Εθνικό Μετσόβιο Πολυτεχνείο

Τμήμα Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών

Τομέας Μαθηματικών

Εξέλιξη και Τεχνητή Νοημοσύνη

Διπλωματική Εργασία

του

Λειβαδά Κωνσταντίνου

Επιβλέπων: Αρβανιτάκης Αλέξανδρος
Αναπληρωτής Καθηγητής Τομέα Μαθηματικών, ΕΜΠ.

Αθήνα, Ιούλιος 2023

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Εκφράσεις όπως τεχνητή νοημοσύνη, μηχανική μάθηση, νευρωνικά δίκτυα (AI, artificial intelligence, machine learning, neural networks) είναι από τις πλέον διαδεδομένες στην εποχή μας. Από αυτοκίνητα με αυτόματο πιλότο, μέχρι το πρόσφατο Chat GPT η τεχνητή νοημοσύνη έχει μπει για τα καλά στις ζωές μας. Προτού προχωρήσουμε στο θέμα αυτής της εργασίας, το οποίο όπως μαρτυράει και ο τίτλος είναι συναφές της τεχνητής νοημοσύνης, θα θέλαμε να εξηγήσουμε τι σημαίνει για εμάς τεχνητή νοημοσύνη αφού δεν υπάρχει κάποιος μονοσήμαντος ορισμός στον οποίο να συγκλίνει η επιστημονική κοινότητα.

Κοιτάζοντας στο όχι και πολύ μακρινό παρελθόν, περίπου στα μέσα του 18ου αιώνα, στην πρώτη Βιομηχανική Επανάσταση, παρατηρούμε την μετάβαση των Ευρωπαϊκών κοινωνιών από τις επαναλαμβανόμενες χειρωνακτικές διαδικασίες παραγωγής στις τυποποιημένες και αυτοματοποιημένες. Η αυτοματοποίηση αύξησε πολύ την παραγωγικότητα και με το πέρασμα του καιρού όλο και περισσότερες διαδικασίες τυποποιούνταν. Έτσι ενώ πλέον η πλειονότητα των φυσικών διαδικασιών είχε αυτοματοποιηθεί, όμως οι επαναλαμβανόμενες νοητικές διαδικασίες στην παραγωγή εξακολουθούσαν να υφίστανται.

Το 1950 περίπου ξεκινάει η Ψηφιακή Επανάσταση και η εποχή της Πληροφορίας, όπου με την χρήση των υπολογιστών αρχίζει η αυτοματοποίηση των νοητικών διαδικασιών. Αυτές πλέον εκτελούνται με ακρίβεια και ταχύτητα π.χ. αριθμητικές πράξεις. Σε αυτό το σημείο η ανθρωπότητα βρίσκει την λύση σε κάποιο πρόβλημα και στην συνέχεια τυποποιεί την λύση με κάποιον αλγόριθμο, τον οποίο στην συνέχεια εκτελεί κατά γράμμα μια μηχανή.

Για εμάς τεχνητή νοημοσύνη είναι το επόμενο βήμα αυτής της εξέλιξης, δηλαδή η αυτοματοποίηση της εύρεσης λύσης σε προβλήματα. Έτσι το μόνο που απαιτείται είναι η σαφής διατύπωση του προβλήματος και στην συνέχεια η μηχανή θα βρει τον τρόπο να το λύσει.
Όπως είναι ήδη γνωστό, χρειαζόμαστε έναν τρόπο με τον οποίο θα εκπαιδεύσουμε μια μηχανή ώστε να μπορεί να λύνει προβλήματα. Έτσι, για εμάς η πλήρης επίτευξη αυτού του στόχου έγκειται στην εύρεση ενός καθολικού τρόπου εκπαίδευσης, γιατί αλλιώς χάνεται η αυτοματοποίηση.

Ένας τέτοιος τρόπος εκπαίδευσης, πρέπει να είναι γενικός (general artificial intelligence), ανεξάρτητος από το πρόβλημα και προσαρμοστικός, δηλαδή να εξελίσσεται ανάλογα με τις ανάγκες του υπό εκπαίδευση συστήματος.

Έτσι ο βασικός στόχος αυτής της εργασίας είναι να υποδείξει ένα πιθανό μονοπάτι προς την επίτευξη ενός τέτοιου ευφυούς συστήματος. Η δυσκολία αυτού του εγχειρήματος έγκειται στην καθολικότητα και προσαρμοστικότητα που απαιτείται στον τρόπο εκπαίδευσης του συστήματος. Μια κύρια διαφορά του τρόπου μάθησης των ανθρώπων, σε σχέση με τον τρόπο μάθησης των μηχανών είναι ότι οι άνθρωποι μπορούν να εμβαθύνουν σε αυτόν και να τον προσαρμόσουν εντός του περιβάλλοντος εκμάθησης. Η ικανότητα αυτή των ανθρώπων, να αυτοαναφέρονται, έρχεται σε αντιδιαστολή με τον στατικό τρόπο εκπαίδευσης των σύγχρονων συστημάτων τεχνητής ευφυΐας. Εμπνευσμένοι από την ανθρώπινη προσέγγιση στη μάθηση, θεωρούμε ότι η αυτοαναφορά είναι απαραίτητο συστατικό ενός προσαρμοστικού, ευφυούς συστήματος.

Πηγαίνοντας ένα βήμα παραπέρα την παραπάνω σκέψη, ένα σύστημα που αυτοαναφέρεται μπορεί να γίνει εξαιρετικά περίπλοκο και δύσκολα αξιολογήσιμο. Επίσης είναι ιδιαίτερα σημαντικό να κρατάμε ένα ιστορικό των αποφάσεων που παίρνει το σύστημα ώστε να μπορεί να αξιολογεί και με βάση αυτό το ιστορικό. Κάτι αντίστοιχο κάνουν και οι άνθρωποι όταν επιλεγούν μια κατεύθυνση για την επίλυση ενός προβλήματος και σε περίπτωση παρατεταμένης αποτυχίας αλλάζουν πορεία και επιχειρούν κάτι διαφορετικό. Έτσι κάθε απόφαση υλοποείται από κάποια αυτοαναφορική μετάβαση και τελικά έχουμε ένα γενεαλογικό δέντρο αποφάσεων όπου το περιβάλλον ορίζει την επιτυχία. Ουσιαστικά το πάντρεμα της λογικής των γενετικών αλγορίθμων με την αυτοαναφορά είναι η οδός που υποδεικνύει αυτή η εργασία για την επίτευξη ενός προσαρμοστικού συστήματος.

Κεφάλαιο 2

Το χρονικό της τεχνητής νοημοσύνης

Δεν μας κάνει καθόλου εντύπωση το γεγονός, ότι η εξέλιξη της τεχνητής νοημοσύνης έγινε χέρι-χέρι με αυτήν της επιστήμης των υπολογιστών. Πολύ πριν την υλική σύλληψη του ηλεκτρονικού υπολογιστή, οι πρωτεργάτες του κλάδου (Alan Turing, John Von Neumann κ.α.) ήδη οραματιζόντουσαν ευφυή συστήματα.

Η κατ΄ εξοχήν μάλλον έναρξη του κλάδου ξεκίνησε με τον John McCarthy και την δημιουργία της γλώσσας Lisp με στόχο την έρευνα στην τεχνητή ευφυΐα. Ο McCarthy επικεντρώθηκε στο πρόβλημα του πως να φτιάξουμε ένα πρόγραμμα που γράφει προγράμματα δεδομένου ενός προβλήματος. Αυτή η αυτοαναφορική προσέγγιση διέπει όχι μόνο την Lisp αλλά και την έννοια του υπολογισμού και της τεχνητής ευφυίας, όπως θα δούμε στο επόμενο κεφάλαιο. Έτσι από το 1956 έως το 1974 ξεκινάει η συμβολική προσέγγιση στον κλάδο της τεχνητής ευφυίας.

Τα αποτελέσματα στην αρχή της περιόδου ήταν εξωπραγματικά. Οι υπολογιστές έλυναν απλά προβλήματα άλγεβρας, διατυπωμένα σε φυσική γλώσσα, αποδείκνυαν θεωρήματα και «μιλούσαν». Αυτά τα αποτελέσματα δημιούργησαν μεγάλες προσδοκίες και έτσι υπήρχε έντονη χρηματοδότηση και ερευνητική δραστηριότητα. Αξιοσημείωτα αποτελέσματα της περιόδου αποτελούν ο λογικός προγραμματισμός και οι ευρετικοί αλγόριθμοι. Αυτές οι τεχνικές αξιοποιήθηκαν για την προσπάθεια λύσης προβλημάτων όπως η αναπαράσταση γνώσης και η επεξεργασία φυσικών γλωσσών. Ενδιαφέρον αποτελεί το γεγονός ότι από τις απαρχές του κλάδου η ερευνητική κοινότητα στράφηκε σε δύσκολα και γενικά ζητήματα της τεχνητής ευφυίας. Ενδεικτικά αναφέρουμε ότι η έρευνα στην όραση υπολογιστών που είχε ξεκινήσει με την χρήση νευρωνικών δικτύων πάγωσε για 10 έτη πράγμα που με βάση τα σημερινά δεδομένα φαντάζει ειρωνικό, αν όχι αλαζονικό. Αυτή η κατά μέτωπον επίθεση στον πυρήνα του προβλήματος, η οποία τροφοδοτούνταν από την υπεραισιοδοξία που επικρατούσε καθώς και την υποτίμηση της υποβόσκουσας δυσκολίας γρήγορα οδήγησε σε τέλμα, γνωστό με τον όρο First AI Winter.

Την δεκαετία του 70΄ ο κλάδος ήρθε αντιμέτωπος με πολλά προβλήματα. Η ανεπαρκής υπολογιστική ισχύς και ο εκθετικός χρόνος καθιστούσαν πρακτικά αδύνατη την επίλυση ακόμη και πολύ απλών προβλημάτων. Όπως ένα μικρό παιδί χρειάζεται πληθώρα πληροφοριών για να

αντιληφθεί τον κόσμο, να χτίσει μια εσωτερική αναπαράσταση της πραγματικότητας, έτσι και ένα πρόγραμμα πρέπει να έχει αυτές τις πληροφορίες και να μπορεί να τις επεξεργαστεί ακόμη και για την επίλυση απλών προβλημάτων, πράγμα που για την δεκαετία του 70′ ήταν αδύνατο. Παρά τα παραπάνω προβλήματα, οι ερευνητές ήρθαν αντιμέτωποι με πιο ουσιώδη ζητήματα. Τα μαθηματικά προβλήματα είναι εξαιρετικά πιο προσιτά σε έναν υπολογιστή εξαιτίας της αυστηρής διατύπωσης τους, σε αντίθεση για παράδειγμα με την αναγνώριση ενός προσώπου, το οποίο είναι κάτι πολύ εύκολο για έναν άνθρωπο αλλά εξαιρετικά δύσκολο για ένα τυπικό σύστημα διότι δεν μπορεί να εκφραστεί αυστηρά. Αυτές οι εγγενείς δυσκολίες, οι οποίες ερευνώνται από διάφορους επιστημονικούς κλάδους όπως η ψυχολογία και η νευροεπιστήμη, δημιούργησαν ένα κύμα κριτικών και αμφισβητήσεων ως προς το εγχείρημα της τεχνητής ευφυίας.

Παρά τα διάφορα προβλήματα, στις αρχές της δεκαετίας του 80′ εμφανίζονται τα «έμπειρα συστήματα» (expert systems).Αυτά είναι προγράμματα που στοχεύουν στην λήψη αποφάσεων. Με την βοήθεια λογικών κανόνων συναγωγής και πολλών if-then-else βρόγχων μιμούνται την συλλογιστική εξειδικευμένων ανθρώπων πάνω σε κάποιο κλάδο. Πιο συγκεκριμένα, ένα έμπειρο σύστημα δομείται σε δύο μέρη: ένα σταθερό (αμετάβλητο), την βάση γνώσης στο οποίο αναπαρίσταται η γνώση και ένα μεταβλητό μέρος το οποίο με χρήση τυπικών λογικών κανόνων εξάγει συμπεράσματα και το οποίο με το πέρασμα του χρόνου αλλάζει. Αργότερα προστέθηκε και ένα τρίτο μέρος το οποίο ασχολούνταν με την διεπαφή (interface) με τον χρήστη. Τα έμπειρα συστήματα χαρακτηρίζονταν από αξιοπιστία και συνέπεια, το οποίο μάλλον οφείλονταν στην αυστηρή τους δομή. Ένα ιδιαίτερα ενδιαφέρον πλεονέκτημα των έμπειρων συστημάτων ήταν ότι δεν «υπέφεραν» από το πρόβλημα της συνδυαστικής έκρηξης (combinatorial explosion) το οποίο θα περίμενε κανείς. Ο λόγος είναι ότι η μηχανή όταν έβλεπε ότι πρόκειται για κάποιο μεγάλο συνδυασμό άρα χρονοβόρα διαδικασία, φόρτωνε αυτόματα κάποιους συνδυασμούς από τους κανόνες. Από την άλλη μεριά, τα έμπειρα συστήματα είχαν και μειονεκτήματα με μεγαλύτερο αυτό της αναπαράστασης της γνώσης με αυστηρούς κανόνες. Αυτό το ζήτημα το οποίο είχε παρουσιαστεί από την δεκαετία του 70′ δεν κατάφερε να επιλυθεί καθώς η «μη αυστηρή» αναπαράσταση γνώσης είναι ιδιαίτερα δύσκολη σε ένα τυπικό σύστημα. Παρατηρούμε, μια μεταστροφή της κοινότητας από το γενικό πρόβλημα της τεχνητής ευφυίας σε πιο οριοθετημένα ζητήματα (domain specific). Ένα άλλο χαρακτηριστικό της δεκαετίας του 80′ είναι η ανάσταση των νευρωνικών δικτύων, σε πρώιμη βέβαια ακόμη μορφή. Επίσης υπήρξε μια στροφή προς τον όγκο της πληροφορίας που χρειάζεται ένα σύστημα. Η πρόοδος της τεχνολογίας έφερε την δυνατότητα επεξεργασίας περισσότερων δεδομένων και έτσι η κοινότητα στράφηκε σε μεγάλες βάσεις δεδομένων. Έτσι ξεκίνησε το εγχείρημα Cyc.Σε αυτό η προσοχή στράφηκε στην δημιουργία ενός συστήματος αναπαράστασης γνώσης το οποίο μπορεί να αποθηκεύει και να οργανώνει πληροφορίες σχετικά με τον κόσμο με ένα δομημένο και λογικό τρόπο. Αυτή η βάση περιέχει πληροφορίες για ένα μεγάλο εύρος θεμάτων, όπως η γλώσσα, τα μαθηματικά, οι επιστήμες αλλά και γενικότερα, γνώση που χρειαζόμαστε για να κατανοήσουμε τον κόσμο γύρω μας. Στόχος είναι η δημιουργία ενός συστήματος τεχνητής ευφυίας το οποίο μπορεί να αναλύσει και να κατανοήσει κείμενο γραμμένο σε φυσική γλώσσα ώστε να αποκτήσει μια αντίληψη του κόσμου. Επιπλέον έχουν δημιουργηθεί διάφορες μηχανές που επιτρέπουν στο σύστημα την εξαγωγή συμπερασμάτων και τη δημιουργία προβλέψεων με βάση τις πληροφορίες που έχει στην γνωσιακή του βάση. Αυτές οι μηχανές περιλαμβάνουν, μία μηχανή κοινής γνώσης, μία επαγωγική μηχανή και μία παραγωγική. Ουσιαστικά το εγχείρημα Cyc θέλει να δημιουργήσει

ένα ευφυές σύστημα που αντιλαμβάνεται τον κόσμο όπως οι άνθρωποι και να το χρησιμοποιήσει για να λύσει διάφορα προβλήματα. Παρόλο που το πρότζεκτ έκανε σημαντική πρόοδο ειδικά στην αρχή του, το ΄84, αλλά και γενικά στο πέρασμα των χρόνων, είναι ακόμα μακριά από την επίτευξη των στόχων του. Είναι ιδιαίτερα σημαντικό, να τονίσουμε ότι κατά τη δεκαετία του ΄80, μπήκαν τα θεμέλια της σύγχρονης προσέγγισης του κλάδου της τεχνητής νοημοσύνης. Τεχνικές της μηχανικής μάθησης όπως τα νευρωνικά δίκτυα, τα δέντρα αποφάσεων και οι γενετικοί αλγόριθμοι αναπτύχθηκαν και εφαρμόστηκαν σε πληθώρα προβλημάτων όπως η αναγνώριση ομιλίας, η επεξεργασία εικόνων και η επεξεργασία φυσικών γλωσσών. Επίσης δημιουργήθηκε ο αλγόριθμος πίσω διάδοσης (back propagation)ο οποίος έπαιξε καθοριστικό ρόλο στην μετέπειτα ανάπτυξη της βαθιάς μάθησης (deep learning).

Από τη δεκαετία του ΄90 και μετά αδιαμφισβήτητα παρουσιάζεται μια τεράστια άνθηση στον κλάδο της τεχνητής νοημοσύνης. Τα επιτεύγματα των προηγούμενων ετών οδήγησαν στην εξέλιξη του κλάδου όχι μόνο στο θεωρητικό και ερευνητικό επίπεδο, αλλά και στο χώρο των εφαρμογών όπως θα δούμε. Αρχικά, η ευρεία εφαρμογή του αλγορίθμου πίσω διάδοσης ως μέθοδος εκπαίδευσης πολυεπίπεδων νευρωνικών δικτύων, επέτρεψε σε αυτά να μαθαίνουν και να αναγνωρίζουν περίπλοκα μοτίβα και πρότυπα σε μεγάλα και πολυδιάστατα σύνολα δεδομένων. Παρά το γεγονός ότι ο αλγόριθμος δε μεταβάλλει την αρχιτεκτονική του νευρωνικού δικτύου, η σαφήνεια και η απλότητα που τον διέπουν τον καθιστούν ακόμη και σήμερα ένα από τα ισχυρότερα εργαλεία της βαθιάς μάθησης.

Στον κλάδο της αναγνώρισης βίντεο και εικόνων, εμφανίστηκαν τα συνελικτικά νευρωνικά δίκτυα (CNN, convolutional neural network).Αυτά, με αυτόματο και προσαρμοστικό τρόπο, μαθαίνουν χωρικές ιεραρχίες των χαρακτηριστικών των δεδομένων εισόδου. Πιο συγκεκριμένα, αν υποθέσουμε ότι έχουμε μια εικόνα, δηλαδή έναν πίνακα με τιμές για κάθε pixel τότε αυτός ο πίνακας αποτελεί τα χαρακτηριστικά της εικόνας. Η έκφραση «χωρικές ιεραρχίες» σημαίνει ότι το μοντέλο αρχίζει να αναγνωρίζει πιο περίπλοκες δομές, ιεραρχίες, (υψηλότερου επιπέδου χαρακτηριστικά) όπως ευθείες, καμπύλες κ.α. και να τα εντοπίζει σε διάφορα σημεία της εικόνας, χωρικά. Οι βασικοί δομικοί λίθοι ενός συνελικτικού νευρωνικού δικτύου είναι τα συνελικτικά επίπεδα (convolutional layers) τα οποία χρησιμοποιούν ένα σύνολο φίλτρων για να σαρώσουν τα δεδομένα εισόδου και να εξάγουν τοπικά χαρακτηριστικά. Ύστερα ο καινούργιος χάρτης χαρακτηριστικών περνάει στο επόμενο συνελικτικό επίπεδο ώστε να εξαχθούν ακόμα συνθετότερα χαρακτηριστικά μέχρι να επιτευχθεί το απαραίτητο επίπεδο αφαίρεσης. Τα συνελικτικά νευρωνικά δίκτυα έχουν ένα μεγάλο εύρος εφαρμογών όπως η ταξινόμηση εικόνων (Image Classification)πχ αναγνώριση αντικειμένων, προσώπων, τοπίων κλπ. Επίσης χρησιμοποιούνται στον εντοπισμό αντικειμένων (Object Detection), μπορεί κανείς να εντοπίσει χωρικά που βρίσκεται ένα αντικείμενο με εφαρμογές στην αυτόνομη οδήγηση (self-driving cars), σε συστήματα ασφαλείας κ.α. Μια άλλη χρήση είναι στην σημασιολογική κατάτμηση (semantic segmentation)με εφαρμογές στην ιατρική απεικόνιση και στα αυτόνομα συστήματα (πχ ρομποτική). Τέλος, υπάρχει ακόμη και εφαρμογή στο χώρο των τεχνών όπου ένα συνελικτικό νευρωνικό δίκτυο μπορεί να χρησιμοποιηθεί για να μάθει ένα καλλιτεχνικό ρεύμα (πχ στη ζωγραφική) και στην συνέχεια να τροποποιήσει την είσοδό του υπό το πρίσμα του ρεύματος που έμαθε.

