ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Βελτιστοποίηση Ερωτημάτων με Eddies και Ενισχυτική Μάθηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΣΤΑ ΤΖΟΥΜΑ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2007

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Βελτιστοποίηση Ερωτημάτων με Eddies και Ενισχυτική Μάθηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΣΤΑ ΤΖΟΥΜΑ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Ιουλίου 2007.

.....	.....	.....
Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.	Ιωάννης Βασιλείου Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Επικ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2007

.....

ΚΩΣΤΑΣ ΤΖΟΥΜΑΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Η εργασία αυτή ξεκίνησε ως μια αυτόνομη περιπλάνηση στην έρευνα στο πεδίο των Βάσεων Δεδομένων και λειτούργησε ως η εισαγωγή μου στην ακαδημαϊκή έρευνα. Ευχαριστώ τον καθηγητή μου κ. Τίμο Σελλή για το ενδιαφέρον και το χρόνο που μου διέθεσε. Ήταν πάντα εκεί για να δώσει την κατεύθυνση ή το εύρος που χρειάζονταν στην αναζήτησή μου. Τέλος, θα ήθελα να ευχαριστήσω όσους με υπομείνανε το τελευταίο δύσκολο για μένα διάστημα.

Περίληψη

Η βελτιστοποίηση ερωτημάτων σε Βάσεις Δεδομένων είναι ένα πολύπλοκο πρόβλημα συνδυαστικής βελτιστοποίησης που καθιστά μη θεμιτή την εξαντλητική αναζήτηση όταν το μέγεθος του ερωτήματος είναι μεγάλο. Ταυτόχρονα, δυναμικά περιβάλλοντα εκτέλεσης καθώς και η πολυπλοκότητα των ίδιων των δεδομένων καθιστούν τις υποθέσεις που γίνονται κατά το χρόνο βελτιστοποίησης μη ρεαλιστικές κατά το χρόνο εκτέλεσης. Τα eddies παρέχουν ένα μηχανισμό εκτέλεσης ερωτημάτων κατά τον οποίο το πλάνο εκτέλεσης προσαρμόζεται ανά πλειάδα, αντιμετωπίζοντας την εκτέλεση του ερωτήματος ως μια διαδικασία δρομολόγησης των πλειάδων στους τελεστές. Στη διπλωματική αυτή εργασία μοντελοποιούμε τη διαδικασία βελτιστοποίησης ερωτημάτων που εκτελούνται με τη βοήθεια του μηχανισμού των eddies και των αρχιτεκτονικών εκτέλεσης συνδέσμων που τον περιβάλλουν (SteMs, STAIRs) ως πρόβλημα Ενισχυτικής Μάθησης. Αναπαριστούμε την πολιτική δρομολόγησης των πλειάδων σαν μια απεικόνιση από ένα χώρο καταστάσεων σε ένα χώρο δράσεων ενσωματώνοντας στο μοντέλο τους σημασιολογικούς περιορισμούς καθώς και τους περιορισμούς δρομολόγησης. Το πρόβλημα της βελτιστοποίησης μετατρέπεται έτσι σε ένα πρόβλημα μη επιβλεπόμενης μάθησης από ποσοτικές ενισχύσεις. Μέσω της μοντελοποίησης αυτής προτείνουμε αλγόριθμους μάθησης μιας βέλτιστης πολιτικής δρομολόγησης διατηρώντας την προσαρμοστικότητα σε επίπεδο πλειάδας.

Λέξεις Κλειδιά: Eddies, Βελτιστοποίηση Ερωτημάτων, Ενισχυτική Μάθηση, Προσαρμοστική Επεξεργασία Ερωτημάτων

Abstract

Database query optimization is a complex combinatorial optimization problem which makes exhaustive search inapplicable as query size grows. Moreover, dynamic execution environments as well as data complexities result that assumptions made at optimization time do not hold at execution time. Eddies provide an execution mechanism in which the query plan can be adapted in a per – tuple fashion by treating query execution as a tuple routing process. This thesis models the execution of a query using an eddy and the corresponding join mechanisms (SteMs, STAIRs) as a Reinforcement Learning problem. The tuple – routing policy is treated as a mapping from a state space to an action space and the semantics as well as the routing constraints get embedded into the model. Thus, the optimization problem is transformed to an unsupervised learning problem by quantitative rewards. Through this process, we propose algorithms that can learn an optimal routing policy without sacrificing the desired per – tuple adaptivity.

