

Online Ακολουθιακή Πρόβλεψη με Χρήση Αυξητικής Τμηματοποίησης

Ο αλγόριθμος Heuristic ALZ

Παπακώστας Δημήτριος Α.Μ 194

7/10/2010

Τα έξυπνα συστήματα έχουν την ικανότητα να προβλέπουν την εξέλιξη των γεγονότων στο μέλλον και να αξιοποιούν αυτή τη γνώση για να παίρνουν πιο αξιόπιστες αποφάσεις. Ο αλγόριθμος Heuristic ALZ είναι ένας αλγόριθμος ακολουθιακής πρόβλεψης, ο οποίος με βάση πιθανολογικά μοντέλα σε συνδυασμό με τη γνώση που του παρέχει το παρελθόν για το χώρο που βρίσκεται αυτός, μπορεί να πάρει απόφαση για την εξέλιξη των γεγονότων μέσα στο χώρο αυτό.

*Στη σύζυγό μου Νεκταρία για την
αμέριστη συμπαράστασή της*

Περιεχόμενα

1. Εισαγωγή	5
1.1 Η εξάπλωση των έξυπνων συστημάτων	6
1.2 Το πρόβλημα της πρόβλεψης σε διακεκριμένες ακολουθίες δεδομένων	6
1.3 Τομείς εφαρμογής των συστημάτων με ικανότητα πρόβλεψης	8
1.4 Η συνεισφορά της Μεταπτυχιακής Διατριβής	12
1.5 Οργάνωση της Μεταπτυχιακής Διατριβής	12
2. Σχετικές εργασίες	13
2.1 Οι προγνώστες Markov	14
2.2 Ο αλγόριθμος LZ78	15
2.3 Ο αλγόριθμος Active LeZi	16
2.4 Κατανομή Πιθανοτήτων	19
3. Ο αλγόριθμος Heuristic ALZ	23
3.1 Πως δουλεύει	24
3.2 Το λεξικό των αποθηκευμένων φράσεων	25
3.3 Ο μηχανισμός πρόβλεψης	26
3.4 Η Ρύθμιση των παραμέτρων του	30
4. Πειράματα	51
4.1 Τα δεδομένα εισόδου	52
4.2 Σύγκριση επίδοσης Heuristic ALZ	53
5. Συμπεράσματα – Μελλοντικές Εργασίες	62
Αναφορές – Παραπομπές	65
Παραρτήματα	66
A. Πειραματικά αποτελέσματα	
B. Πηγαίος Κώδικας	

Εικόνες – Διαγράμματα

A/A	Περιγραφή	Σελ.
1.1	Branch Predictor	8
1.2	Διάθεση πόρων δικτύου κινητής τηλεφωνίας	9
1.3	Call Admission Control	10
2.1	Ο ψευδοκώδικας του LZ78	15
2.2	Το Trie του LZ78	16
2.3	Τα χρησιμοποιούμενα Contexts του LZ78	16
2.4	Το ολισθαίνων παράθυρο του Active LeZi	17
2.5	Ο ψευδοκώδικας του Active LeZi	18
2.6	Το Trie του Active LeZi	19
2.7	Τα χρησιμοποιούμενα Contexts του Active LeZi	20
3.1	Ο ψευδοκώδικας του Heuristic ALZ	24
3.2	Τα χρησιμοποιούμενα Contexts του Heuristic ALZ	25
3.3	Το Trie του Heuristic ALZ	25
3.4	Εύρεση στο Trie του active πλαισίου χαρακτήρων “bba”	27
3.5	Εύρεση στο Trie του active πλαισίου χαρακτήρων “abb”	27
3.6	Εύρεση στο Trie του active πλαισίου χαρακτήρων “bb”	29
3.7	Εύρεση στο Trie του active πλαισίου χαρακτήρων “baa”	30
3.8	Μοντέλο καθορισμού δυνάμενων συμβολοσειρών εισόδου	31
3.9	Υπόδειγμα Γραμμικής (Linear) γραμμής τάσης	32
3.10	Υπόδειγμα Λογαριθμικής (Logarithmic) γραμμής τάσης	32
3.11	Υπόδειγμα Πολυωνυμικής (Polynomial) γραμμής τάσης	33
3.12	Διάγραμμα “Number of Contexts Vs Char Sequence”	34
3.13	Διάγραμμα “Prediction Vs Number of Symbols in Alphabet”	34
3.14	Διάγραμμα “Time Vs Number of Symbols in Alphabet”	35
3.15	Δυνάμενα μονοπάτια σε αλφάβητο “5” συμβόλων	38
3.16	Διακύμανση της ποιότητας πρόβλεψης σε αλφάβητο “5” συμβόλων	38
3.17	Διακύμανση του χρόνου εκτέλεσης σε αλφάβητο “5” συμβόλων	38
3.18	Δυνάμενα μονοπάτια σε αλφάβητο “10” συμβόλων	39

A/A	Περιγραφή	Σελ.
3.19	Διακύμανση της ποιότητας πρόβλεψης σε αλφάβητο “10” συμβόλων	39
3.20	Διακύμανση του χρόνου εκτέλεσης σε αλφάβητο “10” συμβόλων	39
3.21	Δυνάμενα μονοπάτια σε αλφάβητο “15” συμβόλων	40
3.22	Διακύμανση της ποιότητας πρόβλεψης σε αλφάβητο “15” συμβόλων	40
3.23	Διακύμανση του χρόνου εκτέλεσης σε αλφάβητο “15” συμβόλων	40
3.24	Δυνάμενα μονοπάτια σε αλφάβητο “20” συμβόλων	41
3.25	Διακύμανση της ποιότητας πρόβλεψης σε αλφάβητο “20” συμβόλων	41
3.26	Διακύμανση του χρόνου εκτέλεσης σε αλφάβητο “20” συμβόλων	41
3.27	Δυνάμενα μονοπάτια σε αλφάβητο “26” συμβόλων	42
3.28	Διακύμανση της ποιότητας πρόβλεψης σε αλφάβητο “26” συμβόλων	42
3.29	Διακύμανση του χρόνου εκτέλεσης σε αλφάβητο “26” συμβόλων	42
3.30	Χρησιμοποιούμενα Markov μοντέλα με τιμή expediteFactor = 1	43
3.31	Διάγραμμα “Prediction Vs ExpediteFactor”	44
3.32	Διάγραμμα “Time Vs ExpediteFactor”	44
3.33	Διάγραμμα “Prediction Vs trieDepthFactor”	45
3.34	Διάγραμμα “Time Vs trieDepthFactor”	45
3.35	Διάγραμμα “Prediction Vs trieDepthFactor” (Extended)	46
3.36	Διάγραμμα “Time Vs trieDepthFactor” (Extended)	46
3.37	Διάγραμμα “Prediction Vs heuristicFactor”	48
3.38	Διάγραμμα “Time Vs heuristicFactor”	48
3.39	Διάγραμμα “Prediction Vs heuristicFactor” (Extended)	49
3.40	Διάγραμμα “Time Vs heuristicFactor” (Extended)	50
4.1	Τα χρησιμοποιούμενα Contexts επιπέδου “0” του Heuristic ALZ	53
4.2	Διάγραμμα “Zero Order Vs Full Order Model” – Time / “data” set	54
4.3	Διάγραμμα “Zero Order Vs Full Order Model” – Time / “data80” set	54
4.4	Διάγραμμα “Zero Order Vs Full Order Model” – Time / “a_1” set	55
4.5	Διάγραμμα “Zero Order Vs Full Order Model” – Prediction / “data” set	55
4.6	Διάγραμμα “Zero Order Vs Full Order Model” – Prediction / “data80” set	56
4.7	Διάγραμμα “Zero Order Vs Full Order Model” – Prediction / “a_1” set	56
4.8	Διάγραμμα “Prediction Vs Training Instances” – “data” set	57
4.9	Διάγραμμα “Time Vs Training Instances” – “data” set	57
4.10	Διάγραμμα “Prediction Vs Training Instances” – “data80” set	58
4.11	Διάγραμμα “Time Vs Training Instances” – “data80” set	59
4.12	Διάγραμμα “Prediction Vs Training Instances” – “a_1” set	60
4.13	Διάγραμμα “Time Vs Training Instances” – “a_1” set	61

1

Εισαγωγή

A/A	Περιγραφή	Σελ.
2.1	Η εξάπλωση των έξυπνων συστημάτων	6
2.2	Το πρόβλημα της πρόβλεψης σε διακεκριμένες ακολουθίες δεδομένων	6
2.3	Τομείς εφαρμογής των συστημάτων με ικανότητα πρόβλεψης	8
2.4	Η συνεισφορά της Μεταπτυχιακής Διατριβής	12
2.5	Οργάνωση της Μεταπτυχιακής Διατριβής	12

1.1 Η εξάπλωση των έξυπνων συστημάτων

Με το όρο έξυπνα συστήματα αναφερόμαστε σε εκείνα, που εμφανίζουν την ικανότητα να μαθαίνουν κατά τη διάρκεια της ύπαρξής τους. Με άλλα λόγια τα συστήματα αυτά, έχουν την ικανότητα να αντιλαμβάνονται το περιβάλλον μέσα στο οποίο βρίσκονται και για κάθε κατάσταση με την οποία έρχονται αντιμέτωποι, να μαθαίνουν ποια ενέργεια πρέπει να εκτελέσουν, έτσι ώστε να πετύχουν τους αντικειμενικούς σκοπούς για τους οποίους φτιάχτηκαν.

Όσο όμως τα έξυπνα συστήματα γίνονται πιο ευρέως διαδεδομένα, θα πρέπει να έχουν την ικανότητα να προβλέπουν και να προσαρμόζονται στις μελλοντικές καταστάσεις. Την ικανότητα αυτή πραγματεύεται η παρούσα μεταπτυχιακή διατριβή. Μελετούμε τη συμπεριφορά και την επίδοση ενός επιπλέον αλγορίθμου πρόβλεψης, ο οποίος στηρίζεται πάνω σε μεθόδους συμπίεσης δεδομένων.

Ο αλγόριθμος πρόβλεψης Heuristic ALZ που προτείνουμε, προσεγγίζει το πρόβλημα της διαδοχικής πρόβλεψης από τη σκοπιά της θεωρίας δεδομένων (Information Theory), σε συνδυασμό με εμπειρικά δεδομένα που αφορούν το πρόσφατο παρελθόν του συστήματος για το οποίο μελετάται η μελλοντική συμπεριφορά του.

Για κάθε ακολουθία γεγονότων, που αναπαρίστανται με την μορφή συμβόλων ενός συγκεκριμένου αλφαβήτου, και τα οποία μπορούν να μοντελοποιηθούν ως μία πιθανολογική διαδικασία, ο αλγόριθμος Heuristic ALZ χρησιμοποιεί τα σχηματιζόμενα μοντέλα Markov, σε συνδυασμό με την ιστορία του συστήματος, όπως αυτή αποτυπώνεται σε ψηφιακά δένδρα – trie, για τη βέλτιστη πρόβλεψη του επόμενου συμβόλου.

1.2 Το πρόβλημα της πρόβλεψης σε διακεκριμένες ακολουθίες δεδομένων

Για να είναι δυνατόν να προσδιοριστεί το ποσοστό χρησιμότητας του παρελθόντος στην πρόβλεψη του μέλλοντος, απαιτείται αρχικά να γίνει η τυπική μαθηματική διατύπωση του προβλήματος.

Έτσι, αν Σ ένα αλφάβητο που αποτελείται από ένα πεπερασμένο αριθμό από σύμβολα $s_1, s_2, \dots, s_{|\Sigma|}$ (όπου το σύμβολο $|\Sigma|$ αντιπροσωπεύει το μέγεθος / πληθάριθμο των στοιχείων του συνόλου Σ), ένας προγνώστης συσσωρεύει συμβολοσειρές της μορφής $\alpha_i = \alpha_i^1, \alpha_i^2, \alpha_i^3, \dots, \alpha_i^{n_i}$ όπου $\alpha_i^j \in \Sigma, \forall i, j$. Το σύμβολο n_i αντιπροσωπεύει τον αριθμό των συμβόλων που συμμετέχουν στη συμβολοσειρά α_i . Χωρίς να οδηγηθούμε σε απώλεια της γενικότητας, μπορούμε να υποθέσουμε ότι όλη η γνώση του προγνώστη αποτελείται από μία ακολουθία

$\alpha = \alpha^1, \alpha^2, \dots, \alpha^n$. Με βάση το α , ο αντικειμενικός σκοπός του προγνώστη είναι να κατασκευάσει ένα μοντέλο, το οποίο αναθέτει πιθανότητες σε κάποιο μελλοντικό γεγονός δεδομένου “κάποιου” παρελθόντος.

Σε κάθε δεδομένη χρονική στιγμή t (το οποίο σημαίνει ότι t σύμβολα $\chi_t, \chi_{t-1}, \dots, \chi_1$ έχουν κάνει την εμφάνισή τους σε αντίστροφη σειρά), θα πρέπει να υπολογίσουμε την δεσμευμένη πιθανότητα

$$P[X_{t+1} = \chi_{t+1} \mid X_t = \chi_t, X_{t-1} = \chi_{t-1}, \dots]$$

όπου $\chi_i \in \Sigma, \forall \chi_{t+1} \in \Sigma$. Αυτό το μοντέλο εισάγει μία στατική Markov αλυσίδα, από τη στιγμή που οι πιθανότητες δεν είναι εξαρτώμενες του χρόνου.

Το αποτέλεσμα του προγνώστη αφορά στη βαθμονόμηση του κάθε συμβόλου σύμφωνα με την πιθανότητα εμφάνισης του P . Οι προγνώστες που χρησιμοποιούν αυτού του είδους τη μέθοδο πρόβλεψης ονομάζονται Markov προγνώστες.

Η ιστορία $\chi_t, \chi_{t-1}, \dots$ που χρησιμοποιείται στον παραπάνω ορισμό ονομάζεται **φράση** του προγνώστη και αναφέρεται στο κομμάτι εκείνο του παρελθόντος το οποίο επηρεάζει τη μελλοντική συμπεριφορά του συστήματος. Το μήκος της ιστορίας του προγνώστη (που ονομάζεται επίσης και **μήκος** ή **μνήμη** ή **τάξη** του Markov προγνώστη) αναπαρίσταται με το σύμβολο l . Έτσι ένας προγνώστης ο οποίος εκμεταλλεύεται τα l προηγούμενα σύμβολα για να κάνει την πρόβλεψή του, θα υπολογίσει τις ακόλουθες δεσμευμένες πιθανότητες:

$$P[X_{t+1} = \chi_{t+1} \mid X_t = \chi_t, X_{t-1} = \chi_{t-1}, \dots, X_{t-l+1} = \chi_{t-l+1}]$$

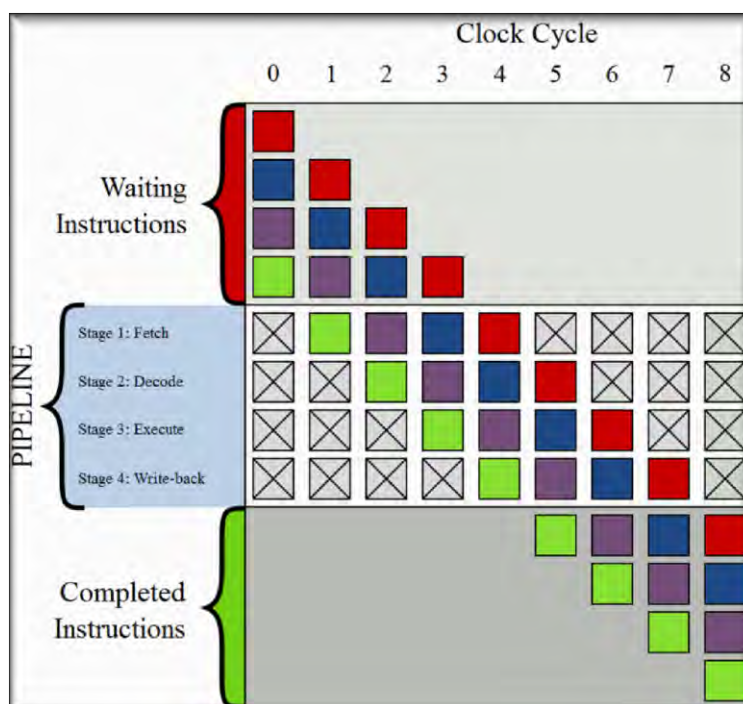
Υπάρχουν δύο τύποι Markov προγνωστών.

- Η πρώτη κατηγορία αφορά αυτούς που έχουν την τιμή του l προ-τοποθετημένη και ονομάζονται σταθερού μήκους Markov προγνώστες τάξης k . Αυτοί υπολογίζουν τις δεσμευμένες πιθανότητες όπως περιγράψαμε παραπάνω, αλλά λαμβάνουν υπόψη τους μόνο τα γεγονότα από t μέχρι $t - k + 1$ με $k < l$. Το μειονέκτημα των προγνωστών αυτών είναι ότι δεν υπάρχει κάποιος τρόπος να βρεθεί κάποια βέλτιστη τιμή για το k , γεγονός που επηρεάζει την ικανότητα πρόβλεψής τους ανάλογα με την περίπτωση.

- Η δεύτερη κατηγορία αφορά αυτούς, που υπολογίζουν τις δεσμευμένες πιθανότητες όπως παραπάνω, αλλά με την τιμή του I να είναι συνάρτηση του χρόνου, δηλαδή είναι $I = I(x_t, x_{t-1}, \dots)$. Αυτοί οι προγνώστες ονομάζονται μεταβλητού μήκους αλυσίδες Markov, και η τιμή του I μπορεί να κυμαίνεται από 1 έως t . Οι προγνώστες αυτού του είδους μπορούν να προσαρμόζονται στα δεδομένα που επεξεργάζονται και βελτιστοποιούν με αυτό τον τρόπο το μοντέλο πρόβλεψής τους.

1.3 Τομείς εφαρμογής των συστημάτων με ικανότητα πρόβλεψης

Τα συστήματα με ικανότητα πρόβλεψης, βρίσκουν ήδη εφαρμογή στο κλάδο της αρχιτεκτονικής των υπολογιστών και πιο συγκεκριμένα στην ανάπτυξη των ψηφιακών κυκλωμάτων πρόγνωσης κλάδου ροής εργασίας (**Branch Predictor**). Τα κυκλώματα αυτά, προσπαθούν να προβλέψουν ποιος κλάδος σε μία δομή **if – then – else** είναι αυτός που θα επιλεγεί, πριν ακόμα αυτό είναι γνωστό στα σίγουρα. Ο σκοπός της ύπαρξης των προγνωστών αυτού του είδους, είναι η βελτιστοποίηση της λειτουργίας των ψηφιακών ηλεκτρονικών συστημάτων που αυτοί χρησιμοποιούνται, μέσω της αύξησης του αριθμού των εντολών που μπορούν να εκτελεστούν, στη μονάδα του χρόνου (**instruction throughput**).

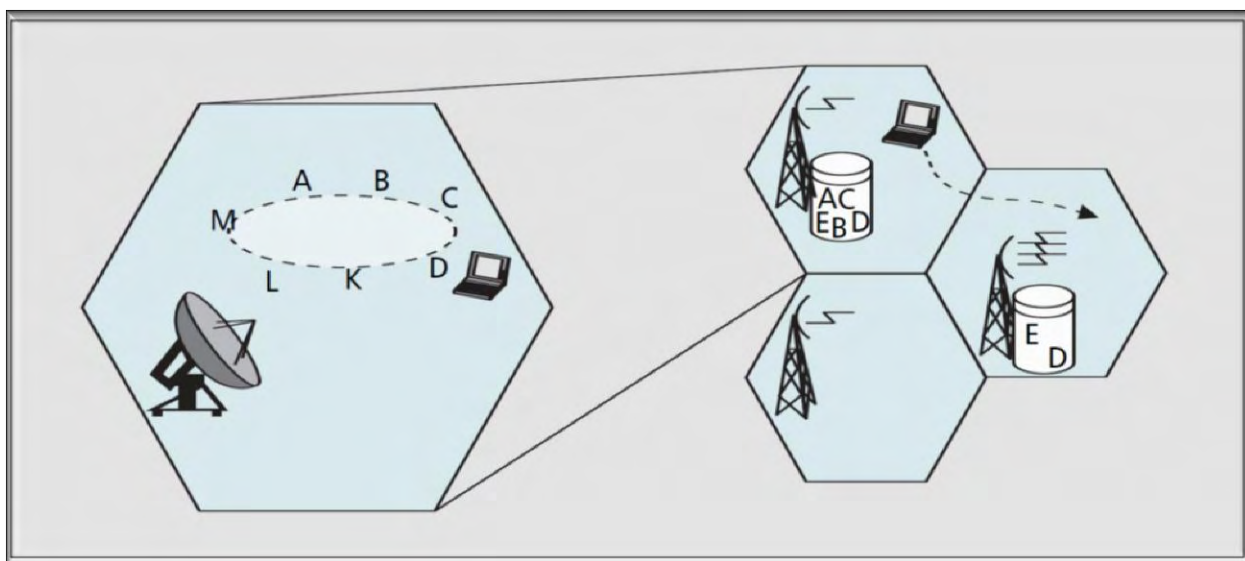


Εικόνα 1.1. Παράδειγμα ενός σωλήνα τεσσάρων (4) σταδίων. Τα χρωματιστά κουτιά παριστάνουν εντολές ανεξάρτητες μεταξύ τους.

Τα συστήματα πρόγνωσης χρησιμοποιούνται και για τη μείωση της καθυστέρησης (**latency**) των λειτουργιών που πραγματοποιούνται στη μνήμη των σύγχρονων υπολογιστικών συστημάτων^[1] (**prefetching**). Ένας προγνώστης διαχείρισης μνήμης λειτουργεί σαν μία

διεπαφή μεταξύ της **on chip cache** και της **off chip cache** και μπορεί να προστεθεί στα υπάρχοντα σχέδια υπολογιστικών συστημάτων. Οι προγνώστες που πραγματεύεται η παρούσα μεταπτυχιακή διατριβή στηρίζονται πάνω σε Markov μοντέλα πρόγνωσης, που στον τομέα διαχείρισης μνήμης παρουσιάζουν την ιδιαιτερότητα να προ-ανακαλούν πολλαπλές αναφορές πρόβλεψης από τη μνήμη και στη συνέχεια να ιεραρχούν την επίδοση των αναφορών αυτών στον επεξεργαστή. Η σχεδίαση αυτή έχει ως αποτέλεσμα το σύστημα να έχει πολύ καλή κάλυψη, να είναι ακριβές και να παράγει έγκαιρα αποτελέσματα που μπορούν να χρησιμοποιηθούν αποτελεσματικά από τον επεξεργαστή.

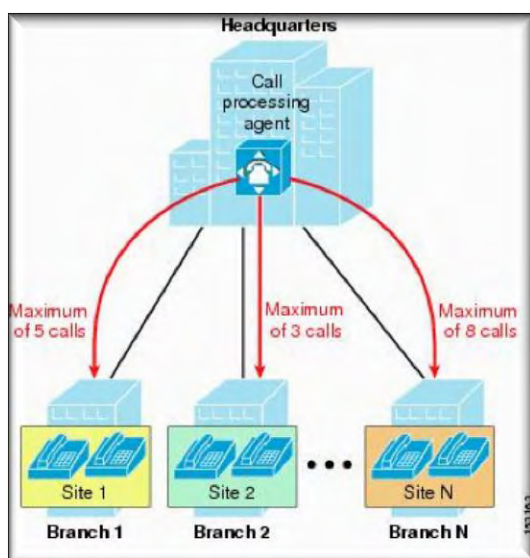
Ένας άλλος τομέας που βρίσκουν εφαρμογή τα συστήματα πρόγνωσης, αφορά στα ασύρματα δίκτυα. Η δυνατότητα πρόβλεψης της θέσης ενός κινητού χρήστη^{[2][3]}, δεδομένης της ιστορίας κίνησης αυτού, μπορεί να βελτιώσει την απόδοση του δικτύου (λόγω της μείωσης του latency που αφορά το συγκεκριμένο χρήστη).



Εικόνα 1.2. Παράδειγμα μίας δοσοληψίας δικτύου κινητής τηλεφωνίας. Το κέντρο μεταγωγής μπορεί να δεσμεύσει και να διαθέσει περισσότερους πόρους (E, D) στο δεξιότερο κελί, λόγω του ότι ο χρήστης αναμένεται να μετακινηθεί εκεί. Στο κελί που βρίσκεται τώρα ο χρήστης (το αριστερότερο και μεγαλύτερο) ενώ αυτός αναμένει το δεδομένο A μπορεί να προ-ανακαλέσει τα δεδομένα C και B από τη στιγμή που προβλέπει τη μελλοντική τους χρήση. Σε πολλές περιπτώσεις, εξαιτίας του τρόπου λειτουργίας που δουλεύει ο αλγόριθμος αναμετάδοσης των δεδομένων, τα στοιχεία B, C μπορεί να φτάσουν νωρίτερα από το A. Η προ-ανάκληση δεδομένων έχει ως αποτέλεσμα τη μείωση της καθυστέρησης του συστήματος στο σύνολό της.

Με την αξιοποίηση των συστημάτων πρόγνωσης, για την πραγματοποίηση των διαδικασιών **handover**^[4], η όποια δέσμευση πόρων δεν θα γίνεται σε πραγματικό χρόνο, αλλά αυτοί θα δεσμεύονται προκαταβολικά, παρακάμπτοντας τις όποιες αρνητικές συνέπειες του handover. Επιπρόσθετα, μέθοδοι διάθεσης πόρων όπως το **Shadow Cluster**^[5], μπορούν να επωφεληθούν από τη δυνατότητα πρόβλεψης θέσης, που προσφέρουν τα συστήματα πρόγνωσης και να μην δεσμεύουν πόρους σε όλες τις κυψέλες που γειτνιάζουν με αυτή που βρίσκεται ο κινητός χρήστης, αλλά μόνο σε αυτές που είναι πιο πιθανό να μετακινηθεί ο χρήστης.

Χρήση των συστημάτων πρόγνωσης μπορεί να γίνει και σε συστήματα ελέγχου αποδοχής κλήσεων^[6] (**Call Admission Control**), η λειτουργία των οποίων αφορά στην προστασία δικτύων VoIP από την εγγραφή σε αυτά υπερβολικά μεγάλου αριθμού συνδρομητών. Τα συστήματα αυτά, φροντίζουν ώστε να υπάρχει αρκετό bandwidth για τους διαπιστωμένους χρήστες του συστήματος, έτσι ώστε η ποιότητα των υπηρεσιών (**Quality of Service**) να παραμένει πάντα υψηλή.



Εικόνα 1.3. Αναπαράσταση ενός συστήματος ελέγχου αποδοχής κλήσεων.

Η εφαρμογή των συστημάτων πρόγνωσης σε αυτή την κατηγορία συστημάτων τηλεφωνίας, θα διατηρήσει την πιθανότητα απόρριψης μιας κλήσης (**Call Dropping Probability**) στις προσχεδιασμένες τιμές του συστήματος και θα ελαχιστοποιήσει την πιθανότητα μία κλήση να μπλοκαριστεί (**Call Blocking Probability**).

Η λειτουργία της πρόβλεψης μπορεί να χρησιμοποιηθεί και για τη βελτίωση της απόδοσης πολλών εφαρμογών διάδοσης δεδομένων (**Digital Video Broadcasting – Handheld**). Οι εφαρμογές αυτές αφορούν στις υπηρεσίες που προσφέρουν εταιρίες του εμπορίου και στηρίζονται πάνω στο πρωτόκολλο 802.11 ή στα δίκτυα κινητής τηλεφωνίας τρίτης γενιάς (3G). Σε αυτού του είδους τις εφαρμογές, οι χρήστες αρχικά ελέγχουν το κανάλι εκπομπής μέχρι να έχουν πρόσβαση στα επιθυμητά δεδομένα. Αυτός ο τρόπος λειτουργίας αυξάνει την καθυστέρηση πρόσβασης ενός χρήστη. Αν και από την πλευρά του χρήστη, η λειτουργικότητα του caching των δεδομένων που αυτός συνήθως προσπελάζει, βελτιώνει τη συμπεριφορά των συστημάτων αυτής της κατηγορίας, αυτό δεν αποτελεί πανάκεια. Η εφαρμογή συστημάτων πρόβλεψης σε αυτή την κατηγορία εφαρμογών, θα επιτρέψει την προ-ανάκληση (prefetching) των δεδομένων που επιθυμεί ο κάθε χρήστης να προσπελάσει και θα έχει ως αποτέλεσμα την περαιτέρω μείωση της καθυστέρησης πρόσβασης, για κάθε χρήστη (access latency). Η

δυνατότητα αυτή θα μπορούσε να χρησιμοποιηθεί επίσης, για τη βελτίωση των παρεχόμενων υπηρεσιών διαδικτύου, όπου η αναζήτηση των ιστοσελίδων που επιθυμεί να προσπελάσει ένας χρήστης θα μπορούσαν να προ-φορτωθούν έτσι ώστε να μειωθεί καθυστέρηση εξυπηρέτησης αυτού^[7].

Ο κλάδος της Βιολογίας μπορεί να χρησιμοποιήσει επίσης τις δυνατότητες των συστημάτων πρόγνωσης, όπως για παράδειγμα κατά την πρόβλεψη της δευτεροταγούς δομής μιας πρωτεΐνης από την αμινοξική της ακολουθία^[8]. Το όλο πρόβλημα έχει να κάνει με την κατανόηση της 'λογικής' στην οποία στηρίζεται η βιολογική δράση μιας πρωτεΐνης, για την οποία είναι αναγκαία η γνώση της στερεοδιάταξής της. Προς το παρόν, μόνο μια λεπτομερής κρυσταλλογραφική ανάλυση με περίθλαση ακτίνων - Χ (μια μέθοδος εξαιρετικά κοπιαστική, που παίρνει πολύ χρόνο και είναι εξαιρετικά δαπανηρή), μπορεί να δώσει πληροφορίες για τη στερεοδιάταξη μιας πρωτεΐνης, με την προϋπόθεση ότι κατάλληλοι κρύσταλλοι της πρωτεΐνης μπορούν να αναπτυχθούν. Ήδη οι τεχνικές της κρυσταλλογραφίας ακτίνων-Χ έχουν προσδιορίσει μ' επιτυχία πάνω από 10.000 πρωτεϊνικές δομές. Επίσης, η φασματοσκοπία NMR. Με τη χρήση των συστημάτων πρόγνωσης κατά τον υπολογισμό της δευτεροταγούς δομής μιας πρωτεΐνης, μπορούμε να πάρουμε μια 'αρχική' στερεοδιάταξη, και στη συνέχεια να οδηγηθούμε στη φυσική στερεοδομή αυτής με την ελάχιστη κατανάλωση ενέργειας και περιορισμό του κόστους.

Τέλος, χωρίς αυτό βέβαια να σημαίνει ότι εδώ περιορίζονται και οι ανεξάντλητοι τομείς εφαρμογής των συστημάτων πρόγνωσης, τα συστήματα αυτά μπορούν να χρησιμοποιηθούν σε πληθώρα στρατιωτικών εφαρμογών. Για παράδειγμα, θα μπορούσαν να χρησιμοποιηθούν, για την παρεμβολή των συστημάτων επικοινωνίας του εχθρού. Οι αντιπαρεμβολικές δυνατότητες των συστημάτων αυτών, αφορούν στην ικανότητά τους να εναλλάσσουν ένα ικανό αριθμό από συχνότητες επικοινωνίας σε μικρό χρονικό διάστημα (**frequency hopping**), έτσι ώστε να μην προλαβαίνει ο αντίπαλος να εντοπίσει και στη συνέχεια να παρεμβάλει την όποια χρησιμοποιούμενη συχνότητα. Ένα σύστημα πρόγνωσης, θα μπορούσε να αντιληφθεί τα **patterns** αλλαγής των συχνοτήτων από πλευράς του εχθρού και στη συνέχεια προκαταβολικά, να παρεμβάλει τις αναμενόμενες συχνότητες λειτουργίας, καθιστώντας την επικοινωνία μεταξύ των συμμετεχόντων αδύνατη. Αντίστοιχη χρησιμοποίηση θα μπορούσε να επιτευχθεί και σε συστήματα RADAR, για την παρεμβολή των συστημάτων αυτών και μείωση της αντίληψης της πραγματικής κατάστασης (Situation Awareness) του εχθρού.

1.4 Η συνεισφορά της Μεταπτυχιακής Διατριβής

Η διατριβή αυτή παρουσιάζει ένα νέο αλγόριθμο πρόβλεψης, τον Heuristic ALZ, ο οποίος στηρίζεται στο μοντέλο πρόγνωσης που κατασκευάζει ο αλγόριθμος Active Lezi, με την εφαρμογή των πιθανολογικών δεδομένων της Θεωρίας Πληροφοριών, αλλά δεν περιορίζεται μόνο σε αυτό και χρησιμοποιεί ντετερμινιστικά μοντέλα για την ενίσχυση της ικανότητας πρόγνωσης.

Ο αλγόριθμος είναι πλήρως παραμετροποιήσιμος έτσι ώστε κάθε φορά, η ποιότητα της πρόγνωσης σε συνδυασμό με την ταχύτητα ολοκλήρωσης των εκάστοτε υπολογισμών, να ικανοποιεί τις απαιτήσεις του χρήστη.

Ο Heuristic ALZ προσφέρει την πλατφόρμα πάνω στην οποία μπορούν να γίνουν περαιτέρω μελέτες για τη συμπεριφορά των ντετερμινιστικών μοντέλων στην ικανότητα πρόβλεψης των συστημάτων αυτής της κατηγορίας.

1.5 Οργάνωση της Μεταπτυχιακής Διατριβής

Στο κεφάλαιο 2 γίνεται αναφορά στα χαρακτηριστικά των Markov προγνωστών και αναλύεται ο τρόπος λειτουργίας των δύο αντιπροσωπευτικότερων αλγορίθμων της κατηγορίας αυτής, του LZ78 και του Active LeZi. Οι δύο αυτοί αλγόριθμοι απετέλεσαν τον πρόγονο και τη ραχοκοκαλιά του πρωτοεμφανιζόμενου Heuristic ALZ.

Στο κεφάλαιο 3 γίνεται λεπτομερής παρουσίαση των χαρακτηριστικών του αλγορίθμου Heuristic ALZ και ρυθμίζονται οι παράμετροι λειτουργίας του έτσι ώστε να επιτυγχάνεται η βέλτιστη λειτουργία αυτού.

Η επίδοση του αλγορίθμου Heuristic ALZ αξιολογείται στο κεφάλαιο 4 όπου τα αποτελέσματα από μία σειρά πειραμάτων, πάνω σε τεχνητά και πραγματικά δεδομένα, συγκρίνονται με τα αντίστοιχα των αλγορίθμων LZ78 και Active LeZi.

Στο τελευταίο κεφάλαιο διατυπώνονται τα συμπεράσματα που αφορούν την εν γένει συμπεριφορά και επίδοση του αλγορίθμου Heuristic ALZ. Το κεφάλαιο κλείνει με κάποιες προτάσεις που αφορούν την περαιτέρω εξέλιξη του αλγορίθμου.

2

Σχετικές εργασίες

A/A	Περιγραφή	Σελ.
2.1	Οι προγνώστες Markov	14
2.2	Ο αλγόριθμος LZ78	15
2.3	Ο αλγόριθμος Active LeZi	16
2.4	Κατανομή Πιθανοτήτων	19

2.1 Οι προγνώστες Markov

Η ανάπτυξη συστημάτων με ικανότητα πρόβλεψης αποτέλεσε αντικείμενο μελέτης της Θεωρίας Πληροφοριών (Information Theory), από πολύ νωρίς. Σύμφωνα με αυτή, κάθε ακολουθία από γεγονότα που μπορεί να μοντελοποιηθεί ως μία πιθανολογική διαδικασία, μπορεί να οδηγήσει σε ικανότητα πρόβλεψης με τη χρήση Markov μοντέλων^[9].

Ο ανωτέρω ισχυρισμός επιβεβαιώθηκε κατά την εκτενή μελέτη των τεχνικών συμπίεσης δεδομένων. Οι αλγόριθμοι αυτής της κατηγορίας, βασίζουν τη λειτουργικότητά τους στη χρήση Markov μοντέλων και η πράξη έδειξε ότι, όταν η επίδοσή τους ως αλγόριθμοι συμπίεσης είναι υψηλή, αυτοί επιτυγχάνουν να είναι και καλοί προγνώστες.

Η ικανότητα πρόβλεψης όμως, δεν είναι το μόνο ζητούμενο από ένα αλγόριθμο αυτής της κατηγορίας. Θα πρέπει επίσης να διαθέτει την ικανότητα της προσαρμογής στον όγκο των δεδομένων που εισέρχονται στο σύστημα πρόβλεψης (και αυξάνουν την εντροπία του), έτσι ώστε η ποιότητα πρόβλεψης να παραμένει πάντα υψηλή.

Την απάντηση στο πρόβλημα αυτό έρχεται να φέρει για άλλη μία φορά η Θεωρία Πληροφοριών, σύμφωνα με την οποία, ένα μοντέλο προγνώστη του οποίου η τάξη αυξάνει σύμφωνα με το ρυθμό αύξησης της εντροπίας του συστήματος, μπορεί να επιτύχει καλά ποσοστά πρόβλεψης^[10]. Οι αλγόριθμοι συμπίεσης δεδομένων διαθέτουν αυτή την ικανότητα, μέσω της προοδευτικής τμηματοποίησης των δεδομένων που επεξεργάζονται και για το λόγο αυτό επιλέγονται ως προγνώστες πραγματικού χρόνου και αποτελούν τη βάση για τη λειτουργία του προτεινόμενου αλγορίθμου Heuristic ALZ.

Οι Meir Feder, Neri Merhav, και Michael Gutman ^[11] είναι οι πρώτοι που εξέτασαν το ενδεχόμενο της κατασκευής ενός γενικού αλγορίθμου πρόβλεψης, για εφαρμογή πάνω σε τυχαίες μεν καθορισμένες δε ακολουθίες δεδομένων, που εισέρχονται σε ένα σύστημα πρόβλεψης. Με τις μελέτες τους απέδειξαν την ύπαρξη αυτού του προγνώστη, που θα μπορούσε να προβλέψει το επόμενο στοιχείο, σε κάθε καθορισμένη ακολουθία δεδομένων. Απέδειξαν επίσης, ότι οι Markov προγνώστες οι οποίοι στηρίζονται στην οικογένεια των αλγορίθμων συμπίεσης που προέρχονται από τον LZ78, επιτυγχάνουν τη βέλτιστη προβλεπτικότητα^[11].

2.2 Ο αλγόριθμος LZ78

Ο LZ78 είναι ένας αλγόριθμος συμπίεσης, ο οποίος στηρίζει τη λειτουργία του στην ύπαρξη ενός λεξικού, που δημιουργείται από την προοδευτική τμηματοποίηση μίας συμβολοσειράς εισόδου.

Ο αλγόριθμος αυτός τεμαχίζει τη συμβολοσειρά εισόδου x_1, x_2, \dots, x_i σε $c(i)$ φράσεις $w_1, w_2, \dots, w_{c(i)}$ τέτοιες ώστε για κάθε $j > 0$, το πρόθεμα της φράσης w_j (που σημαίνει όλη η φράση εκτός του τελευταίου χαρακτήρα αυτής), να είναι όμοιο με κάποιες άλλες w_i για $1 < i < j$.

```
loop

    wait for next symbol v

    if ((w.v) in dictionary):

        w = w.v

    else

        add (w.v) to dictionary

        w = null

        increment frequency for every possible prefix of phrase

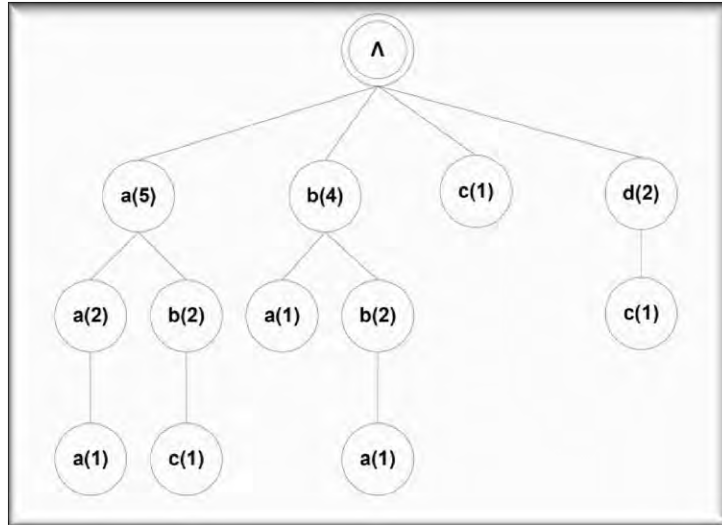
forever
```

Εικόνα 2.1. Ο ψευδοκώδικας του LZ78.

Εξαιτίας αυτής της ιδιότητας του αλγορίθμου κάθε μία από τις φράσεις που προκύπτει από το μηχανισμό τμηματοποίησης, μαζί με τη σχετική συχνότητά εμφάνισής της, μπορεί αποτελεσματικά να παρασταθεί πάνω σε ένα ψηφιακό δένδρο – trie.

Ας πάρουμε για παράδειγμα τη συμβολοσειρά εισόδου $x^n = \mathbf{aaababbbbbaabccddcbaaaa}$. Ο μηχανισμός τμηματοποίησης του αλγορίθμου LZ78 θα έφτιαχνε το trie της εικόνας 2.2 και θα παρήγαγε τις φράσεις **a**, **aa**, **b**, **ab**, **bb**, **bba**, **abc**, **c**, **d**, **dc**, **ba**, και **aaa**.

Όπως τονίσαμε και προηγουμένως, ο αλγόριθμος διατηρεί στατιστικά για κάθε συμβολοσειρά μέσα στις φράσεις w_i . Για παράδειγμα, το σύμβολο **a** εμφανίζεται πέντε φορές (στην αρχή των φράσεων **a**, **aa**, **ab**, **abc**, και **aaa**) και η συμβολοσειρά **bb** εμφανίζεται δύο φορές (στις φράσεις **bb** και **bba**).



Εικόνα 2.2. Το trie που δημιουργείται από τον LZ78 κατά την επεξεργασία της συμβολοσειράς εισόδου “aaababbbbbaabccddcbaaaa”

Καθώς ο LZ78 τεμαχίζει τη συμβολοσειρά εισόδου, όλο και μεγαλύτερες φράσεις συγκεντρώνονται μέσα στο λεξικό του αλγορίθμου. Σαν αποτέλεσμα ο αλγόριθμος συνάγει την ικανότητα για πρόβλεψη, από μοντέλα Markov ολοένα μεγαλύτερης τάξης, γεγονός που βελτιώνει τη συμπεριφορά αυτού.

aa (order 2)	a (order 1)	Λ (order 0)	
a aa (1)	a a (1)	a (1)	ba (1)
Λ aa (1)	aa a (1)	aa (1)	bb (1)
	b a (1)	aaa (1)	bba (1)
	bc a (1)	ab (1)	c (1)
	Λ a (1)	abc (1)	d (1)
		b (1)	dc (1)

Εικόνα 2.3. Τα πιθανά μονοπάτια που λαμβάνονται υπόψη για τον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου κατά την τμηματοποίηση της συμβολοσειράς **aaababbbbbaabccddcbaaaa**, και με τελευταία φράση που εισέρχεται στο λεξικό του προγνώστη την “aaa”.

2.3 Ο αλγόριθμος Active LeZi

Το μειονέκτημα του αλγορίθμου LZ78 είναι ο αργός ρυθμός σύγκλισης που διαθέτει προκειμένου να επιτύχει βέλτιστη πρόβλεψη, το οποίο και σημαίνει ότι απαιτεί να επεξεργαστεί ένα αρκετά μεγάλο αριθμό συμβόλων, έτσι ώστε αξιόπιστα να μπορέσει να πραγματοποιήσει συνεχή πρόβλεψη.