Ένας διαφορετικός τύπος νευρωνικού δικτύου που έκανε την εμφάνισή του τη δεκαετία του ′90 είναι τα αναδρομικά νευρωνικά δίκτυα (recursive/recurrent neural network) τα οποία έχουν τη δυνατότητα να αναλύουν σειριακά δεδομένα. Σε αντίθεση με τα παραδοσιακά πρόσθιας τροφοδότησης νευρωνικά δίκτυα, τα αναδρομικά έχουν λούπες στην αρχιτεκτονική τους που τους επιτρέπει να διατηρούν μια κατάσταση ή μνήμη των προηγούμενων δεδομένων καθώς επεξεργάζονται νέα δεδομένα. Αυτό το γεγονός τα καθιστά ιδιαίτερα βολικά για εργασίες στις οποίες το γενικό πλαίσιο είναι σημαντικό ή εργασίες στις οποίες το σύστημα θέλουμε να καταλάβει ένα δυναμικό μοτίβο. Τέτοιες εργασίες είναι, η επεξεργασία φυσικών γλωσσών όπου πρέπει να υπάρχει μια εσωτερική αναπαράσταση του νοήματος, οι χρονοσειρές όπου πρέπει να έχουμε μνήμη της προηγούμενης κατάστασης κ.α. Υπάρχουν διάφορα είδη αναδρομικών δικτύων όπως τα απλά, τα LSTM (Long Short-Term Memory)και τα GRU (Gated Recurrent Units).Ένα ενδιαφέρον χαρακτηριστικό των αναδρομικών νευρωνικών δικτύων είναι ότι είναι Turing Complete που σημαίνει ότι θεωρητικά μπορούν να υλοποιήσουν οποιαδήποτε υπολογίσιμη συνάρτηση, δεν έχει όμως ακόμη βρεθεί μια πρακτική αναπαράσταση για την υλοποίηση κλασσικών αλγορίθμων σε αυτά.

Τέλος, μέσα στην δεκαετία του ′90, έκαναν την εμφάνισή τους τα SVMs (Support Vector Machines)ή αλλιώς μηχανές διανυσμάτων υποστήριξης τα οποία έχουν εξαιρετική απόδοση σε προβλήματα ταξινόμησης όπου τα δεδομένα είναι πολλών διαστάσεων και δεν υπάρχει γραμμική εξάρτηση των χαρακτηριστικών τους.

Γενικότερα η δεκαετία του ′90 ήταν εξαιρετικά εύφορη για τον κλάδο της τεχνητής νοημοσύνης, ειδικά για τον υποκλάδο της μηχανικής μάθησης, και εγκαθίδρυσε τα θεμέλια για τη μετέπειτα έρευνα. Δεν θα ασχοληθούμε με τις επόμενες δεκαετίες καθώς σε αυτές δεν υπήρξε κάποια διαφορά στον τρόπο προσέγγισης του κλάδου όμως απογειώθηκε η εφαρμογή της τεχνητής ευφυίας σε πληθώρα προβλημάτων τα οποία ξεπέρασαν τον ακαδημαϊκό και επιστημονικό χώρο και πλέον έχουν μπει στην καθημερινή μας ζωή και γενικά στις σύγχρονες κοινωνίες.

Κεφάλαιο 3

Γενετικοί Αλγόριθμοι και Αυτοαναφορά

3.1 Γενική περιγραφή του γενετικού αλγορίθμου

Όπως έχουμε αναφέρει, αντιλαμβανόμαστε το μοντέλο που θέλουμε να παρουσιάσουμε ως ένα συνδυασμό της ιδέας του γενετικού αλγορίθμου και του φαινόμενου της αυτοαναφοράς.
Ο γενετικός αλγόριθμος ανήκει στην κλάση των εξελικτικών αλγορίθμων οι οποίοι είναι βιολογικά εμπνευσμένοι μεταευρετικοί αλγόριθμοι βελτιστοποίησης.

Ως αλγόριθμοι βελτιστοποίησης, ο βασικός σκοπός που εξυπηρετούν είναι η εύρεση θέσης ακρότατων συναρτήσεων. Αυτή η συνάρτηση, στην κλάση των εξελικτικών αλγορίθμων, καλείται συνάρτηση αξιολόγησης (fitness function). Οι συναρτήσεις αξιολόγησης, συνήθως είναι απεικονίσεις από τον χώρο των λύσεων σε κάποιο ολικά διατεταγμένο σύνολο, οι οποίες ουσιαστικά επάγουν την διάταξη του συνόλου τιμών στο χώρο λύσεων. Συμβολικά έχουμε, ότι αν X είναι ο χώρος των λύσεων, και $(A, \leq)$ ένα ολικά διατεταγμένο σύνολο, τότε μια απεικόνιση $f : X \longrightarrow A$ είναι συνάρτηση αξιολόγησης και επάγει την διάταξη $\leq_f$ στον χώρο λύσεων ως εξής, για κάθε $x, y \in X \quad x \leq_f y \Longleftrightarrow f(x) \leq f(y)$.
Εκτός από τον προφανή περιορισμό ότι η f πρέπει να είναι μια υπολογίσιμη συνάρτηση εφόσον είμαστε σε ένα αλγοριθμικό πλαίσιο, παρατηρούμε ότι δεν έχει κανέναν άλλο περιορισμό (π.χ. συνέχειας, ομαλότητας). Αυτό το χαρακτηριστικό καθιστά τους εξελικτικούς αλγορίθμους, πολύ γενικούς, αφού μπορούν να εφαρμοστούν σε προβλήματα στα οποία άλλες μέθοδοι δεν μπορούν να εφαρμοστούν εξαιτίας διαφόρων περιορισμών. Ένα βολικό χαρακτηριστικό μιας συνάρτησης αξιολόγησης είναι ο εύκολος και γρήγορος υπολογισμός της, διότι, όπως θα δούμε παρακάτω, θα χρειαστεί να υπολογιστεί πολλές φορές.

Ο λόγος που ο γενετικός αλγόριθμος ανήκει στους μεταευρετικούς αλγορίθμους είναι ότι περιγράφει μια υψηλότερου επιπέδου διαδικασία, η οποία παράγει μια ευρετική για την εξερεύνηση του χώρου λύσεων. Σε αντίθεση με τις κλασσικές επαναληπτικές μεθόδους, η εξερεύνηση του χώρου των λύσεων γίνεται με μη-ντετερμινιστικό τρόπου στον οποίο οφείλονται ο ολικός χαρακτήρας του αλγορίθμου (εξερευνεί μεγάλο μέρος του χώρου λύσεων) και η έλλειψη εγγύησης εύρεσης ολικού ακρότατου. Βασικά χαρακτηριστικά των μεταευρετικών αλγορίθμων είναι το μεγάλο εύρος προβλημάτων στα οποία μπορούν να εφαρμοστούν εξαιτίας του γενικού τους

χαρακτήρα και η αποδοτική εύρεση λύσης σχετικά κοντά στη βέλτιστη.

Στην κλασσική προσέγγιση του γενετικού αλγορίθμου, αρχικοποιούμε ένα τυχαίο πληθυσμό από λύσεις τον οποίο σταδιακά βελτιώνουμε. Η βελτίωση του πληθυσμού επυτηγχάνεται με τη διαδοχική εφαρμογή των τριών βασικών γενετικών τελεστών, της επιλογής, της διασταύρωσης και της μετάλλαξης. Να σημειώσουμε ότι υπάρχουν πάρα πολλές παραλλαγές, απλά εδώ θα παρουσιάσουμε την βασική και την σχέση της με την δική μας εκδοχή. Για την υλοποίηση ενός γενετικού αλγορίθμου στην απλούστερη μορφή πρέπει να παρθούν κάποιες σχεδιαστικές αποφάσεις. Αρχικά πρέπει να επιλεγεί μια κατάλληλη αναπαράσταση των λύσεων. Η αναπαράσταση πρέπει να είναι εύχρηστη ώστε να μπορούμε να επεξεργαζόμαστε τις λύσεις και φυσικά ένα προς ένα ώστε σε κάθε λύση να αντιστοιχίζεται μία και μοναδική αναπαράσταση. Για αυτούς τους λόγους συνήθως χρησιμοποιούνται οι λίστες για την αναπαράσταση των λύσεων, είτε ως λίστες αριθμών, είτε ως λίστες συμβόλων. Εν συνεχεία πρέπει να βρεθεί ένας τρόπος δημιουργίας ενός αρχικού πληθυσμού. Εδώ το σημαντικό είναι να δημιουργήσουμε έναν πληθυσμό ο οποίος να καλύπτει όσο το δυνατόν περισσότερες περιοχές του χώρου των λύσεων. Το επόμενο βήμα είναι η επιλογή μιας συνάρτησης αξιολόγησης. Στόχος είναι η συνάρτηση αυτή να είναι αντικειμενική, δηλαδή να αντικατοπτρίζει όσο είναι δυνατόν την πραγματική απόδοση των λύσεων.

Αφού επιλεγούν και υλοποιηθούν τα παραπάνω, ακολουθεί η βάση του γενετικού αλγορίθμου η οποία είναι η επιλογή και η υλοποίηση των γενετικών τελεστών. Πρώτος είναι ο τελεστής της επιλογής. Από αυτόν τον τελεστή ονομάστηκε ο γενετικός αλγόριθμος αφού εδώ προσομοιώνουμε την δαρβινική επιλογή. Αυτός ο τελεστής συνήθως επιλέγεται να είναι ελιτίστικος, δηλαδή να επιλέγει είτε ντετερμινιστικά είτε μη-ντετερμινιστικά τις καλύτερες λύσεις ως προς την επαγόμενη από την συνάρτηση αξιολόγησης διάταξη. Ένα παράδειγμα ενός ντετερμινιστικού, τελεστή επιλογής είναι αυτός που επιλέγει το 50% του πληθυσμού με την καλύτερη επίδοση. Ένα παράδειγμα ενός μη-ντετερμινιστικού τελεστή επιλογής είναι το εξής:

Έστω f μια συνάρτηση αξιολόγησης και P ο αρχικός πληθυσμός των λύσεων. Τότε η πιθανότητα επιλογής της λύσης $x \in P$ είναι

$$p_x = \frac{f(x)}{\sum_{y \in P} f(y)}.$$

Αυτός ο τελεστής επιλογής είναι πολύ διαδεδομένος και είναι γνωστός με το όνομα, μέθοδος της εξαναγκασμένης ρουλέτας.

Ο επόμενος τελεστής προς υλοποίηση είναι αυτός της διασταύρωσης (recombination). Εδώ προσομοιώνουμε την αναπαραγωγή, δηλαδή την ανταλλαγή γενετικού υλικού μεταξύ δύο λύσεων-γονέων. Η υλοποίηση αυτού του τελεστή είναι εξαιρετικά ρευστή και εξαρτάται από διάφορους παράγοντες, κυρίως την αναπαράσταση των λύσεων που έχει επιλεχθεί αλλά και την

φύση του προς επίλυση προβλήματος. Συνήθως αυτός ο τελεστής περιέχει μια δόση ντετερμινισμού αλλά και μια δόση τυχαιότητας. Πρέπει να προσομοιωθεί το φαινόμενο της αναπαραγωγής και συγκεκριμένα θέλουμε οι λύσεις-τέκνα να έχουν ποικιλία (τυχαιότητα) αλλά ταυτόχρονα να υπάρχουν έντονα τα χαρακτηριστικά των γονέων τους (ντετερμινισμός). Μια δημοφιλής μέθοδος υλοποίησης του τελεστή της διασταύρωσης είναι η διασταύρωση ενός σημείου,
όπου αν

$$(a_1, a_2, \cdots, a_m)$$

$$(b_1, b_2, \cdots, b_m)$$

είναι οι δύο λύσεις-γονείς. Επιλέγουμε τυχαία μια θέση $r \in \{2, \cdots, m-1\}$ και παίρνουμε τους απογόνους

$$(a_1, a_2, \cdots, a_r, b_{r+1}, \cdots, b_m)$$

$$(b_1, b_2, \cdots, b_r, a_{r+1}, \cdots, a_m)$$

οι οποίοι ανάλογα με το μοντέλο του γενετικού αλγορίθμου που έχουμε επιλέξει μπορεί να προστεθούν στον πληθυσμό ή να αντικαταστήσουν τους γονείς τους.

Σειρά έχει ο τελεστής της μετάλλαξης. Στόχο έχει να διατηρήσει την γενετική ποικιλία του πληθυσμού των λύσεων. Ουσιαστικά τροφοδοτεί τον πληθυσμό των δυνατών λύσεων που εξελίσσεται με νέες λύσεις (δηλαδή καινούργια σημεία του χώρου λύσεων), προστατεύοντας έτσι τον πληθυσμό από τον κίνδυνο να κυριευθεί από λίγες καλές λύσεις και βοηθώντας στη διασφάλιση της ισορροπίας ανάμεσα στην εξερεύνηση του χώρου των λύσεων και στην εκμετάλλευση των καλύτερων από αυτές. Το ζήτημα της ισορροπίας ανάμεσα στην εξερεύνηση του χώρου των λύσεων και στην εκμετάλλευση των καλύτερων από αυτές είναι ιδιαίτερα σημαντικό στον κλάδο της τεχνητής νοημοσύνης και είναι γνωστό με το όνομα exploration vs exploitation tradeoff. Πρακτικά πρόκειται για το δίλημμα μεταξύ της επιλογής κάποιας γνωστής επιτυχημένης στρατηγικής (exploitation) και της επιλογής να πειραματιστούμε με κάποια καινούργια στρατηγική με απώτερο σκοπό να ανακαλύψουμε κάτι καλύτερο. Γενικά θα λέγαμε ότι ο τελεστής της μετάλλαξης μας προστατεύει από τον εγκλωβισμό σε κάποιο τοπικό ακρότατο (local optima). Εδώ, και πάλι ανάλογα με το πρόβλημα, μπορούμε να επιλέξουμε έναν τελεστή που να μεταλλάξει ελαφρά μια λύση όπως π.χ. ο τελεστής της σημειακής μετάλλαξης σε δυαδικές αναπαραστάσεις λύσεων

1010010

1010110

όπου το πέμπτο bit μεταλλάχθηκε από 0 σε 1 ή κάποιο τελεστή που να μεταλλάξει έντονα μια λύση όπως π.χ. ο τελεστής πολλαπλών σημείων μετάλλαξης, τελεστής αντιστροφής και αλλοί.

Αφού γίνει οι κατάλληλες σχεδιαστικές αποφάσεις και υλοποιήσεις πλέον η υλοποίηση του γενετικού αλγορίθμου είναι εξαιρετικά απλή. Στο πρώτο βήμα αρχικοποιούμε ένα πληθυσμό λύσεων. Η αρχικοποίηση συνήθως γίνεται απευθείας στην αναπαράσταση που έχει επιλεγεί για τις λύσεις. Έπειτα, με βάση την συνάρτηση αξιολόγησης, εφαρμόζουμε τον τελεστή επιλογής από τον οποίο παίρνουμε ένα καινούργιο πληθυσμό, συνήθως υποσύνολο του αρχικού πληθυσμού. Στη συνέχεια χωρίζουμε τον καινούργιο πληθυσμό σε τυχαία επιλεγμένα ζεύγη λύσεων

τα οποία θα αποτελέσουν ζεύγη λύσεων-γονέων για τον τελεστή της διασταύρωσης. Σε αυτό το σημείο εφαρμόζουμε σε κάθε ζεύγος λύσεων-γονέων τον τελεστή της διασταύρωσης και παίρνουμε το σύνολο των λύσεων-τέκνων. Μετά εφαρμόζουμε, με μικρή πιθανότητα συνήθως $(1\% - 10\%)$ τον τελεστή την μετάλλαξης στο σύνολο των λύσεων-τέκνων. Στη συνέχεια οι λύσεις-τέκνα, προστίθενται ή αντικαθιστούν τους γονείς τους, στον πληθυσμό και έτσι ολοκληρώνεται μια γενιά. Αυτή η διαδικασία συνήθως επαναλαμβάνεται έως ότου ικανοποιηθεί κάποιο κριτήριο τερματισμού που συνήθως σχετίζεται με την συνάρτηση αξιολόγησης ή με τους διαθέσιμους πόρους π.χ. χρόνος εκτέλεσης του προγράμματος.

Πέραν του γενικού τους χαρακτήρα, οι γενετικοί αλγόριθμοι εξαιτίας της απλής της μορφής συνεργάζονται πολύ εύκολα και με άλλους αλγορίθμους. Αν γνωρίζουμε κάποιον αλγόριθμο που μπορεί να βελτιώσει μια λύση τότε μπορούμε πολύ εύκολα να τον αξιοποιήσουμε ως έναν επιπλέον τελεστή μετάλλαξης. Έτσι μπορεί να χρησιμοποιηθούν ως ένα γενικό πλαίσιο μέσα στο οποίο εκτελούνται και άλλοι αλγόριθμοι για ακόμη καλύτερα αποτελέσματα. Επιπλέον, είναι αλγόριθμοι οι οποίοι είναι ιδανική για παράλληλη υλοποίηση σε μεγάλα κατανεμημένα υπολογιστικά συστήματα. Μπορούμε να αρχικοποιούμε πολλούς πληθυσμούς οι οποίοι εξελίσσονται παράλληλα και σε ορισμένες στιγμές να τους ενοποιούμε για να μεταφέρουμε πετυχημένα χαρακτηριστικά ενός πληθυσμού σε κάποιον άλλο.

3.2 Βολικό πλαίσιο για προσαρμοστικά συστήματα;

Η γενική φύση του γενετικού αλγορίθμου και το ευρύ φάσμα εφαρμογών που έχει τον καθιστούν πολύ ελκυστικό για διάφορους πειραματισμούς. Εδώ θα παρουσιάσουμε τα μέρη του γενετικού αλγορίθμου που αποφασίσαμε να κρατήσουμε, τροποποιήσουμε και να απορρίψουμε.

Αρχικά θεωρούμε ότι η βασική μορφή του γενετικού αλγορίθμου, όπως την παρουσίασε ο Holland δεν είναι σε θέση να δημιουργήσει ένα προσαρμοστικό σύστημα. Ίσως, σωστότερη έκφραση θα ήταν να πούμε ότι δεν είναι σε θέση να βρει ένα προσαρμοστικό σύστημα, γιατί εμείς, όπως και ο Holland αντιλαμβανόμαστε τον γενετικό αλγόριθμο ως μια μέθοδο εξερεύνησης του χώρου λύσεων. Όμως προκύπτει το πολύ φυσικό ερώτημα: ¨Ποιος είναι ο χώρος τον οποίο καλούμαστε να εξερευνήσουμε για να βρούμε ένα προσαρμοστικό σύστημα;¨.

