backslash B[m-1])}{q_{ml}(A[n]|A\backslash A[n-1])q_{ml}(B[m]\backslash B[m-1])}\right). \quad [3.5]$$

In formula [3.5] *A* and *B* are character sequences; we can understand them as potential prefix and suffix of the word which combined give a correct word of the language. Values of particular interest are the negative values of $mi(x, y)$ these values indicate that x and y tend to be separated by a word separator symbol more often than to appear grouped together inside of a word. The reason to extend character based mutual information to string based mutual information comes from the fact that "Cryptic" words do not follow rules of natural language words. One of such rules is the common word root. Some words from the English language (e.g. 'informal' and 'formal') share common root. In scope of the "Cryptic" language we can encounter words of only limited length, on average close to three characters. If we observe words 'jun' and 'jul', which are valid words in the "Cryptic" language, we could conclude that the common root is 'ju'. This conclusion would be incorrect since words in the "Cryptic" language do not share roots because they are generated as compressed strings. Based on this fact we can conclude that character based mutual information cannot be applied in its original form in the case of the "Cryptic" language. On average all character pairs are almost equiprobable therefore character based mutual information mean value would be

close to 0, meaning we would not be able to decide on word barriers.

An interesting idea is presented in [8], and it is denoted as a boundary condition function. Boundary condition function is based on multiple mutual information values, and it presents an extension of the previous discussion. In this case, we observe a concrete position in the string, and the decision to place a word separator is made based on n-gram mutual information. Let's define a string as a sequence of characters:

$$s = c_1 c_2 \dots c_i c_{i+1} \dots c_{n-1} c_n. \quad [3.6]$$

Now the boundary confidence function is defined as following:

$$BC_{min}(L|R) = min\{mi(c_{i-1}, c_i),$$
$$mi(c_{i-2}c_{i-1}, c_i),$$
$$mi(c_{i-1}, c_i c_{i+1}),$$
$$mi(c_{i-2}c_{i-1}, c_i c_{i+1})\}. \quad [3.7]$$

In formula [3.7] | represents a word separator position, and L and R represent left and right substrings of the initial string that are results of placing word separator in specified position.

In our solution, we propose a modification of this approach. Our proposal is parameterized, instead of using a fixed number of comprising factors $k = 4$, we allow that value k is a variable. Another modification applied in our solution is the change of the min{} operator with the $avg\{\}$ operator, which is more resilient to local contextual minima. Indeed, if the min{} operator is chosen it may happen that the decision is based on the mutual information between two individual characters which bares the least contextual information while all other mutual information values might be indicating not to place the separator in this position. Using the $avg\{\}$ operator this issue is avoided; in addition, this approach can be further generalized by applying a weighted average, in this thesis denoted as $wavg\{w[n], v[n]\}$ operator, where $w[n]$ represents the list of weights corresponding to a list of values denoted as $v[n]$.

Usage of $wavg\{w[n], v[n]\}$ allows the application of various weighting strategies, the most obvious one being the strategy which defines weights based on the size of the contexts, defined as:

$$w_i = \frac{log\big(length(L) + length(R)\big)}{N_f}, \quad [3.8]$$

$$N_f = \sum_{\forall(L,R)} log\big(length(L) + length(R)\big). \quad [3.9]$$

Where log() represents logarithmic function with base equal to $e \approx 2.78$. We can notice one possible undesirable behaviour when summation is used to combine strengths of left and right contexts. Consider following equations:

$$length(L_k) < length(R_k) \wedge lenght(L_k) + length(R_k) = n, \quad [3.10]$$

$$length\big(L_g\big) = length\big(R_g\big) \wedge lenght\big(L_g\big) + length\big(R_g\big) = n. \quad [3.11]$$

In this case, these two factors are weighted with the same value, while it is noticeable that the second factor is more stable than the first one. Stability of the contexts in Fromula 3.11 comes from the contextual information being equally balanced between *L* and *R*. To avoid this issue the weight formula can be modified in the following manner:

$$w_i = log\big(lenth(L)\big) * \frac{log\big(leght(R)\big)}{N_f}, \quad [3.12]$$

$$N_f = \sum_{\forall(L,R)} log\big(length(L) * length(R)\big). \quad [3.13]$$

In this way we firstly average the lengths of both contexts and then we weight the comprising factors, obtaining almost a linear progression if $length(L) = length(R)$ and a logarithmic progression if $length(L) \neq length(R)$, while maximal values are observed in the portion of the space where $length(L) \approx length(R)$. A complete plot of this weight function is provided in Figure 3.1.

Another possible weighting function is the so-called exponential weighting, usually used in the Exponential Moving Average processes in the control theory as explained in [8]. We can observe comprising factors in the boundary condition as values of a signal at different sample times and then use Exponential Average, later referred as EA, to extract final indicator. EA is defined as follows:

$$EA\{x_1 x_2 \dots x_n\} = (1-\alpha)^{n-1} x_1 + \alpha \sum_{s=0}^{n-2} (1-\alpha)^s x_{n-s}. \quad [3.14]$$

In our case, x sequence is ordered in increasing order of values obtained based on formula [3.12], or more precisely:

$$x_i = log\big(lenth(L)\big) * log\big(leght(R)\big). \quad [3.15]$$

As already have been explained, values obtained from this formula prioritize balanced contexts w.r.t. to imbalanced ones. Exponential moving average processes behave extremely desirable in highly volatile environments, meaning that it gives higher weights to more related terms and it discriminates to less related terms. Consider an example of the stock market, which is indeed highly volatile environment. In the case of the stock market Exponential moving average processes consider temporarly close events to be more related one to another and they are given higher weights. The “Cryptic” user can also be considered as an exceedingly unpredictable system. The commands produced by the user are governed by many factors, such as the current economic situation in the region he operates in, desires of his clients, changes in the set of his clients, promotions, and many other. In this case, Exponential moving average processes would cosider contextualy close events to be more related. Additionally, in [9] it has been shown that this strategy in producing the mean estimators, which average value is by its intended usage, gives highly stable results w.r.t. simple arithmetic or geometric average.

Just for the sake of completeness we provide here the definition of geometric average, since in [9] it has been shown that geometric and arithmetic average behave

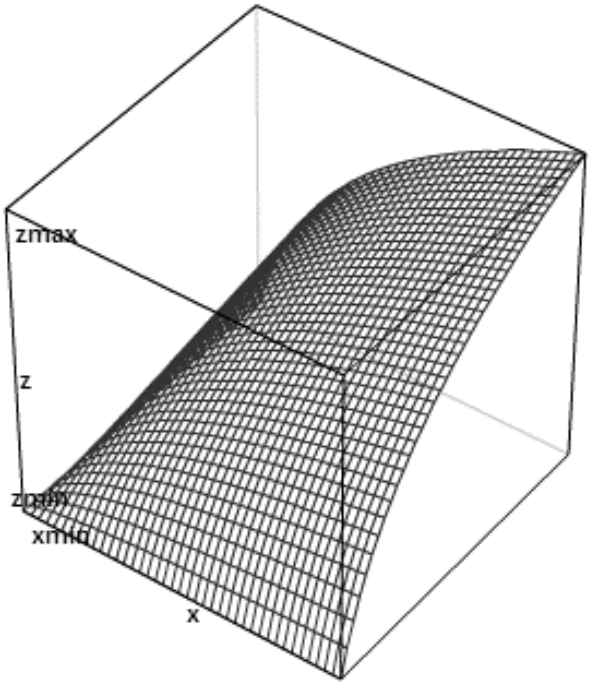

Figure 3.1: Plot of the function $f = \log\big(lenth(L)\big) * \log(leght(R))$

comparably in highly fluctuating environments:

$$GA\{x_1x_2 \dots x_n\} = \sqrt[n]{\prod_{i=1}^{n} x_i}\ . \quad [3.16]$$

After this exhaustive analysis of the possible modifications to $BC_{min}(L|R)$ we propose a new criterion denoted as $BC_{wa}(L|R; n, m, w[k])$, where subscript 'wa' denotes weighted average and $w[k]$ represents a list of weights pre-computed in one of the possible ways (here we have explained just few possibilities, other strategies may be possible). The complete procedure for producing this segmentation criteria is presented below:

$$(L, R) = comb(s; k, g), \quad [3.17]$$

$$x_i = log\big(length(L[i]) * length(R[i])\big), \quad [3.18]$$

$$x[k] = sort_{inc}(x[k]), \quad [3.19]$$

$$L[k] = sort_{inc}(L[k]; x[k]), \quad [3.20]$$

$$R[k] = sort_{inc}(R[k]; x[k]), \quad [3.21]$$

$$EA\{L|R; x_1x_2 \dots x_n\} = (1-\alpha)^1 * mi(L[n-1], R[n-1]), \quad [3.22]$$

$$+\alpha \sum_{s=0}^{n-2} (1-\alpha)^{n-s} * mi(L[n-s], R[n-s]), \quad [3.23]$$

$$BC_{wa}(L|R; n, m, w[k]) = EA\{L|R; x_1x_2 \dots x_n\}, \quad [3.24]$$

$L[k], R[k], x[k]$ should be regarded as fixed triplets, while *L* and *R* represent arrays of all possible combinations of context whose combined length are equal to *k*, and x represent corresponding sorting criterion, since EA is sensitive to sorting order. In addition $comb()$ produces all possible context combinations up to a combined length equal to *k*, extracted form the string *s.* Position *g* represents a hypothesis that the word separator will be placed at this position in the string *s*. Detailed visualization of this procedure is provided in Figure 3.2. In Formula 3.18 we apply previously defined contextually balance logarithmic function. We proced on by sorting triplets $(L[k], R[k], x[k])$ in Formulas 3.19-3.21. This sorting can be understood as sorting with respect to the joined contextual information of terms $L[k]$ and $R[k]$. Finally in Formula 3.23 we compute the Exponential average of each triplet while prioritizing longer and balanced contexts.

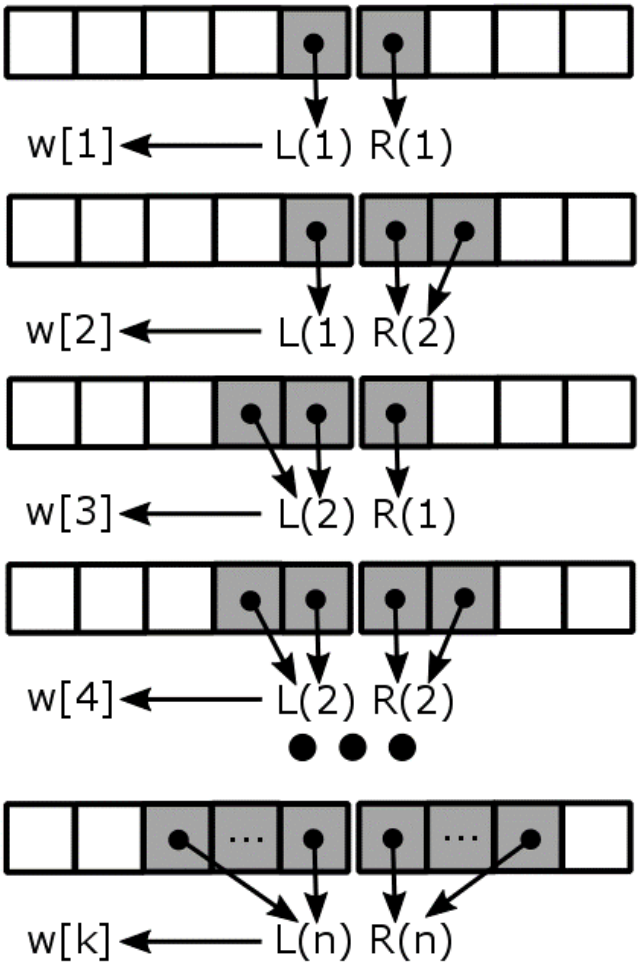

Figure 3.2: Process of incrementally increasing left and right context

We use $BC_{wa}(L|R;n,m,w[k])$ as follows:

$$if\ BC_{wa}(L|R;n,m,w[k])\epsilon[-1,1]\ \ then\ we\ cant\ decide, criterion\ is\ ignored$$

$$if\ BC_{wa}(L|R;n,m,w[k]) > 1\ then\ no\ separator\ is\ placed$$

$$if\ BC_{wa}(L|R;n,m,w[k]) < -1\ then\ separator\ is\ placed$$

The logic behind using this criterion is identical to the usage of $BC_{min}(L|R)$ described in [3]. Actually $BC_{min}(L|R)$ is a restriction of $BC_{wa}(L|R;n,m,w[k])$, if we set weight in such way that:

$$k = index(min\{mi(L[j],R[j])\}), \quad [3.25]$$

$$x[i] = \begin{cases} 0, i \neq k \\ 1, i = k \end{cases}, \quad [3.26]$$

$$\alpha = 0, \quad [3.27]$$

$$k = 2. \quad [3.28]$$

$BC_{wa}(L|R; n, m, w[k])$ will produce identical results as $BC_{min}(L|R)$; in our system we use more general criterion provided by $BC_{wa}(L|R; n, m, w[k])$.

3.3 Entropy-oriented segmentation method

Another possible approach to word segmentation without an available dictionary is based on the entropy values of individual characters in a sequence. It has been shown in [10] that characters belonging to a word observe a decrease in the entropy value as more context is present. Here entropy value corresponds to the probability distributions of possible successors of a particular character. Therefore, entropy should be considered as a monotonically decreasing function inside of the word boundaries. An illustration of an example taken from the English language is presented in Figure 3.3.

The starting point to present this idea in more formal way is the definition of conditional entropy:

$$H(X|X_n) = -\sum_{x_n \epsilon \chi_n} p(X_n = x_n)$$

$$* \sum_{x \epsilon \chi} p(X = x|X_n = x_n)\ log\big(p(X = x|X_n = x_n)\big). \quad [3.29]$$

Where χ represents the set of elements, in our case the set of characters and χ_n represents n-gram sequences produced from χ. If the value of interest is conditional entropy for a given fixed n-gram sequence, or in other words, for a given fixed context, the previous formula simplifies into:

$$H(X|X_n = x_n) = -\sum_{x \epsilon \chi} p(X = x|X_n = x_n)\, log\big(p(X = x|X_n = x_n)\big). \quad [3.30]$$

Based on the previous observation which states that entropy function is monotonically decreasing while inside of the word boundaries, can be applied to conditional entropy as well. Stated in more formal way, if we have a n-gram sequence x_n of length n and another n-gram sequence of length $n + 1$ denoted as x_{n+1} while x_n is a prefix of x_{n+1}, the following condition holds in most common cases:

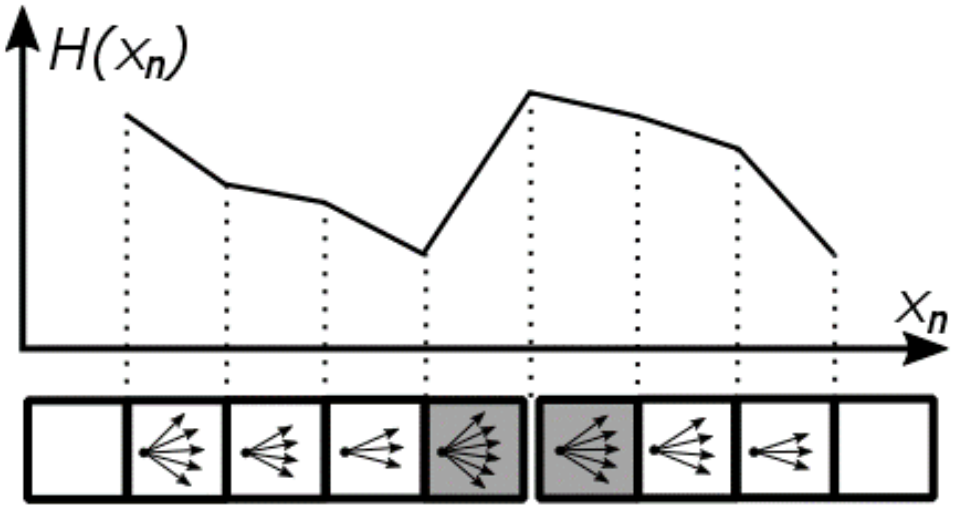

Figure 3.3: Principle of entropy monotonicity inside of word boundaries

$$H(X|X_n = x_n) > H(X|X_n = x_{n+1}). \quad [3.31]$$

The cases in which this condition does not hold are usually words with same roots where we can observe an increase in entropy if a large number of words share the same root in the training corpus. Many words sharing the same root is not an alarming issue since, from the “Cryptic” language perspective, the majority of words are made from two to three characters and thus speaking of word’s root holds no significance. Cases in which this does occur in the “Cryptic” language may be the first or second names of a passenger, however, as previously have been stated, these are one time words and usually have no statistical significance. Moreover, the intention of using n-gram statistics in the scope of this system is for prediction and correction. From auto-completion and auto-correction perspective, it is obvious that if the decision is reached in two steps instead of one it does not produce any complications. These considerations fortify the fact that perfect word segmentation is not needed, but it is desirable. It is worth noticing that, too liberal word segmentation, meaning too many missing separators, may lead to issues. On the other hand, too aggressive segmentation, meaning there are false positive separators, even though not optimal, is not creating any major issue.

In the original version of the criterion (proposed in [10]) word boundaries are decided based on complete n-gram for every sentence; in our case sentence is equal to a command. This criterion, as previously have been stated, is based on the monotonicity

of conditional entropy; indeed, conditional entropy is monotonically decreasing while inside of word boundaries. This fact implies that if we observe an entropy increase between two consecutive characters, or more precisely between two n-gram sequences x_n and x_{n+1} where x_n is a prefix of x_{n+1}, this observation indicates the position of a word separator. Visualization of this principle is presented in Figure 3.3.

In our solution, we use both this initial version and our modification of this approach. The proposed modification is based on the notion of sliding context; we average conditional entropy value based on all possible n-gram sequence of length up to k which is a parameter of the criterion. Averaging approach follows the same logic as already explained in Section 3.2. It is extremely similar to the $BC_{wa}(L|R;n,m,w[k])$ with a difference in the fact that in this case the decision is based on monotonicity of entropy while in case of $BC_{wa}(L|R;n,m,w[k])$ the decision was based on mutual information. We formally define this criteria as follows:

$$y_i = log\big(length(x_i)\big), \quad [3.31]$$

$$y = sort_{inc}(y), \quad [3.32]$$

$$x = sort_{inc}(y;x), \quad [3.33]$$

$$EA\{x_1x_2 \dots x_n\} = (1-\alpha)^n * H(X|X_n = x_n)$$

$$+\alpha \sum_{s=1}^{n-1} (1-\alpha)^s * H(X|X_n = x_{n-s}), \quad [3.34]$$

$$EC_{wa}(k;w[k]) = EA\{x_1x_2 \dots x_n\}, \quad [3.35]$$

where $EC_{wa}(k;w[k])$ denotes Entropy Condition with weighted average for maximum length of context equal to k. A visualization of the application of this criteria is presented in Figure 3.4. This approach takes into account all possible context up to length of k then averages among these context to get an indicator of existence or non-existence of separator in the position at hand. For example, let's assume we are interested in finding out whether there is a boundary or not after sequence 'anpar' in "Cryptic" command. In this case we compute conditional entropy for contexts 'anpar', 'npar','par', 'ar', and 'r' while giving them weights which prioritize longer contexts. Strategies for weighting contexts are dealt with in more details in Section 3.2. This way we allow dispersion of decision criteria throughout complete corpus. We are using weighted average as operator to avoid one particular context's dominance. The only case in which one context can dominate the weighted average is if it is the longest

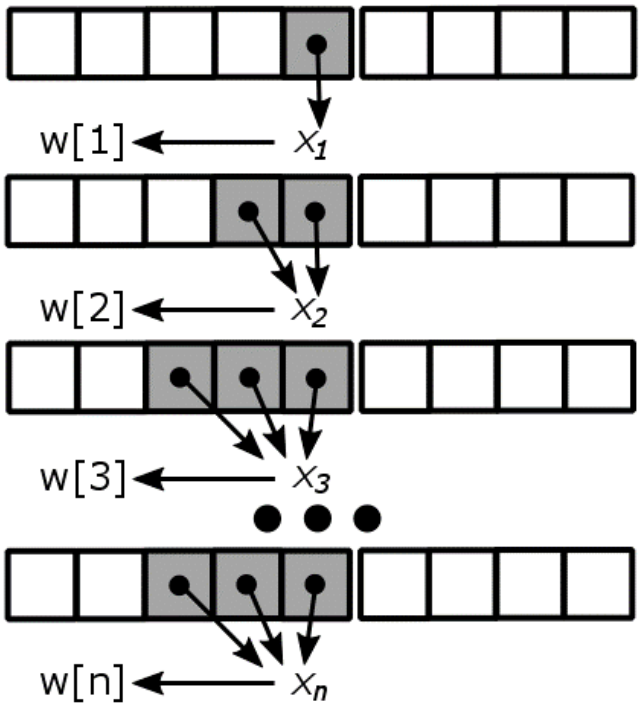

Figure 3.4: Process of incrementally increasing context used for entropy computation

context and its value is by far greater than the other values. In that case enough information is presented in the longest context, so it is desirable that this factor affects decision.

3.4 Majority voting boundary decision

Now that both $BC_{wa}(L|R;n,m,w[k])$ and $EC_{wa}(k;w[k])$ have been defined we can propose the final definition of the word boundary decision criterion. In our solution the decision criterion is based on majority voting among many individual criteria. All individual criteria are based either on $BC_{wa}(L|R;n,m,w[k])$ or $EC_{wa}(k;w[k])$ or in the simplest case just on the value of $H(X|X_n = x_n)$. At current state of the solution the decision is based on the following individual criteria:

- $EC_{max}(k)$
- $EC_{min}(k)$
- $EC_{wa}(k;w[k])$
- $EC_{ea}(k;w[k])$

- $EC_{bwa}(k; w[k])$
- $EC_{bea}(k; w[k])$
- $H(X|X_n = x_n)$
- $H_b(X|X_n = x_n)$
- $BC_{wa}(L|R; n, m, w[k])$
- $BC_{ea}(L|R; n, m, w[k])$

Each individual indicator participates in the voting procedure by giving one vote point, once all points are collected for a particular position a decision is made by the following rule:

$$if\ v[i] \geq \frac{n}{2}\ then\ place\ a\ separator\ on\ position\ i$$

$$else\ no\ separator\ on\ position\ i$$

$$n = number\ of\ indicators$$

$$i\ \epsilon[0, length(command)]$$

Another possible approach is that instead of equal voting procedure we use priority based voting procedure. In Chapter 7, which deals with validation and performance, we have conducted performance measurements of each segmentation criteria and ranked them in order of decreasing performance. Once ranking of individual criteria is obtained ranked voting gives increasing vote power with increasing rank. This way we prioritize decisions made by more precise criterion while two or more combined criteria can overpower decision of higher rank criterion. All other features of voting are the same; we still need more than half total votes to make a decision of placing word separator.

To finalize the discussion on word segmentation, we will explain the notation used for listing all individual indicators and explain the meaning of each one of them. $EC_{min}(k)$ and $EC_{max}(k)$ represent simplifications of weighted average approach to max and min operators, as previously stated in Section 3.2., this simplification can be understood as setting all weights to 0 and for min/max setting the weigh to 1.

$EC_{wa}(k; w[k])$ and $EC_{ea}(k; w[k])$ represent weighted average based on logarithmic length of n-gram sequence and exponential average based on logarithmic length of n-gram sequence respectively, as explained in Section 3.3. While $EC_{bwa}(k; w[k])$ and $EC_{bea}(k; w[k])$ represent same indicators just applied on reversed

data (letter b comes from the word backward). Reversed context may as well bring interesting new information in some cases where words appear only toward the end of the commands in these cases forward context bring less information than backward indicators since more probability mass is contained in backward indicator therefor entropy indicator is more stable.

On the other hand $BC_{wa}(L|R;n,m,w[k])$ and $BC_{ea}(L|R;n,m,w[k])$ represent indicators based on mutual information function as explained in Section 3.2. These two indicators do not have backward versions since initially they take into account both pre-context and post-context, reversing the data will lead to the more or less identical results and therefore bias the voting procedure, since almost the same information will be counted twice.

Lastly, we have two indicators based on full n-gram of every command and the entropy monotonicity extracted from this n-gram. These two indicators are denoted as $H(X|X_n = x_n)$ ans $H_b(X|X_n = x_n)$ corresponding to forward and backward train data respectively. Procedures for extracting forward and backward indicators do not differ in any aspect, only difference is observed in data pre-processing when train data is reversed and given to the same algorithm twice, which will produce two different indicators from two different data samples.

4 N-GRAM TREE DATA STRUCTURE

After the notions of n-gram statistics and word segmentation have been established, we can place these concepts in the frame of the "Cryptic" language. Starting point should be the command start symbol denoted as Ш. Train data provided is in the form of user logs of the "Cryptic" command line system. Every line in the log represents one correct command issued at a particular time by a user U. It is important to stress out that incorrect commands are discarded in the acquisition of data logs since incorrect commands can bias predictors toward suggesting the user to input errors which is the undesired behavior. Symbol Ш is an implicit symbol that represents the start of every command. This particular symbol is important since it can be considered as a root of a tree data structure that will capture n-gram relations between symbols in the U's data logs.

Commands in the logs can be considered as symbol sequences whose probability is defined as:

$$P(s_i \epsilon S) = p(s_i|Ш, s_1, s_2, \dots, s_{i-1}), \quad [4.1]$$

where S denotes a symbol sequence that represents a valid command in the "Cryptic" syntax, s_i denotes any symbol in this sequence and Ш is the beginning of any command. If we compare previously stated probability measure with n-gram probability measure:

$$P_n(s_i \epsilon S) = p(s_i | s_{i-n}, s_{i-n+1}, \dots, s_{i-1}). \quad [4.2]$$

It is obvious that n-gram probability measure is just fixed history length restriction of the probability measure presented in Formula 4.1.

In this thesis we propose various types of n-gram statistics based tree structures that are used for different purposes:

1) Character level n-gram tree with fixed root symbol, root symbol is Ш
2) Character level n-gram tree with context-free root, root symbol is Ц
3) Word level n-gram tree
4) Command level n-gram tree

Case 1 is constructed based on Formula 4.1, while cases 2, 3 and 4 are constructed based on Formula 4.2. Firstly let’s concentrate on shared attributes of these four types of trees and then the discussion will move on describing particularities of each individual type.

The construct of a node is the same in all cases. Every node in a tree has as the attribute value of n-gram probability estimator. Based on the depth of the node position in the tree we can deduce the length of the context, for the node at depth k we have k-2 meaningful predecessors since the root bares no contextual information. In addition to n-gram probability estimator, every node contains as well a total probability estimator. This estimator takes into account the probability of a complete path that leads to this concrete node. Another two paramount values are entropy value and alpha value. Entropy value is computed w.r.t. to Shanon’s entropy definition and it is explained in details in Chapter 3. Alpha value contains ‘missing’ probability mass for every node in the tree, missing mass refers to the portion of probability mass discarded with discounting n-gram estimators defined in details in Section 2.3.