Keywords: Eddies, Query Optimization, Reinforcement Learning, Adaptive Query Processing

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 Αντικείμενο διπλωματικής

Τα Σχεσιακά ΣΔΒΔ (Συστήματα Διαχείρισης Βάσεων Δεδομένων) που έχουν τις ρίζες τους στα πρώτα συστήματα System R και Ingres χρησιμοποιούν μια δηλωτική γλώσσα προγραμματισμού για την πρόσβαση και την εκτέλεση ερωτήσεων στα δεδομένα. Ο χρήστης του συστήματος δεν είναι υποχρεωμένος έτσι να προδιαγράψει πώς τα δεδομένα που χρειάζεται θα συλλεχθούν αλλά μόνο ποια είναι αυτά. Το πρώτο είναι η αποστολή του Επεξεργαστή Ερωτημάτων (Quey Executor), σκοπός του οποίου είναι ο μετασχηματισμός ενός δηλωτικού ερωτήματος (παράσταση σχεσιακής άλγεβρας) σε ένα φυσικό πλάνο εκτέλεσης. Δυστυχώς τα διαφορετικά πλάνα είναι πάρα πολλά και έχουν μεγάλες διαφορές μεταξύ τους όσον αφορά το χρόνο εκτέλεσης. Η έρευνα στο χώρο των πλάνων και η επιλογή ενός καλού πλάνου εκτέλεσης είναι η αποστολή του Βελτιστοποιητή Ερωτημάτων, ο οποίος καλείται να λύσει ένα πολύ δύσκολο πρόβλημα βελτιστοποίησης, ιδιαίτερα όσον αφορά τη σειρά των συνδέσμων (joins). Τεχνικές όπως ο Δυναμικός Προγραμματισμός, Προσομοιωμένη Ανόπτηση και Γενετικοί Αλγόριθμοι έχουν προταθεί για την επίλυση του προβλήματος.

Η ευθύγραμμη όμως πορεία ανάλυσης στατιστικών, βελτιστοποίησης και εκτέλεσης πολλές φορές δεν αρκεί. Πολύπλοκα ερωτήματα σε πολλούς πίνακες, η αδυναμία κράτησης ενημερωμένων και σωστών στατιστικών, καθώς και νέου τύπου περιβάλλοντα διαχείρισης

δεδομένων καθιστούν την παραπάνω φιλοσοφία ανεπαρκή. Δυναμικά περιβάλλοντα με δεδομένα που αλλάζουν συνέχεια και ανάγκη on – line επεξεργασίας ερωτημάτων απαιτούν μια διαφορετική προσέγγιση, στην οποία τα παραπάνω βήματα θα πραγματοποιούνται αλληλένδετα στην ίδια κλίμακα χρόνου. Η Προσαρμοστική Επεξεργασία Ερωτημάτων είναι ο κλάδος αυτός την έρευνας που προσπαθεί να βρει νέους τρόπους να απαντήσει στο πρόβλημα της γρήγορης εκτέλεσης ερωτημάτων σε συστήματα με τα παραπάνω χαρακτηριστικά. Ο μηχανισμός των Eddies, που προτάθηκε στο [AH00] και χρησιμοποιείται στο TelegraphCQ ([CCD+03]) είναι το πιο χαρακτηριστικό δείγμα της παραπάνω φιλοσοφίας. Ο Eddy είναι ένα module που μεσολαβεί ανάμεσα στα δεδομένα και τους τελεστές επεξεργασίας τους και αντιμετωπίζει την εκτέλεση ενός ερωτήματος ως ένα πρόβλημα συνεχής δρομολόγησης πλειάδων στους τελεστές με γρήγορο και συνεπή τρόπο. Τελικά, τα τρία βήματα της επεξεργασίας του ερωτήματος γίνονται συνέχεια κατά το χρόνο εκτέλεσης και καταργείται η ανάγκη ύπαρξης βελτιστοποιητή ερωτημάτων και σε τελικό στάδιο καταργείται η ανάγκη κράτησης στατιστικών, ακόμα και η ανάγκη προγραμματισμού εναλλακτικών αλγορίθμων για συνδέσμους.