Σε εμάς, αυτή η προσέγγιση μοιάζει πολύ μακρινή και ξένη. Μια προσαρμοστική δομή δεν ψάχνεις να την βρεις αλλά την κατασκευάζεις ή μάλλον η ίδια κατασκευάζει τον εαυτό της. Ο λόγος που πιστεύουμε ότι μια προσαρμοστική δομή κατασκευάζει η ίδια τον εαυτό της είναι ανάλογη του τρόπου με τον οποίο εμείς οι άνθρωποι εξελισσόμαστε στο πέρασμα της ζωής μας. Αντιλαμβανόμαστε τον κόσμο, όχι απλώς παρατηρώντας τον και συνδυάζοντας αυτές τις παρατηρήσεις με την θεωρητική γνώση που έχουμε, αλλά παρατηρώντας και αλληλεπιδρώντας με τον κόσμο και προσαρμόζοντας την υπάρχουσα γνώση μας για αυτόν.

Θα λέγαμε ότι υιοθετούμε την κατά Jean Piaget κονστρουκτουβιστική/επιστημολογική άποψη για την φύση της γνώσης και θεωρούμε ότι μια δομή που έχει την δυνατότητα να την κατακτήσει πρέπει να διέπεται από τα ίδια κατασκευαστικά χαρακτηριστικά.

Έτσι αποφασίσαμε να αξιοποιήσουμε τον γενετικό αλγόριθμο για να κατευθύνουμε την σταδιακή πορεία του συστήματός μας. Αρχικά απορρίπτουμε την ιδέα του πληθυσμού, πράγμα που μοιάζει λογικό αφού όπως είπαμε δεν ψάχνουμε σε κάποιον χώρο, αλλά χτίζουμε μια δομή. Ο πλέον σημαντικός γενετικός τελεστής για εμάς είναι αυτός της επιλογής. Αυτός ενσαρκώνει όλη την βασική ιδέα της, κατά Δαρβίνου, εξέλιξης την οποία έχουμε αποφασίσει να υιοθετήσουμε. Όμως πώς γίνεται να υπάρξει επιλογή όταν υπάρχει μόνο μία δομή;

Εδώ έρχεται η σκέψη του βιολογικού πολλαπλασιασμού. Σε αυτό το σημείο θέλουμε να σταθούμε λίγο παραπάνω γιατί είναι αρκετά λεπτό και δημιουργούνται συχνά παρανοήσεις. Αρχικά να ξεκαθαρίσουμε ότι με τον όρο του βιολογικού πολλαπλασιασμού, η αναλογία που περιμένουμε να έχει κάποιος στο μυαλό του είναι ο πολλαπλασιασμός της διπλής έλικας του DNA . Ο λόγος που το επικεντρωνόμαστε στο DNA είναι γιατί σε αυτό υπάρχει κωδικοποιημένη η περιγραφή της διαδικασίας της αντιγραφής αλλά και από αυτό δίνεται η εντολή εκκίνησης αυτής της διαδικασίας. Ουσιαστικά το DNA δεν έχει μόνο την γνώση της αντιγραφής αλλά αποφασίζει και το ίδιο για την αντιγραφή του εαυτού του. Θα δούμε αναλυτικότερα αυτό το φαινόμενο το οποίο αποτελεί κεντρική ιδέα της δομής που θέλουμε να περιγράψουμε, όταν θα μιλήσουμε για την αυτοαναφορά.

Οπότε βλέπουμε ότι η δομή μας θα πολλαπλασιάζεται, με κάποιο τρόπο που θυμίζει τον πολλαπλασιασμό του DNA , το οποίο με άμεσο τρόπο επάγει την δημιουργία ενός γενεαλογικού δέντρου του οποίου κάθε γραμμή είναι μια γενιά. Ο τελεστής της επιλογής τώρα μεταφέρεται σε αυτή τη δομή με ανάλογο τρόπο αφού θεωρούμε ότι κάποιοι από τους απογόνους είναι πιο ικανοί από κάποιους άλλους με αποτέλεσμα κάποια κλαδιά του δέντρου να έχουμε μεγαλύτερο μήκος από κάποια άλλα. Έτσι μια δεδομένη χρονική στιγμή το κλαδί με το μεγαλύτερο μήκος (μπορεί να είναι και πολλά) είναι αυτά που μέχρι στιγμής έχουν κριθεί επιτυχημένα από το περιβάλλον, δηλαδή τον τελεστή της επιλογής και πολλαπλασιάζονται για να δώσουν την επόμενη γενιά.

Όπως βλέπουμε και από τα παραπάνω ο γενετικός τελεστής της διασταύρωσης δεν θα αξιοποιηθεί στο σύστημα που θέλουμε να περιγράψουμε αφού τα προγράμματα πολλαπλασιάζονται ατομικά για να δώσουν απογόνους για την επόμενη γενιά. Επίσης θεωρούμε ότι δεν εξυπηρετεί τον προσαρμοστικό χαρακτήρα που θέλουμε να πετύχουμε γιατί είναι μια στατική λειτουργία η οποία δεν έχει δυνατότητα εξέλιξης με το πέρασμα των γενεών.

Τέλος ο γενετικός τελεστής της μετάλλαξης μεταφέρεται με πολύ φυσιολογικό τρόπο στο σύστημά μας. Η αλήθεια είναι ότι οι μεταλλάξεις που προκύπτουν ως λάθη κατά την αντιγραφή του DNA συνήθως χειροτερεύουν την επίδοση του συστήματος, όμως αν σκεφτούμε

την πιο πρωτόλεια δομή του RNA οι μεταλλάξεις του οδήγησαν στην τεράστια ποικιλιών των οργανισμών που συναντάμε σήμερα. Έτσι και εμείς θεωρούμε τις μεταλλάξεις ως μιας μορφής πειραματισμού του μοντέλου μας ώστε να ανακαλύψει τρόπους για μεγαλύτερη επιτυχία μέσα στο περιβάλλον. Επιπλέον θεωρούμε ότι όχι μόνο δεν είναι «λάθη» οι μεταλλάξεις αλλά δεν είναι ούτε απολύτως τυχαίες. Στόχος μας είναι η μη-πιστή αντιγραφή ακριβώς για να υπάρχει εξέλιξη και πιο συγκεκριμένα αναζητάμε ένα μηχανισμό ο οποίος θα προτείνει τις αλλαγές που θα γίνουν στην επόμενη αντιγραφή. Έτσι τις σκόπιμα, μη-πιστές αντιγραφές, στο πνεύμα όμως του DNA, οι οποίες θα περιέχουν μεταλλάξεις, θα τις καλούμε αυτοαναφορικές μεταβάσεις. Αυτές οι μεταβάσεις, για εμάς δεν είναι εντελώς τυχαίες γιατί είναι κατευθυνόμενες από αυτόν τον μηχανισμό τον οποίο θα περιγράψουμε στο κύριο θέμα της εργασίας και θα καλούμε, διαγωνιοποίηση.
Ως διαδικασίες εξερεύνησης, περιέχουν τον πειραματισμό και συνεπώς την τύχη, δεν αποτελούν όμως τυφλές απόπειρες, αλλά στοχευμένες ώστε να εκπληρώσουν κάποιον συγκεκριμένο σκοπό.

Βλέπουμε λοιπόν, ότι αφού το μοντέλο μας υπόκειται στην υποχρεωτική κατά την γνώμη μας διαδικασία της εξέλιξης, η οποία αποτελεί και το μοναδικό στατικό και μη-προσαρμοστικό χαρακτηριστικό, το πλαίσιο του γενετικού αλγορίθμου είναι απαραίτητο. Βέβαια, το έχουμε τροποποιήσει αρκετά σε σχέση με την βασική του μορφή, γιατί για τους λόγους που περιγράψαμε θεωρούμε ότι χωρίς την αυτοαναφορά πραγματική προσαρμοστικότητα δεν μπορεί να επιτευχθεί.

3.3 Αυτοαναφορά

Η αυτοαναφόρα είναι μάλλον η βασική μας διαφορά σε σχέση με την συντριπτική πλειονότητα των μοντέλων τεχνητής νοημοσύνης που ερευνώνται και αξιοποιούνται από την επιστημονική κοινότητα. Μας ενδιαφέρει να υποδείξουμε ότι αυτός ο δρόμος αξίζει παραπάνω σημασίας για την επίτευξη ενός γενικά ευφυούς συστήματος.

Αν αναλογιστούμε τον αυτοαναφορικό χαρακτήρα που έχουν οι φυσικές γλώσσες, είτε σημασιολογικά μέσω των αντωνυμιών είτε δομικά μέσω της γραμματικής και του συντακτικού τα οποία αξιοποιούμε και αναδρομικά κατασκευάζουμε τις προτάσεις, θα λέγαμε ότι όλοι οι άνθρωποι είμαστε αρκετά εξοικειωμένοι με το φαινόμενα της αυτοαναφοράς. Διέπει ολοκληρωτικά το λόγο μας και συνεπώς την σκέψη μας. Για εμάς το γεγονός ότι διδαχθήκαμε την κατανόηση και χρήση της γλώσσας από το τυπικό και φορμαλιστικό επίπεδο μέχρι την απλή πρακτική εφαρμογή μέσω της ίδιας της γλώσσας αποτελεί ένα σημαντικό παράδειγμα της χρήσης της αυτοαναφοράς για την βελτίωση ενός συστήματος. Η γλώσσα είναι εξαιρετικά σημαντική και αποτελεί αναπόσπαστο κομμάτι ενός ευφυούς προσαρμοστικού συστήματος.

Όσον αφορά τα μαθηματικά και γενικώς τα τυπικά συστήματα η αυτοαναφορά κάνει πολύ αισθητή την παρουσία της από τον Cantor και μετά. Το διαγώνιο επιχείρημα του Cantor με το οποίο απέδειξε ότι η πληθικότητα του συνόλου των πραγματικών αριθμών είναι γνήσια

μεγαλύτερη από αυτήν των φυσικών είναι για εμάς σημείο αναφοράς και από αυτό αργότερα, θα ονομάσουμε μια αλγοριθμική διαδικασία του συστήματος που προτείνουμε. Οι μετέπειτα ιδέες που επηρεάστηκαν από αυτό το επιχείρημα είναι πάρα πολλές. Το παράδοξο του Russell που οδήγησε στην περιορισμένη αρχή της συμπερίληψης, τα θεωρήματα μη πληρότητας του Godel,το θεώρημα μη ορισιμότητας του Tarski, το δημοφιλές πρόβλημα του τερματισμού Halting Problem είναι μερικές μόνο από τις επιρροές της αυτοαναφοράς στον κλάδο των μαθηματικών και της θεωρητικής πληροφορικής.

Επιπλέον, στον προγραμματισμό μπορούμε να δούμε την έντονη χρήση της αυτοαναφοράς και μάλιστα ως μέσο κατασκευής δομών όπως σκοπεύουμε να την χρησιμοποιήσουμε και εμείς. Έχει ουσιαστική χρήση στην κατασκευή δομών δεδομένων όπου παρουσιάζεται μέσω αναδρομικών ορισμών. Ενδεικτικά δίνουμε τους ορισμούς της απλά συνδεδεμένης λίστας και του δυαδικού δέντρου στην αμιγώς συναρτησιακή γλώσσα προγραμματισμού Haskell όπου φαίνεται καθαρά ο αναδρομικός χαρακτήρας τους.

Λίστα

$$\text{data List a} = \text{EmptyList} \mid \text{Cons a (List a)}$$

Δέντρο

$$\text{data Tree a} = \text{EmptyTree} \mid \text{Node a (Tree a) (Tree a)}$$

Επίσης η πολύ συνηθισμένη τεχνική του bootstrapping κατά την οποία ένας μεταγλωττιστής μεταγλωττίζει το ίδιο του το πρόγραμμα. Αυτό γίνεται για λόγους βελτιστοποίησης γιατί στην πρώτη υλοποίηση ενός μεταγλωττιστή, που συνήθως γίνεται σε κάποια γλώσσα χαμηλότερου επιπέδου, είναι εξαιρετικά δύσκολο να γραφτούν με το χέρι βελτιστοποιήσεις. Ένα άλλο φαινόμενο καθαρής αυτοαναφοράς είναι αυτό της ανάκλασης. Κατά την ανάκλαση ένα πρόγραμμα μπορεί να διαβάζει και να τροποποιεί το πρόγραμμά του. Η ανάκλαση χρησιμοποιούνταν πολύ στο παρελθόν και ιδιαίτερα στις χαμηλού επιπέδου γλώσσες assembly για να μειωθεί ο αριθμός των εντολών ενός προγράμματος. Επίσης όπως αναφέραμε στο προηγούμενο κεφάλαιο η γλώσσα προγραμματισμού Lisp φτιάχτηκε πλήρως προσανατολισμένη γύρω από την ανάκλαση. Γενικά η χρήση τέτοιων αυτο-επεξεργαστικών προγραμμάτων θεωρείται κακή προγραμματιστική τεχνική καθώς γίνεται δύσκολη η απασφαλμάτωση (debugging). Εμείς όμως πάνω σε αυτό ακριβώς το φαινόμενο θα στηριχτούμε για την κατασκευή της δομής μας. Τέλος βλέπουμε ότι η αυτοαναφορά συναντάται ακόμα και στο υλικό των υπολογιστών (computer hardware) όπου η μνήμη υλοποιείται από τις πύλες flip-flop οι οποίες είναι δύο NOR πύλες που δέχονται και δίνουν είσοδο και έξοδο η μία στην άλλη ταυτόχρονα. Αυτή η παράξενη συσχέτιση των δύο πυλών δημιουργεί την έννοια της κατάστασης και έτσι αποθηκεύεται πληροφορία. Η έννοια της κατάστασης είναι σημαντική για εμάς, πιο συγκεκριμένη θεωρούμε ότι μια αυτοαναφορική δομή μεταβαίνει από μια κατάσταση σε μια άλλη μέσω μιας αυτοαναφορικής μετάβασης.

Για παράδειγμα αν S ένα σύνολο καταστάσεων και $f : S \longrightarrow S$ ένας τελεστής μετάβασης από μια κατάσταση σε μια άλλη τότε αν $s \in S$ έχουμε:

$$s \xrightarrow{f} f(s)$$
$$(s, f) \longrightarrow (f(s), f)$$

όπου στην πρώτη περίπτωση έχουμε την συνήθης περίπτωση στην οποία ο τελεστής μετάβασης f είναι «εξωτερικός» του s, ενώ στην δεύτερη περίπτωση η δομή περιέχει τον τελεστή που σημαίνει ότι τον βλέπει και τον ελέγχει, δηλαδή είναι «εσωτερικός» της δομής.

Περνώντας τώρα στην βιολογική σκοπιά της αυτοαναφοράς, όπως αναφέραμε και πιο πάνω το DNA είναι μια αυτοαναφορική δομή που μας αρέσει να σκεφτόμαστε σαν πρότυπο. Εδώ θα δούμε ότι το DNA μοιάζει πολύ με έναν αυτο-επεξεργαστικό κωδικό. Το DNA σχεδόν για όλες τις λειτουργίες που θέλει να υλοποιήσει, είτε αφορούν το ίδιο είτε κάποιο άλλο μέρος του κυττάρου, χρησιμοποιεί τις πρωτεΐνες. Το DNA δεν περιέχει τις πρωτεΐνες αλλά, ως κωδικός, περιέχει μια περιγραφή τους. Έτσι σε κάποιο σημείο του DNA βλέπουμε μια ακολουθία

$$c = (c_1, c_2, \cdots, c_n)$$

όπου με c συμβολίζουμε την περιγραφή της πρωτεΐνης και με c_i την περιγραφή των αμινοξέων από τα οποία δομείται των οποίων η σειρά μας ενδιαφέρει οπότε η περιγραφή θεωρούμε ότι είναι μια διατεταγμένη λίστα. Εδώ να σημειώσουμε ότι μια πρωτεΐνη περιγράφεται πλήρως από τα αμινοξέα της με βάση το δόγμα του Άνφινσεν (Anfinsen's Dogma). Αρχικά την περιγραφή $c = (c_1, c_2, \cdots, c_n)$ παίρνει το ριβόσωμα και δημιουργεί το

$$a(c) = (a(c_1), a(c_2), \cdots, a(c_n))$$

όπου τα $a(c_i)$ είναι αμινοξέα και $a(c)$ το πολυπεπτίδιο. Ένα πολυπτεπτίδιο είναι μια πρωτεΐνη που απλά δεν έχει πάρει ακόμα μια σταθερή μορφή στο χώρο. Στη συνέχεια το πολυπεπτίδιο παίρνει μια σταθερή μορφή στο χώρο μέσω ενός μετασχηματισμού s και επειδή μια πρωτεΐνη μπορεί να επιτελέσει πολλές λειτουργίες μπορούμε να πούμε ότι για κάθε λειτουργία ευθύνεται μια συγκεκριμένη περιοχή κάθε φόρα όποτε συμβολίζουμε με f τον τελεστή «προβολής» της πρωτεΐνης στην κατάλληλη περιοχή. Αν θεωρήσουμε ότι το DNA έχει την μορφή

$$DNA = (\cdots, c_1, c_2, \cdots, c_n, \cdots)$$

και ότι η πρωτεΐνη εκτελεί κάποια λειτουργία πάνω στο ίδιο το DNA, είναι δηλαδή ένας τελεστής τότε

$$(f \circ s \circ a)(c)(DNA) \tag{3.1}$$

αποτελεί μια αυτοαναφορική λειτουργία του DNA αφού η μόνη πληροφορία που απαιτείται είναι ουσιαστικά η περιγραφή της πρωτεΐνης. Να τονίσουμε ότι όσον αφορά την δική μας σκοπιά η απεικόνιση $f \circ s \circ a$ είναι κάτι στατικό, δηλάδη δεν μεταβάλλεται και απλώς επιτελεί την διαδικασία της μετάφρασης από την συμβολική αφαίρεση της περιγραφής στην υλοποίσή της δηλαδή στην πρωτεΐνη.

Όπως θα δούμε αναλυτικότερα στην επόμενη ενότητα ακριβώς επειδή στη σχέση η πληροφορία που μας ενδιαφέρει υπάρχει μόνο στο DNA και στο c, θα γράφουμε την σχέση 3.1 ως

$$\textit{self-ed}(DNA[\overline{c}])$$

οπού σε αυτή τη μορφή έχουμε αφαιρέσει τις περιττές λεπτομέρειες και φαίνεται ότι το DNA περιέχει την πληροφορία της περιγραφής του c από τις αγκύλες και ότι αυτή δρα στο DNA από την γραμμή πάνω από το c.