What is important from the perspective of the tree structure is the organization of node’s children. To facilitate fast maximum likelihood search, children of every node are ordered with respect to the probability estimate value, or more precisely to the value of n-gram estimators. In addition to this ordering, every node contains a hash map of its children, this extension provides constant time search in the tree for any random path.

As we have previously established, every command has implicit start symbol denoted as Ш. In this case the root of the n-gram tree data structure will be labeled with Ш symbol, and this will be the starting point in the construction of the data structure.

The estimator of the root probability will be set to 1.0 since there is no uncertainty of Ш appearing in the command. Once root node is formed, the first level of n-gram tree is computed. The first level of n-gram tree represents probability estimates of the first characters in the command. Nodes from the first level of the tree can be considered as bigram estimators:

$$p_2(s) \cong q_{ml2}(s) = \frac{count(Шs)}{count(Ш)}. \quad [4.3]$$

Every further level of the tree represents estimation of probability of the i^{th} symbol in the command symbol sequence:

$$p_2(s_i|S) \cong q_{ml2}(s_i|S) = \frac{count(ШSs_i)}{count(ШS)}. \quad [4.4]$$

Where S denotes prefix string of length *i* and s_i denotes symbol at the position *i*. This concept can be considered as a spatial restriction of n-gram statistics, where n-gram of history length *i* is taken into account only for i^{th} character.

One important part of the tree creation algorithm is that every node contains an array with occurrence positions in the train data. Using the occurrence array enables traversing the train data in more economic way; train data is not traversed completely for every node, just the parts of train data that satisfy provided context are visited.

Another approach with same complexity is that of FP trees construction algorithm. As stated in [21], every sequence encountered in train data is added as a path in FP tree. If nodes corresponding to n leftmost symbols in the symbol sequence are already present in the tree, their counters are incremented. After prefix part of the sequence is processed; at (n+1)th node new branch will be created and the rest of the sequence will result in newly created nodes with counters set to 1. The approach applied in this thesis is not significantly different since it will result in the same tree, the difference is that FP tree approach is depth-first creation, while approach applied in our solution is breadth-first approach.

Why the decision made was in favor of breadth first creation strategy is reflected in the fact that entropy values computed at each node requires all children's parameters to be available, as well as the alpha values. With depth-first creation strategy, we would firstly need to create the complete tree, then run the algorithm for computing the parameters of the fully expanded tree. Asymptotically, number of operations is the same in both cases; in our implementation we have a smaller constant

multiplying the complexity. Our approach results in faster runtime, but also in more memory consumption, i.e., memory allocated to occurrence positions of each n-gram in the training corpus. In this particular implementation we are more concerned with running time than with memory consumption, due to the fact that the “Cryptic” system is run on state of the art servers, and therefore memory restriction is of less significance.

As previously stated in Section 2.3 n-grams are usually used for values of n close to 4 or 5, here we are not restraining the value of n, and the context is explored till maximal depth. In this case value of *n* goes until the value of command’s length, or more precisely until symbol Ж, is encountered. This particular tree is not used at runtime nor for actual predictions in auto-complete nor auto-correct capabilities. This tree is used in word segmentation procedures, one of segmentation criteria is based on fully expanded n-gram tree and entropy values, this particular word boundary criteria is explained in more details in Section 3.3. Visualization of such tree structure is provided in Figure 4.1.

One important thing to be added is that discounting factor β and the α value affect entropy value of each node. As previously stated in Section 2.3 n-gram estimators are computed with discount factor β and in accordance with the following formula:

$$q_{ml}(x|Y) = \frac{count(seq(Y)x) - \beta}{count\big(seq(Y)\big)}, \qquad \beta\epsilon(0.0,1.0), \quad [4.5]$$

$$\alpha(Y) = \sum_{\forall q_{ml}(x|Y)\neq 0} \beta, \quad [4.6]$$

$$\alpha(Y) + \sum_{\forall q_{ml}(x|Y)\neq 0} q_{ml}(x|Y) = 1.0. \quad [4.7]$$

From equations [4.5-4.7] it becomes obvious that small portion of probability mass is not assigned to any of the children, but it is reserved for generalization purpose, or more precisely it is assumed to be assigned to cases not encountered in the training corpus. This probability mass is accumulated in the α value. α value can be understood as an additional implicit child of every node which affects value of the entropy for each node. This fact implies that formulae provided for $H(X|X_n = x_n)$ and $mi(x = A, y = B)$ need to be modified to take into account this missing probability mass.

Firstly, let’s consider $H(X|X_n = x_n)$ since it is a more simple case, this value is computed in accordance with the following formula:

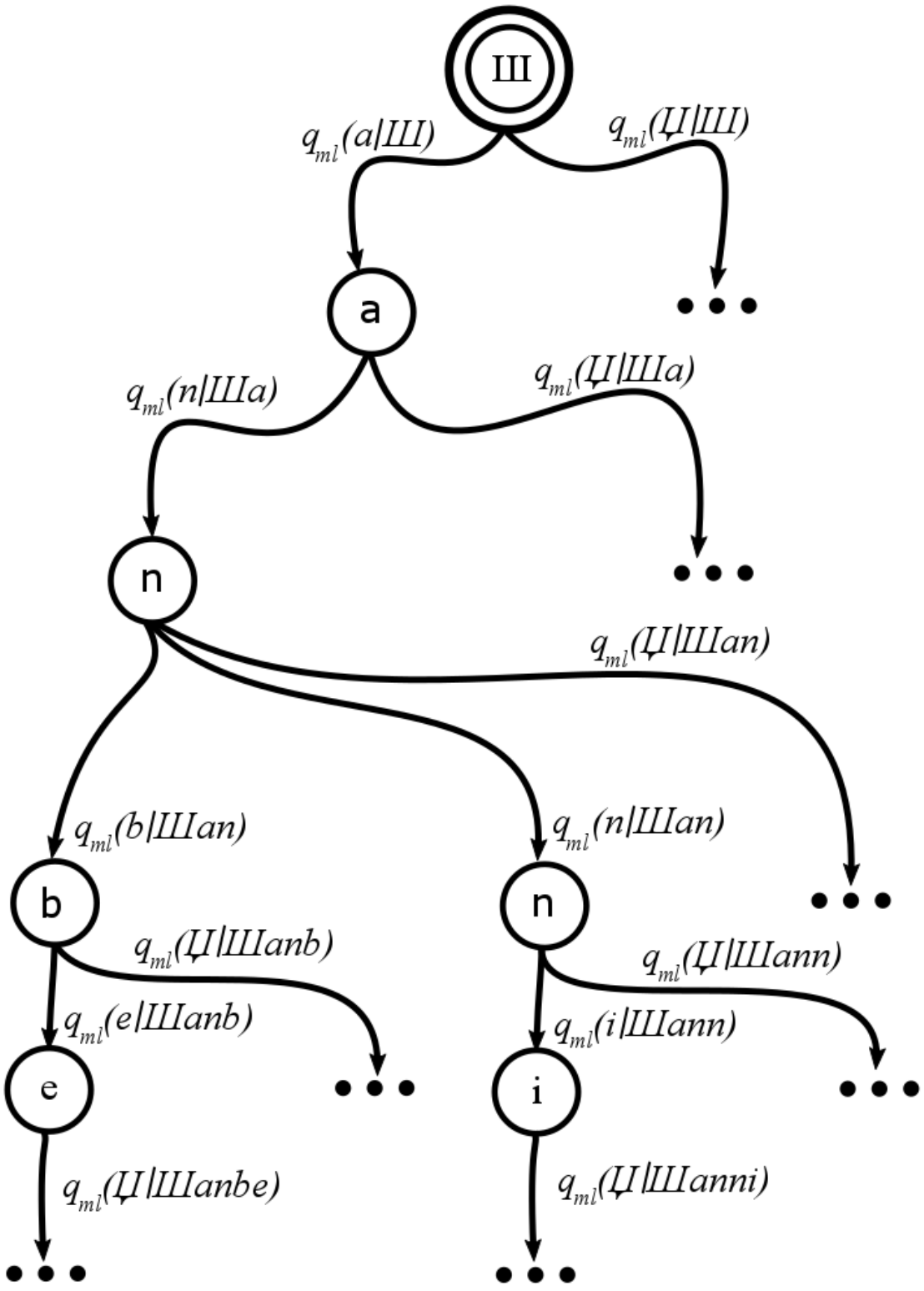

Figure 4.1: Example of character level n-gram tree with fixed root symbol set to III

$$H(X|X_n = x_n) = -\sum_{x \epsilon \chi} p(X = x|X_n = x_n)\, log\big(p(X = x|X_n = x_n)\big). \quad [4.8]$$

This particular formula, when applied onto n-gram tree structure, assumes that χ represents the set of the children of the concrete node for which the entropy value is computed. In addition it assumes that the context denoted as x_n represents full path till the root of the tree. To facilitate easy search of ancestral paths backward pointers toward the node's parent is added in the node construction. What is clear in this situation is that computed entropy value is not complete since the α value is missing from the entropy; proposed modification is simple, and it is presented in the following formula:

$$H(X|X_n = x_n) = -\alpha(X_n = x_n)\, log\big(\alpha(X_n = x_n)\big)$$

$$-\sum_{x \epsilon \chi} p(X = x|X_n = x_n)\, log\big(p(X = x|X_n = x_n)\big)$$

$$= -\alpha(X_n = x_n)\, log\big(\alpha(X_n = x_n)\big)$$

$$-\sum_{x \epsilon \chi} q_{ml}(X = x|X_n = x_n)\, log\big(q_{ml}(X = x|X_n = x_n)\big). \quad [4.9]$$

Where $\alpha(X_n = x_n)$ univocally denotes each individual node by its full path in the tree. This is a small modification, but extremely important due to possibility of applying tree pruning w.r.t. the count value or w.r.t. the confidence level (n-gram probability estimate). In that case $\alpha(X_n = x_n)$ will hold probability mass of the pruned tree branches.

The mutual information function, denoted as $mi(x = A|\, y = B)$, also needs a revision due to the presence of $\alpha(X_n = x_n)$. The initial formulation of the mutual information function is stated as follows:

$$mi(x = A|\, y = B) = log\left(\frac{p(x = A|\, y = B)}{p(x = A)p(y = B)}\right). \quad [4.10]$$

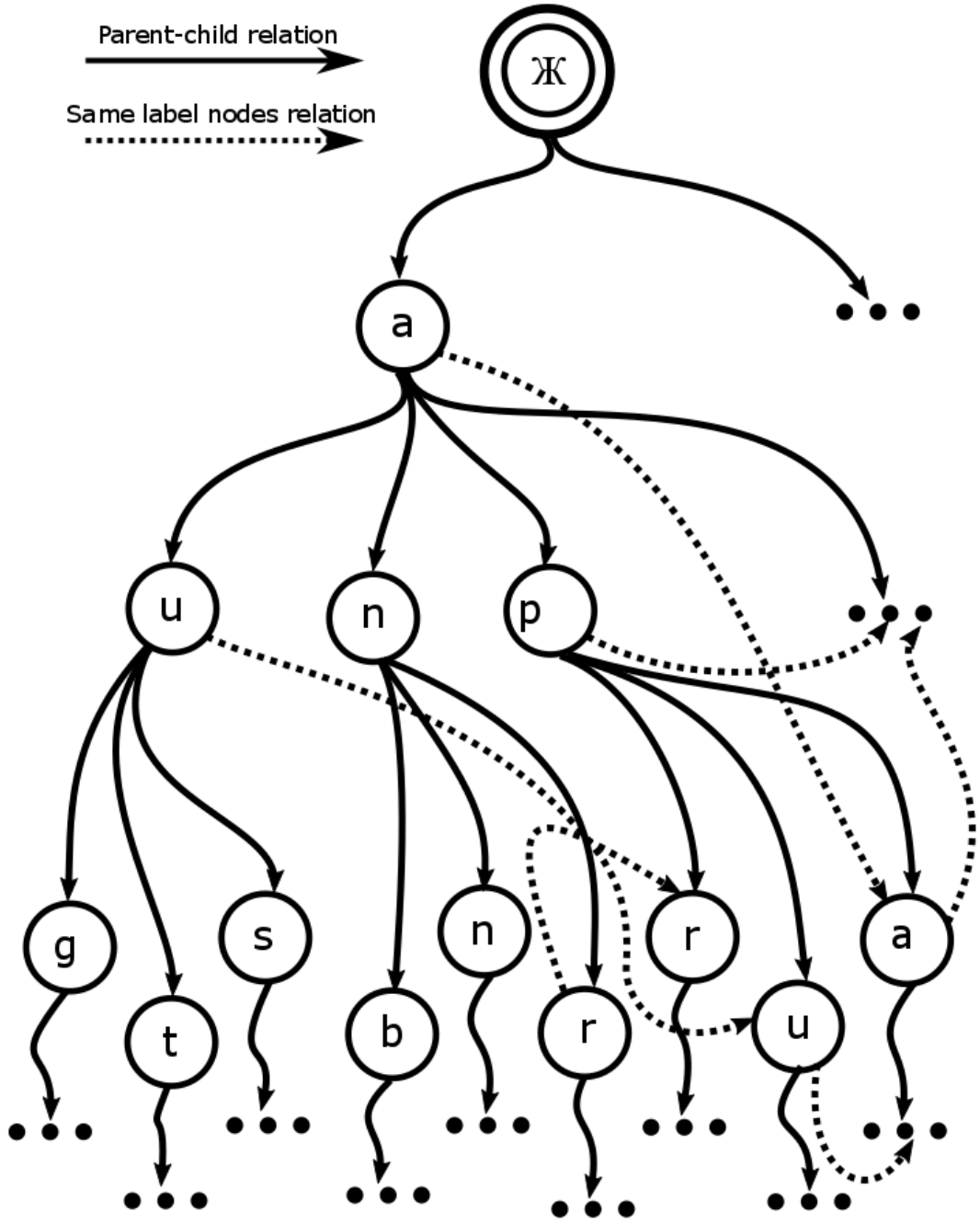

Figure 4.2: Visualization of additional linkage between nodes with the same label

One can notice that this particular definition is based on total probabilities $p(x = A, y = B)$, $p(x = A)$ and $p(y = B)$, which are not available in our particular data structure. To avoid this issue our data structure is enhanced by horizontal pointers that connect all nodes with the same label, visualization of this enhancement is presented in Figure 4.2. What is extremely important to be stated is that total probability of any particular symbol is affected by the missing probability mass in the tree, or more precisely by every $\alpha(x \epsilon x_n)$. The total probability of any particular symbol is then defined by following formalism:

$$\varphi(x) = \{x_n | x = x_n[n]\}, \quad [4.11]$$

$$\delta(x_n) = \sum_{x_i \epsilon x_n} \left(\frac{p(x_i)\alpha(x_i)}{k - h(x_i)} \right)^{n-i}, \quad [4.12]$$

$$k = sizeOf(alphabet), \quad [4.13]$$

$$h(x_n) = sizeOf\big(C(x_n[n])\big), \quad [4.14]$$

$$g(x = A) = \max_{x_n \epsilon \varphi(A)} (p(X = x_n) + \delta(x_n)),$$

$$= \max_{x_n \epsilon \varphi(A)} \left[\prod_{x_i \epsilon x_n} q_{ml}(X = x_i) + \sum_{x_i \epsilon x_n} \left(\frac{\alpha(x_i) \prod_{x_j \epsilon x_i} q_{ml}(X = x_j)}{k - h(x_i)} \right)^{n-i} \right]. \quad [4.15]$$

Where $C(x_n[n])$ denotes function that returns the set of children whose parent is the last node in the sequence x_n, and $x_n[n]$ denotes a last element of the sequence. By applying this formalism $\alpha(x_n)$ value is equally divided among all not encountered symbols. If a particular symbol of interest is not found in node's children then that node's $\alpha(x_n)$ value portion corresponding to the symbol of interest is added to total probability of this particular symbol. Missing probability mas of a sequence x_n is denoted as $\delta(x_n)$; this value is computed as sum of *n* possible missing paths, where n is the length of the sequence. Let's assume that the prefix of the sequence x_n, with the length equal to k, has been matched. A portion of the α value of the last node in the prefix path should be considered as a missing probability mass of the sequence x_n. We can observe that there are n such prefixes and each of them should be taken into account. In addition observed propability mass of the sequence x_n is computed as a product $\prod_{x_i \epsilon x_n} q_{ml}(x_i)$; this is clear choice since $q_{ml}(x_i)$ represents an estimate of the probability of each symbol in the sequence. It is worth mentioning that more complex division techniques could be applied for partitioning the $\alpha(x_n)$ value. Since this is expected to be just a small portion of probability, a uniform division strategy is reasonable choice, but if one would decide to assume Gaussian distribution this would as well be a reasonable choice. We have restrained ourselves to the choice of uniform division of $\alpha(X_n)$ value. Visualization of technique performed for estimating value of $\delta(x_n)$ is presented in Figure 4.3. (Note that in visualizations of the tree structures some nodes are colored in light gray, these nodes represent nodes of the matched path.)

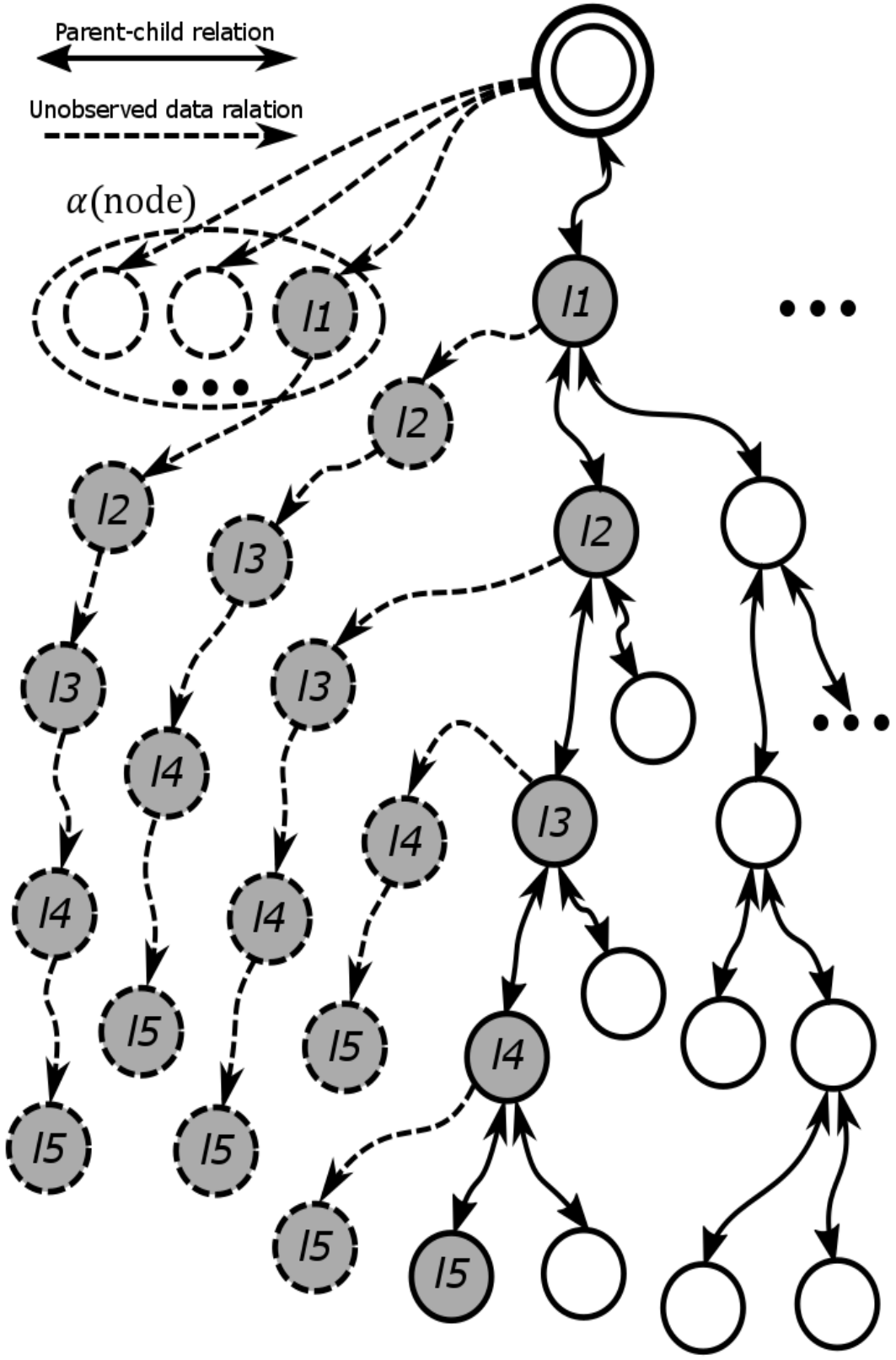

Figure 4.3: Visualization of generating δ(x_n) based on α value of each node in the path of interest

This is not enough to unbias total unconditional probability, we also need to perform similar modification for the $p(x = A|\ y = B)$ term. The underlying formalism is stated bellow:

$$\varphi(x|y) = \{x_n|y = x_n[n-1] \wedge x = x_n[n]\}, \quad [4.16]$$

$$\delta(x_n) = \sum_{x_i \epsilon x_n} \left(\frac{p(x_i)\alpha(x_i)}{k - h(x_i)}\right)^{n-i}, \quad [4.17]$$

$$k = sizeOf(alphabet), \quad [4.18]$$

$$h(x_n) = sizeOf\big(C(x_n[n])\big), \quad [4.19]$$

$$g(x = A|\ y = B) = \max_{x_n \epsilon \varphi(A|B)} \big(p(X = x_n) + \delta(x_n)\big),$$

$$= \max_{x_n \epsilon \varphi(A|B)} \left[\prod_{x_i \epsilon x_n} q_{ml}(X = x_i) + \sum_{x_i \epsilon x_n} \left(\frac{\alpha(x_i) \prod_{x_j \epsilon x_i} q_{ml}(X = x_j)}{k - h(x_i)}\right)^{n-i}\right]. \quad [4.20]$$

Now that this formalism have been established, a modified $mi(x = A|\ y = B)$ is defined as follows:

$$mi(x = A|\ y = B) = log\left(\frac{g(x = A|\ y = B)}{g(x = A)g(y = B)}\right). \quad [4.21]$$

Following the same logic, extensions can be provided in the cases of *A* and *B* being sequences of symbols, the only differences are in the definitions of the set φ. The set φ has been redefined in order to capture the fact that *x* is conditioned by *y*. In addition, the mutual information function is defined in slightly different manner:

$$mi(x = A, y = B) = log\left(\frac{q_{ml}(B[m]|A, B[m] \backslash B[m-1])}{q_{ml}(A[n]|A \backslash A[n-1]) q_{ml}(B[m]|\backslash B[m-1])}\right). \quad [4.22]$$

This case represents an extension of the previously defined mutual information function. The extension is observed in the fact that conditional part of probability estimator is a sequence of symbols while up until now we considered only the case where conditional part was a single symbol. To facilitate this extension we need to redefine $g(x = A|\ \ y = B)$ in the following manner:

$$\varphi(x|Y_n) = \{x_m|(\wedge_{i=1}^{n} Y_n[n-i] = x_m[n-i-1]) \wedge x = x_n[n]\}, \quad [4.23]$$

$$\delta(x_n) = \sum_{x_i \epsilon x_n} \left(\frac{p(x_i)\alpha(x_i)}{k - h(x_i)}\right)^{n-i}, \quad [4.24]$$

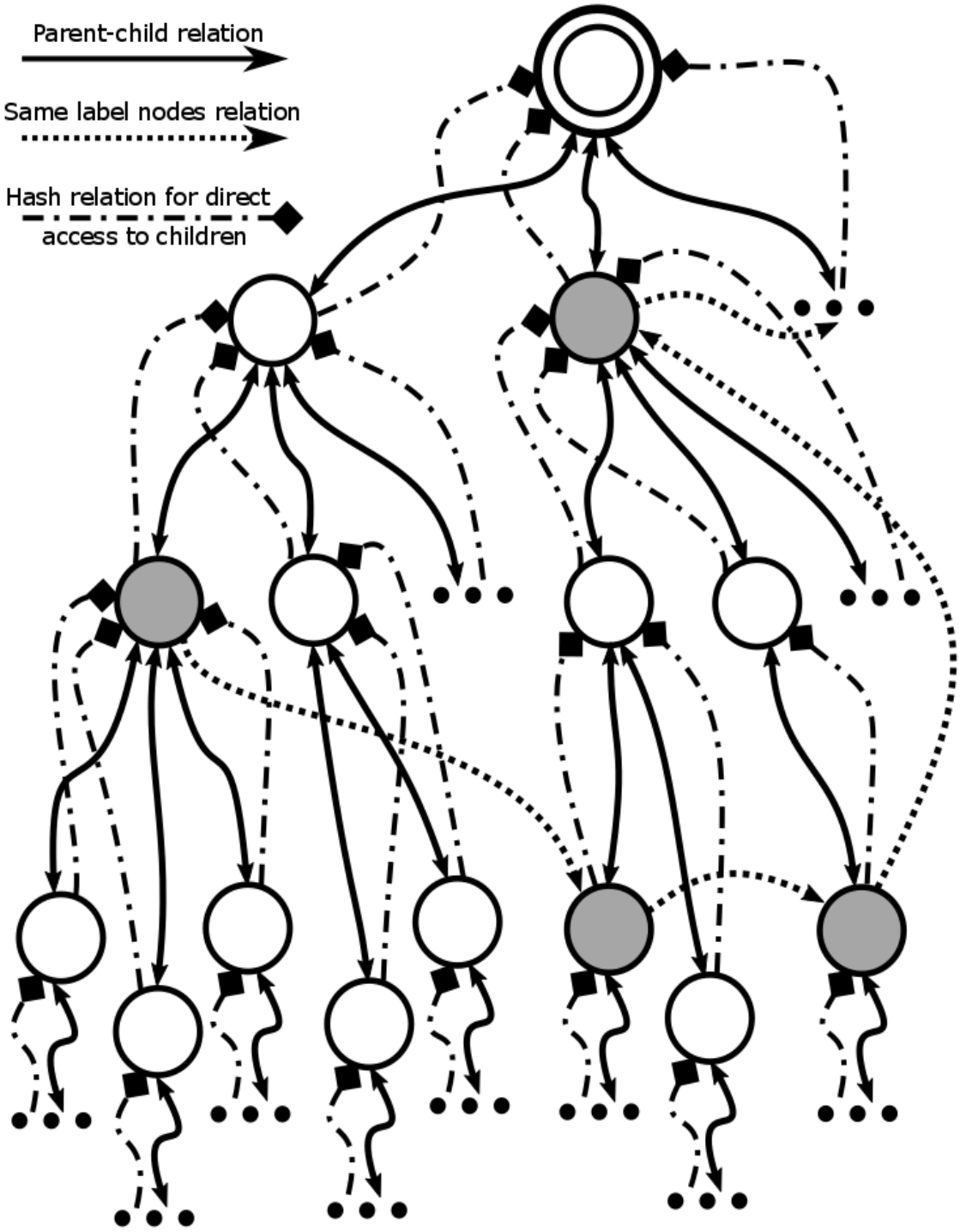

Figure 4.4: Complete visualization of n-gram tree data structure

$$k = sizeOf(alphabet), \quad [4.25]$$

$$h(x_n) = sizeOf\big(C(x_n[n])\big), \quad [4.26]$$

$$g(x = A|\, y = B_n) = \max_{x \epsilon \varphi(A|B_n)} \big(p(X = x_n) + \delta(x_n)\big)$$

$$= \max_{x \epsilon \varphi(A|B_n)} \left[\prod_{x \epsilon x_n} q_{ml}(X = x) + \sum_{x_i \epsilon x_n} \left(\frac{\alpha(x_i) \prod_{x_j \epsilon x_i} q_{ml}(X = x_j)}{k - h(x_i)} \right)^{n-i} \right], \quad [4.27]$$

where we assume that the conditional part can be a sequence while conditioned event can only assume the value of a single symbol. This particular restriction is in accordance with the use of $g(x = A|\, y = B_n)$ and therefore we do not lose generality. After redefinition of $g(x = A|\, y = B_n)$ we have obtained all needed comprising parts of mutual information function and we are able to redefine it as:

$$mi(x = A, y = B) = log\left(\frac{g(B[m]|A, B[m]|/B[m-1])}{g(A[n]|A/A[n-1])g(B[m]|B/B[m-1])} \right). \quad [4.28]$$

To facilitate all redefined concepts, our tree structure has been modified by adding horizontal linkage between nodes labeled with the same symbol. In addition backward linkage is added to facilitate fast search of set φ and δ value, whose search complexity are in the worst case O($|\varphi| * |Y_n|$) and O($|\delta| * |Y_n| * \max_{\forall x \epsilon \chi_{n+1}} \{C(x[n])\}$) respectively. Here χ_{n+1} represents all possible paths in the tree data structure of length equal to $n + 1$. Complete visualization of tree data structure used in this thesis is provided in the Figure 4.3. It is obvious that there are many additional pointers present in the data structure. Motivation behind usage of each individual additional pointer is provided throughout this section. Moreover, it is obvious that additional backward and horizontal linkage are dominated by links toward children. Inclusion of additional hash map has approximately same data consumption as list of children; therefore this is only a constant increase in memory consumption.