Η αδυναμία του αυτή οφείλεται κυρίως, στο ότι δεν εκμεταλλεύεται όλες τις διαθέσιμες πληροφορίες που εμπεριέχουν τα δεδομένα εισόδου. Ο αλγόριθμος δεν γνωρίζει για την ύπαρξη συμβολοσειρών οι οποίες διασχίζουν τα όρια των φράσεων που ήδη έχουν

προστεθεί στο λεξικό. Στο παράδειγμά μας όπου $\chi^n = \text{aaababbbbbaabccddcbaaaa}$, το τέταρτο σύμβολο (**b**) και το πέμπτο και έκτο (**ab**) δημιουργούν δύο διαφορετικές φράσεις, οι οποίες αν δεν είχαν διαχωριστεί ο LZ78 θα είχε βρει την φράση (**bab**) και θα έδινε μία μεγαλύτερη φράση για prediction. Δυστυχώς ο αλγόριθμος μπορεί να επεξεργαστεί μόνο μία φράση κάθε χρονική στιγμή και δεν έχει τη δυνατότητα να κοιτάξει πίσω στο πρόσφατο παρελθόν για νέες συμβολοσειρές που δεν έχουν κάνει την εμφάνισή τους στο λεξικό.

Ο Amiya Bhattacharya και ο Sajal Das έλυσαν μερικώς το αργό ρυθμό σύγκλισης του αλγορίθμου σε μία παραλλαγή αυτού που ονομάστηκε LeZi Update με το να παρακολουθούν για πιθανές συμβολοσειρές μέσα σε μία δεδομένη φράση [6]. Ομοίως οι Peter Franaszek, Joy Thomas και Παντελής Τσούκας [12] αντιμετώπισαν το πρόβλημα με την αποθήκευση πολλαπλών λεξικών για κάθε διαφορετικού μεγέθους συμβολοσειρά η οποία προηγούνταν από μία νέα φράση, και επέλεξαν τελικά εκείνο το λεξικό που θα επέφερε τη μέγιστη συμπίεση δεδομένων. Πάντως καμία από τις δύο αυτές προσεγγίσεις δεν επιχειρούσε να αναμοχλεύσει τις πληροφορίες οι οποίες είχαν χαθεί μεταξύ των ορίων των φράσεων που είχαν προστεθεί στο λεξικό κατά την αρχική τμηματοποίηση από τον LZ78.

Ο Active LeZi που αποτελεί μία βελτιωμένη έκδοση των LZ78 και LeZi Update αντιμετωπίζει το πρόβλημα της αργής σύγκλισης με τη χρήση ενός ολισθαίνοντος παραθύρου[13].

Όσο ο αριθμός των καταστάσεων σε μία συμβολοσειρά εισόδου αυξάνει (οι διάφορες καταστάσεις παρουσιάζονται ως διαφορετικοί κόμβοι πάνω στο δημιουργούμενο trie), η ποσότητα της πληροφορίας η οποία χάνεται λόγω των τμηματοποιήσεων αυξάνει απότομα. Ο Active LeZi για να αντιμετωπίσει το πρόβλημα αυτό διατηρεί ένα παράθυρο μεταβλητού μεγέθους με σύμβολα του παρελθόντος. Το μέγεθος του παραθύρου κάθε χρονική στιγμή, είναι ίσο με μέγεθος της μεγαλύτερης φράσης που μπήκε στο λεξικό.

Ο ALZ μπορεί να υπολογίσει το μέγεθος αυτό σε πραγματικό χρόνο, κάθε φορά που ένα νέο σύμβολο κάνει την είσοδό του στον αλγόριθμο, χωρίς να απαιτείται επιπλέον υπολογιστικό κόστος.



The diagram shows a rectangular box containing the string "aaababbbbbaabccddcba[aaa]". The last three characters "aaa" are enclosed in a smaller, darker box, representing the sliding window of the Active LeZi algorithm.

Εικόνα 2.4. Το παράθυρο που δημιουργείται από τον Active Lezi με την ολοκλήρωση επεξεργασίας της συμβολοσειράς εισόδου "aaababbbbbaabccddcbaaaa"

```

initialize Max_LZ_length = 0

loop

    wait for next symbol v

    if ((w.v) in dictionary):

        w := w.v

    else

        add (w.v) to dictionary

        update Max_LZ_length if necessary

        w := null

        add v to window

        if (length(window) > Max_LZ_length)

            delete window[0]

        Update frequencies of all possible contexts within window that includes v

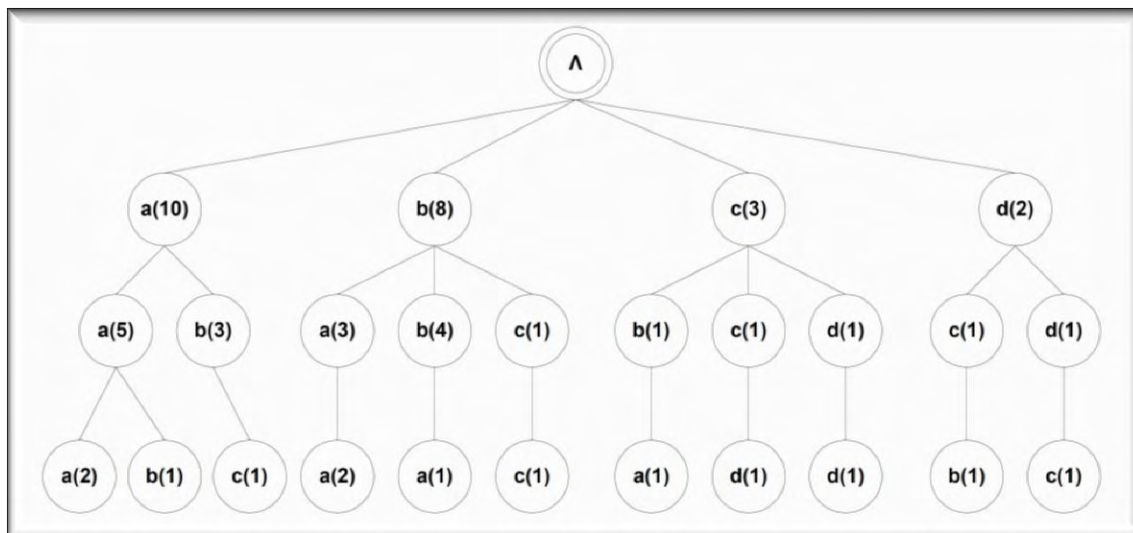
forever

```

Εικόνα 2.5. Ο ψευδοκώδικας του Active LeZi.

Ο αλγόριθμος πραγματοποιεί όλες τις στατιστικές του εντός αυτού του παραθύρου. Αν για παράδειγμα η τιμή του παραθύρου είναι ίση με **“aaa”** τότε θα προστεθεί στο trie κάθε φράση η οποία περιλαμβάνεται μέσα στο παράθυρο και εμπεριέχει τον τελευταίο χαρακτήρα που μπήκε στο σύστημα. Έτσι λοιπόν θα προστεθούν οι φράσεις **“aaa”, “aa”, “a”** και το trie του συστήματος θα ενημερωθεί είτε με την εγγραφή μίας νέας συμβολοσειράς σε αυτό, αν αυτή δεν υπάρχει, είτε με την απλή αύξηση της συχνότητας εμφάνισης κάποιας άλλης, που ήδη υπάρχει. Ο μηχανισμός τμηματοποίησης του παραθύρου επομένως, δεν επιτρέπει την απώλεια πληροφοριών, το οποίο έχει ως αποτέλεσμα την πιο γρήγορη σύγκλιση του αλγορίθμου.

Η εικόνα **2.6** δείχνει το trie που δημιουργείται από τον ALZ κατά την τμηματοποίηση της συμβολοσειράς εισόδου **aaababbbbbaabccddcbaaaa**. Σε σύγκριση με το αντίστοιχο trie του LZ78 βλέπουμε ότι διαθέτει περισσότερους κόμβους, λόγω του μηχανισμού του ολισθαίνοντος παραθύρου και είναι πιο πλήρες, ιδιότητα ιδιαίτερη χρήσιμη για τη βελτίωση της ικανότητας πρόβλεψης του συστήματος.



Εικόνα 2.6. Το trie που δημιουργείται από τον Active Lezi κατά την επεξεργασία της συμβολοσειράς εισόδου “aaababbbbbaabccddcbaaaa”

2.4 Κατανομή Πιθανοτήτων

Δεδομένης μίας συμβολοσειράς που μπαίνει στο σύστημα πρόγνωσης ως είσοδος, η πρόβλεψη του επόμενου συμβόλου είναι δυνατό να γίνει, αν υπολογιστεί η πιθανότητα εμφάνισης κάθε διαφορετικού συμβόλου που έχει παρουσιαστεί στην συμβολοσειρά και η πρόβλεψη να αφορά αυτό με τη μεγαλύτερη πιθανότητα.

Για να πετύχει ο αλγόριθμος καλύτερο ρυθμό σύγκλισης προς τη βέλτιστη πρόβλεψη, θα πρέπει αυτός να περιοριστεί στο ελάχιστο δυνατό σύνολο καταστάσεων που παρουσιάζονται στη συμβολοσειρά εισόδου. Για συνεχόμενη πρόβλεψη αυτό είναι δυνατό με τη μίξη όλων των τάξεων Markov μοντέλων που σχηματίζονται (όλων δηλαδή των φράσεων διαφορετικού μεγέθους).

Για να γίνει χρήση των μοντέλων διαφορετικών τάξεων θα πρέπει να χρησιμοποιηθεί η οικογένεια των προγνωστών μερικής αντιστοίχισης (**Prediction by Partial Match – PPM**) που οι Bhattacharya και Das χρησιμοποίησαν με πολύ θετικά αποτελέσματα για την βελτίωση της επίδοσης πρόβλεψης του αλγορίθμου LZ78. Παρόλα αυτά υπάρχει μία λεπτομέρεια που θα πρέπει να τονιστεί και αφορά τους προγνώστες PPM, η μέθοδός αυτή επικεντρώνεται στον υπολογισμό της πιθανότητας εμφάνισης του επόμενου συμβόλου στη φράση που ο αλγόριθμος επεξεργάζεται κάθε φορά, αντί του επόμενου συμβόλου στη συμβολοσειρά εισόδου.

Οι αλγόριθμοι PPM λαμβάνουν υπόψη Markov μοντέλα διαφορετικής τάξης για να υπολογίσουν την κατανομή των πιθανοτήτων, στα διάφορα σύμβολα που έχουν κάνει την εμφάνισή τους στο σύστημα^[14]. Στο παράδειγμά μας ο ALZ (εικόνα 2.6) κατασκευάζει ένα Markov μοντέλο k – τάξης. Για τον υπολογισμό των πιθανοτήτων εμφάνισης των συμβόλων με βάση όλα τα μοντέλα τάξης 1 έως k χρησιμοποιείται η τεχνική “PPM strategy of exclusion”^[15].

Το παράθυρο του ALZ προσδιορίζει το σύνολο των συμβολοσειρών που ο ALZ χρησιμοποιεί για να υπολογίσει την πιθανότητα εμφάνισης του επόμενου συμβόλου. Το παράδειγμά μας χρησιμοποιεί την τελευταία φράση “aaa”, η οποία αντιπροσωπεύει και το ισχύον παράθυρο του ALZ αλγορίθμου. Στη φράση αυτή οι συμβολοσειρές οι οποίες μπορούν να χρησιμοποιηθούν είναι όλες καταλήξεις της φράσης “aaa” (εκτός από το ίδιο το παράθυρο) και είναι οι “aa”, “a” και το σύμβολο “Λ” που αποτελεί τη ρίζα του σχηματιζόμενου trie, που αντιπροσωπεύει την περίπτωση ο αλγόριθμος να κάνει μηδενική πρόβλεψη.

Με δεδομένο λοιπόν την τελευταία φράση που έκανε την είσοδό της στο σύστημα, πρώτα θα πρέπει να βρούμε όλα τα πιθανά μονοπάτια τα οποία μπορούν να αποτελέσουν πρόβλεψη του αλγορίθμου. Ο πίνακας της εικόνας 2.7 παρουσιάζει όλα αυτά τα πιθανά μονοπάτια όπως και τη συχνότητα εμφάνισης καθενός από αυτά. Στη συνέχεια η πιθανότητα εμφάνισης του κάθε μονοπατιού (φράσης) υπολογίζεται σύμφωνα με τη μέθοδο του συνδυασμού των μοντέλων διαφορετικής τάξης που προαναφέραμε.

aa (order 2)	a (order 1)	Λ (order 0)		
a aa (2)	a a (2)	a (2)	ba (1)	ccd (1)
Λ aa (2)	aa a (2)	aa (2)	baa (2)	cdd (1)
	ab a (1)	aaa (2)	bb (3)	dcb (1)
	Λ a (2)	aab (1)	bba (1)	ddc (1)
		ab (3)	bcc (1)	b (2)
		abc (1)	cba (1)	bc (1)

Εικόνα 2.7. Τα πιθανά μονοπάτια που λαμβάνονται υπόψη για τον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου κατά την τμηματοποίηση της συμβολοσειράς **aaababbbbbaabccddcbaaaa**, και με τιμή window = “aaa”.

Χωρίς να μπούμε σε μαθηματικές λεπτομέρειες θα περιγράψουμε τη μέθοδο υπολογισμού των πιθανοτήτων για το παράδειγμά μας, την επεξεργασία δηλαδή της συμβολοσειράς εισόδου **aaababbbbbaabccddcbaaaa** και το προκύπτον trie της εικόνας 2.6.

- Η τελευταία φράση που έκανε την είσοδό της στο σύστημα είναι η “aaa” και έστω ότι θέλουμε να υπολογίσουμε ποια είναι η πιθανότητα το επόμενο σύμβολο να είναι το “a” που εμφανίζεται σε κάθε Markov μοντέλο, οποιασδήποτε τάξης.

- Ξεκινάμε πάντα από το μοντέλο υψηλότερης τάξης και ακολουθώντας το trie του συστήματος εντοπίζουμε στη φράση “aaa” τη συμβολοσειρά “aa” (δηλαδή τους δύο τελευταίους χαρακτήρες αυτής αφού ενδιαφερόμαστε για το μοντέλο Markov υψηλότερης τάξης).

- Στη συμβολοσειρά “aa”, η φράση “a” εμφανίζεται μόνο δύο (2) από τις πέντε (5) φορές που εμφανίζεται η ίδια. Η μία (1) εμφάνιση οδηγεί σε αποτέλεσμα “b” και οι υπόλοιπες δύο παράγουν μηδενική πρόβλεψη. Έτσι, η πιθανότητα να συναντήσουμε το σύμβολο “a” στη συμβολοσειρά “aa” είναι 2 στα 5.

- Για να υπολογίσουμε την πιθανότητα εμφάνισης του συμβόλου “a” στο μοντέλο 1^{ης} τάξης θα πρέπει να πολλαπλασιάσουμε τις πιθανότητες αυτού του επιπέδου με τον παράγοντα διαφυγής του συστήματος προς το επίπεδο αυτό (**escape factor**) που αντιστοιχεί στην πιθανότητα η φράση “aa” να κάνει μηδενική πρόβλεψη (2/5 για την περίπτωση μας).

- Στη συνέχεια στο μοντέλο 1^{ης} τάξης, κάνουμε πρόβλεψη για το σύμβολο “a” με πιθανότητα 2 στα 10 και πηγαίνουμε στο μοντέλο 0^{ης} τάξης με πιθανότητα διαφυγής 2/10.

- Στο επίπεδο μηδέν τέλος βλέπουμε το σύμβολο “a” με πιθανότητα 2 στα 23. Ως αποτέλεσμα υπολογίζουμε την τελική πιθανότητα για την εμφάνιση του συμβόλου “a” ως εξής:

$$\frac{2}{5} + \frac{2}{5} * \left\{ \frac{1}{5} + \left(\frac{1}{5} * \frac{2}{23} \right) \right\}$$

Η ανωτέρω διαδικασία θα πρέπει να επαναληφθεί για το σύνολο των φράσεων που μπορούν να αποτελέσουν πρόβλεψη του αλγορίθμου. Τα επιμέρους αποτελέσματα θα πρέπει να αθροιστούν έτσι ώστε το τελικό αποτέλεσμα να αντιπροσωπεύει την κατανομή των πιθανοτήτων εμφάνισης του κάθε συμβόλου που συμμετέχει στο λεξικό του συστήματος στο ενημερωμένο trie αυτού.

Αυτή η μέθοδος ανάθεσης πιθανοτήτων έχει αρκετά πλεονεκτήματα όπως:

- Λύνει το πρόβλημα της μηδενικής συχνότητας εμφάνισης μίας συμβολοσειράς καθότι μπορεί να υπάρξει πιθανότητα εμφάνισης αυτής σε κάποιο μοντέλο μικρότερης τάξης.

- Αυτή η στρατηγική της ανάμειξης των πιθανοτήτων εμφάνισης του εκάστοτε συμβόλου προσδίδει μεγαλύτερο βάρος στα μοντέλα υψηλότερης τάξης αν βρει μέσα στη συμβολοσειρά το σύμβολο για το οποίο γίνεται λόγος ενώ παράλληλα περιορίζει την επίδραση των μοντέλων χαμηλότερης τάξης που αφορούν τις περιπτώσεις όπου το σύστημα κάνει μηδενική πρόβλεψη. Με τον τρόπο αυτό το σύστημα παίρνει την πιο ενημερωμένη απόφαση σε ότι αφορά την πρόβλεψη που θα κάνει.

3

Ο αλγόριθμος Heuristic ALZ

A/A	Περιγραφή	Σελ.
3.1	Πως δουλεύει	24
3.2	Το λεξικό των αποθηκευμένων φράσεων	25
3.3	Ο μηχανισμός πρόβλεψης	26
3.4	Η Ρύθμιση των παραμέτρων του	30
3.4.1	Η σταθερά observationFactor	34
3.4.2	Η σταθερά alphabetSize	34
3.4.3	Το μέγεθος της συμβολοσειράς εισόδου	36
3.4.4	Η σταθερά expediteFactor	44
3.4.5	Η σταθερά trieDepthFactor	45
3.4.6	Η σταθερά heuristicFactor	48
3.4.7	Η σταθερά zeroLevelContexts	51

Ο αλγόριθμος Heuristic ALZ

3.1 Πως δουλεύει

Ο αλγόριθμος Heuristic ALZ στηρίζει την λειτουργία του πάνω στον αλγόριθμο τμηματοποίησης του αλγορίθμου LZ 78 και αποτελεί τροποποίηση του βασικού αλγορίθμου Active LeZi.

Η διαφορά με τον τελευταίο έγκειται σε δύο σημαντικούς τομείς:

α. Στον τρόπο με τον οποίο ο καθένας κατασκευάζει το λεξικό (dictionary) των φράσεων που μετέχουν στο προκύπτον ψηφιακό δένδρο – trie.

β. Στον τρόπο με τον οποίο κάνουν την πρόβλεψη.

```
initialize Max_LZ_length = 0

loop
    wait for next symbol v
    if ((w.v) in dictionary):
        w := w.v
    else
        add (w.v) to dictionary
        update Max_LZ_length if necessary
        w := null
    add v to window
    if (length(window) > Max_LZ_length)
        delete window[0]
    Update dictionary with of all possible contexts within window that includes v
    if (w.v added recently in the dictionary)
        delete w.v

forever
```

Εικόνα 3.1. Ο ψευδοκώδικας του Heuristic ALZ

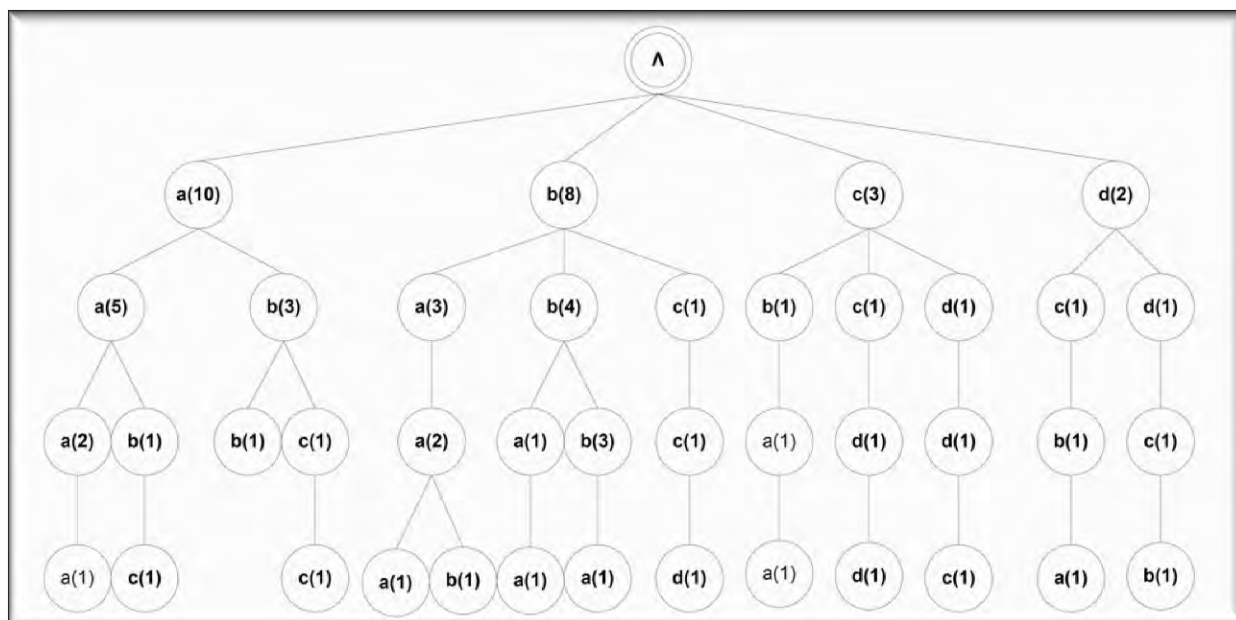
3.2 Το λεξικό των αποθηκευμένων φράσεων

Ο αλγόριθμος Active LeZi προσθέτει στο λεξικό του μόνο τις φράσεις που παράγει ο μηχανισμός τμηματοποίησης του. Οι φράσεις που προκύπτουν από το μηχανισμό του ολισθαίνοντος παραθύρου προστίθενται απευθείας στο παραγόμενο trie του συστήματος.

Σε αντιδιαστολή ο Heuristic ALZ προσθέτει στο λεξικό του όλες τις φράσεις που παράγονται από τους δύο μηχανισμούς τμηματοποίησης. Με τον τρόπο αυτό το λεξικό που προκύπτει είναι αισθητά πλουσιότερο και το παραγόμενο trie είναι μεγαλύτερο τόσο κατά πλάτος όσο και κατά ύψος (Εικόνα 3.3).

aaa (order 3)	aa (order 2)	a (order 1)		Λ (order 0)			
a aaa (1)	a aa (1)	a a (2)	bcc a (1)	a (2)	abb (1)	bbba (1)	ddcb (1)
Λ aaa (1)	aa aa (1)	aa a (1)	Λ a (2)	aa (2)	abcc (1)	bccd (1)	bc (1)
	bc aa (1)	aaa a (1)		aaa (1)	baaa (1)	cbaa (1)	abc (1)
	Λ aa (2)	abc a (1)		aaaa (1)	baab (1)	ccdd (1)	b (1)
		b a (1)		aabc (1)	bbaa (1)	cddc (2)	bb (1)
		bb a (1)		ab (1)	bbb (2)	dcba (1)	bcc (1)

Εικόνα 3.2. Τα πιθανά μονοπάτια που λαμβάνονται υπόψη για τον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου κατά την τμηματοποίηση της συμβολοσειράς aaababbbbbaabccddcbaaaa, και με τιμή window = “aaa”.



Εικόνα 3.3. Το trie που δημιουργείται από τον Heuristic ALZ αλγόριθμο κατά την επεξεργασία της συμβολοσειράς εισόδου “aaababbbbbaabccddcbaaaa”

Η αποθήκευση στο trie περισσότερων και μεγαλύτερων συμβολοσειρών, που δικαιολογούν και το μεγαλύτερο ύψους αυτού, έχει ως άμεση συνέπεια στη συμπεριφορά του αλγόριθμου τη γρηγορότερη σύγκλιση αυτού.

Η συμπεριφορά αυτή ενισχύεται κατά ένα μεγάλο ποσοστό και από το γεγονός ότι η πρόβλεψη για το επόμενο σύμβολο στον Heuristic ALZ, δεν γίνεται αποκλειστικά όπως σε κάθε Markov προγνώστη, με τον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου που συμμετέχει στη συμβολοσειρά εισόδου, αλλά χρησιμοποιείται μία πιο εμπειρική μέθοδος, σε συνδυασμό με τον υπολογισμό των ανωτέρω πιθανοτήτων, που έχει να κάνει με την ιστορία του συστήματος που βρίσκεται ο προγνώστης.

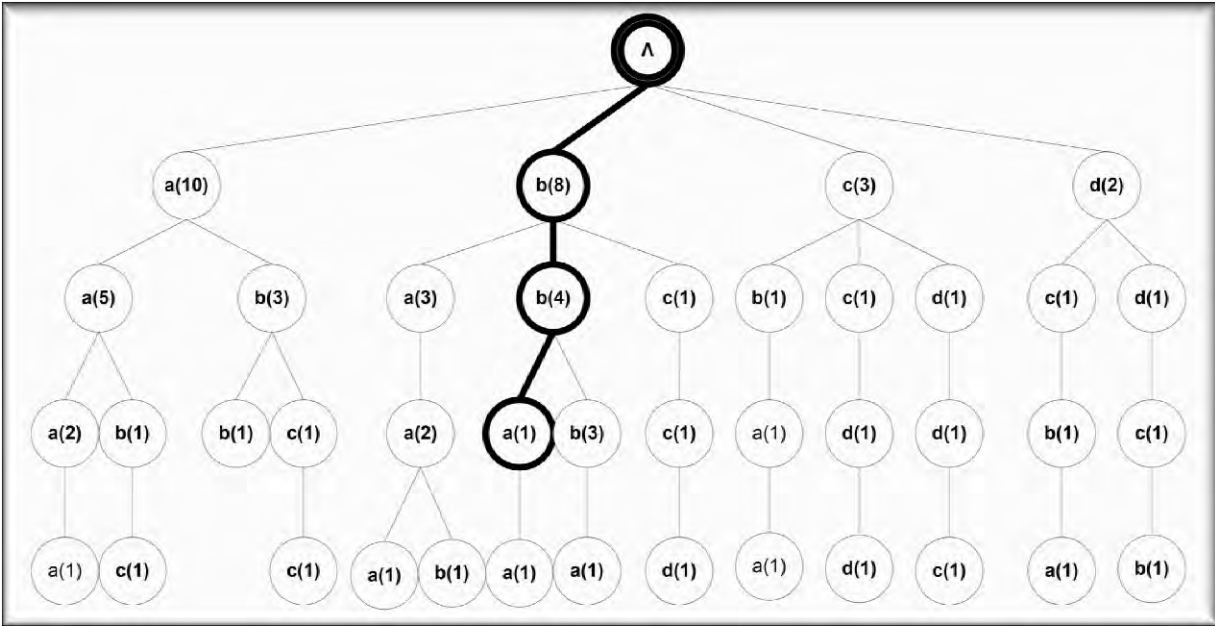
3.3 Ο μηχανισμός πρόβλεψης

Πιο συγκεκριμένα ο αλγόριθμος τμηματοποίησης του Heuristic ALZ (και ο Active LeZi) ανά πάσα χρονική στιγμή διαθέτει ένα active πλαίσιο χαρακτήρων του οποίου την ύπαρξη θα αναζητήσει στο trie του συστήματος. Αυτό διαμορφώνεται από το πλαίσιο χαρακτήρων της προηγούμενης χρονικής στιγμής με την προσθήκη του τελευταίου χαρακτήρα που έκανε την εμφάνισή του στο σύστημα.

Στο παράδειγμα της Εικόνας **3.4** έστω ότι το active πλαίσιο χαρακτήρων είναι το “bba”. Από τη μελέτη του ψευδοκώδικα του αλγορίθμου καταλαβαίνουμε ότι πριν από ακριβώς τρεις χρονικές στιγμές¹, όσο δηλαδή είναι και ο πληθάριθμος του active πλαισίου χαρακτήρων, μία νέα φράση προστέθηκε στο λεξικό γεγονός που στη συνέπεια μηδένισε την τιμή του active πλαισίου χαρακτήρων. Στη συνέχεια έκανε την εμφάνισή του η αλληλουχία των συμβόλων “b”, “b”, “a” οι τιμές των οποίων προστίθεντο συνεχώς και ανανέωναν την τιμή του active πλαισίου χαρακτήρων.

Ο αλγόριθμος Heuristic ALZ για να κάνει πρόβλεψη ανιχνεύει αυτό το νέο πλαίσιο χαρακτήρων μέσα στο trie. Είναι σημαντικό να τονιστεί ότι η αναζήτηση γίνεται μετά την ανανέωση του λεξικού του αλγορίθμου, γεγονός που εξασφαλίζει ώστε να υπάρχει τουλάχιστον μία εμφάνιση του εκάστοτε active πλαισίου χαρακτήρων στο trie.

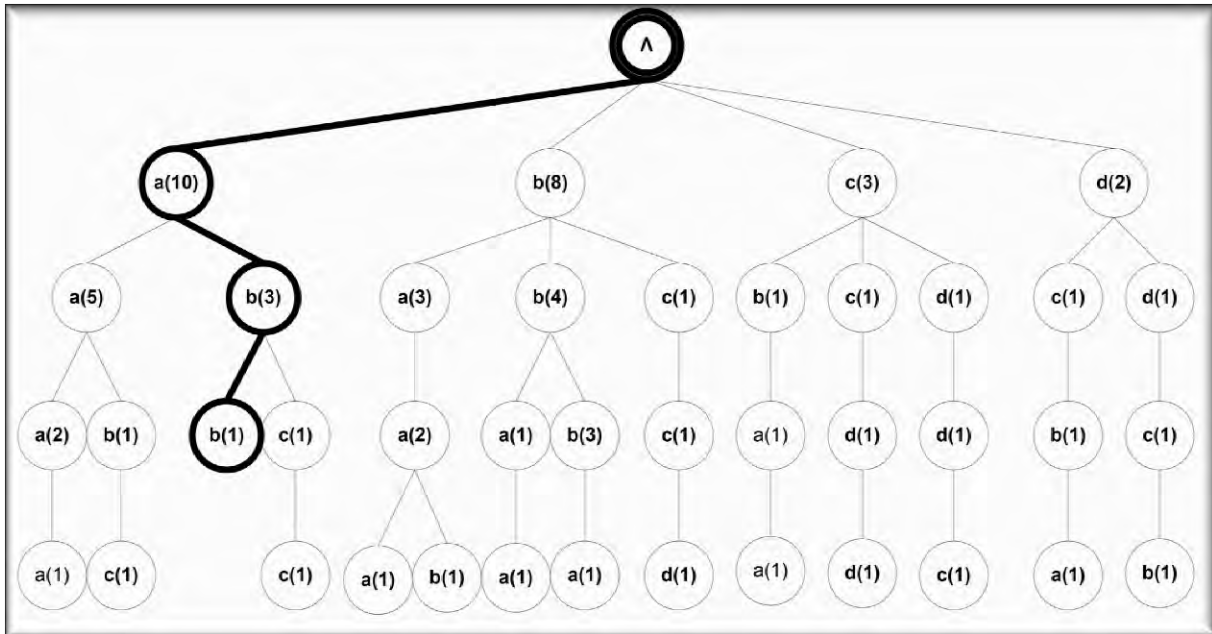
¹ Ως χρονική στιγμή ορίζεται κάθε νέα είσοδος δεδομένων στο σύστημα.



Εικόνα 3.4. Εύρεση στο trie του active πλαισίου χαρακτήρων “bba”.

Από τη στιγμή που γίνει εύρεση του πλαισίου ελέγχεται αν υπάρχει στο trie ιστορικό που να αφορά στο ποιος χαρακτήρας ακολούθησε σε παρελθούσα χρονική στιγμή το πλαίσιο χαρακτήρων για το οποίο γίνεται λόγος.

Αν το πλαίσιο εισόδου αφορά κάποιο φύλλο (leaf) στο trie τότε δεν είναι δυνατόν να υπάρχει ιστορία που να επιτρέπει την πρόβλεψη. Στην περίπτωση αυτή ο αλγόριθμος λειτουργεί όπως ο Active LeZi και η πρόβλεψη γίνεται με βάση το σύμβολο που συγκεντρώνει τα μεγαλύτερα ποσοστά πιθανοτήτων.



Εικόνα 3.5. Εύρεση του active πλαισίου χαρακτήρων “abb” που οδηγεί σε φύλλο του trie.

Αν όμως υπάρχει ιστορία έπειτα από το active πλαίσιο εισόδου, όπως στην περίπτωση της εικόνας **3.4**, τότε η πρόβλεψη γίνεται με βάση το σύμβολο που ακολουθεί. Στο παράδειγμα μας λοιπόν βλέπουμε ότι υπάρχει ιστορία που να ακολουθεί το active πλαίσιο χαρακτήρων και αυτό είναι το σύμβολο “a”. Αυτό είναι και το σύμβολο που θα κάνει ως πρόβλεψη ο αλγόριθμος Heuristic ALZ.

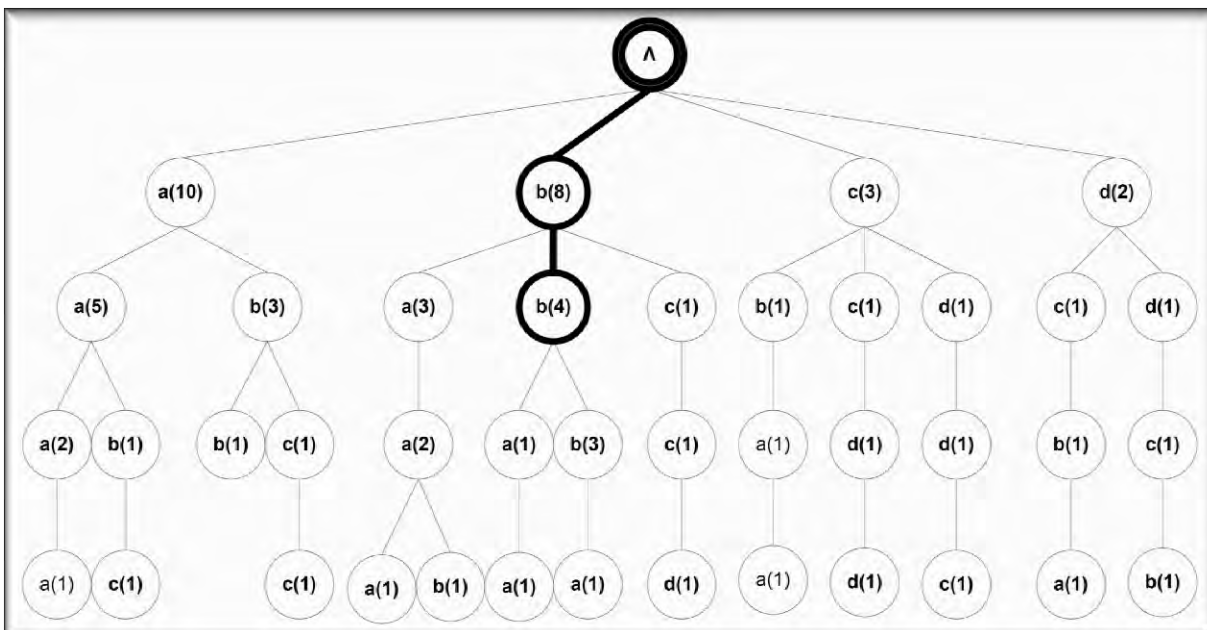
Τι γίνεται όμως στην περίπτωση που το active πλαίσιο χαρακτήρων ακολουθείται από περισσότερα από ένα σύμβολα; Στην περίπτωση αυτή ο γενικός κανόνας είναι να τίθεται ως πρόβλεψη το σύμβολο με τη μεγαλύτερη συχνότητα εμφάνισης.

Η λογική πίσω από αυτή τη θεώρηση έχει να κάνει με το γεγονός ότι ο αλγόριθμος στηρίζει τη λειτουργία του στην ικανότητά του να ανακαλύπτει τις επαναλαμβανόμενες αλληλουχίες συμβόλων που συμμετέχουν στο σύστημα (pattern recognition). Συνεπώς αν μία ακολουθία συμβόλων επαναλαμβάνεται τότε το μονοπάτι που αυτή σχηματίζει πάνω στο trie θα έχει και τις περισσότερες εμφανίσεις και θα μπορεί να διαχωριστεί από παρακλάδια του trie που δημιουργήθηκαν περιστασιακά λόγω των περιπτώσεων εισόδου και συμβολοσειρών που διαφέρουν λίγο από αυτή που επαναλαμβάνεται συνεχώς (θόρυβος).

Η αδυναμία της ανωτέρω θεώρησης έχει να κάνει με τις περιπτώσεις όπου η συχνότητα εμφάνισης των συμβόλων που ακολουθούν το active πλαίσιο χαρακτήρων δεν μπορεί να αποτελέσει παράγοντα πρόβλεψης καθώς αυτές μπορεί να διαφέρουν λίγο. Για την αντιμετώπιση λοιπόν όλων των ανωτέρω περιπτώσεων όπου το active πλαίσιο χαρακτήρων ακολουθείται από περισσότερα από ένα σύμβολα, ο Heuristic ALZ εφαρμόζει μία ευριστική μέθοδο, στην οποία οφείλει και το όνομά του.

Χρησιμοποιείται μία σταθερά **heuristicFactor** η οποία παίρνει την τιμή της εμπειρικά (μετά από ένα αριθμό πειραμάτων αυτή έχει καθοριστεί σε 0.167), και αφορά σε ποσοστά εμφάνισης των διάφορων συμβόλων σε κάποιο συγκεκριμένο τμήμα του trie. Πιο συγκεκριμένα η σταθερά αυτή αποτελεί το μέτρο σύγκρισης για το ποσοστό που αποτελεί η διαφορά στη συχνότητα εμφάνισης μεταξύ δύο συμβόλων για τα οποία γίνεται λόγος προς τη συχνότητα εμφάνισης του συμβόλου από τα οποία προέρχονται τα δύο σύμβολα (που βρίσκεται ένα επίπεδο πιο πάνω στο trie).

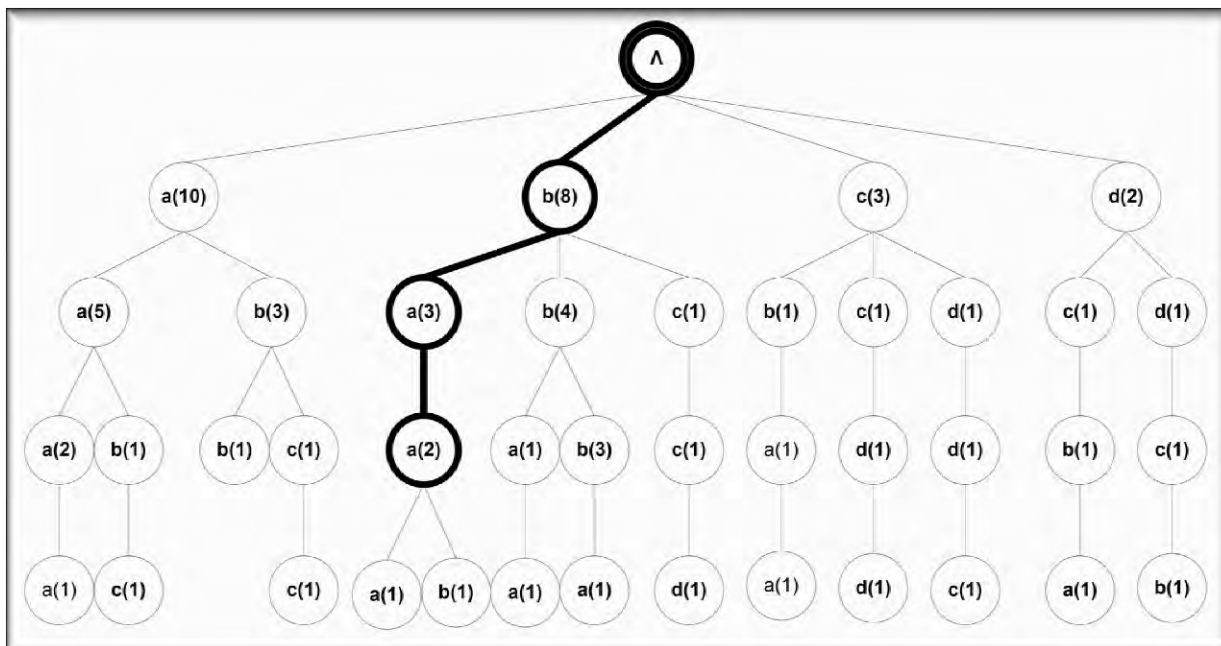
Στο παράδειγμα της εικόνας **3.6** το active πλαίσιο χαρακτήρων “bb” ακολουθείται από τα σύμβολα “a” και “b” τα οποία διαθέτουν τις συχνότητες εμφάνισης “1” και “3” αντίστοιχα. Ο αλγόριθμος ψάχνει την ιστορία που ακολουθεί το active πλαίσιο χαρακτήρων και συγκρίνει ανά δύο σύμβολα το ποσοστό της διαφοράς τους με τη σταθερά heuristicFactor. Το ποσοστό της διαφοράς των δύο συμβόλων στην περίπτωσή μας είναι $(3 - 1) / 4 = 0,5$.



Εικόνα 3.6. Εύρεση του active πλαισίου χαρακτήρων "bb".

Σε περίπτωση που το ποσοστό είναι μεγαλύτερο από την τιμή του heuristicFactor τότε προτείνεται το σύμβολο που διαθέτει τη μεγαλύτερη συχνότητα εμφάνισης, στην περίπτωση μας δηλαδή όπου το ποσοστό της διαφοράς των δύο συμβόλων είναι μεγαλύτερο από τη μεταβλητή heuristicFactor, θα προταθεί το σύμβολο "b".

Όταν όμως το ποσοστό είναι μικρότερο από την τιμή του heuristicFactor τότε προτείνεται το σύμβολο με τη μεγαλύτερη πιθανότητα εμφάνισης εκ των δύο που μπορεί τελικά να διαθέτει και την μικρότερη συχνότητα εμφάνισης.



Εικόνα 3.7. Εύρεση του active πλαισίου χαρακτήρων “baa” που οδηγεί σε σύμβολα με ίδιες συχνότητες εμφάνισης.

Η διαδικασία αυτή συνεχίζεται μέχρι να γίνει έλεγχος όλων των συμβόλων που αποτελούν την ιστορία που ακολουθεί το active πλαίσιο χαρακτήρων, που στη χειρότερη των περιπτώσεων μπορεί να αφορά το σύνολο του λεξιλογίου (256 σύμβολα) του συστήματος.

3.4 Ρύθμιση Παραμέτρων

Ο αλγόριθμος Heuristic ALZ παραμετροποιείται εύκολα έτσι ώστε η συμπεριφορά του να αντικατοπτρίζει κάθε φορά είτε γρήγορα αποτελέσματα σε συνδυασμό με ικανοποιητική πρόβλεψη, είτε βέλτιστη πρόβλεψη με άμεση συνέπεια στο χρόνο επεξεργασίας των δεδομένων του συστήματος.

Οι παραμετροποιήσιμες σταθερές του αλγορίθμου βρίσκονται συγκεντρωμένες στο header αρχείο “parameters.h” έτσι ώστε να είναι δυνατή στο χρήστη η άμεση ρύθμιση όλων των παραμέτρων λειτουργίας του αλγορίθμου.

Ο καθορισμός των τελικών τιμών των παραμέτρων του αλγορίθμου κατέστη δυνατός, μετά από ένα αριθμό πειραμάτων στα οποία αξιολογήθηκε η επίδοση του πάνω στα αντικείμενα της ποιότητας της πρόβλεψης (όπου το μεγαλύτερο ποσοστό επιτυχίας του αλγορίθμου αποτελούσε και το επιθυμητό αποτέλεσμα) και του χρόνου ολοκλήρωσης της επεξεργασίας μίας συμβολοσειράς εισόδου μεγέθους 20000 χαρακτήρων (όπου το ζητούμενο ήταν ο μικρότερος δυνατός χρόνος τερματισμού).