Η αυτοαναφορά, για εμάς είναι ένα εργαλείο με το οποίο εκμεταλλευόμενοι την λεπτή γραμμή που υπάρχει μεταξύ της περιγραφής μια διαδικασίας, γενετικός κωδικός του DNA, πηγαίος κώδικας ενός προγράμματος κ.α. μπορούμε να κατασκευάσουμε ενδιαφέρουσες δομές. Εδώ, αν και σχετίζεται περισσότερο με τους γενετικούς αλγορίθμους παρά με το φαινόμενο της αυτοαναφοράς και συχνά δημιουργείται σύγχυση, θα θέλαμε να αναφέρουμε τον Von Neumann και τον καθολικό κατασκευαστή (Universal Constructor). Στον καθολικό κατασκευαστή ο στόχος είναι η εξέλιξη της πολυπλοκότητας ενός συστήματος μέσω του γενετικού αλγορίθμου όπου όμως ο τελεστής της διασταύρωσης έχει αντικατασταθεί από αυτο-πολλαπλασιαστικές μεταβάσεις. Ο λόγος που δεν τις καλούμε αυτοαναφορικές και δεν τις θεωρούμε κιόλας, είναι ότι το σύστημα απλά και τυφλά πολλαπλασιάζεται χωρίς να ελέγχει πότε και πώς και γιατί πρέπει να πολλαπλασιαστεί όπως κάνει το DNA και το σύστημα που θα περιγράψουμε στο επόμενο κεφάλαιο. Όμως στον καθολικό κατασκευαστή παρουσιάζεται εξαιρετικά όμορφα το γεγονός ότι πράγματι η περιγραφή μιας διαδικασίας από την ίδια την διαδικασία διαφέρουν ελάχιστα. Έτσι θεωρούμε ένα σύνολο μηχανών M και ένα σύνολο περιγραφών μηχανών D και δύο μηχανές την c η οποία δέχεται σαν είσοδο μια μηχανή και δίνει την περιγραφή της και την μηχανή d η οποία δέχεται σαν είσαι μια μηχανή και δίνει στην έξοδο την μηχανή που περιγράφεται από την περιγραφή που πήρε. Τότε η σύνθεση $d \circ c$ αποτελεί μια μηχανή που αντιγραφεί μηχανές. Στο γενικό πλαίσιο στοχεύεται η επίτευξη μεγάλης πολυπλοκότητα εξαιτίας των «λαθών» δηλαδή των μεταλλάξεων που κάνει η μηχανή $d \circ c$.

Κλείνοντας το κεφάλαιο αυτό να πούμε ότι η δομή που θα κατασκευάσουμε στο επόμενο κεφάλαιο θα είναι στο πλαίσιο που περιγράφηκε εδώ, δηλαδή αυτοαναφορική, την εξέλιξης της οποίας θα ελέγχουμε με την βοήθεια της εκδοχής του γενετικού αλγορίθμου που περιγράψαμε. Όσον αφορά την αυτοαναφορά αρκετές φορές δυσκολεύει και περιπλέκει τα πράγματα για αυτό και η έκθεση της εδώ ελπίζουμε να εξοικείωσε κάπως τον αναγνώστη για την συστηματικότερη ανάλυση που θα ακολουθήσει.

Κεφάλαιο 4

Μια νέα πρόταση

Σε αυτήν την ενότητα θα παρουσιάσουμε ένα μοντέλο μάθησης το οποίο είναι εμπνευσμένο από την αυτοαναφορική κατά την άποψη μας, φύση της μάθησης, και από την δαρβινική επιλογή που μάλλον διέπει τον κόσμο μας.

Παρατηρώντας κανείς μερικά μοντέλα τεχνητής ευφυίας μπορεί να εξάγει ένα συμπέρασμα σχετικά με τον τρόπο λειτουργίας τους: υπάρχει ένα μοντέλο m και ένας αλγόριθμος εκπαίδευσης t ο οποίος παίρνει σαν είσοδο το m_0 και δίνει ένα m_1, στη συνέχεια παίρνει το m_1 σαν είσοδο και δίνει ένα m_2 κλπ. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να έχουμε ένα ικανοποιητικό μοντέλο βάσει κάποιου μέτρου επίδοσης. Ωστόσο, ο αλγόριθμος εκπαίδευσης t μπορεί να δέχεται και άλλα δεδομένα, όπως μια συνάρτηση ανταμοιβής (reinforcement learning)ή μια ετικέτα (supervised learning).Σε αυτήν τη διαδικασία παρατηρούμε ότι η βελτίωση του μοντέλου γίνεται με έναν στατικό τρόπο. Ωστόσο, υπάρχουν και περιπτώσεις όπου το μοντέλο εκπαιδεύεται αρχικά με έναν αλγόριθμο t και αφού βελτιωθεί, το εκπαιδεύουμε σε ένα πιο πολύπλοκο περιβάλλον που ωστόσο είναι συναφές, χρησιμοποιώντας έναν διαφορετικό αλγόριθμο t'.

Εμείς θεωρούμε ότι η αυτοαναφορά είναι η λύση σε αυτό το πρόβλημα. Πιο συγκεκριμένα εάν θεωρήσουμε τον αλγόριθμο εκπαίδευσης t ως μέρος του μοντέλου μας m τότε η προηγούμενη διαδικασία μετατρέπεται στην εξής: Το μοντέλο m_0 δέχεται σαν είσοδο τον εαυτό του m_0 και δίνει ένα μοντέλο m_1. Το μοντέλο m_1 δέχεται σαν είσοδο τον εαυτό του m_1 και δίνει ένα μοντέλο m_2 κλπ. Αυτός ο τρόπος έχει ένα βασικό πλεονέκτημα σε σχέση με την προηγούμενη διαδικασία. Ο τρόπος εκπαίδευσης βελτιώνεται με αυτόματο τρόπο μαζί με το μοντέλο. Θα λέγαμε ότι μέσα στις επαναλήψεις το μοντέλο δεν μαθαίνει απλώς, αλλά μαθαίνει πως να μαθαίνει. Αυτό σημαίνει ότι δημιουργείται ένα μοτίβο εκπαίδευσης του t το οποίο όμως αφού είναι εσωτερικό του μοντέλου με την σειρά του βελτιώνεται. Έτσι μπορούμε να πούμε ότι δεν πρόκειται για ένα απλό μοντέλο μάθησης, ούτε για ένα απλό μοντέλο μέτα-μάθησης (meta-learning),που θα μετέβαλε τον t με ένα σταθερό τρόπο είτε χειροκίνητα είτε αλλάζοντας κάποιες υπερπαταμέτρους, αλλά για ένα μοντέλο αναδρομικής μέτα-μάθησης(recursive meta-learning).

Μιας και μια τέτοια διαδικασία μάλλον έχει έντονες μεταβολές, είτε σύγκλισης είτε όχι, θα θέλαμε να αξιοποιήσουμε την δαρβινική επιλογή για να κατευθύνουμε τη σύγκλιση. Αυτό

μπορούμε να το πετύχουμε με μια σχετικά απλή διαφοροποίηση στο μοντέλο μας. Το μοντέλο m_0 δέχεται σαν είσοδο το μοντέλο m_0 και δίνει τους απογόνους m_i^0, μετά καθένας από τους απογόνους δέχεται ως είσοδο τον εαυτό του και δίνει απογόνους m_j^{i0} κλπ. Ουσιαστικά έτσι έχουμε ένα γενεαλογικό δέντρο μοντέλων στο οποίο όπως θα δούμε παρακάτω μπορούμε να έχουμε μια μορφή δαρβινικής επιλογής.

4.1 Βασική αρχή της αυτο-επεξεργασίας

Αρχικά θεωρούμε ότι ένα πρόγραμμα c είναι μια διατεταγμένη λίστα εντολών

$$c = (c_1, c_2, \cdots, c_n).$$

Η αλήθεια είναι ότι δεν υπάρχει σαφής διαφορά μεταξύ εντολής και προγράμματος, μιας και μια εντολή θα μπορούσε με την σειρά της να είναι ένα ολόκληρο πρόγραμμα. Παρόλα αυτά σίγουρα μπορούμε να θεωρήσουμε ότι για κάθε πρόγραμμα $c = (c_1, c_2, \cdots, c_n)$ κάθε c_i είναι απλούστερο από το c. Έτσι θα μπορούσαμε να πούμε ότι κάθε πρόγραμμα έχει πολλές αναπαραστάσεις και κάθε φορά επιλέγουμε την κατάλληλη ανάλογα σε το επίπεδο λεπτομέρειας που επιθυμούμε να το δούμε.

Μια άλλη έννοια που θα χρειαστούμε είναι αυτή του υπολογισμού. Θα θεωρήσουμε ως υπολογισμό την μετάβαση από μια κατάσταση σε μια άλλη λόγω της εκτέλεσης ενός προγράμματος. Πιο συγκεκριμένα αν κατά την εκτέλεση του προγράμματος c το σύστημα μεταβαίνει σειριακά στις καταστάσεις

$$(s_0, s_1, \cdots, s_{n+1})$$

όπου κάποιες από αυτές μπορεί να επαναλαμβάνονται, τότε θεωρούμε ως υπολογισμό την ακολουθία $(i_0, i_1, \cdots, i_n)$ όπου το i_j είναι υπεύθυνο για την μετάβαση $s_j \to s_{j+1}$.

Σε αντιστοιχία με την αναπαράσταση του προγράμματος, ένας υπολογισμός μπορεί να είναι μια απλή μετάβαση από μια κατάσταση σε μια άλλη όπως πχ στην εκτέλεση μιας πρωτοβάθμιας εντολής ή μια πιο σύνθετη μετάβαση όπως πχ στην εκτέλεση ενός ολόκληρου προγράμματος. Έτσι εκφράζουμε έναν υπολογισμό ως μια λίστα απλούστερων υπολογισμών τους οποίους θα καλούμε υπολογιστικά βήματα. Επίσης θα λέμε ότι μια εντολή c_j είναι ενεργοποιημένη εντολή όταν λόγω αυτής το σύστημα μεταβαίνει από την κατάσταση s_j στην κατάσταση s_{j+1}, δηλαδή εκτελείται το υπολογιστικό βήμα i_j. Αξίζει να σημειωθεί ότι αν το πρόγραμμα c είναι μέρος ενός μεγαλύτερου προγράμματος τότε ανάλογα με το επίπεδο λεπτομέρειας που επιθυμούμε θα μπορούσαμε να θεωρήσουμε και το ίδιο το c ως ενεργοποιημένη εντολή, αφού περιέχει την c_j. Επειδή θα ασχοληθούμε αρκετά με προγράμματα που έχουν υποπρογράμματα/εντολές θα εισάγουμε έναν συμβολισμό. Θα λέμε ότι το πρόγραμμα b είναι υποπρόγραμμα του c και θα γράφουμε $c = c[b]$. Αν το b είναι ενεργοποιημένο θα γράφουμε $c = c[\overline{b}]$.

Όπως είπαμε στην αρχή, μας ενδιαφέρει να μελετήσουμε προγράμματα τα οποία δέχονται σαν είσοδο τον εαυτό τους. Αυτό θα θέλαμε να το συμβολίσουμε με ένα ξεχωριστό τρόπο για να το ξεχωρίζουμε από τις παραδοσιακές μεταβάσεις.

Αν λοιπόν έχουμε ένα πρόγραμμα $c = c[b]$ όπου το b υλοποιεί κάποιον αλγόριθμο, που θα συμβολίζουμε με $alg(b)$, και αυτός δέχεται σαν είσοδο το c τότε θα γράφουμε:

$$\textit{self-ed}(c[\overline{b}]) = alg(b)(c[\overline{b}]).$$

Σε αυτό το σημείο έχει ενδιαφέρον να δούμε κάτι το οποίο αν και είναι ιδιαίτερα προφανές είναι πολύ ισχυρό. Παρατηρούμε ότι αν έχουμε έναν αλγόριθμο B ο οποίος υλοποιείται από ένα πρόγραμμα b. Τότε ισχύει το εξής ενδιαφέρον:

$$\textit{self-ed}(c[\overline{b}]) = alg(b)(c[\overline{b}]) = B(c[\overline{b}]) \tag{4.1}$$

Δηλαδή, όσες διαδικασίες μπορούμε να υλοποιήσουμε στο συνηθισμένο πλαίσιο, μπορούμε να τις υλοποιήσουμε και «εσωτερικά», πράγμα που μας δίνει την δυνατότητα να τις τροποποιούμε και να τις προσαρμόζουμε με έναν αυτόματο τρόπο. Σε αυτό βέβαια το σημείο, μπορεί κανείς να υποστηρίξει ότι δεν μπορεί να συμβαίνει κάτι τέτοιο μιας και κάθε πρόγραμμα έχει συγκεκριμένο τύπο δεδομένων που δέχεται ως είσοδο και δίνει ως έξοδο. Αυτό το πρόβλημα είναι εύκολο να αντιμετωπιστεί θεωρώντας ότι το αντίστοιχο δεδομένο βρίσκεται σε κάποιο σημείο του c. Θα το δούμε με περισσότερη λεπτομέρεια αυτό παρακάτω όταν θα οργανώσουμε το πρόγραμμά μας με διευθύνσεις μνήμης. Στην έκφραση 4.1 θα θέλαμε να σταθούμε λίγο στο γεγονός ότι το b είναι ενεργοποιημένη εντολή. Ουσιαστικά το πρόγραμμα που υλοποιεί τον αλγόριθμο B υπάρχει μέσα στο c και είναι το b. Με το να είναι ενεργοποιημένο το b πρακτικά εκτελούμε τον αλγόριθμο και η ενδιαφέρουσα δυνατότητα που έχει αυτό είναι ότι ένας αλγόριθμος μπορεί να «βλέπει» ότι εκτελείται. Από εδώ και πέρα θα αναφερόμαστε στην σχέση 4.1 ως **Βασική αρχή της αυτο-επεξεργασίας**. Παρακάτω θα δούμε μερικές ελκυστικές δυνατότητες που έχει ένα πρόγραμμα που μπορεί να επεξεργάζεται τον εαυτό του.

Γενικά θέλουμε να σκεφτόμαστε λειτουργίες που όταν εκτελούνται «εσωτερικά» δίνουν ενδιαφέροντα αποτελέσματα. Μάλλον η πρώτη λειτουργία που μας έρχεται στο μυαλό είναι η αντιγραφή, ειδικότερα ο πολλαπλασιασμός, με την βιολογική του έννοια. Η δυνατότητα ενός προγράμματος, που επεξεργάζεται τον εαυτό του, να πολλαπλασιάζεται είτε πίστα είτε με παραλλαγές είναι ιδανική για να έχουμε ένα φαινόμενο δαρβινικής επιλογής.

Επίσης ένα τέτοιο πρόγραμμα μπορεί να κρατάει μνήμη του εαυτού του. Αυτό είναι εμφανές, διότι αν f είναι το πρόγραμμα ενός αλγορίθμου τότε μπορούμε να ορίσουμε τον αλγόριθμο G ώστε $G(x) = (x, alg(f)(x))$ με πρόγραμμα g. Τότε

$$\textit{self-ed}(c[\overline{g}]) = alg(g)(c) = (c, alg(f)(c)).$$

Αυτή η ικανότητα είναι μάλλον η σημαντικότερη, καθώς θέλουμε το πρόγραμμά μας να έχει πρόσβαση σε ένα ιστορικό της εξέλιξής του ώστε να μπορεί να εξάγει κάποια πρότυπη συμπεριφορά.

4.2 Δομημένα προγράμματα και διευθύνσεις

Σε αυτό το σημείο θα δούμε πως μπορούμε να οργανώσουμε το πρόγραμμά μας, με διευθύνσεις, ώστε να μπορούμε να αναφερόμαστε στα διάφορα μέρη του (υποπρογράμματα, δεδομένα κλπ). Αρχικά ας θεωρήσουμε ένα πρόγραμμα c το οποίο μπορούμε να δούμε ως σύνθεση τριών υποπρογραμμάτων, έστω $c = c[b_1, b_2, b_3]$. Κατά αναλογία με τον υπάρχων συμβολισμό για λίστες ή διανύσματα σε διάφορες γλώσσες προγραμματισμού θα λέμε ότι το b_1 βρίσκεται στην θέση 1, το b_2 βρίσκεται στη θέση 2 και το b_3 βρίσκεται στη θέση 3 του c. Επειδή όπως είδαμε προηγουμένως, ένα πρόγραμμα μπορεί να έχει πολλές αναπαραστάσεις ανάλογα με το επίπεδο λεπτομέρειας που επιθυμεί κανείς να έχει, θα μπορούσαμε να έχουμε μια αναπαράσταση της μορφής

$$c = c[b_1[b_1^1, b_1^2], b_2[b_2^1, b_2^2], b_3[b_3^1, b_3^2]].$$

Σε αυτή την περίπτωση για να αποδώσουμε ότι το b_j^i βρίσκεται στην i-θέση του j-οστου υποπρογράμματος θα λέμε ότι βρίσκεται στη θέση $[j, i]$. Τέτοιες ακολουθίες θα τις αποκαλούμε διευθύνσεις. Εάν θέλουμε να αναφερθούμε σε ένα πρόγραμμα c με ένα υποπρόγραμμα b το οποίο βρίσκεται στη διεύθυνση θ θα γράφουμε $c[(\theta)b]$.

Τώρα θα επιστρέψουμε στην έννοια του υπολογισμού και θα μελετήσουμε τον τρόπο εκτέλεσης ενός αυτο-επεξεργαστικού προγράμματος. Όπως είπαμε, ένα πρόγραμμα μπορούμε να το δούμε ως μια σύνθεση υποπρογραμμάτων. Πιο συγκεκριμένα, ως μια πεπερασμένη ακολουθία $B_1, \cdots, B_n$ από στοιχεία ενός συνόλου $\mathcal{B}$ (το οποίο λέμε σύνολο εντολών) και ενός αλγορίθμου B ο οποίος ελέγχει τη ροή εκτέλεσης των εντολών $B_1, \cdots, B_n$. Έτσι λοιπόν μπορούμε να πούμε, ότι πρώτα εκτελείται η εντολή B_1, στη συνέχεια ο αλγόριθμος B επιλέγει την επόμενη εντολή με βάση την σειρά της αλλά και με βάση το αποτέλεσμα του υπολογισμού της.

Πιο επίσημα: ένας $\mathcal{B}-$αλγοριθμικός υπολογισμός με πρόγραμμα $(B : B_1, \cdots, B_n)$, όπου $B_1, \cdots, B_n \in \mathcal{B}$, και ένας αλγόριθμος B που ελέγχει τη ροή του υπολογισμού, περιγράφεται ως εξής: Δεδομένου ενός προγράμματος x ως είσοδο, ο $(B : B_1, \cdots, B_n)$ παράγει την ακολουθία προγραμμάτων

$$x = x_1 \to x_2 \to \cdots$$

όπου:

1. Για κάθε $k = 1, 2, \cdots$, υπάρχει ένας δείκτης j_k στο εύρος $1, \cdots, n$, τέτοιο ώστε $x_{k+1} = B_{j_k}(x_k)$. Σε αυτή την περίπτωση το B_{j_k} καλείται ενεργοποιημένη ακολουθία του υπολογιστικού βήματος $x_k \to x_{k+1}$.
2. Υποθέτουμε ότι $j_1 = 1$, άρα $x_2 = B_1(x_1)$.
3. Για κάθε $k = 1, 2 \cdots$, έχουμε ότι $j_{k+1} = B(j_k, x_{k+1})$, δηλαδή η επιλογή της εντολής που θα ενεργοποιηθεί είναι συνάρτηση της προηγούμενης εντελής (ως προς την θέση) που χρησιμοποιήθηκε και του τρέχοντος υπολογισμού.

Να υπενθυμίσουμε εδώ ότι οι εντολές $B_1, B_2, \cdots, B_n$ μπορούν με τη σειρά τους να είναι ολόκληρα προγράμματα, οπότε έχουμε ένα δομημένο πρόγραμμα (structured program).Θα χρησιμοποιήσουμε την έννοια του δομημένου προγράμματος ώστε να έχουμε την επιθυμητή αναπαράσταση ανάλογα με την περίσταση.