1.1.1 Συνεισφορά

Στην παρούσα διπλωματική εργασία ερευνούμε τον τρόπο με τον οποίο κλασσικές και μη τεχνικές βελτιστοποίησης μπορούν να ενσωματωθούν σε ένα περιβάλλον εκτέλεσης ερωτημάτων με Eddies χωρίς να χάνεται η προσαρμοστικότητα. Ισχυριζόμαστε ότι το πεδίο της Ενισχυτικής Μάθησης παρέχει τα απαραίτητα εργαλεία και δείχνουμε τον τρόπο με τον οποίο η εκτέλεση ενός ερωτήματος με Eddies μπορεί να μοντελοποιηθεί ως πρόβλημα Ενισχυτικής Μάθησης και πώς η βελτιστοποίησης ενός ερωτήματος στο παραπάνω περιβάλλον ισοδυναμεί με τη μάθηση μια βέλτιστης πολιτικής δρομολόγησης. Τέλος, προτείνουμε αλγόριθμους μάθησης μιας βέλτιστης πολιτικής. . Η μόνη δημοσιευμένη εργασία την παρούσα στιγμή πάνω σε ανάλυση πολιτικών δρομολόγησης για eddies είναι η [TD03], η οποία όμως ασχολείται με κατανεμημένα eddies και δανείζεται τεχνικές βέλτιστης δρομολόγησης σε δίκτυα δεδομένων.

1.2 Οργάνωση κειμένου

Η εργασία αυτή χωρίζεται σε 8 κεφάλαια. Στο κεφάλαιο 2 γίνεται μια εισαγωγή στη Βελτιστοποίηση Ερωτημάτων βασισμένη στο κόστος (Cost – based Query Optimization) καθώς και μια βιβλιογραφική αναφορά σε πιθανοτικούς αλγόριθμους για τη λύση του προβλήματος. Στο κεφάλαιο 3 γίνεται μια εισαγωγή στο πεδίο της Προσαρμοστικής Επεξεργασίας Ερωτημάτων και παρουσιάζεται εκτενώς ο μηχανισμός των eddies. Αναλύεται η εκτέλεση ερωτημάτων επιλογών και συνδέσμων με eddies, παρουσιάζεται το πρόβλημα του

«βάρους της ιστορίας», δίνονται οι λύσεις των STAIRs και των SteMs που έχουν προταθεί για αυτό και τέλος γίνεται αναφορά στις ήδη προταθείσες μέθοδοι δρομολόγησης. Στο κεφάλαιο 4 γίνεται μια εισαγωγή στο πεδίο της Ενισχυτικής Μάθησης. Παρουσιάζεται το πρόβλημα καθώς και οι πιο διαδεδομένοι αλγόριθμοι λύσης. Στο κεφάλαιο 5 προτείνεται η μοντελοποίηση των Eddies ως πρόβλημα Ενισχυτικής Μάθησης και αναλύονται τα επιμέρους προβλήματα που προκύπτουν. Το κεφάλαιο 6 προσφέρει μια ολοκληρωμένη ματιά στη μάθηση σε διάφορες αρχιτεκτονικές δίνοντας τρόπους με τους οποίους η μάθηση μπορεί να συνδυαστεί με την εκτέλεση του ερωτήματος. Στο κεφάλαιο 7 επιχειρείται μια σύνοψη της εργασίας και παρουσιάζονται δυνατότητες μελλοντικής έρευνας. Τέλος, το κεφάλαιο 8 παρουσιάζει τη σχετική βιβλιογραφία.