Character level n-gram tree with context-free root is created in an almost identical manner as it has been the case with character level n-gram tree with fixed root symbol set to Ш. In this particular instance root symbol is set to Џ. Symbol Џ in the scope of this thesis represents a universal symbol, meaning that this symbol is equal to any other symbol from the alphabet. Once we have imposed this condition, it becomes clear that this particular tree should observe depth limit. We denote depth limit as n, meaning that node at level 3 represents 3-gram estimator, node at level 4 represents 4-gram estimator and maximal is k-gram estimator where $k = n$. This tree represents a more conventional approach to n-gram estimators, as already stated in introduction, where, on average, and in state of the art natural language processing we encounter up to 5-gram estimators not more.

Character level n-gram tree with context-free root is used both in word segmentation procedures and, at runtime, as an underlying data structure in auto-complete and auto-correct capabilities, more on them in Chapters 5 and 6. All additional conditions that are valid for character level n-gram tree with fixed root symbol are valid for character level n-gram tree with context-free root. More precisely, modification imposed on mutual information function and entropy function are as well applied in the case of character level n-gram tree with context-free root. Also, the horizontal and backward linkage are likewise present, as well as a hash map of children for each node. This particular data structure is used in $BC_{wa}(L|R; n, m, w[k])$ and $EC_{wa}(k; w[k])$ and also in the cases where we apply exponential weighting or backward train data.

Once we have performed word segmentation based on the majority voting procedure, which uses both character level n-gram tree with context-free root symbol and with fixed root symbol, we can construct word level n-gram tree. The word level n-gram tree is based on the notion of words as labels instead of individual characters. Construction assumes context-free root symbol, which, in this case, represents the universal word. The depth of this particular tree is parameterized with maximal depth as it was the case with character level n-gram tree with context-free root, so the maximal n-gram estimator is k-gram where $k = n$.

Lastly, after word segmentation is performed, we can extract command ids from the train data. Ability to extract command ids easily comes from the fact that the "Cryptic" language is a pre-order language, and thus all the first words are identifiers of individual operations performed at each sample time. Every command starts with a special start symbol denoted as Ш, and in word segmentation it has been explicitly stated that Ш is treated as a word of the "Cryptic" language. Following this fact, command ids are always encountered as second words in each command. If we discard all other words except start word containing only symbol Ш, ending word containing only Ж, and command id word, we obtain the command sequence train corpus. Visualization of this procedure is provided in Figure 4.5.

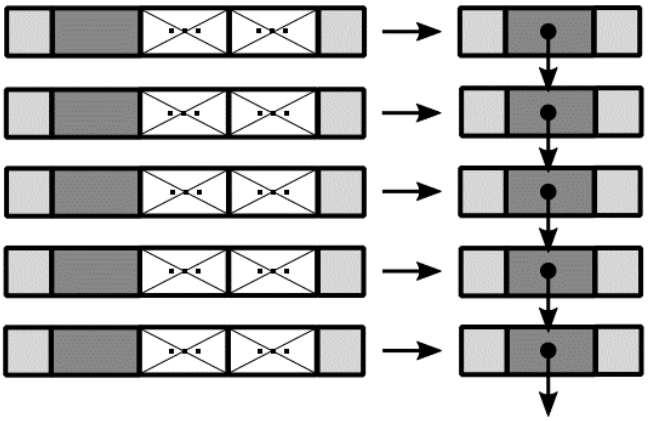

Figure 4.5: Modifying train corpus by removing all words except command identifiers

Based on this modified train data we can construct command level n-gram tree, which differs from word level n-gram tree only in the performed modification of the train data. Command level n-gram tree is used to predict which particular command user will type at each moment. This prediction is used to navigate in word level n-gram tree, more on this in Chapters 5 and 6.

5 “CRYPTIC” AUTO-COMPLETE

“Cryptic” auto-complete capabilities are based on the character level n-gram tree with context-free root (further referred as CHT) and the word level n-gram tree (further referred as WRDT), with addition that the command level n-gram tree (further referred as CMDT) can be used to predict future commands. Each of these particular data structures is used at a specific moment to provide more flexible predictions. As we have explained in Chapter 4, the children of each tree node are sorted in decreasing order of q_{ml}, from this fact emerges that depth first search of the n-gram tree provides maximum likelihood prediction, since all left most children observe highest values of probability estimations.

Due to different treatment of train data CHT and WRDT provide somewhat different estimates, and we combine their outputs to provide the final prediction presented to the user. Since all children of a particular node are ordered w.r.t the decrease of q_{ml} value, maximum likelihood prediction is actually just the leftmost path that matches the current user’s input. It could happen that a path that matches current user’s input does not exist; in this case we firstly apply auto-correct on partial user’s input and on corrected input we apply auto-complete. Auto-correct capability is explained in details in Chapter 6.

CHT is used in a similar manner as it was the case with word segmentation. We can understand this as a window of length n, where n represents depth of n-gram

tree, that slides over the user’s input and provides prediction of how to complete the current word. It is required to be stated that after word segmentation is performed, CHT is restructured to be used at runtime. Reconstruction assumes that CHT is responsible for character level predictions inside of word boundaries, so the stopping condition is not the end of the command but a word delimiter, and the starting symbol is set to word delimiter as well. This way CHT is able to complete current word.

Once we obtain current word completion, we can finalize prediction of current command by consulting WRDT data structure. Prediction procedure is conceptualized as search for the word sequence that contains command end word while the total probability accumulated in this sequence is maximized. Once such sequence is encountered the final prediction is provided to the user. It is worth mentioning that we observe prefix matching while we are computing prediction. Current input is firstly segmented into words. Then this list of words is considered as a prefix list that must be matched and, based on that path, a prediction is provided as the leftmost path till the command end word. Since WRDT is a tree with fixed depth, it could happen that the number of words inside of the command is larger than the actual depth of the tree. Depth misalignment is not likely for WRDT with the depth approximately equal to 5. However, depth misalignment could happen due to the imperfection of word segmentation technique and the fact that the number of actual words is on average smaller than the number of words in data produced by segmentation algorithm.

As it has been shown in Figure 5.1, if we were to find prefix matched path from which we cannot extract a path that ends with the command end word, we could apply once again sliding technique. Prefix will slide for one position to the right; the search will be performed again based on newly obtained maximum likelihood prefix path. The search will be continued until the command end word is encountered. It is obvious that in the “Cryptic” language settings there are many sources of noise in the prediction; this implies that prediction precision might be hindered by some factors such as sliding technique. On the other hand, n-gram trees exponentially explode with the increase of depth, especially in case of context-free root symbol, so in this particular instance we need to keep depth bounded to a reasonable value. Another source of imperfections comes from the tree pruning procedure. Tree pruning is necessary because many commands are encountered only one or two times, and these words are statistically insignificant. However, these words can increase the size of data structure significantly on average.

Finally, CMDT can be used in the case when user have not started typing the current command. At this particular moment user’s current input is comprised of only the implicit start symbol. The best word prediction we can extract from CHT or WRDT might lead into a situation where the prediction provided to the user is a command that is not following the logical flow of commands. From CMDT, we can extract maximum likelihood estimate of command id w.r.t. to the observed flow of previous commands issued by the user. By using CMDT at this step, we provide to the user a word that is actual maximum likelihood estimator of the current command, even if there is another word in WRDT which has higher value of q_{ml}. Once user actually starts to type we use CHT to predict the word that is being typed, and when we obtain prediction we consult CMDT dictionary to test if the word is actually a command identifier, if this is not the case then we perform auto-correct and on corrected input we predict the current command.

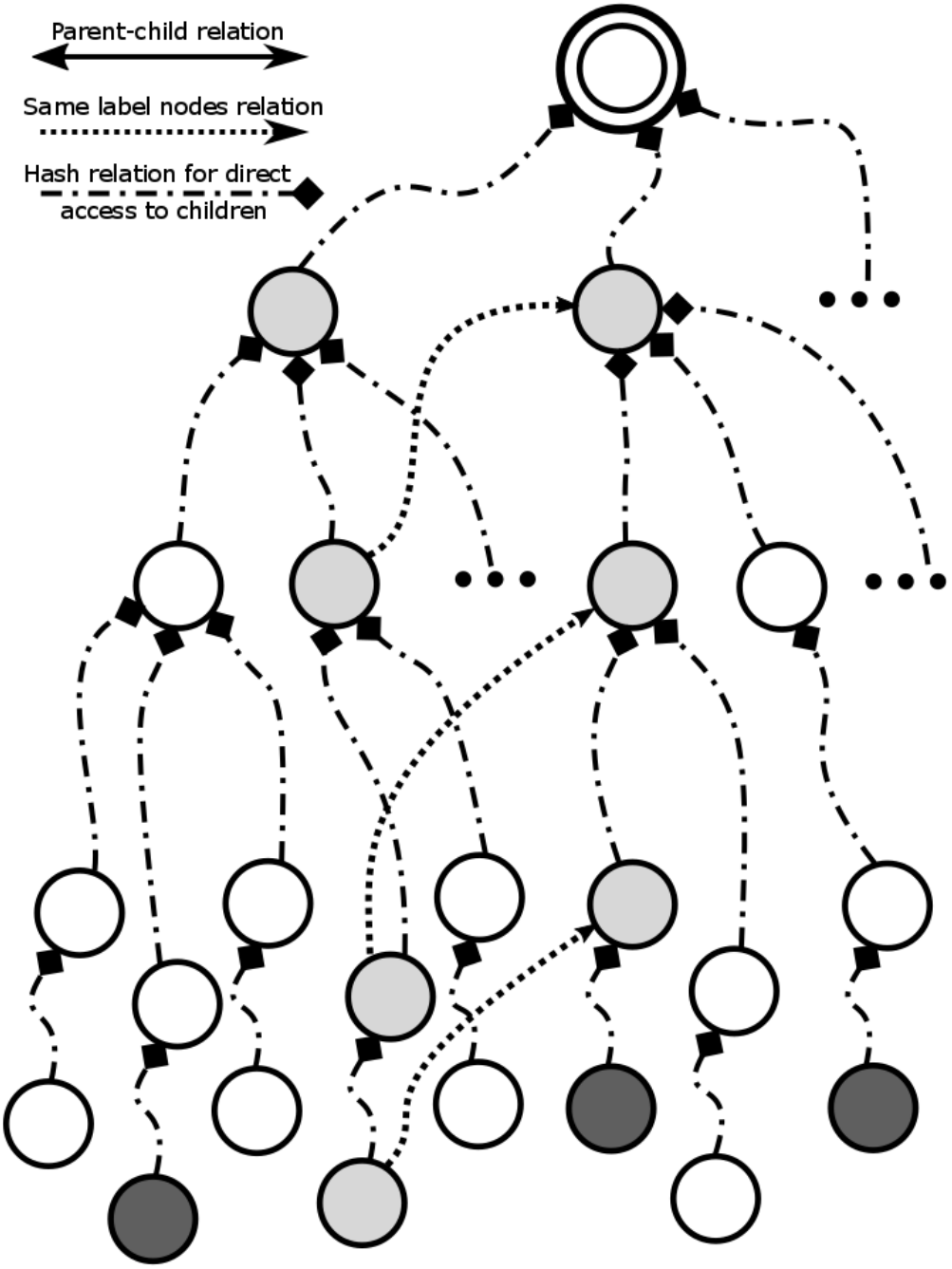

Figure 5.1: Principle of sliding context in auto-completion procedure

6 “CRYPTIC” AUTO-CORRECT

Auto-correct feature of the “Cryptic” language is based on the character level n-gram tree with a context-free root (further referred as CHT), the word level n-gram tree (further referred as WRDT) and the command level n-gram tree (further referred as CMDT). As it has been presented in Chapter 5, the auto-correct functionality is used in the auto-complete functionality as a smoothing technique, when there is an indication that the input is malformed. In addition, we always test user’s input for errors and provide additional predictor with corrected input that observes an increase in probability value, even when there is no indication of typographical mistakes. Both auto-complete and auto-correct outputs are presented to the user, and he can decide which one, if any, is the correct prediction.

When the user has not started typing, input is comprised only by the start symbol. At this moment, the auto-correction output is extracted from CMDT, and this output is identical to the one provided by the auto-completion procedure. Once the user starts typing, his input becomes a string that we desire to correct. Firstly, this input is segmented using the same functionality used in word segmentation phase. If the input consists of only two words, the first word is always the command start word, and we use CMDT to facilitate the correction procedure.

As presented in Figure 6.1, the first important step is to match the command pattern in CMDT. Command pattern is based on from user session command history. For pattern matching purpose, we use the longest common subsequence relation. Maximization of the longest common subsequence is equivalent to minimization of edit

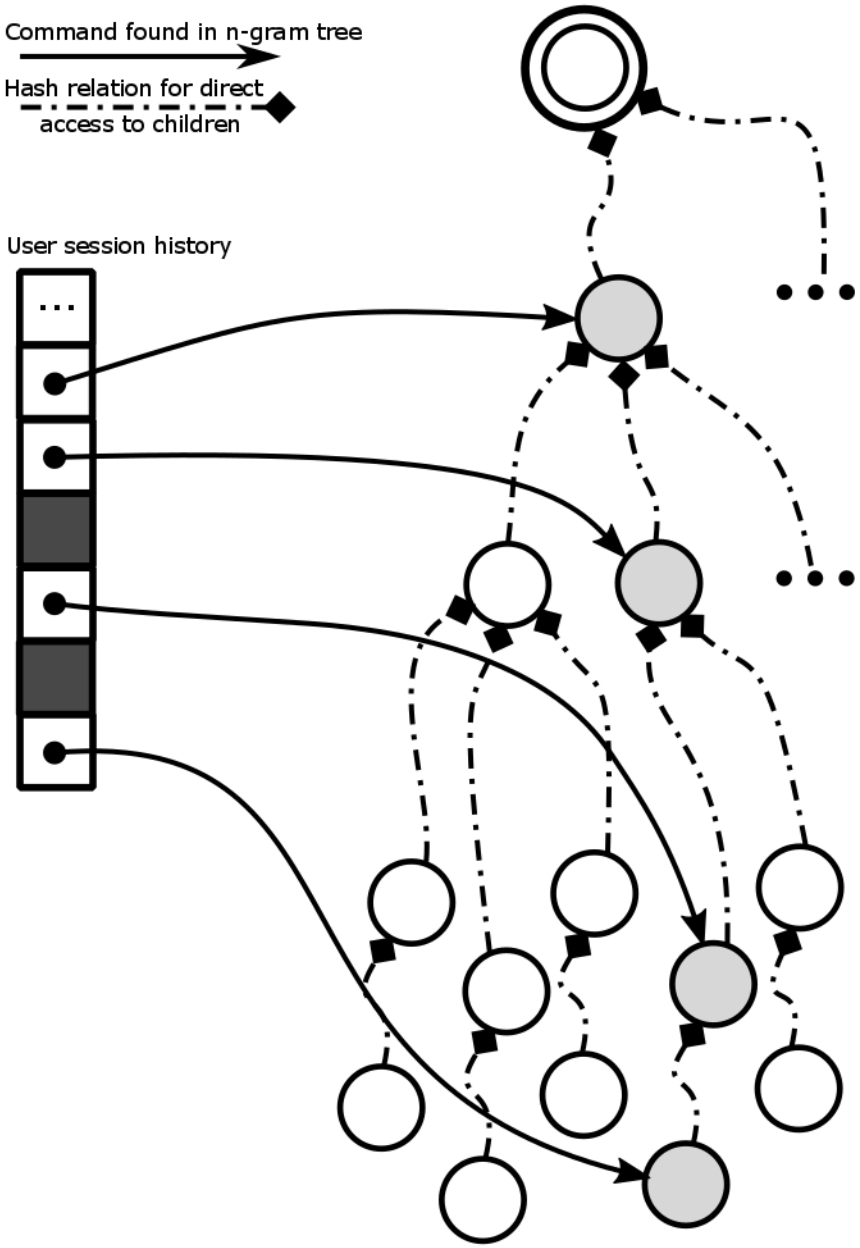

Figure 6.1: Longest common subsequence matching between session history and n-gram tree

distance between two command sequences. This way we do not enforce perfect match as long as sequences observed in CMDT are encountered in user session history in same order. N-gram relations, in the case of command identifiers, can be understood as “happens before” relations that allow other commands to occur in between observed commands. More precisely, if we are observing commands x and y in the user session command history, and we want to match them in CMDT, what is important is the order of command executions. In this case x was executed before y. What is of less significance is that in between x and y we have observed command z which if matched

in CMDT would significantly decrease the probability of the matched pattern. It is worth mentioning that large gaps of not matched commands between commands x and y should be penalized. We allow the longest common subsequence procedure to partially match user’s command sequence as long as probability estimate is increased, and there are not many (nor large) gaps between matched commands. Another possible approach to the session history pattern matching is the sliding technique used in the auto-completion functionality. It should be noted that CMDT depth is usually chosen as a small value close to 5 while user history could be a lot larger, it is not uncommon that user performs hundreds of commands in one session. Every time we slide a window on CMDT tree we perform $O(k)$ operations where k is the depth of the CMDT tree, since we try to match as much context as possible. This approach would be expensive, moreover at every slide, due to context free root, we add some uncertainty which implies that benefits from this more complex approach would be minor and this fact pushed choice of session history pattern matching toward longest common subsequence in our implementation.

Once a session history pattern is matched and we have obtained a context of length $k-1$, where k is the depth of CMDT tree, we can search in the children of the last node in the context for prediction of current command. At this step once again we perform pattern matching with the difference that in this particular case we match labels of children nodes with user’s current input. To facilitate this functionality we need to define a possible typographical error. We can observe three different cases of typographical errors:

- Missing character
- Redundant character
- Swapped characters

We have introduced additional symbol to help us represent errors in the string. Symbol reserved for this use in this thesis is З. Symbol З is a Cyrillic letter and not number 3, with the flexibility of being able to set this symbol to any other value outside of “Cryptic” language alphabet.

The strategy we decided to follow was to firstly generate all possible errors in the string up to value k, where k is a parameter of a procedure and it can be changed. Somewhat empirical choice has fallen to value close to 3, any value larger than this would be too liberal in correcting the input, while just one error allowed per word is

valid choice as well. Once we have generated all possible strings containing an error, symbol 3 denotes position of the imputed error, we perform error correction of each individual string containing errors and we treat every 3 symbol in all three possible ways.

In the first case, we add one more symbol to the word and to produce highly likely word we use the CHT tree. As it has been depicted in Figure 6.2, the word is split into three parts, error-prefix (further referred as EP), symbol 3 and error-suffix (further denoted as ES). Firstly we need to match EP if we manage to match EP we proceed with matching ES. ES is matched using backward linkage, and we are interested in finding a set of nodes which connect these two paths if any exists. If we obtain non-empty set of such nodes, we choose as additional symbol label of the node that produces complete path that has the largest value of q_{ml}.

In the second case, we treat symbol 3 as the demarcation of the redundant symbol which should be removed. The approach is, in general, similar to the previous case. The only observed difference is in the treatment of symbol 3. Same as before, we split the word into three parts, EP, 3, and ES. Here we remove the first symbol from ES, and the rest of the strategy is the same. Firstly we need to match EP path in CHT tree and then we need to match ES using backward linkage. Once we connect these two sets of paths we propose to the user path that has the highest value of q_{ml}.

Lastly, we treat swapped character errors in the following manner. Word is split into three parts EP, 3, and ES. In this instance, the last symbol of EP and the first symbol of ES are exchanged and then matching is performed by using CHT tree. EP with changed last symbol is matched in the forward direction, and ES with changed first symbol is matched in the backward direction. Once the sets produced by EP and ES matching are connected, the path with the highest value of q_{ml} is chosen as corrected word.

These three procedures are executed in the same way regardless of the word position in the command. The only difference arising from the different position of the word inside of the command is the usage of CMDT if the word we are trying to correct is placed in the second position. We remind the reader that first word is always command start word. Thus the second word is the command identifier, for all other words WRDT tree is used. Once the set of corrected words is produced we replace the corresponding word in the current input word list. The word that is being corrected is

assumed always to be the last word in the input word list because we incrementally provide to the user corrected and completed output. At this point we assume user had already observed corrected output for all previous words and acted upon the provided output. This assumption implies that strategy for choosing corrected output word is consisting of matching a path in WRDT tree with sliding approach up until the last word has been matched. Once maximum likelihood path is obtained, we can extract the

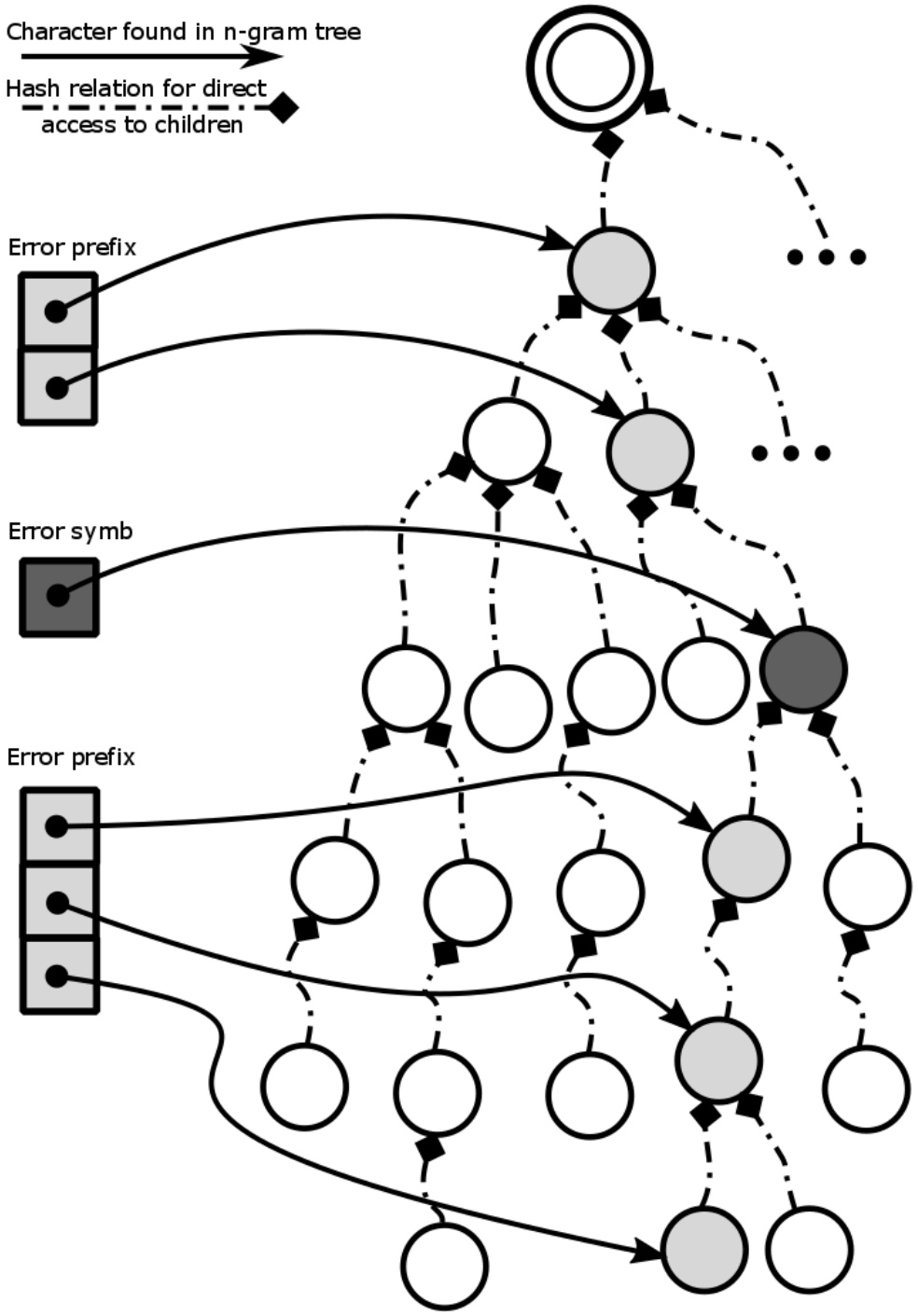

Figure 6.2: Visualization of the word split into EP, ES, and 3 and mapping of each part to corresponding path in the n-gram tree

children of the last node in this path. The set of the children obtained this way is then compared with the set of possibly correct words and the intersection of these two sets provides a set of probable words. Word with the highest value of q_{ml} is chosen as corrected word.