4.3 Πολλαπλασιαστικοί υπολογισμοί

Σε ότι ακολουθεί θα συμβολίζουμε με $\{x, x\}$ το πρόγραμμα που περιέχει δύο ίδια προγράμματα x. Να σημειώσουμε επίσης, ότι η ονομασία «πολλαπλασιαστικός υπολογισμός» είναι εμπνευσμένη από τη βιολογία και υπονοεί τον βιολογικό πολλαπλασιασμό.

Ένα αρχικό απλό παράδειγμα είναι η εξής συνάρτηση:

$$R : x \to \{x, x\}$$

όπου το x είναι πρόγραμμα. Η R δέχεται σαν είσοδο ένα πρόγραμμα x και παράγει δύο αντίγραφα του x. Πιο γενικά, αν $R_1, \cdots, R_k$ είναι αλγοριθμικές συναρτήσεις, όχι κατ' ανάγκη διαφορετικές μεταξύ τους, συμβολίζουμε με $\{R_1, \cdots, R_k\}$ τον αλγόριθμο που ορίζεται ως εξής:

$$\{R_1, \cdots, R_k\}(x) = \{R_1(x), \cdots, R_k(x)\}.$$

Ένα πρόγραμμα, σαν το παραπάνω, που δίνει σαν έξοδο περισσότερα από ένα προγράμματα θα το λέμε πολλαπλασιαστικό. Τα προγράμματα $R_1(x), R_2(x), \cdots, R_k(x)$ καλούνται παιδιά ή άμεσοι απόγονοι του x κι θα συμβολίζονται με $im\text{-}suc(x)$.Σε σύμπλευση με ότι έχουμε πει, η εντολή $\{R_1, \cdots, R_k\}$ καλείται ενεργοποιημένη εντολή του βήματος και αντίστοιχα , η R_i είναι η ενεργοποιημένη ακολουθία του (μερικού) υπολογιστικού βήματος $x \to R_i(x)$.

Όπως αναφέραμε πιο πάνω, τα πολλαπλασιαστικά προγράμματα είναι αυτά που περιέχουν μία ή περισσότερες πολλαπλασιαστικές εντολές. Εδώ θα θέλαμε να επανέλθουμε στην ιδέα που είχαμε αναφέρει στην εισαγωγή του κεφαλαίου, ότι δηλαδή ένα πολλαπλασιαστικό πρόγραμμα κάθε φορά δημιουργεί απογόνους, τελικά δημιουργεί ένα γενεαλογικό δέντρο. Ένα τέτοιο δέντρο θα το συμβολίζουμε με $(x_t)_{t \in T}$, όπου το σύνολο των δεικτών T έχει την ίδια δομή με το γενεαλογικό μας δέντρο. Κάθε στοιχείο του δέντρου έχει από 0 μέχρι οποιονδήποτε πεπερασμένο αριθμό άμεσων απογόνων. Θα συμβολίζουμε την ρίζα του δέντρου με $x_\emptyset$.

Δεδομένης της μέχρι τώρα πορείας μας, μοιάζει φυσικό επακόλουθο να επιχειρήσουμε να υλοποιήσουμε έναν πολλαπλασιαστικό αλγόριθμο $\{R_1, \cdots, R_n\}$ εσωτερικά, δηλαδή μέσω ενός αυτο-επεξεργαστικού προγράμματος.

Πρώτα θα δούμε την υποπερίπτωση της απλής αντιγραφής. Θεωρούμε έναν αλγόριθμο B ο οποίος ορίζεται ως εξής:

$$B(x) = \{x, x\}.$$

Από την βασική αρχή της αυτό-επεξεργασίας έχουμε ότι αφού ο αλγόριθμος B υλοποιείται από κάποιο πρόγραμμα b τότε για ένα πρόγραμμα c που περιέχει το b ως υποπρόγραμμα, δηλαδή είναι της μορφής $c[b]$, η εκτέλεση του b συμβαίνει όταν αυτό είναι ενεργοποιημένο άρα:

$$\textit{self-ed}(c[\overline{b}]) = \textit{alg}(b)(c[\overline{b}]) = B(c[\overline{b}]) = \{c[\overline{b}], c[\overline{b}]\}.$$

Εδώ θα θέλαμε να τονίσουμε ότι αφού η εντολή b παραμένει ενεργοποιημένη, σε καθένα από τους δύο απογόνους, το πρόγραμμα δεν σταματάει, αλλά παράγει ασταμάτητα ένα δέντρο $(x_t)_{t\in T}$ όπου το T είναι το δυαδικό δέντρο. Όπως θα δούμε, πιο περίπλοκοι αυτό-επεξεργαστικοί αλγόριθμοι παράγουν δέντρα όπου το πλήθος των απογόνων δεν είναι σταθερό. Επίσης θα θέλαμε να τονίσουμε ότι δεν μας ενδιαφέρουν πολλαπλασιαστικοί αυτο-επεξεργαστικοί αλγόριθμοι που παράγουν πιστά αντίγραφα, αλλά θέλουμε να παράγονται διαφορετικοί απόγονοι. Κάτι αντίστοιχο συνέβη μάλλον και στη φύση όταν από το απλό μόριο του RNA προέκυψε η σημερινή ποικιλία οργανισμών, η οποία οφείλεται στα «λάθη» που συνέβησαν κατά την αντιγραφή του.

Τώρα θα δούμε την πιο γενική περίπτωση. Έστω $B_1, \cdots, B_n$ αλγόριθμοι. Τότε υπάρχει ένα πρόγραμμα b το οποίο υλοποιεί τον αλγόριθμο $\{B_1, \cdots, B_n\}$. Για ένα πρόγραμμα c που έχει το b ως ενεργοποιημένο υποπόγραμμα, από την βασική αρχή της αυτο-επεξεργασίας έχουμε:

$$\textit{self-ed}(c[\overline{b}]) = \textit{alg}(b)(c[\overline{b}]) = \{B_1, \cdots, B_n\}(c[\overline{b}]) = \{B_1(c[\overline{b}]), \cdots, B_n(c[\overline{b}])\}. \tag{4.2}$$

Αυτό το αποτέλεσμα είναι ιδιαίτερα σημαντικό για εμάς και θα αναφερόμαστε σε αυτό ως **Πολλαπλασιαστική αρχή**. Θα δούμε μερικές ενδιαφέρουσες εφαρμογές της πολλαπλασιαστικής αρχής που θα υποδείξουν τη σημασία της.

Παράδειγμα 1. Θεωρούμε δύο αλγόριθμους B_1, B_2 όπου ο B_1 προσθέτει 1 στην θέση 2 και ο B_2 προσθέτει 2 στη θέση 2. Τους ορίζουμε ως εξής:

$$B_1(c[(2)n]) = c[(2)n + 1]$$

$$B_2(c[(2)n]) = c[(2)n + 2].$$

Θεωρούμε τον αλγόριθμο B τον οποίο ορίζουμε ως $B = \{B_1, B_2\}$. Τότε αν b είναι πρόγραμμα για τον αλγόριθμο B από την 4.2 έχουμε:

$$\textit{self-ed}(c[\overline{b}, (2)0]) = \{B_1, B_2\}(c[\overline{b}, (2)0]) = \{c[\overline{b}, (2)1], c[\overline{b}, (2)2]\}.$$

Φυσικά αυτή η διαδικασία επαναλαμβάνεται και στους απογόνους. Αρκετά αργότερα θα δούμε πως ο αλγόριθμος B μπορεί να μεταβάλλει το πρόγραμμα του ανά τις γενιές, το οποίο θα μας απασχολήσει εκτενώς.

Παράδειγμα 2. Έστω θ_1 και θ_2 δύο διευθύνσεις. Θα συμβολίζουμε την συνένωση τους ως $\widehat{\theta_1\theta_2}$ και θα εννοούμε την θ_2 διεύθυνση του προγράμματος στη θ_1 διεύθυνση. Για παράδειγμα αν $\theta_1 = (1,2)$ και $\theta_2 = (1)$ τότε $\widehat{\theta_1\theta_2} = (1,2,1)$.

Έστω $B(n,\theta)$ η αλγοριθμική συνάρτηση:

$$B(n,\theta) : a[(\theta)k] \to \{\underbrace{a[(\theta)k+1], \cdots, a[(\theta)k+1]}_{n \text{ times}}\}$$

οποία μας δίνει n ίδια προγράμματα στα οποία έχει προστεθεί 1 στην θ διεύθυνση. Έστω b ένα πρόγραμμα για τον αλγόριθμο $B(n,\theta)$. Τότε το πρόγραμμα b σε κάποιο μέρος του πρέπει να περιέχει την πληροφορία ότι ο $B(n,\theta)$ δημιουργεί n αντίγραφα. Άρα ο b είναι της μορφής $b = b[(\theta_1)n][\theta]$. Αυτό υποδεικνύει ότι η πληροφορία για το πλήθος των αντιγράφων βρίσκεται στην διεύθυνση θ_1 του b και η πληροφορία για την διεύθυνση θ του προγράμματος εισόδου που θα του προσθέσουμε 1 βρίσκεται σε κάποιο σημείο του b (γι΄ αυτό δεν υποδεικνύουμε την διεύθυνση που βρίσκεται το θ). Τώρα, αν υποθέσουμε ότι το πρόγραμμα b βρίσκεται στην διεύθυνση θ_2 του προγράμματος c, δηλαδή $c = c[(\theta_2)[b[(\theta_1)n][\theta]]]$, παρατηρούμε ότι η πληροφορία για το πλήθος των αντιγράφων n βρίσκεται στην διεύθυνση $\theta_0 = \widehat{\theta_2\theta_1}$ του προγράμματος c. Για λόγους ευκολίας, θα συμβολίσουμε:

$$n \times c = \underbrace{c, \cdots, c}_{n \text{ times}}$$

και έτσι έχουμε:

$$alg(b[(\theta_1)n][\theta_0])(c[(\theta_2)[b[(\theta_1)n][\theta_0]]]) = \{n \times c[(\theta_2)[b(\theta_1)[n+1][\theta_0]]]\}$$

δηλαδή αν βάλουμε στον B ως θέση που προσθέτει 1 την θέση στην οποία έχει την πληροφορία για το πλήθος των αντιγράφων, τότε, όπως θα δούμε, αν κάνουμε αυτήν την διαδικασία εσωτερικά μπορούμε να δημιουργήσουμε ένα δέντρο στο οποίο το πλήθος των απογόνων δεν είναι σταθερό. Έχουμε:

$$\textit{self-ed}(c[\overline{b[n][\theta_0]}]) = (n \times c[\overline{b[n+1][\theta_0]}])$$

έτσι στην πρώτη γενιά έχουμε n απογόνους, η επόμενη γενιά κάνει $n+1$ απογόνους κλπ. Είναι αρκετά εύκολο, να δει κανείς ότι μπορούμε να έχουμε και έναν αλγόριθμο στον οποίο τα προγράμματα μιας γενιάς θα δίνουν διαφορετικό πλήθος απογόνων. Για παράδειγμα:

$$B(n,\theta) : c[(\theta)k] \to \{\lfloor n/3 \rfloor \times c[k+1], \lfloor n/3 \rfloor \times c[k], \lfloor n/3 \rfloor \times c[k-1]\}$$

είναι ένας αλγόριθμος ο οποίος αν υλοποιηθεί εσωτερικά μας δίνει ένα δέντρο όπου το πλήθος των απογόνων δεν είναι σταθερό ανά γενιά.

Η πολλαπλασιαστική αρχή σε συνδυασμό με την αρχή της αυτο-επεξεργασίας, επάγουν με ένα πολύ φυσικό τρόπο την έννοια του πολλαπλασιαστικού, αυτο-επεξεργαστικού υπολογισμού:

Δεδομένου ενός προγράμματος $c_\emptyset$ (γιατί είναι η ρίζα του γενεαλογικού δέντρου), ένας πολλαπλασιαστικός αυτο-επεξεργαστικός υπολογισμός είναι ένα δέντρο $(c_t)_{t\in T}$ με ρίζα $c_\emptyset$ τέτοιο ώστε οι απόγονοι κάθε c_t να είναι:

$$\textit{im-suc}(c_t) = \textit{self-ed}(c_t) \text{ για κάθε } t \in T.$$

4.4 Αυτο-επεξεργαστικοί υπολογισμοί πλήρους μνήμης

Έστω $(c_t)_{t\in T}$ ένας υπολογισμός. Θα καλούμε ιστορία του c_t για κάποιο t καλούμε την ακολουθία:

$$c_{t_1} \to c_{t_2} \to \cdots \to c_{t_n} = c_t$$

όπου $c_{t_1} = c_\emptyset$ είναι η ρίζα του δέντρου και για κάθε $i < n$, $c_{t_{i+1}}$ είναι απόγονος του c_{t_i}. Αυτή την ακολουθία θα την συμβολίζουμε με $\textit{hist}(c_t)$. Ουσιαστικά σαν ιστορία βλέπουμε το κλαδί στο γενεαλογικό δέντρο το οποίο ξεκινάει από τη ρίζα και φτάνει σε κάποιο κόμβο. Επειδή όπως έχουμε πει, στόχος μας είναι η κατασκευή ενός αλγορίθμου που μπορεί να βλέπει ένα κλαδί και να εξάγει κάποιο πρότυπο, εισάγουμε την έννοια του αυτο-επεξεργαστικού υπολογισμού πλήρους μνήμης ο οποίος είναι ένα δέντρο $(c_t)_{t\in T}$ τέτοιο ώστε για κάθε κάθε πρόγραμμα υπολογίζει τα παιδιά του βασισμένο όχι μόνο στον εαυτό του, αλλά σε όλο το κλαδί από την ρίζα μέχρι τον εαυτό του. Έτσι αν ο αλγόριθμος που είναι υπεύθυνος για τον πολλαπλασιασμό (ο οποίος πλέον δέχεται ως είσοδο όλη την ιστορία) είναι ο B τότε:

$$\textit{im-suc}(c_t) = B(\textit{hist}(c_t)) \text{ για κάθε } t \in T.$$

Είναι πολύ εύκολο να δει κανείς ότι τα αποτελέσματα που έχουμε δει ως τώρα μεταφέρονται με φυσικό τρόπο σε υπολογισμούς πλήρους μνήμης. Δεν θα δέλαμε όμως να αναλωθούμε σε αυτό, καθώς όπως θα δούμε, παρά το γεγονός ότι ένας αλγόριθμος πλήρους μνήμης μοιάζει ισχυρότερος στην πραγματικότητα δεν είναι, γιατί οπώς έχουμε ήδη μαρτηρήσει αντί κανείς να βλέπει όλο το γενεαλογικό κλαδί, μπορεί να βλέπει απλώς το τελευταίο πρόγραμμα δεδομένου ότι αυτό αποθηκεύει την προηγούμενή του κατάσταση σε κάθε υπολογιστικό βήμα.

Για να το δούμε αυτό καλύτερα, θεωρούμε μια απεικόνιση ϕ από ακολουθίες προγραμμάτων σε προγράμματα, ως εξής:

$$\phi : (c_1, \cdots, c_n) \to [c_1, \cdots, c_n].$$

Να τονίσουμε ότι η ϕ είναι υπολογίσιμη και αντιστρέψιμη καθώς και ότι παρόλο που δεν ελαττώνει την χωρική πολυπλοκότητα (space-complexity), μας εξυπηρετεί πάρα πολύ σε αυτή την θεωρητική προσέγγιση. Από εδώ και στο εξής θα συμβολίζουμε με $H(c_n) = \phi(\textit{hist}(c_n))$ και

θα εννοούμε το πρόγραμμα $[c_1, \cdots, c_n]$ του οποίου η παρούσα κατάσταση θα βρίσκεται πάντα στην τελευταία θέση. Να σημειωθεί ότι με την αντικατάσταση ενός προγράμματος x από το $\phi(hist(x))$ υποθέτουμε ότι όλες οι διευθύνσεις, εκτός της τελευταίας, είναι ανενεργές ακόμη και αν περιέχουν ενεργοποιημένες συνιστώσες. Ο λόγος που το κάνουμε αυτό είναι ότι αλλιώς θα καταλήγαμε τετριμμένα να θυμόμαστε ολόκληρο το δέντρο αντί για το κλαδί που μας ενδιαφέρει. Με αυτή τη σύμβαση κατά νου, φτάνουμε στον παρακάτω ορισμό:

Ένας αυτο-επεξεργαστικός υπολογισμό θα λέμε ότι κρατάει μνήμη εάν είναι της μορφής:

$$[c_1], [c_1, c_2], \cdots, [c_1, c_2, \cdots, c_n].$$

Σε αυτό το σημείο βλέπουμε ότι για να αντικαταστήσουμε του υπολογισμούς πλήρους μνήμης με προγράμματα που κρατάνε μνήμη αρκεί να πούμε το εξής:

Αν $[c_1, \cdots, c_n]$ είναι η είσοδος, εκτελούμε το c_n με είσοδο την ακολουθία $c_1, \cdots, c_n$ και προσθέτουμε στο τέλος του $[c_1. \cdots, c_n]$ το αποτέλεσμα. Ακόμα πιο συγκεκριμένα, έχουμε:

Αν m είναι μια εντολή για την παραπάνω διαδικασία τότε:

$$\textit{self-ed}([c_1, \cdots, c_n[\overline{m}]) = [c_1, \cdots, c_n, \textit{alg}(m)([c_1, \cdots, c_n])]$$

όπου το $\textit{alg}(m)([c_1, \cdots, c_n])$ είναι το c_{n+1}.

Παρά το γεγονός ότι η παραπάνω διαδικασία φαίνεται απλή, εμπεριέχει αρκετές τεχνικές δυσκολίες. Από την άλλη μεριά, η ανάλυση και η επίλυση αυτών των δυσκολιών επιδεικνύει την ευελιξία των αυτο-επεξεργαστικών προγραμμάτων. Ας δούμε μερικές από αυτές:

1) Αρχικά πρέπει το πρόγραμμα m στο $c_n[\overline{m}]$ να έχει μεγαλύτερη προτεραιότητα από τις υπόλοιπες ενεργοποιημένες εντολές του c_n. Αυτό πρέπει να γίνει, γιατί οι ενεργοποιημένες εντολές στο c_n που μπορεί να μεταβάλλουν την είσοδο $c_1, \cdots, c_n$. Έτσι πρέπει να χρησιμοποιήσουμε κάποια έννοια προτεραιότητας στις εντολές που εκτελούνται.

2) Η δεύτερη τεχνική δυσκολία προκύπτει από το γεγονός ότι πρέπει να βρούμε μια διεύθυνση για το m που δεν χρησιμοποιείται ή δημιουργείται κατά την υπολογιστική ακολουθία. Η εύκολη λύση που χρησιμοποιούμε εδώ είναι να προσθέσουμε μια διάσταση στη δομή του προγράμματος μετατρέποντας το $c_n[\overline{m}]$ στο $[[c_n], \overline{m}]$.

Να σημειώσουμε εδώ ότι την έννοια της προτεραιότητας την συναντάμε και στη βιολογία, όπου ένα γονίδιο μπορεί να σταματήσει την έκφανση κάποιου άλλου, όπως και στη νευροεπιστήμη, όπου ένας νευρώνας μπορεί να σταματήσει την πυροδότηση κάποιου άλλου. Εδώ θα

μοντελοποιήσουμε την έννοια της προτεραιότητας υποθέτοντας ότι σε κάποια σχετική διεύθυνση μιας εντολής, θ_p βρίσκεται ένας φυσικός αριθμός που προσδιορίζει την «δύναμη» μιας εντολής ως προς τις άλλες.
Θα χρησιμοποιήσουμε το συμβολισμό $c[(+)\overline{m}]$ για να δηλώσουμε ότι επεκτείνουμε το πρόγραμμα c με μια διεύθυνση στην οποία περιέχεται το $\overline{m}$.

Λήμμα της Μνήμης.