2

Εισαγωγή στη Βελτιστοποίηση Ερωτημάτων σε Βάσεις Δεδομένων

2.1 Αρχιτεκτονική ενός βελτιστοποιητή ερωτημάτων

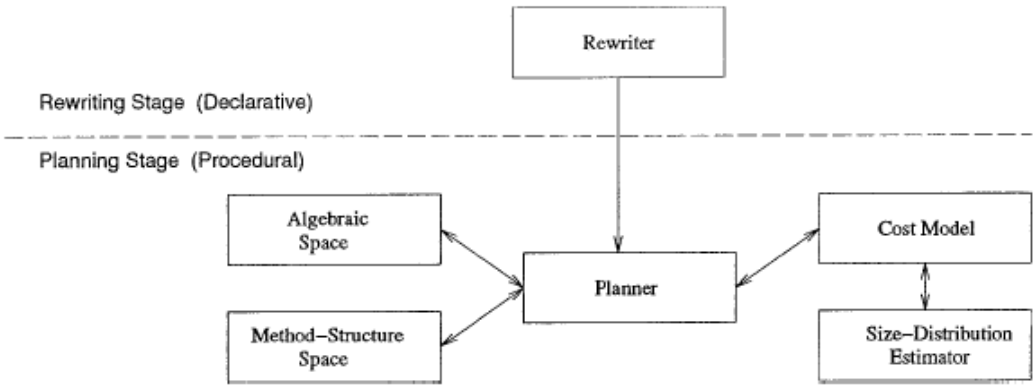


Figure 1. Query optimizer architecture.

Σχήμα 2.1 (πηγή [Ioa96])

Ένα ερώτημα φτάνει σε ένα ΣΔΒΔ σε μια δηλωτική γλώσσα (SQL). Για μια τέτοια έκφραση υπάρχουν πολλοί τρόποι (πλάνα εκτέλεσης) να εκτελεστεί. Η αποστολή του τμήματος του *Βελτιστοποιητή Ερωτημάτων* (Query Optimizer) είναι να ερευνήσει το χώρο αυτό των πλάνων για ένα δεδομένο ερώτημα και να επιλέξει από αυτά το καλύτερο (ή ένα αρκετά καλό)

[Ioa96]. Το τμήμα που παρέχει την παραπάνω λειτουργικότητα μπορεί αφαιρετικά να διαχωριστεί στα τμήματα Rewriter και Planner. Αποστολή του Query Rewriter είναι να μετατρέψει το ερώτημα σε ένα ισοδύναμο αλλά πιο απλό ερώτημα σχεσιακής άλγεβρας, αντικαθιστώντας όψεις με τους ορισμούς τους, μετασχηματίζοντας «φωλιασμένα» ερωτήματα και άλλα, χωρίς όμως να λαμβάνει υπ' όψιν μετρήσεις κόστους, δηλαδή βασίζεται μόνο στη στατική σύνταξη και σημασιολογία του ερωτήματος. Σκοπός του Planner είναι να εξερευνήσει το χώρο των πλάνων (που δίνεται από το Algebraic Space και το Method – Structure Space) προσπαθώντας να βρει ένα καλό πλάνο ως προς ένα κόστος που δίνεται από το Cost Model και το Size – distribution Estimator. Η είσοδος του Planner είναι μια παράσταση σχεσιακής άλγεβρας ενώ η έξοδός του είναι ένα φυσικό πλάνο εκτέλεσης. Η αποστολή του είναι λοιπόν.

1. Να δημιουργήσει ένα *λογικό πλάνο εκτέλεσης* επιλέγοντας την σειρά με την οποία θα εκτελεστούν οι σχεσιακές πράξεις.
2. Να επιλέξει τους φυσικούς τελεστές οι οποίοι θα υλοποιήσουν τις πράξεις σχεσιακής άλγεβρας, παράγοντας έτσι ένα *φυσικό πλάνο εκτέλεσης*.

Στην εργασία αυτή θα περιοριστούμε σε ερωτήματα που περιλαμβάνουν επιλογές, προβολές και φυσικούς συνδέσμους (natural joins). Ένας σύνδεσμος ισότητας (equi – join) εκτελείται ουσιαστικά με τον ίδιο τρόπο με ένα natural join με τη διαφορά ότι τα πεδία που ταυτίζονται μπορεί να έχουν διαφορετικά ονόματα, και ένας θ – σύνδεσμος είναι ο συνδυασμός ενός συνδέσμου και μιας επιλογής.