There are other possible approaches to deal with the issue of correcting typographical errors. Our solution tries to tackle as many as possible restrictions imposed by the “Cryptic” language structure. Results obtained by using this strategy are provided in Chapter 7 in more detailed fashion.

7 VALIDATION

7.1 Word segmentation validation

As it has been explained in Section 3.4, our solution bases segmentation decision on the following individual criteria:

- $EC_{max}(k)$
- $EC_{min}(k)$
- $EC_{wa}(k; w[k])$
- $EC_{ea}(k; w[k])$
- $EC_{bwa}(k; w[k])$
- $EC_{bea}(k; w[k])$
- $H(X|X_n = x_n)$
- $H_b(X|X_n = x_n)$
- $BC_{wa}(L|R; n, m, w[k])$
- $BC_{ea}(L|R; n, m, w[k])$

These criteria are then used for one of two possible voting procedures. Either all criteria involved are voting with equal voting rights or based on performance rank each criterion has unique voting value.

Firstly we need to define measurements used for validation procedure. In this

thesis we use three measurement values, denoted as:

- Precision
- Accuracy
- False omission ratio

A precision measurement is defined as the ratio between true positive decisions and all positive decisions. In this case, positive decisions are a decision to place word separator in a particular position in the text, implying that true positive decisions are correctly placed word separators. On the other hand, false positive decisions are word separators placed in the position where they should not appear. With respect to our system, false positive decisions are not considered as strict errors, the only nuisance induced by these errors is a slightly longer path in the WRDT data structure. From a language perspective, these errors could appear if some words share a common prefix, especially if segmentation techniques are based on mutual information or entropy value, as in our case. The following formula defines precision measurement:

$$p = \frac{tp}{tp + fp}. \quad [7.1]$$

In Formula 7.1 tp represents the count of true positive decisions and fp represents the count of false positive decisions. Precision as a measurement w.r.t. our solution might be misleading since, as we stated, false positive errors are not considered as strict errors, therefore precision of 0.5 might imply bad results but in the frame of the “Cryptic” system this can be understood as simply observing two times more segmented data which will still result with more compressed predictor than in case of unsegmented commands.

Accuracy measurement is a more stable measurement in the scope of the “Cryptic” system since it treats both positive and negative decisions. Segmentation procedure is making decisions for each pair of characters in a sense of placing them together or separately. It is not reasonable to neglect decisions to group two characters together when speaking of the performance of a segmentation technique. The negative decision, w.r.t. our system is a decision not to place word separator between two characters. Once negative decisions are defined, we can proceed and state that true negative decisions are decisions made to group two character together when they should be grouped together while a false negative is the omission of word separator. In the scope of the “Cryptic” system, false negative errors are considered to be strict errors

since they are forcing two separate words to be grouped together. Accuracy measurement takes into account both positive and negative decisions, and it is formally defined by the following formula:

$$a = \frac{tp + tn}{tp + fp + tn + fn}. \quad [7.3]$$

In Formula 7.3 tp represents the count of true positive decisions, tn represents the count of true negative decisions, fp represents the count of false positive decisions and finally fn represents the count of false negative decisions.

Like we have already stated, we are more concerned with false negative decisions than with false positive decisions. We can go one step further and formalize the false omission ratio measurement. This measurement is based only on negative decisions like a counterpart of precision measurement. False omission ratio is formally defined as follows:

$$for = \frac{fn}{tn + fn}. \quad [7.2]$$

In Formula 7.2 tn represents count of true negative decisions and fn represents count of false negative decisions.

Now that measurements are defined we can focus the discussion toward the segmentation decision criteria. Firstly, let's observe $BC_{ea}(L|R; n, m, w[k])$, this particular criteria, like explained in Section 3.2, is dependent on the maximum size of the left and right contexts. We have run this criteria on values of *k* varying from 1 to 5 on the train corpus based on Shakespeare's "Romeo and Juliette". This book has approximately 26000 words which roughly corresponds to the average data available per user in the "Cryptic" system. The reason for conducting segmentation validation on English language corpus instead of the "Cryptic" corpus is reflected in the fact that there does not exist any "Cryptic" data that is properly segmented, neither by hand nor by automatic procedure, thus forcing the choice of validation data to be chosen from some other language. Results of validation of $BC_{ea}(L|R; n, m, w[k])$ criteria are presented in Figures 7.1-7.3.

As it can be observed from the data, accuracy and precision measurements are decreasing with the increase of contextual data. This discovery might be a surprise, but it comes from the fact that including more contextual data leads to extracting more segmentation information on phoneme level ergo segmenting words into root, prefixes,

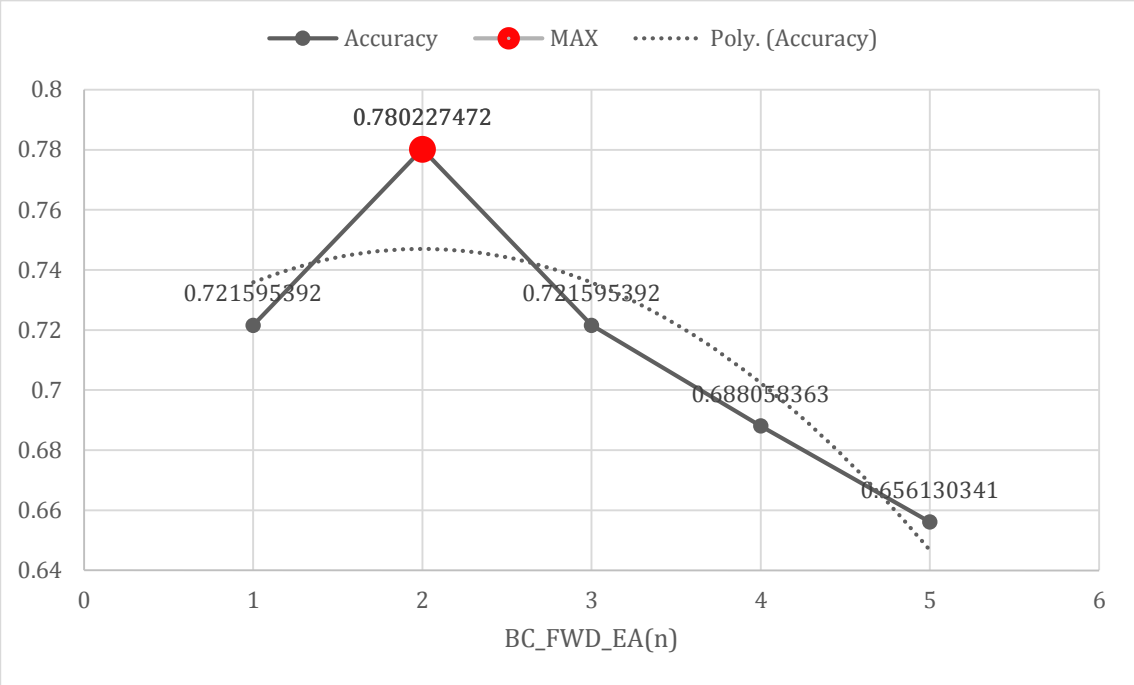

Figure 7.1: **Accuracy measurement of $BC_{ea}(L|R;n,m,w[k])$**

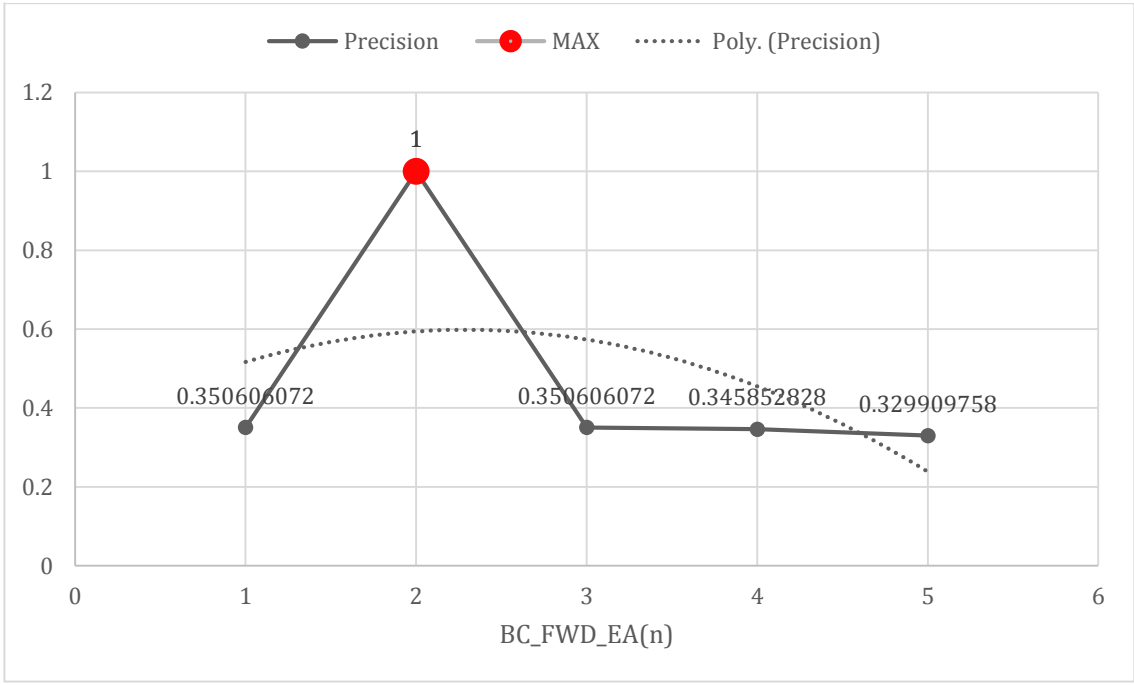

Figure 7.2: **Precision measurement of $BC_{ea}(L|R;n,m,w[k])$**

and suffixes. This particular issue, as previously have been stated is not considered as an actual error in the “Cryptic” system, while this is a mistake in the scope of the English language. What is more interesting is that false omission ratio is as well monotonically decreasing with the increase of contextual data, which is of more importance. On this particular segmentation criterion, we can observe a peak value of

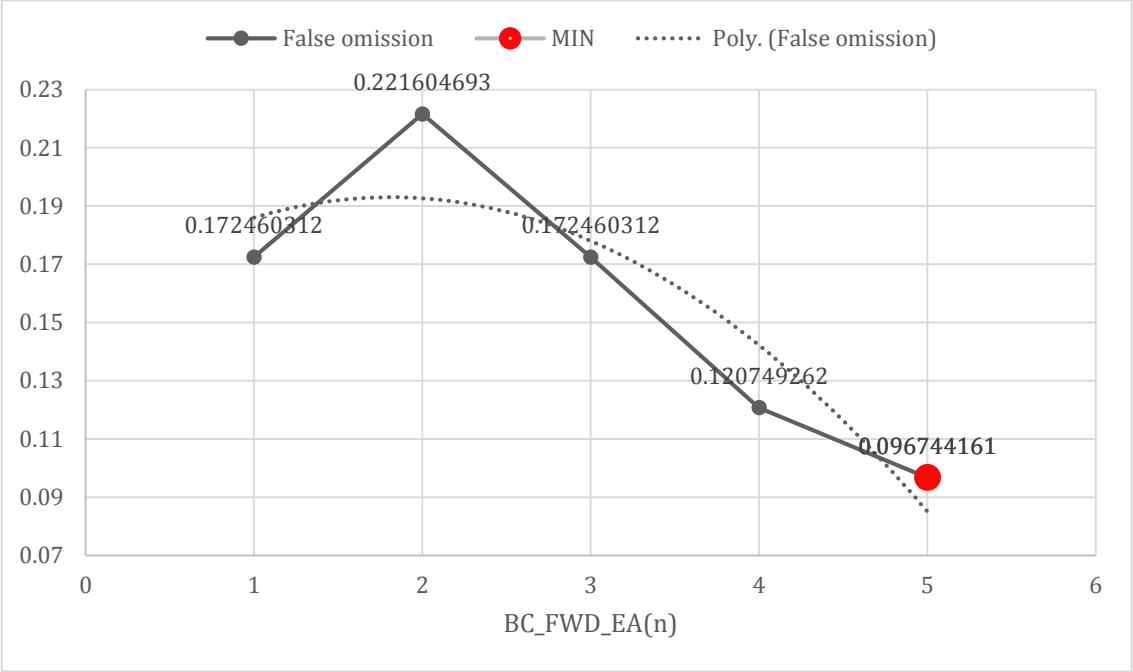

Figure 7.3: False omission ratio measurement of $BC_{ea}(L|R;n,m,w[k])$

1.0 for precision measurement while accuracy is 0.78 approximately, and false omission ratio is around 0.22, for a value of maximum context set to 2. In this concrete case, we observe a misleading quality of precision measurement since precision measurement equal to 1.0 implies no false positive decisions are made. However, we observe 22% of false negative decisions; these decisions are considered as strict errors. In the "Cryptic" system, we give priority to false omission ratio and only in the cases when we do not observe a significant improvement in this measurement that we consult accuracy and precision measurements.

Following the same logic what emerges as best segmentation criteria is $BC_{wa}(L|R;n,m,w[k])$, data observed from this criterion is presented in Figures 7.4-7.6. $BC_{wa}(L|R;n,m,w[k])$ observes a peak value at context length equal to 2, after which it monotonically decreases. If the context length is further increased all measurements improve up till a maximum values for precision and accuracy at context length equal to 5. For this context length, we also observe minimum value for false omission ratio which makes context length equal to 5 as best choice.

For an extensive comparison of each segmentation criterion and the underlying results on which particular choices of parameters were made refer to Appendix R. Using these findings we have created a comparison between all proposed criteria and we have extracted the ordering among criteria that was later used for voting procedure with priority votes. Following ordering of criteria in descending order was obtained:

1. $BC_{wa}(L|R; n, m, w[k]); k = 5$
2. $EC_{wa}(k; w[k]); \mathrm{k} = 3$
3. $EC_{ea}(k; w[k]); \mathrm{k} = 3$
4. $BC_{ea}(L|R; n, m, w[k]); k = 4$
5. $EC_{min}(k); \mathrm{k} = \infty$
6. $H(X|X_n = x_n)$
7. $H_b(X|X_n = x_n)$
8. $EC_{bea}(k; w[k]); \mathrm{k} = 3$
9. $EC_{bwa}(k; w[k]); \mathrm{k} = 4$
10. $EC_{max}(k); k = \infty$

This ordering is based on the previously explained notion of prioritizing minimization of false omission ratio measurement while, if possible, maximizing accuracy and precision measurements respectivelly. Comparison between all segmentation criteria based on the value of false omission ratio is presented in Figure 7.7 while other two measurement comparisons are presented in Appendix R.

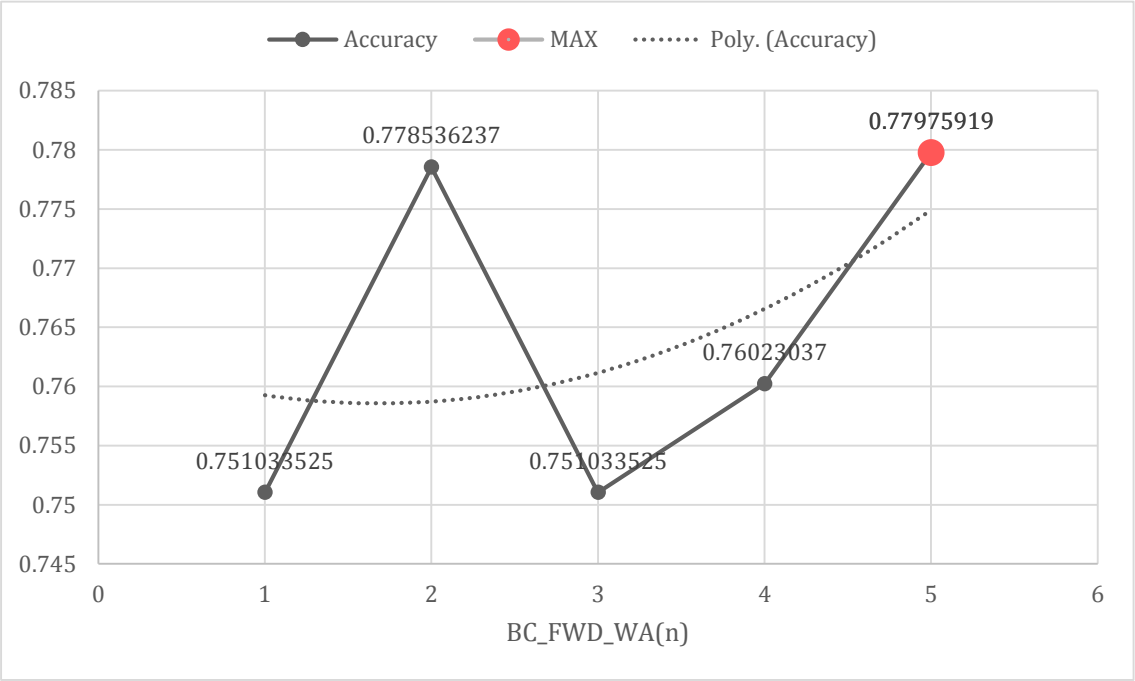

Figure 7.4: Accuracy measurement of $BC_{wa}(L|R; n, m, w[k])$

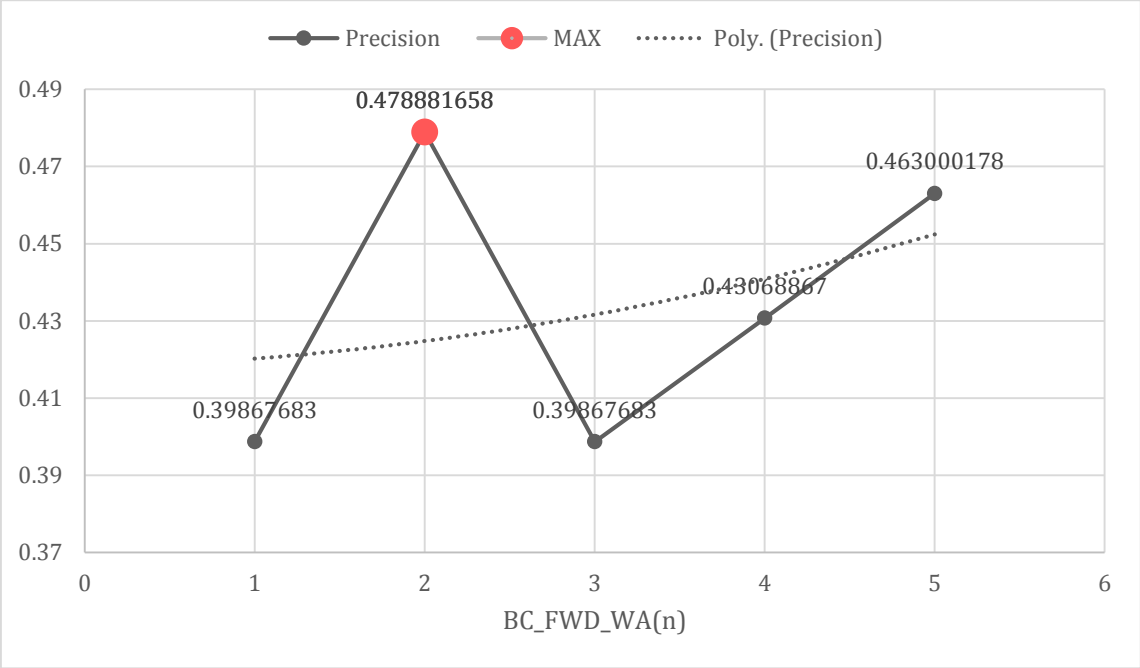

Figure 7.5: Precision measurement of $BC_{wa}(L|R; n, m, w[k])$

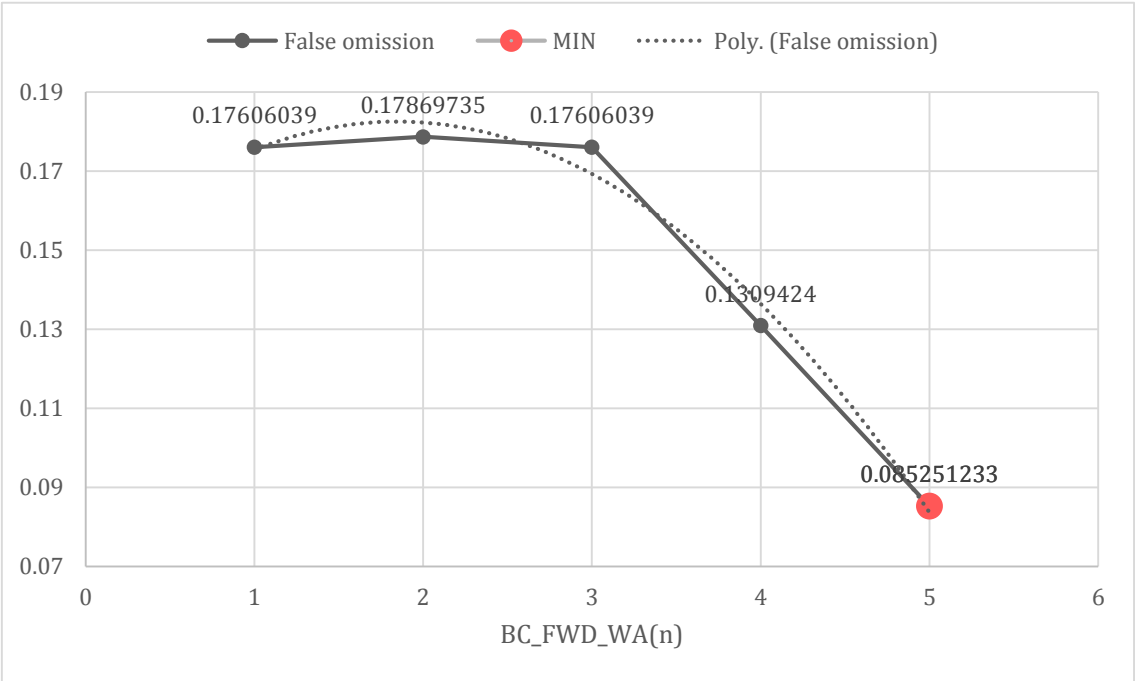

Figure 7.6: False omission ratio measurement of $BC_{wa}(L|R; n, m, w[k])$

Once ordering among criteria was established, we have run a comparison between majority voting procedure with equal right votes and voting procedure with ranked vote values. Results obtained from voting procedures are considered as complex segmentation criteria and are interpreted in the same manner as it was the case with individual criteria. Comparison between these two complex criteria is presented in

Figures 7.8 and 7.9. From these Figures we can observe that equal voting procedure monotonically decreases the value of false omission ratio, which implies an improvement of performance with an increase of corpus size. We have limited corpus size to obtain more reliable estimate since the running data is of limited size. It would be unreasonable to run validation procedures on corpus size of 300K or 3M words while actual user log contains roughly 30K words on average. This limited size of the user logs comes from the storage utilization in the “Cryptic” system. Once a file age reaches 100 days this file is permanently deleted. Thus limiting user’s data to 100 days long history. On the other hand, we could combine more users and create combined corpus used just for segmentation. Unfortunately, this option was unfeasible to be tested practically due to the company’s security policy, only one user’s history logs were made available for this research. Ranked voting procedure w.r.t. false omission ratio outperforms equal voting, which is observed in Figure 7.9. We can notice that false omission ratio stabilizes after corpus size becomes larger than 10K words reaching value of only 1.7% with variance of only 0.1%. On the other hand, equal voting procedure was performing at 12.9% with variance of 0.3%. Comparison based on accuracy and precision is given in Appendix R, but as it has been already stated these two measurements are used in cases where no improvement is made on false omission ratio. It is obvious that significant improvement was achieved by applying ranked voting procedure.