Ο μόνος περιορισμός κατά την εναλλαγή των χαρακτήρων ήταν να ικανοποιούνται οι προϋποθέσεις επιλογής του επόμενου χαρακτήρα, όπως για παράδειγμα αυτές που απεικονίζονται στα ακόλουθα δύο σχήματα. Σε αυτά αναπαρίσταται ένα δίκτυο κινητής τηλεφωνίας με τους σταθμούς εξυπηρέτησής των πελατών του (base stations). Η εξυπηρέτηση ενός πελάτη από τον κατάλληλο σταθμό εδάφους έχει άμεση σχέση με την κίνηση του πελάτη αυτού στο χώρο. Έτσι για το παράδειγμά μας, δεν είναι δυνατό ο πελάτης ενώ εξυπηρετείται από το σταθμό “a”, την επόμενη χρονική στιγμή να εξυπηρετηθεί από το σταθμό “d”, καθότι δεν υπάρχει διαδρομή που να ενώνει αυτούς τους δύο προορισμούς.

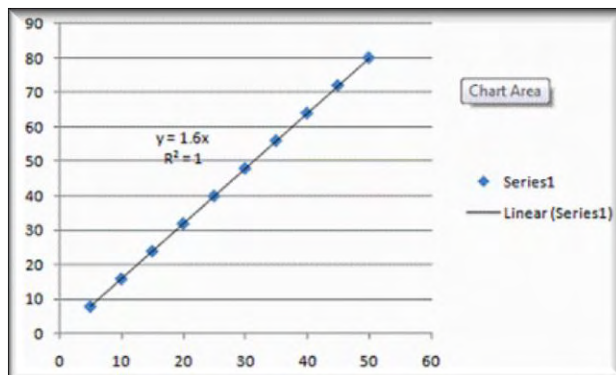
The figure consists of two parts. On the left is a planar map \$M\$ represented as a collection of polygons meeting at vertices. Some regions are shaded light blue. These regions are labeled with lowercase letters: 'h' (top-left), 'a' (top-middle), 'c' (top-right), 'g' (middle-left), 'b' (middle), 'd' (middle-right), 'f' (bottom-left), and 'e' (bottom). Dashed lines indicate boundaries between regions. On the right is a graph \$G\$. Vertices are represented by black dots and are labeled with lowercase letters: 'a' (top-left), 'b' (top-middle), 'c' (top-right), 'h' (middle-left), 'g' (middle), 'd' (middle-right), 'f' (bottom-left), and 'e' (bottom-right). Edges connect vertices that share a boundary segment in the map \$M\$: \$(a,b)\$, \$(b,c)\$, \$(c,d)\$, \$(d,e)\$, \$(e,f)\$, \$(f,g)\$, \$(g,h)\$, \$(h,a)\$, \$(a,h)\$, \$(b,g)\$, \$(g,d)\$, \$(d,e)\$, \$(e,f)\$, \$(f,g)\$, \$(g,h)\$, and a diagonal edge \$(b,d)\$. The graph \$G\$ is a planar graph with 8 vertices and 17 edges.

Για τη μελέτη της συμπεριφοράς του αλγορίθμου Heuristic ALZ χρησιμοποιούνται τρία είδη γραμμών τάσεων (trendline), για τη γραφική αναπαράσταση της τάσης που παρουσιάζει το σύστημα σε ότι αφορά την ποιότητα πρόγνωσης και το χρόνο ολοκλήρωσης των υπολογισμών του.

Μία γραμμή τάσης είναι αξιόπιστη όταν η τιμή της αποφασιστικής συνιστώσας (coefficient of determination), που συμβολίζεται με R^2 , προσεγγίζει ή έχει την τιμή 1. Πιο συγκεκριμένα χρησιμοποιούνται οι ακόλουθες γραμμές τάσης:

α. Γραμμική (Linear)

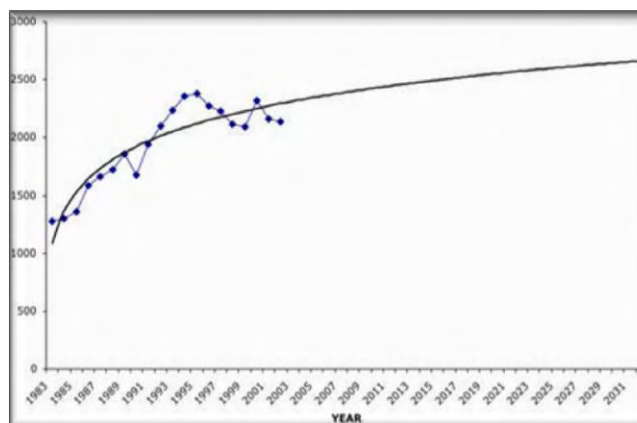
Η γραμμική γραμμή τάσης χρησιμοποιείται με απλά σύνολα δεδομένων η γραφική αναπαράσταση των οποίων μπορεί να αναπαρασταθεί με μία ευθεία γραμμή.



Εικόνα 3.9. Υπόδειγμα γραμμικής γραμμής τάσης (Linear Trendline)

β. Λογαριθμική (Logarithmic)

Η λογαριθμική γραμμή τάσης είναι χρήσιμη όταν ο ρυθμός με τον οποίο αλλάζουν τα δεδομένα που αναπαρίστανται γραφικά αρχικά αυξάνει ή μειώνεται γρήγορα μέχρι κάποιο σημείο και στη συνέχεια αυτός ισιώνει.

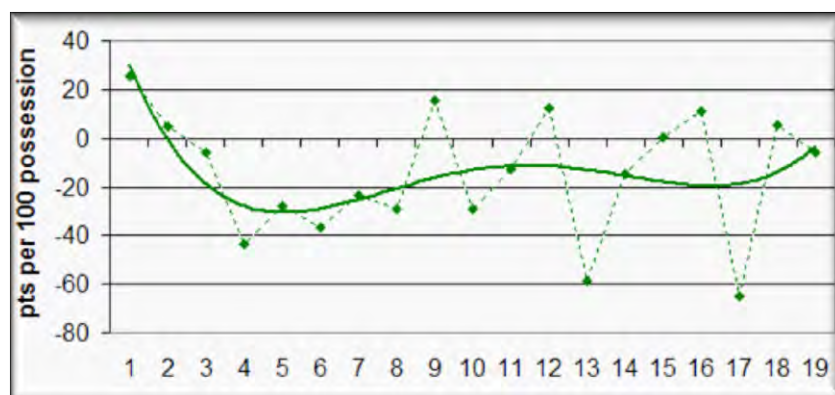


Εικόνα 3.10. Υπόδειγμα λογαριθμικής γραμμής τάσης (Logarithmic Trendline)

γ. Πολυωνυμική (Polynomial)

Μία πολυωνυμική καμπύλη γραμμή τάσης χρησιμοποιείται όταν υπάρχει διακύμανση των τιμών των δεδομένων που αναπαρίστανται γραφικά. Η τάξη του πολυωνύμου

μπορεί να καθοριστεί από τον αριθμό των διακυμάνσεων που παρατηρούνται στα δεδομένα ή αλλιώς από τον αριθμό λόφων – κοιλάδων που εμφανίζονται στο γράφημα.



Εικόνα 3.11. Υπόδειγμα πολυωνυμικής γραμμής τάσης (Polynomial Trendline)

Στη συνέχεια ακολουθεί ανάλυση των παραμετροποιήσιμων σταθερών του αλγορίθμου Heuristic ALZ.

3.4.1 Η σταθερά **observationFactor** χρησιμοποιείται για τον έλεγχο της ακρίβειας του prediction και καθορίζει τον αριθμό των συμβόλων που θα χρησιμοποιούνται κάθε φορά για τον έλεγχο της διακύμανσης του prediction accuracy.

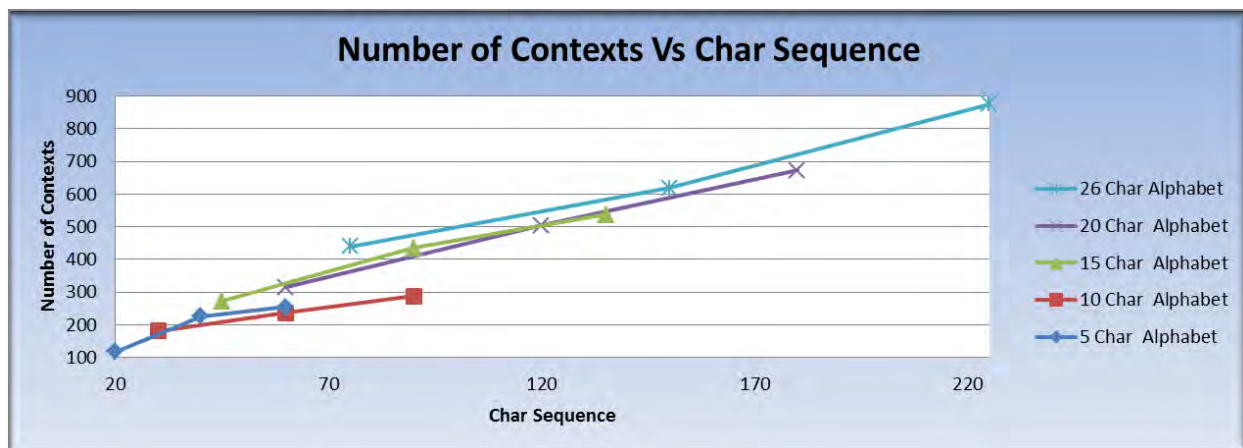
3.4.2 Η σταθερά **alphabetSize** καθορίζει το πλήθος των διαφορετικών συμβόλων που μπορεί να αναγνωρίσει το σύστημα.

Το πρόγραμμα είναι ρυθμισμένο ώστε να δέχεται το σύνολο των ASCII συμβόλων, δηλαδή τα 128 διαφορετικά σύμβολα αυτού.

Πως επηρεάζει την επίδοση του συστήματος η σταθερά alphabetSize;

Η αύξηση του αριθμού των συμβόλων αυξάνει την εντροπία του συστήματος, με άμεσα αρνητικά αποτελέσματα στην ακρίβεια και το χρόνο ολοκλήρωσης των υπολογισμών της πρόβλεψης.

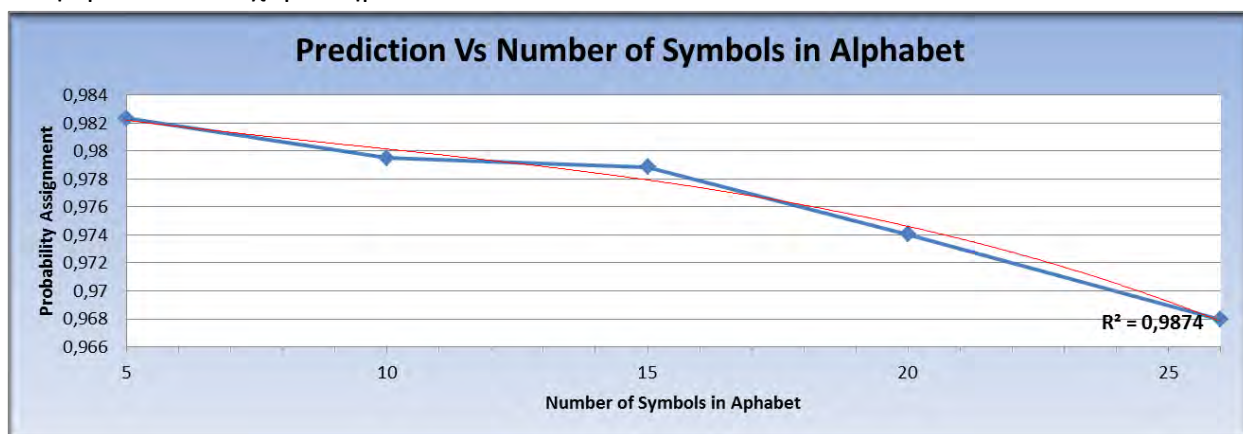
Όσο μεγαλύτερο είναι το πλήθος των διαφορετικών συμβόλων που χρησιμοποιούνται, τόσο περισσότεροι είναι και οι υπολογισμοί των πιθανοτήτων που θα πρέπει να κάνει το σύστημα λόγω αύξησης των δυναμένων μονοπατιών που λαμβάνονται υπόψη στον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου.



Εικόνα 3.12. Διακύμανση της ποσότητας των φράσεων που λαμβάνει υπόψη του ο Heuristic ALZ για τον υπολογισμό των πιθανοτήτων εμφάνισης του εκάστοτε συμβόλου σε σύγκριση με το μέγεθος της συμβολοσειράς εισόδου του συστήματος.

Στην εικόνα **3.12** παρουσιάζεται γραφικά ο παραπάνω ισχυρισμός. Με την αύξηση του αριθμού των συμβόλων, βλέπουμε ότι για την ίδια ποσότητα δεδομένων εισόδου (που αριθμεί 20000 χαρακτήρες για το παράδειγμά μας) ο αριθμός των φράσεων που λαμβάνονται υπόψη για τον καθορισμό της πιθανότητας εμφάνισης του επόμενου συμβόλου αυξάνεται συνεχώς, γεγονός που μπορεί να καθυστερήσει δραματικά την πρόβλεψη του συστήματος, ιδιαίτερα όταν η εντροπία του συστήματος είναι μεγάλη. Το σύστημα έχει μεγάλη εντροπία όσο πιο άναρχη είναι η ακολουθία των χαρακτήρων που επεξεργάζεται το σύστημα.

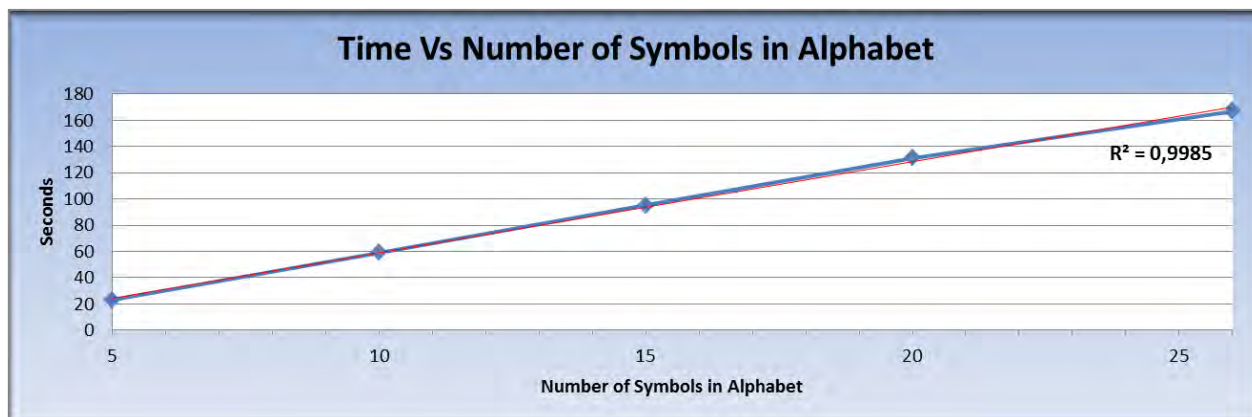
Στα ακόλουθα διαγράμματα απεικονίζεται η επίδοση του αλγορίθμου Heuristic ALZ όταν τα διαφορετικά σύμβολα που εμφανίζονται στη συμβολοσειρά εισόδου είναι αντίστοιχα 5, 10, 15, 20 και 26. Ως διαφορετικά λογίζονται τα σύμβολα όταν αντιστοιχούν σε διαφορετικό ASCII χαρακτήρα.



Εικόνα 3.13. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε σύγκριση με το μέγεθος του λεξικού του συστήματος.

Η αύξηση του αριθμού των συμβόλων στο αλφάβητο του προγνώστη έχει ως αποτέλεσμα τη μείωση της ακρίβειας του προγνώστη. Η κόκκινη γραμμή αναπαριστά την τάση

της ακρίβειας του συστήματος σε σχέση με το αριθμό των συμβόλων στο αλφάβητο του προγνώστη. Η απόκλιση αυτής από τα πραγματικά δεδομένα επίδοσης του αλγορίθμου είναι αμελητέα ($R^2 = 0,984$).



Εικόνα 3.14. Χρόνος ολοκλήρωσης εκτέλεσης του Heuristic ALZ σε σύγκριση με το μέγεθος του λεξικού του συστήματος.

Η ίδια αρνητική εικόνα σε ότι αφορά την επίδοση του αλγορίθμου κατά την αξιολόγηση του σε ότι αφορά το χρόνο ολοκλήρωσης των υπολογισμών του προγνώστη. Όσο περισσότερα τα σύμβολα στο αλφάβητο του προγνώστη τόσο περισσότερος χρόνος απαιτείται για την ολοκλήρωση της διαδικασίας πρόγνωσης. Πλήρης είναι για άλλη μία φορά η ταύτιση της γραμμής αναπαράστασης της τάσης του συστήματος σε ότι αφορά το χρόνο ολοκλήρωσης των υπολογισμών σε σχέση με τον αριθμό των συμβόλων στο αλφάβητο του προγνώστη ($R^2 = 0,9985$).

Το πλεονέκτημα από τη χρήση ενός λεξικού του μεγέθους που διαθέτει ο Heuristic ALZ, έχει να κάνει με το ότι ο αλγόριθμος μπορεί να χρησιμοποιηθεί ως έχει, σε οποιοδήποτε είδος δεδομένων, αρκεί αυτά να μπορούν να παρασταθούν στο ASCII λεξικό.

Το μειονέκτημα όμως από τη χρήση αυτού, είναι ότι ο αλγόριθμος κατά τον υπολογισμό των πιθανοτήτων εμφάνισης του εκάστοτε συμβόλου, μπορεί να εκτελεί άσκοπες αναζητήσεις στο σχηματιζόμενο ψηφιακό δένδρο - trie για σύμβολα που έτσι και αλλιώς δεν αναμένεται να συναντηθούν, αφού δεν περιλαμβάνονται σε κάποια από τις φράσεις που έχουν καταχωρηθεί στο λεξικό του συστήματος. Η ιδιότητα αυτή του αλγορίθμου ρίχνει την απόδοση του συστήματος.

3.4.3 Άλλη μία παράμετρος η οποία έχει άμεση σχέση με το μέγεθος του αλφαβήτου του συστήματος και επηρεάζει καταλυτικά τη συμπεριφορά του αλγορίθμου αφορά το μήκος της συμβολοσειράς η οποία εισέρχεται ως είσοδος στο σύστημα και επαναλαμβάνεται.

Πως επηρεάζει το μέγεθος της συμβολοσειράς εισόδου την επίδοση του συστήματος;

Όσο μεγαλώνει ο αριθμός των διαφορετικών συμβόλων που μετέχουν στη συμβολοσειρά εισόδου είδαμε ότι μειώνεται η ικανότητα πρόβλεψης του αλγορίθμου για την ίδια ποσότητα δεδομένων. Η ιδιότητα αυτή του αλγορίθμου σε συνδυασμό με την αύξηση του μεγέθους της συμβολοσειράς εισόδου ενισχύει την μειωμένη αποδοτικότητα του σε σημείο να απαιτούνται αρκετά περισσότερα δεδομένα για επεξεργασία από αυτόν προκειμένου να φτάσει σε αποδεκτά επίπεδα πρόβλεψης.

Το μήκος της συμβολοσειράς εισόδου αφορά τη διαδικασία παραμετροποίησης του αλγορίθμου, μόνο στην περίπτωση της On Line λειτουργίας αυτού, δηλαδή κατά την πρόβλεψη των γεγονότων που εκτυλίσσονται σε πραγματικό χρόνο.

Στην περίπτωση αυτή ο δειγματοληπτικός έλεγχος του συστήματος για τον έλεγχο της ακρίβειας πρόβλεψης (που αποτελεί την είσοδο αυτού), θα πρέπει να γίνεται με τέτοια συχνότητα, ώστε από τη μία να μην χάνεται καμία κατάσταση (βέλτιστη αντίληψη της πραγματικής κατάστασης από το σύστημα), ενώ από την άλλη στο σύστημα να εισέρχεται ο ελάχιστος δυνατός πληθάρθμος συμβόλων.

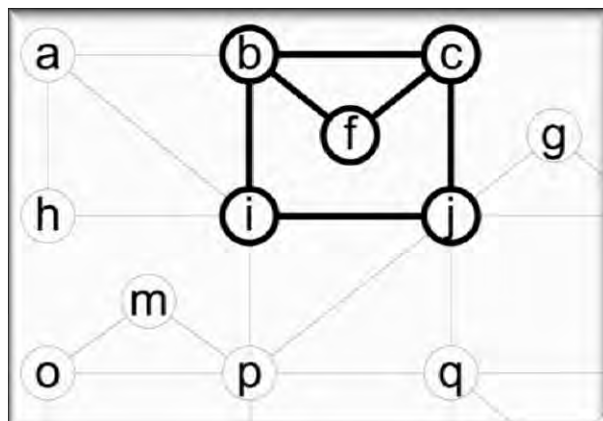
Στα ακόλουθα διαγράμματα αντικατοπτρίζεται η επίδραση του διαφορετικού μήκους μίας συμβολοσειράς εισόδου στην επίδοση του αλγορίθμου. Τα patterns των συμβολοσειρών εισόδου αφορούν κάθε φορά διαφορετικό αριθμό επιτρεπτών συμβόλων όπως και διαδρομών μεταξύ των συμβόλων αυτών.

Η μελέτη της συμπεριφοράς του αλγορίθμου έγινε για αλφάβητα των πέντε (5), δέκα (10), δεκαπέντε (15), είκοσι (20) και είκοσι έξι (26) διαφορετικών χαρακτήρων και για διαφορετικό πληθάρθμο συμβόλων ανά συμβολοσειρά εισόδου (ανάλογα με την περίπτωση).

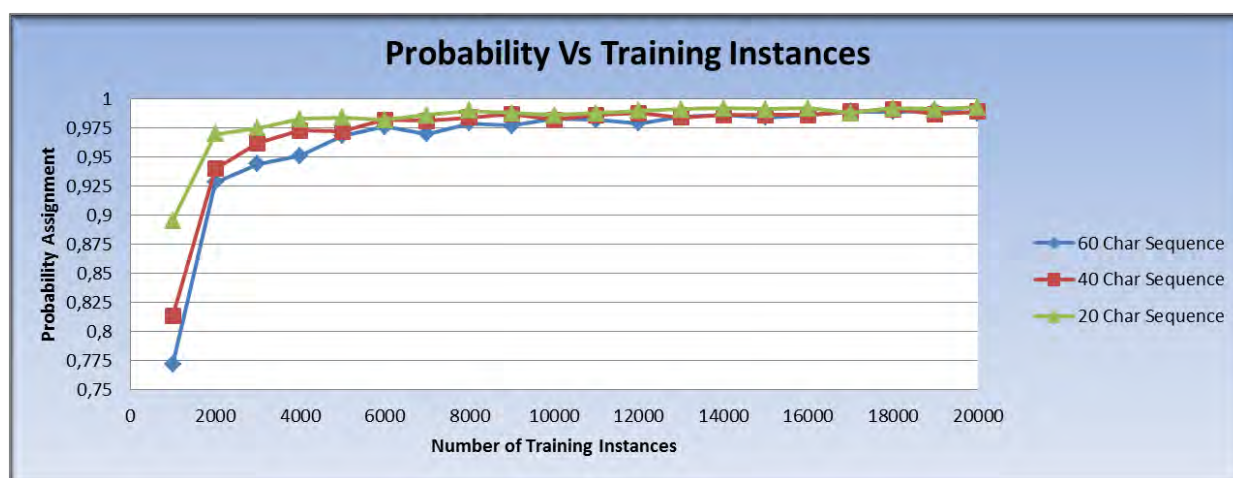
Σε όλες τις περιπτώσεις που ακολουθούν επιβεβαιώνονται οι παραπάνω ισχυρισμοί. Η επίδοση του αλγορίθμου σε ότι αφορά την ικανότητα πρόβλεψης αν θα μπορούσε να περιγραφεί μαθηματικά αυτό θα γινόταν βέλτιστα με τη χρήση μίας λογαριθμικής συνάρτησης.

Σε ότι αφορά το χρόνο ολοκλήρωσης των υπολογισμών του συστήματος είναι ξεκάθαρο ότι, η αύξηση του αριθμού των συμβόλων στο αλφάβητο του συστήματος αλλά και η αύξηση του πληθάρθμου των συμβόλων που επαναλαμβάνονται ανά τακτά χρονικά διαστήματα επηρεάζουν άμεσα τη συμπεριφορά του αλγορίθμου.

Σε ότι αφορά τη διαδικασία παραμετροποίησης του αλγορίθμου, απαιτείται κάθε φορά ο αριθμός των συμβόλων που αποτελούν το αλφάβητο του προγνώστη να παραμένει πάντα στο ελάχιστο δυνατό, ενώ σε ότι αφορά τη ρύθμιση της On Line λειτουργίας του αλγορίθμου θα πρέπει η δειγματοληψία για τον έλεγχο της ακρίβειας του συστήματος, να γίνεται με την είσοδο του ελάχιστου αριθμού συμβόλων έτσι ώστε να εξασφαλίζεται η πλήρης κάλυψη όλων των καταστάσεων του συστήματος με τον ελάχιστο αριθμό συμβόλων.



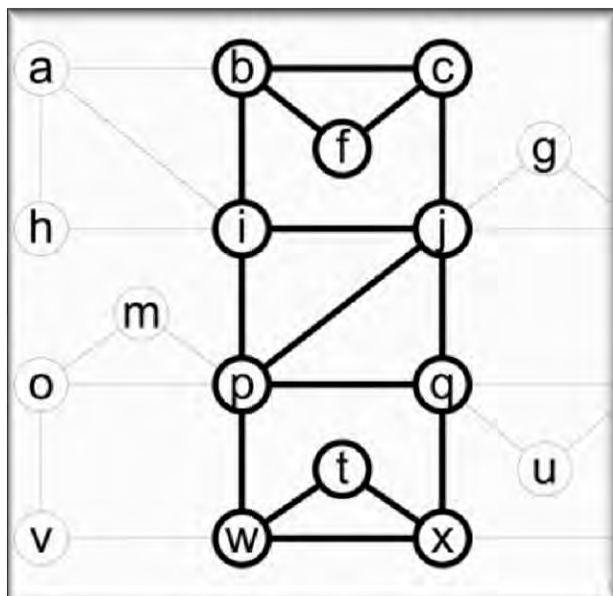
Εικόνα 3.15. Επιτρεπτά μονοπάτια σε αλφάβητο πέντε (5) συμβόλων για τη δημιουργία νέας συμβολοσειράς εισόδου.



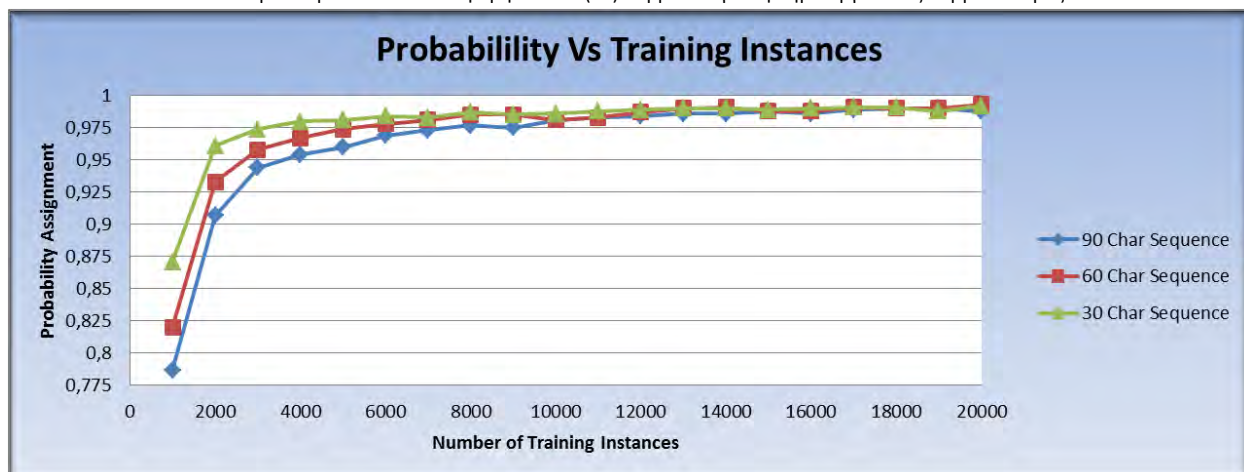
Εικόνα 3.16. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε αλφάβητο πέντε (5) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



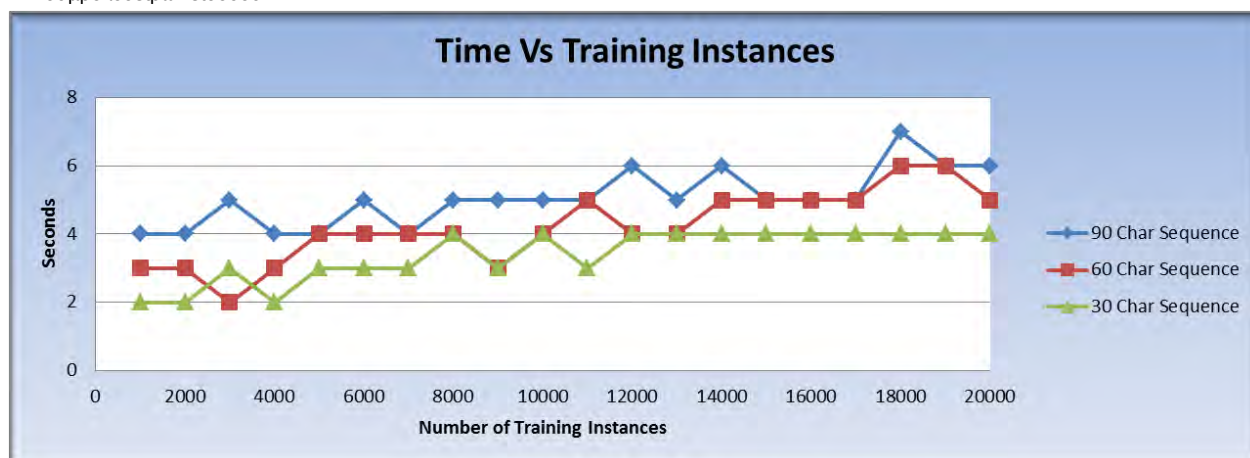
Εικόνα 3.17. Διακύμανση χρόνου εκτέλεσης του Heuristic ALZ σε αλφάβητο πέντε (5) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



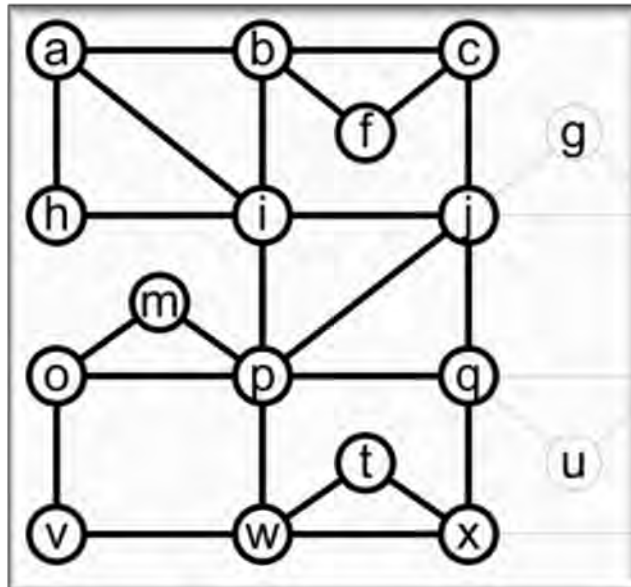
Εικόνα 3.18. Επιτρεπτά μονοπάτια σε αλφάβητο δέκα (10) συμβόλων για τη δημιουργία νέας συμβολοσειράς εισόδου.



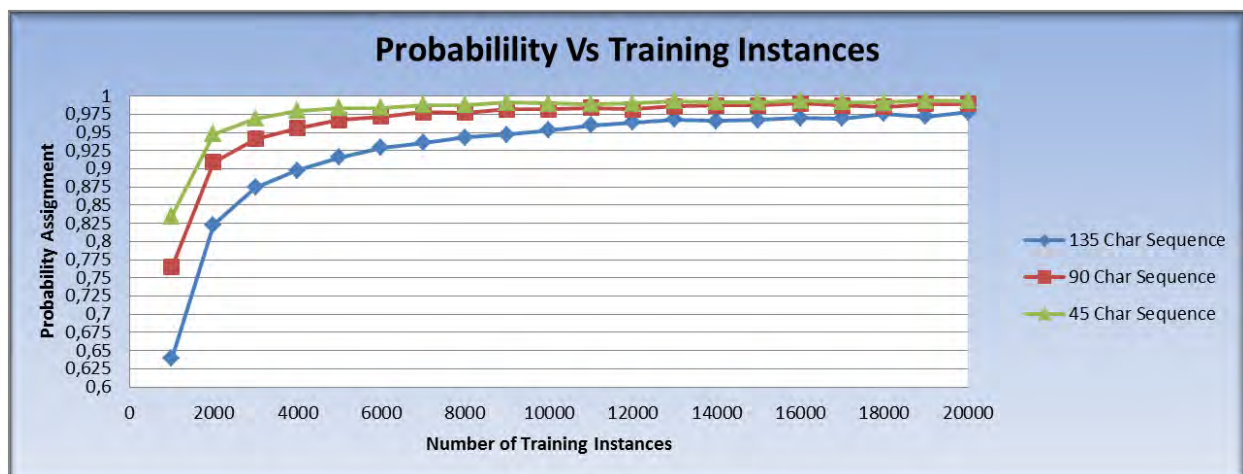
Εικόνα 3.19. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε αλφάβητο δέκα (10) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



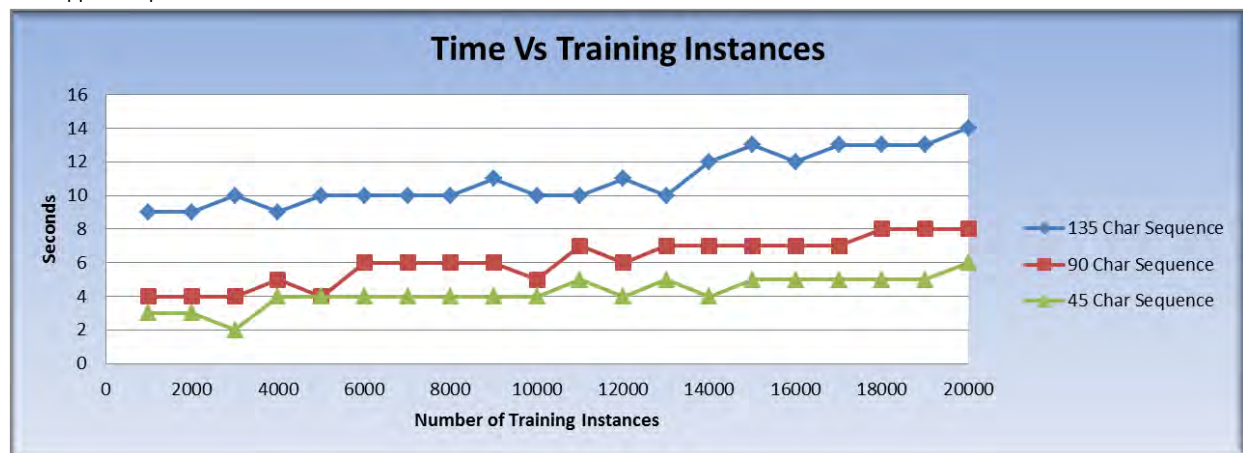
Εικόνα 3.20. Διακύμανση χρόνου εκτέλεσης του Heuristic ALZ σε αλφάβητο δέκα (10) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



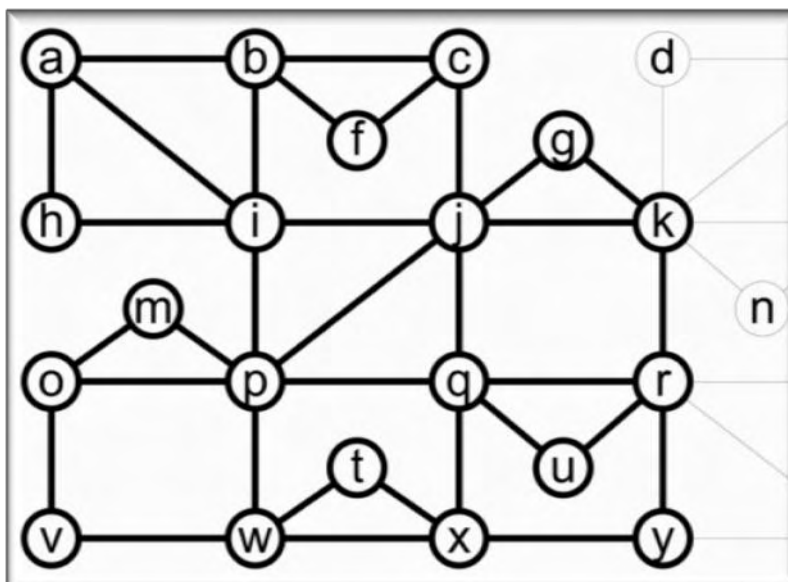
Εικόνα 3.21. Επιτρεπτά μονοπάτια σε αλφάβητο δέκα πέντε (15) συμβόλων για τη δημιουργία νέας συμβολοσειράς εισόδου.



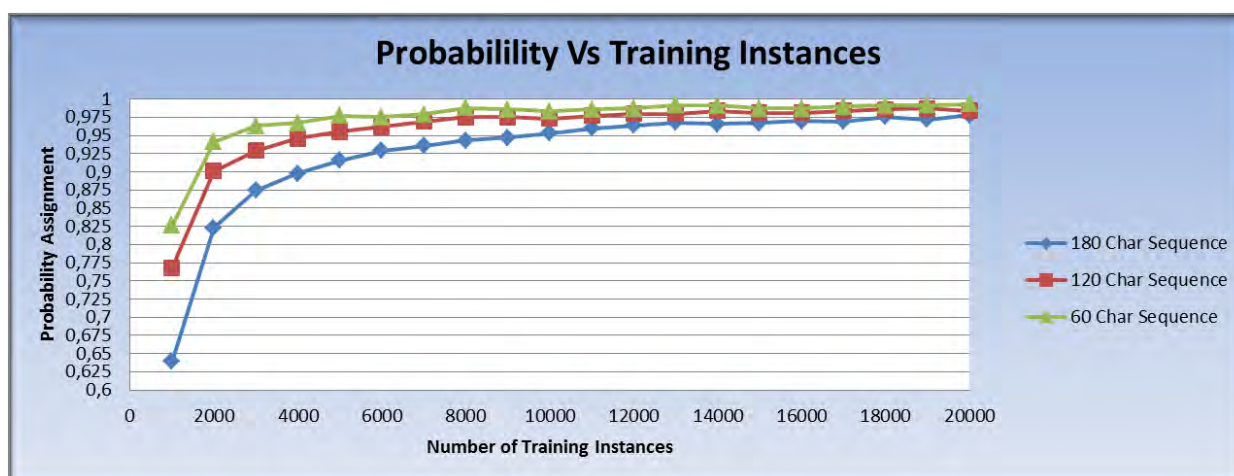
Εικόνα 3.22. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε αλφάβητο δέκα (10) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



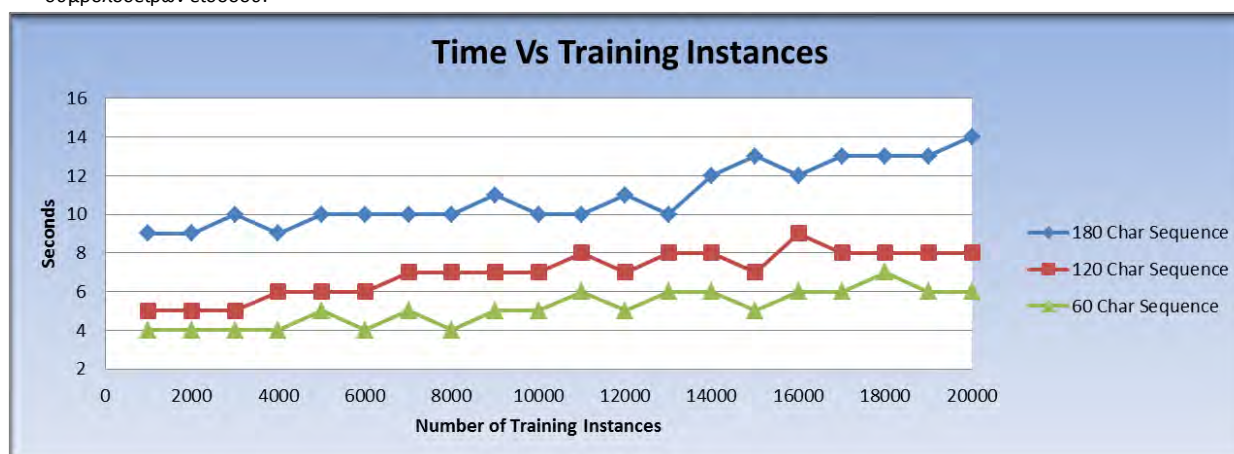
Εικόνα 3.23. Διακύμανση χρόνου εκτέλεσης του Heuristic ALZ σε αλφάβητο δέκα πέντε (15) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



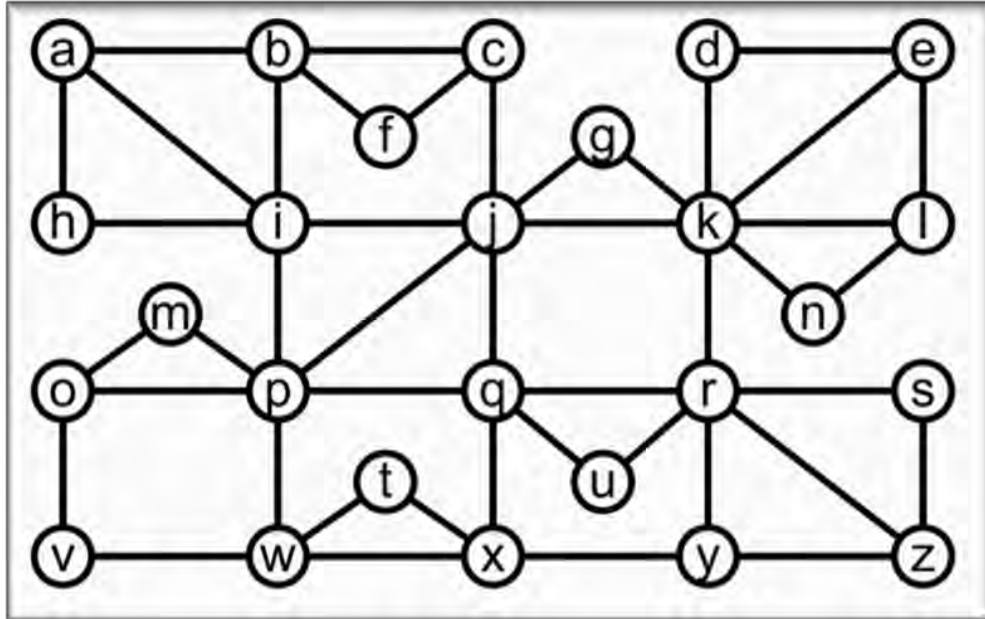
Εικόνα 3.24. Επιτρεπτά μονοπάτια σε αλφάβητο είκοσι (20) συμβόλων για τη δημιουργία νέας συμβολοσειράς εισόδου.