Ο υπολογισμός του κόστους και η εκτίμηση του μεγέθους των ενδιάμεσων σχέσεων γίνεται με τη βοήθεια αποθηκευμένων στο σύστημα στατιστικών πληροφοριών (μοντέλο δεδομένων - data model), όπως το μέσο κόστος ενός τελεστή, η επιλεκτικότητα των επιλογών και το *fanout* των συνδέσμων καθώς και πιο προχωρημένες μορφές πληροφορίας, όπως ιστογράμματα. Στις παρακάτω ενότητες παρουσιάζουμε στοιχειωδώς τα υποσυστήματα Cost Model (εκτιμήσεις κόστους τελεστών) και Size – Distribution Estimator (εκτιμήσεις μεγεθών) καθώς και μια πληθώρα αλγορίθμων αναζήτησης για την εύρεση του βέλτιστου πλάνου εκτέλεσης.

2.2 Μοντέλα κόστους και εκτιμήσεις μεγεθών

Μια παράμετρος του συστήματος είναι το μοντέλο κόστους, δηλαδή η συνάρτηση $C : expr \to \mathbb{R}$. Η πιο δημοφιλής μετρική είναι το άθροισμα του χρόνου CPU και των προσβάσεων στο δίσκο για την εκτέλεση ενός ερωτήματος, η οποία απλοποιείται πολλές φορές στον αριθμό των προσβάσεων στο δίσκο. Πολλές άλλες μετρικές όμως μπορούν να ενδιαφέρουν ανάλογα με τις ιδιαιτερότητες του συστήματος, για παράδειγμα ο ρυθμός με τον

οποίο η εκτέλεση παραδίδει πλειάδες του αποτελέσματος στο χρήστη. Οι τελεστές σχεσιακής άλγεβρας ($\sigma, \pi, \triangleright \triangleleft$) μπορούν να υλοποιηθούν με πολλούς διαφορετικούς φυσικούς τελεστές οι οποίοι διαφέρουν ως προς το κόστος και τη μνήμη που απαιτούν.

2.2.1 Συμβολισμοί

Στην ενότητα αυτή, συμβολίζουμε

$B(R)$ ο αριθμός των blocks που καταλαμβάνει η σχέση R

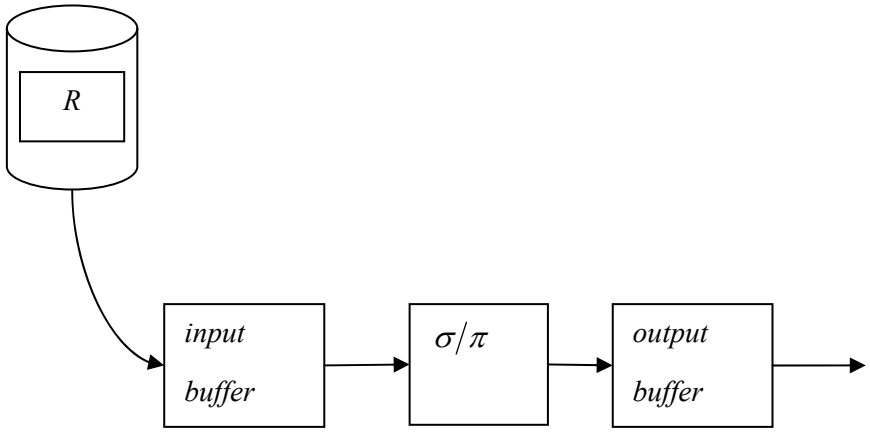
$T(R)$ ο αριθμός των πλειάδων της σχέσης R

$V(R,a)$ το πλήθος των τιμών της ιδιότητας α της σχέσης R

M το μέγεθος της διαθέσιμης κύριας μνήμης (buffer)

2.2.2 Κόστος επιλογής και προβολής

Οι πράξεις της επιλογής και της προβολής είναι γενικά «εύκολες», αφού μπορούν να υλοποιηθούν με ένα πέρασμα στα δεδομένα. Τα blocks της σχέσης διαβάζονται σειριακά και τροποποιούνται από τον τελεστή, όπως υπονοεί και το παρακάτω σχήμα.