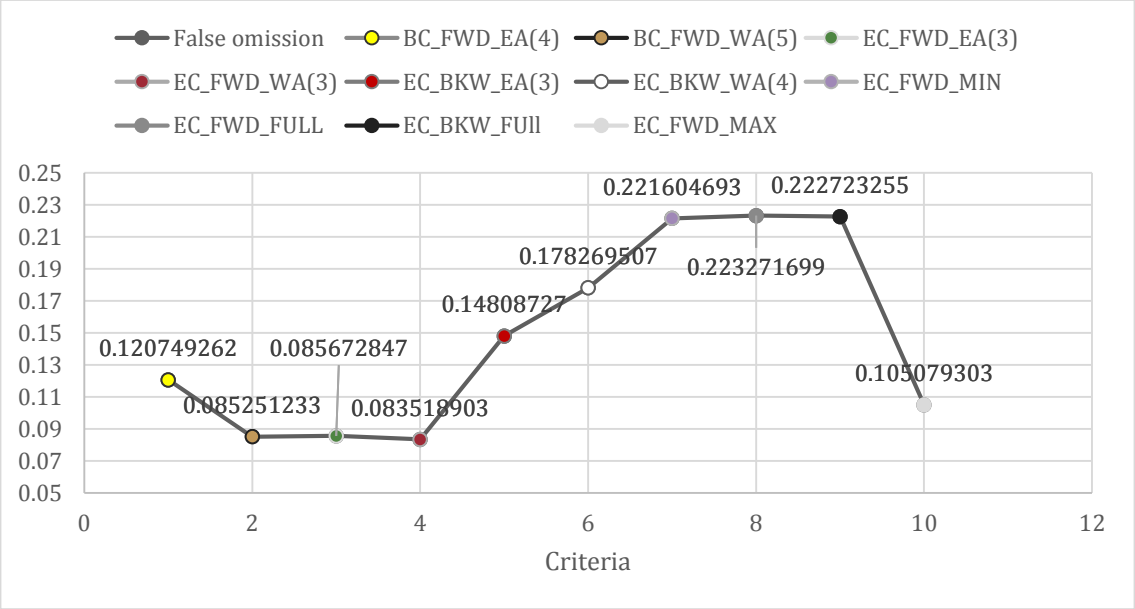

Figure 7.7: False omission ratio measurement comparison

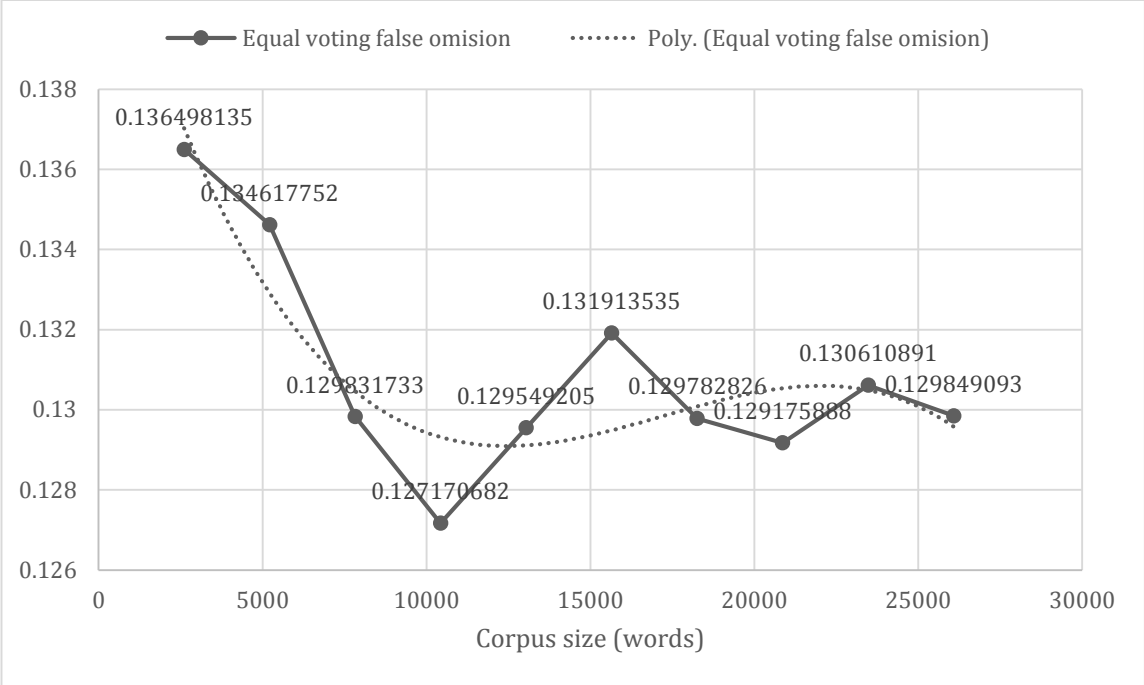

Figure 7.8: False omission ratio measurement of equal voting procedure

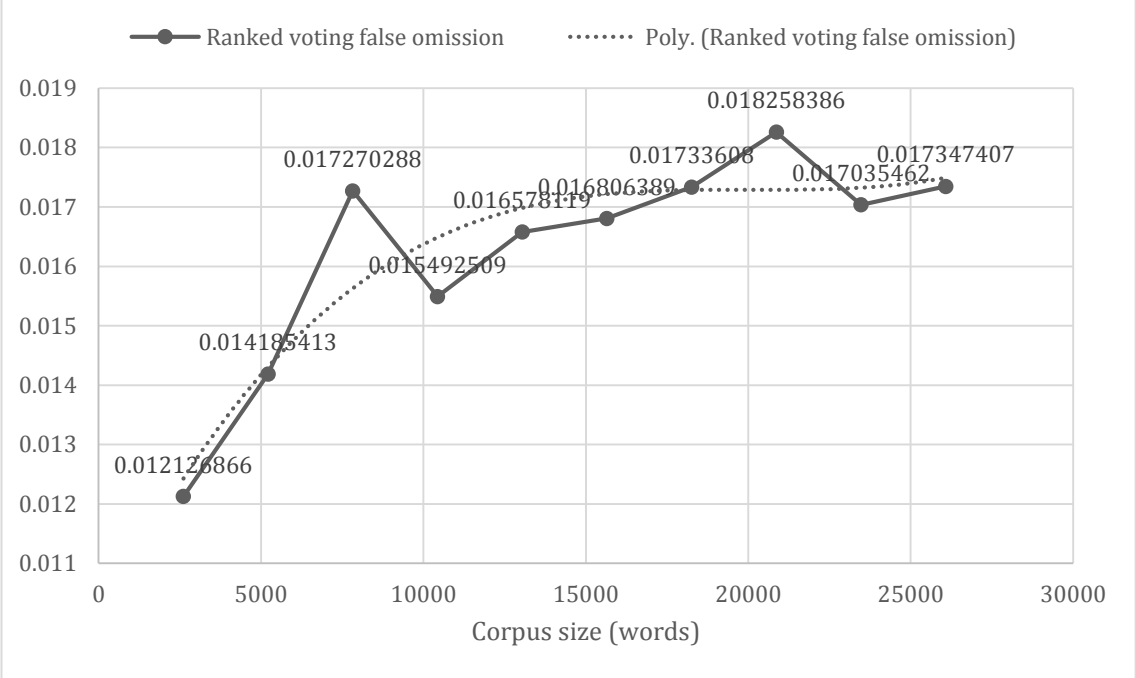

Figure 7.9: False omission ratio measurement of ranked voting procedure

7.2 Auto-complete/correct validation

Performing validation of procedures used for auto-complete and auto-correct was somewhat more challenging. These operations are to some degree more dynamic than word segmentation because they involve online user interaction. As we have

previously established, while discussing auto-complete and auto-correct features in Chapters 5 and 6, user’s input is firstly segmented into words and only the last word is treated with auto-complete and auto-correct. This choice came from the fact that user’s input can be understood as a pipeline through which words are coming at some specific rate. From the system’s perspective word arriving at time t_1 implies that words that arrived at any time t_2 such that $t_2 < t_1$have been already completed and corrected.

Based on this fact and dynamic behavior, we have performed a simulation of auto-complete and auto-correct features. Train data and validation data are divided in ratio 9:1 meaning 10% of the user’s logs are left for validation purposes. Division among train data and validation data is done using sampling without replacement, meaning that data can appear either in train data or validation data but not in both data sets. Validation procedure is conceptualized on three following measurements:

- Average in common string ratio
- Maximum in common string ratio
- Average in common right-hand side string ratio

Average common string ratio, further denoted as ACSR, is conceptualized as a mean value, or sample mean value, of common string ratios for every partial input of a particular command. Stated in more simple words, we simulate every command as the user would type it in real time environment, one character at the time. At every character arrival, the new partial input is created by appending the newly arrived character to previous character sequence and this string is treated with auto-complete and auto-correct features. Corrected and completed string is then compared with the full command input from validation set and ratio between common prefix string length and actual command length is computed. Common prefix string is computed as prefix part of two compared strings until first character not matching. From this point, ACSR is defined as the average value of the ratio between common prefix string length and complete command length over all partial inputs for the concrete command.

The reason for defining this type of measurement are the one time words in the language. Examples of these words are passenger names, ticket serial number, and others. Concrete examples of one time words appear only statistically insignificant number of times in data corpus, but on average this class of words occurs often. Consequently, we can expect flight numbers to appear often in data, but we cannot expect the same number to appear more than the small limited number of times. These

words cannot be predicted by auto-complete and, therefore, they skew the performance estimates. To deal with this issue, ACSR was devised.

Another possible measurement is the maximum common string ratio, further denoted as MCSR, defined on the same notion of left common prefix string ratio. For this measurement, the only difference is that maximum length of the partial input, in order to be accounted in the measurement, is 2/3 of the length of command from validation set. This way we restrain MCSR from always obtaining value equal to 1.0 when auto-complete matches current validation command completely, once the partial input is close to the end of actual validation command.

Lastly we define average in common right-hand side string ratio as more aggressive measurement, further denoted as RACSR. Here we compute the ratio between in common string length and actual validation string length, just in this case we used trimmed partial input and trimmed validation string. Strings are trimmed in such way that words that have been already completed are removed from the string and after removal of such words the common string is computed. This measurement might be biased toward producing a more pessimistic estimate. If the partial input was correctly completed up until the end of the current word and the rest was miss predicted this measurement will produce small percentage value. Previously stated fact implies that if the partial input is correctly predicted in an incremental manner, meaning that it needs at least first few characters of each word, RACSR will estimate performance at really low level.

The Ranked voting segmentation procedure on train data is used to build a model, and once predictors have been obtained, validation is run to produce these three measurements. We have incrementally increased the corpus size and computed ACSR, MCSR, and RACSR. In addition, random intentional typographical errors were included in validation data. For actual values of ACSR measurements consult Figure 7.10. Due to the correlated nature of auto-complete and auto-correct, meaning that these features are inter-dependent, results of one are used for computing the other, validation is not run separately but jointly. It can be noticed that MCSR produces a measurement that is biased toward implication that completion is extremely successful while RACSR is biased toward implication that completion is somewhat unsuccessful. We can state that for the purpose of validating auto-complete and auto-correct features, ACSR should be regarded as the most stable measurement. Other two measurements have been proposed just as alternatives to ACSR.

Another important thing to be noticed is that ACSR measure is more or less stable on a value close to 81%, this can imply that user corresponding to train data is biased toward using short command. After manual inspection user’s bias toward short commands was verified to be true. Unfortunately, this was the only data provided by Amadeus, and statistics obtained from some other user might be more pessimistic.

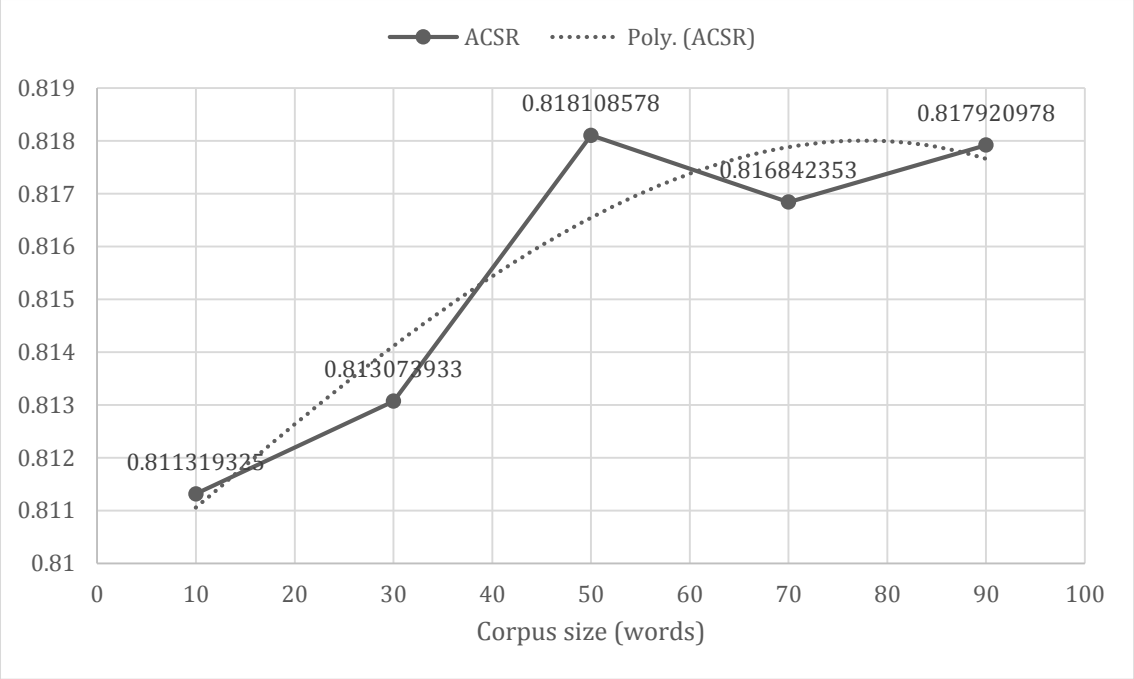

Figure 7.10: ACSR measurement of auto-complete/correct run on validation data with typographical errors

8 CONCLUSIONS

As it has been presented throughout this thesis, there are many limitations imposed by the lack of strict structural approach in the design of the "Cryptic" language during its genesis. Most notable limitations are observed in the absence of word segmentation in the language propositions. Word segmentation is observed to liberally allowing the user to choose whether he/she will put word separation or not; most users choose not to place word separators to increases the speed of typing. In addition, there is no data that is already segmented, forcing validation of word segmentation technique to be performed on a corpus of English language. Otherwise, we would not be able to provide any notion of precision.

Word segmentation strategy was directed by current work performed on natural language processing of languages from Asia region due to the fact these languages observe the lack of word separators, same like the "Cryptic" language. These techniques are explained in many research papers [14-20], and we have decided to apply techniques provided in [3] and [10] while slightly modifying these two approaches. Modifications and application of these techniques are explained in detailed fashion in Chapters 3 and 4. Word segmentation is a major part of this thesis because all predictors produced for the "Cryptic" language are relying on the precision and quality of word segmentation. From results obtained by word segmentation of user logs three types of trees are obtained. The fourth type of tree data structure, denoted as character level n-gram tree with context fixed root, is a specific shape of the character level n-gram tree with

context-free root. This specific shape is obtained by fixing the root symbol while at the sam time we have relaxed the condition of the tree depth. These data structures capture n-gram relations on the level of characters inside of word boundaries, n-gram relations between words inside of the commands, and n-gram relations between commands in the log. It is noticeable how the provided solution tries to capture as much as possible correlations between words and commands. However, this approach can be extended even further but for this considerable architectural design changes need to be performed in the whole “Cryptic” system, and these are left for further research.

Results obtained from small train corpus provided from Amadeus might be too optimistic due to the bias of the user toward shorter commands, which could indicate that this user was a novice user. This reason might indicate that statistics obtained on the other corpus could be not so high, as ACSR of 81% can be considered as high value. If the results obtained from this research could be regarded as somewhat not maximally precise, the reason of this could be found it the lack of organized design of the “Cryptic” language from the very beginning. As previously stated in the introduction, this language was created over a long period, and many commands are created by different groups of people and following different naming conventions increasing language overall complexity and randomness. On the other hand, only some parts of the language have formal grammar definitions provided, and these definitions can be added to the proposed solution. Static dictionaries and parsers are not included in the solution because the goal was to provide a generalized solution with no static subparts; these can be easily added to improve performance. One can observe that natural languages have been created over a lot larger period by a lot larger number of people. An advantage of natural languages is the presence of better quality train corpora and significant involvement of research energy. The same cannot be said for the case of the “Cryptic” language and research provided in this thesis is the first one performed on this particular language, and there is a lot of space for improvement.

9 FUTURE WORK

In this thesis, the focus was on the similarities between “Cryptic” language and Asia region languages such as Japanese, Chinese, and others. Another possible approach to the problem at hand would be to observe words of the “Cryptic” language as a compressed representation of words from the English language. This is a logical assumption since the majority, if not all, words in the “Cryptic” language are created as abbreviations of words or groups of words from the English language. If the process could be automatically reversed, this idea is motivated by [12], we could extract English words. After reversing compression, we could perform n-gram relation extraction by combining “Cryptic” logs and English language corpora already labeled and segmented that contains information of higher quality. At this point, even some notion of grammar could be extracted, and more inter-command relations could be formed. Commands that are not too far away could, in principle, share arguments.

At this stage, we have not been dealing with these relations because the flow of commands for a particular user is hard to grasp. The external execution context is needed to connect commands of individual flow since the user can switch this context at any time. The context we are referring to is execution context at the server machines; this information contains all details about a reservation that commands are manipulating with. Although this context is not present in the logs, it could be extracted from other components of the system’s logic. However, this would imply some architectural redesign which would require work that exceeds the time available for this research. This research was limited to the length of the internship in the company Amadeus, or more precisely four months.

Another direction of future work is exploring other possible segmentation techniques that are not based on mutual information or entropy. There are various other approaches one of such is proposed in [15] and it is based on the notion of the T-score of pairs or groups of characters. This particular solution does not seem exceptionally different from the approach followed in our proposition, even so it might be fruitful to invest time into this or similar approaches and investigate them more thoroughly. The extremely interesting idea would be to correlate work provided in [14], which uses n-

gram model for correlating musical compositions. We could observe each word as a tone in some imaginary notation of sounds and borrow concepts from research performed in this field.

The alternative approach could be borrowing concepts of pattern matching carried out in fields dealing with DNA analysis, such as [22] and [23]. In our case, even techniques that might be computationally infeasible for DNA analysis might scale well due to the fact that DNA chains are considerably longer than the average commands of “Cryptic” language.

It is worth mentioning again that the main direction of future work will be decompression techniques. Once decompression, if proven possible, is performed we could continue down the line of extracting and formalizing grammar of the “Cryptic” language based on the decompressed language. Language grammar could be extracted by using the English language labeled corpora and defining words in the same manner as it has been done in natural language (e.g. nouns, verbs, adverbs). In addition to this we could form classes of words based on n-gram relations in a similar manner as it has been done in [11]. Additionally, future requirement is to introduce update techniques that will adapt data structures proposed in this solution at runtime. These techniques must be concerned with the fact that adding counts to one node at level n will affect n-gram estimators at any level larger or equal to n. This implies that runtime update might be too expensive since it would update large portion of the trees used for auto-complete and auto-correct. Future work on this particular part of the system will be concerned with providing fast and flexible algorithms that can handle runtime updates.

It is noticeable that there are many possible ways to improve the current solution, and this comes from the fact that this is the first research performed on this particular problem. Thus, research focused on the “Cryptic” language inside Amadeus Corporation is still at early stages and there is much space for improvement. Additional work on the topic of the “Cryptic” language is planned, and it will be conducted in cooperation with Amadeus in the following years.

10 REFERENCES

1. “Zipf's word frequency law in natural language: a critical review and future directions”, Steven T. Piantadosi, June 2, 2015.
2. “Elements of information theory”, Thomas M. Cover and Joy A. Thomas, 1991.
3. “A Boundary-Oriented Chinese Segmentation Method Using N-Gram Mutual Information”, Ling-Xiang Tang, Shlomo Geva, Andrew Trotman and Yue Xu, 2010.
4. “An Introduction to Hidden Markov Models”, L. R. Rabiner and B. H. Juang, 1989.
5. “Introduction to Probability”, Charles M. Grinstead and J. Laurie Snell, 1998.
6. “Human Behaviour and the Principle of Least Effort”, G. K. Zipf, 1949.
7. “On the Applicability of Zipf’s Law in Chinese Word Frequency Distribution”, H. Xiao, 2008.
8. “Exponential Moving Average versus Moving Exponential Average”, F. Klinker, 2011.
9. “Arithmetic versus Geometric Mean Estimators: Setting Discount Rates for Capital Budgeting”, I. Cooper, 1996.
10. “Entropy as an Indicator of Context Boundaries – An Experiment Using a Web Search Engine”, K. Tanaka-Ishii, 2005.
11. “Class-Based n-gram Models of Natural Languages”, P. F. Brown, P. V. de Souza, R. L. Mercer, V. J. Della Pietra and J. C. Lai, 1992.
12. “A Ngram-based Statistical Machine Translation Approach for Text Normalization on Chat-speak Style Communication”, C. A. Henriquez Q., A. Hernandez H., 2009.
13. “A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases for Large Text Data of Japanese”, M. Nagao and S. Mori, 1994.
14. “N-gram Based Statistical Makam Detection on Makam Music in Turkey Using Symbolic Data”, E.Unal, B. Bozkurt, and M. K. Karaosmanoglu, 2012.
15. “Chinese Word Segmentation without Using Lexicon and Hand-crafted Training Data”, S. Maosong, S. Dayang and B. K. Tsou, 1998.

16. “An Improved Succinct Representation for Dynamic k-arty Trees”, D. Arroyuelo, 2008.
17. “Computing N-gram Statistics in MapReduce”, K. Berberich and S. Bedathur, 2013.
18. “An Empirical Comparison of Goodness Measures for Unsupervised Chinese Word Segmentation with a Unified Framework”, H. Zhao and C. Kit, 2008.
19. “Data-driven Language Independent Word Segmentation Using Character-Level Information”, D. H. Lim and S. S. Kang, 2006.
20. “Incremental Chinese Lexicon Extraction with Minimal Resources on a Domain-Specific Corpus”, G. Patin, 2010.
21. “Research of the FP-Growth Algorithm Based on Cloud Environments”, L. Zhou and X. Wang, 2014.
22. “An Efficient Matching Algorithm for Encoded DNA Sequences and Binary Strings”, S. Faro and T. Lecroq, 2009.
23. “Exact Multiple Pattern Matching Algorithm using DNA Sequences and Pattern Pair”, R. Bhukya and D.V.L.N. Somayajulu, 2011.

R: RESULTS

- $BC_{ea}(L|R; n, m, w[k])$

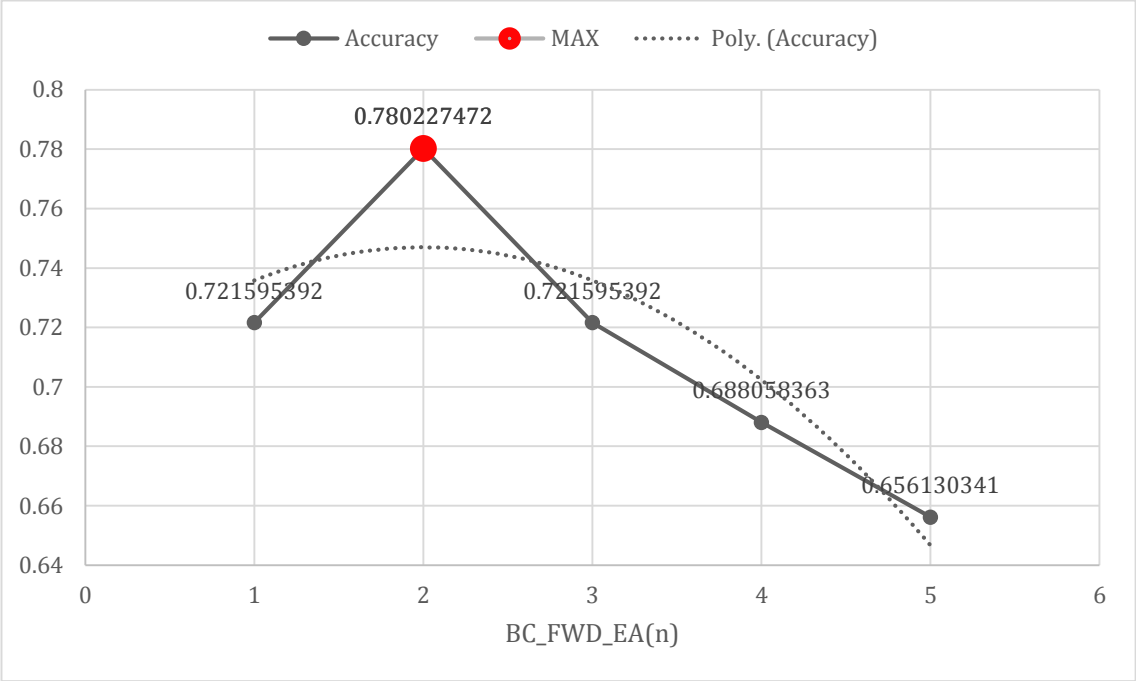

Figure 10.1: Accuracy measurement of $BC_{ea}(L|R; n, m, w[k])$

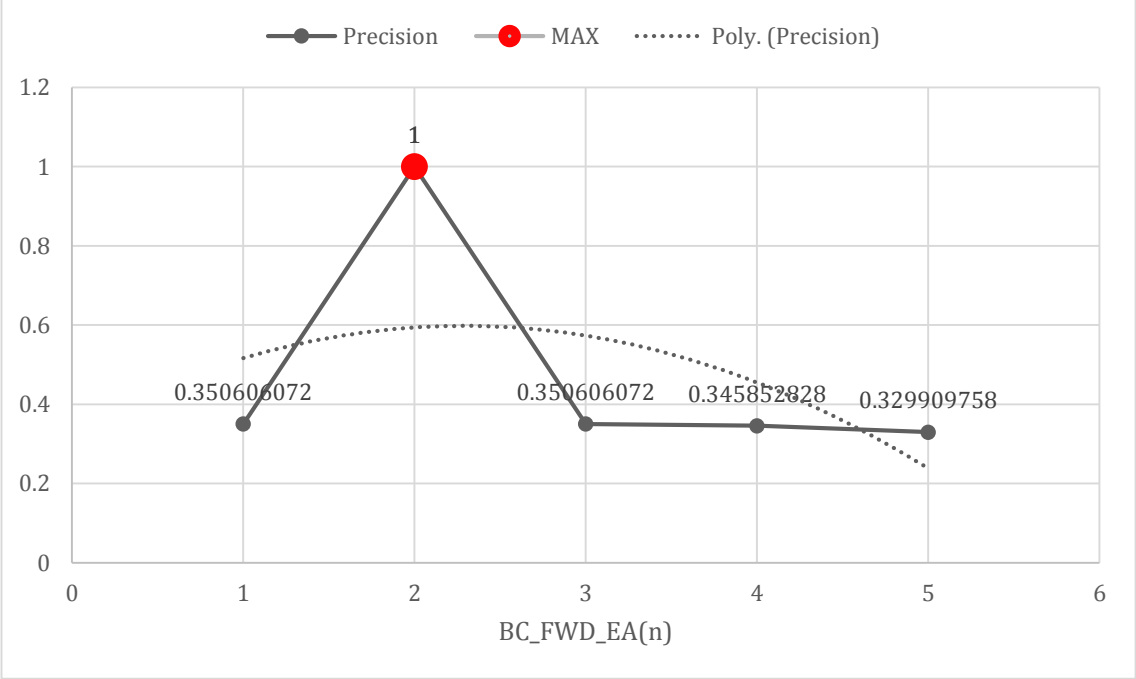

Figure 10.2: Precision measurement of $BC_{ea}(L|R; n, m, w[k])$

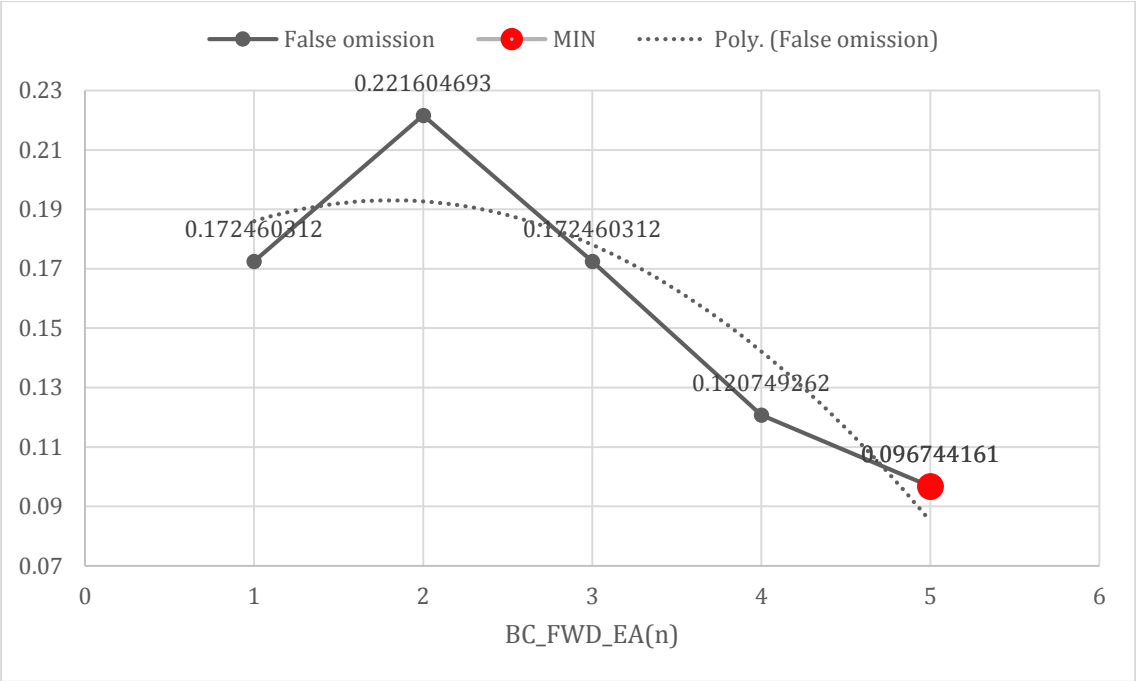

Figure 10.3: False omission ratio measurement of $BC_{ea}(L|R;n,m,w[k])$

We select parameter n mainly on value of false omission ratio, and only if there is no improvement in this value that we consult accuracy and precision respectfully. Here minimal value for false omission ratio is observed for $n = 5$ and in accordance with this observation value is set to 5.