Έστω $(c_t)_{t\in T}$ ένας αυτο-επεξεργαστικός υπολογισμός πλήρους μνήμης. Τότε υπάρχει ένα πρόγραμμα m τέτοιο ώστε το πρόγραμμα $c_\emptyset[(+)\overline{m}]$ να δίνει τον υπολογισμό $(x_t)_{t\in T}$, όπου για κάθε $t \in T$, αν $c_1, \cdots, c_n = c_t$ είναι η ιστορία του c_t στο $(c_t)_{t\in T}$, τότε

$$x_t = [c_1[(+)\overline{m}], \cdots, c_n[(+)\overline{m}]].$$

Απόδειξη. Έστω m ένα πρόγραμμα για τον ακόλουθο αλγόριθμο:

1. Έστω x η είσοδος. (Θεωρούμε ότι το x είναι της μορφής $[[[c_1], \overline{m}], \cdots, [[c_n], \overline{m}]]$ και ότι η προτεραιότητα του m στα προγράμματα υπερβαίνει το αντίστοιχο c_i).
2. Σταμάτα την εκτέλεση οποιασδήποτε εντολής στη διεύθυνση (last,1,1).
3. Έστω n το μήκος του x.
4. Έστω z_i το πρόγραμμα στη διεύθυνση $(i, 1, 1)$ του x, για κάθε $i = 1, \cdots, n$.
5. Έστω $\{w_1, \cdots, w_s\}$ το αποτέλεσμα τη εκτέλεσης του z_n με είσοδο $z_1, \cdots, z_n$.
6. Έστω y το πρόγραμμα στη διεύθυνση (last, 2)της εισόδου.
7. Για $j = 1, \cdots, s$:
   (α′) Θέσε την προτεραιότητα του y αυστηρά μεγαλύτερη από κάθε ενεργοποιημένη εντολή στο πρόγραμμα w_j.
   (β′) Πρόσθεσε $[[w_j], \overline{y}]$ στο τέλος της εισόδου x και δώσε το αποτέλεσμα στην έξοδο.

Τώρα θέτουμε την προτεραιότητα του m να είναι αυστηρά μεγαλύτερη από όλες τις ενεργοποιημένες εντολές στο πρόγραμμα $c_\emptyset$ και θέτουμε $x_\emptyset = [[c_\emptyset], \overline{m}]$. Αφού φέραμε το x στην κατάλληλη μορφή, προκύπτει επαγωγικά το ζητούμενο από την εκτέλεση του αλγορίθμου που περιγράψαμε.

4.5 Πιθανές αποφάσεις ενός αυτο-επεξεργαστικού προγράμματος

Τώρα θα δούμε πως ένα αυτο-επεξεργαστικό πρόγραμμα μπορεί να διαχειρίζεται αποφάσεις. Έστω r και c προγράμματα, θεωρούμε την αλγοριθμική συνάρτηση $B(r)$, την οποία ορίζουμε ως εξής:

$$B(r) : c \to \begin{cases} [c_1. \cdots, c_n, r] & \text{if } c = [c_1, \cdots, c_n], \\ [c, r] & \text{if not} \end{cases} \tag{4.3}$$

η οποία απλώς παραθέτει στο τέλος του c το r. Αν $b = b[r]$ είναι ένα πρόγραμμα για τον αλγόριθμο $B(r)$ και θεωρήσουμε ένα πρόγραμμα $c = c[\overline{b}]$ τότε από την βασική αρχή της αυτο-επεξεργασίας έχουμε *self-ed*$(c[\overline{b}]) = B(r)(c[\overline{b}])$. Επειδή η αλγοριθμική συνάρτηση B «επεκτείνει» το πρόγραμμα c, χρησιμοποιεί μια καινούργια διεύθυνση. Μια τέτοια διεύθυνση θα λέμε ότι είναι διαθέσιμη διεύθυνση και θα γράφουμε $c = c[(+\theta)s]$, υποδηλώνοντας ότι το πρόγραμμα c επεκτείνεται με μια διεύθυνση θ στην οποία τοποθετούμε το πρόγραμμα s.

Προσωρινές αποφάσεις. Έστω r και c προγράμματα και θ μια διαθέσιμη διεύθυνση στο c. Έστω επίσης $s = s[r]$ ένα πρόγραμμα για τον αλγόριθμο:

1. Έστω x_1 η είσοδος.
2. Έστω x_2 το αποτέλεσμα της διαγραφής της διεύθυνσης θ από το x_1.
3. Έξοδος *alg*$(r)(x_2)$.

Μπορούμε εύκολα να δούμε ότι το αυτο-επεξεργαστικό βήμα του $c[(+\theta)\overline{s}]$ είναι *self-ed*$(c[(+\theta)\overline{s}]) =$ *alg*$(r)(c)$. Ουσιαστικά αυτό που συμβαίνει είναι ότι καθώς τρέχει το πρόγραμμα r διαγραφεί τον εαυτό του. Ένα τέτοιο πρόγραμμα θα το λέμε **προσωρινά διαφοροποιητικό** πρόγραμμα.

Μόνιμες αποφάσεις. Στην πρώτη περίπτωση η απόφαση (r) αποθηκεύτηκε στο τέλος του προγράμματος, στη δεύτερη περίπτωση η προσωρινή απόφαση (r) έδρασε στο πρόγραμμα και ύστερα διέγραψε τον εαυτό της, τώρα θα δούμε μόνιμες αποφάσεις.

Έστω r και c προγράμματα και θ μια διαθέσιμη διεύθυνση στο c. Ορίζουμε $s = s[r]$ να είναι το πρόγραμμα για τον εξής αλγόριθμο:

1. Έστω x_1 η είσοδος.
2. Έστω x_2 το αποτέλεσμα του r με είσοδο το x_1.
3. Ενεργοποίησε το πρόγραμμα στη διεύθυνση θ του x_2 και δώσε το αποτέλεσμα στην έξοδο.

Όπως και πριν, κοιτάζοντας το αυτο-επεξεργαστικό βήμα του $c[(+\theta)\overline{s}]$ παίρνουμε το πρόγραμμα $alg(r)(c[(+\theta)\overline{s}])$ με ενεργοποιημένη τη θ διεύθυνση. Αν θεωρήσουμε ότι ο $alg(r)$, σε κάθε υπολογιστικό βήμα για το οποίο είναι υπεύθυνος, δεν μεταβάλλει την διεύθυνση θ, τότε το αποτέλεσμα του αυτο-επεξεργαστικού βήματος θα είναι της μορφής $c_1[(\theta)\overline{s}]$, για κάποιο πρόγραμμα c_1. Έναν αλγόριθμο σαν τον $alg(r)$ θα τον λέμε θ-ευσταθή. Μπορούμε εύκολα να δούμε, ότι από επαγωγή ένας απόγονος $c_i \in hist(c[(+\theta)\overline{s}])$ θα είναι της μορφής $(alg(r))^i(c(+\theta)\overline{s})$. Το πρόγραμμα r θα λέμε ότι είναι ένα **μόνιμα διαφοροποιητικό** πρόγραμμα.

Θα θέλαμε να τονίσουμε παρόλο που δεν υπάρχει ιδιαίτερη διαφορά μεταξύ μιας προσωρινής απόφασης και μιας μόνιμης, αφού η πρώτη μπορεί να επαναλαμβάνεται συνεχώς, ενώ η δεύτερη να διακόπτεται από ένα πρόγραμμα μεγαλύτερης προτεραιότητας, πρακτικά υπάρχει μεγάλη διαφορά διότι η επαναλαμβανόμενη προσωρινή απόφαση επιβαρύνεται με το κόστος επιλογής της.

ϕ-Διαφοροποιητικές αποφάσεις. Έστω ένα αυτο-επεξεργαστικό πρόγραμμα c, μια διεύθυνση ϕ του c και ένα διαφοροποιητικό πρόγραμμα r. Μια απόφαση της οποία το προσωρινό (αντ. μόνιμο) διαφοροποιητικό πρόγραμμα είναι το πρόγραμμα $s = s[\phi, r]$ της ακόλουθης αλγοριθμικής συνάρτησης:

$$c = c[(\phi)y] \rightarrow c[(\phi)alg(r)(y)]$$

που αντικαθιστά το περιεχόμενο y της ϕ με το $alg(r)(y)$, θα καλείται προσωρινή (αντ. μόνιμη) απόφαση που διαφοροποιεί την ϕ με το r. Η διεύθυνση του r στο c θα καλείται διεύθυνση του διαφοροποιητικού προγράμματος του ϕ.

4.6 Διαγωνιοποίηση

Τώρα θα αναλύσουμε την κεντρική ιδέα αυτής εργασίας, τη διαγωνιοποίηση. Όπως έχουμε αναφέρει, στόχος είναι να βρούμε μια αλγοριθμική συνάρτηση η οποία να βελτιώνει όχι μόνο το πρόγραμμά μας, αλλά και τον εαυτό της.

Ο συνδυασμός της αυτο-επεξεργαστικής αρχής πλήρους μνήμης και του Λήμματος της Μνήμης, μας επιτρέπουν να θεωρήσουμε ένα αυτο-επεξεργαστικό πρόγραμμα που κρατάει μνήμη ως ένα αυτο-επεξεργαστικό πρόγραμμα πλήρους μνήμης. Θα δουλέψουμε με προγράμματα πλήρους μνήμης γιατί είναι ευκολότερο, αλλά είναι σημαντικό να έχουμε στο πίσω μέρος του μυαλού μας ότι μπορούμε να τα σκεφτόμαστε και σαν προγράμματα που κρατάνε μνήμη.

Έστω ότι, κοιτάζοντας την ιστορία $c_1, \cdots, c_n$ ενός αυτο-επεξεργαστικού προγράμματος, αποφασίζουμε να αλλάξουμε το c_n με κάποιον αλγοριθμικό τρόπο. Τότε η ίδια απόφαση μπορεί ενδεχομένως να γίνει από το c_n, δεδομένου ότι ο υπολογισμός είναι αυτο-επεξεργαστικός και ότι το πρόγραμμα κρατάει μνήμη.

Θα μιλήσουμε για ακολουθίες που επιβιώνουν και για ακολουθίες που είναι επιτυχημένες. Μια ακολουθία θα λέμε ότι επιβιώνει όταν είναι το αποτέλεσμα επιλογής (φυσικής ή τεχνητής) σε ένα δέντρο αυτο-επεξεργαστικών προγραμμάτων, ενώ θα λέμε ότι είναι επιτυχημένη αν είναι το αποτέλεσμα μιας εσωτερικής επιλογής επιτυχημένων υπολογισμών.

Ας αναρωτηθούμε το εξής:
Έστω ότι έχουμε μια ακολουθία που είτε επιβιώνει είτε είναι επιτυχημένη. Τι θα κάναμε για να βοηθήσουμε την εξέλιξη ενός τέτοιου προγράμματος, δεδομένου ότι δεν γνωρίζουμε με ποια κριτήρια επιλέχθηκε η ακολουθία και ποια είναι η λειτουργία του προγράμματος; Η δική μας απάντηση είναι ότι πρέπει να ψάξουμε για μοτίβα στο πρόγραμμα και στις αλλαγές του και αν καταφέρουμε να βρούμε κάποια, να προσπαθήσουμε να τα διαιωνίσουμε.
Πιο αυστηρά θα λέμε ότι:

Δεδομένου ότι έχουμε μια ακολουθία (επιτυχημένη ή που επιβιώνει) $c_1 \to c_2 \to \cdots \to c_n$ ένα μοτίβο θα είναι ένας αλγόριθμος R, τέτοιος ώστε για κάθε $k < n$, $R(c_k) \sqsubseteq c_{k+1}$.

Παράδειγμα. Έστω ότι παρατηρούμε σε μια ακολουθία (επιτυχημένη ή που επιβιώνει) ότι σε κάποιο μέρος του προγράμματος μια παράμετρος (φυσικός αριθμός), σε κάθε μετάβαση αυξάνεται κατά 1. Αν αυτή η παράμετρος βρίσκεται στη θ διεύθυνση, τότε ορίζουμε τον αλγόριθμο R ως εξής:

$$R(x[(\theta)n]) = x[(\theta)n + 1].$$

Παρατηρούμε ότι ένα τέτοιος αλγόριθμος ταιριάζει στην ακολουθία. Η βασική ιδέα αυτής της

εργασίας είναι ότι δεδομένης μια ακολουθίας (επιτυχημένης ή που επιβιώνει) προγραμμάτων:

$$c_1[\delta_1], c_2[\delta_2], \cdots, c_n[\overline{\delta_n}]$$

αν Δ είναι ένας αλγόριθμος που αναγνωρίζει πρότυπα (pattern-recognition algorithm) τέτοιος ώστε $\Delta = alg(\delta_1)$ τότε αυτός βελτιώνει και το πρόγραμμα c_i αλλά και τον εαυτό του δ_i αφού η ακολουθία είναι επιτυχημένη ή επιβιώνει. Ουσιαστικά θεωρούμε ότι ο Δ και οι μετέπειτα εκδοχές του σταδιακά ενσαρκώνουν όλο και περισσότερο την δομή του περιβάλλοντος. Φυσικά αυτή η ιδέα είναι αρκετά θεωρητική και με πολλές τεχνικές δυσκολίες, αλλά θεωρούμε ότι είναι προς τη σωστή κατεύθυνση για την προσέγγιση μιας γενικής ευφυίας (general intelligence). Θα αναλύσουμε αρκετά τη διαγωνιοποίηση με την ελπίδα να ενθαρρύνουμε τις προσπάθειες προσέγγισης του ζητήματος από αυτήν τη σκοπιά, μιας και θεωρούμε ότι υπάρχει στο πίσω μέρος του μυαλού πολλών ερευνητών.

Ένα μεγάλο πρόβλημα που έχουμε εδώ, είναι η πολυπλοκότητα του αλγορίθμου Δ ώστε να δουλεύει σωστά, δηλαδή να βρίσκει σωστά μοτίβα αλλά και να παράγει προγράμματα με σωστή σύνταξη. Η πρόταση μας είναι ένας απλός αλγόριθμος ο οποίος να εξελίσσει τον εαυτό του. Αν θεωρήσουμε ότι η ακολουθία είναι πετυχήμενη ή επιβιώνει τότε ο αλγόριθμος πρέπει να βελτιώνεται. Ουσιαστικά θα πρέπει να αναγνωρίζει εξελικτικά μοτίβα όχι μόνο στο πρόγραμμα c αλλά και στο πρόγραμμά του δ.

Αλγόριθμοι διαγωνιοποίησης.

Θα χρησιμοποιούμε τον όρο σύστημα απόφασης για έναν αλγόριθμο Δ με την εξής γενική περιγραφή:

Ο Δ χρησιμοποιείται για να πάρει μια απόφαση D και αποτελείται από δύο μέρη, τον ευρετή (searcher) Δ_s και τον ελεγκτή (tester) Δ_t. Ο searcher Δ_s, θεωρούμε ότι προτείνει προγράμματα $r_1, r_2, \cdots$ στη σειρά. Θα αναφερόμαστε σε αυτά τα προγράμματα ως προτεινόμενα προγράμματα. Το σύνολο των προτεινόμενων προγραμμάτων μπορεί να είναι είτε πεπερασμένο, είτε άπειρο. Στην περίπτωση που είναι άπειρο, υποθέτουμε ότι ο ευρετής κατασκευάζει τα προγράμματα $r_1, r_2, \cdots$ ως συνθέσεις προγραμμάτων ενός πεπερασμένου συνόλου εντολών/προγραμμάτων. Η προτεραιότητα ενός προτεινόμενου προγράμματος καθορίζει την εμφάνιση ενός προγράμματος στην ακολουθία $r_1, r_2, \cdots$ και θα λέμε ότι ένα πρόγραμμα είναι απλό αν έχει μεγάλη προτεραιότητα, δηλαδή εμφανίζεται νωρίς στην ακολουθία. Ο tester Δ_t, υποθέτουμε ότι χρησιμοποιεί έναν αλγοριθμικό έλεγχο T με τον οποίο επιλέγει το απλούστερο προτεινόμενο πρόγραμμα r έτσι ώστε $T(r) = true$. Θα καλούμε T τον ελεγκτικό αλγόριθμο του Δ_t.

Έστω τώρα ένας αλγόριθμος S που δέχεται ως είσοδο την ιστορία ενός αυτο-επεξεργαστικού προγράμματος c και δίνει ως έξοδο μια υπακολουθία $c_1, c_2, \cdots, c_n = c$, μπορούμε να θεωρήσουμε τον εξής ελεγκτικό αλγόριθμο T:

Έστω $c_1, \cdots, c_n$ να είναι η έξοδος του S. Επιλέγουμε το απλούστερο πρόγραμμα r από την ακολουθία προτεινόμενων προγραμμάτων $r_1, r_2, \cdots$ που προτείνει ο Δ_s τέτοιο ώστε να ταιριάζει στην ακολουθία $c_1, \cdots, c_n$, δηλαδή για κάθε $i < n$, $alg(r)(c_i) = c_{i+1}$ και το χρησιμοποιούμε ως προσωρινό ή μόνιμο διαφοροποιητικό πρόγραμμα.

Θα χρησιμοποιούμε τον όρο ακολουθιακή διαγωνιοποίηση για να περιγράψουμε τον τρόπο λειτουργίας ενός συστήματος απόφασης $\Delta = (\Delta_s, \Delta_t)$, τέτοιο ώστε το Δ_t να χρησιμοποιεί το πρόγραμμα ελέγχου T που περιγράψαμε παραπάνω. Η ακολουθία $c_1, \cdots, c_n$ θα καλείται η ελεγκτική ακολουθία της διαγωνιοποίησης.

Αρχή της ακολουθιακής διαγωνιοποίησης (Απλή μορφή)

Έστω ότι Δ είναι ένας αλγόριθμος ακολουθιακής διαγωνιοποίησης με πρόγραμμα $\delta = \delta_n$ και $c = c[\overline{\delta}]$ ένα αυτο-επεξεργαστικό πρόγραμμα. Έστω ότι:

$$c_1[\delta_1], c_2[\delta_2], \cdots, c_n[\overline{\delta_n}] = c[\overline{\delta}]$$

είναι η ιστορία του c. Τότε:

$$\textit{self-ed}(c[\overline{\delta}]) = \Delta(c_1[\delta_1], c_2[\delta_2], \cdots, c_n[\overline{\delta_n}]).$$

Αυτό το αποτέλεσμα αν και είναι εξαιρετικά απλό αφού αποτελεί άμεση εφαρμογή της Βασικής Αρχής της Αυτο-επεξεργασίας και του Λήμματος της Μνήμης, μας λέει κάτι πολύ ενδιαφέρον: ότι το πρόγραμμά μας μπορεί να εντοπίσει εξελικτικά μοτίβα στον εαυτό του και να τα διαιωνίσει. Προτού προχωρήσουμε, ας δούμε μερικά παραδείγματα χρήσης της διαγωνιοποίησης. Σε αυτά θα ζητήσουμε από ένα αυτό-επεξεργαστικό πρόγραμμα να συμπληρώσει κάποιες δοσμένες ακολουθίες. Θα θεωρήσουμε ότι τα γνωστά μέρη των ακολουθιών τα έχει βρει το πρόγραμμα και θα αναφερόμαστε σε αυτό το γεγονός ως βασική σύμβαση.