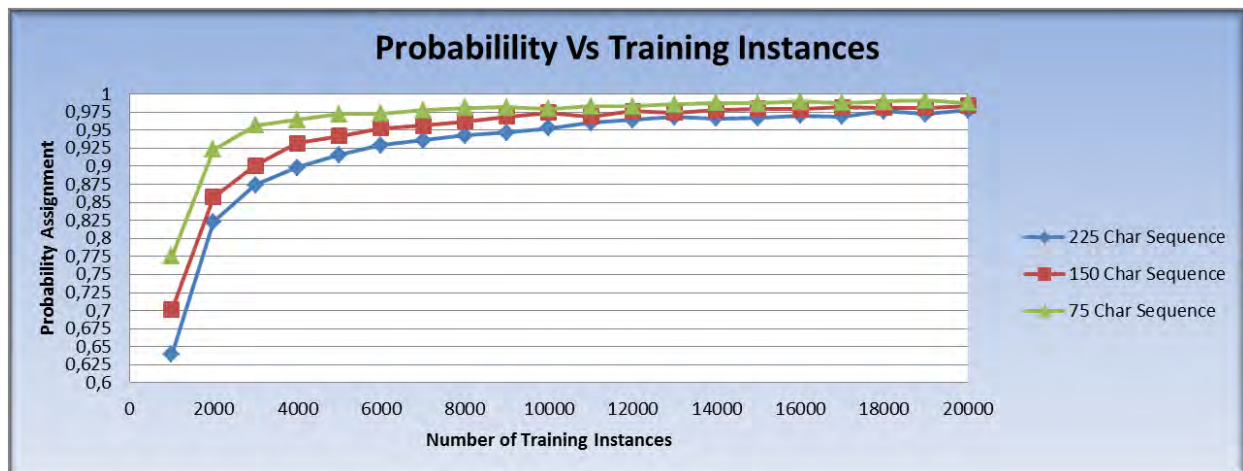
Εικόνα 3.25. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε αλφάβητο είκοσι (20) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



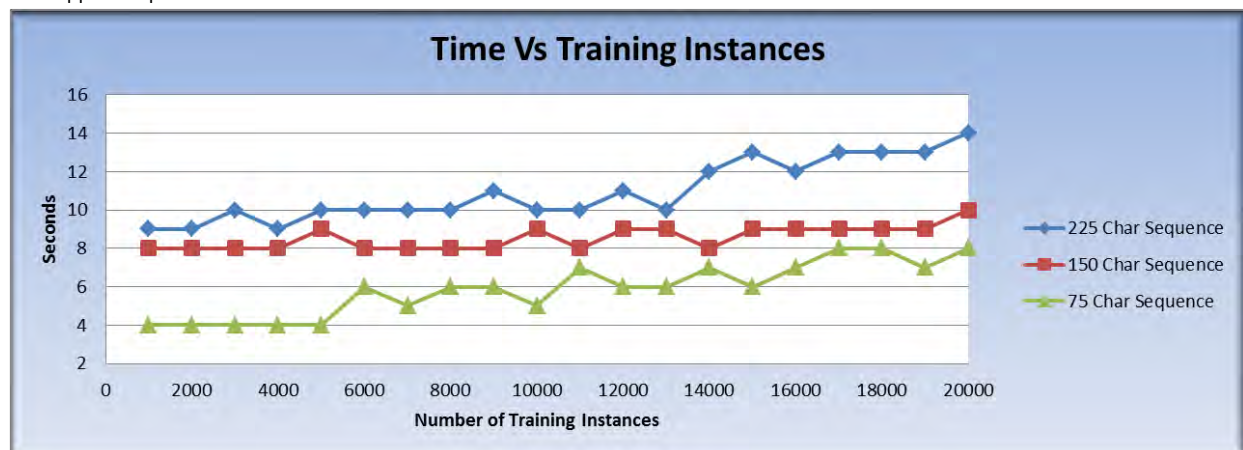
Εικόνα 3.26. Διακύμανση χρόνου εκτέλεσης του Heuristic ALZ σε αλφάβητο είκοσι (20) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



Εικόνα 3.27. Επιτρεπτά μονοπάτια σε αλφάβητο είκοσι έξι (26) συμβόλων για τη δημιουργία νέας συμβολοσειράς εισόδου.



Εικόνα 3.28. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε αλφάβητο είκοσι έξι (26) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.



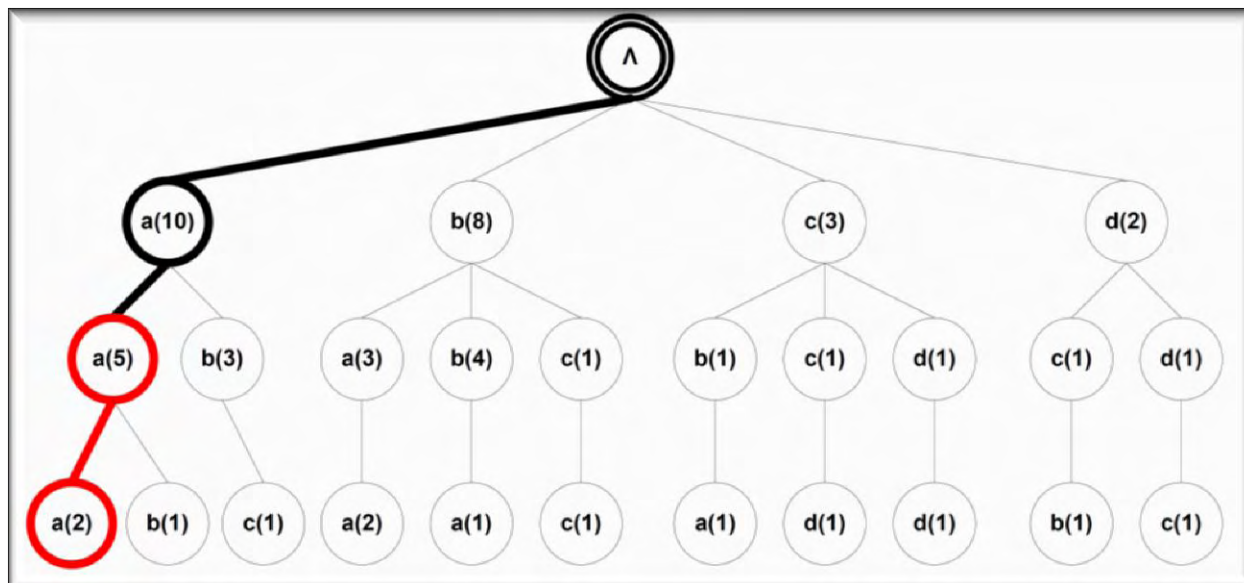
Εικόνα 3.29. Διακύμανση χρόνου εκτέλεσης του Heuristic ALZ σε αλφάβητο είκοσι έξι (26) συμβόλων για διαφορετικά μεγέθη συμβολοσειρών εισόδου.

3.4.4 Η σταθερά **expediteFactor** χρησιμοποιείται κατά τον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου και καθορίζει το μέγιστο βάθος στο παραγόμενο trie για το οποίο γίνονται υπολογισμοί.

Ποιος ο λόγος ύπαρξης της σταθεράς expediteFactor και πως επηρεάζει αυτή τη συμπεριφορά του αλγορίθμου;

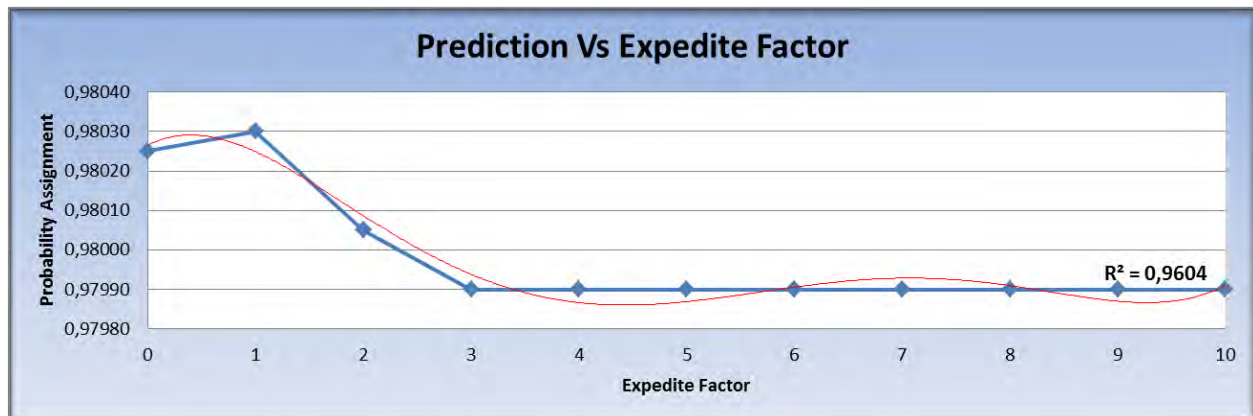
Η σταθερά χρησιμοποιείται ως παράγοντας βελτιστοποίησης για την απόδοση του αλγορίθμου. Συγκεκριμένα, καθορίζει τις τάξεις των Markov μοντέλων που θα λάβει υπόψη του το σύστημα προκειμένου να υπολογίσει τις πιθανότητες εμφάνισης του κάθε συμβόλου.

Για παράδειγμα, αν μετά την τμηματοποίηση που πραγματοποίησε ο αλγόριθμος η τελευταία συμβολοσειρά που έκανε την εμφάνισή της είναι η “aaa” που σημαίνει ότι το σύστημα βρίσκεται στο δεύτερο επίπεδο του trie, τότε με τιμή expediteFactor = 1, οι υπολογισμοί των πιθανοτήτων θα γίνουν μέχρι και ένα επίπεδο πιο πάνω (δηλαδή για τα επίπεδα δύο και ένα) και δεν θα λάβουν υπόψη τους το επίπεδο μηδέν(Εικόνα 3.30).

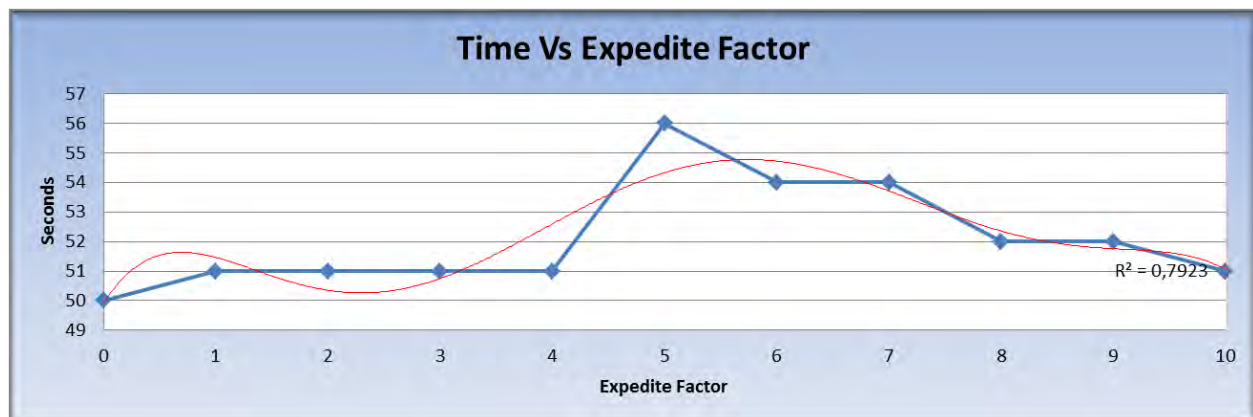


Εικόνα 3.30. Καθορισμός των τάξεων των Markov μοντέλων που θα χρησιμοποιηθούν για τον υπολογισμό των πιθανοτήτων εμφάνισης των συμβόλων του συστήματος, με τιμή σταθεράς expediteFactor = 1 και τελευταία συμβολοσειρά εισόδου την “aaa”.

Η συμπεριφορά του αλγορίθμου, σε ότι αφορά την επίδραση της σταθεράς expediteFactor πάνω στην ποιότητα πρόβλεψης του αλγορίθμου, περιγράφεται αρκετά αξιόπιστα ($R^2 = 0,9604$ για την περίπτωση της ποιότητας πρόβλεψης και $R^2 = 0,7923$ για την περίπτωση του χρόνου ολοκλήρωσης των υπολογισμών του αλγορίθμου) από μία πολυωνυμική συνάρτηση πέμπτου ($5^{ου}$) βαθμού (κόκκινη γραμμή) λόγω της διακύμανσης που παρουσιάζεται στις τιμές των αποτελεσμάτων του αλγορίθμου.



Εικόνα 3.31. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε σύγκριση με την παράμετρο ExpediteFactor.



Εικόνα 3.32. Χρόνος ολοκλήρωσης εκτέλεσης του Heuristic ALZ σε σύγκριση με την παράμετρο ExpediteFactor.

Η προσπάθεια των διαχειριστών του συστήματος θα πρέπει να είναι κάθε φορά η διατήρηση υψηλού ποσοστού επιτυχούς πρόγνωσης στο ελάχιστο δυνατό χρονικό διάστημα για να είναι το σύστημα εύχρηστο και λειτουργικό. Αυτό για την περίπτωση μας επιτυγχάνεται για τιμή σταθεράς `expediteFactor` = 1 που αποτελεί και την προ-τοποθετημένη τιμή λειτουργίας του αλγορίθμου. Η επιλογή αυτή βελτιώνει την ακρίβεια των υπολογισμών, ενώ παράλληλα μειώνει το χρόνο ολοκλήρωσης τους, ιδιαίτερα όταν το ύψος του trie είναι μεγάλο.

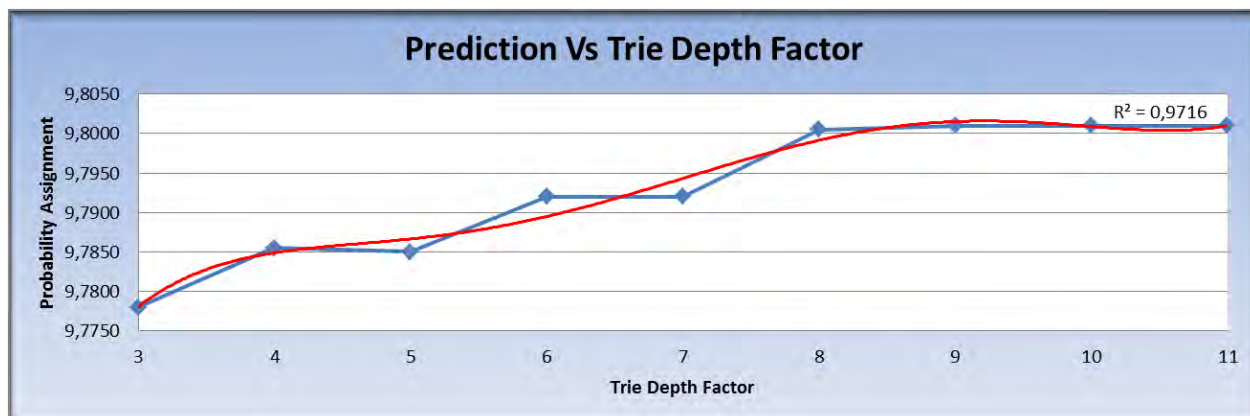
3.4.5 Η σταθερά `trieDepthFactor` βελτιστοποιεί επίσης τη συμπεριφορά του αλγορίθμου. Αυτή καθορίζει το μέγιστο μέγεθος των συμβολοσειρών οι οποίες θα λαμβάνονται υπόψη για τον υπολογισμό των πιθανοτήτων. Αυτό πρακτικά σημαίνει ότι ενώ το trie συνεχώς ανανεώνεται και αυξάνει σε βάθος με τη συνεχή ροή δεδομένων εισόδου, οι πιθανότητες εμφάνισης των συμβόλων θα υπολογίζονται για ένα ποσοστό των συμβολοσειρών που θα έχουν κάνει την εμφάνισή τους στο trie.

Πως επηρεάζει την ποιότητα πρόβλεψης η χρήση της σταθεράς trieDepthFactor;

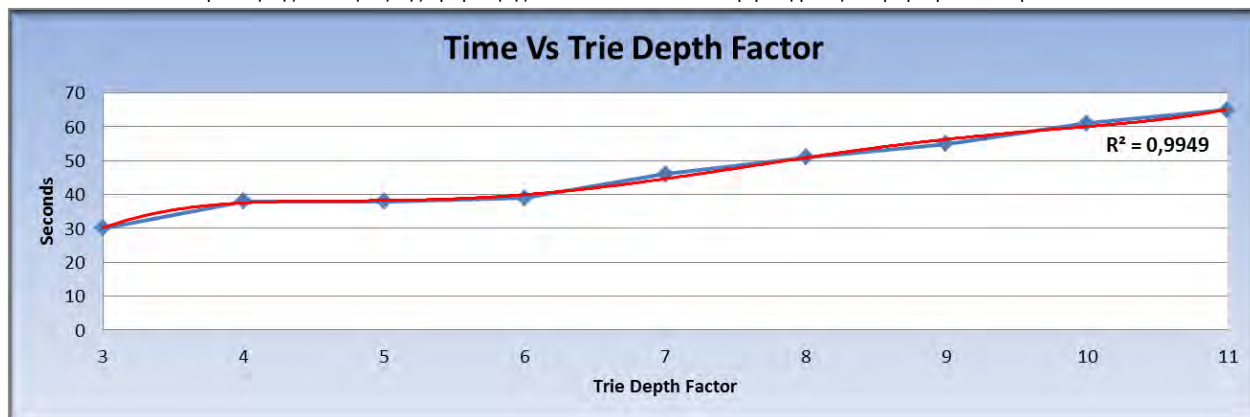
Αν και φαινομενικά η αφαίρεση ενός ποσοστού συμβολοσειρών από τη διαδικασία υπολογισμού των πιθανοτήτων εμφάνισης των συμβόλων που μετέχουν στο αλφάβητο του συστήματος μπορεί διαισθητικά να οδηγεί σε μείωση της ακρίβειας του, αυτό δεν συμβαίνει στην πραγματικότητα.

Αυτό δικαιολογείται από το γεγονός ότι ο αλγόριθμος δεν χρησιμοποιεί αποκλειστικά τα πιθανολογικά μοντέλα κατά τη διαδικασία της πρόγνωσης. Αυτά χρησιμοποιούνται ταυτόχρονα με τα ντετερμινιστικά μοντέλα που έχουν να κάνουν με το πρόσφατο παρελθόν του συστήματος, δηλαδή την ίδια τη δομή του ψηφιακού δένδρου – trie που ανανεώνεται συνεχώς όσο αυξάνονται τα δεδομένα εισόδου.

Η τάση του συστήματος όπως χαρακτηριστικά αναπαρίσταται στα ακόλουθα διαγράμματα με υψηλή ακρίβεια είναι η ποιότητα της ακρίβειας πρόβλεψης από ένα σημείο και μετά να παραμένει σταθερή ($R^2 = 0,9716$), ενώ ο χρόνος ολοκλήρωσης των όποιων υπολογισμών να αυξάνεται όσο αυξάνει η τιμή της σταθεράς trieDepthFactor ($R^2 = 0,9949$).

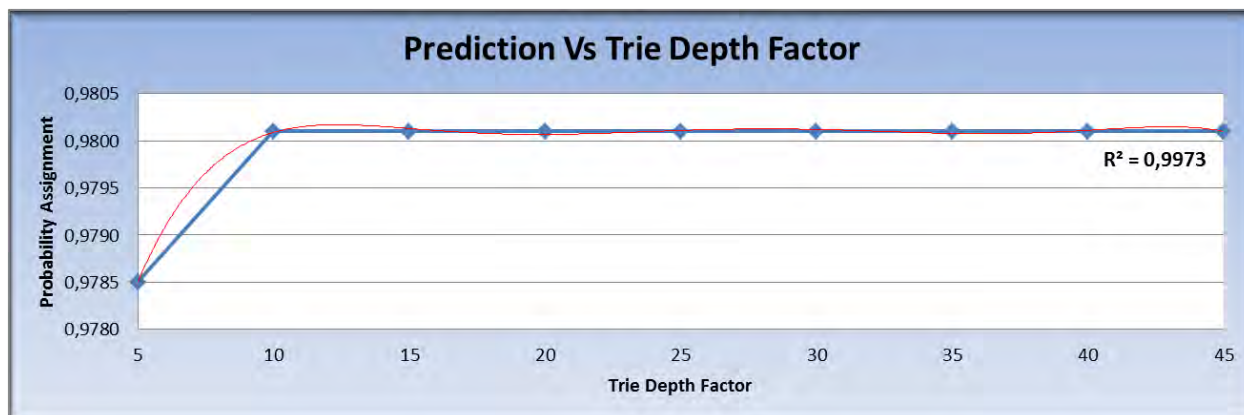


Εικόνα 3.33. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε σύγκριση με την παράμετρο TrieDepthFactor.

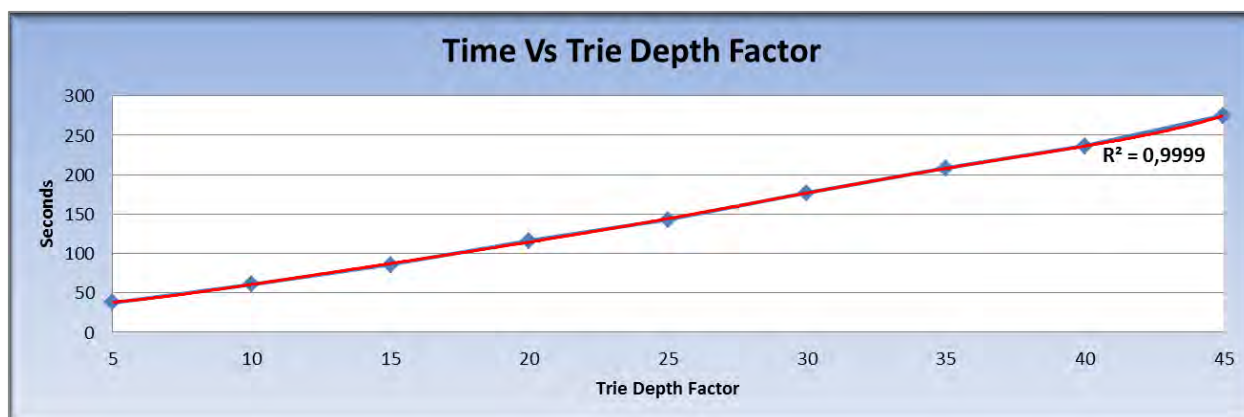


Εικόνα 3.34. Χρόνος ολοκλήρωσης εκτέλεσης του Heuristic ALZ σε σύγκριση με την παράμετρο TrieDepthFactor.

Την ίδια συμπεριφορά παρουσιάζει ο αλγόριθμος και για μεγαλύτερες τιμές της σταθεράς *expediteFactor*. Όπως φαίνεται χαρακτηριστικά στα ακόλουθα δύο διαγράμματα ενώ η ποιότητα της πρόβλεψης παραμένει ουσιαστικά αμετάβλητη από τη στιγμή που σταθεροποιήθηκε στην ανώτερη τιμή της (0,9801) ο χρόνος ολοκλήρωσης των όποιων υπολογισμών του συστήματος εξακολουθεί να αυξάνεται όσο αυξάνει η τιμή της σταθεράς *trieDepthFactor*.



Εικόνα 3.35. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε σύγκριση με μεγαλύτερες τιμές της παραμέτρου *trieDepthFactor*.



Εικόνα 3.36. Χρόνος ολοκλήρωσης εκτέλεσης του Heuristic ALZ σε σύγκριση με μεγαλύτερες τιμές της παραμέτρου *trieDepthFactor*.

Η σταθερά *trieDepthFactor* είναι σε άμεση εξάρτηση με την *expediteFactor* για την εξαγωγή ασφαλών συμπερασμάτων και αποτελεσμάτων ακριβείας. Η προ-τοποθετημένη τιμή *trieDepthFactor* = 8 αποτελεί ένα συμβιβασμό για την αύξηση της απόδοσης και την ανεπαίσθητη μείωση της ακρίβειας.

Ενώ η βέλτιστη επίδοση πρόβλεψης επιτυγχάνεται όταν ο παράγοντας *trieDepthFactor* = 9, η αύξηση που συνεπάγεται στο χρόνο εκτέλεσης του αλγορίθμου η τιμή αυτή, είναι δυσανάλογα μεγαλύτερη σε ότι αφορά το κέρδος στην ακρίβεια πρόβλεψης του αλγορίθμου και για το λόγο αυτό επιλέγεται η τιμή *trieDepthFactor* = 8 ως προ-τοποθετημένη τιμή λειτουργίας για αυτόν. Η επιπτώσεις της επιλογής αυτής είναι θεαματικότερες, σε ότι

αφορά τη μείωση του χρόνου εκτέλεσης του αλγορίθμου, όσο το πλήθος των συμβόλων και το μέγεθος της συμβολοσειράς εισόδου αυξάνονται.

Η αυξημένη τιμή της παραμέτρου `trieDepthFactor` = 8 έχει το μειονέκτημα ότι στις περιπτώσεις όπου τα δεδομένα εισόδου δεν ακολουθούν κάποιο συγκεκριμένο Pattern (υπάρχει δηλαδή υψηλή εντροπία) ο αλγόριθμος αναγκάζεται να ανανεώνει συνεχώς το trie με νέες συμβολοσειρές οι οποίες θα χρειαστεί αργότερα να ληφθούν υπόψη κατά τον υπολογισμό των πιθανοτήτων εμφάνισης του εκάστοτε συμβόλου.

Στην χειρότερη των περιπτώσεων μπορεί να απαιτηθούν μέχρι και $\sum_{n=1}^8 (128^n)$ επιμέρους υπολογισμοί πιθανοτήτων. Η πολυπλοκότητα αυτή των υπολογισμών έχει να κάνει με την τιμή της ίδιας της μεταβλητής `trieDepthFactor`, έχει να κάνει όμως και το μέγεθος του αλφάβητου του συστήματος (128 σύμβολα).

Για την αντιμετώπιση των περιπτώσεων αυτών, η καλύτερη λύση προκειμένου την επιτάχυνση της διαδικασίας πρόβλεψης, είναι η μείωση της τιμής της σταθεράς `trieDepthFactor`, γεγονός που συνεπάγεται όμως ελάττωση στην ακρίβεια του αλγορίθμου και μείωση του ρυθμού σύγκλισης αυτού (μικρού όμως συγκριτικά με τη μείωση του χρόνου εκτέλεσης του αλγορίθμου).

3.4.6 Η σταθερά **heuristicFactor** λειτουργεί ως παράγοντας βελτιστοποίησης στον αλγόριθμο και η τιμή της, όπως έχουμε ήδη επισημάνει σε προηγούμενο κεφάλαιο όπου αναλύθηκε η χρησιμότητά της, έχει καθοριστεί σε 0.167.

Ποια η χρησιμότητα της σταθεράς heuristicFactor;

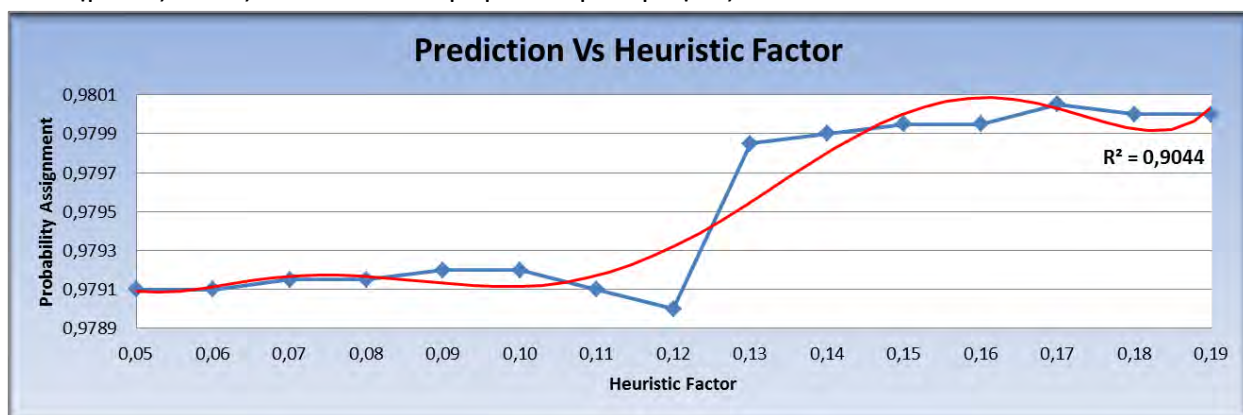
Τονίσαμε ήδη ότι κατά τη διαδικασία του prediction αν υπάρχει ιστορία έπειτα από το active πλαίσιο εισόδου, τότε η πρόβλεψη γίνεται με βάση το σύμβολο που έχει τη μεγαλύτερη συχνότητα εμφάνισης. Τι γίνεται όμως αν υπάρχει και κάποιο άλλο σύμβολο η συχνότητα εμφάνισης του οποίου είναι κοντά στη μέγιστη συχνότητα εμφάνισης που έχει βρει το σύστημα έως εκείνη τη χρονική στιγμή;

Στις περιπτώσεις αυτές κρίθηκε σκόπιμο να γίνει ένας περαιτέρω έλεγχος με βάση τα πιθανολογικά μοντέλα για την εμφάνιση του κάθε συμβόλου, έτσι ώστε να γίνει μία καλύτερη πρόβλεψη για το επόμενο σύμβολο. Εδώ υπεισέρχεται και η χρησιμότητα της σταθεράς `heuristicFactor`.

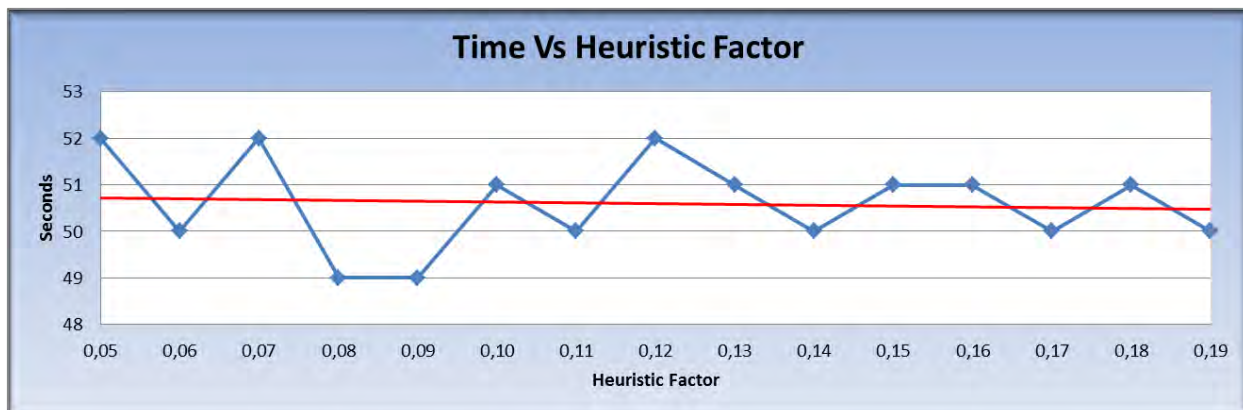
Πιο συγκεκριμένα, η σταθερά αυτή συνιστά για δύο σύμβολα του αλφάβητου του προγνώστη, μέτρο σύγκρισης για το ποσοστό της διαφοράς στη συχνότητα εμφάνισης μεταξύ των δύο συμβόλων προς τη συχνότητα εμφάνισης του συμβόλου από τα οποία προέρχονται τα σύμβολα αυτά (που βρίσκεται ένα επίπεδο πιο πάνω στο trie).

Σε περίπτωση που το ποσοστό είναι μικρότερο από την τιμή του heuristicFactor, τότε προτείνεται το σύμβολο με τη μεγαλύτερη πιθανότητα εμφάνισης, που μπορεί τελικά να διαθέτει και την μικρότερη συχνότητα εμφάνισης σε εκείνο το επίπεδο του trie. Η διαδικασία αυτή συνεχίζεται μέχρι να γίνει έλεγχος όλου αλφάβητου (128 σύμβολα) του συστήματος.

Η μελέτη των δύο επόμενων διαγραμμάτων δικαιολογεί την τιμή που ανατέθηκε στη σταθερά αυτή. Ο αλγόριθμος στην εικόνα **3.37** πραγματοποιεί ένα άλμα στην επίδοσή του σε τιμές HeuristicFactor > 0.12, με το αποκορύφωμα της επίδοσης να επιτυγχάνεται σε τιμή HeuristicFactor = 0.17. Η τάση του συστήματος όπως πολύ αξιόπιστα αναπαρίσταται ($R^2 = 0,9044$) είναι να διατηρήσει τα υψηλά ποσοστά ακρίβειας πρόβλεψης όσο αυξάνει η τιμή της σταθεράς heuristicFactor, ενώ για το χρόνο ολοκλήρωσης των υπολογισμών του συστήματος αυτός ουσιαστικά παραμένει αμετάβλητος.



Εικόνα 3.37. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε σύγκριση με την παράμετρο HeuristicFactor.

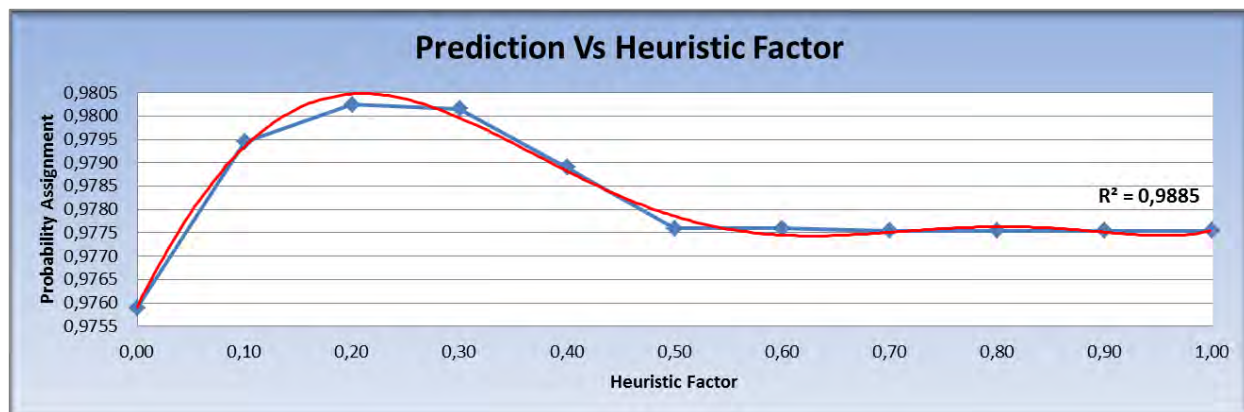


Εικόνα 3.38. Χρόνος ολοκλήρωσης εκτέλεσης του Heuristic ALZ σε σύγκριση με την παράμετρο HeuristicFactor.

Το σύστημα όντως λοιπόν όπως αποδεικνύεται πειραματικά πιο κάτω, διατηρεί την υψηλή αξιοπιστία του και για μεγαλύτερες τιμές της σταθεράς expediteFactor, όχι όμως και για όλο το εύρος των τιμών που μπορεί να πάρει αυτή.

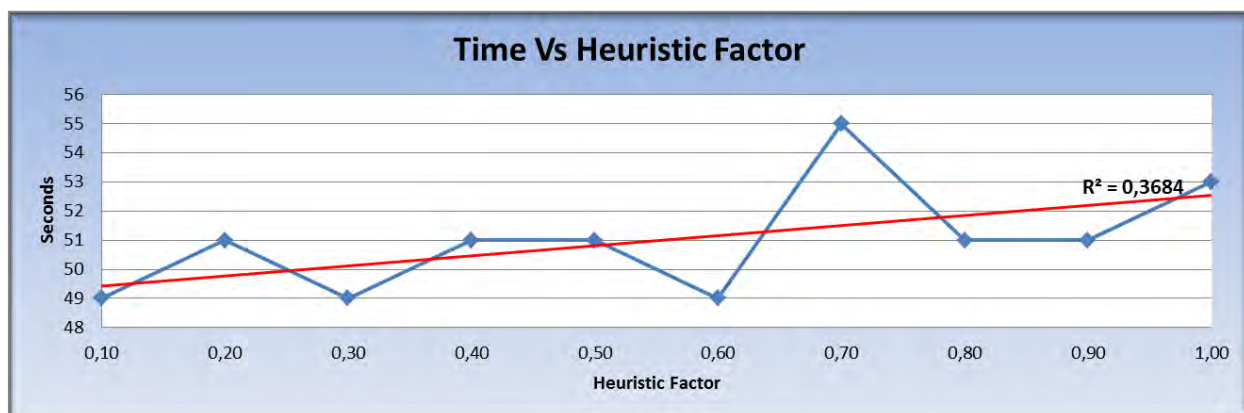
Υψηλή τιμή στη σταθερά heuristicFactor συνεπάγεται ότι το σύστημα στηρίζεται σε μεγαλύτερο ποσοστό στα πιθανολογικά μοντέλα που αυτό εξετάζει και λιγότερο στα ντετερμινιστικά μοντέλα. Μία τέτοια ρύθμιση του συστήματος θα εξαλείψει το σημαντικό πλεονέκτημα του αλγορίθμου της χρήσης των ντετερμινιστικών μοντέλων που αφορούν την ιστορία του συστήματος, γεγονός που διαισθητικά μπορεί να οδηγήσει σε πτώση της επίδοσης του συστήματος.

Στα ακόλουθα δύο διαγράμματα απεικονίζεται η επίδοση του αλγορίθμου για μεγαλύτερες τιμές της σταθεράς heuristicFactor μέχρι και την πλήρη εξάλειψη των ντετερμινιστικών χαρακτηριστικών του (heuristicFactor = 1). Φαίνεται χαρακτηριστικά ότι ο αλγόριθμος διατηρεί τα υψηλά ποσοστά επίδοσης μέχρι την τιμή για τη σταθερά heuristicFactor = 0,3 ενώ στη συνέχεια ακολουθεί μία πτώση της ποιότητας πρόβλεψης μέχρι τιμή heuristicFactor = 0,5. Από το σημείο εκείνο και μετά ουσιαστικά ο αλγόριθμος έχει χάσει τα ντετερμινιστικά χαρακτηριστικά του και συμπεριφέρεται με τον ίδιο τρόπο μέχρι την τιμή heuristicFactor = 1.



Εικόνα 3.39. Διακύμανση της ποιότητας της πρόβλεψης του Heuristic ALZ σε σύγκριση με μεγαλύτερες τιμές της παράμετρου HeuristicFactor.

Σε ότι αφορά δε το χρόνο ολοκλήρωσης των υπολογισμών, αυτός δείχνει την τάση να εμφανίσει μία ελαφρά αύξηση με τη συνεχή αύξηση της τιμής της σταθεράς heuristicFactor.



Εικόνα 3.40. Χρόνος ολοκλήρωσης εκτέλεσης του Heuristic ALZ σε σύγκριση με τις μεγαλύτερες τιμές της παραμέτρου HeuristicFactor.

3.4.7 Η σταθερά **zeroLevelContexts** καθορίζει αν θα χρησιμοποιηθούν όλα τα επίπεδα του trie ή μόνο το επίπεδο μηδέν αυτού, για τον καθορισμό των φράσεων που θα συμμετάσχουν στον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου, που μετέχει στο αλφάβητο του συστήματος.

4

Πειράματα

A/A	Περιγραφή	Σελ.
4.1	Τα δεδομένα εισόδου	52
4.2	Σύγκριση επίδοσης Heuristic ALZ	53
4.2.1	Ψευδοτυχαία δεδομένα εισόδου	56
4.2.2	Ψευδοτυχαία δεδομένα εισόδου με προσθήκη θορύβου	58
4.2.3	Πραγματικά δεδομένα εισόδου	60

4.1 Τα δεδομένα εισόδου

Για την αξιολόγηση της επίδοσης του αλγορίθμου Heuristic ALZ, χρησιμοποιήθηκαν τρία διαφορετικά set δεδομένων:

α. Το set ψευδοτυχαίων δεδομένων “data”, στο οποίο έχουμε ήδη αναφερθεί, με τη διαφοροποίηση ότι το σύνολο των χαρακτήρων που αυτό περιλαμβάνει έχει αυξηθεί στους 34000. Υπενθυμίζουμε ότι, το αλφάβητο του συστήματος σε αυτό το set δεδομένων, περιορίζεται σε τέσσερις χαρακτήρες (4) (“a”, “b”, “c”, “d”) και η συμβολοσειρά που επαναλαμβάνεται, μέχρις ότου συμπληρωθεί το απαραίτητο μέγεθος του set, αριθμεί είκοσι τρεις (23) χαρακτήρες.

Διαισθητικά αναμένουμε η επίδοση των αλγορίθμων με αυτό το set δεδομένων να είναι η καλύτερη δυνατή, διότι τεχνητά και εσκεμμένα η εντροπία του συστήματος παραμένει χαμηλή, αφού η συμβολοσειρά των είκοσι τριών χαρακτήρων επαναλαμβάνεται κάθε φορά αναλλοίωτη.

β. Το set ψευδοτυχαίων δεδομένων “data80”, που αποτελεί εικόνα του προηγούμενου set με τη διαφορά της προσθήκης θορύβου στο 20% των δεδομένων αυτού.

Το σκεπτικό πίσω από τη χρήση αυτού του set δεδομένων, είναι να αξιολογήσουμε τη συμπεριφορά των αλγορίθμων σε περιβάλλον αυξημένης εντροπίας. Με την προσθήκη θορύβου στα δεδομένα του set, οι επιτρεπτές διαδρομές (μόνο αυτές γιατί το αλφάβητο του συστήματος παραμένει σταθερά στους τέσσερις χαρακτήρες που αναφερθήκαμε), δεν εξασφαλίζεται, γεγονός που αναμένεται να ρίξει την αξιοπιστία του συστήματος.

γ. Το τελευταίο set δεδομένων “a_1” είναι μέρος ενός συνόλου αρχείων με το όνομα “proteins” και αναφέρεται στη δομή των πρωτεϊνών όπως αυτές αναπαρίστανται με τη χρήση του συνόλου των κεφαλαίων αγγλικών χαρακτήρων.

Τα set αυτού του είδους, χρησιμοποιούνται από τους αλγορίθμους τμηματοποίησης όπως ο Heuristic ALZ, για την κατηγοριοποίηση των πρωτεϊνών και όχι για έλεγχο της επίδοσης των αλγορίθμων πρόβλεψης. Το πλεονέκτημα όμως από τη χρήση του set αυτού, έγκειται στον έλεγχο της συμπεριφοράς του αλγορίθμου όταν η εντροπία του συστήματος είναι μέγιστη.

Διαισθητικά αναμένουμε ότι, η επίδοση των αλγορίθμων θα είναι η χειρότερη δυνατή κατά την αξιολόγησή τους με αυτό το set δεδομένων, όχι κυρίως σε ότι αφορά την

ακρίβεια της πρόβλεψης, αφού άλλωστε κάτι τέτοιο είναι αναμενόμενο λόγω της υψηλής εντροπίας του συστήματος, αλλά από πλευράς χρόνου ολοκλήρωσης των υπολογισμών.

4.2 Σύγκριση επίδοσης Heuristic ALZ

Στη συνέχεια ακολουθεί η αξιολόγηση της επίδοσης του αλγορίθμου Heuristic ALZ σε ότι αφορά την ακρίβεια της πρόβλεψης και το χρόνο ολοκλήρωσης των υπολογισμών του συστήματος. Τα αποτελέσματα αντιπαραβάλλονται με τα αντίστοιχα των αλγορίθμων Active LeZi και LZ78.