Στην περίπτωση αυτή το κόστος των τελεστών είναι

$$C(\sigma(R)) = B(R)$$
$$C(\pi(R)) = B(R) \qquad (2.2.1)$$

Στην περίπτωση μιας επιλογής ισότητας, αν χρησιμοποιήσουμε ένα ευρετήριο στην ιδιότητα της επιλογής, η επιλογή έχει κόστος

$$C\left(\sigma_{a=u}\left(R\right)\right)=\frac{B\left(R\right)}{V\left(R,a\right)} \tag{2.2.2}$$

2.2.3 Κόστος συνδέσμου

Για την εκτέλεση ενός συνδέσμου μπορούν να χρησιμοποιηθούν πολλοί αλγόριθμοι καταλήγοντας σε διαφορετικά κόστη και απαιτήσεις μνήμης. Τα κόστη των πιο βασικών παρουσιάζονται παρακάτω υποθέτοντας ότι $B\left(S\right)<B\left(R\right)$ και ότι οι σχέσεις είναι clustered.

Ένα απλό nested – loop – join έχει κόστος

$$C\left(R\underset{nlj}{\triangleright\triangleleft}S\right)=B\left(S\right)+\frac{B\left(S\right)B\left(R\right)}{M-1}\simeq\frac{B\left(R\right)B\left(S\right)}{M} \tag{2.2.3}$$

και

$$C\left(R\underset{nlj}{\triangleright\triangleleft}S\right)=B\left(R\right)+T\left(R\right)h \tag{2.2.4}$$

αν υπάρχει υποστήριξη ευρετηρίου στην S, όπου h είναι ο μέσος χρόνος που χρειάζεται για την αναζήτησης μιας πλειάδας στο ευρετήριο ο οποίος εξαρτάται από το είδος του ευρετηρίου (B+ tree/hash – πρωτεύον ή δευτερεύον). Ένα sort – merge join έχει κόστος

$$C\left(R\underset{smj}{\triangleright\triangleleft}S\right)=C_R+C_S \tag{2.2.5}$$

όπου

$$C_X=\begin{cases}B\left(X\right) & ,\text{η } X \text{ είναι ταξινομημένη ή έχει πρωτεύον B+ ευρετήριο}\\ \left\lceil T\left(X\right)a\right\rceil+B\left(X\right) & ,\text{η } X \text{ έχει δευτερεύον B+ ευρετήριο, } a \text{ σταθερά}\\ B\left(X\right)\log_M B\left(X\right)+B\left(X\right) & ,\text{η } X \text{ δεν είναι ταξινομημένη}\end{cases}$$

Τέλος ένα Hybrid Hash Join έχει κόστος

$$C\left(R\underset{hhj}{\triangleright\triangleleft}S\right)=B\left(R\right)+B\left(S\right)+2\left(B\left(R\right)+B\left(S\right)\right)\left(1-q\right) \tag{2.2.6}$$

όπου q είναι το ποσοστό της S του οποίου ο πίνακας κατακερματισμού χωράει στην κύρια μνήμη $q=\dfrac{M-\left\lceil\dfrac{1.4B\left(S\right)-M}{M-1}\right\rceil}{B\left(S\right)}$

2.2.4 Στατιστικές εκτιμήσεις αποτελεσμάτων παραστάσεων

Για την εκτίμηση του κόστους μιας σχεσιακής παράστασης χρειάζεται εκτός από το κόστος των τελεστών και η εκτίμηση του μεγέθους του αποτελέσματος. Η εκτίμηση του μεγέθους μιας επιλογής (σε πλειάδες) υποθέτοντας ομοιόμορφη κατανομή των δεδομένων είναι

$$T\left(\sigma_{a=u}\left(R\right)\right)=\frac{T\left(R\right)}{V\left(R,a\right)} \tag{2.2.7}$$

$$T\left(\sigma_{a\le u}\left(R\right)\right)=T\left(R\right)\frac{u-\min\left(R,a\right)}{\max\left(R,a\right)-\min\left(R,a\right)} \tag{2.2.8}$$