Παράδειγμα 1. Σε αυτό το αρχικό παράδειγμα, ζητάμε από το πρόγραμμα να συμπληρώσει την ακολουθία $0, 1, 2, \cdots$. Θα θεωρήσουμε ότι θ_e είναι η διεύθυνση του προγράμματος όπου δίνει έξοδο στο περιβάλλον. Έτσι λοιπόν από την βασική σύμβαση, θα ισχύει ότι για $i = 0, 1, 2, c_i = c_i[(\theta_e)i]$. Έτσι το c_0 γράφει 0 στη θ_e, το c_1 γράφει 1 στη θ_e και το c_2 γράφει 2 στη θ_e. Σε αυτό το σημείο είναι εύκολο να δούμε ότι το πρόγραμμα $\mathrm{add}(1)$ που προσθέτει 1 σε κάποιον ακέραιο είναι θ_e-διαφοροποιητική απόφαση που ταιριάζει στην ακολουθία c_0, c_1, c_2. Έτσι αν θεωρήσουμε ότι είναι αρκετά απλό πρόγραμμα θα πρέπει να χρησιμοποιηθεί ως προσωρινό ή μόνιμο θ_e-διαφοροποιητικό πρόγραμμα. Συνεπώς αν η διαγωνιοποίηση είναι ενεργοποιημένη στο c_2 το αυτο-επεξεργαστικό βήμα $c_2 \to c_3$ θα πρέπει να δώσει $c_3 = c_3[(\theta_e)3]$ που σημαίνει ότι το c_3 δίνει 3 στο περιβάλλον. Εύκολα βλέπουμε, ότι επαγωγικά συμπληρώνεται ορθά η ζητούμενη ακολουθία.

Παράδειγμα 2. Σε αυτό το δεύτερο παράδειγμα θέλουμε να μελετήσουμε την συμπεριφορά ενός αυτο-επεξεργαστικού προγράμματος με διαγωνιοποίηση σε μια συνθετότερη ακολουθία, η οποία είναι η εξής:

$$(0, 1, 2, \cdots), (0, 2, 4, \cdots), (0, 3, 6, \cdots), \cdots .$$

Ουσιαστικά ζητάμε από το πρόγραμμα όχι απλώς να συμπληρώσει τις πρώτες τρεις ακολουθίες, αλλά και να μαντέψει ποια θα είναι η επόμενη ακολουθία. Αρχικά παρατηρούμε ότι ένα υπο-πρόβλημα οριοθετείται από παρενθέσεις, πιο συγκεκριμένα κάθε υπο-πρόβλημα ξεκινάει με μια αριστερή παρένθεση και ένα 0 και κλείνει με μια δεξιά παρένθεση. Παρατηρούμε ακόμη ότι και εμείς για να αντιληφθούμε το τέλος ενός υπο-προβλήματος πρέπει να μας το πει το περιβάλλον. Έτσι ορίζουμε τον αλγόριθμο r ως εξής:

Αν στην διεύθυνση θ_e υπάρχει δεξιά παρένθεση, τότε στο επόμενο βήμα βάλε μηδέν στην θ_e. Ουσιαστικά το πρόγραμμα μαντέυει τα στοιχεία της ακολουθίας και όταν του δίνουμε ′)′ ξεκινάει το επόμενο υπο-πρόβλημα. Προφανώς ένα πρόγραμμα για τον αλγόριθμο r ταιριάζει στην ακολουθία αφού αυτή η διαδικασία έχει ακολουθηθεί μέχρι στιγμής. Τώρα παρατηρούμε ότι για $i = 0, 1, 2$ κατά την διάρκεια του $i-$στου υπο-προβλήματος (που έχει επιλυθεί στο παράδειγμα 1) η διεύθυνση, ας πούμε θ του θ_e-διαφοροποιητικού προγράμματος περιέχει το πρόγραμμα $add(i)$. Συνεπώς αν θ_0 είναι η σχετική διεύθυνση του i στο $add(i)$, τότε το πρόγραμμα που υλοποιεί τον αλγόριθμο:

Σε κάθε νέο υπο-πρόβλημα πρόσθεσε 1 στη διεύθυνση $\widehat{\theta\theta_0}$,

ταιριάζει στην ακολουθία, έτσι ώστε αν ένα πρόγραμμα για τον αλγόριθμο r είναι απλό, από διαγωνιοποίηση θα χρησιμοποιηθεί ως διαφοροποιητικό πρόγραμμα. Έτσι ένα αυτο-επεξεργαστικό πρόγραμμα πρέπει να μπορεί να μαντέψει ότι στο τέταρτο υπο-πρόβλημα τα στοιχεία της ακολουθίας ξεκινάνε με το μηδέν και στη συνέχεια αυξάνονται κατά 4.

Παρατήρηση 1. Βλέπουμε ότι η επιλογή της διεύθυνσης που δίνει έξοδο στο περιβάλλον δεν επηρεάζει την ροή των προβλημάτων και θα μπορούσε να αντικατασταθεί από οποιαδήποτε διεύθυνση του προγράμματος. Μάλιστα και τα δύο προβλήματα υποδεικνύουν ότι ένα αυτο-επεξεργαστικό πρόγραμμα μπορεί να βελτιώνει τις παραμέτρους του με βάση κάποια αρχική κρίση ότι κάτι είναι επιτυχημένο.

Παρατήρηση 2. Επιπλέον παρατηρούμε ότι ένα αυτο-επεξεργαστικό πρόγραμμα μπορεί να επιλέξει μια ενέργεια ανάλογα με το μήκος της μνήμης στο οποίο αυτή ταιριάζει.
Έστω μια απόφαση $D(r)$, με πρόγραμμα $d[r]$, σχετική με το πρόγραμμα r που έχει προτείνει ο searcher. Τότε ένα αυτο-επεξεργαστικό πρόγραμμα μπορεί τυχαία να επιλέξει να εφαρμόσει την $D(r)$ ενεργοποιώντας το πρόγραμμά της $d[r]$. Υποθέτουμε επίσης, ότι για να είναι η απόφαση επιτυχημένη, πρέπει το πρόγραμμα r να ταιριάζει τουλάχιστον σε k_d μήκος πρόσφατης μνήμης. Γνωρίζουμε ότι, αφού η ακολουθία $c_1, \cdots, c_n$ είναι επιτυχημένη, πρέπει για την απόφαση $D(r)$,

το πρόγραμμα r να ταιριάζει σε τουλάχιστον k_d μήκος πρόσφατης μνήμης.
Συνεπώς το πρόγραμμα:

Ενεργοποίησε το πρόγραμμα $d[r]$ όποτε το προτεινόμενο πρόγραμμα r ταιριάζει σε τουλάχιστον k_d βήματα πρόσφατης μνήμης.

ταιριάζει στην ακολουθία $c_1, \cdots, c_n$ και αν επιπλέον θεωρήσουμε ότι είναι αρκετά απλό, πρέπει να καθιερωθεί από την διαγωνιοποίηση.

Παρατήρηση 3. Τέλος, θα θέλαμε να σημειωθεί, ότι αν ένα αυτο-επεξεργαστικό πρόγραμμα είναι εφοδιασμένο με μια «εσωτερική αναπαράσταση του περιβάλλοντος», τότε είναι δυνατό μέσω της διαγωνιοποίησης να προβλέψει ομαλές μεταβολές σε αυτό.

4.7 Άλλες μορφές διαγωνιοποίησης

Στατιστική διαγωνιοποίηση (ή στατιστικό διαγώνιο).

Επιστρέφοντας για μια ακόμη φορά στην βάση της ιδέας μας, σκεφτόμαστε πώς μπορούμε να βοηθήσουμε μια επιτυχημένη ακολουθία $c_1, \cdots, c_n$ δεδομένου ότι δεν γνωρίζουμε τίποτα για το περιβάλλον. Όπως έχουμε δει, θα αναζητήσουμε επαναλαμβανόμενα μοτίβα στην ακολουθία με στόχο να τα διαιωνίσουμε. Στην μέχρι τώρα προσέγγισή μας ψάχναμε ένα πρόγραμμα το οποίο έπρεπε να ταιριάζει σε όλη την ακολουθία, δηλαδή σε κάθε μετάβαση $c_i \to c_{i+1}$, για $i < n$. Τώρα θα «χαλαρώσουμε» λίγο την συνθήκη, απαιτώντας το πρόγραμμα που ψάχνουμε να ταιριάζει σε ένα ποσοστό των μεταβάσεων. Αυτή η εκδοχή της διαγωνιοποίησης αποκτά ιδιαίτερο ενδιαφέρον στην περίπτωση που έχουμε πολλαπλασιαστικό πρόγραμμα γιατί τότε μπορούμε να χρησιμοποιήσουμε το ποσοστό αποδοχής ως πιθανότητα οι απόγονοι να κληρονομήσουν κάποιο χαρακτηριστικό.

Για παράδειγμα, έστω ότι σε μια επιτυχημένη ακολουθία παρατηρήσουμε ότι ένα πρόγραμμα ταιριάζει στις μισές μεταβάσεις. Τότε μπορούμε να συμβουλέψουμε το πρόγραμμά μας να δώσει το πρόγραμμα αυτό με πιθανότητα $1/2$ σε κάθε του απόγονο. Παρακάτω θα δούμε πιο αναλυτικά αυτή την στατιστική εκδοχή της διαγωνιοποίησης.

Έστω μια επιτυχημένη ακολουθία $c_1, \cdots, c_n$ και ένας searcher Δ_s, μπορούμε να μεταβάλλουμε την λειτουργία του tester ώστε να ελέγχει την σχετική συχνότητα με την οποία ένα προτεινόμενο πρόγραμμα r ταιριάζει στις μεταβάσεις $c_i \to c_{i+1}, i < n$ της ακολουθίας και να αποφασίζει να χρησιμοποιήσει το r με την ίδια σχετική συχνότητα ως διαφοροποιητικό πρόγραμμα. Αν δ είναι το πρόγραμμα που περιγράφει την παραπάνω διαδικασία και είναι ενεργοποιημένο στο $c_n = c_n[\overline{\delta}]$, τότε από την βασική αρχή της αυτο-επεξεργασίας και το λήμμα της μνήμης, το

c_n μπορεί να εκτελέσει τον ίδιο αλγόριθμο και να πάρει την ίδια απόφαση.

Παρατηρούμε ότι, όπως και στην κλασσική μορφή της διαγωνιοποίησης, αποφασίζεται αν το r θα χρησιμοποιηθεί είτε ως μόνιμο είτε ως προσωρινό διαφοροποιητικό πρόγραμμα, είτε ως ϕ-διαφοροποιητικό πρόγραμμα για κάποια διεύθυνση ϕ. Και πάλι το μήκος της πρόσφατης μνήμης στην οποία το r ταιριάζει με την αντίστοιχη σχετική συχνότητα, παίζει καθοριστικό ρόλο για το αν το r θα είναι μόνιμη ή προσωρινή συμπεριφορά. Αξίζει να σημειωθεί ότι το στατιστικό διαγώνιο που περιγράψαμε περιέχει την κλασσική μορφή της διαγωνιοποίησης και ταυτόχρονα είναι πολύ πιο πρακτικό.

Παράλληλη διαγωνιοποίηση (ή παράλληλο διαγώνιο).

Ξεκινώντας και πάλι από την σκέψη ότι δεδομένου ενός προγράμματος $c = c_n$ και μιας ακολουθίας $c_1, \cdots, c_n$ που επιβιώνει, τι πρέπει να κάνουμε για να βοηθήσουμε αυτό το πρόγραμμα. Θα θεωρήσουμε επιπλέον ότι σε κάθε πρόγραμμα υπάρχουν δύο διευθύνσεις θ_i και θ_o στις οποίες αναγράφονται η είσοδος και η έξοδος από και προς το περιβάλλον αντίστοιχα. Υποθέτουμε επίσης ότι είναι αρκετά απλή η απεικόνιση R της εισόδου στην έξοδο και ότι είναι χρονικά ανεξάρτητη. Από την υπόθεση ότι η ακολουθία $c_1, \cdots, c_n$ επιβιώνει, έπεται ότι για κάθε $k < n$ το περιεχόμενο στην διεύθυνση εισόδου θ_i σχετίζεται με το περιεχόμενο στην διεύθυνση εξόδου θ_o του επόμενου, δηλαδή του c_{k+1}. Θα συμβολίζουμε με $a \restriction \theta$ την τιμή του περιεχομένου του c στη διεύθυνση θ.

Έτσι από τα προηγούμενα έχουμε ότι:

$$R(c_k \restriction \theta_o) = c_{k+1} \restriction \theta_o,$$

για κάθε $k < n$. Άρα αν το r είναι πρόγραμμα για τον παρακάτω αλγόριθμο:

Δώσε έξοδο στο περιβάλλον το αποτέλεσμα του υπολογισμού $R(i)$, όπου i είναι το περιεχόμενου της διεύθυνσης εισόδου,

τότε αυτό ταιριάζει στην ακολουθία $c_1, \cdots, c_n$ ώστε να μπορεί να το βρει η διαγωνιοποίηση στο c_n. Παρατηρούμε ότι θα ήταν πολύ πιο φυσικό να ζητήσουμε την εύρεση ενός προγράμματος r το οποίο αντί να ταιριάζει στην ακολουθία $c_1, \cdots, c_n$, να ταιριάζει στο σύνολο διατεταγμένων ζευγών

$$\{(c_1, c_2), (c_2, c_3), \cdots, (c_{n-1}, c_n)\}.$$

Στο συγκεκριμένο δεν υπάρχει μεγάλη διαφορά αλλά εν γένει υπάρχει. Θα καλούμε μια διαγωνιοποίηση η οποία αντί να ψάχνει ένα πρόγραμμα που ταιριάζει σε μια ακολουθία προγραμμάτων, ψάχνει ένα πρόγραμμα που να ταιριάζει σε ένα σύνολο προγραμμάτων, παράλληλη διαγωνιοποίηση.

Υποθέτουμε ένα σύνολο $\{s_1, \cdots, s_n\}$ υπο-προγραμμάτων ενός αυτο-επεξεργαστικού προγράμματος c. Ουσιαστικά η παράλληλη διαγωνιοποίηση μπορεί να εξηγήσει γενικεύσεις και αφαιρέσεις σε σύνολα αντικειμένων. Για παράδειγμα, εάν θεωρήσουμε ότι το πρόγραμμά μας έχει μια εσωτερική αναπαράσταση του περιβάλλοντος, τότε αν τα $s_1, \cdots, s_2$ αποτελούν εμπειρίες αλληλεπίδρασης του προγράμματος με κάποιο αντικείμενο του περιβάλλοντος, τότε η παράλληλη διαγωνιοποίηση δημιουργεί την αφηρημένη έννοια του αντικειμένου. Ας πούμε ότι τα $s_1, \cdots, s_n$ είναι n ζεύγη αντικειμένων, τότε περιμένουμε από την παράλληλη διαγωνιοποίηση να δημιουργήσει την έννοια του αριθμού 2.

4.8 Ακολουθίες που επιβιώνουν και επιτυχημένες υπακολουθίες

Στα προηγούμενα προβλήματα (με τα ψηφία), υποθέσαμε ότι τα δοσμένα ψηφία έχουν προκύψει από πολλαπλασιαστικές μεταβάσεις στις οποίες τουλάχιστον ένας απόγονος βρήκε την σωστή απάντηση.

Με βάση την παραπάνω υπόθεση, όσον αφορά το πρώτο πρόβλημα, μπορούμε να θεωρήσουμε ότι οι σωστές απαντήσεις $0, 1, 2$ είναι τυχαία αποτελέσματα που προέκυψαν σε κάποιους απογόνους του προγράμματός μας. Έστω ότι ξεκινάμε με το πρόγραμμα $c_0 = c_0[(\theta_e)0]$ το οποίο βρίσκει κατά τύχη ότι πρέπει να δώσει σαν έξοδο τον αριθμό 0, καθώς και ότι το c_1 είναι κάποιος από τους άμεσους απογόνους του c_0 και το c_2 κάποιος από τους άμεσους απογόνους του c_1 ώστε κατά τύχη να μαντεύουν τις απαντήσεις $1, 2$ αντίστοιχα. Τότε το c_2 θα καθοδηγηθεί από την διαγωνιοποίηση να υπολογίσει όλους του τους απογόνους χρησιμοποιώντας το $add(1)$ ως θ_e-διαφοροποιητικό πρόγραμμα.

Παρατηρούμε τώρα ότι στο δεύτερο πρόβλημα, όταν το περιβάλλον απαιτεί καινούργιο υπο-πρόβλημα η απόφαση να χρησιμοποιηθεί το $add(1)$ ως θ_e-διαφοροποιητικό πρόγραμμα για όλους τους απογόνους θα είχε ως αποτέλεσμα να μην επιβιώσει κανένας τους, γιατί όλοι τους θα έχουν αποκριθεί λανθασμένα στο περιβάλλον. Η κατανόηση της τοπικής φύσης των αιτημάτων του περιβάλλοντος θα είχε ως αποτέλεσμα μια πιο συντηρητική απόφαση ως προς την εφαρμογή ενός διαφοροποιητικού προγράμματος, για παράδειγμα:

Αύξησε το ποσοστό των απογόνων σου που υπολογίζονται με το $add(1)$ ως θ_e-διαφοροποιητικό πρόγραμμα.

Μια τοπική απαίτηση του περιβάλλοντος μπορεί να διαχωριστεί από μια μη τοπική από το μήκος του πρόσφατου τμήματος της ακολουθίας που επιβιώνει και στην οποία η επιλογή επιβάλλεται από την απαίτηση του περιβάλλοντος. Επίσης βλέπουμε ότι η παραπάνω συντηρητική στρατηγική είναι καλύτερη δεδομένου ότι η ακολουθία που επιβιώνει τα έχει καταφέρει κατά τύχη και το επιλεγμένο διαφοροποιητικό πρόγραμμα ταιριάζει σε ένα μικρό μέρος της ακολουθίας. Συνεπώς το πρόγραμμα:

Αν το διαφοροποιητικό πρόγραμμα ταιριάζει σε ένα μικρό μέρος της πρόσφατης ακολουθίας στη μνήμη, τότε εφάρμοσε το συντηρητικά,

είναι ένα πρόγραμμα που ταιριάζει σε ολόκληρη την ακολουθία που επιβιώνει και συνεπώς μπορεί να ευρεθεί από την διαγωνιοποίηση.

Ένα ενδιαφέρον παράδειγμα, είναι η περίπτωση όπου η ιστορία των προγόνων του προγράμματος c που έχουν επιβιώσει για ένα μεγάλο διάστημα παραμένουν ίδιοι, δηλαδή $c_1, c_2, \cdots, c_n$ είναι ίδια με το c. Σε αυτή την περίπτωση αν το πρόγραμμα $c_n = c$ υπολογίσει τους απογόνους του μέσω διαγωνιοποίησης, τότε αναμένουμε ότι θα αντιγράψει τον εαυτό του στην πλειονότητα αυτών.

Να σημειώσουμε ότι αυτή η λειτουργία ωθεί ένα αυτο-επεξεργαστικό πρόγραμμα σε ένα σταθερό πρόγραμμα, με την έννοια ότι μπορεί να μάθει να κληρονομεί χρήσιμα χαρακτηριστικά στους απογόνους του.

4.9 Διαγωνιοποίηση σε επιτυχημένες υπακολουθίες

Σε αυτή την παράγραφο θα εξετάσουμε αν ένα αυτο-επεξεργαστικό πρόγραμμα μπορεί να μάθει να εφαρμόζει τη διαγωνιοποίηση σε επιτυχημένες υπακολουθίες και η μέθοδος εκμάθησης για αυτή τη διαδικασία θα είναι η ίδια η διαγωνιοποίηση.