Όπως έχουμε τονίσει, το μεγάλο πλεονέκτημα του αλγορίθμου Heuristic ALZ είναι ότι η ικανότητα πρόγνωσης του, στηρίζεται σε ένα συνδυασμό πιθανολογικών και ντετερμινιστικών μεθόδων. Η μοναδικότητά του αυτή, του επιτρέπει να διατηρεί την ακρίβεια της πρόγνωσης σε υψηλά επίπεδα ενώ στηρίζεται για τον υπολογισμό των πιθανοτήτων εμφάνισης των συμβόλων που αποτελούν το αλφάβητο του συστήματος, σε ένα ποσοστό των αποθηκευμένων συμβολοσειρών του σχηματιζόμενου trie. Για παράδειγμα, με την ολοκλήρωση τμηματοποίησης της συμβολοσειράς εισόδου “**aaababbbbbaabccddcbaaaa**”, στους υπολογισμούς των πιθανοτήτων εμφάνισης του κάθε συμβόλου, χρησιμοποιούνται μόνο οι φράσεις του επιπέδου μηδέν (0) του trie και όχι και των υπόλοιπων επιπέδων.

aaa (order 3)	aa (order 2)	a (order 1)		Λ (order 0)			
a aaa (1)	a aa (1)	a a (2)	bcc a (1)	a (2)	abb (1)	bbba (1)	ddcb (1)
A aaa (1)	aa aa (1)	aa a (1)	A a (2)	aa (2)	abcc (1)	bccd (1)	bc (1)
	bc aa (1)	aaa a (1)		aaa (1)	baaa (1)	cbaa (1)	abc (1)
	A aa (2)	abc a (1)		aaaa (1)	baab (1)	ccdd (1)	b (1)
		b a (1)		aabc (1)	bbaa (1)	cddc (2)	bb (1)
		bb a (1)		ab (1)	bbb (2)	dcba (1)	bcc (1)

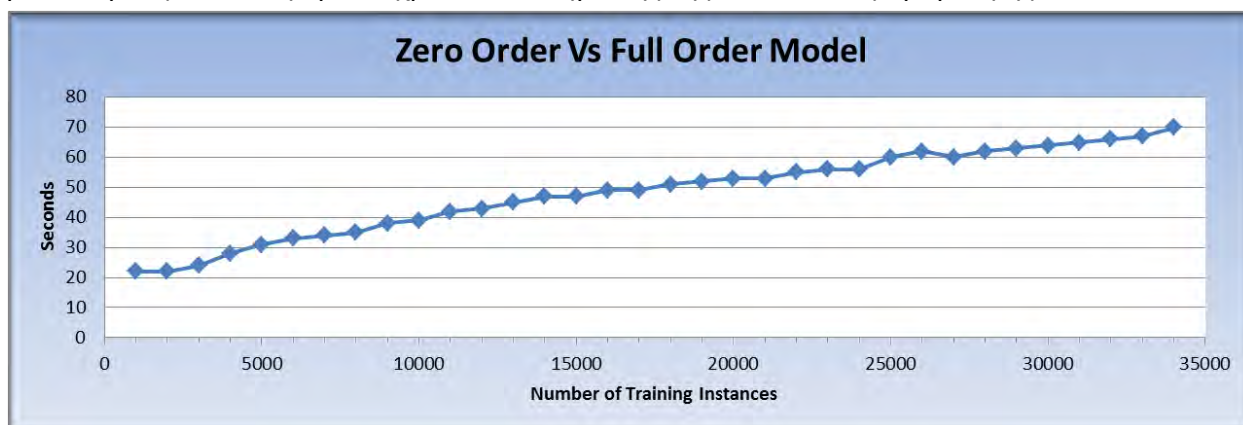
Εικόνα 4.1. Η πλήρης τμηματοποίηση της συμβολοσειράς “aaababbbbbaabccddcbaaaa” από τον αλγόριθμο Heuristic ALZ, παράγει ένα πλήθος συμβολοσειρών, από τα οποία όμως κατά τον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου του αλφαβήτου του συστήματος, χρησιμοποιούνται μόνο αυτές του επιπέδου μηδέν (0) .

Το κέρδος από τη χρήση μόνο των συμβολοσειρών του επιπέδου μηδέν (0) είναι τεράστιο σε ότι αφορά το χρόνο εκτέλεσης του αλγορίθμου. Για την ανακάλυψη όλων των έγκυρων συμβολοσειρών των άλλων επιπέδων, είναι απαραίτητο να γίνει αναζήτηση αυτών στο trie μέσω μίας αναδρομικής διαδικασίας , η οποία γίνεται πιο χρονοβόρα όσο αυξάνει το ύψος (όταν δηλαδή έχουμε πολλά δεδομένα), ή το πλάτος (όταν η εντροπία του συστήματος

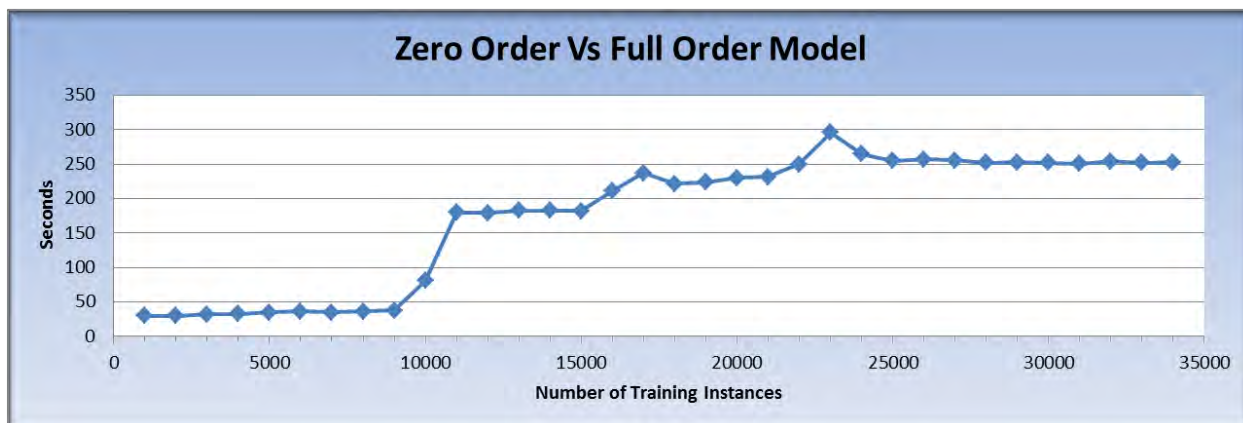
είναι μεγάλη) του trie. Σε ότι αφορά την επίδραση της χρήσης μόνο των συμβολοσειρών του επιπέδου μηδέν (0) στην ακρίβεια της πρόβλεψης αυτή είναι αμελητέα.

Στα παρακάτω διαγράμματα απεικονίζεται πως επηρεάζει την επίδοση του αλγορίθμου heuristic ALZ (για καθένα από τα χρησιμοποιούμενα set δεδομένων), η χρήση μόνο των συμβολοσειρών του μηδενικού επιπέδου του trie του συστήματος, αντί του συνόλου αυτών.

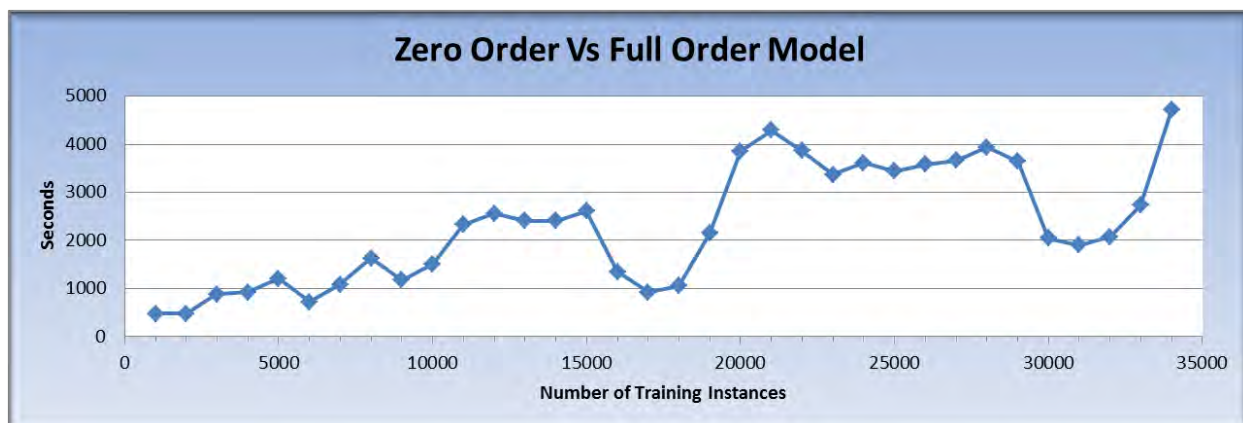
Στην περίπτωση που το διάγραμμα αναφέρεται στο χρόνο ολοκλήρωσης των υπολογισμών του συστήματος, οι θετικές τιμές δείχνουν το κέρδος σε δευτερόλεπτα για κάθε χίλιους χαρακτήρες που επεξεργάζεται το σύστημα. Σε όλες τις περιπτώσεις που εξετάζουμε , από την αποκλειστική χρήση των συμβολοσειρών του μηδενικού επιπέδου του trie, έχουμε μόνο κέρδος σε ότι αφορά το χρόνο ολοκλήρωσης της διαδικασίας πρόβλεψης.



Εικόνα 4.2. Κέρδος χρόνου σε δευτερόλεπτα για κάθε 1000 χαρακτήρες που επεξεργάζεται ο αλγόριθμος heuristic ALZ, όταν χρησιμοποιείται set δεδομένων "data".

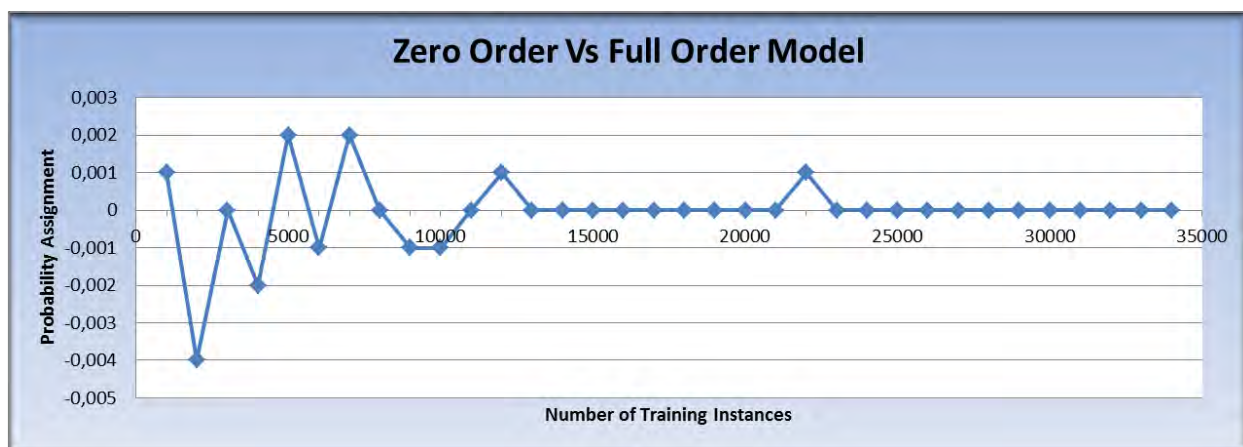


Εικόνα 4.3. Κέρδος χρόνου σε δευτερόλεπτα για κάθε 1000 χαρακτήρες που επεξεργάζεται ο αλγόριθμος heuristic ALZ, όταν χρησιμοποιείται set δεδομένων "data80".

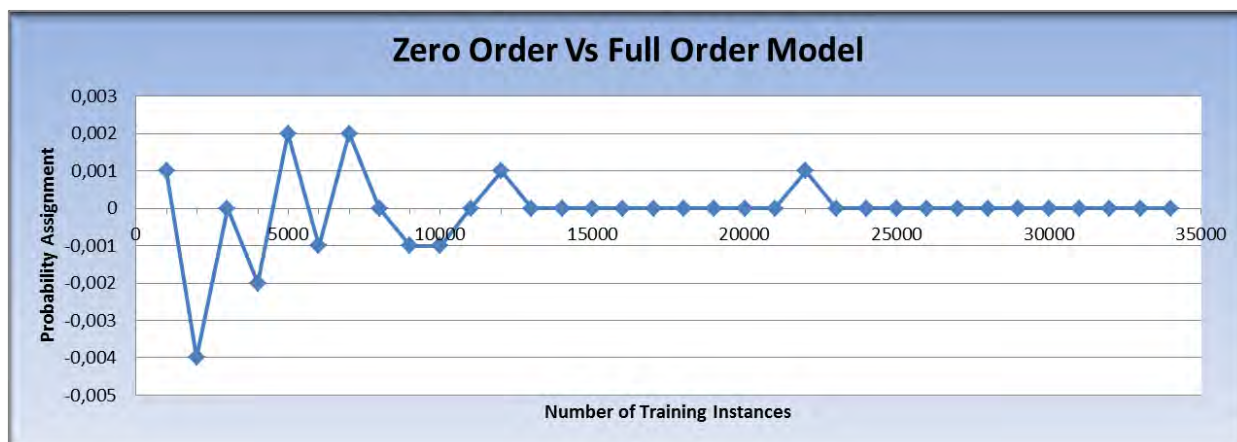


Εικόνα 4.4. Κέρδος χρόνου σε δευτερόλεπτα για κάθε 1000 χαρακτήρες που επεξεργάζεται ο αλγόριθμος heuristic ALZ, όταν χρησιμοποιείται set δεδομένων "a_1".

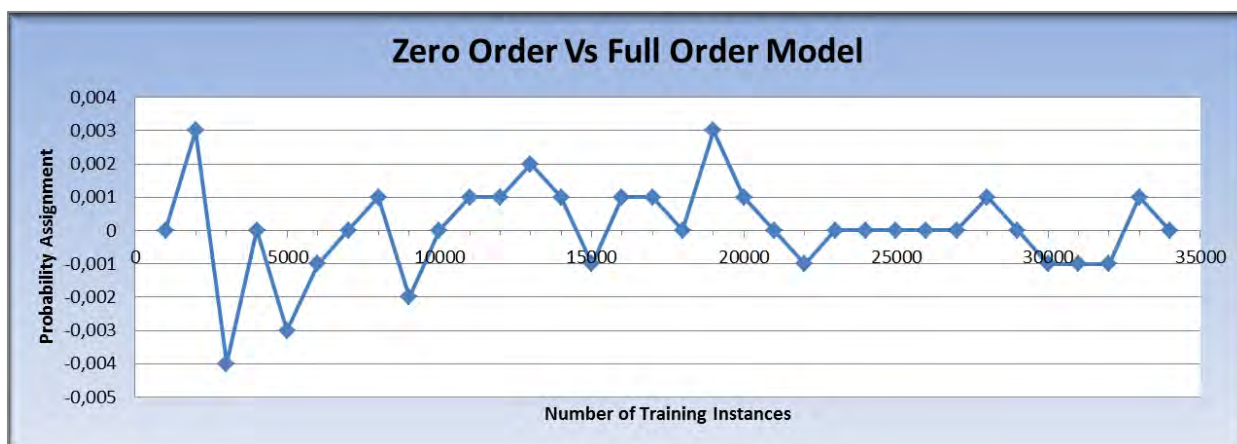
Στην περίπτωση της κατανομής των πιθανοτήτων εμφάνισης του εκάστοτε συμβόλου που μετέχει στο αλφάβητο του συστήματος, οι αρνητικές τιμές στον άξονα της κατανομής πιθανοτήτων δείχνουν ότι, το σύστημα απώλεσε κάποιους χαρακτήρες, τους οποίους θα είχε προβλέψει σωστά σε διαφορετική περίπτωση. Αντίθετα, οι θετικές τιμές δείχνουν ότι το σύστημα βγήκε κερδισμένο από τη χρησιμοποίηση μόνο των συμβολοσειρών του μηδενικού επιπέδου του trie. Τιμή πρόβλεψης για παράδειγμα, (-) 0,004, σημαίνει ότι το σύστημα απώλεσε τέσσερις χαρακτήρες κατά τους τελευταίους χίλιους χαρακτήρες που επεξεργάστηκε.



Εικόνα 4.5. Κέρδη (+) / Απώλειες (-) του συστήματος σε ότι αφορά την ακρίβεια της πρόβλεψης σε χαρακτήρες, για κάθε 1000 χαρακτήρες που επεξεργάζεται ο αλγόριθμος heuristic ALZ, όταν χρησιμοποιείται set δεδομένων "Data".



Εικόνα 4.6. Κέρδη (+) / Απώλειες (-) του συστήματος σε ότι αφορά την ακρίβεια της πρόβλεψης σε χαρακτήρες, για κάθε 1000 χαρακτήρες που επεξεργάζεται ο αλγόριθμος heuristic ALZ, όταν χρησιμοποιείται set δεδομένων "Data80".

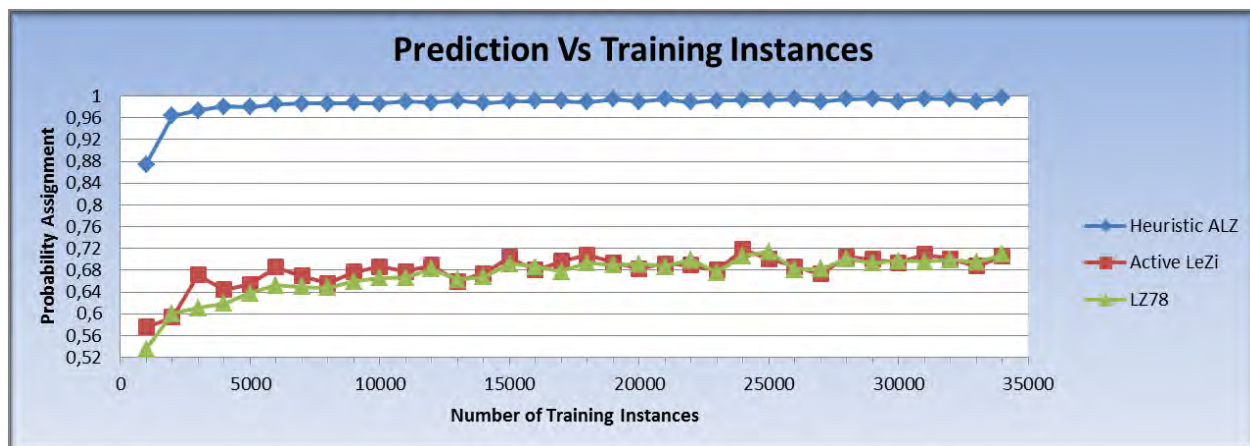


Εικόνα 4.7. Κέρδη (+) / Απώλειες (-) του συστήματος σε ότι αφορά την ακρίβεια της πρόβλεψης σε χαρακτήρες, για κάθε 1000 χαρακτήρες που επεξεργάζεται ο αλγόριθμος heuristic ALZ, όταν χρησιμοποιείται set δεδομένων "a_1".

4.2.1 Ποια η συμπεριφορά του αλγορίθμου Heuristic ALZ κατά τη χρήση του Dataset "data";

Η αρχική διαίσθηση τουλάχιστον σε ότι αφορά την υψηλή επίδοση του αλγορίθμου Heuristic ALZ για το set δεδομένων "data", επιβεβαιώνεται στο παρακάτω διάγραμμα. Η επίδοση μετά την είσοδο των πρώτων 4000 χαρακτήρων, εκτοξεύεται και διατηρείται σταθερά πάνω από 98%, με την τάση του συστήματος να φτάσει στο 100%, όσο αυξάνουν τα δεδομένα εισόδου.

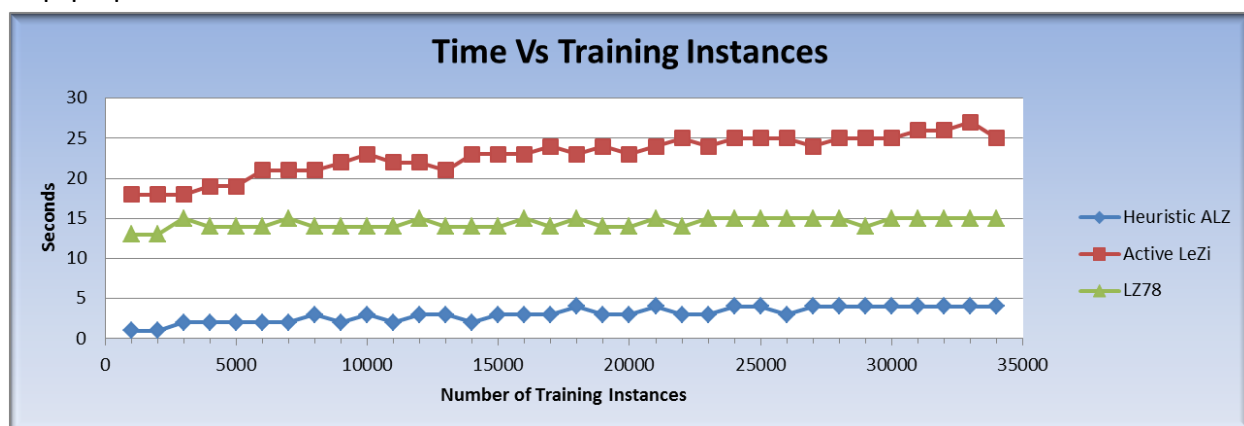
Για τους άλλους δύο αλγόριθμους παρατηρούμε σαφή διαφορά στην επίδοσή τους έναντι του Heuristic ALZ. Αυτή παραμένει και στους δύο στα επίπεδα του 70%.



Εικόνα 4.8. Διακύμανση της ποιότητας της πρόβλεψης των υπό εξέταση αλγορίθμων πρόγνωσης, κατά την τμηματοποίηση του set δεδομένων "data".

Μεγάλη διαφορά παρατηρείται επίσης και στο χρόνο ολοκλήρωσης της διαδικασίας πρόβλεψης για κάθε αλγόριθμο. Η τάση είναι ο κάθε αλγόριθμος να αυξάνει γραμμικά το χρόνο επεξεργασίας των δεδομένων εισόδου, όσο αυξάνει το μέγεθος του trie. Αυτό γίνεται πιο έντονα αισθητό στην περίπτωση του Active LeZi έναντι του LZ78 εξαιτίας του ότι το λεξικό του συστήματος σε αυτόν είναι χαρακτηριστικά μεγαλύτερο λόγω της ύπαρξης του ολισθαίνοντος παραθύρου.

Στην περίπτωση του heuristic ALZ, αν και το μέγεθος του trie είναι μεγαλύτερο, η χρησιμοποίηση κατά τον υπολογισμό των πιθανοτήτων εμφάνισης του κάθε συμβόλου, μόνο των συμβολοσειρών του πρώτου επιπέδου του σχηματιζόμενου trie, δίνει μεγάλο πλεονέκτημα έναντι των ανταγωνιστών του. Σε αυτόν, ενώ η ποιότητα πρόβλεψης παραμένει σε υψηλά ποσοστά, ο χρόνος για την πραγματοποίηση της διαδικασίας πρόβλεψης παραμένει αρκετές τάξεις μεγέθους μικρότερος, από την καλύτερη επίδοση εκ των δύο άλλων αλγορίθμων.



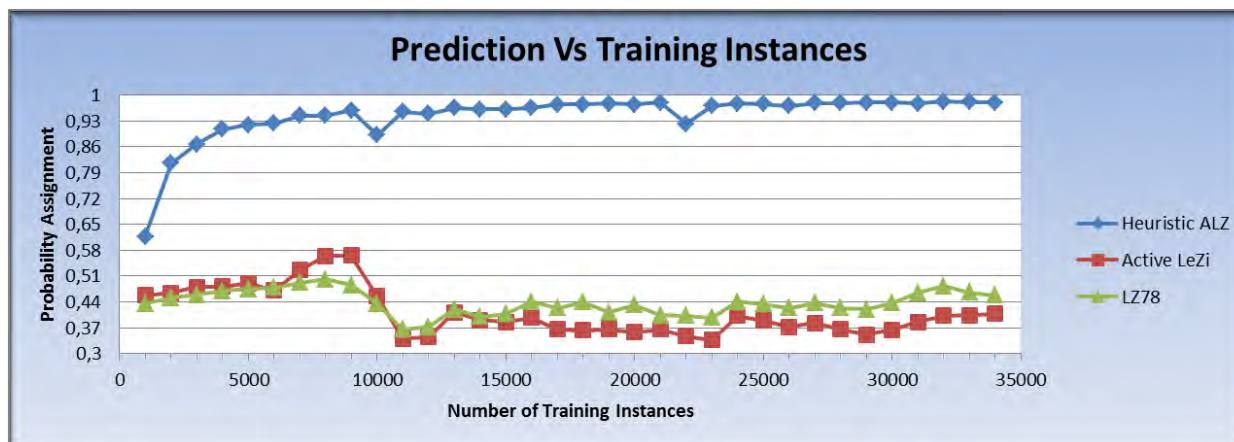
Εικόνα 4.9. Διακύμανση του χρόνου εκτέλεσης του Heuristic ALZ και των ανταγωνιστών του για το set δεδομένων "data".

4.2.2 Ποια η συμπεριφορά του αλγορίθμου Heuristic ALZ κατά τη χρήση του Dataset “data80”;

Με το set δεδομένων “data80” οι παρατηρήσεις της προηγούμενης παραγράφου ισχύουν αυτούσιες. Η επίδοση του αλγορίθμου Heuristic ALZ έναντι των ανταγωνιστών του εξακολουθεί να είναι τάξεις μεγέθους καλύτερη των άλλων. Η μόνη παρατήρηση αφορά την πτώση της ακρίβειας πρόβλεψης και την αύξηση του χρόνου ολοκλήρωσης των υπολογισμών κατανομής πιθανοτήτων του κάθε αλγορίθμου.

Η πτώση στην επίδοση όλων των αλγορίθμων, οφείλεται στην αυξημένη εντροπία του συστήματος, λόγω της εισαγωγής θορύβου στα δεδομένα εισόδου. Επειδή ο θόρυβος δεν ακολουθεί κάποιο συγκεκριμένο pattern, που πιθανόν θα μπορούσε να αναγνωριστεί από τους αλγόριθμους έτσι ώστε αυτοί να βελτιώσουν μακροπρόθεσμα τη συμπεριφορά τους, αξίζει να σχολιαστεί ξεχωριστά η φιλοσοφία αντιμετώπισης του προβλήματος της υψηλής εντροπίας από τον Heuristic ALZ σε σχέση με τους ανταγωνιστές του.

α. Ο αλγόριθμος Heuristic ALZ όσο το trie του συστήματος δεν διαθέτει αρκετά δεδομένα, στηρίζεται αρκετά στο πιθανολογικό μοντέλο που στηρίζονται και οι άλλοι δύο αλγόριθμοι. Μετά από τους πρώτους 4000 χαρακτήρες δεδομένων εισόδου, όπου όπως φαίνεται το ντετερμινιστικό μοντέλο του συστήματος τείνει να γίνει αρκετά αξιόπιστο, η επίδοση αυτού, σε ότι αφορά την ακρίβεια πρόβλεψης, παραμένει σταθερά πάνω από 90% και συνεχίζει να αυξάνεται γραμμικά μέχρι την επίδοση του 97%. Στη συνέχεια ο ρυθμός βελτίωσης πέφτει και φτάνει στο σημείο να είναι σχεδόν μηδενικός από τη στιγμή που φτάσει στο 98%.

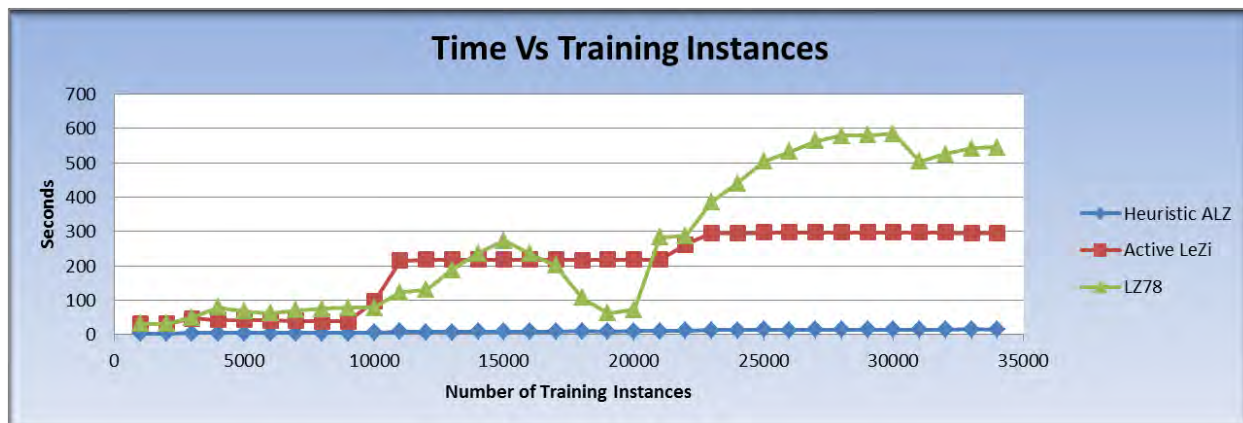


Εικόνα 4.10. Διακύμανση της ποιότητας της πρόβλεψης των υπό εξέταση αλγορίθμων πρόγνωσης, κατά την τμηματοποίηση του set δεδομένων “data80”.

Από το σημείο αυτό και πέρα η τυχαιότητα που χαρακτηρίζει τα δεδομένα θορύβου παίζει καθοριστικό ρόλο στην επίδοση του αλγορίθμου και δεν επιτρέπει τη

σύγκλιση του στο 100% σε ότι αφορά την ακρίβεια πρόβλεψης. Η μέση μείωση της ακρίβειας του συστήματος λόγω της εισόδου θορύβου στα δεδομένα εισόδου κυμαίνεται στο 5%.

Σε ότι αφορά το χρόνο ολοκλήρωσης των υπολογισμών κατανομής πιθανοτήτων στα σύμβολα του αλφαβήτου του αλγορίθμου, αυτός αυξάνεται γραμμικά στο πλήθος των δεδομένων εισόδου, παρατήρηση που δεν γίνεται εύκολα αντιληπτή από το διάγραμμα, λόγω της διαφοράς τάξης μεγέθους στο χρόνο ολοκλήρωσης των υπολογισμών αυτών σε σχέση με τους άλλους αλγόριθμους.



Εικόνα 4.11. Διακύμανση του χρόνου εκτέλεσης του Heuristic ALZ και των ανταγωνιστών του για το set δεδομένων "data80".

β. Για τον αλγόριθμο Active LeZi και LZ78 η είσοδος θορύβου στα δεδομένα εισόδου δρα καταλυτικά, σε ότι αφορά τη συμπεριφορά τους. Η πτώση της απόδοσή τους σε σχέση με το set δεδομένων "data80", κυμαίνεται περίπου στο 25% και αξίζει να σημειωθεί ότι ενώ η επίδοση του Active LeZi είναι αρχικά καλύτερη έναντι του LZ78, στη συνέχεια ο δεύτερος όχι μόνο αντισταθμίζει τη διαφορά στην ακρίβεια της πρόβλεψης αλλά τελικά επιτυγχάνει και καλύτερη επίδοση.

Εκεί που φαίνεται χαρακτηριστικά η επίδραση του θορύβου σε ότι αφορά τη συμπεριφορά των αλγορίθμων Active LeZi και LZ78 είναι στο πως επηρεάζει αυτός το χρόνο ολοκλήρωσης των υπολογισμών κατανομής πιθανοτήτων στα σύμβολα του αλφαβήτου αυτών. Η χρόνος αυτός είναι τάξεις μεγέθους μεγαλύτερος από τον αντίστοιχο στο set δεδομένων "data". Το πρόβλημα της ραγδαίας αύξησης του χρόνου ολοκλήρωσης των όποιων υπολογισμών των δύο αυτών αλγορίθμων καθιστά τη χρησιμοποίησή τους απαγορευτική στις περιπτώσεις όπου το trie του συστήματος είναι μεγάλο είτε λόγω ποσότητας δεδομένων, είτε λόγω αυξημένης εντροπίας.

4.2.3 Ποια η συμπεριφορά του αλγορίθμου Heuristic ALZ κατά τη χρήση του Dataset “a_1”;

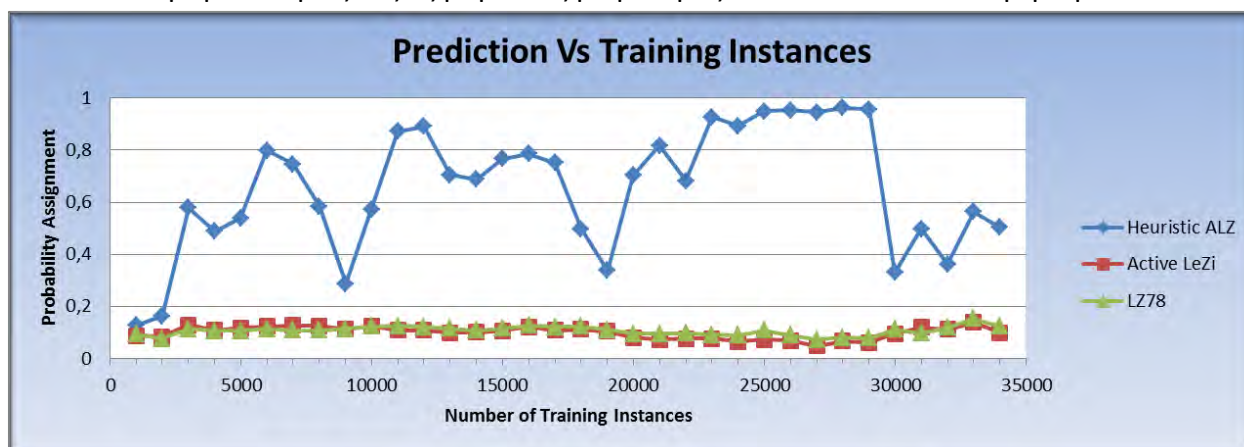
Η ιδιαίτερη φύση των δεδομένων του set “a_1” επηρεάζει για άλλη μία φορά καταλυτικά τη συμπεριφορά όλων των αλγορίθμων. Πιο συγκεκριμένα τα αποτελέσματα μπορούν να σχολιαστούν για κάθε περίπτωση ως εξής :

α. Ο αλγόριθμος Heuristic ALZ, κατορθώνει να αναγνωρίσει τα patterns που ακολουθούν τα δεδομένα εισόδου, δεν επιτυγχάνει όμως να διατηρήσει μία σταθερή επίδοση. Παρόλα αυτά δείχνει μία τάση να διατηρήσει τα υψηλά ποσοστά ακρίβειας πρόβλεψης για περισσότερο χρόνο όσο αυξάνουν τα δεδομένα εισόδου όπως επίσης και να βελτιώσει κάθε φορά τη χειρότερη τιμή πρόγνωσης.

Αν και διαφαίνεται μία τάση βελτίωσης της συμπεριφοράς του αλγορίθμου, με όσο το δυνατόν περισσότερα δεδομένα εισόδου, αυτή δεν μπορεί να χαρακτηριστεί αξιόπιστη για χρήση σε οποιοδήποτε σύστημα χρησιμοποιεί τέτοιου είδους δεδομένα εισόδου.

Για άλλη μία φορά θα πρέπει να τονιστεί ότι, η ακρίβεια της πρόβλεψης και η ευρωστία του αλγορίθμου Heuristic ALZ, διασφαλίζεται όταν υπάρχει κάποιο μοτίβο στην εναλλαγή των χαρακτήρων που εισέρχονται στο σύστημα πρόγνωσης. Αυτό θα πρέπει να ανακαλύψει ο αλγόριθμος για να διατηρήσει υψηλά ποσοστά πρόγνωσης. Στα προηγούμενα πειράματα αυτό διασφαλιζόταν με τον καθορισμό συγκεκριμένων διαδρομών που θα μπορούσαν να επιλεγούν μετά την είσοδο οποιουδήποτε χαρακτήρα στο σύστημα.

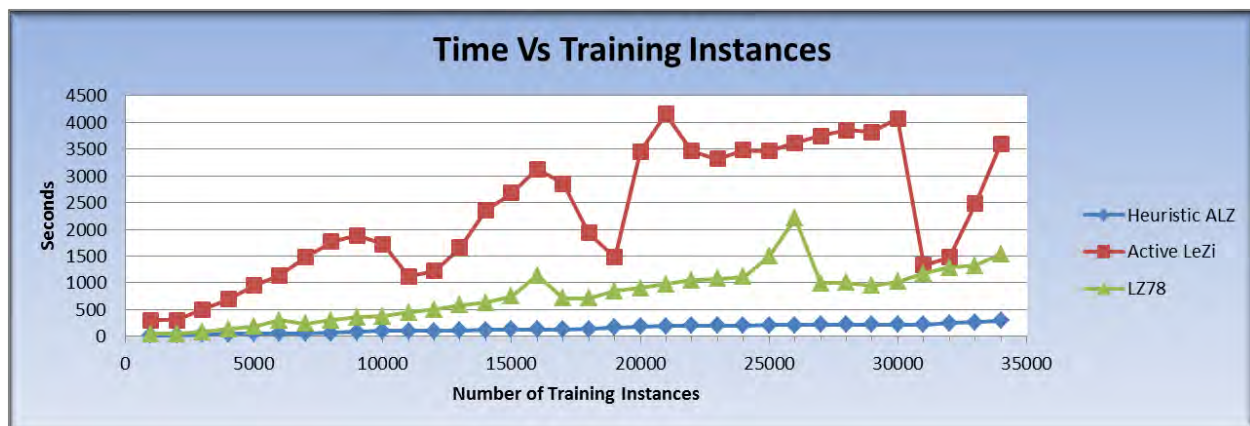
Σε ότι αφορά το χρόνο ολοκλήρωσης των υπολογισμών αυτός, είναι ο μεγαλύτερος από κάθε άλλη φορά και γραμμικά αυξανόμενος στο πλήθος των δεδομένων εισόδου. Παραμένει όμως τάξεις μεγέθους μικρότερος έναντι των άλλων αλγορίθμων.



Εικόνα 4.12. Διακύμανση της ποιότητας της πρόβλεψης των υπό εξέταση αλγορίθμων πρόγνωσης, κατά την τμηματοποίηση του set δεδομένων “a_1”.

β. Για τους αλγορίθμους Active LeZi και LZ78 η φύση των δεδομένων εισόδου δεν τους επιτρέπει να επιτύχουν μία μέση τιμή επίδοσης μεγαλύτερη από 10%. Για άλλη μία φορά ο αλγόριθμος LZ78 επιτυγχάνει καλύτερα αποτελέσματα όσο αυξάνουν τα δεδομένα εισόδου έναντι του Active LeZi.

Σε ότι αφορά το χρόνο ολοκλήρωσης των όποιων υπολογισμών αυτοί γίνονται απαγορευτικοί για χρήση των αλγορίθμων σε οποιοδήποτε σύστημα διαθέτει υψηλή εντροπία.



Εικόνα 4.13. Διακύμανση του χρόνου εκτέλεσης του Heuristic ALZ και των ανταγωνιστών του για το set δεδομένων "data".

5

Συμπεράσματα – Μελλοντικές Εργασίες

Ο Heuristic ALZ είναι ένας ακόμα αλγόριθμος πρόβλεψης της οικογένειας των προγνωστών που προέρχονται από τον αλγόριθμο συμπίεσης δεδομένων LZ78 και στηρίζονται σε μοντέλα Markov, για την κατασκευή και υπολογισμό των πιθανολογικών τους μοντέλων.

Η διαφοροποίησή του έναντι των άλλων αλγορίθμων, έγκειται στη δυνατότητά του για εμπλοκή στη διαδικασία πρόβλεψης, πλέον των πιθανολογικών μοντέλων και ντετερμινιστικών, που χρησιμοποιούν την ιστορία του συστήματος στο οποίο είναι εγκατεστημένος και λειτουργεί αυτός και η οποία είναι δεδομένη.

Η επίδοση του αλγορίθμου, σε ότι αφορά την ακρίβεια πρόβλεψης, είναι σε κάθε περίπτωση καλύτερη των ανταγωνιστών του και μπορεί να φτάσει και το 100%, όταν η εντροπία του συστήματος δύναται να περιοριστεί μέσα σε κάποια λογικά όρια.

Η εντροπία ενός συστήματος πρόγνωσης μπορεί να περιοριστεί, όσο το αλφάβητο² του συστήματος αυτού παραμένει περιορισμένο, όσο η αλληλουχία των χαρακτήρων που συμμετέχουν στο αλφάβητο αυτό ακολουθεί αυστηρά προσδιορισμένες διαδρομές, και όταν ο θόρυβος που εισέρχεται στα δεδομένα εισόδου παραμένει στα ελάχιστα δυνατά επίπεδα.

Το πλεονέκτημα του Heuristic ALZ είναι ότι ο ευρωστία του αυξάνει όσο αυξάνουν τα δεδομένα εισόδου. Η αρνητική επίπτωση από την αύξηση του όγκου των δεδομένων που αυτός έχει να επεξεργαστεί έχει να κάνει με την αύξηση του χρόνου ολοκλήρωσης των όποιων υπολογισμών είναι απαραίτητοι για την πραγματοποίηση της διαδικασίας πρόγνωσης, καθώς και την αύξηση της χρησιμοποιούμενης μνήμης για την αποθήκευση του trie του συστήματος που ολοένα αυξάνει.

Τα δύο αυτά προβλήματα θα μπορούσαν να ξεπεραστούν, με την περαιτέρω βελτίωση του αλγορίθμου, έτσι ώστε αυτός να απαλείφει εκείνα τα κλαδιά του trie που έχει καιρό να επισκεφτεί (γεγονός που σημαίνει ότι πιθανόν να αποτελεί αποτέλεσμα εισόδου θορύβου στα δεδομένα εισόδου). Αυτό βέβαια μπορεί να γίνει μόνο αφού ωριμάσει κατάλληλα το ντετερμινιστικό μοντέλο του συστήματος, το οποίο δεν είναι και εύκολο, αφού εξαρτάται άμεσα από τη φύση των δεδομένων εισόδου στο σύστημα και το χρόνο που απαιτείται να περάσει για να επαναληφθούν αυτά.

Επίσης, ο αλγόριθμος μπορεί να βελτιώσει ακόμα περισσότερο την επίδοσή του στον τομέα της ακρίβειας αν η πρόβλεψή του δεν αφορά μόνο μία κατάσταση ως επόμενη δυνατή, αλλά περισσότερες από μία, με την προσθήκη μίας παραμέτρου που θα καθορίζεται από το χρήστη. Αυτή η δυνατότητα θα μπορούσε να χρησιμοποιηθεί για παράδειγμα στην περίπτωση των συστημάτων κινητής τηλεφωνίας, όπου για την πραγματοποίηση της διαδικασίας handover δεν θα ήταν απαραίτητο να γίνεται δέσμευση συχνοτήτων σε όλους τους

² Με τον όρο αλφάβητο αναφερόμαστε στις διαφορετικές καταστάσεις που μπορεί να βρεθεί το σύστημα

γειτνιάζοντες σταθμούς βάσης (base stations), αλλά μόνο σε εκείνους που η ιστορία του συστήματος δείχνει ως πιο πιθανό να κινηθεί ο χρήστης.

Ο Heuristic ALZ έχει τη δυνατότητα να παραμετροποιείται πολύ εύκολα έτσι ώστε ανάλογα με τις επιθυμίες του χρήστη του να μπορεί να προσφέρει κάθε φορά υψηλή ποιότητα πρόγνωσης άσχετα του χρόνου που απαιτείται για την πραγματοποίηση των υπολογισμών του συστήματος ή αποδεκτά ποσοστά ακρίβειας πρόβλεψης με το πλεονέκτημα του μικρού χρόνου για την επεξεργασία των δεδομένων εισόδου.

Στο σημείο αυτό τελειώνει η παρουσίαση του Heuristic ALZ, ενός αλγορίθμου ευέλικτου και αποτελεσματικού, που φέρει σε πέρας αξιόπιστα τη δύσκολη αποστολή της πρόβλεψης των καταστάσεων που λαμβάνουν χώρα σε κάθε σύστημα που μπορεί να κωδικοποιήσει τις καταστάσεις αυτές στο ASCII αλφάβητο.