- Discount parameter for exponential average

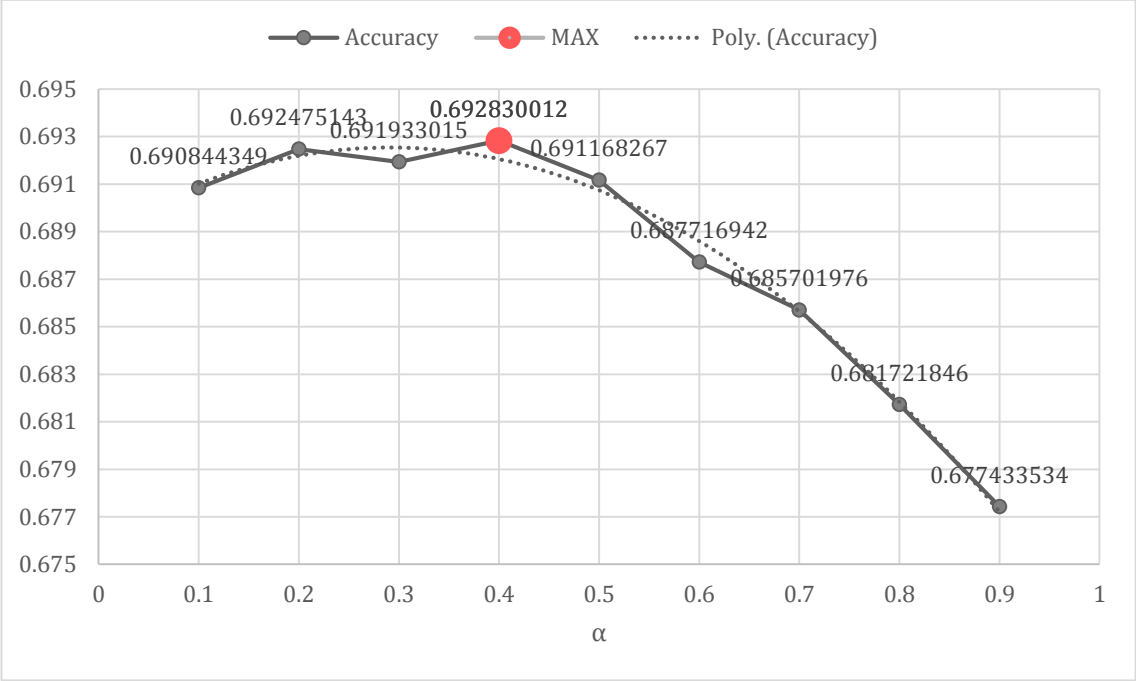

Figure 10.4: Accuracy measurement of $BC_{ea}(L|R;n,m,w[k])$ with varying value of discount parameter

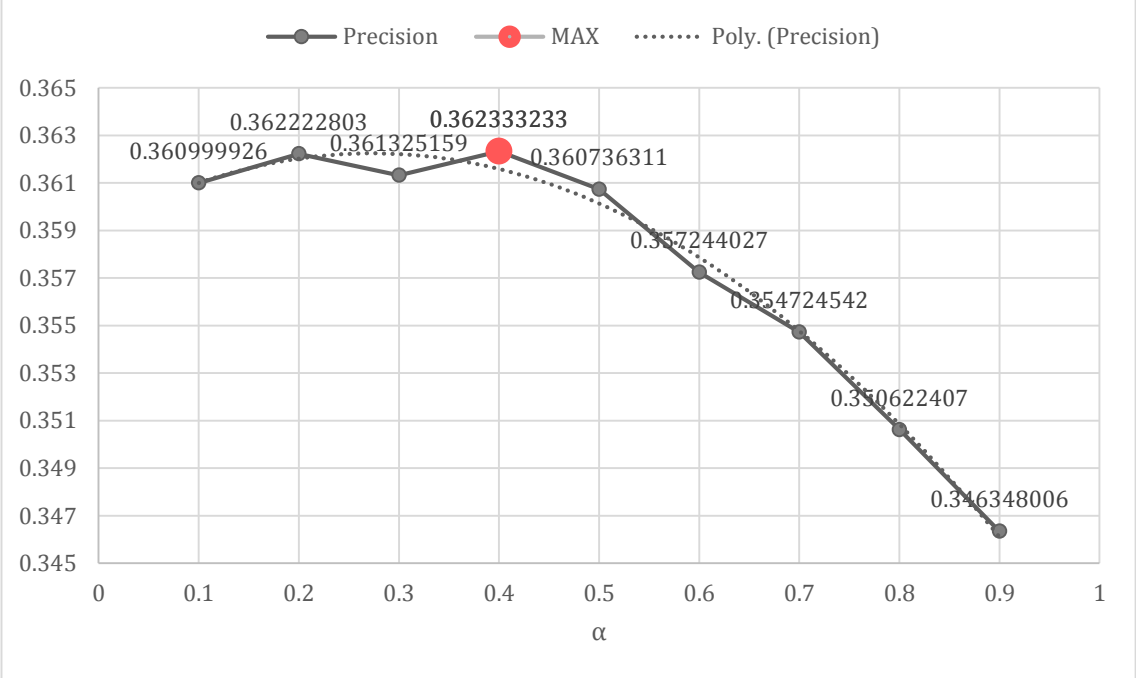

Figure 10.5: Precision measurement of $BC_{ea}(L|R;n,m,w[k])$ with varying value of discount parameter

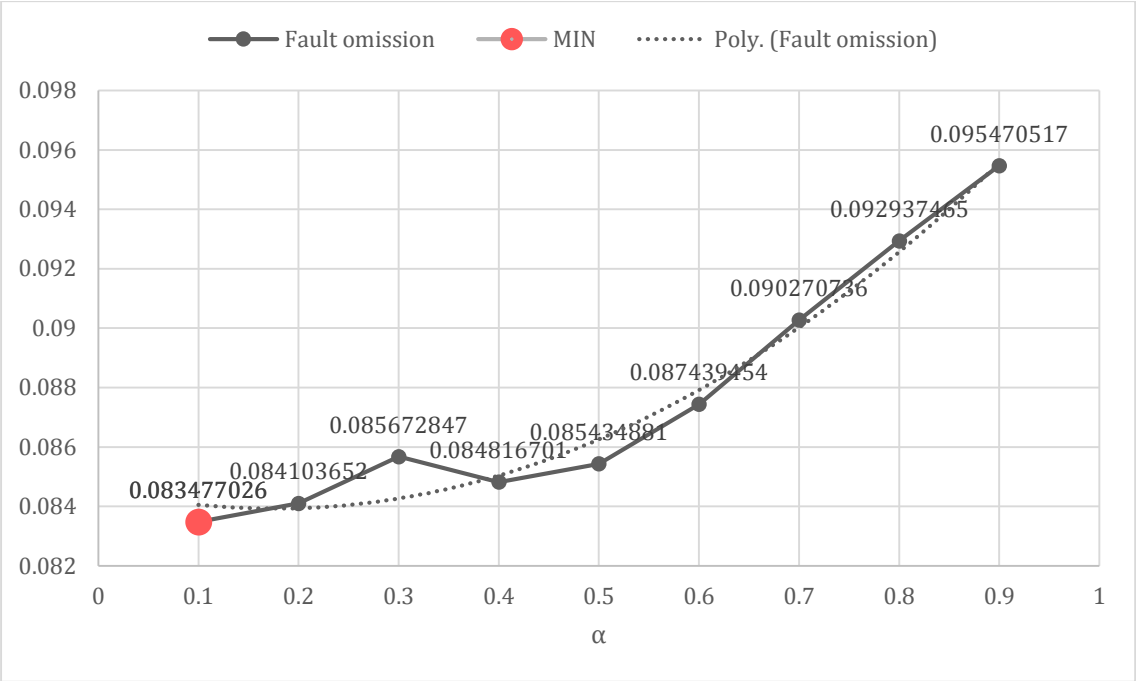

Figure 10.6: False omission ratio measurement of $BC_{ea}(L|R;n,m,w[k])$ with varying value of discount parameter

We have run statics for choosing discount parameter only on $BC_{ea}(L|R;n,m,w[k])$ and applied obtained value for all exponential average procedures. Chosen value is $\alpha = 0.4$, and even though for this discount value we do not observe minimum for false omission ratio we are in vicinity of minimum while accuracy and precision values have both observed their respectful maximums in this statistical point.

- $BC_{wa}(L|R; n, m, w[k])$

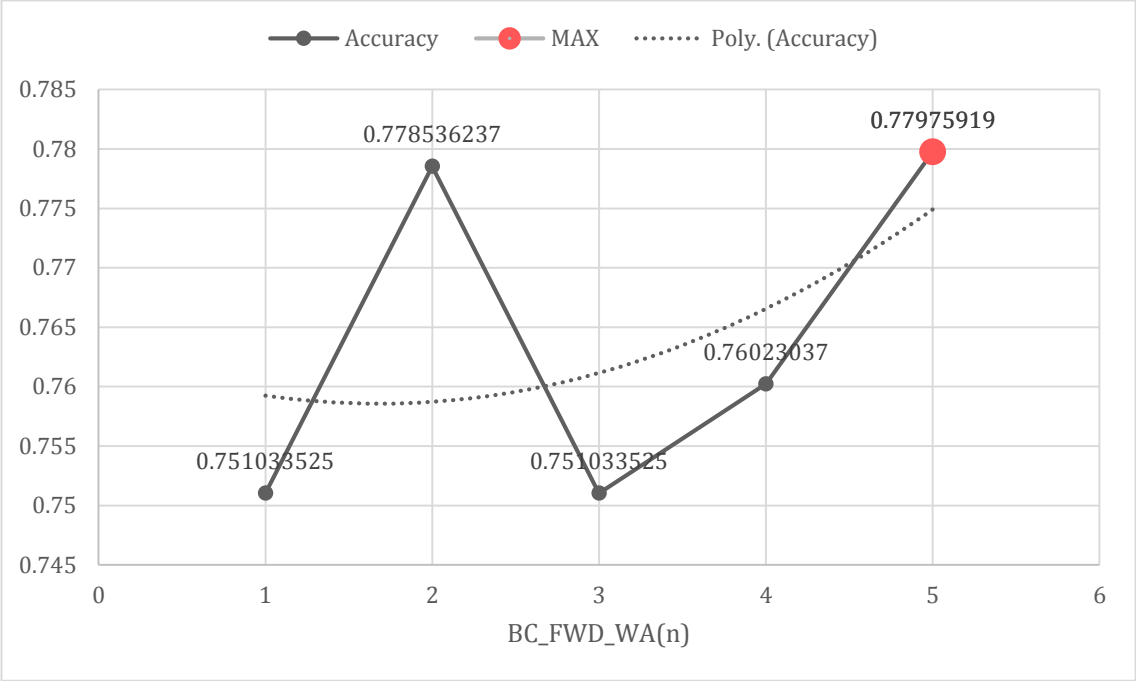

Figure 10.7: Accuracy measurement of $BC_{wa}(L|R; n, m, w[k])$

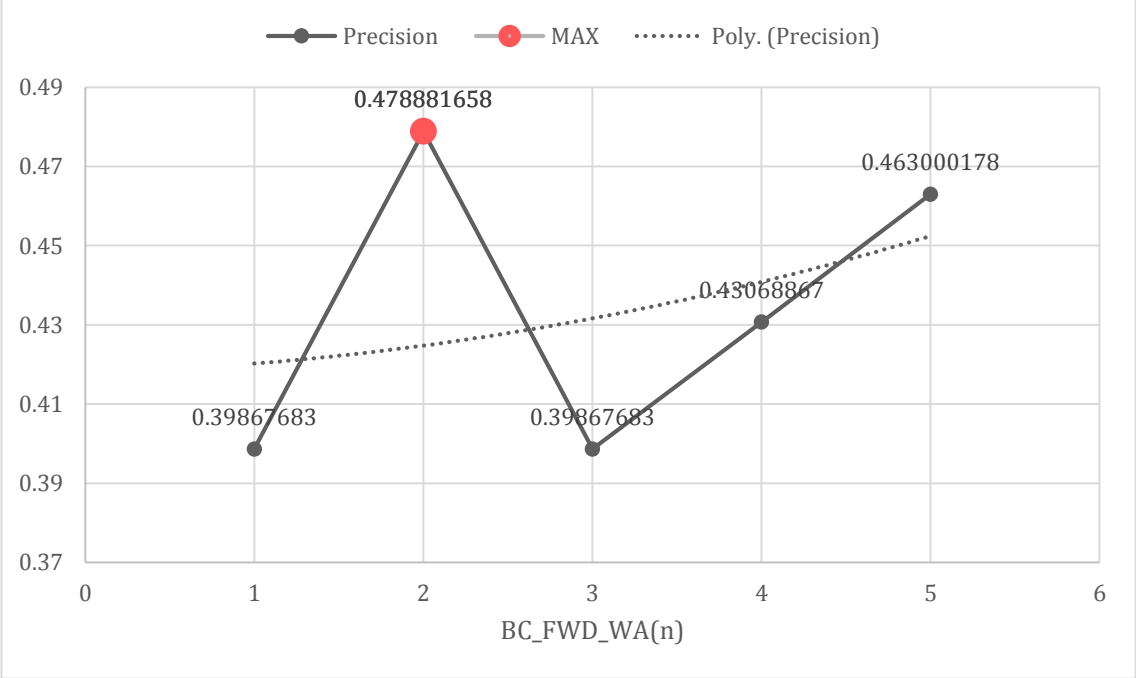

Figure 10.8: Precision measurement of $BC_{wa}(L|R; n, m, w[k])$

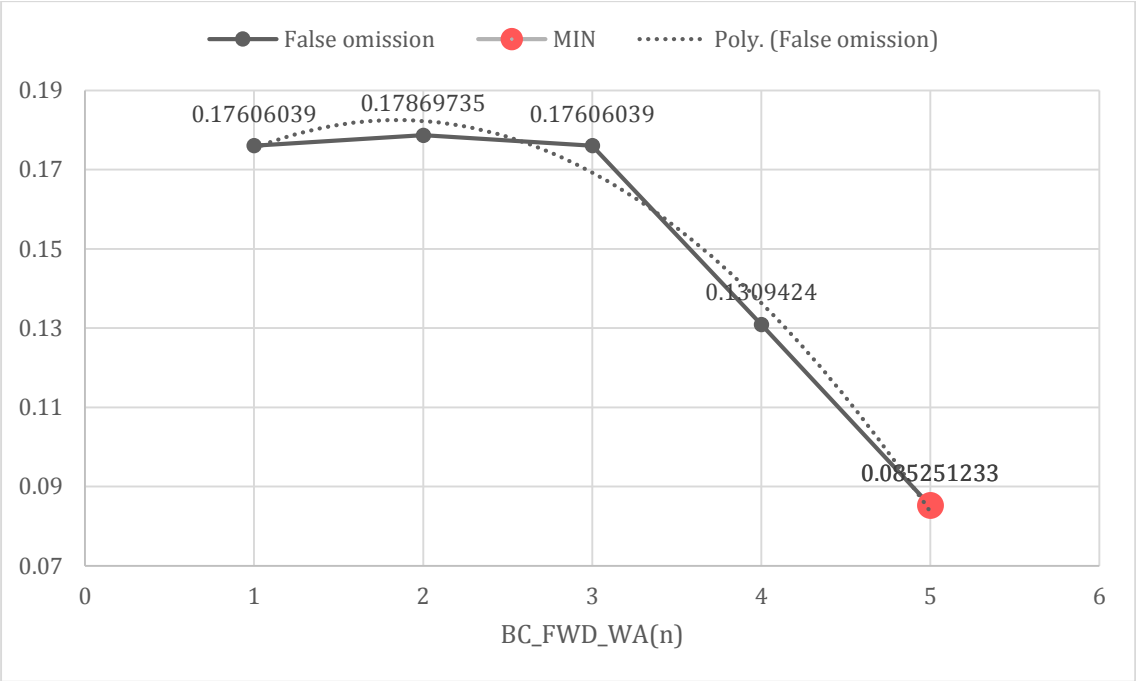

Figure 10.9: False omission ratio measurement of $\mathbf{BC_{wa}(L|R; n, m, w[k])}$

Selected value for parameter n is 5 due to the fact that false omission ratio observes its minimum in this point, while precision observes its maximum in same point. Accuracy in this case is not maximized but it is in close vicinity of maximum value ergo $n = 5$ is a reasonable choice.

- $EC_{bea}(k; w[k])$

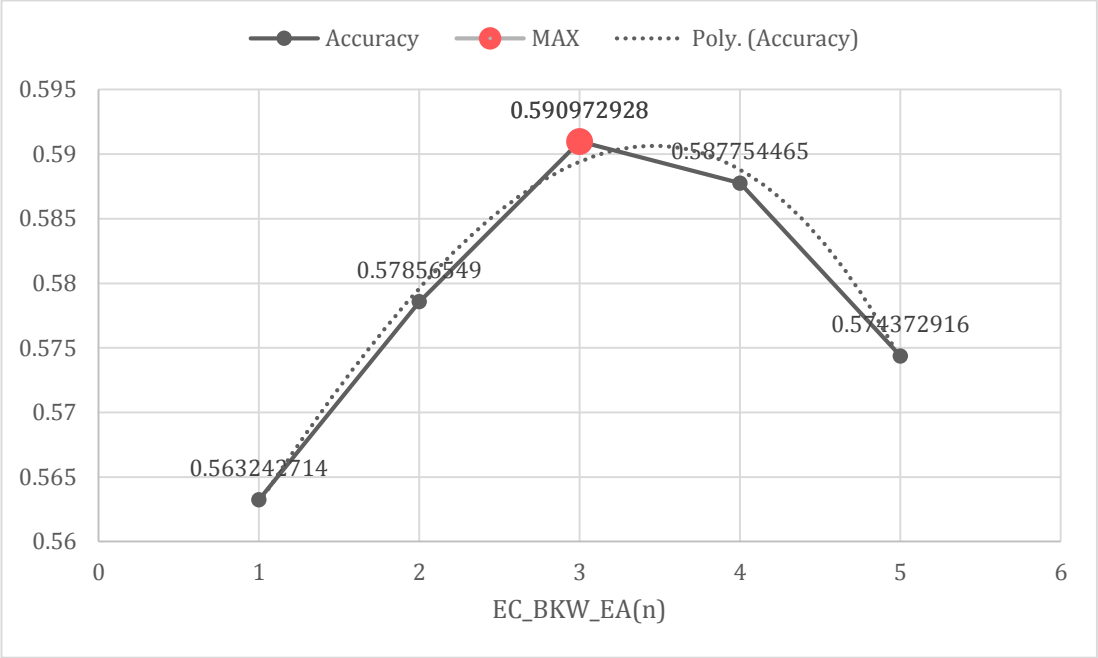

Figure 10.10: Accuracy measurement of $EC_{bea}(k; w[k])$

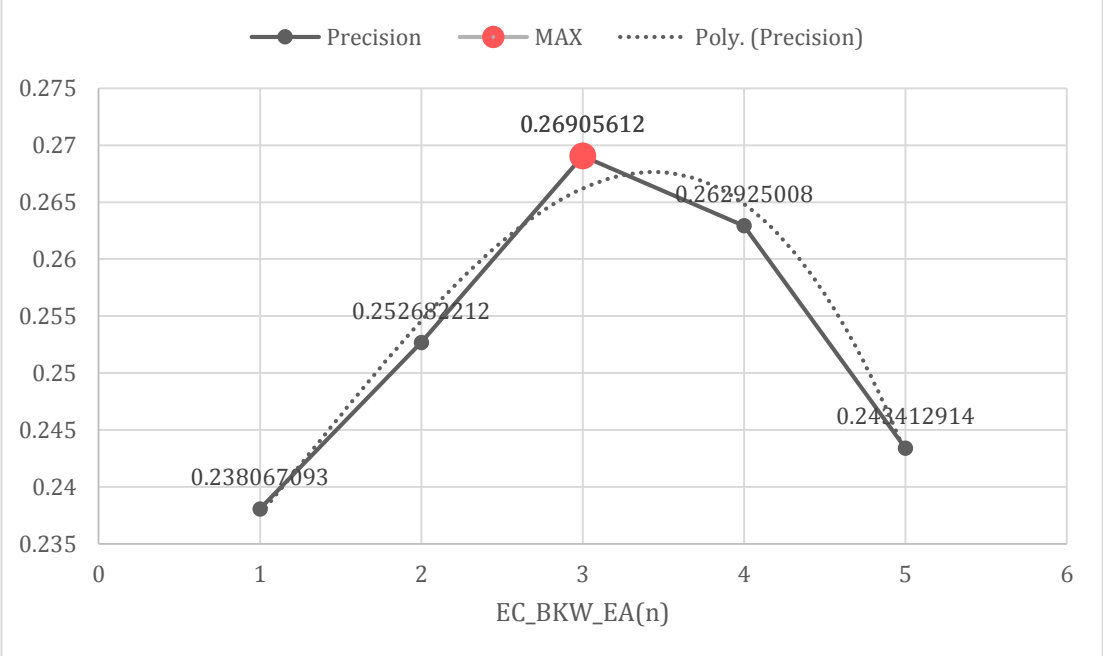

Figure 10.11: Precision measurement of $EC_{bea}(k; w[k])$

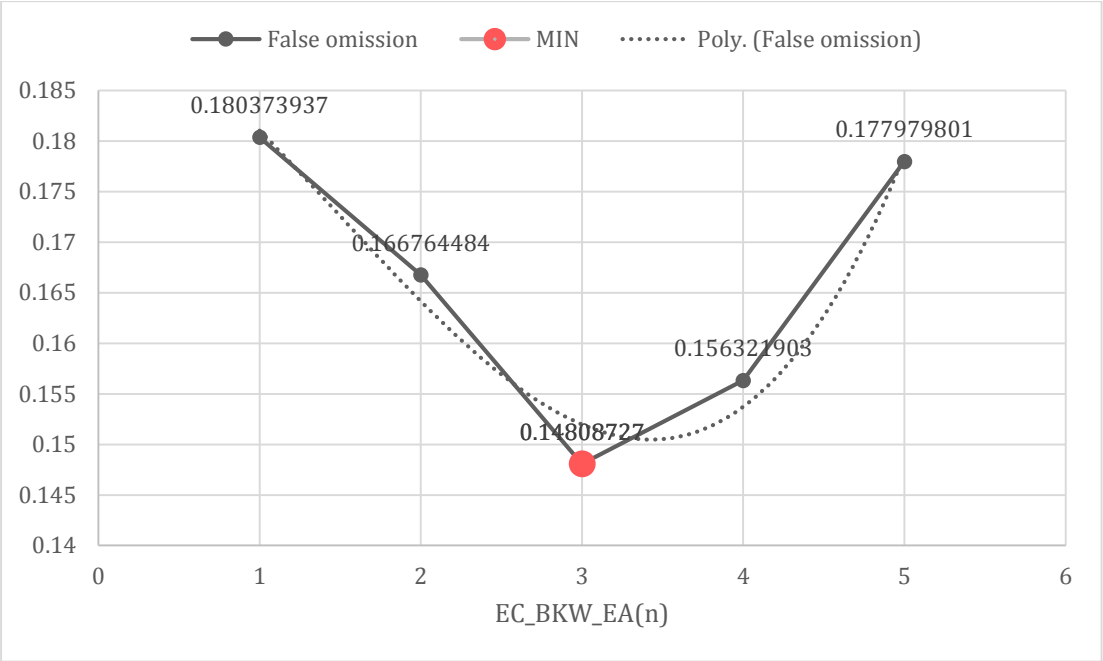

Figure 10.12: False omission ratio measurement of $EC_{bea}(k; w[k])$

In case of this particular criteria only logical choices for parameter is $n = 3$, all three measurements observe their local extremes and ergo this is clearly the best possible choice in observed data.