Έστω $\delta' = \delta[(+\theta_t)x_1, \cdots, x_m]$ ένα πρόγραμμα για έναν αλγόριθμο διαγωνιοποίησης με $x_1, \cdots, x_m$ να είναι η υπό εξέταση ακολουθία για διαγωνιοποίηση. Ο αλγόριθμος $alg(\delta')$ είναι σχεδιασμένος να βρίσκει το απλούστερο πρόγραμμα r που ταιριάζει στην ακολουθία $x_1, \cdots, x_m$. Αν $c_1, \cdots, c_n = c_n[(\theta)\delta']$ είναι η ιστορία του προγράμματος c_n και $k \leq n$, τότε η συνάρτηση

$$c_1, \cdots, c_k, \cdots, c_n = c_n[(\widehat{\theta\theta_t})x_1, \cdots, x_m] \rightarrow c_n[(\widehat{\theta\theta_t})x_1, \cdots, x_m, c_k]$$

που προσθέτει το c_k στη διεύθυνση $\widehat{\theta\theta_t}$ του c_n στην οποία βρίσκεται η υπό εξέταση για διαγωνιοποίηση ακολουθία, είναι αλγοριθμική συνεπώς από την βασική αρχή της αυτο-επεξεργασίας μπορεί να υλοποιηθεί εσωτερικά του c_n. Συνεπώς ένα αυτο-επεξεργαστικό πρόγραμμα μπορεί να αποφασίσει για την κατασκευή της υπό εξέταση ακολουθίας πάνω στην οποία θα εφαρμοστεί η διαγωνιοποίηση.

Έστω τώρα ότι

$$c_1 \rightarrow c_1^1 \rightarrow \cdots \rightarrow c_1^{n_1}$$

είναι ένας αυτο-επεξεργαστικός υπολογισμός χωρίς πολλαπλασιασμό. Ας υποθέσουμε περαιτέρω ότι το $c_1^{n_1}$ κάνει ένα πολλαπλασιαστικό βήμα στους απογόνους του οποίου βρίσκεται το c_2. Όπως και πρίν

$$c_2 \rightarrow c_2^1 \rightarrow \cdots \rightarrow c_2^{n_2}$$

είναι ένας αυτο-επεξεργαστικός υπολογισμός χωρίς πολλαπλασιασμό. Συνεχίζοντας την διαδικασία φτάνουμε σε μια ακολουθία που επιβιώνει $c_1, c_2, \cdots, c_k$, όπου για κάθε $i \leq k$ το c_i παράγει τον αυτο-επεξεργαστικό υπολογισμό

$$c_i \to c_i^1 \to \cdots \to c_i^{n_i}.$$

Καλούμε την ακολουθία

$$(c_1, \cdots, c_1^{n_1}), (c_2, \cdots, c_2^{n_2}), \cdots, (c_k, \cdots, c_k^{n_k})$$

σύνθετη ή ακολουθία κύκλων. Οι υποθέσεις που κάνουμε σε αυτή τη διαδικασία είναι:

1. Για κάθε $i \leq k$, ο υπολογισμός $c_i, c_i^1, \cdots, c_i^{n_i}$ δεν περιέχει πολλαπλασιαστικά βήματα.
2. Για κάθε $i < k$, το βήμα $c_i^{n_i} \to c_{i+1}$ είναι κύκλος, δηλαδή το c_{i+1} είναι απόγονος του $c_i^{n_i}$.
3. Για κάθε $i < k$, το αυτο-επεξεργαστικό βήμα που κάνει το $c_i^{n_i}$ είναι πολλαπλασιαστικό, συνεπώς η $c_1, c_2, \cdots, c_k$ είναι ακολουθία που επιβιώνει.

Πιο κάτω θα αντικαταστήσουμε την τρίτη υπόθεση με την απαίτηση η ακολουθία $c_1, c_2, \cdots, c_k$ να είναι επιτυχημένη. Ουσιαστικά η ιδέα είναι ότι οι σύνθετες ακολουθίες αποτελούν ένα αναδρομικό σχήμα. Αυτό που θέλουμε να μελετήσουμε είναι ότι η επιβίωση είναι η βάση αυτής της αναδρομικής διαδικασίας να χαρακτηρίζει την επιτυχία.

Οι σύνθετες ακολουθίες εντοπίζουν κάποια απαραίτητα χαρακτηριστικά των επιτυχημένων υπολογισμών. Έτσι το πρόγραμμα ψάχνει κοντινές παραλλαγές του για να βρει τα απαραίτητα χαρακτηριστικά και μετά τα κατοχυρώνει. Για να γίνει αυτό πιο καθαρό χωρίζουμε τα αυτο-επεξεργαστικά βήματα σε δύο κατηγορίες: Τα σίγουρα βήματα τα οποία είναι τα $c_i^{n_i} \to c_{i+1}$ για κάθε $i < k$ και τα αμφίβολα τα οποία είναι τα $c_i^m \to c_i^{m+1}$ για κάθε $i < k$ και $m < n_i$.
Όπως υποδεικνύει και η ονομασία τα σίγουρα βήματα είναι αποτελέσματα επιλογής, δηλαδή επιλέγονται από το περιβάλλον, ενώ τα αμφίβολα είναι αποτέλεσμα αυτο-αξιολόγησης του προγράμματος.

Υποθέτουμε τώρα ότι υπάρχει μια αλγοριθμική διαδικασία ελέγχου T τέτοια ώστε $T(c) = \mathit{true}$ αν και μόνο αν το c πρέπει να περιλαμβάνεται στην υπό εξέταση ακολουθία της διαγωνιοποίησης κατά την διάρκεια ενός αμφίβολου βήματος. Αν υποθέσουμε ότι έχουν επιλεχθεί τα σωστά προγράμματα για να συμπεριληφθούν στην υπό εξέταση ακολουθία τότε αυτό πρέπει να έχει συμβεί στην ακολουθία που επιβιώνει $c_1, \cdots, c_k$. Άρα το πρόγραμμα r όπου:

Έστω x είσοδος.
Αν $T(x) = \mathit{true}$, πρόσθεσε το x στην υπό εξέταση ακολουθία,

πρέπει, τουλάχιστον στατιστικά, να ταιριάζει στην ακολουθία, δηλαδή για κάποιο $i \leq k$ και $j < n_i, T(c_i^j) = \mathit{true}$, τότε το c_i^j πρέπει να περιλαμβάνεται στην υπό εξέταση ακολουθία του c_i^{j+1}. Παρατηρούμε ότι μια επιτυχημένη διαγωνιοποίηση παράγει προγράμματα με καλύτερη ικανότητα επιβίωσης, συνεπώς η εγκαθίδρυση του r από την διαγωνιοποίηση αποτελεί παράδειγμα

ότι η διαγωνιοποίηση μπορεί να εξελίσσει τον εαυτό της.

Σε αυτό το σημείο θα δούμε ότι μπορούμε να αντικαταστήσουμε την τρίτη υπόθεση με μια επιτυχημένη ακολουθία. Έστω ότι για $i \leq k, c_i, c_i^1, \cdots, c_i^{n_i}$ είναι ένας υπολογισμός της διαγωνιοποίησης, που επιτυγχάνει να απαντήσει στο περιβάλλον, δηλαδή δίνει σωστή απάντηση στην διεύθυνση της εξόδου στο περιβάλλον του προγράμματος $c_i^{n_i}$. Σε αυτή την περίπτωση η διαγωνιοποίηση μπορεί να οδηγήσει σε βελτίωση του εαυτού της.

Γενικά μιλώντας, θα μπορούσαμε να πούμε ότι η διαγωνιοποίηση σε σύνθετες ακολουθίες μοιάζει με την έννοια της εμπειρίας στη διαδικασία της μάθησης. Πράγματι μια τέτοια διαδικασία πρέπει να περιέχει έννοιες όπως η επιτυχία και η αποτυχία (σίγουρα βήματα), καθώς και την έννοια της επαναλαμβανόμενης δοκιμής (αμφίβολα βήματα). Για ακόμη μια φορά επιλέγουμε το αγαπημένο μας πρόβλημα.

Παράδειγμα 1. Έστω ένα αυτο-επεξεργαστικό πρόγραμμα το οποίο θέλουμε να μαντέψει την ακολουθία

$$0, 1, 2, \cdots$$

Αυτή τη φόρα θεωρούμε ότι στα $0, 1, 2$ απλά προσπαθεί στην τύχη να τα μαντέψει, όπως μάλλον θα κάναμε και εμείς. Έστω λοιπόν ότι κάποια στιγμή το πρόγραμμα καταφέρνει να μαντέψει τα $0, 1, 2$. Έστω επίσης μια διεύθυνση στην οποία δέχεται είσοδο από το περιβάλλον και ένα 0 ή 1 σε αυτήν αναπαριστά αποτυχία και επιτυχία αντίστοιχα. Αντικαθιστούμε τον αλγοριθμικό έλεγχο T έτσι ώστε $T(c) = true$ αν στην διεύθυνση εισόδου υπάρχει το 1 και $T(c) = false$ αν στην διεύθυνση εισόδου υπάρχει το 0. Αν λοιπόν, $c_i, c_i^1, \cdots, c_i^{n_i}$ είναι οι απόπειρες του προγράμματος να μαντέψει το σωστό ψηφίο (δηλαδή το i) τότε με διαγωνιοποίηση στην ακολουθία $c_0^{n_0}, c_1^{n_1}, c_2^{n_2}$, δηλαδή στην ακολουθία των προγραμμάτων που έχουν 1 στην περιβαλλοντική είσοδο, το πρόγραμμα θα πρέπει να βρει ότι πρέπει να προσθέτει 1 στην περιβαλλοντική έξοδο.

4.10 Διαγώνιες γενικεύσεις.

Σε αυτή την παράγραφο θα μελετήσουμε την εφαρμογή της διαγωνιοποίησης για την εγκαθίδρυση ενός προγράμματος απόφασης της μορφής $s[r]$ όπου το r πληρεί κάποιες προϋποθέσεις. Ως προϋποθέσεις θεωρούμε έναν αλγοριθμικό έλεγχο T. Έστω ότι:

$$(c_1, c_1^1, \cdots, c_1^{n_1}), (c_2, c_2^1, \cdots, c_2^{n_2}), \cdots, (c_k, c_k^1, \cdots, c_k^{n_k})$$

είναι μια σύνθετη ακολουθία που είτε επιβιώνει είτε είναι επιτυχημένη. Έστω $t = t[\emptyset]$ ότι είναι ένα πρόγραμμα για τον έλεγχο T όπου στην κενή διεύθυνση μπαίνει το πρόγραμμα το οποίο ελέγχεται. Επειδή το r μπορεί πολύ συχνά να πρέπει να εφαρμοστεί σε κάποιο άλλο πρόγραμμα, μας εξυπηρετεί το πρόγραμμα $t = t[\emptyset, \emptyset]$ όπου η δεύτερη κενή διεύθυνση είναι για το πρόγραμμα στο οποίο το r εφαρμόζεται (ή δανείζεται πληροφορίες). Άρα $T(r) = true$ αν και μόνο αν $alg(t[r, c_i^j]) = true$ για $i \leq k$ και $j < n_i$. Έτσι η αλγοριθμική διαδικασία (την ονομάζουμε M):

Αν για κάποιο r, το $alg(t)(r, c) = true$, τότε πάρε την απόφαση $alg(s)(r)$,

μοιάζει απαραίτητη ώστε η ακολουθία να είναι επιτυχημένη ή να επιβιώνει. Αυτό φαίνεται πιο καθαρά αφού το πρόγραμμα m για τον αλγόριθμο M ταιριάζει στην ακολουθία και συνεπώς εγκαθιδρύεται από διαγωνιοποίηση.

Παράδειγμα 2. Μέχρι τώρα υποθέταμε ότι οι προτεραιότητες στο σύνολο των προτεινόμενων προγραμμάτων κατά την διαγωνιοποίηση ήταν σταθερές. Αν δ_s είναι ένα πρόγραμμα για τον searcher τότε μπορούμε να μεταβάλλουμε την προτεραιότητα ενός προτεινόμενου προγράμματος r θεωρώντας ένα πρόγραμμα $s[r, n]$ το οποίο υλοποιεί τον αλγόριθμο:

Δώσε το r με προτεραιότητα n.

Έτσι βλέπουμε ότι αφού το δ_s είναι υπο-πρόγραμμα ενός αυτο-επεξεργαστικού προγράμματος που χρησιμοποιεί διαγωνιοποίηση, η προτεραιότητα ενός προτεινόμενου διαφοροποιητικού προγράμματος μπορεί να αλλάξει.

Έστω τώρα το αυτο-επεξεργαστικό βήμα $x \to x'$, στο οποίο το x τυχαία αποφασίζει να αυξήσει την προτεραιότητα του r όπως αυτό προτείνεται από τον searcher κατά την διαγωνιοποίηση. Θεωρούμε επίσης ότι αυτή η απόφαση είναι επιτυχημένη. Αν

$$\{(x_1, x_1^1, \cdots, x_1^{n_1}), (x_2, x_2^1, \cdots, x_2^{n_2}), \cdots, (x_k, x_k^1, \cdots, x_k^{n_k})\}$$

είναι το σύνολο των υπολογισμών της διαγωνιοποίησης στην ιστορία του x, τότε αναμένουμε ότι η σχετική συχνότητα των περιπτώσεων στις οποίες ο r έχει γίνει αποδεκτός είναι μεγαλύτερη από την σχετική προτεραιότητα που δίνει ο searcher στον r. Αυτό είναι πράγματι αληθές αφού η αύξηση της προτεραιότητα του r ως προτεινόμενο πρόγραμμα θεωρείται επιτυχημένη.
Συνεπώς αν $c_i \to c_i', i = 1, \cdots, m$ είναι επιτυχημένα αυτο-επεξεργαστικά βήματα στην ιστορία του προγράμματος c, στα οποία η προτεραιότητες των προτεινόμενων προγραμμάτων r_i αυξάνονται για κάθε $i = 1, \cdots, m$, τότε το βήμα $c_i \to c_i'$ είναι όμοιο με αυτό που περιγράφηκε παραπάνω. Άρα ένα πρόγραμμα για τον αλγόριθμο:

Αν η σχετική συχνότητα της αποδοχής ενός προγράμματος r κατά την διαγωνιοποίηση, είναι μεγαλύτερη από την σχετική προτεραιότητα πρότασής του, τότε αύξησε του την προτεραιότητα

ταιριάζει σε μια υπακολουθία της ιστορία του c που επιβιώνει ή που είναι επιτυχημένη. Βλέπουμε λοιπόν ότι η διαγωνιοποίηση μπορεί να μάθει στο πρόγραμμα να χρησιμοποιεί απλά και χρήσιμα προγράμματα. Επίσης βλέπουμε πως μπορεί η ίδια η διαγωνιοποίηση να βελτιώνεται μέσα στο γενικό αυτοαναφορικό πλαίσιο που έχουμε χτίσει.

Κεφάλαιο 5

Επίλογος

Κλείνοντας αυτήν την εργασία θα θέλαμε να κάνουμε μια συνοπτική περιγραφή αυτού του συστήματος και να επισημάνουμε ομοιότητες με τις προσεγγίσεις που έχουν ακολουθηθεί και ακολουθούνται από την επιστημονική κοινότητα.

Όπως είδαμε, οι βασικές έννοιες στις οποίες στηρίζεται η δομή που περιγράφηκε είναι η αυτοαναφορά και ο γενετικός αλγόριθμος. Επηρεασμένη από τον επιστημονικό χώρο της Βιολογίας, η κεντρική ιδέα είναι η «καλλιέργεια» προγραμμάτων μέσω μιας διαδικασίας που προσομοιώνει την Δαρβινική θεώρηση της εξέλιξης.

Για να το πετύχουμε αυτό, αντί να θεωρήσουμε έναν εξεζητημένο, εξωτερικό, στατικό εκπαιδευτή, εμφυτεύουμε στο μοντέλο έναν πρωτόλειο εκπαιδευτή με την σκέψη ότι θα λειτουργήσει σαν ένα είδος εσωτερικής, ανθρώπινης κρίσης. Έτσι κάθε φορά που το μοντέλο κάνει αυτοκριτική έχουμε μια αυτοαναφορική μετάβαση, μια καινούργια γενιά. Τα επιτυχημένα μοντέλα είναι αυτά που επιβιώνουν με βάση κάποιας μορφής επιλογή (φυσική ή τεχνητή) το οποίο κρίνεται από το περιβάλλον με το οποίο αλληλεπιδρούν.

Η σκέψη ότι μπορούμε να προσεγγίσουμε την ευφυΐα ένος ενήλικου εγκεφάλου ξεκινώντας από μια απλούστερη δομή και στη συνέχεια εκπαιδεύοντας την δεν είναι καθόλου καινούργια. Ο Alan Turing ήδη από το 1950 είχε πει ότι η απευθείας απομίμηση ενός ενήλικου εγκεφάλου είναι πολύ δύσκολη διαδικασία και ότι ίσως είναι προτιμότερο, να ξεκινήσουμε από μια εύκολη αρχική κατάσταση π.χ. η κατάσταση του εγκεφάλου στη γέννηση και στη συνέχεια να εκπαιδεύσουμε αυτόν τον εγκέφαλο θεωρητικά και εμπειρικά μέχρι να προσεγγίσουμε την ενήλικη νοημοσύνη.

Επίσης το κεντρικό συστατικό της αυτοαναφοράς είναι κάτι που έχει διερευνηθεί και διερευνάται. Το πεδίο στο οποίο μάλλον έχει τον πρωταγωνιστικό ρόλο η αυτοαναφορά είναι η ψυχολογία και η φιλοσοφία. Ο 20ος αιώνας που υπήρξε έντονη ανάπτυξη στον χώρο των θεμελίων των μαθηματικών και της επιστήμης των υπολογιστών, ήταν ιδιαίτερα σημαντικός για την αυτοαναφορά αν και τα πρώτα αποτελέσματα δεν την αξιοποιούσαν με τον κατασκευαστικό

της χαρακτήρα. Σίγουρα στο δεύτερο μισό του 20ου αιώνα η αυτοαναφορά έκανε έντονη την παρουσία της σε διάφορους διεπιστημονικούς κλάδους των μαθηματικών και της Βιολογίας όπως η υπολογιστική Βιολογία και η υπολογιστική νευροεπιστήμη.

Τέλος, θα ήθελα να τονίσω ότι η παρούσα εργασία είναι εμπνευσμένη από το ερευνητικό έργο του καθηγητή μου Αλέξανδρου Αρβανιτάκη, ο οποίος αποτελεί διαρκή πηγή έμπνευσης, και τον ευχαριστώ βαθύτατα που με έφερε σε επαφή με τον κλάδο της Επιστήμης των Υπολογιστών καθώς και για την συμβολή του, στην διαμόρφωση των επιστημονικών μου ενδιαφερόντων, και στην υλοποίηση αυτής της εργασίας.

Βιβλιογραφία

[1] Arvanitakis A. *Recursion, evolution and conscious self* arXiv, April 2023

[2] J. R. Koza *Genetic Programming On the Programming of Computers by Means of Natural Selection* The MIT Press, Cambridge, Massachusetts

[3] Christos H. Papadimitriou, Santosh S. Vempala, Daniel Mitropolsky, Michael Collins, Wolfgang Maass *Brain computation by assemblies of neurons*

[4] Peter Norvig *Paradigms of Artificial Intelligence Programming:* Case studies in Common Lisp, Morgan Kaufmann Publishers, San Francisco, California

[5] Alan Turing *Computing Machinery and Intelligence*, October 1950

[6] J. Von Neumann *Theory of self-reproducing automata*, 1966

[7] J. Schmidhuber *Godel Machines: Fully Self-referential Optimal Universal Self-improvers*, Springer

[8] Tsvi Tlusty *The self-referring DNA and protein:* A remark on physical and geometrical aspects, September 4, 2015