Αναφορές – Παραπομπές

-
- ¹ J. S. Vitter and P. Krishnan, "Optimal Prefetching via Data Compression," J. ACM, vol. 43, no. 5, 1996, pp.771–93.
- ² T. Liu, P. Bahl, and I. Chlamtac, "Mobility Modeling, Location Tracking, and Trajectory Prediction in Wireless ATM Networks," IEEE JSAC, vol. 16, no. 6, 1998, pp. 922–36.
- ³ A. Misra, A. Roy, and S. K. Das, "An Information-Theoretic Framework for Optimal Location Tracking in Multi-System 4G Wireless Networks," Proc. IEEE INFOCOM, vol. 1, 2004, pp. 286–97.
- ⁴ M. Kyriakakos et al., "Enhanced Path Prediction for Network Resource Management in Wireless LANs," IEEE Wireless Communication., vol. 10, no. 6, 2003, pp. 62–69.
- ⁵ D. A. Levine, I. F. Akyildiz, and M. Naghshineh, "A Resource Estimation and Call Admission Algorithm for Wireless Multimedia Networks using the Shadow Cluster Concept," IEEE/ACM Trans. Net., vol. 5, no. 1, 1997, pp. 1–12.
- ⁶ A. Bhattacharya and S. K. Das, "LeZi-Update: An Information- Theoretic Framework for Personal Mobility Tracking in PCS Networks," ACM/Kluwer Wireless Net., vol. 8, no. 2–3, 2002, pp. 121–35.
- ⁷ M. Deshpande and G. Karypis, "Selective Markov Models for Predicting Web Page Accesses," ACM Trans. Internet Tech., vol. 4, no. 2, 2004, pp. 163–84.
- ⁸ http://biophysics.biol.uoa.gr/courses/biophysics/prognwsh/ASKISI_1%5B2%5D.html
- ⁹ M. Weinberg and G. Seroussi, Sequential Prediction and Ranking in Universal Context Modeling and Data Compression, tech. report HPL-94-111, HP Labs, 1997.
- ¹⁰ J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable Rate Coding," IEEE Trans. Information Theory, vol. 24, no. 5, 1978, pp. 530–536.
- ¹¹ M. Feder, N. Merhav, and M. Gutman, "Universal Prediction of Individual Sequences," IEEE Trans. Information Theory, vol. 38, no. 4, 1992, pp. 1258–1270.
- ¹² P. Franaszek, J. Thomas, and P. Tsoucas, "Context Allocation with Application to Data Compression," IEEE Int'l Symp. Information Theory, IEEE Press, 1994, p. 12.
- ¹³ K. Gopalratnam and D. J. Cook, "Online Sequential Prediction via Incremental Parsing: The Active LeZi Algorithm," IEEE Intelligent Sys., vol. 22, no. 1, 2007, pp. 52–58.
- ¹⁴ J.G. Cleary and W.J. Teahan, "Unbounded Length Contexts for PPM," Computer J., vol. 40, nos. 2–3, 1997, pp. 7–75.
- ¹⁵ T.C. Bell, J.G. Cleary, and I.H. Witten, Text Compression, Prentice Hall, 1990.

Α Παράρτημα

1.	Η επίδραση της σταθεράς trieDepthFactor	A-4
➤	trieDepthFactor = 3, 4, 5	A-4
➤	trieDepthFactor = 6, 7, 8	A-5
➤	trieDepthFactor = 9, 10, 11	A-6
➤	trieDepthFactor = 5, 10, 15	A-7
➤	trieDepthFactor = 20, 25, 30	A-8
➤	trieDepthFactor = 35, 40, 45	A-9
2.	Η επίδραση της σταθεράς heuristicFactor	A-10
➤	heuristicFactor = 0.05, 0.06, 0.07	A-10
➤	heuristicFactor = 0.08, 0.09, 0.1	A-11
➤	heuristicFactor = 0.11, 0.12, 0.13	A-12
➤	heuristicFactor = 0.14, 0.15, 0.16	A-13
➤	heuristicFactor = 0.17, 0.18, 0.19	A-14
➤	heuristicFactor = 0.0, 0.1, 0.2	A-15
➤	heuristicFactor = 0.3, 0.4, 0.5	A-16
➤	heuristicFactor = 0.6, 0.7, 0.8	A-17
➤	heuristicFactor = 0.9, 1.0	A-18
3.	Η επίδραση της σταθεράς expediteFactor	A-19
➤	expediteFactor = 0, 1, 2	A-19
➤	expediteFactor = 3, 4, 5	A-20
➤	expediteFactor = 6, 7, 8	A-21
➤	expediteFactor = 9, 10	A-22
4.	Η επίδραση της σταθεράς alphabetSize / Char Sequence	A-23
➤	alphabetSize = 5 / Char Sequence = 20, 40, 60	A-23
➤	alphabetSize = 10 / Char Sequence = 30, 60, 90	A-24
➤	alphabetSize = 15 / Char Sequence = 45, 90, 135	A-25
➤	alphabetSize = 20 / Char Sequence = 60, 120, 180	A-26
➤	alphabetSize = 26 / Char Sequence = 75, 150, 225	A-27

5.	Σύγκριση επίδοσης αλγορίθμων Heuristic ALZ, Active LeZi, LZ78	A-28
➤	Set δεδομένων “data”	A-28
➤	Set δεδομένων “data80”	A-29
➤	Set δεδομένων “a_1”	A-30

Το παράρτημα αυτό, αφορά τα πειραματικά αποτελέσματα, πάνω στα οποία στηρίζονται όλα τα γραφήματα στα οποία αναφερθήκαμε, κατά την παρουσίαση του αλγορίθμου Heuristic ALZ.

Αναλυτικά, παρουσιάζεται πως επηρεάζεται η ακρίβεια της ικανότητας πρόβλεψης του αλγορίθμου, καθώς και ο χρόνος ολοκλήρωσης των όποιων υπολογισμών του συστήματος, για τις διάφορες τιμές που μπορούν να πάρουν οι παρακάτω παράμετροι λειτουργίας αυτού:

- α. trieDepthFactor
- β. heuristicFactor
- γ. expediteFactor

Παρουσιάζονται επίσης, τα αναλυτικά αποτελέσματα από την επίδραση που έχει στη συμπεριφορά του αλγορίθμου, το μέγεθος του αλφαβήτου του συστήματος καθώς και το μέγεθος της συμβολοσειράς εισόδου.

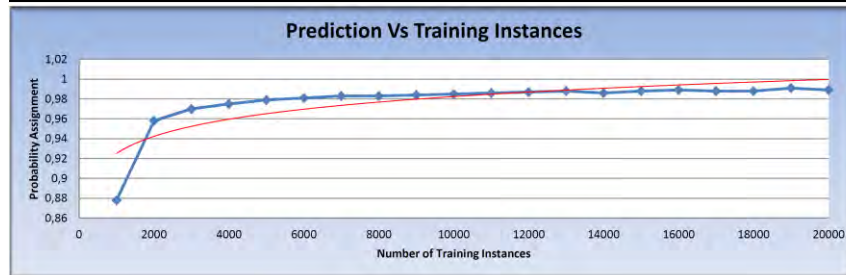
Τέλος, παρουσιάζονται και τα αναλυτικά αποτελέσματα της σύγκρισης απόδοσης του αλγορίθμου Heuristic ALZ με τους Active LeZi και LZ78.

Για την ανάλυση της τάσης του συστήματος κάθε φορά, χρησιμοποιείται

- α. η λογαριθμική γραμμή τάσης, για την κάλυψη των περιπτώσεων ελέγχου ακρίβειας πρόβλεψης,
- β. η γραμμική γραμμή τάσης, για την κάλυψη των περιπτώσεων ελέγχου του χρόνου εκτέλεσης του κάθε πειράματος.

Τις γραφικές παραστάσεις των αποτελεσμάτων, συνοδεύει ένας πίνακας, στον οποίο αναγράφονται οι παραδοχές κάτω από τις οποίες πραγματοποιήθηκαν τα πειράματα, καθώς και οι αριθμητικές τιμές των αποτελεσμάτων.

trieDepthFactor



alphabetSize	128
expediteFactor	2
trieDepthFactor	3
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.98900
mean Prediction	0.97780

Dataset	Prediction	Time
1000	0.87800	1
2000	0.95800	1
3000	0.97000	1
4000	0.97500	1
5000	0.97900	1
6000	0.98100	1
7000	0.98300	1
8000	0.98300	1
9000	0.98400	2
10000	0.98500	1
11000	0.98600	2
12000	0.98700	1
13000	0.98800	2
14000	0.98600	2
15000	0.98800	1
16000	0.98900	2
17000	0.98800	2
18000	0.98800	2
19000	0.99100	2
20000	0.98900	3
sum		30



alphabetSize	128
expediteFactor	2
trieDepthFactor	4
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.98900
mean Prediction	0.97855

Dataset	Prediction	Time
1000	0.87900	1
2000	0.96000	1
3000	0.97100	1
4000	0.97600	1
5000	0.98000	1
6000	0.98200	2
7000	0.98400	1
8000	0.98400	2
9000	0.98400	2
10000	0.98600	1
11000	0.98700	2
12000	0.98700	3
13000	0.98900	2
14000	0.98600	2
15000	0.98900	3
16000	0.98900	2
17000	0.98900	3
18000	0.98800	2
19000	0.99200	3
20000	0.98900	3
sum		38

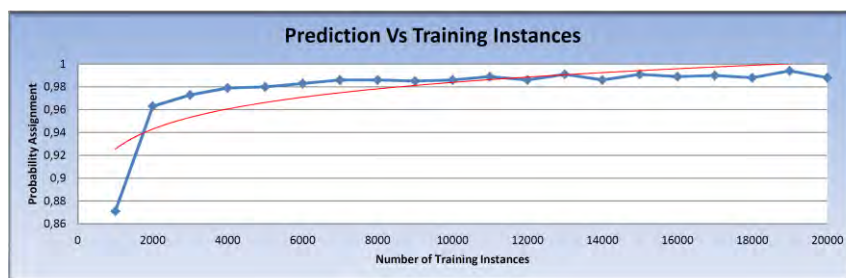


alphabetSize	128
expediteFactor	2
trieDepthFactor	5
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.98800
mean Prediction	0.97850

Dataset	Prediction	Time
1000	0.87100	1
2000	0.96100	1
3000	0.97200	1
4000	0.97700	1
5000	0.98000	1
6000	0.98200	2
7000	0.98500	2
8000	0.98500	1
9000	0.98400	2
10000	0.98600	2
11000	0.98800	2
12000	0.98600	2
13000	0.99000	2
14000	0.98600	3
15000	0.99000	2
16000	0.98800	2
17000	0.99000	3
18000	0.98800	2
19000	0.99300	3
20000	0.98800	3
sum		38



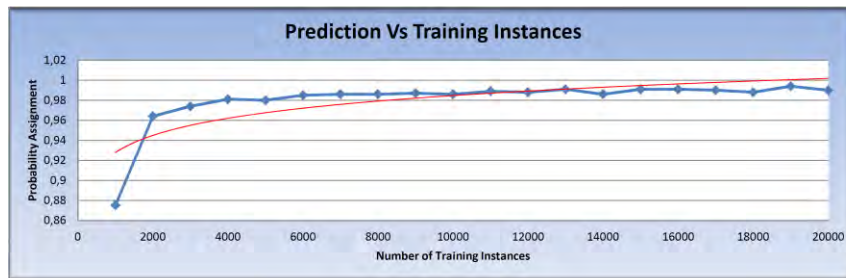
alphabetSize	128	
expediteFactor	2	
trieDepthFactor	6	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0,98800	
mean Prediction	0,97920	
Dataset	Prediction	Time
1000	0,87100	1
2000	0,96300	1
3000	0,97300	1
4000	0,97900	1
5000	0,98000	2
6000	0,98300	2
7000	0,98600	1
8000	0,98600	2
9000	0,98500	2
10000	0,98600	2
11000	0,98900	2
12000	0,98600	2
13000	0,99100	3
14000	0,98600	2
15000	0,99100	2
16000	0,98900	3
17000	0,99000	2
18000	0,98800	3
19000	0,99400	3
20000	0,98800	2
sum		39



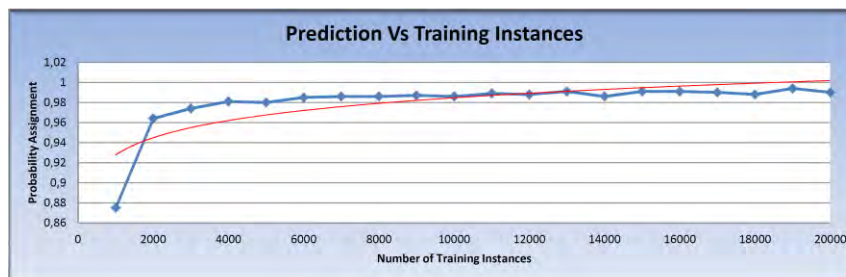
alphabetSize	128	
expediteFactor	2	
trieDepthFactor	7	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0,98800	
mean Prediction	0,97920	
Dataset	Prediction	Time
1000	0,87100	1
2000	0,96300	1
3000	0,97300	2
4000	0,97900	2
5000	0,98000	1
6000	0,98300	2
7000	0,98600	2
8000	0,98600	3
9000	0,98500	2
10000	0,98600	3
11000	0,98900	2
12000	0,98600	2
13000	0,99100	3
14000	0,98600	3
15000	0,99100	2
16000	0,98900	3
17000	0,99000	3
18000	0,98800	3
19000	0,99400	3
20000	0,98800	3
sum		46



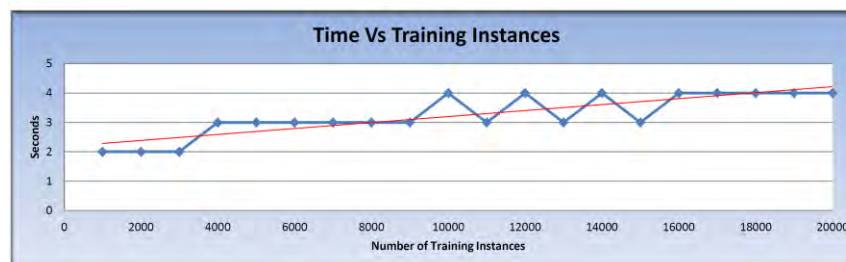
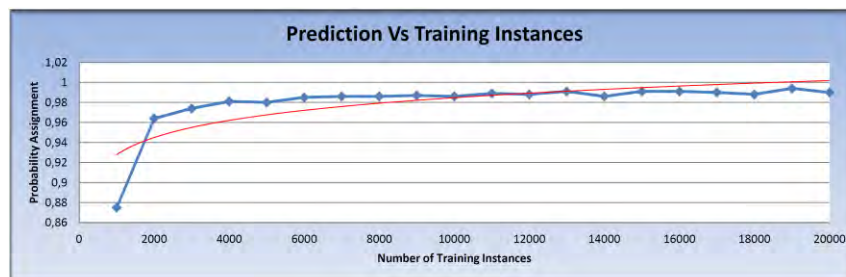
alphabetSize	128	
expediteFactor	2	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0,99000	
mean Prediction	0,98005	
Dataset	Prediction	Time
1000	0,87400	1
2000	0,96400	1
3000	0,97400	2
4000	0,98100	2
5000	0,98000	2
6000	0,98500	2
7000	0,98600	3
8000	0,98600	2
9000	0,98700	2
10000	0,98600	3
11000	0,98900	3
12000	0,98800	3
13000	0,99100	2
14000	0,98600	3
15000	0,99100	3
16000	0,99100	3
17000	0,99000	4
18000	0,98800	3
19000	0,99400	4
20000	0,99000	3
sum		51



alphabetSize	128	
expediteFactor	2	
trieDepthFactor	9	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.98010	
Dataset	Prediction	Time
1000	0.87500	1
2000	0.96400	1
3000	0.97400	2
4000	0.98100	3
5000	0.98000	2
6000	0.98500	3
7000	0.98600	2
8000	0.98600	3
9000	0.98700	2
10000	0.98600	3
11000	0.98900	3
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	4
16000	0.99100	3
17000	0.99000	3
18000	0.98800	4
19000	0.99400	3
20000	0.99000	4
sum		55



alphabetSize	128	
expediteFactor	2	
trieDepthFactor	10	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.98010	
Dataset	Prediction	Time
1000	0.87500	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	3
5000	0.98000	3
6000	0.98500	2
7000	0.98600	3
8000	0.98600	3
9000	0.98700	3
10000	0.98600	3
11000	0.98900	3
12000	0.98800	3
13000	0.99100	4
14000	0.98600	3
15000	0.99100	4
16000	0.99100	3
17000	0.99000	4
18000	0.98800	4
19000	0.99400	3
20000	0.99000	4
sum	61	



alphabetSize	128
expediteFactor	2
trieDepthFactor	11
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98010

Dataset	Prediction	Time
1000	0,87500	2
2000	0,96400	2
3000	0,97400	2
4000	0,98100	3
5000	0,98000	3
6000	0,98500	3
7000	0,98600	3
8000	0,98600	3
9000	0,98700	3
10000	0,98600	4
11000	0,98900	3
12000	0,98800	4
13000	0,99100	3
14000	0,98600	4
15000	0,99100	3
16000	0,99100	4
17000	0,99000	4
18000	0,98800	4
19000	0,99400	4
20000	0,99000	4
sum		65

trieDepthFactor 2



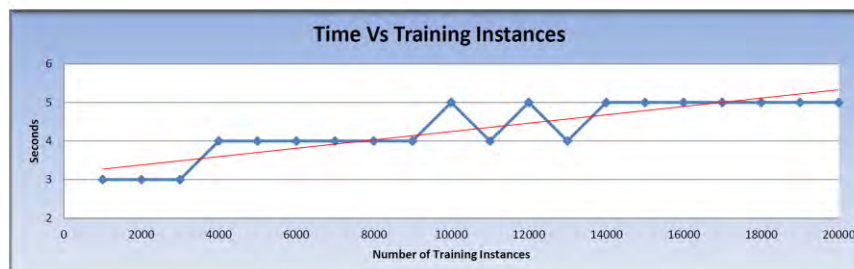
alphabetSize	128
expediteFactor	2
trieDepthFactor	5
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.98800
mean Prediction	0.97850

Dataset	Prediction	Time
1000	0.87100	1
2000	0.96100	1
3000	0.97200	1
4000	0.97700	1
5000	0.98000	1
6000	0.98200	2
7000	0.98500	2
8000	0.98500	1
9000	0.98400	2
10000	0.98600	2
11000	0.98800	2
12000	0.98600	2
13000	0.99000	2
14000	0.98600	3
15000	0.99000	2
16000	0.98800	2
17000	0.99000	3
18000	0.98800	2
19000	0.99300	3
20000	0.98800	3
sum		38



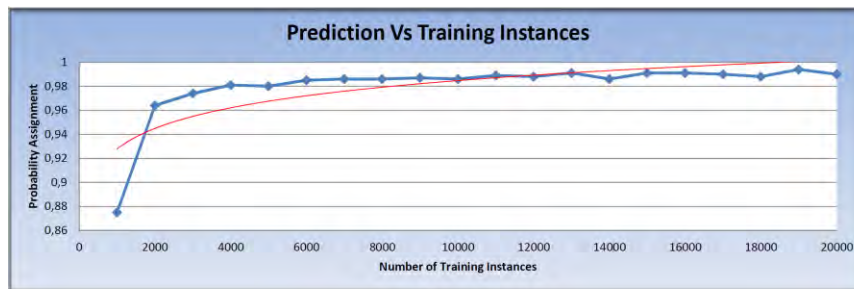
alphabetSize	128
expediteFactor	2
trieDepthFactor	10
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98010

Dataset	Prediction	Time
1000	0.87500	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	3
5000	0.98000	3
6000	0.98500	2
7000	0.98600	3
8000	0.98600	3
9000	0.98700	3
10000	0.98600	3
11000	0.98900	3
12000	0.98800	3
13000	0.99100	4
14000	0.98600	3
15000	0.99100	4
16000	0.99100	3
17000	0.99000	4
18000	0.98800	4
19000	0.99400	3
20000	0.99000	4
sum		61



alphabetSize	128
expediteFactor	2
trieDepthFactor	15
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98010

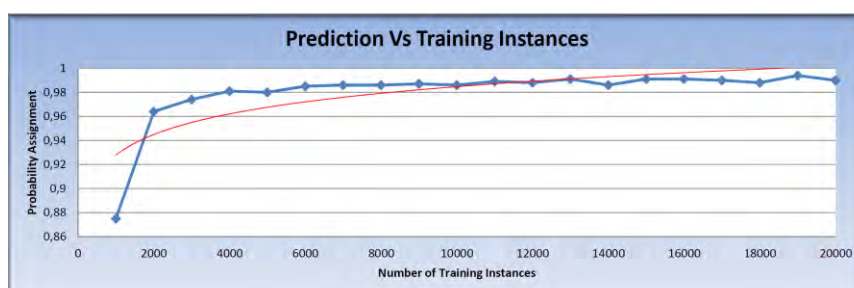
Dataset	Prediction	Time
1000	0.87500	3
2000	0.96400	3
3000	0.97400	3
4000	0.98100	4
5000	0.98000	4
6000	0.98500	4
7000	0.98600	4
8000	0.98600	4
9000	0.98700	4
10000	0.98600	5
11000	0.98900	4
12000	0.98800	5
13000	0.99100	4
14000	0.98600	5
15000	0.99100	5
16000	0.99100	5
17000	0.99000	5
18000	0.98800	5
19000	0.99400	5
20000	0.99000	5
sum		86



alphabetSize	128	
expediteFactor	2	
trieDepthFactor	20	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.98010	
Dataset	Prediction	Time
1000	0.87500	5
2000	0.96400	5
3000	0.97400	4
4000	0.98100	6
5000	0.98000	5
6000	0.98500	5
7000	0.98600	5
8000	0.98600	6
9000	0.98700	6
10000	0.98600	6
11000	0.98900	5
12000	0.98800	7
13000	0.99100	6
14000	0.98600	6
15000	0.99100	6
16000	0.99100	7
17000	0.99000	6
18000	0.98800	7
19000	0.99400	7
20000	0.99000	6
sum	116	



alphabetSize	128	
expediteFactor	2	
trieDepthFactor	25	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.98010	
Dataset	Prediction	Time
1000	0.87500	5
2000	0.96400	5
3000	0.97400	6
4000	0.98100	7
5000	0.98000	7
6000	0.98500	6
7000	0.98600	7
8000	0.98600	7
9000	0.98700	7
10000	0.98600	8
11000	0.98900	7
12000	0.98800	8
13000	0.99100	7
14000	0.98600	8
15000	0.99100	8
16000	0.99100	7
17000	0.99000	8
18000	0.98800	8
19000	0.99400	9
20000	0.99000	8
sum	143	



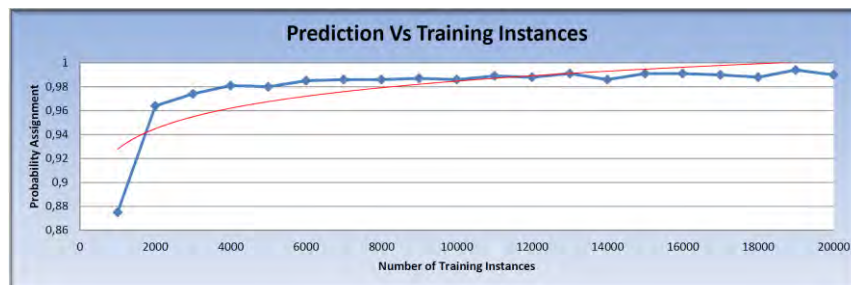
alphabetSize	128
expediteFactor	2
trieDepthFactor	30
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98010

Dataset	Prediction	Time
1000	0.87500	7
2000	0.96400	7
3000	0.97400	8
4000	0.98100	8
5000	0.98000	8
6000	0.98500	8
7000	0.98600	9
8000	0.98600	9
9000	0.98700	9
10000	0.98600	8
11000	0.98900	10
12000	0.98800	9
13000	0.99100	9
14000	0.98600	9
15000	0.99100	10
16000	0.99100	10
17000	0.99000	10
18000	0.98800	9
19000	0.99400	10
20000	0.99000	10
sum	177	



alphabetSize	128
expediteFactor	2
trieDepthFactor	35
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98010

Dataset	Prediction	Time
1000	0.87500	8
2000	0.96400	8
3000	0.97400	9
4000	0.98100	10
5000	0.98000	9
6000	0.98500	10
7000	0.98600	10
8000	0.98600	11
9000	0.98700	10
10000	0.98600	11
11000	0.98900	10
12000	0.98800	11
13000	0.99100	11
14000	0.98600	11
15000	0.99100	11
16000	0.99100	11
17000	0.99000	12
18000	0.98800	11
19000	0.99400	12
20000	0.99000	12
sum		208



alphabetSize	128
expediteFactor	2
trieDepthFactor	40
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98010

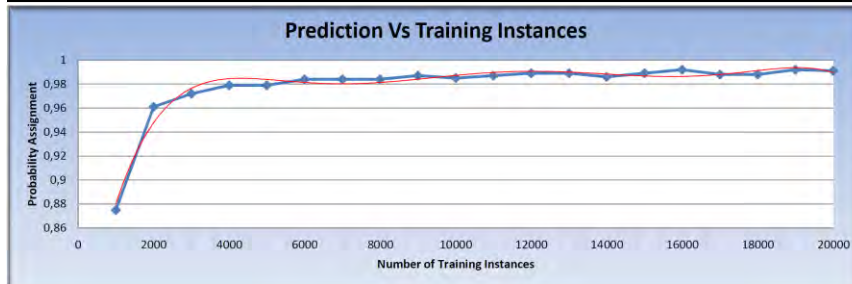
Dataset	Prediction	Time
1000	0.87500	9
2000	0.96400	9
3000	0.97400	11
4000	0.98100	11
5000	0.98000	11
6000	0.98500	11
7000	0.98600	12
8000	0.98600	11
9000	0.98700	12
10000	0.98600	12
11000	0.98900	12
12000	0.98800	12
13000	0.99100	13
14000	0.98600	12
15000	0.99100	12
16000	0.99100	13
17000	0.99000	13
18000	0.98800	14
19000	0.99400	13
20000	0.99000	13
sum		236



alphabetSize	128
expediteFactor	2
trieDepthFactor	45
heuristicFactor	0.167
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98010

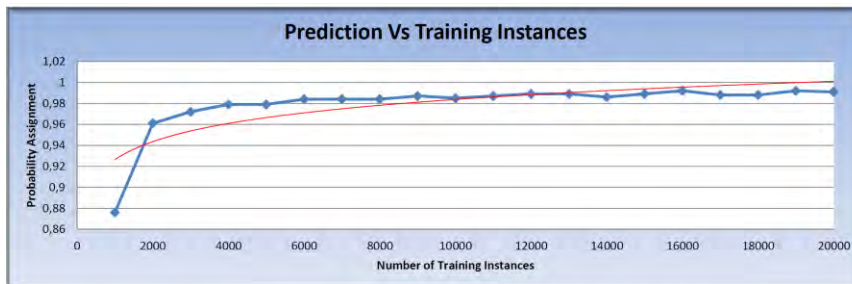
Dataset	Prediction	Time
1000	0.87500	11
2000	0.96400	11
3000	0.97400	12
4000	0.98100	13
5000	0.98000	13
6000	0.98500	13
7000	0.98600	14
8000	0.98600	13
9000	0.98700	14
10000	0.98600	14
11000	0.98900	14
12000	0.98800	14
13000	0.99100	14
14000	0.98600	16
15000	0.99100	14
16000	0.99100	15
17000	0.99000	15
18000	0.98800	15
19000	0.99400	15
20000	0.99000	15
sum		275

heuristicFactor



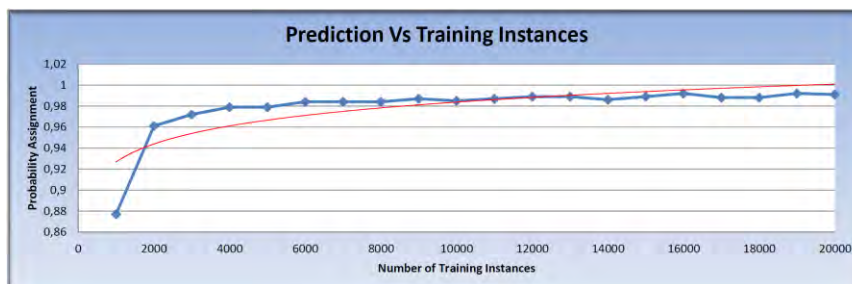
alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.05
observationFactor	1000
Last Prediction	0.99100
mean Prediction	0.97910

Dataset	Prediction	Time
1000	0.87500	2
2000	0.96100	2
3000	0.97200	2
4000	0.97900	2
5000	0.97900	2
6000	0.98400	2
7000	0.98400	2
8000	0.98400	3
9000	0.98700	2
10000	0.98500	3
11000	0.98700	2
12000	0.98900	3
13000	0.98900	3
14000	0.98600	3
15000	0.98900	3
16000	0.99200	3
17000	0.98800	3
18000	0.98800	3
19000	0.99200	4
20000	0.99100	3
sum		52



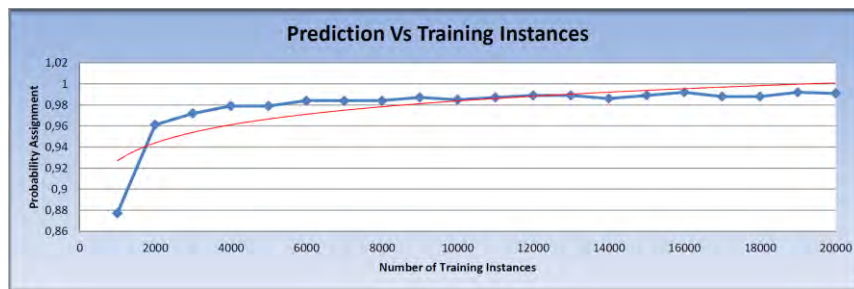
alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.06
observationFactor	1000
Last Prediction	1
mean Prediction	1

Dataset	Prediction	Time
1000	0.87600	1
2000	0.96100	1
3000	0.97200	2
4000	0.97900	2
5000	0.97900	2
6000	0.98400	2
7000	0.98400	2
8000	0.98400	3
9000	0.98700	2
10000	0.98500	3
11000	0.98700	2
12000	0.98900	3
13000	0.98900	3
14000	0.98600	3
15000	0.98900	3
16000	0.99200	3
17000	0.98800	3
18000	0.98800	3
19000	0.99200	3
20000	0.99100	4
sum		50



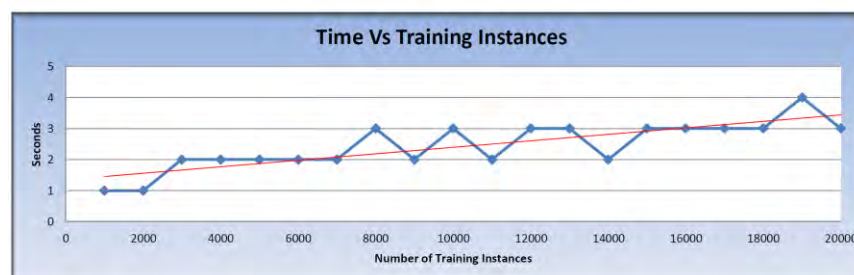
alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.07
observationFactor	1000
Last Prediction	0.99100
mean Prediction	0.97915

Dataset	Prediction	Time
1000	0.87700	2
2000	0.96100	2
3000	0.97200	2
4000	0.97900	2
5000	0.97900	2
6000	0.98400	2
7000	0.98400	2
8000	0.98400	3
9000	0.98700	2
10000	0.98500	3
11000	0.98700	2
12000	0.98900	3
13000	0.98900	3
14000	0.98600	3
15000	0.98900	3
16000	0.99200	3
17000	0.98800	3
18000	0.98800	3
19000	0.99200	3
20000	0.99100	4
sum		52



alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.08
observationFactor	1000
Last Prediction	0,99100
mean Prediction	0,97915

Dataset	Prediction	Time
1000	0,87700	1
2000	0,96100	1
3000	0,97200	2
4000	0,97900	2
5000	0,97900	2
6000	0,98400	2
7000	0,98400	3
8000	0,98400	2
9000	0,98700	2
10000	0,98500	3
11000	0,98700	2
12000	0,98900	3
13000	0,98900	3
14000	0,98600	3
15000	0,98900	2
16000	0,99200	3
17000	0,98800	4
18000	0,98800	3
19000	0,99200	3
20000	0,99100	3
sum		49



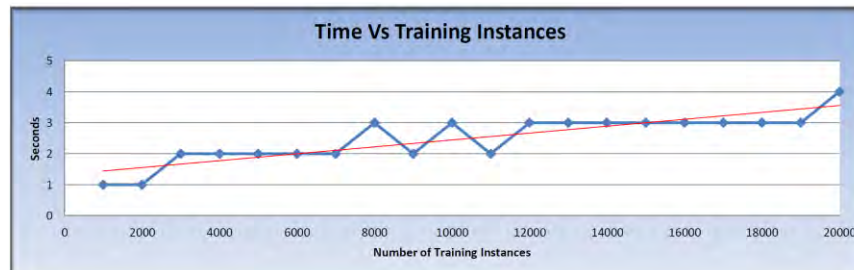
alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.09
observationFactor	1000
Last Prediction	0,99100
mean Prediction	0,97920

Dataset	Prediction	Time
1000	0,87800	1
2000	0,96100	1
3000	0,97200	2
4000	0,97900	2
5000	0,97900	2
6000	0,98400	2
7000	0,98400	2
8000	0,98400	3
9000	0,98700	2
10000	0,98500	3
11000	0,98700	2
12000	0,98900	3
13000	0,98900	3
14000	0,98600	2
15000	0,98900	3
16000	0,99200	3
17000	0,98800	3
18000	0,98800	3
19000	0,99200	4
20000	0,99100	3
sum		49



alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.1
observationFactor	1000
Last Prediction	0,99100
mean Prediction	0,97920

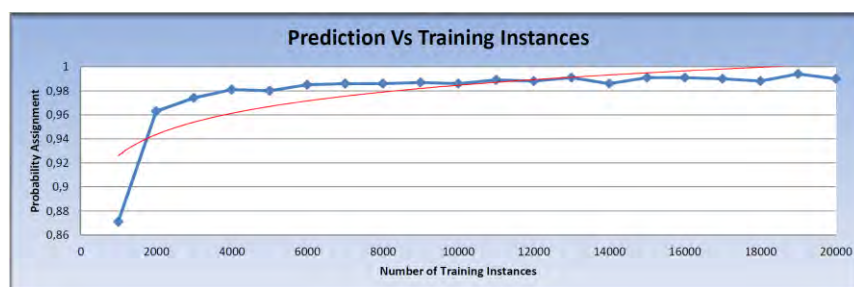
Dataset	Prediction	Time
1000	0,87800	2
2000	0,96100	2
3000	0,97200	2
4000	0,97900	2
5000	0,97900	2
6000	0,98400	2
7000	0,98400	2
8000	0,98400	2
9000	0,98700	3
10000	0,98500	2
11000	0,98700	3
12000	0,98900	3
13000	0,98900	3
14000	0,98600	2
15000	0,98900	3
16000	0,99200	3
17000	0,98800	3
18000	0,98800	3
19000	0,99200	4
20000	0,99100	3
sum		51



alphabetSize	128	
expediteFactor	2	
trieDepthFactor	8	
heuristicFactor	0.11	
observationFactor	1000	
Last Prediction	0.99100	
mean Prediction	0.97910	
Dataset	Prediction	Time
1000	0.87600	1
2000	0.96100	1
3000	0.97200	2
4000	0.97900	2
5000	0.97900	2
6000	0.98400	2
7000	0.98400	2
8000	0.98400	3
9000	0.98700	2
10000	0.98500	3
11000	0.98700	2
12000	0.98900	3
13000	0.98900	3
14000	0.98600	3
15000	0.98900	3
16000	0.99200	3
17000	0.98800	3
18000	0.98800	3
19000	0.99200	3
20000	0.99100	4
	sum	50



alphabetSize	128	
expediteFactor	2	
trieDepthFactor	8	
heuristicFactor	0.12	
observationFactor	1000	
Last Prediction	0.99100	
mean Prediction	0.97900	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96100	2
3000	0.97200	2
4000	0.97900	2
5000	0.97900	2
6000	0.98400	2
7000	0.98400	2
8000	0.98400	2
9000	0.98700	3
10000	0.98500	2
11000	0.98700	3
12000	0.98900	3
13000	0.98900	3
14000	0.98600	3
15000	0.98900	3
16000	0.99200	3
17000	0.98800	3
18000	0.98800	3
19000	0.99200	4
20000	0.99100	3
sum		52



alphabetSize	128	
expediteFactor	2	
trieDepthFactor	8	
heuristicFactor	0.13	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97985	
Dataset	Prediction	Time
1000	0.87100	1
2000	0.96300	1
3000	0.97400	2
4000	0.98100	2
5000	0.98000	3
6000	0.98500	2
7000	0.98600	2
8000	0.98600	3
9000	0.98700	2
10000	0.98600	3
11000	0.98900	2
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	4
19000	0.99400	3
20000	0.99000	3
	sum	51



alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.14
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.97990

Dataset	Prediction	Time
1000	0.87100	1
2000	0.96400	1
3000	0.97400	2
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98600	2
8000	0.98600	3
9000	0.98700	2
10000	0.98600	3
11000	0.98900	2
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	3
19000	0.99400	3
20000	0.99000	4
sum		50



alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.15
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.97995

Dataset	Prediction	Time
1000	0.87200	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	1
5000	0.98000	2
6000	0.98500	3
7000	0.98600	2
8000	0.98600	2
9000	0.98700	3
10000	0.98600	2
11000	0.98900	3
12000	0.98800	3
13000	0.99100	2
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	3
19000	0.99400	3
20000	0.99000	4
sum		51



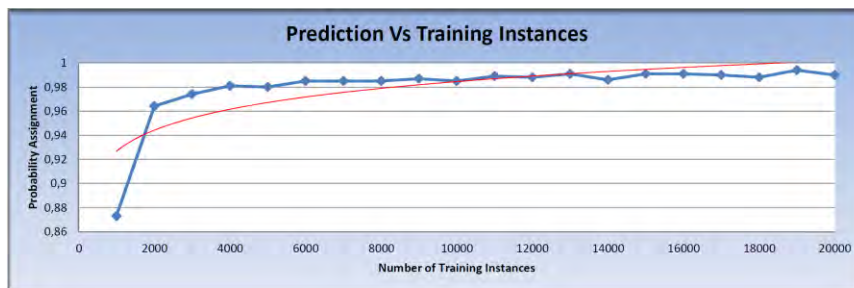
alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.16
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.97995

Dataset	Prediction	Time
1000	0.87200	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98600	2
8000	0.98600	2
9000	0.98700	3
10000	0.98600	2
11000	0.98900	3
12000	0.98800	3
13000	0.99100	2
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	3
19000	0.99400	4
20000	0.99000	3
sum		51



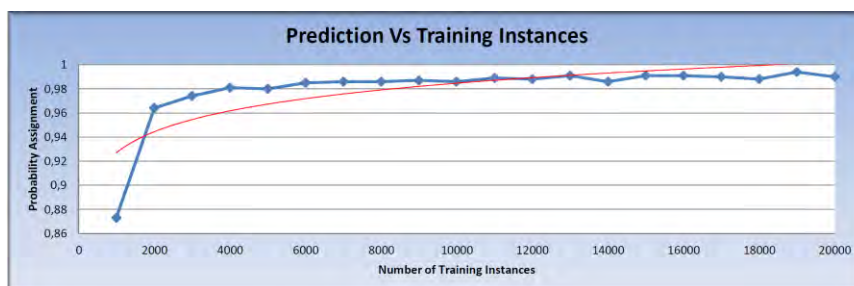
alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.17
observationFactor	1000
Last Prediction	0,99000
mean Prediction	0,98005

Dataset	Prediction	Time
1000	0,87400	1
2000	0,96400	1
3000	0,97400	2
4000	0,98100	2
5000	0,98000	2
6000	0,98500	2
7000	0,98600	3
8000	0,98600	2
9000	0,98700	2
10000	0,98600	3
11000	0,98900	3
12000	0,98800	2
13000	0,99100	3
14000	0,98600	3
15000	0,99100	3
16000	0,99100	3
17000	0,99000	3
18000	0,98800	3
19000	0,99400	4
20000	0,99000	3
sum		50



alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.18
observationFactor	1000
Last Prediction	0,99000
mean Prediction	0,98000

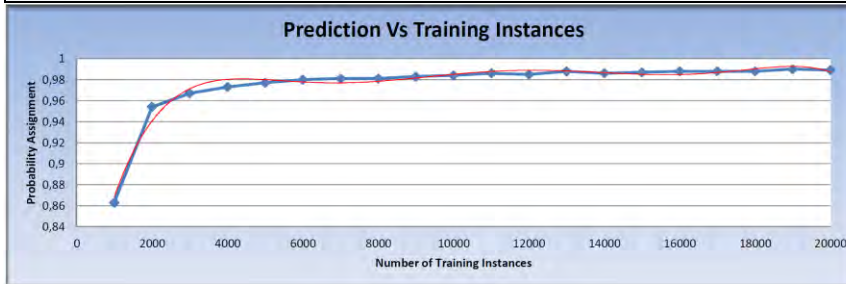
Dataset	Prediction	Time
1000	0,87300	2
2000	0,96400	2
3000	0,97400	1
4000	0,98100	2
5000	0,98000	2
6000	0,98500	2
7000	0,98500	2
8000	0,98500	3
9000	0,98700	2
10000	0,98500	3
11000	0,98900	2
12000	0,98800	3
13000	0,99100	3
14000	0,98600	3
15000	0,99100	3
16000	0,99100	3
17000	0,99000	3
18000	0,98800	4
19000	0,99400	3
20000	0,99000	3
sum		51



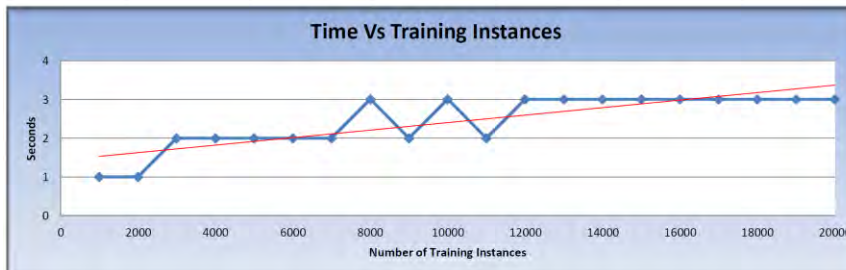
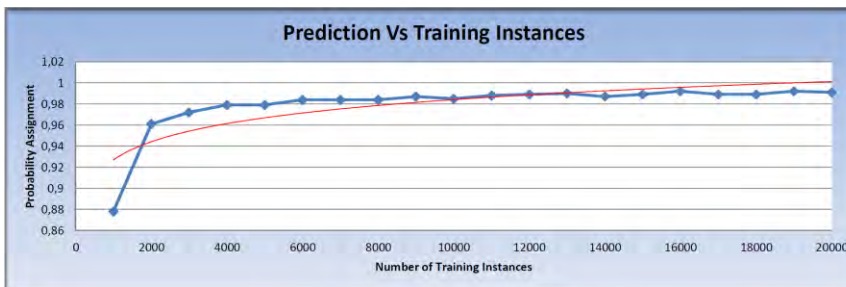
alphabetSize	128
expediteFactor	2
trieDepthFactor	8
heuristicFactor	0.19
observationFactor	1000
Last Prediction	0,99000
mean Prediction	0,98000

Dataset	Prediction	Time
1000	0,87300	1
2000	0,96400	1
3000	0,97400	2
4000	0,98100	2
5000	0,98000	2
6000	0,98500	2
7000	0,98600	3
8000	0,98600	2
9000	0,98700	2
10000	0,98600	3
11000	0,98900	2
12000	0,98800	3
13000	0,99100	3
14000	0,98600	3
15000	0,99100	3
16000	0,99100	3
17000	0,99000	3
18000	0,98800	4
19000	0,99400	3
20000	0,99000	3
sum		50