- $EC_{bwa}(k; w[k])$

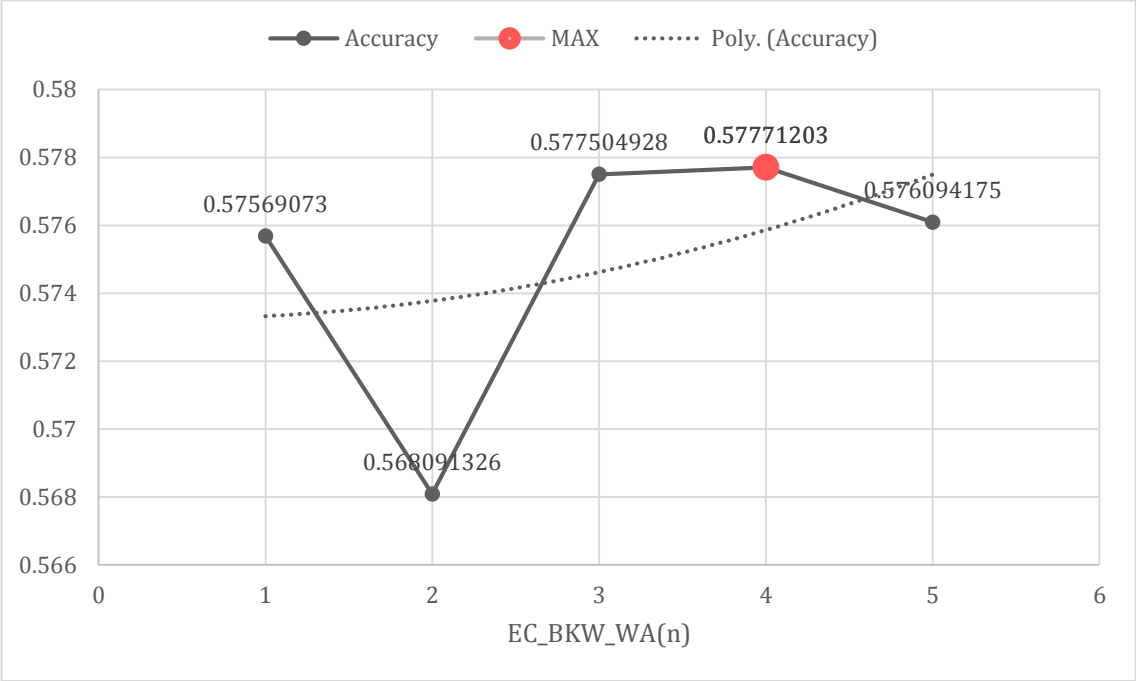

Figure 10.13: Accuracy measurement of $EC_{bwa}(k; w[k])$

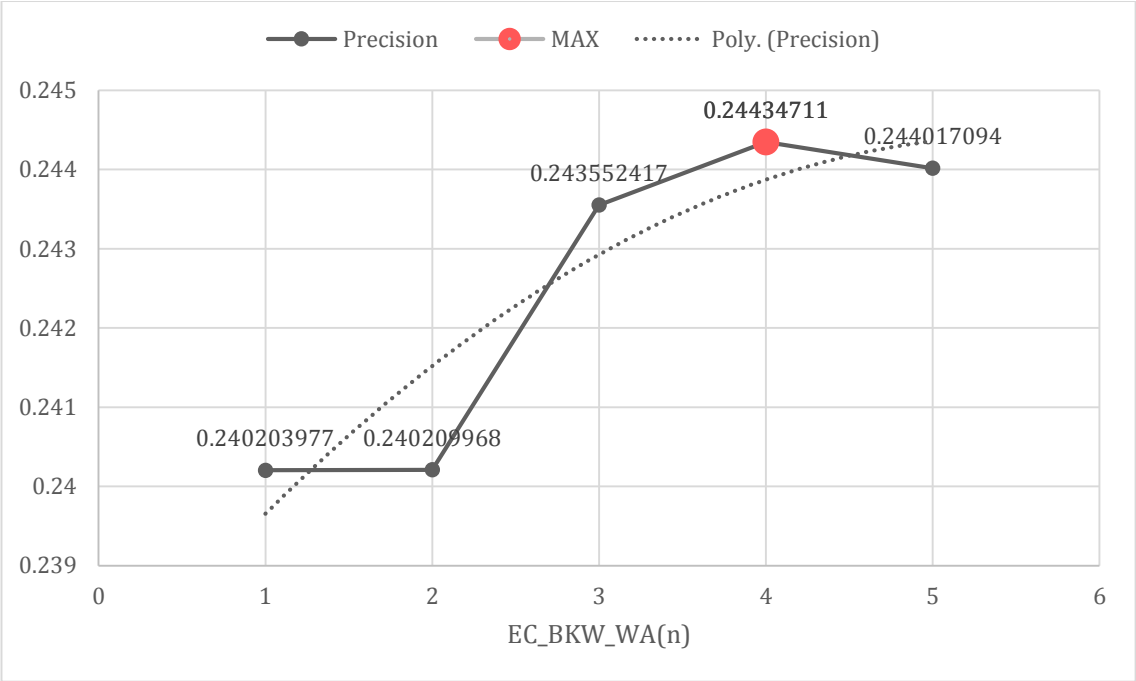

Figure 10.14: Precision measurement of $EC_{bwa}(k; w[k])$

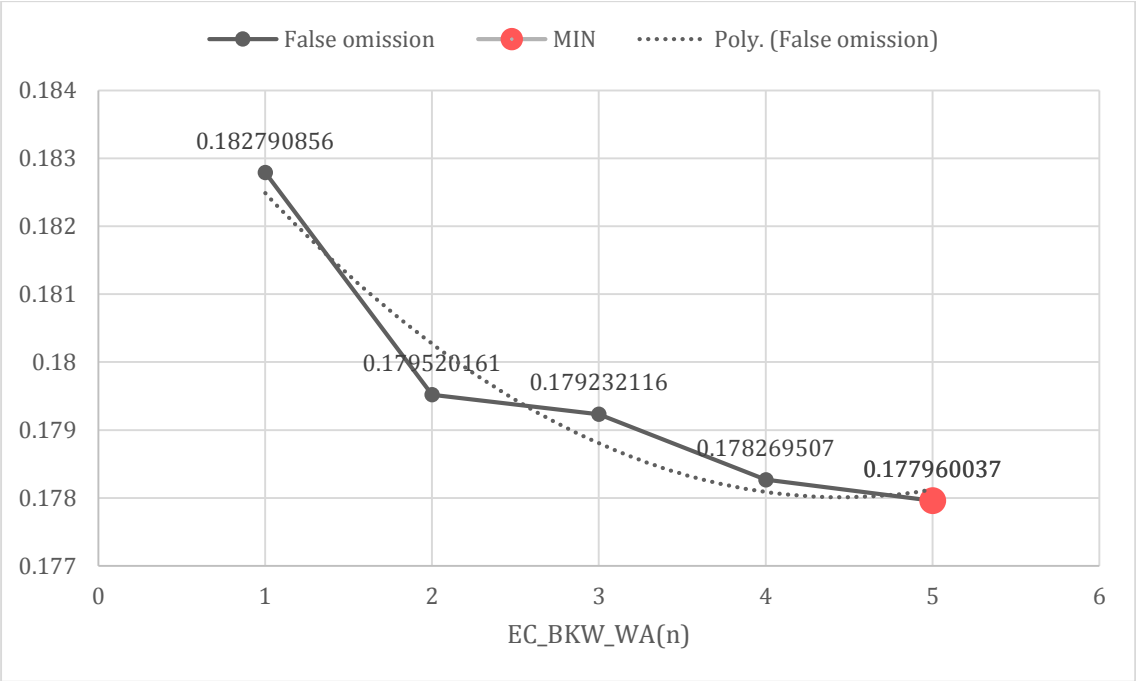

Figure 10.15: False omission measurement of $EC_{bwa}(k; w[k])$

In case of this particular criteria we observe local extremes for both accuracy and precision for value $n = 4$ while for false omission ratio minimum is obtained for value $n = 5$. False omission ratio observes stable value for $n\epsilon[2,4]$ and thus we choose parameter $n = 4$. This way we balance maximization of accuracy and precision with minimization of false omission ratio.

- $EC_{ea}(k; w[k])$

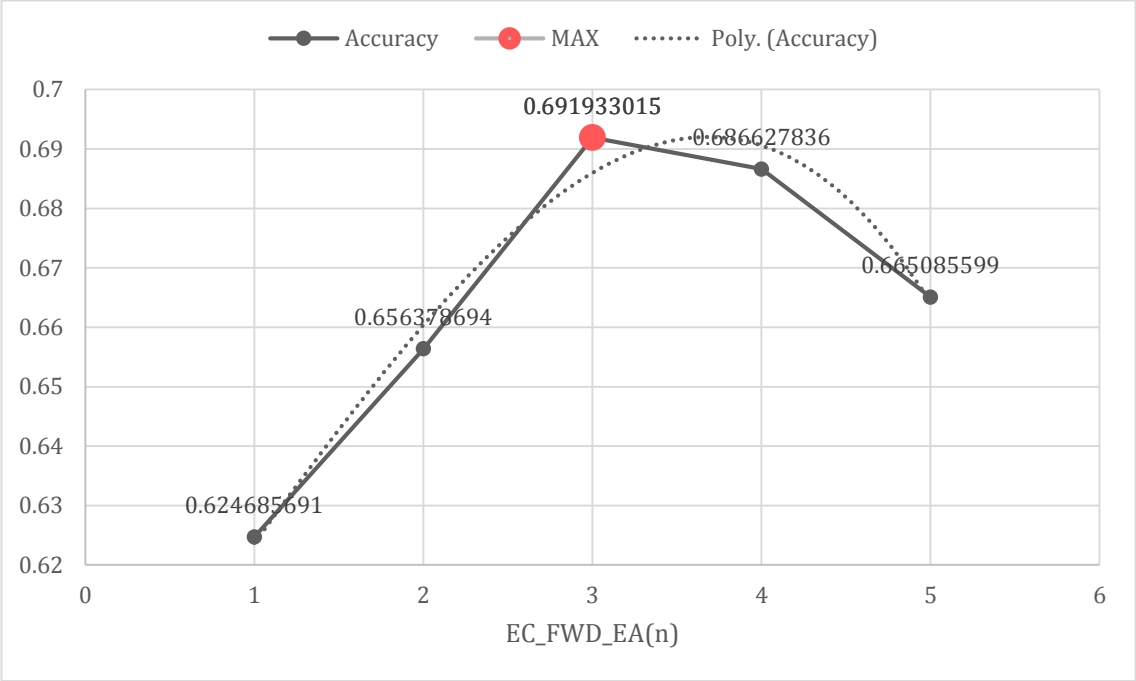

Figure 10.16: Accuracy measurement of $EC_{ea}(k; w[k])$

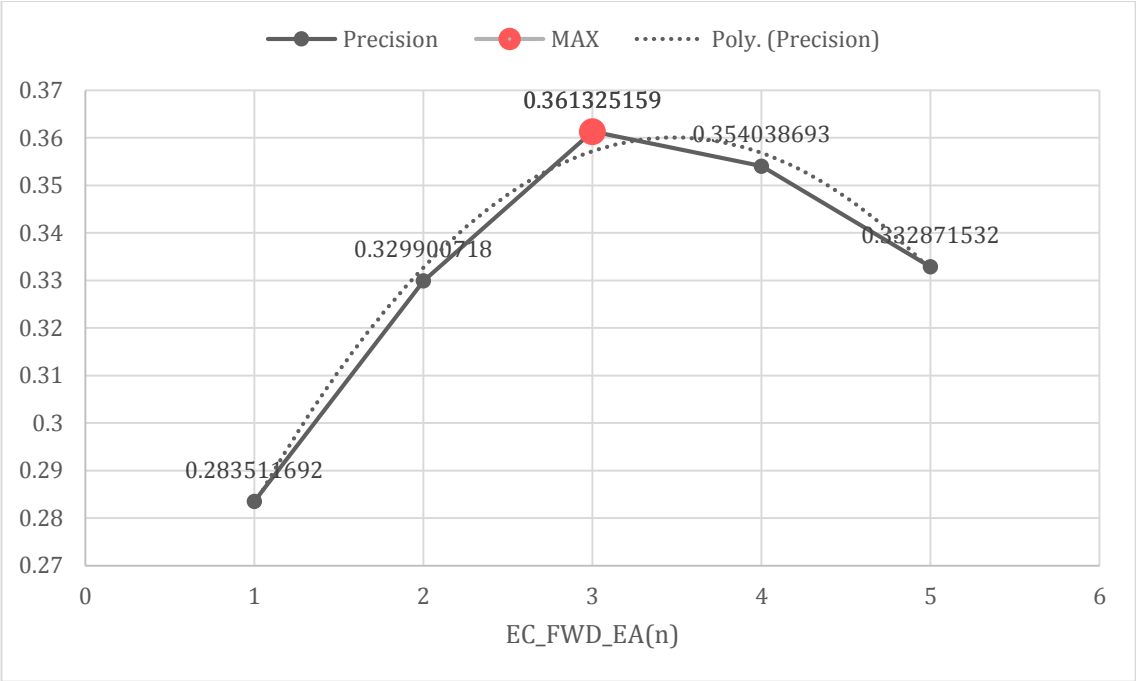

Figure 10.17: Precision measurement of $EC_{ea}(k; w[k])$

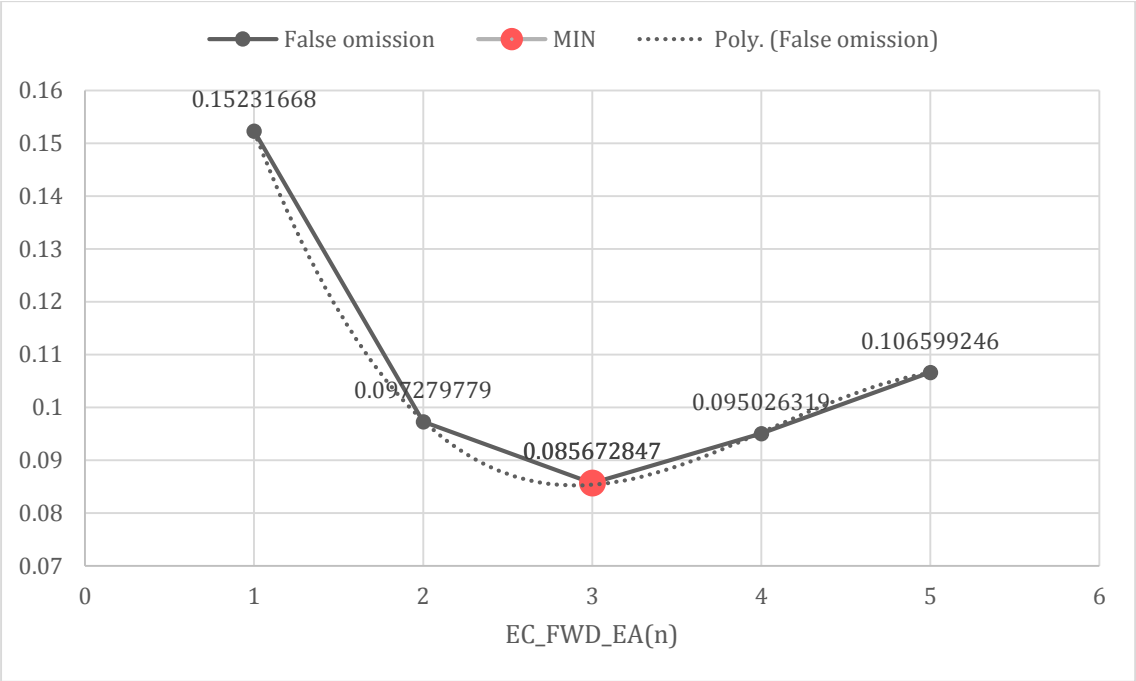

Figure 10.18: False omission ratio measurement of $EC_{ea}(k; w[k])$

In case of this particular criteria, same as it was the case with $EC_{bea}(k; w[k])$, only logical choices for parameter is $n = 3$, all three measurements observe their local extremes and ergo this is clearly the best possible choice in observed data.

- $EC_{wa}(k; w[k])$

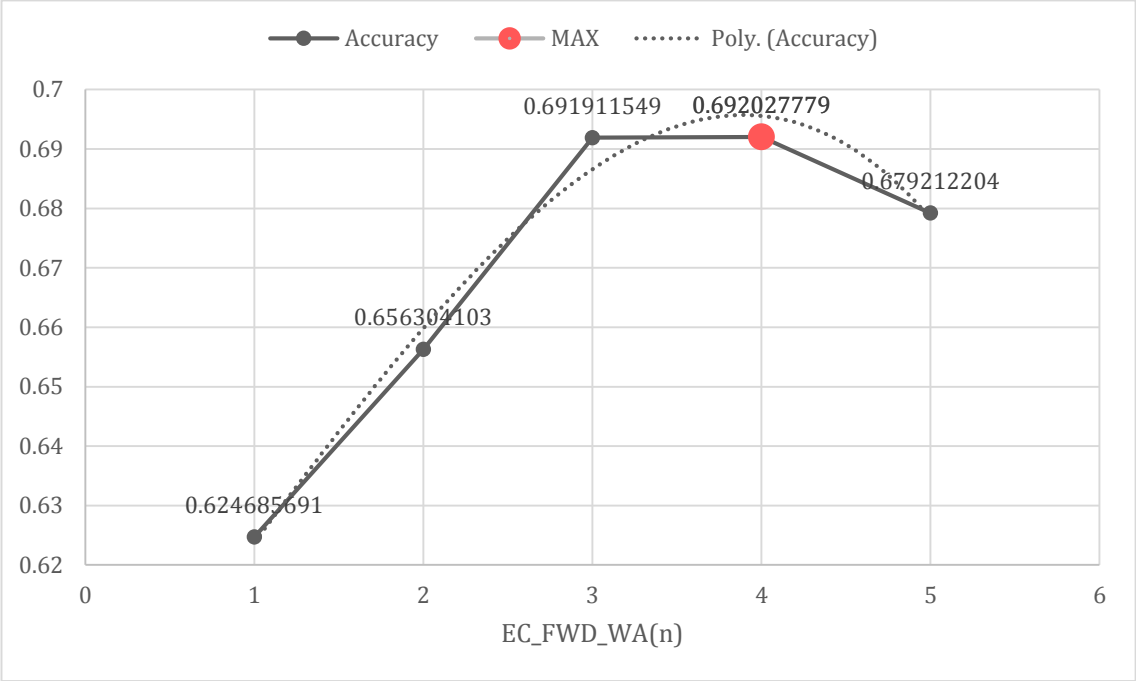

Figure 10.19: Accuracy measurement of $EC_{wa}(k; w[k])$

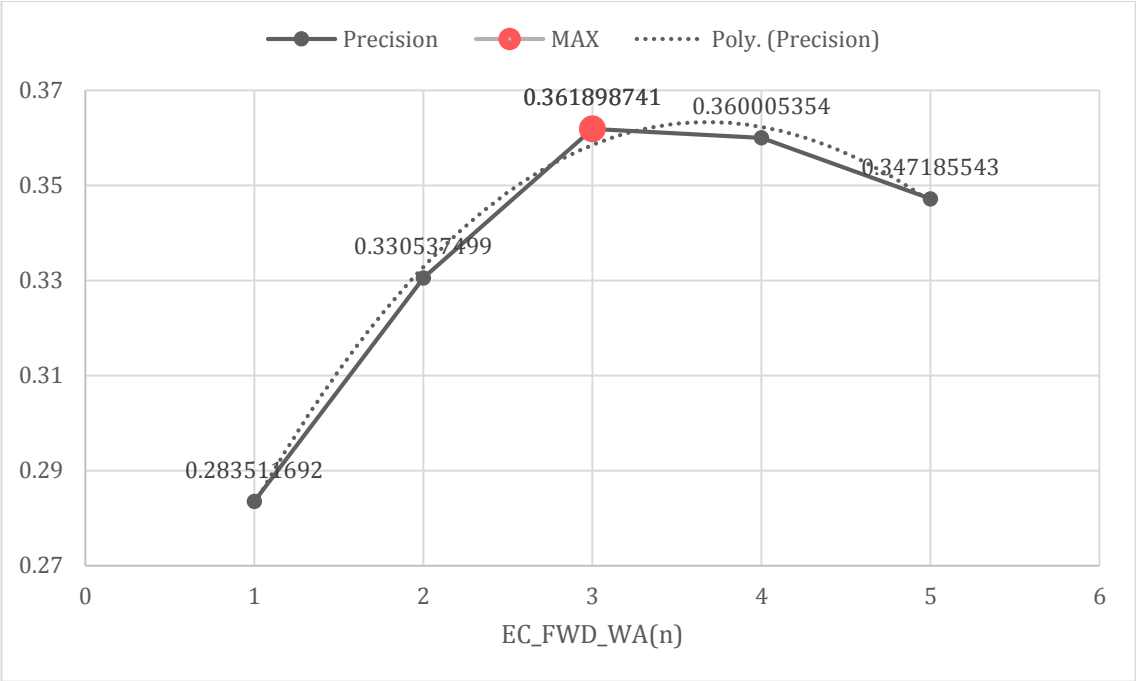

Figure 10.20: Precision measurement of $EC_{wa}(k; w[k])$

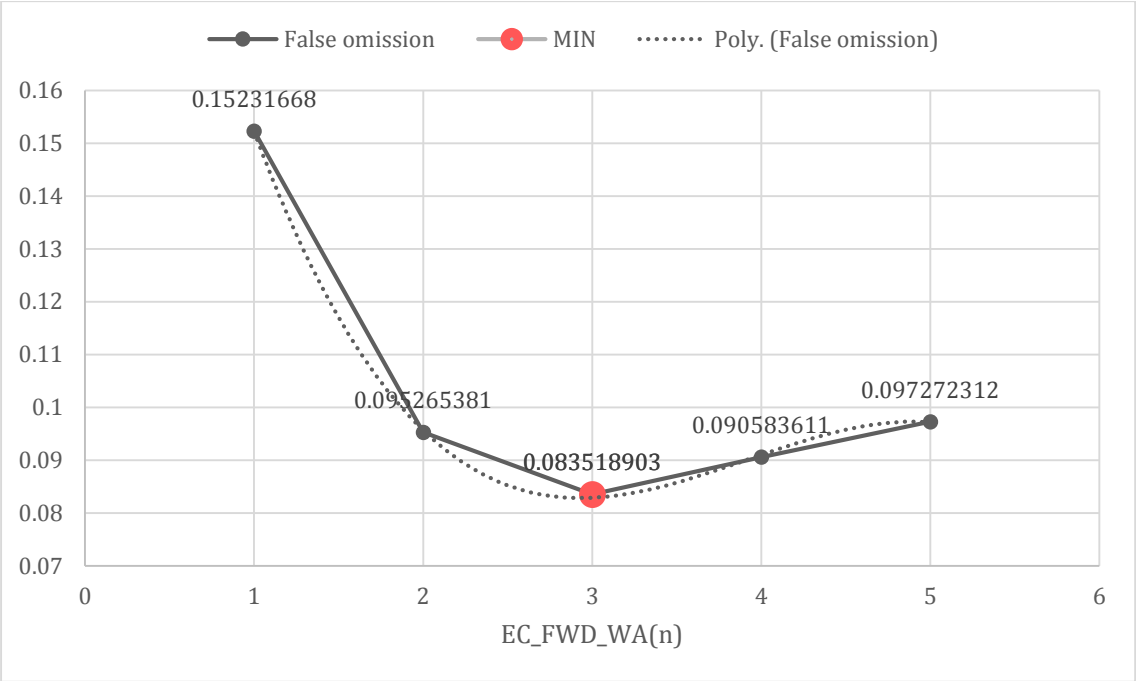

Figure 10.21: False omission ratio measurement of $EC_{wa}(k; w[k])$

Here we observe more or less same situation as it was the case with $EC_{ea}(k; w[k])$, only difference is that the accuracy is not precisely at its local extreme but it is in its close vicinity while other two measurements are at their local extremes at $n = 3$, thus making this a valid choice of parameter value.

- Comparison individual criteria

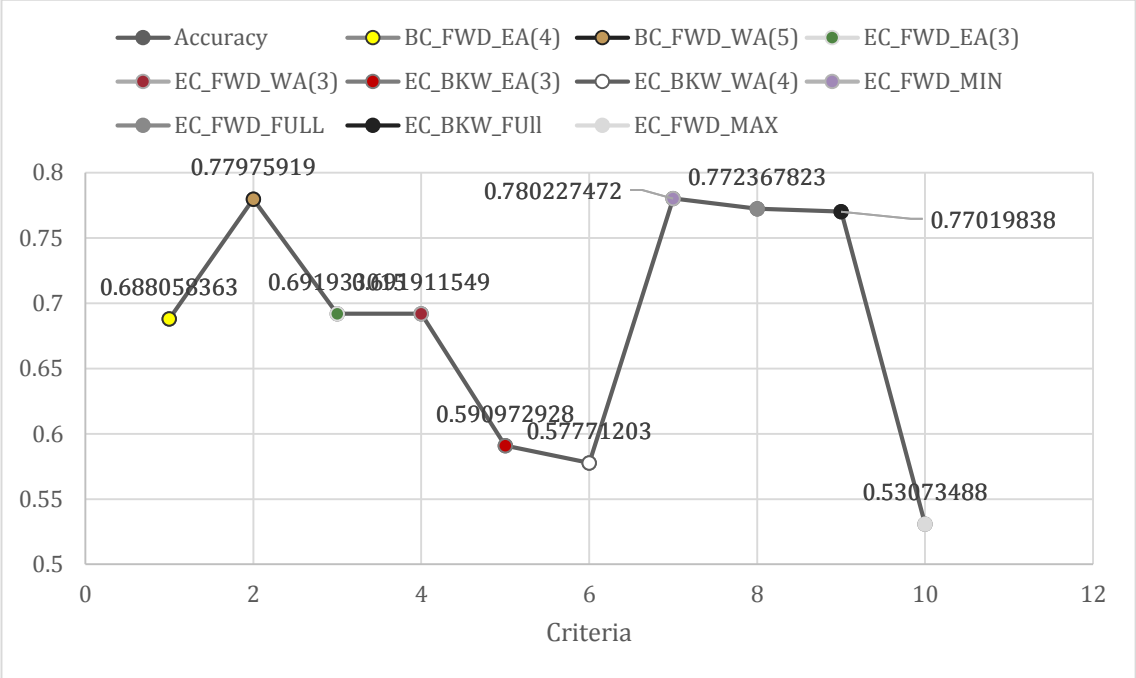

Figure 10.22: Accuracy measurement comparison between all individual criteria

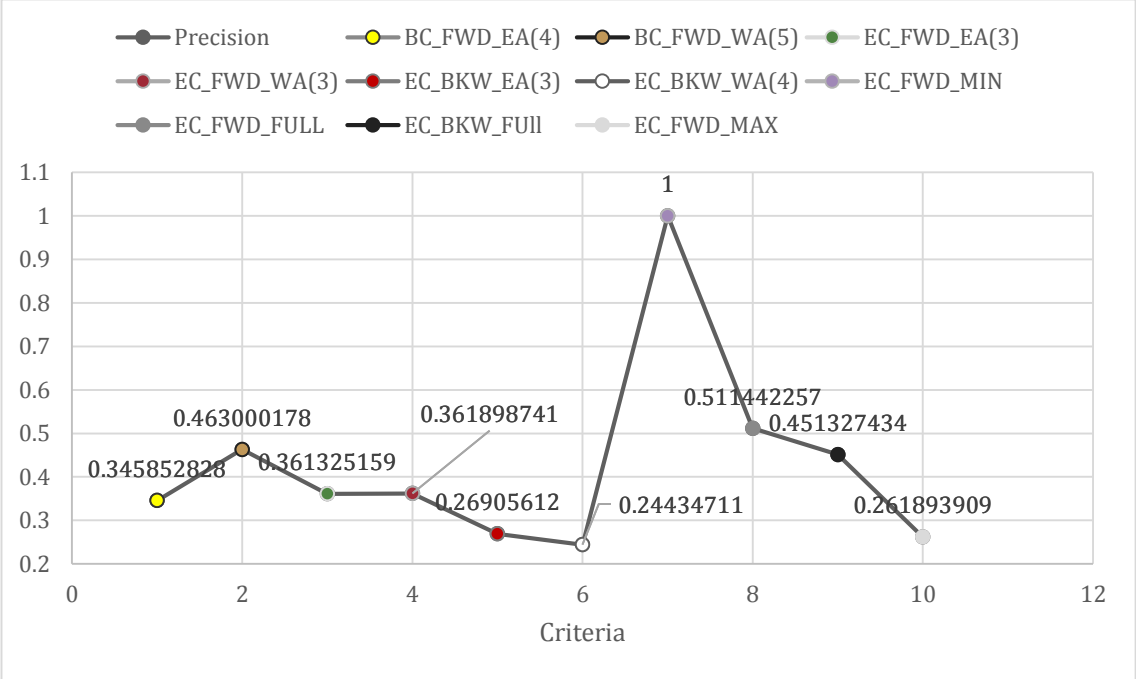

Figure 10.23: Precision measurement comparison between all individual criteria

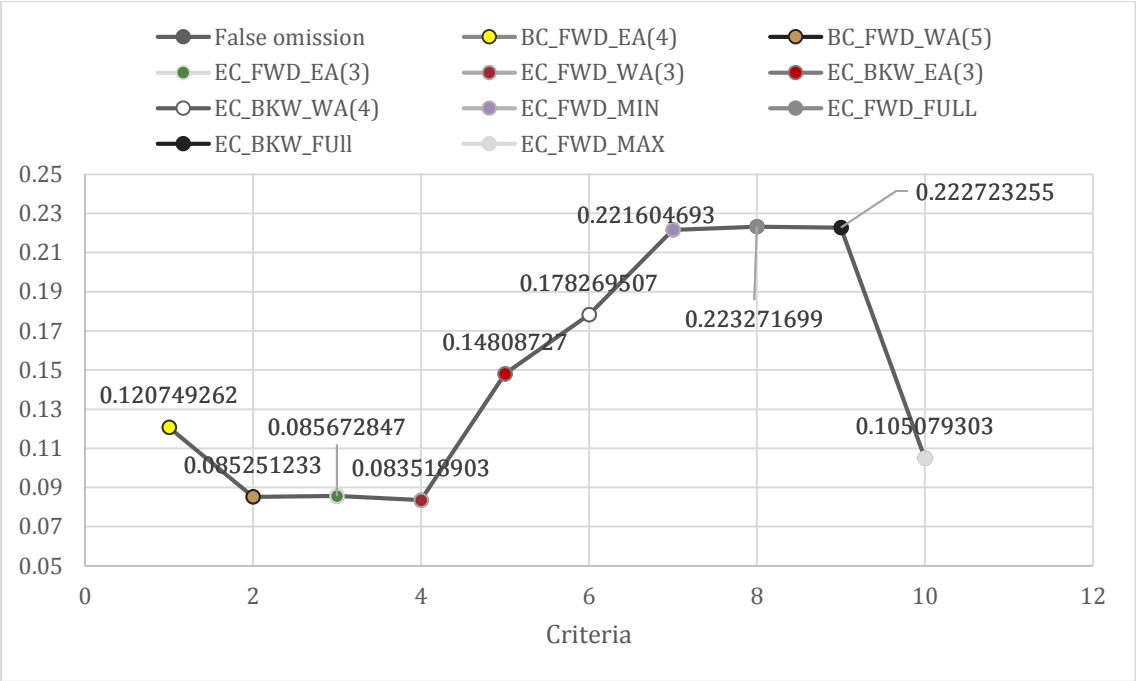

Figure 10.24: False omission ratio measurement comparison between all individual criteria

These three graphs represent comparisons of all three measurements between each criterion with parameters selected in accordance with data presented in this appendix. These graphs were used as intuition to form ranking criteria in ranked voting procedure.