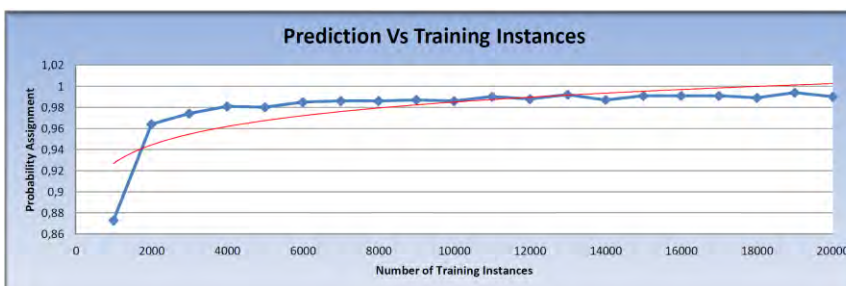
heuristicFactor 2



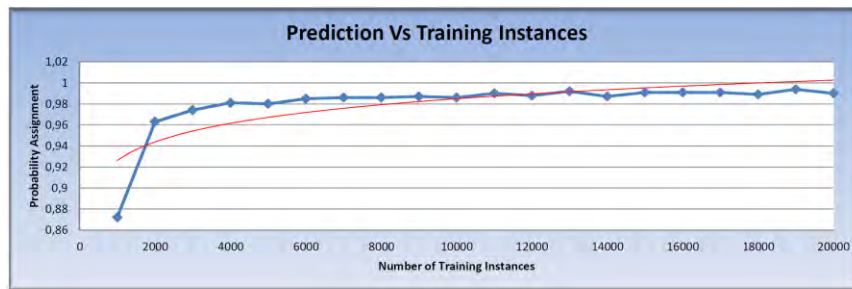
alphabetSize	128	
expediteFactor	2	
trieDepthFactor	8	
heuristicFactor	0,0	
observationFactor	1000	
Last Prediction	0,98900	
mean Prediction	0,97590	
Dataset	Prediction	Time
1000	0,86300	1
2000	0,95400	1
3000	0,96700	2
4000	0,97300	2
5000	0,97700	2
6000	0,98000	2
7000	0,98100	2
8000	0,98100	2
9000	0,98300	3
10000	0,98400	2
11000	0,98600	3
12000	0,98500	3
13000	0,98800	2
14000	0,98600	3
15000	0,98700	3
16000	0,98800	3
17000	0,98800	4
18000	0,98800	3
19000	0,99000	3
20000	0,98900	3
sum	49	



alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0,1	
observationFactor	1000	
Last Prediction	0,99100	
mean Prediction	0,97945	
Dataset	Prediction	Time
1000	0,87800	1
2000	0,96100	1
3000	0,97200	2
4000	0,97900	2
5000	0,97900	2
6000	0,98400	2
7000	0,98400	2
8000	0,98400	3
9000	0,98700	2
10000	0,98500	3
11000	0,98800	2
12000	0,98900	3
13000	0,99000	3
14000	0,98700	3
15000	0,98900	3
16000	0,99200	3
17000	0,98900	3
18000	0,98900	3
19000	0,99200	3
20000	0,99100	3
sum		49

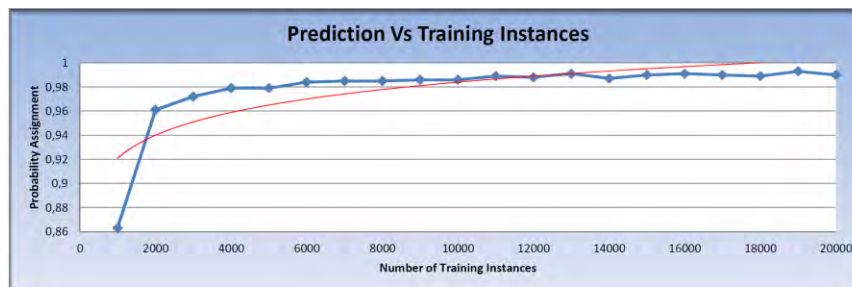


alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0,2	
observationFactor	1000	
Last Prediction	0,99000	
mean Prediction	0,98025	
Dataset	Prediction	Time
1000	0,87300	2
2000	0,96400	2
3000	0,97400	2
4000	0,98100	2
5000	0,98000	2
6000	0,98500	2
7000	0,98600	2
8000	0,98600	2
9000	0,98700	3
10000	0,98600	2
11000	0,99000	3
12000	0,98800	2
13000	0,99200	3
14000	0,98700	3
15000	0,99100	3
16000	0,99100	3
17000	0,99100	3
18000	0,98900	4
19000	0,99400	3
20000	0,99000	3
sum		51

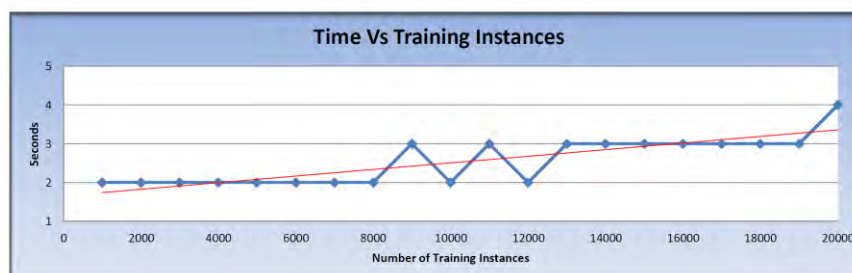


alphabetSize	128
expediteFactor	1
trieDepthFactor	8
heuristicFactor	0.3
observationFactor	1000
Last Prediction	0.99000
mean Prediction	0.98015

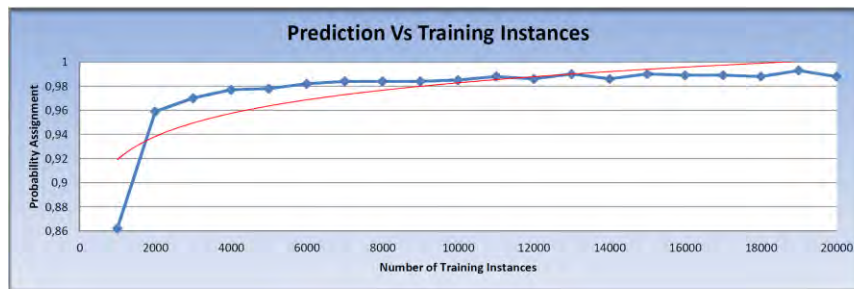
Dataset	Prediction	Time
1000	0.87200	1
2000	0.96300	1
3000	0.97400	2
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98600	2
8000	0.98600	3
9000	0.98700	2
10000	0.98600	3
11000	0.99000	2
12000	0.98800	3
13000	0.99200	3
14000	0.98700	3
15000	0.99100	3
16000	0.99100	3
17000	0.99100	3
18000	0.98900	3
19000	0.99400	3
20000	0.99000	3
sum		49



alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.4	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97890	
Dataset	Prediction	Time
1000	0.86300	2
2000	0.96100	2
3000	0.97200	2
4000	0.97900	2
5000	0.97900	2
6000	0.98400	2
7000	0.98500	2
8000	0.98500	2
9000	0.98600	3
10000	0.98600	2
11000	0.98900	3
12000	0.98800	3
13000	0.99100	3
14000	0.98700	2
15000	0.99000	3
16000	0.99100	3
17000	0.99000	4
18000	0.98900	3
19000	0.99300	3
20000	0.99000	3
sum		51



alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.5	
observationFactor	1000	
Last Prediction	0.98800	
mean Prediction	0.97760	
Dataset	Prediction	Time
1000	0.86200	2
2000	0.95900	2
3000	0.97000	2
4000	0.97700	2
5000	0.97800	2
6000	0.98200	2
7000	0.98400	2
8000	0.98400	2
9000	0.98400	3
10000	0.98500	2
11000	0.98800	3
12000	0.98600	2
13000	0.99000	3
14000	0.98600	3
15000	0.99000	3
16000	0.98900	3
17000	0.98900	3
18000	0.98800	3
19000	0.99300	3
20000	0.98800	4
sum		51



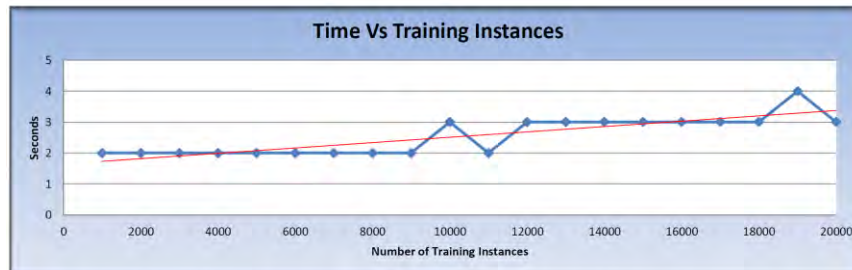
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0,6	
observationFactor	1000	
Last Prediction	0,98800	
mean Prediction	0,97760	
Dataset	Prediction	Time
1000	0,86200	1
2000	0,95900	1
3000	0,97000	2
4000	0,97700	2
5000	0,97800	2
6000	0,98200	2
7000	0,98400	2
8000	0,98400	2
9000	0,98400	3
10000	0,98500	3
11000	0,98800	2
12000	0,98600	3
13000	0,99000	2
14000	0,98600	3
15000	0,99000	3
16000	0,98900	3
17000	0,98900	3
18000	0,98800	3
19000	0,99300	3
20000	0,98800	4
	sum	49



alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0,7	
observationFactor	1000	
Last Prediction	0,98800	
mean Prediction	0,97755	
Dataset	Prediction	Time
1000	0,86100	1
2000	0,95900	1
3000	0,97000	2
4000	0,97700	2
5000	0,97800	2
6000	0,98200	2
7000	0,98400	3
8000	0,98400	2
9000	0,98400	3
10000	0,98500	3
11000	0,98800	2
12000	0,98600	3
13000	0,99000	4
14000	0,98600	3
15000	0,99000	3
16000	0,98900	4
17000	0,98900	3
18000	0,98800	4
19000	0,99300	4
20000	0,98800	4
	sum	55



alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0,8	
observationFactor	1000	
Last Prediction	0,98800	
mean Prediction	0,97755	
Dataset	Prediction	Time
1000	0,86100	2
2000	0,95900	2
3000	0,97000	1
4000	0,97700	2
5000	0,97800	2
6000	0,98200	2
7000	0,98400	3
8000	0,98400	2
9000	0,98400	3
10000	0,98500	2
11000	0,98800	3
12000	0,98600	2
13000	0,99000	3
14000	0,98600	3
15000	0,99000	3
16000	0,98900	3
17000	0,98900	3
18000	0,98800	3
19000	0,99300	4
20000	0,98800	3
sum		51

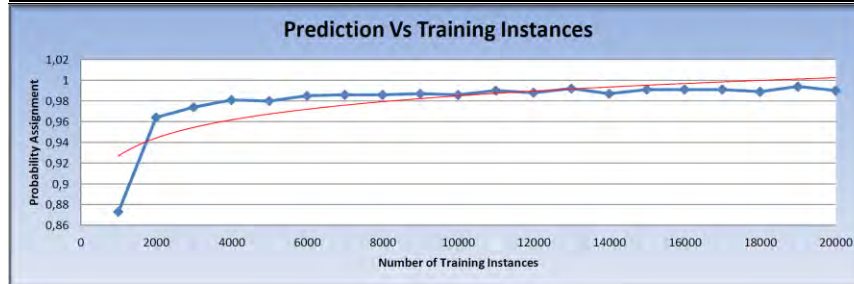


alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0,9	
observationFactor	1000	
Last Prediction	0,98800	
mean Prediction	0,97755	
Dataset	Prediction	Time
1000	0,86100	2
2000	0,95900	2
3000	0,97000	2
4000	0,97700	2
5000	0,97800	2
6000	0,98200	2
7000	0,98400	2
8000	0,98400	2
9000	0,98400	2
10000	0,98500	3
11000	0,98800	2
12000	0,98600	3
13000	0,99000	3
14000	0,98600	3
15000	0,99000	3
16000	0,98900	3
17000	0,98900	3
18000	0,98800	3
19000	0,99300	4
20000	0,98800	3
sum		51

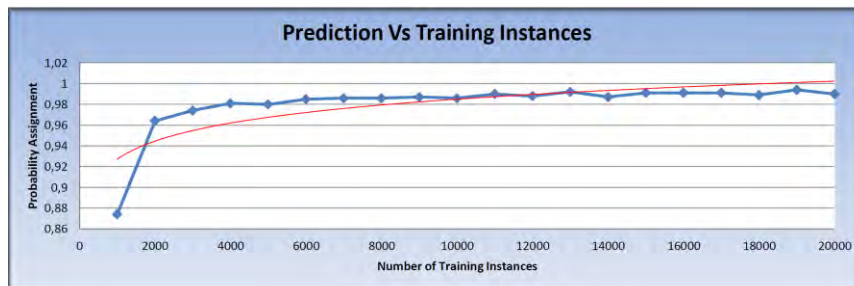


alphabetSize	128	
expediteFactor	2	
trieDepthFactor	8	
heuristicFactor	1,0	
observationFactor	1000	
Last Prediction	0,98800	
mean Prediction	0,97755	
Dataset	Prediction	Time
1000	0,86100	1
2000	0,95900	1
3000	0,97000	2
4000	0,97700	2
5000	0,97800	2
6000	0,98200	2
7000	0,98400	3
8000	0,98400	2
9000	0,98400	3
10000	0,98500	3
11000	0,98800	3
12000	0,98600	2
13000	0,99000	3
14000	0,98600	4
15000	0,99000	3
16000	0,98900	3
17000	0,98900	3
18000	0,98800	4
19000	0,99300	3
20000	0,98800	4
sum		53

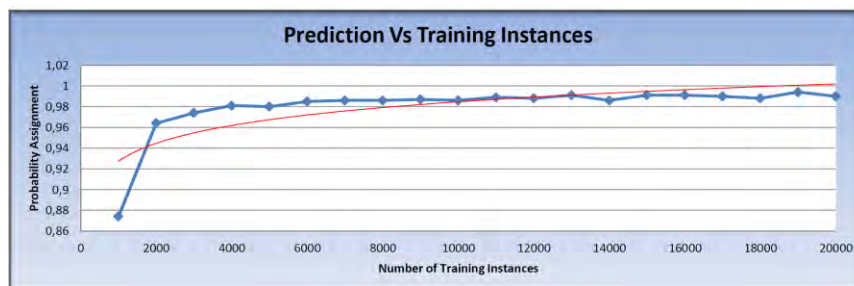
expediteFactor



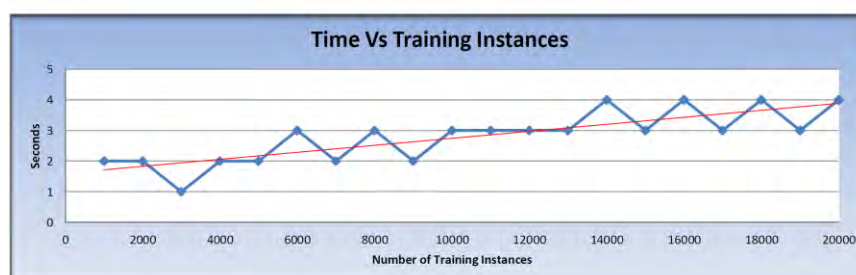
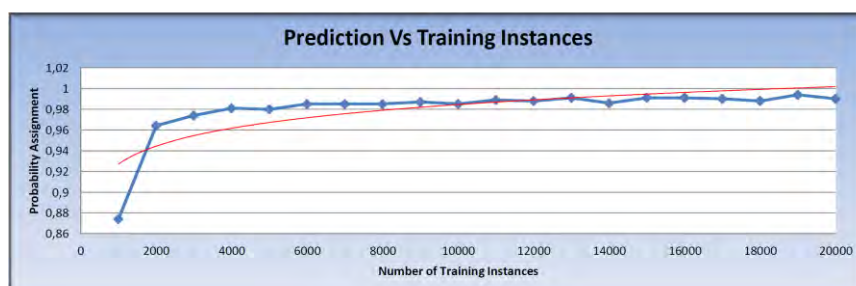
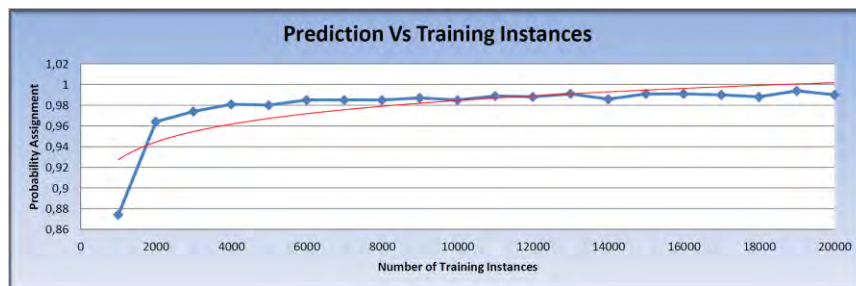
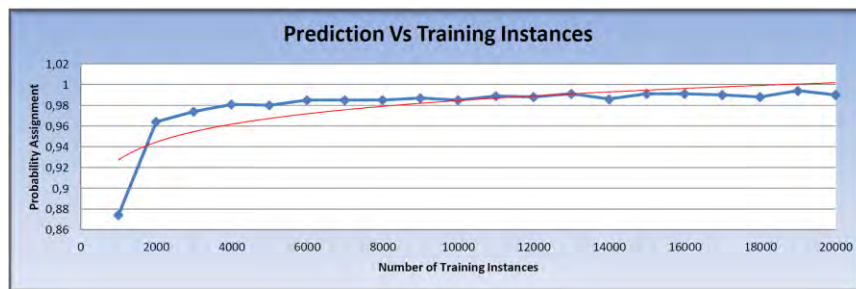
alphabetSize	128	
expediteFactor	0	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.98025	
Dataset	Prediction	Time
1000	0.87300	2
2000	0.96400	2
3000	0.97400	1
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98600	3
8000	0.98600	2
9000	0.98700	2
10000	0.98600	3
11000	0.99000	2
12000	0.98800	3
13000	0.99200	3
14000	0.98700	2
15000	0.99100	3
16000	0.99100	4
17000	0.99100	3
18000	0.98900	3
19000	0.99400	3
20000	0.99000	3
sum		50



alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.98030	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98600	2
8000	0.98600	3
9000	0.98700	2
10000	0.98600	3
11000	0.99000	2
12000	0.98800	3
13000	0.99200	3
14000	0.98700	2
15000	0.99100	3
16000	0.99100	3
17000	0.99100	3
18000	0.98900	4
19000	0.99400	3
20000	0.99000	3
sum	51	



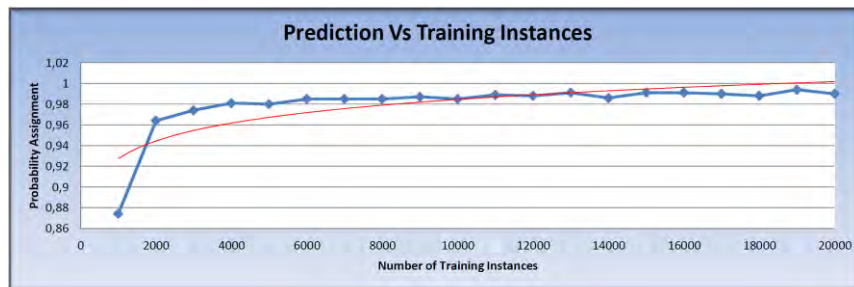
alphabetSize	128	
expediteFactor	2	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.98005	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96400	2
3000	0.97400	1
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98600	3
8000	0.98600	2
9000	0.98700	2
10000	0.98600	3
11000	0.98900	2
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	3
19000	0.99400	4
20000	0.99000	3
sum	51	



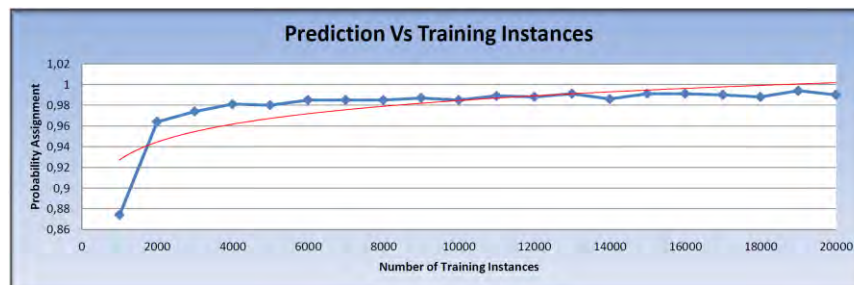
alphabetSize	128	
expediteFactor	3	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97990	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96400	2
3000	0.97400	1
4000	0.98100	2
5000	0.98000	2
6000	0.98500	3
7000	0.98500	2
8000	0.98500	2
9000	0.98700	3
10000	0.98500	2
11000	0.98900	3
12000	0.98800	3
13000	0.99100	3
14000	0.98600	2
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	3
19000	0.99400	4
20000	0.99000	3
sum		51

alphabetSize	128	
expediteFactor	4	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97990	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	1
5000	0.98000	3
6000	0.98500	2
7000	0.98500	2
8000	0.98500	2
9000	0.98700	3
10000	0.98500	3
11000	0.98900	2
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	3
19000	0.99400	3
20000	0.99000	3
sum	51	

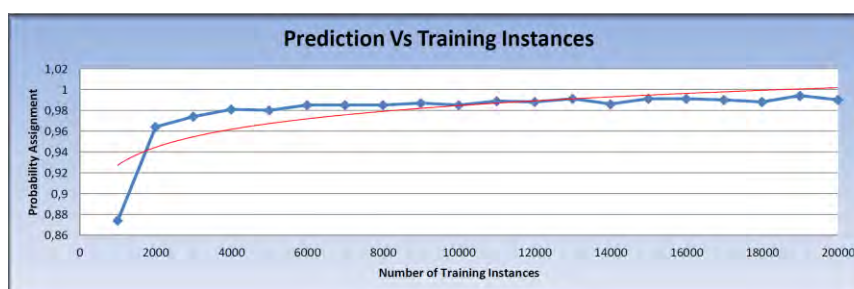
alphabetSize	128	
expediteFactor	5	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97990	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96400	2
3000	0.97400	1
4000	0.98100	2
5000	0.98000	2
6000	0.98500	3
7000	0.98500	2
8000	0.98500	3
9000	0.98700	2
10000	0.98500	3
11000	0.98900	3
12000	0.98800	3
13000	0.99100	3
14000	0.98600	4
15000	0.99100	3
16000	0.99100	4
17000	0.99000	3
18000	0.98800	4
19000	0.99400	3
20000	0.99000	4
sum		56



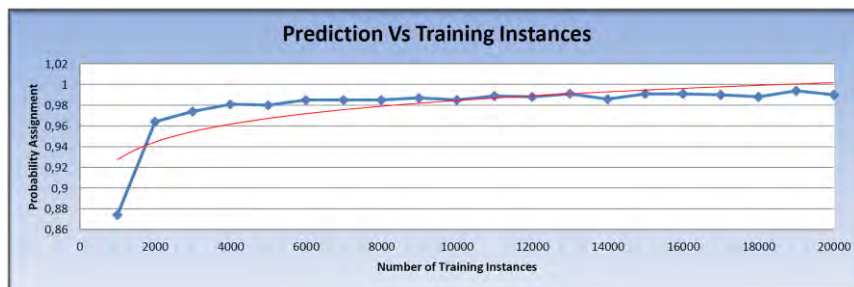
alphabetSize	128	
expediteFactor	6	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97990	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98500	2
8000	0.98500	3
9000	0.98700	2
10000	0.98500	3
11000	0.98900	3
12000	0.98800	2
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	4
17000	0.99000	3
18000	0.98800	3
19000	0.99400	4
20000	0.99000	4
sum		54



alphabetSize	128	
expediteFactor	7	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97990	
Dataset	Prediction	Time
1000	0.87400	2
2000	0.96400	2
3000	0.97400	2
4000	0.98100	2
5000	0.98000	2
6000	0.98500	2
7000	0.98500	3
8000	0.98500	2
9000	0.98700	3
10000	0.98500	2
11000	0.98900	3
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	4
18000	0.98800	3
19000	0.99400	3
20000	0.99000	4
sum		54



alphabetSize	128	
expediteFactor	8	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97990	
Dataset	Prediction	Time
1000	0.87400	1
2000	0.96400	1
3000	0.97400	2
4000	0.98100	2
5000	0.98000	3
6000	0.98500	2
7000	0.98500	2
8000	0.98500	3
9000	0.98700	3
10000	0.98500	2
11000	0.98900	3
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	3
18000	0.98800	4
19000	0.99400	3
20000	0.99000	3
sum		52

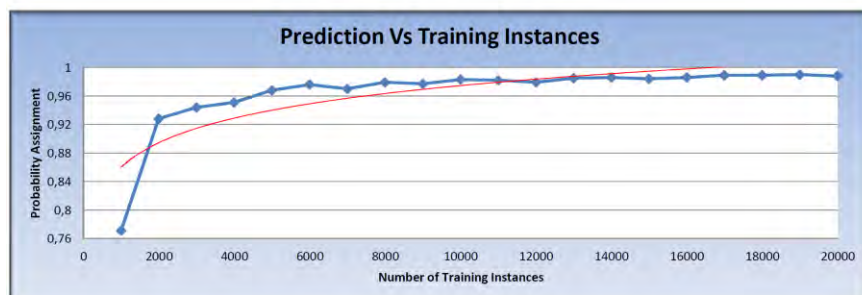
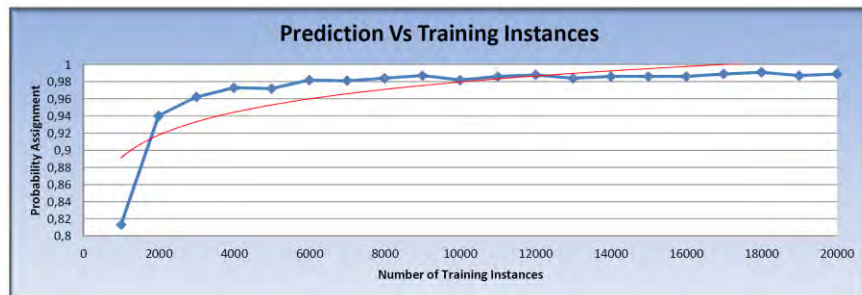
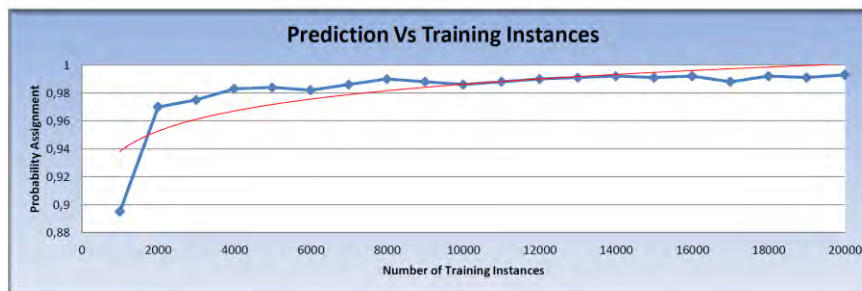


alphabetSize	128	
expediteFactor	9	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99000	
mean Prediction	0.97990	
Dataset	Prediction	Time
1000	0.87400	1
2000	0.96400	1
3000	0.97400	2
4000	0.98100	2
5000	0.98000	2
6000	0.98500	3
7000	0.98500	2
8000	0.98500	2
9000	0.98700	3
10000	0.98500	3
11000	0.98900	2
12000	0.98800	3
13000	0.99100	3
14000	0.98600	3
15000	0.99100	3
16000	0.99100	3
17000	0.99000	4
18000	0.98800	3
19000	0.99400	3
20000	0.99000	4
sum		52



alphabetSize	128	
expediteFactor	10	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0,99000	
mean Prediction	0,97990	
Dataset	Prediction	Time
1000	0,87400	1
2000	0,96400	1
3000	0,97400	2
4000	0,98100	2
5000	0,98000	2
6000	0,98500	3
7000	0,98500	2
8000	0,98500	2
9000	0,98700	3
10000	0,98500	3
11000	0,98900	2
12000	0,98800	3
13000	0,99100	3
14000	0,98600	3
15000	0,99100	3
16000	0,99100	3
17000	0,99000	3
18000	0,98800	4
19000	0,99400	3
20000	0,99000	3
sum		51

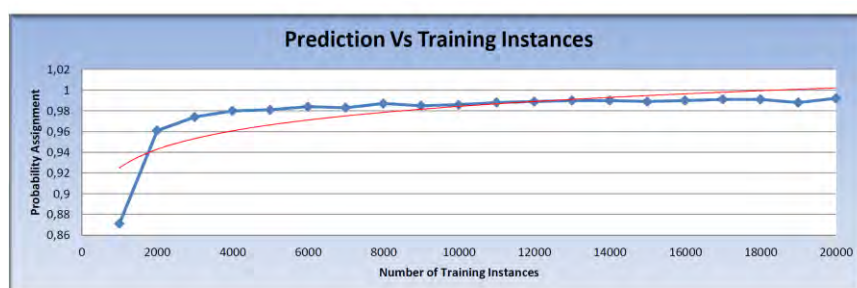
Alphabet Size / Char Sequence



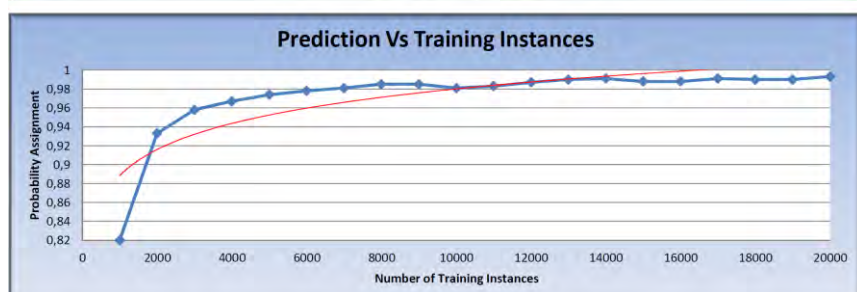
char_Set		5 / 20
alphabetSize		128
expediteFactor		1
trieDepthFactor		8
heuristicFactor		0.167
observationFactor		1000
Last Prediction		0.99300
mean Prediction		0.98235
Dataset	Prediction	Time
1000	0,89500	1
2000	0,97000	1
3000	0,97500	2
4000	0,98300	2
5000	0,98400	2
6000	0,98200	3
7000	0,98600	2
8000	0,99000	2
9000	0,98800	3
10000	0,98600	2
11000	0,98800	3
12000	0,99000	3
13000	0,99100	3
14000	0,99200	2
15000	0,99100	4
16000	0,99200	3
17000	0,98800	3
18000	0,99200	3
19000	0,99100	4
20000	0,99300	3
Sum		51

char_Set	5 / 40	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.98900	
mean Prediction	0.97240	
Dataset	Prediction	Time
1000	0.81300	2
2000	0.94000	2
3000	0.96200	2
4000	0.97300	3
5000	0.97200	4
6000	0.98200	3
7000	0.98100	4
8000	0.98400	4
9000	0.98700	3
10000	0.98200	5
11000	0.98600	4
12000	0.98800	4
13000	0.98400	5
14000	0.98600	4
15000	0.98600	5
16000	0.98600	5
17000	0.98900	5
18000	0.99100	5
19000	0.98700	5
20000	0.98900	5
	Sum	79

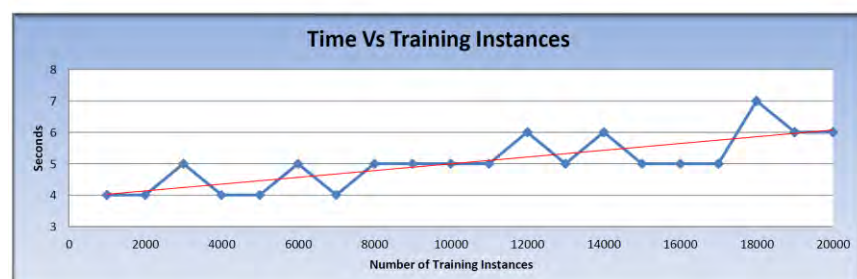
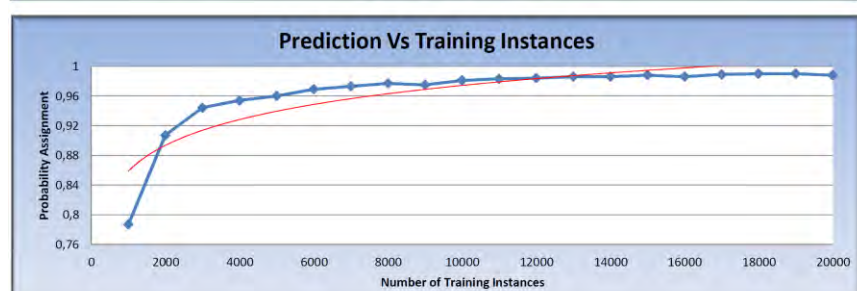
char_Set		5 / 60
alphabetSize		128
expediteFactor		1
trieDepthFactor		8
heuristicFactor		0.167
observationFactor		1000
Last Prediction		0.98800
mean Prediction		0.96525
Dataset	Prediction	Time
1000	0.77100	3
2000	0.92800	3
3000	0.94400	2
4000	0.95100	3
5000	0.96800	4
6000	0.97600	3
7000	0.97000	4
8000	0.97900	5
9000	0.97700	4
10000	0.98300	4
11000	0.98200	5
12000	0.97900	5
13000	0.98500	5
14000	0.98600	5
15000	0.98400	6
16000	0.98600	5
17000	0.98900	6
18000	0.98900	6
19000	0.99000	6
20000	0.98800	6
Sum		90



char_Set		10 / 30
alphabetSize		128
expediteFactor		1
trieDepthFactor		8
heuristicFactor		0.167
observationFactor		1000
Last Prediction		0.99200
mean Prediction		0.97950
Dataset	Prediction	Time
1000	0.87100	2
2000	0.96100	2
3000	0.97400	3
4000	0.98000	2
5000	0.98100	3
6000	0.98400	3
7000	0.98300	3
8000	0.98700	4
9000	0.98500	3
10000	0.98600	4
11000	0.98800	3
12000	0.98900	4
13000	0.99000	4
14000	0.99000	4
15000	0.98900	4
16000	0.99000	4
17000	0.99100	4
18000	0.99100	4
19000	0.98800	4
20000	0.99200	4
Sum		68



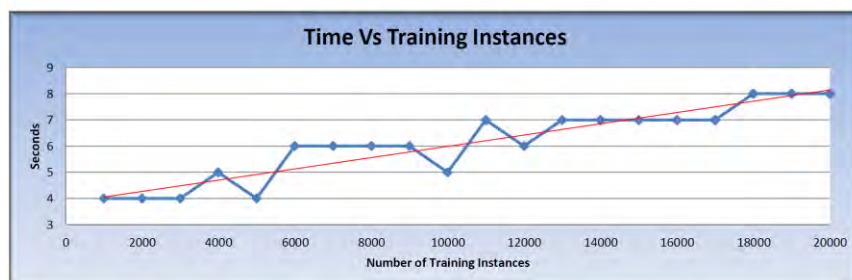
char_Set		10 / 60
alphabetSize		128
expediteFactor		1
trieDepthFactor		8
heuristicFactor		0.167
observationFactor		1000
Last Prediction		0.99300
mean Prediction		0.97265
Dataset	Prediction	Time
1000	0.82000	3
2000	0.93300	3
3000	0.95800	2
4000	0.96700	3
5000	0.97400	4
6000	0.97800	4
7000	0.98100	4
8000	0.98500	4
9000	0.98500	3
10000	0.98100	4
11000	0.98300	5
12000	0.98700	4
13000	0.99000	4
14000	0.99100	5
15000	0.98800	5
16000	0.98800	5
17000	0.99100	5
18000	0.99000	6
19000	0.99000	6
20000	0.99300	5
Sum		84



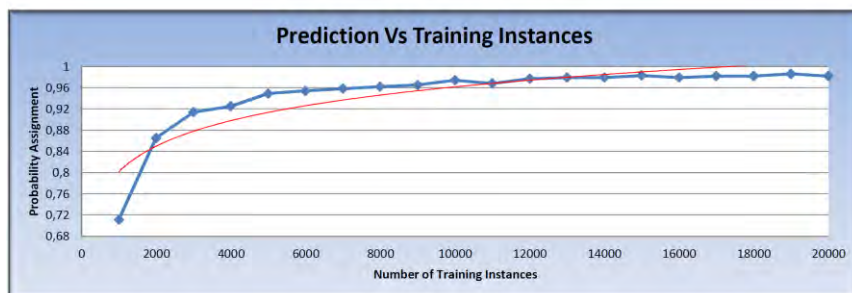
char_Set		10 / 90
alphabetSize		128
expediteFactor		1
trieDepthFactor		8
heuristicFactor		0.167
observationFactor		1000
Last Prediction		0.98800
mean Prediction		0.96485
Dataset	Prediction	Time
1000	0.78700	4
2000	0.90700	4
3000	0.94400	5
4000	0.95400	4
5000	0.96000	4
6000	0.96900	5
7000	0.97300	4
8000	0.97700	5
9000	0.97500	5
10000	0.98100	5
11000	0.98300	5
12000	0.98400	6
13000	0.98600	5
14000	0.98600	6
15000	0.98800	5
16000	0.98600	5
17000	0.98900	5
18000	0.99000	7
19000	0.99000	6
20000	0.98800	6
Sum		101



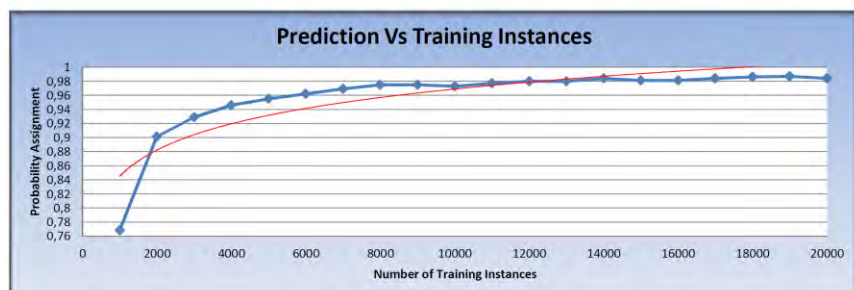
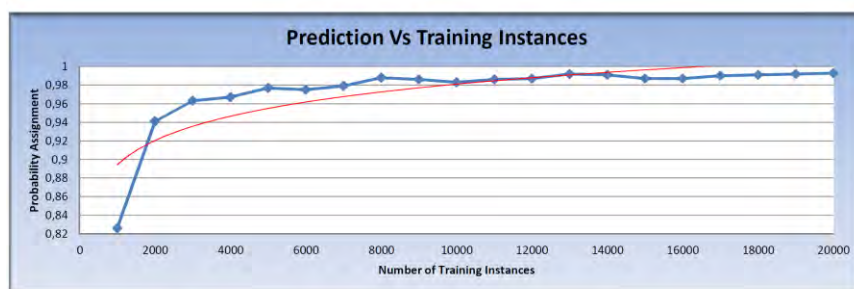
char_Set	15 / 45	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99300	
mean Prediction	0.97885	
Dataset	Prediction	Time
1000	0.83500	3
2000	0.94800	3
3000	0.96900	2
4000	0.98000	4
5000	0.98400	4
6000	0.98400	4
7000	0.98800	4
8000	0.98800	4
9000	0.99100	4
10000	0.99000	4
11000	0.98900	5
12000	0.99000	4
13000	0.99300	5
14000	0.99200	4
15000	0.99200	5
16000	0.99400	5
17000	0.99200	5
18000	0.99100	5
19000	0.99400	5
20000	0.99300	6
Sum	85	



char_Set	15 / 90	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.98900	
mean Prediction	0.96475	
Dataset	Prediction	Time
1000	0,76500	4
2000	0,90900	4
3000	0,94100	4
4000	0,95600	5
5000	0,96700	4
6000	0,97200	6
7000	0,97800	6
8000	0,97700	6
9000	0,98200	6
10000	0,98200	5
11000	0,98400	7
12000	0,98200	6
13000	0,98600	7
14000	0,98700	7
15000	0,98700	7
16000	0,99000	7
17000	0,98700	7
18000	0,98500	8
19000	0,98900	8
20000	0,98900	8
Sum		122



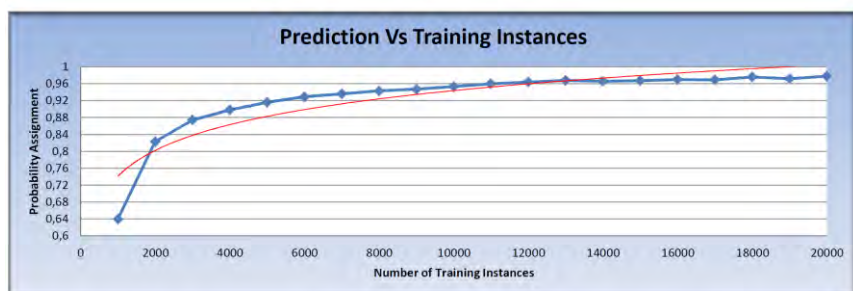
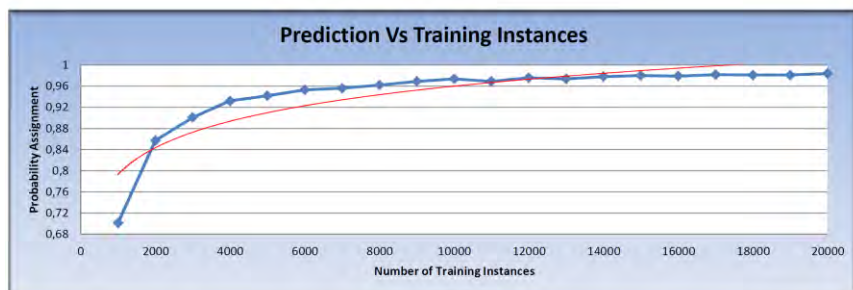
char_Set	15 / 135	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.98200	
mean Prediction	0.94870	
Dataset	Prediction	Time
1000	0.71100	6
2000	0.86500	6
3000	0.91400	6
4000	0.92500	5
5000	0.94900	6
6000	0.95400	6
7000	0.95800	6
8000	0.96200	8
9000	0.96500	8
10000	0.97400	7
11000	0.96800	8
12000	0.97700	8
13000	0.97900	8
14000	0.97900	8
15000	0.98300	8
16000	0.97900	8
17000	0.98200	9
18000	0.98200	8
19000	0.98600	9
20000	0.98200	8
Sum		146



char_Set	20 / 60	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.99300	
mean Prediction	0.97405	
Dataset	Prediction	Time
1000	0,82600	4
2000	0,94100	4
3000	0,96300	4
4000	0,96700	4
5000	0,97700	5
6000	0,97500	4
7000	0,97900	5
8000	0,98800	4
9000	0,98600	5
10000	0,98300	5
11000	0,98600	6
12000	0,98700	5
13000	0,99200	6
14000	0,99100	6
15000	0,98700	5
16000	0,98700	6
17000	0,99000	6
18000	0,99100	7
19000	0,99200	6
20000	0,99300	6
Sum		103

char_Set	20 / 120	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.98400	
mean Prediction	0.95885	
Dataset	Prediction	Time
1000	0.76800	5
2000	0.90100	5
3000	0.92900	5
4000	0.94600	6
5000	0.95500	6
6000	0.96200	6
7000	0.96900	7
8000	0.97500	7
9000	0.97500	7
10000	0.97300	7
11000	0.97700	8
12000	0.98000	7
13000	0.98000	8
14000	0.98400	8
15000	0.98100	7
16000	0.98100	9
17000	0.98400	8
18000	0.98600	8
19000	0.98700	8
20000	0.98400	8
Sum		140

char_Set	20 / 180	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.98200	
mean Prediction	0.94195	
Dataset	Prediction	Time
1000	0.71000	9
2000	0.84500	9
3000	0.89600	9
4000	0.91800	9
5000	0.92900	9
6000	0.94200	9
7000	0.95900	9
8000	0.95700	9
9000	0.96500	9
10000	0.96500	9
11000	0.96600	10
12000	0.96800	9
13000	0.97400	9
14000	0.97500	10
15000	0.97900	9
16000	0.97500	9
17000	0.97800	10
18000	0.97700	10
19000	0.97900	9
20000	0.98200	10
Sum		185



char_Set	26 / 75	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.98800	
mean Prediction	0.96795	
Dataset	Prediction	Time
1000	0.77500	4
2000	0.92300	4
3000	0.95700	4
4000	0.96400	4
5000	0.97200	4
6000	0.97300	6
7000	0.97800	5
8000	0.98100	6
9000	0.98200	6
10000	0.98000	5
11000	0.98300	7
12000	0.98300	6
13000	0.98600	6
14000	0.98800	7
15000	0.98700	6
16000	0.99000	7
17000	0.98800	8
18000	0.99000	8
19000	0.99100	7
20000	0.98800	8
	Sum	118

char_Set	26 / 150	
alphabetSize	128	
expediteFactor	1	
trieDepthFactor	8	
heuristicFactor	0.167	
observationFactor	1000	
Last Prediction	0.98400	
mean Prediction	0.94655	
Dataset	Prediction	Time
1000	0.70100	8
2000	0.85700	8
3000	0.90100	8
4000	0.93200	8
5000	0.94200	9
6000	0.95300	8
7000	0.95600	8
8000	0.96200	8
9000	0.96900	8
10000	0.97400	9
11000	0.96900	8
12000	0.97600	9
13000	0.97400	9
14000	0.97800	8
15000	0.98000	9
16000	0.97900	9
17000	0.98200	9
18000	0.98100	9
19000	0.98100	9
20000	0.98400	10
	Sum	171

char_Set		26 / 225
alphabetSize		128
expediteFactor		1
trieDepthFactor		8
heuristicFactor		0.167
observationFactor		1000
Last Prediction		0.97800
mean Prediction		0.92740
Dataset	Prediction	Time
1000	0.63900	9
2000	0.82300	9
3000	0.87400	10
4000	0.89800	9
5000	0.91600	10
6000	0.92900	10
7000	0.93600	10
8000	0.94300	10
9000	0.94700	11
10000	0.95300	10
11000	0.96000	10
12000	0.96400	11
13000	0.96800	10
14000	0.96600	12
15000	0.96700	13
16000	0.97000	12
17000	0.96900	13
18000	0.97600	13
19000	0.97200	13
20000	0.97800	14
Sum		219

LZ78 - Data DataSet

Prediction Vs Training Instances



Time Vs Training Instances



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,70900
expediteFactor	-	heuristicFactor	-			mean Prediction	0,67141

Active LeZi - Data DataSet

Prediction Vs Training Instances



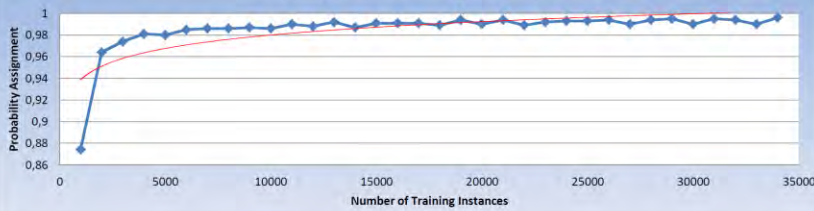
Time Vs Training Instances



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,70600
expediteFactor	-	heuristicFactor	-			mean Prediction	0,68015

Heuristic ALZ - Data DataSet

Prediction Vs Training Instances



Time Vs Training Instances



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,99600
expediteFactor	1	heuristicFactor	0,167			mean Prediction	0,95974

Dataset	Prediction	Time
1000	0,53500	13
2000	0,60000	13
3000	0,61100	15
4000	0,61900	14
5000	0,63800	14
6000	0,65200	14
7000	0,65000	15
8000	0,64800	14
9000	0,66000	14
10000	0,66600	14
11000	0,66700	14
12000	0,68200	15
13000	0,66300	14
14000	0,66800	14
15000	0,69200	14
16000	0,68700	15
17000	0,67800	14
18000	0,69400	15
19000	0,68900	14
20000	0,69200	14
21000	0,68700	15
22000	0,69900	14
23000	0,67600	15
24000	0,70700	15
25000	0,71400	15
26000	0,68100	15
27000	0,68200	15
28000	0,70100	15
29000	0,69300	14
30000	0,69800	15
31000	0,69600	15
32000	0,69900	15
33000	0,69500	15
34000	0,70900	15
Sum		491

Dataset	Prediction	Time
1000	0,57600	18
2000	0,59400	18
3000	0,67100	18
4000	0,64400	19
5000	0,65400	19
6000	0,68600	21
7000	0,67000	21
8000	0,65600	21
9000	0,67700	22
10000	0,68700	23
11000	0,67800	22
12000	0,68900	22
13000	0,66000	21
14000	0,67400	23
15000	0,70400	23
16000	0,68000	23
17000	0,69700	24
18000	0,70800	23
19000	0,69400	24
20000	0,68300	23
21000	0,69200	24
22000	0,68900	25
23000	0,68000	24
24000	0,71800	25
25000	0,70100	25
26000	0,68700	25
27000	0,67300	24
28000	0,70500	25
29000	0,70000	25
30000	0,69400	25
31000	0,70900	26
32000	0,70100	26
33000	0,68800	27
34000	0,70600	25
Sum		779

Dataset	Prediction	Time
1000	0,87400	1
2000	0,96400	1
3000	0,97400	2
4000	0,98100	2
5000	0,98000	2
6000	0,98500	2
7000	0,98600	2
8000	0,98600	3
9000	0,98700	2
10000	0,98600	3
11000	0,99000	2
12000	0,98800	3
13000	0,99200	3
14000	0,98700	2
15000	0,99100	3
16000	0,99100	3
17000	0,99100	3
18000	0,98900	4
19000	0,99400	3
20000	0,99000	3
21000	0,99400	4
22000	0,98900	3
23000	0,99200	3
24000	0,99300	4
25000	0,99300	4
26000	0,99400	3
27000	0,99000	4
28000	0,99400	4
29000	0,99500	4
30000	0,99000	4
31000	0,99500	4
32000	0,99400	4
33000	0,99000	4
34000	0,99600	4
Sum		102

LZ78 - Data80 DataSet



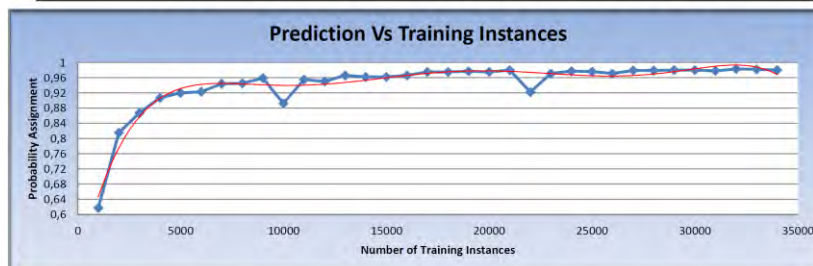
alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,45800
expediteFactor	-	heuristicFactor	-			mean Prediction	0,43765

Active LeZi - Data80 DataSet



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,40800
expediteFactor	-	heuristicFactor	-			mean Prediction	0,41032

Heuristic ALZ - Data80 DataSet



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,98000
expediteFactor	1	heuristicFactor	0,167			mean Prediction	0,94382

Dataset	Prediction	Time
1000	0,43500	32
2000	0,45100	32
3000	0,45900	50
4000	0,47000	79
5000	0,47500	69
6000	0,47800	63
7000	0,49200	71
8000	0,50100	76
9000	0,48500	78
10000	0,43500	78
11000	0,36600	124
12000	0,37200	131
13000	0,42000	188
14000	0,40000	237
15000	0,40800	273
16000	0,44100	235
17000	0,42400	202
18000	0,43900	108
19000	0,41200	62
20000	0,43100	72
21000	0,40300	284
22000	0,40200	288
23000	0,39700	387
24000	0,44100	440
25000	0,43400	505
26000	0,42300	533
27000	0,43700	564
28000	0,42400	580
29000	0,42000	581
30000	0,43600	585
31000	0,46400	504
32000	0,48200	525
33000	0,46500	543
34000	0,45800	545
Sum		9124

Dataset	Prediction	Time
1000	0,45800	31
2000	0,46400	31
3000	0,48000	47
4000	0,48100	43
5000	0,49100	43
6000	0,47100	42
7000	0,52600	40
8000	0,56400	38
9000	0,56500	39
10000	0,45500	97
11000	0,34100	216
12000	0,34500	218
13000	0,41100	218
14000	0,39100	218
15000	0,38500	218
16000	0,39700	218
17000	0,36500	219
18000	0,36300	217
19000	0,36600	219
20000	0,35800	219
21000	0,36600	219
22000	0,34700	262
23000	0,33600	296
24000	0,40200	295
25000	0,38900	297
26000	0,37200	297
27000	0,38300	297
28000	0,36600	297
29000	0,35100	298
30000	0,36400	297
31000	0,38500	298
32000	0,40200	297
33000	0,40300	295
34000	0,40800	296
Sum		6672

Dataset	Prediction	Time
1000	0,61700	3
2000	0,81500	3
3000	0,86700	4
4000	0,90700	4
5000	0,92000	4
6000	0,92300	4
7000	0,94400	5
8000	0,94500	5
9000	0,95900	5
10000	0,89200	6
11000	0,95500	9
12000	0,95000	8
13000	0,96600	8
14000	0,96200	9
15000	0,96200	9
16000	0,96600	9
17000	0,97500	9
18000	0,97500	10
19000	0,97700	9
20000	0,97500	10
21000	0,98000	10
22000	0,92200	12
23000	0,97100	13
24000	0,97700	13
25000	0,97600	14
26000	0,97100	13
27000	0,97900	14
28000	0,97900	14
29000	0,98000	15
30000	0,98000	14
31000	0,97800	15
32000	0,98300	15
33000	0,98200	16
34000	0,98000	15
Sum		326

LZ78 - a_1 DataSet

Prediction Vs Training Instances



Time Vs Training Instances



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,12400
expediteFactor	-	heuristicFactor	-			mean Prediction	0,10724

Active LeZi - a_1 DataSet

Prediction Vs Training Instances



Time Vs Training Instances



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,10000
expediteFactor	-	heuristicFactor	-			mean Prediction	0,09915

Heuristic ALZ - a_1 DataSet

Prediction Vs Training Instances



Time Vs Training Instances



alphabetSize	128	trieDepthFactor	8	observationFactor	1000	Last Prediction	0,50400
expediteFactor	1	heuristicFactor	0,167			mean Prediction	0,65371

Dataset	Prediction	Time
1000	0,09400	45
2000	0,07600	45
3000	0,11400	84
4000	0,10700	140
5000	0,10700	184
6000	0,11500	304
7000	0,10800	237
8000	0,10800	300
9000	0,11400	359
10000	0,12300	372
11000	0,12500	444
12000	0,12200	504
13000	0,11600	582
14000	0,10800	630
15000	0,11700	748
16000	0,12800	1144
17000	0,12200	708
18000	0,12400	713
19000	0,11100	847
20000	0,09500	906
21000	0,09500	973
22000	0,09800	1047
23000	0,09200	1080
24000	0,08900	1113
25000	0,10900	1492
26000	0,09000	2202
27000	0,07300	991
28000	0,08100	1001
29000	0,07800	951
30000	0,11200	1024
31000	0,09800	1174
32000	0,11900	1282
33000	0,15400	1326
34000	0,12400	1542
Sum		26494

Dataset	Prediction	Time
1000	0,08800	304
2000	0,08400	304
3000	0,12900	500
4000	0,10900	694
5000	0,11800	947
6000	0,12400	1136
7000	0,12700	1480
8000	0,12600	1771
9000	0,11200	1888
10000	0,12600	1718
11000	0,10900	1122
12000	0,10900	1224
13000	0,10000	1658
14000	0,10200	2352
15000	0,10600	2674
16000	0,12200	3127
17000	0,10900	2853
18000	0,11200	1928
19000	0,10500	1473
20000	0,08000	3453
21000	0,07300	4149
22000	0,07600	3457
23000	0,07600	3319
24000	0,06600	3479
25000	0,07300	3463
26000	0,06900	3613
27000	0,04800	3747
28000	0,06700	3852
29000	0,06200	3817
30000	0,09500	4067
31000	0,11900	1329
32000	0,11200	1474
33000	0,13800	2480
34000	0,10000	3601
Sum		78453

Dataset	Prediction	Time
1000	0,12700	23
2000	0,16400	23
3000	0,57800	35
4000	0,48900	44
5000	0,53900	56
6000	0,79800	58
7000	0,74600	61
8000	0,58500	69
9000	0,28800	86
10000	0,57300	99
11000	0,87100	101
12000	0,89100	104
13000	0,70400	111
14000	0,68800	122
15000	0,76700	127
16000	0,78600	127
17000	0,75200	130
18000	0,49800	142
19000	0,33800	162
20000	0,70200	182
21000	0,81700	189
22000	0,68200	200
23000	0,92700	203
24000	0,89300	205
25000	0,94800	211
26000	0,95300	215
27000	0,94500	218
28000	0,96200	220
29000	0,95700	221
30000	0,33000	223
31000	0,49800	224
32000	0,36200	250
33000	0,56400	264
34000	0,50400	293
Sum		4998

Β Παράρτημα

1.	To header αρχείο "parameters.h"	B-2
2.	To header αρχείο "parse.h"	B-3
3.	To αρχείο "parse.cpp"	B-4
4.	To header αρχείο "trie.h"	B-5
5.	To αρχείο "trie.cpp"	B-8
6.	To αρχείο "Heuristic ALZ.cpp"	B-20


```
1 #ifndef PARAMETERS_H
2 #define PARAMETERS_H
3
4 #define alphabetSize 128
5 #define expediteFactor 1
6 #define trieDepthFactor 8
7 #define heuristicFactor 0.167
8 #define observationFactor 1000
9
10 #define zeroLevelContexts true
11
12 #endif /* #ifndef PARAMETERS_H */
```

```
1 #include <string>
2
3 using namespace std;
4
5 class LZ78
6 {
7     string context;
8 public:
9     LZ78(): context("") {}
10    void setContext(string arg);
11    string getContext();
12    ~LZ78();
13 };
```

```
1 #include "parse.h"
2
3 string LZ78::getContext()
4 {
5     return context;
6 }
7
8 void LZ78::setContext(string arg)
9 {
10     context = arg;
11 }
12
13 LZ78::~~LZ78()
14 {
15     context = "";
16 }
```

```

1 #ifndef ALGORITHM_TRIE_H
2 #define ALGORITHM_TRIE_H
3
4 #include <string>
5 #include <vector>
6
7 #ifdef __cplusplus
8 extern "C" {
9 #endif
10
11
12
13     using namespace std;
14
15     typedef struct _Trie Trie;
16
17     typedef struct _TrieNode TrieNode;
18
19     /**
20      * Create a new trie.
21      *
22      * @return          Pointer to a new trie structure.
23      */
24
25     Trie *trie_new(void);
26
27     /**
28      * Destroy a trie.
29      *
30      * @param trie      The trie to destroy.
31      */
32
33     void trie_free(Trie *trie);
34
35     /**
36      * Insert a new key-value pair into a trie.
37      *
38      * @param trie      The trie.
39      * @param key       The new key.
40      */
41
42     void trie_insert(Trie *trie, char *key);
43
44     void LZ78_insert(Trie *trie, char *key);
45
46     /**
47      * Remove an entry from a trie.
48      *
49      * @param trie      The trie.
50      * @param key       The key of the entry to remove.
51      */
52
53     void trie_remove(Trie *trie, char *key);
54
55     /**
56      * Look up a value from its key in a trie.
57      *
58      * @param trie      The trie.
59      * @param key       The key.
60      * @return          The pointer associated with the key, or NULL if
61      *                  not found in the trie.
62      */
63
64     void *trie_lookup(Trie *trie, char *key);
65
66     /**
67      * Look up a value from its key in a level of a Trie.
68      *
69      * @param trie      The parent node of the level.
70      * @param key       The key.
71      * @return          The pointer associated with the key, or NULL if
72      *                  not found in the trie.
73      */

```

```

74 void *sub_lookup(TrieNode *trie, char *key);
75
76
77 /**
78  * Find the context occurrences that exist after a particular node.
79  *
80  * @param node          The node.
81  * @return              Count of the number of entries after the node.
82  */
83
84
85 int last_use_count(TrieNode *node);
86
87 /**
88  * Find the number of entries in a trie.
89  *
90  * @param trie          The trie.
91  * @return              Count of the number of entries in the trie.
92  */
93
94 void trie_num_entries(Trie *trie);
95
96 /**
97  * Calculate the escape ALZ_probabilities at each Trie level.
98  *
99  * @param trie          The trie.
100  * @param key           The current context window.
101  */
102
103 double* escape_probabilities(Trie *trie, char *key);
104
105 /**
106  * Find the occurrences of each context.
107  *
108  * @param trie          The trie.
109  * @param window        The current context window.
110  * @param key           The context we are looking for
111  */
112
113 double *find_occurrences(Trie *trie, char *lastContext, char *key);
114
115 /**
116  * If there is history after context key and has the same
117  * occurrences with it then key context must not be used
118  * because it doesn't exist in reality.
119  * Key is a part of a bigger context
120  */
121
122 TrieNode *appropriateContext(Trie *trie, char *context);
123
124 /**
125  * Find the probabilities of each char.
126  *
127  * @param trie          The trie.
128  * @param lastContext   The last context we are referencing to calculate the probabilities.
129  * @key                The context we are calculating the probabilities.
130  * @escape             The escape probabilities of each Trie level
131  * @Return             ALZ_probabilities which is an array with the blended probabilities
132  */
133
134 void calculate_probabilities(Trie *trie, char *lastContext, char *key, double *escape, double* & ALZ_probabilities);
135
136 /**
137  * Find the max element in an array.
138  *
139  * @param arg          The array.
140  * @Return             The index of the max element
141  */
142
143 int max(double *arg);
144
145 /**

```



```

146 * Predict the next symbol to come.
147 *
148 * @param probabilities    An array with the blended probabilities.
149 * @return                The predicted char.
150 */
151
152 char predict(Trie *trie, char * context, double* probabilities);
153
154 /**
155  * Show the data of a node of the trie.
156  *
157  * @param trie            The trie.
158  * @param key             The context we are looking for in the trie.
159  */
160
161 void showData(Trie *trie, char* key);
162
163 /**
164  * Generates a null-terminated sequence of characters (c-string)
165  * with the same content as the string object
166  *
167  * @param arg             The string object
168  * @return                Pointer to an internal array containing
169  *                        the c-string equivalent to the string content.
170  */
171
172 char *modifyInput(string arg);
173
174 /**
175  * Searches the trie to find all valid contexts
176  * that can be used in probability assignment
177  *
178  * @param phrase          A string object that is passed to the next trie level
179  * @param trie            The trie of the system
180  * @param nodePtr         A pointer to a specific node in the trie
181  * @param contextsTrie    The trie that stores the valid contexts of the system
182  * @param prob_vector     The vector that stores the valid contexts of the system
183  */
184
185 void updateContexts(string phrase, Trie *trie, TrieNode *nodePtr, Trie *contextsTrie, vector<string> &
prob_vector);
186
187 #ifdef __cplusplus
188 }
189 #endif
190
191 #endif /* #ifndef ALGORITHM_TRIE_H */

```

```

1  /* Trie: fast mapping of strings to values */
2
3  #include <cstdlib>
4  #include <iostream>
5  #include "parameters.h"
6  #include "trie.h"
7
8  using namespace std;
9
10 extern unsigned int Max_LZ_length;
11
12 extern unsigned int contexts;
13
14 struct _TrieNode
15 {
16     unsigned int use_count;
17     char symbol;
18     int lvl;
19     TrieNode *previous;
20     TrieNode *next[alphabetSize];
21 };
22
23 struct _Trie
24 {
25     TrieNode *root_node;
26 };
27
28 static void trie_free_node(TrieNode *node)
29 {
30     int i;
31
32     if (node == NULL)
33         return;
34
35     /* First, free all subnodes */
36
37     for (i=0; i < alphabetSize; ++i)
38     {
39         trie_free_node(node->next[i]);
40     }
41
42     /* Free this node */
43
44     delete node;
45 }
46
47 Trie *trie_new(void)
48 {
49     Trie *new_trie;
50
51     new_trie = new Trie;
52     new_trie->root_node = NULL;
53
54     return new_trie;
55 }
56
57 void trie_free(Trie *trie)
58 {
59     /**
60      * Free the subnode, and all others by implication
61      */
62
63     trie_free_node(trie->root_node);
64
65     /**
66      * Free the trie
67      */
68
69     delete trie;
70 }
71
72 void trie_insert(Trie *trie, char *key)
73 {

```

```

74     TrieNode **rover;
75     TrieNode *node;
76     TrieNode *temp = NULL;
77     char *p;
78     int c;
79     int counter = -2;
80
81     /**
82     * Search down the trie until we reach the end of string,
83     * creating nodes as necessary
84     */
85
86     rover = &trie->root_node;
87     p = key;
88
89     for (;;)
90     {
91
92         node = *rover;
93
94         if (node == NULL)
95         {
96             /**
97             /* Node does not exist, so create it
98             */
99
100            node = new TrieNode;
101            memset(node, 0, sizeof(TrieNode));
102
103            node->previous = temp;
104
105            /**
106            * Increase the level at this node
107            */
108
109            node->lvl++;
110            node->lvl += counter;
111
112            /**
113            * Link in to the trie
114            */
115
116            *rover = node;
117        }
118
119        /**
120        /* Current character
121        */
122
123        c = *p;
124
125        /**
126        * Reached the end of string? If so, we're finished.
127        */
128
129        if (c == '\0')
130        {
131            /**
132            /* One more use of this node
133            */
134
135            ++node->use_count;
136            node->symbol = 'X';
137
138            break;
139        }
140
141        temp = node;
142
143        /**
144        /* Advance to the next node in the chain
145        */
146

```

```

147     rover = &node->next[c];
148     ++p;
149     counter = node->lvl;
150 }
151 }
152
153
154 void LZ78_insert(Trie *trie, char *key)
155 {
156     TrieNode **rover;
157     TrieNode *node;
158     TrieNode *temp = NULL;
159     char *p;
160     int c;
161     int counter = -2;
162
163     /* Search down the trie until we reach the end of string,
164     * creating nodes as necessary */
165
166     rover = &trie->root_node;
167     p = key;
168     for (;;) {
169         node = *rover;
170         if (node == NULL) {
171             /* Node does not exist, so create it */
172             node = new TrieNode;
173             memset(node, 0, sizeof(TrieNode));
174             node->previous = temp;
175             /* Increase the level at this node */
176             node->lvl++;
177             node->lvl += counter;
178             /* Link in to the trie */
179             *rover = node;
180         }
181         /* One more use of this node */
182         ++node->use_count;
183         /* Current character */
184         c = *p;
185         /* Reached the end of string? If so, we're finished. */
186         if (c == '\0') {
187             break;
188         }
189         temp = node;
190         /* Advance to the next node in the chain */
191         rover = &node->next[c];
192         ++p;
193         counter = node->lvl;
194     }
195 }
196
197 void trie_remove(Trie *trie, char *key)
198 {

```



```

220     TrieNode *node;
221     TrieNode *next;
222     TrieNode **last_next_ptr;
223     char *p;
224     int c;
225
226     /**
227     * First, search down to the ending node
228     * so that the data can be removed.
229     */
230
231     /**
232     * Search down the trie until the end of string is reached
233     */
234
235     node = trie->root_node;
236
237     for (p=key; *p != '\0'; ++p)
238     {
239
240         if (node == NULL)
241         {
242             /**
243             * Not found in the tree. Return.
244             */
245
246             return;
247         }
248
249         /**
250         * Jump to the next node
251         */
252
253         c = *p;
254         node = node->next[c];
255     }
256
257     /**
258     * Now traverse the tree again as before,
259     * decrementing the use count of each node.
260     * Free back nodes as necessary.
261     */
262
263     node = trie->root_node;
264     last_next_ptr = &trie->root_node;
265     p = key;
266
267     for (;;)
268     {
269         /**
270         * Find the next node
271         */
272
273         c = *p;
274         next = node->next[c];
275
276         /**
277         * Free this node if necessary
278         */
279
280         if (node->use_count <= 0)
281         {
282             delete node;
283
284             /**
285             * Set the "next" pointer on the previous node to NULL,
286             * to unlink the free'd node from the tree.
287             * This only needs to be done once in a remove.
288             * After the first unlink, all further nodes are also
289             * going to be free'd.
290             */
291
292             if (last_next_ptr != NULL)

```

```

293         {
294             *last_next_ptr = NULL;
295             last_next_ptr = NULL;
296         }
297     }
298
299     /**
300     * Go to the next character or finish
301     */
302
303     if (c == '\0')
304     {
305         --node->use_count;
306         break;
307     }
308     else
309     {
310         ++p;
311     }
312
313     /**
314     * If necessary, save the location of the "next" pointer
315     * so that it may be set to NULL on the next iteration if
316     * the next node visited is freed.
317     */
318
319     if (last_next_ptr != NULL)
320     {
321         last_next_ptr = &node->next[c];
322     }
323
324     /**
325     * Jump to the next node
326     */
327
328     node = next;
329 }
330 }
331
332 void *trie_lookup(Trie *trie, char *key)
333 {
334     TrieNode *node;
335     char *p;
336     int c;
337
338     /**
339     * Search down the trie until the end of string is found
340     */
341
342     node = trie->root_node;
343     p = key;
344
345     while (*p != '\0')
346     {
347         if (node == NULL)
348         {
349             /**
350             * Not found - reached end of branch
351             */
352
353             return NULL;
354         }
355
356         /**
357         * Advance to the next node in the chain, next character
358         */
359
360         c = *p;
361         node = node->next[c];
362         ++p;
363     }
364
365     return node;

```

```

366 }
367
368 void *sub_lookup(TrieNode *trie, char *key)
369 {
370     TrieNode *node;
371     char *p;
372     int c;
373
374     /**
375      * Search down the trie until the end of string is found
376      */
377
378     node = trie->previous;
379     p = key;
380
381     while (*p != '\0')
382     {
383         if (node == NULL)
384         {
385             /**
386              * Not found - reached end of branch
387              */
388
389             return NULL;
390         }
391
392         /**
393          * Advance to the next node in the chain, next character
394          */
395
396         c = *p;
397         node = node->next[c];
398         ++p;
399     }
400
401     return node;
402 }
403
404 int last_use_count(TrieNode *node)
405 {
406     int counter = 0;
407     int temp = 0;
408
409     /**
410      * If there is no history after the node return
411      */
412     if (node->next == NULL)
413     {
414         /**
415          * No history after the node - reached end of branch
416          */
417
418         return 0;
419     }
420     else
421     {
422         /**
423          * Sum all the descendant instances of the node
424          */
425
426         while (counter < alphabetSize)
427         {
428             if (node->next[counter]) temp += node->next[counter]->use_count;
429             counter++;
430         }
431     }
432
433     return temp;
434 }
435
436 void trie_num_entries(Trie *trie)
437 {
438     /**

```

```

439     * To find the number of entries in the trie,
440     * simply look at the use count of the root node.
441     */
442
443     int counter = 0;
444     int temp = 0;
445
446     if (trie->root_node == NULL)
447     {
448         return;
449     }
450     else
451     {
452
453         temp = last_use_count(trie->root_node);
454
455         trie->root_node->use_count = temp;
456     }
457 }
458
459 double *escape_probabilities(Trie *trie, char *key)
460 {
461     TrieNode *tempPtrLower;
462     TrieNode *tempPtrHigher;
463     double *tempArray;
464     int expedite = 0;
465     int counter = 0;
466     int sum = 0;
467
468     tempArray = new double[Max_LZ_length];
469     memset(tempArray, 0, Max_LZ_length * sizeof(double));
470
471     /**
472     * Search down the trie until the end of string is found
473     */
474
475     tempPtrLower = (TrieNode*) trie_lookup(trie, key);
476
477     /**
478     * Calculate the escape Factor for every trie level or
479     * suspend calculations to a number of levels
480     */
481
482     while ((tempPtrLower->lvl > 0) && (expedite <= expediteFactor))
483     {
484         tempPtrHigher = tempPtrLower->previous;
485
486         sum = last_use_count(tempPtrHigher);
487
488         /**
489         * The escape Factor to a trie level equals to
490         * the node occurrences at that level
491         * minus node's descendant occurrences.
492         */
493
494         tempArray[tempPtrLower->lvl - 1] =
495             (tempPtrHigher->use_count - sum) / (double)tempPtrHigher->use_count;
496
497         /**
498         * Go one level up
499         */
500
501         tempPtrLower = tempPtrHigher;
502
503         sum = 0;
504         counter = 0;
505         expedite++;
506     }
507
508     return tempArray;
509 }
510
511 double *find_occurrences(Trie *trie, char *lastContext, char *key)

```



```

512 {
513     TrieNode *tempPtrLower;
514     TrieNode *tempPtrHigher;
515     TrieNode *node;
516     int subtract = 0;
517
518     double *tempArray = new double[Max_LZ_length];
519     memset(tempArray, 0, Max_LZ_length * sizeof(double));
520
521     int expedite = 0;
522
523     /**
524     * Search down the trie until the end of string is found
525     */
526
527     tempPtrLower = (TrieNode*) trie_lookup(trie, lastContext);
528
529     /**
530     * Calculate the occurrences of lastContext for every trie level or
531     * suspend calculations to a number of levels
532     */
533
534     while ((tempPtrLower->lvl > -1) && (expedite <= expediteFactor))
535     {
536         tempPtrHigher = tempPtrLower->previous;
537
538         /**
539         * The search should not be started from the root node
540         * but from the last node in lastContext sequence
541         */
542
543         node = (TrieNode*) sub_lookup(tempPtrLower, key);
544
545         if (node)
546         {
547             /**
548             * Calculate the occurrences of key context
549             * in the sub-trie we are searching
550             */
551
552             subtract = last_use_count(node);
553             tempArray[tempPtrLower->lvl] = (node->use_count - subtract) /
554                 (double) tempPtrHigher->use_count;
555
556             subtract = 0;
557         }
558
559         /**
560         * Go one level up
561         */
562
563         tempPtrLower = tempPtrHigher;
564         expedite++;
565     }
566
567     return tempArray;
568 }
569
570 bool checkContext(char *key)
571 {
572     char *p;
573     char c;
574
575     p = key;
576
577     while (*p != '\0')
578     {
579         c = *p;
580         ++p;
581
582         if (!(c == *p) && (*p != '\0')) return false;
583     }
584

```

```

585     return true;
586 }
587
588 TrieNode *appropriateContext(Trie *trie, char *context)
589 {
590     TrieNode *node;
591
592     /**
593      * Find key context in the trie
594      */
595
596     node = (TrieNode*) trie_lookup(trie, context);
597
598     /**
599      * If there is history after context key and has the same
600      * occurrences with it then key context must not be used
601      * because it doesn't exist in reality.
602      * Key is a part of a bigger context
603      */
604
605     if (!(node->use_count - last_use_count(node))) return NULL;
606
607     return node;
608 }
609
610 void calculate_probabilities
611 (Trie *trie, char *lastContext, char *key, double *escape, double* &probabilities)
612 {
613     char *p;
614
615     /**
616      * Given the context lastContext find the
617      * occurrences of key context in all
618      * levels of the trie
619      */
620
621     double *instances = find_occurrences(trie, lastContext, key);
622
623     int index = strlen(lastContext) - 1;
624
625     double sum = instances[index];
626
627     int counter = index - 1;
628
629     double tmp = 0;
630
631     /**
632      * Take in to account only the last levels
633      */
634
635     int expedite = 0;
636
637     /**
638      * Calculate the probability of occurrence
639      * of the context key in the trie
640      */
641
642     while ((counter > -1) && (expedite <= expediteFactor) )
643     {
644         /**
645          * For one more time probabilities are
646          * calculated for every trie level or are
647          * suspended to a number of levels
648          */
649
650         tmp = escape[index - 1];
651
652         for (int i = counter; i < (index - 1); i++)
653         {
654             tmp *= escape[i];
655         }
656
657         tmp *= instances[counter];

```

```

658         sum += tmp;
659
660         expedite++;
661         counter--;
662
663         tmp = 0;
664     }
665
666     delete []instances;
667
668     tmp = sum / (double) strlen(key);
669
670     p = key;
671
672     /**
673     * Distribute the probability of occurrence
674     * of key context equally
675     * to every character in it
676     */
677
678     while (*p != '\0')
679     {
680         index = *p;
681         probabilities[index] += tmp;
682         ++p;
683     }
684 }
685
686 int max(double *arg)
687 {
688     int counter = 0;
689     double temp = -1;
690     int index = -1;
691
692     while (counter < alphabetSize)
693     {
694         if (temp < arg[counter])
695         {
696             temp = arg[counter];
697             index = counter;
698         }
699
700         counter++;
701     }
702     return index;
703 }
704
705 char predict(Trie *trie, char *context, double *probabilities)
706 {
707     TrieNode *nodePtr;
708     double tmp = probabilities[0];
709     double check = 0;
710     int counter = 0;
711     unsigned int posistion_use_count = 0;
712     int index = -1;
713
714     /**
715     * Search down the trie until the end of string is found
716     */
717
718     nodePtr = (TrieNode*) trie_lookup(trie, context);
719
720     /**
721     * Repeat until you check all symbols
722     * in the alphabet
723     */
724
725     while (counter < alphabetSize)
726     {
727         if (nodePtr->next[counter])
728         {
729             if (posistion_use_count < nodePtr->next[counter]->use_count)
730                 {

```

```

731         check = (nodePtr->next[counter]->use_count - position_use_count) /
732                 (double) nodePtr->next[counter]->previous->use_count;
733
734         if (check < heuristicFactor)
735         {
736             if (tmp < probabilities[counter])
737             {
738                 tmp = probabilities[counter];
739                 position_use_count = nodePtr->next[counter]->use_count;
740                 index = counter;
741             }
742         }
743         else
744         {
745             tmp = probabilities[counter];
746             position_use_count = nodePtr->next[counter]->use_count;
747             index = counter;
748         }
749     }
750     else if (position_use_count >= nodePtr->next[counter]->use_count)
751     {
752         check = (position_use_count - nodePtr->next[counter]->use_count) /
753                 (double) nodePtr->next[counter]->previous->use_count;
754
755         if (check < heuristicFactor)
756         {
757             if (tmp < probabilities[counter])
758             {
759                 tmp = probabilities[counter];
760                 position_use_count = nodePtr->next[counter]->use_count;
761                 index = counter;
762             }
763         }
764     }
765 }
766
767     counter++;
768 }
769
770 if (index == -1)
771 {
772     /**
773     * There is no history after the context
774     * so the predicted symbol would be
775     * the one with the biggest probability
776     */
777
778     index = max(probabilities);
779 }
780
781 return char(index);
782 }
783 }
784
785 void showData(Trie *trie, char *key)
786 {
787     TrieNode *nodePtr;
788
789     nodePtr = (TrieNode*) trie_lookup(trie, key);
790
791     cout << endl << key << " " << nodePtr->use_count;
792 }
793
794 char *modifyInput(string arg)
795 {
796     int buffer = arg.size();
797     char *temp = new char[buffer + 1];
798
799     strcpy(temp, arg.c_str());
800
801     return temp;
802 }
803

```



```

804 void updateContexts(string phrase, Trie *trie, TrieNode *nodePtr, Trie *contextsTrie, vector<string> & prob_vector)
805 {
806     TrieNode *node;
807     TrieNode *checkNode;
808     TrieNode *check;
809     string data;
810     char *tmpContext;
811     int counter = 0;
812
813
814     data = phrase;
815
816     if (data.length() > trieDepthFactor) return;
817     if (!nodePtr) return;
818     while (counter < alphabetSize)
819     {
820         node = (TrieNode*) nodePtr->next[counter];
821
822         if (node)
823         {
824             data = data + char(counter);
825             tmpContext = modifyInput(data);
826
827             checkNode = (TrieNode*) trie_lookup(contextsTrie, tmpContext);
828
829             if (!checkNode)
830             {
831                 trie_insert(contextsTrie, tmpContext);
832                 prob_vector.push_back(data);
833             }
834             else if (checkNode && (checkNode->symbol != 'X'))
835             {
836                 if (!(checkNode->use_count == last_use_count(checkNode)))
837                 {
838                     trie_insert(contextsTrie, tmpContext);
839                     prob_vector.push_back(data);
840                 }
841             }
842             delete []tmpContext;
843             updateContexts(data, trie, node, contextsTrie, prob_vector);
844             data = phrase;
845         }
846         counter++;
847     }
848 }

```

```

1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 #include <vector>
5 #include <cstring>
6 #include <time.h>
7 #include "parameters.h"
8 #include "parse.h"
9 #include "trie.h"
10
11 using namespace std;
12
13 /**
14  * Builds a database to store all the contexts
15  * that are anticipated in the trie
16  */
17
18 vector<LZ78> ALZ_vector;
19
20 /**
21  * Builds a database to store all the contexts
22  * that are used in probability calculations
23  */
24
25 vector<string> prob_vector;
26
27 /**
28  * The active phrase that it is entered in the algorithm
29  */
30
31 string currentEntry = "";
32
33 /**
34  * Builds a trie to store all the contexts
35  * that the algorithm is parsing
36  */
37
38 Trie *ALZ_trie = trie_new();
39
40 /**
41  * Builds a trie to store all the contexts
42  * that the algorithm is using in probability assignment
43  */
44
45 Trie *contextsTrie;
46
47 extern unsigned int Max_LZ_length = 1;
48
49 /**
50  * Takes care that the size of the window is less or equal to MAX_LZ_length parameter
51  *
52  * @param window          The sliding window of the algorithm
53  * @param vector           The vector database that stores the contexts of the system
54  * @param trie             The trie.
55  * @param MAX_LZ_length    The longest phrase in the trie
56  */
57
58 void slidingWindow(string &window, vector<LZ78> &vector, Trie *trie, unsigned int Max_LZ_length);
59
60 /**
61  * Updates the trie with the contexts of the parced window
62  * the vector database is also updated
63  *
64  * @param window          The sliding window
65  * @param vector           The vector database that stores the contexts of the system
66  * @param trie             The trie.
67  */
68
69 void updateFrequencies(string window, vector<LZ78> &vector, Trie *trie);
70
71 int main () {
72     char c, str[256];
73     char prediction = ' ';

```

```

74 unsigned int localWindow = 0;
75 string window = "";
76 char *tmp;
77 char *tmpContext;
78 TrieNode *nodePtr;
79 string sourceContext;
80 unsigned int entry = 0, good = 0, current = 0;
81 bool update = false;
82 time_t seconds;
83
84 /**
85  * The array that stores the probability assignments
86  */
87
88 double *ALZ_probabilities = new double[alphabetSize];
89
90 /**
91  * Pointer to an array that stores the escape factor
92  * for every level of the trie
93  */
94
95 double *escape;
96
97 ifstream file;
98 ofstream outfile_prediction;
99 ofstream outfile_time;
100
101 cout << "Enter the name of an existing text file: ";
102 cin.get (str,256);
103
104 file.open (str);
105 outfile_prediction.open("prediction1.txt");
106 outfile_time.open("time1.txt");
107 LZ78 temp;
108
109 /**
110  * loop while extraction from file is possible or
111  * use the same loop for On Line prediction
112  */
113
114 while (file.good())
115 {
116     /**
117     * Defines whether or not only
118     * zero level contexts will be used
119     */
120
121     if (!zeroLevelContexts) contextsTrie = trie_new();
122
123     c = file.get();
124     if (file.good())
125     {
126         /**
127         * reset the system when a carriage return character is found
128         */
129
130         if (c == '\n')
131         {
132             currentEntry = "";
133             localWindow = 1;
134             continue;
135         }
136
137         /**
138         * statistics
139         */
140
141         entry++;
142         if (c == prediction) good++;
143         if (!(entry % observationFactor))
144         {
145             seconds = time(NULL);
146             current = good - current;

```

```

147         cout << good / (double) entry << " "
148             << current / (double) observationFactor
149             << " " << seconds << endl;
150
151         outfile_prediction << current / (double) observationFactor << endl;
152         outfile_time.flush() << seconds << endl;
153         current = good;
154     }
155
156     currentEntry += c;
157     tmp = modifyInput(currentEntry);
158
159     /**
160     * Look up tmp context in the trie
161     *
162     * if the context is found -> read the next symbol
163     * and update the length of the phrase that you have already found
164     *
165     * if the context is not found -> insert context in the trie
166     * and in contexts database if its length is less
167     * or equal to trieDepthFactor
168     */
169
170     nodePtr = (TrieNode*) trie_lookup(ALZ_trie, tmp);
171
172     if (nodePtr)
173     {
174         localWindow++;
175     }
176     else
177     {
178         if (currentEntry.size() <= trieDepthFactor)
179         {
180             temp.setContext(currentEntry);
181             ALZ_vector.push_back(temp);
182             temp.~LZ78();
183         }
184
185         update = true;
186
187         trie_insert(ALZ_trie, tmp);
188
189         Max_LZ_length = (localWindow > Max_LZ_length) ? localWindow : Max_LZ_length;
190
191         currentEntry = "";
192         localWindow = 1;
193     }
194
195     window += c;
196
197     slidingWindow(window,ALZ_vector, ALZ_trie, Max_LZ_length);
198
199     /**
200     * Remove the last entry in the trie when it is required
201     * to avoid double entries in the trie
202     */
203
204     if (update)
205     {
206         trie_remove(ALZ_trie, tmp);
207         update = false;
208     }
209
210     /**
211     * Use the active context in the system
212     * to calculate the escape factor for every
213     * Markov order model
214     */
215
216     escape = escape_probabilities(ALZ_trie, tmp);
217     memset(ALZ_probabilities, 0, alphabetSize * sizeof(double));
218
219     /**

```



```

220      * By using the appropriate contexts from the vector database
221      * calculate a probability estimate for every symbol
222      * in the dictionary of the system
223      */
224
225      /**
226      * Defines whether or not only
227      * zero level contexts will be used
228      */
229
230      if (!zeroLevelContexts)
231      {
232          for (unsigned int i = 0; i < ALZ_vector.size(); i++)
233          {
234              tmpContext = modifyInput(ALZ_vector[i].getContext());
235
236              if (appropriateContext(ALZ_trie, tmpContext))
237              {
238                  trie_insert(contextsTrie, tmpContext);
239                  prob_vector.push_back(ALZ_vector[i].getContext());
240              }
241
242              delete []tmpContext;
243          }
244
245          sourceContext = window;
246
247          sourceContext.erase(0, 1);
248
249          string data = "";
250
251          while (sourceContext.length() > 0)
252          {
253              tmpContext = modifyInput(sourceContext);
254
255              nodePtr = (TrieNode*) trie_lookup(ALZ_trie, tmpContext);
256              updateContexts(data, ALZ_trie, nodePtr, contextsTrie, prob_vector);
257              delete []tmpContext;
258              sourceContext.erase(0, 1);
259          }
260
261          for (unsigned int i = 0; i < prob_vector.size(); i++)
262          {
263              tmpContext = modifyInput(prob_vector[i]);
264              calculate_probabilities(ALZ_trie, tmp, tmpContext, escape, ALZ_probabilities);
265              delete []tmpContext;
266          }
267          prob_vector.clear();
268          trie_free(contextsTrie);
269      }
270      else
271      {
272          for (unsigned int i = 0; i < ALZ_vector.size(); i++)
273          {
274              tmpContext = modifyInput(ALZ_vector[i].getContext());
275              if (appropriateContext(ALZ_trie, tmpContext))
276              {
277                  calculate_probabilities(ALZ_trie, tmp, tmpContext, escape, ALZ_probabilities);
278              }
279              delete []tmpContext;
280          }
281      }
282  }
283
284      /**
285      * predict the next symbol to come
286      */
287
288      prediction = predict(ALZ_trie, tmp, ALZ_probabilities);
289
290      delete []tmp;
291      delete []escape;
292

```

```

293     /**
294     * All statistics would concern the amount of data
295     * that are specified here
296     */
297
298     if (entry == 34000)
299     {
300         outfile_prediction.close();
301         outfile_time.close();
302         file.close();
303         exit(0);
304     }
305 }
306 }
307
308 outfile_prediction.close();
309 outfile_time.close();
310 file.close();
311
312 delete []ALZ_probabilities;
313
314 entry = 0;
315
316 /**
317 * Print out all the contexts that are used
318 * to calculate the probability estimate for every symbol
319 * in the dictionary of the system
320 */
321
322 while (entry < ALZ_vector.size())
323 {
324     tmp = modifyInput(ALZ_vector[entry].getContext());
325     showData(ALZ_trie, tmp);
326     entry++;
327 }
328
329 ALZ_vector.clear();
330 trie_free(ALZ_trie);
331 return 0;
332 }
333
334 void updateFrequencies(string window, vector<LZ78> &vector, Trie *trie)
335 {
336     TrieNode *var;
337     char *tmp;
338     LZ78 temp;
339
340     while (window.size())
341     {
342         tmp = modifyInput(window);
343
344         var = (TrieNode*) trie_lookup(trie, tmp);
345
346         if (!var && (window.size() <= trieDepthFactor))
347         {
348             temp.setContext(tmp);
349             vector.push_back(temp);
350             temp.~LZ78();
351         }
352
353         trie_insert(ALZ_trie, tmp);
354
355         window.erase(0, 1);
356         delete []tmp;
357     }
358
359     trie_num_entries(ALZ_trie);
360 }
361
362 void slidingWindow(string &window, vector<LZ78> &vector, Trie *trie, unsigned int Max_LZ_length)
363 {
364     if (window.length() > Max_LZ_length) window.erase(0,1);
365

```

```
366     updateFrequencies(window, vector, trie);  
367 }
```