- Voting procedures comparison

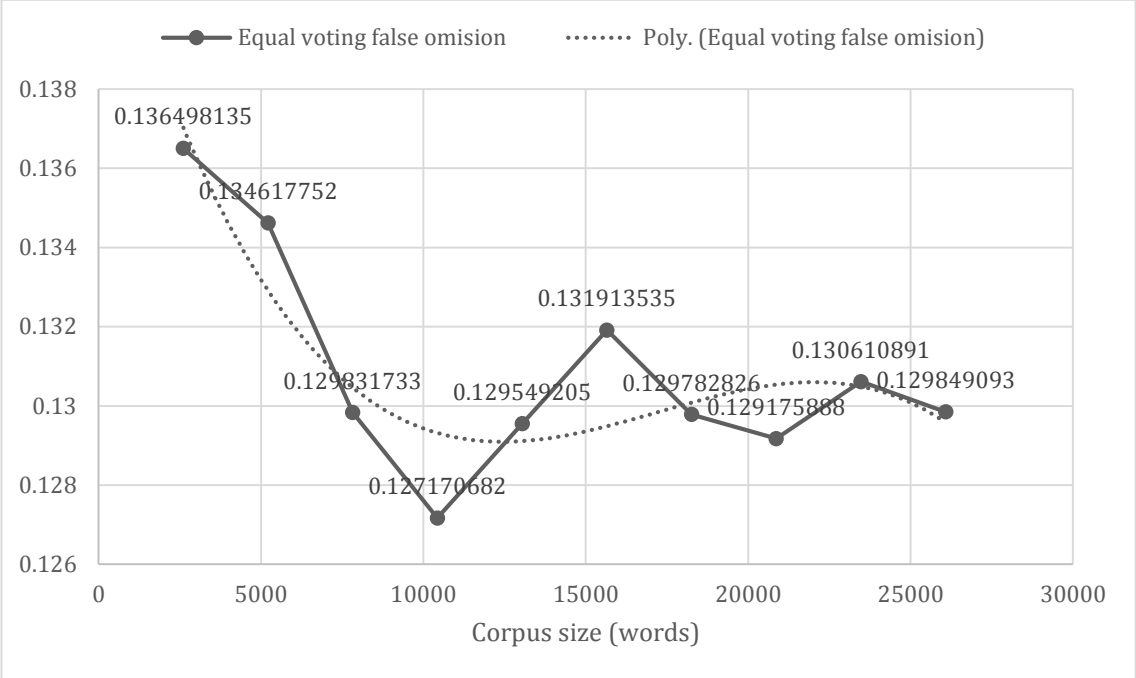

Figure 10.25: False omission ratio measurement of equal voting procedure

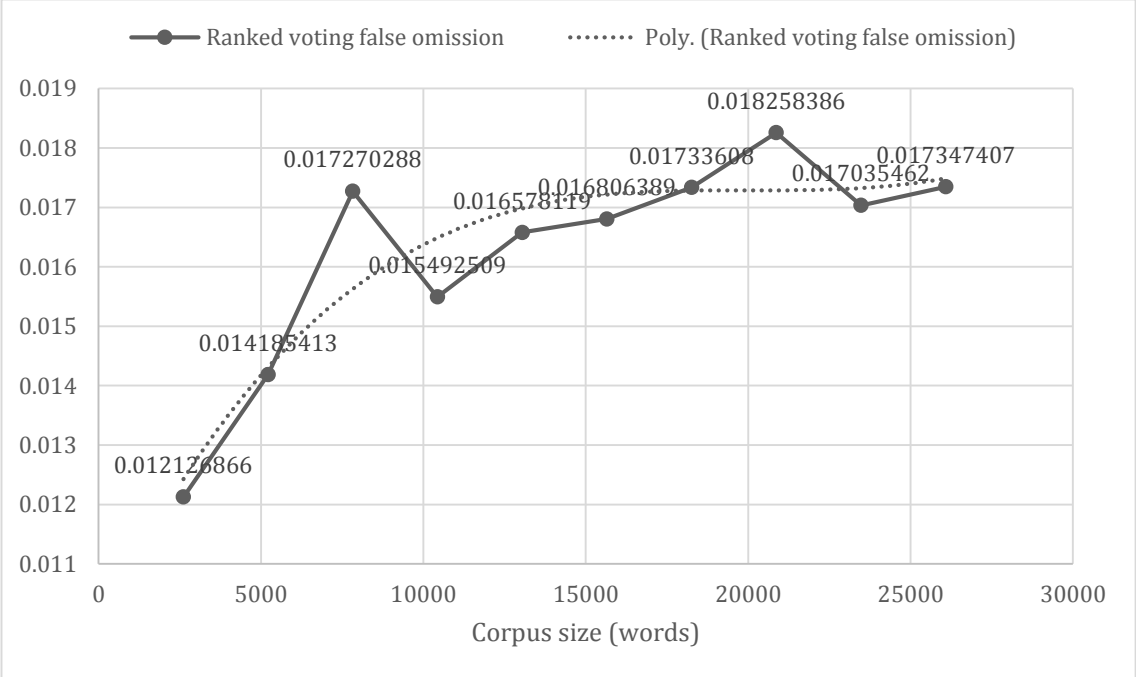

Figure 10.26: False omission ratio measurement of ranked voting procedure

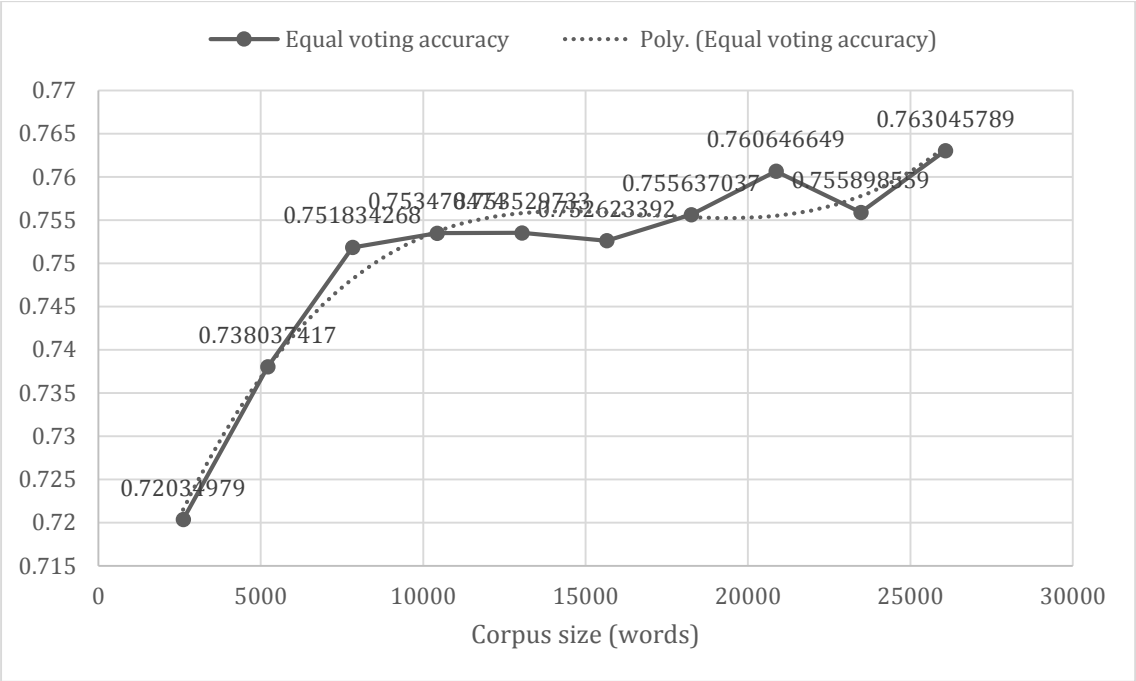

Figure 10.27: Accuracy measurement of equal voting procedure

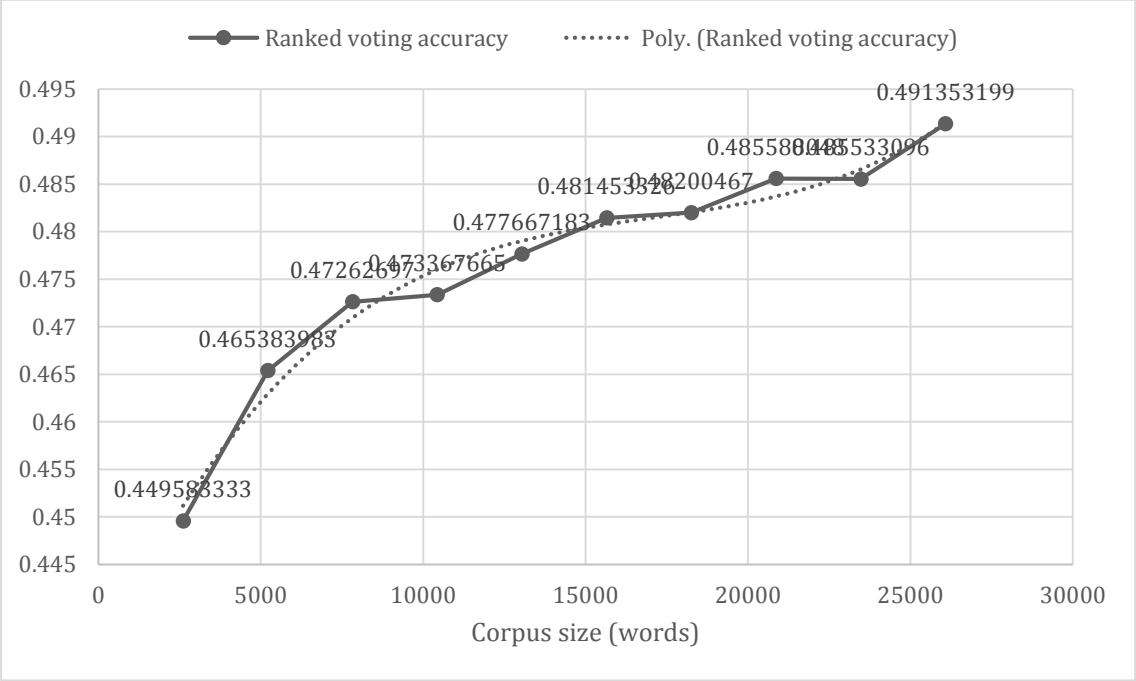

Figure 10.28: Accuracy measurement of equal voting procedure

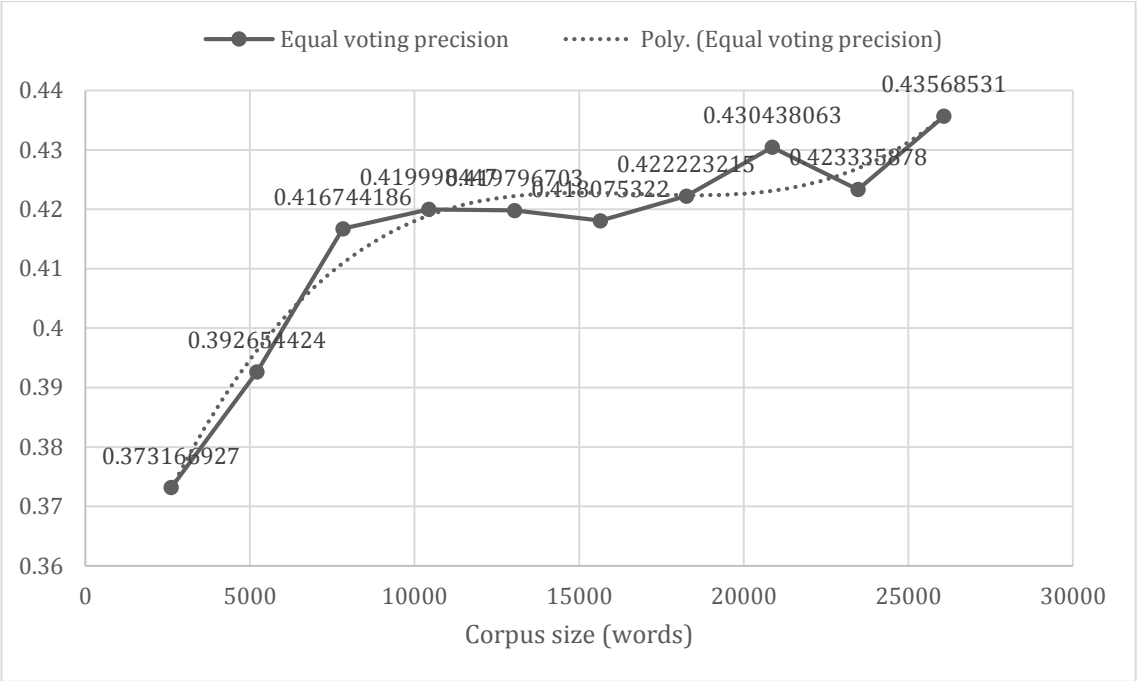

Figure 10.29: Precision measurement of equal voting procedure

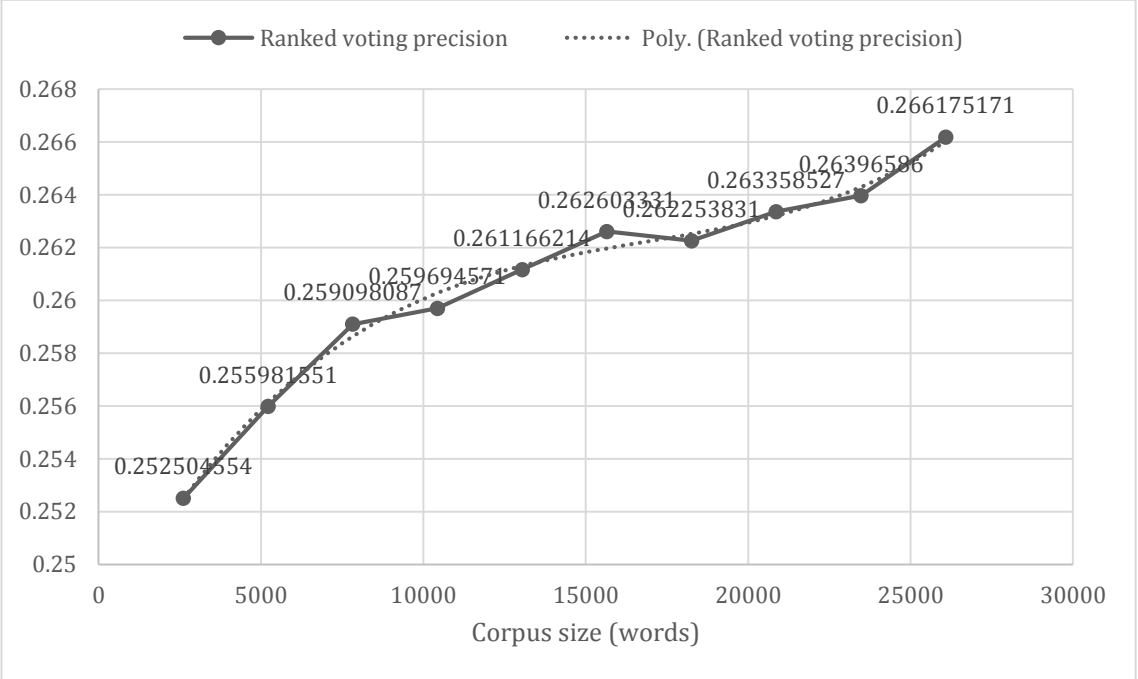

Figure 10.30: Precision ratio measurement of equal voting procedure

Here we have presented an extensive comparison between equal voting and ranked voting procedures in decreasing order of measurement importance. Even though equal voting outperforms ranked voting in accuracy and precision, ranked voting is performing significantly better on account of false omission ratio.

- Auto-complete/correct with typographical errors in validation set

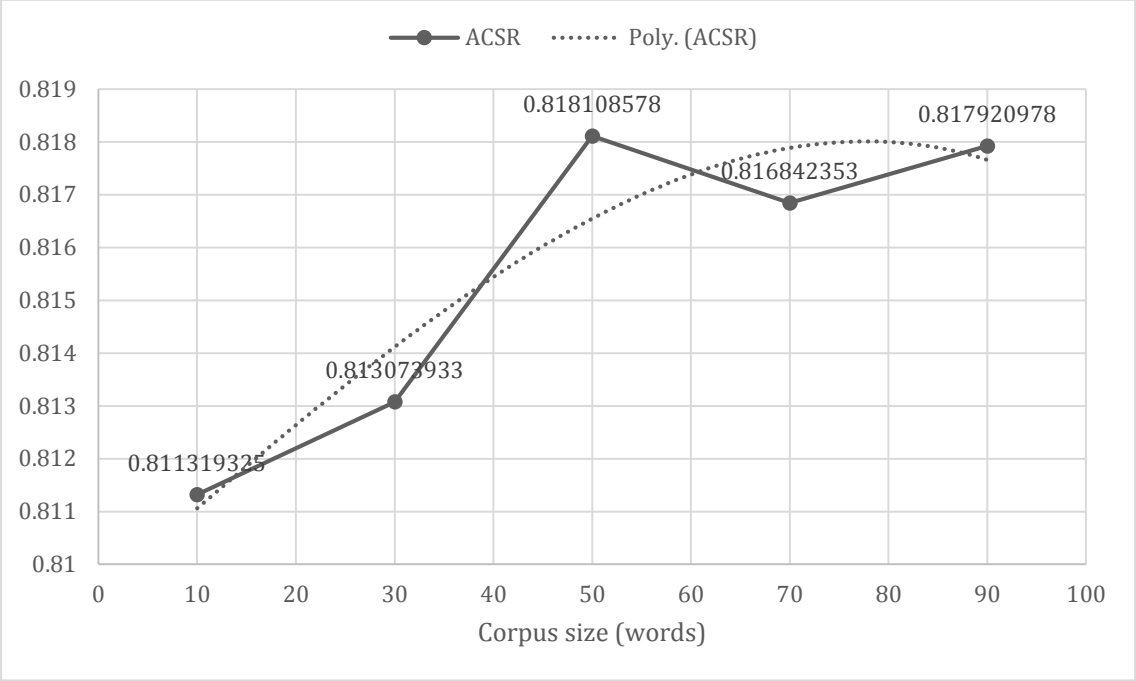

Figure 10.31: ACSR measurement of auto-complete/correct procedure

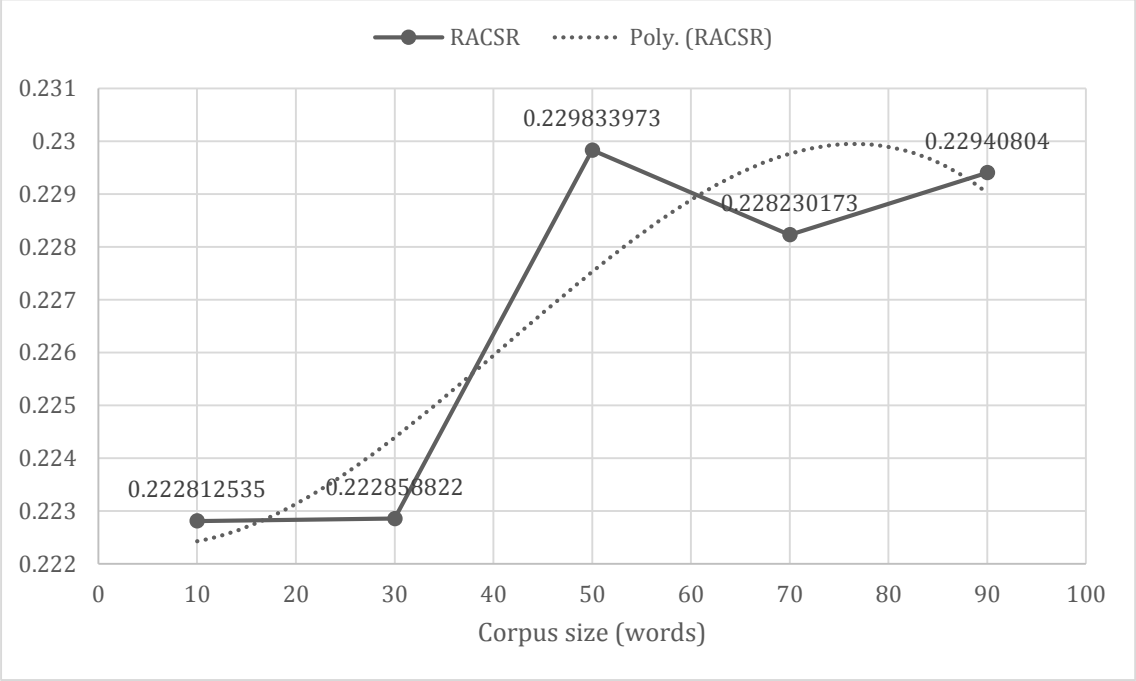

Figure 10.32: RACSR measurement of auto-complete/correct procedure

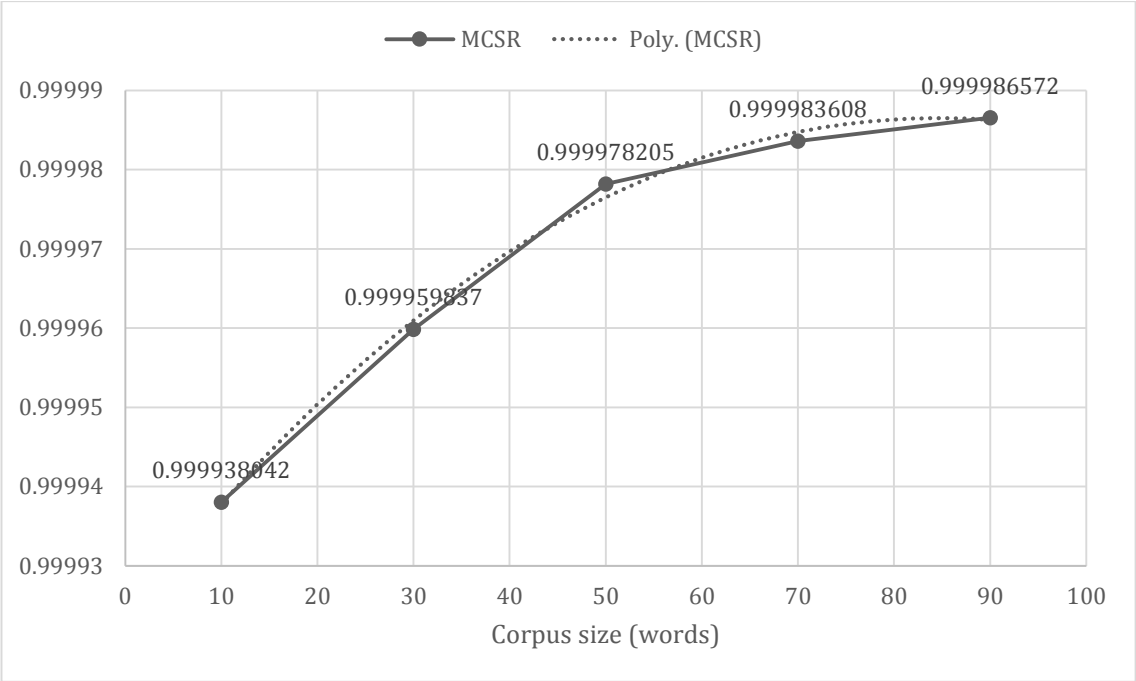

Figure 10.33: MCSR measurement of auto-complete/correct procedure

From data obtained from logs produced by this particular available user, it could be noticed that MCSR should be discarded as an indicator. MCSR produces too optimistic values, and this comes from the fact that the user is biased toward short commands. For the same reason, RACSR appears to be too pessimistic measurement. RACSR is not counted for commands shorter than three characters, due to the fact that in this case we only predict one character which does not bring any value to the measurement. The most stable estimate is in this case ACSR since it takes into account equally short and long commands with no bias toward any of them. Still even ACSR observes the influence of increased frequency of short commands in the logs and the reason for the value of 81% should be found there. We expect smaller values for more balanced train data.