Ορίζοντας επιλεκτικότητα μιας επιλογής το μέγεθος $s=\frac{T\left(\sigma\left(R\right)\right)}{T\left(R\right)}$ μπορούμε να εκτιμήσουμε το κόστος σύζευξης επιλογών

$$T\left(\sigma_1\left(\cdots\sigma_n\left(R\right)\cdots\right)\right)=T\left(R\right)s_1\cdots s_n \tag{2.2.9}$$

Μια εκτίμηση μεγέθους ενός φυσικού συνδέσμου που δεν είναι καρτεσιανό γινόμενο είναι

$$T\left(R\underset{a}{\triangleright\triangleleft}S\right)=\frac{T\left(R\right)T\left(S\right)}{\max\left\{V\left(R,a\right),V\left(S,a\right)\right\}} \tag{2.2.10}$$

Για την εκτίμηση του κόστους ενός πλήρους πλάνου εκτέλεσης γίνεται με ένα bottom – up τρόπο, εκτιμώντας το κόστος της πράξης στα φύλλα καθώς και του κόστους του ενδιάμεσου αποτελέσματος και συνεχίζοντας μέχρι τη ρίζα.

2.3 Μετασχηματισμοί σχεσιακών παραστάσεων

Η βελτιστοποίηση ερωτημάτων βασίζεται κατά πολύ στην εφαρμογή ιδιοτήτων της σχεσιακής άλγεβρας για την παραγωγή μιας ισοδύναμης παράστασης με μικρότερο κόστος. Οι πιο σημαντικοί από αυτούς αναφέρονται παρακάτω.

Η πράξη συνδέσμου έχει την προσεταιριστική και την αντιμεταθετική ιδιότητα

$$R\triangleright\triangleleft S=S\triangleright\triangleleft R \tag{2.3.1}$$

$$R\triangleright\triangleleft\left(S\triangleright\triangleleft T\right)=\left(R\triangleright\triangleleft S\right)\triangleright\triangleleft T \tag{2.3.2}$$

Η πράξη της επιλογής έχει την αντιμεταθετική ιδιότητα και είναι επιμεριστική ως προς το σύνδεσμο

$$\sigma_{p\wedge q}\left(R\right)=\sigma_p\left(\sigma_q\left(R\right)\right)=\sigma_q\left(\sigma_p\left(R\right)\right) \tag{2.3.3}$$

$$\sigma_p\left(R \triangleright\triangleleft S\right) = \sigma_{p_R}\left(R\right) \triangleright\triangleleft \sigma_{p_S}\left(S\right) \tag{2.3.4}$$

όπου το κατηγόρημα p_R (p_S) είναι το υποσύνολο του p το οποίο αναφέρεται μόνο σε ιδιότητες της R (S).

Ο βασικός νόμος για την πράξη της προβολή είναι ότι αυτή μπορεί να δημιουργηθεί σε οποιοδήποτε σημείο του πλάνου εκτέλεσης δεδομένου ότι δεν διαγράφει ιδιότητες που θα χρειαστούν αργότερα στο πλάνο.

$$\pi_L\left(R \triangleright\triangleleft S\right) = \pi_L\left(\pi_{L_R}\left(R\right) \triangleright\triangleleft \pi_{L_S}\left(S\right)\right) \tag{2.3.5}$$

όπου οι ιδιότητες L_R (L_S) περιέχουν τις ιδιότητες του συνδέσμου (join attributes) καθώς και αυτές που εμφανίζονται στις L και ανήκουν στην R (S).

$$\pi_L\left(\sigma_p\left(R\right)\right) = \pi_L\left(\sigma_p\left(\pi_M\left(R\right)\right)\right) \tag{2.3.6}$$

όπου οι ιδιότητες M εμφανίζονται στο κατηγόρημα p ή ανήκουν στις ιδιότητες L.

2.4 Κανόνες ευριστικής βελτιστοποίησης

Ως πρώτο βήμα, ένας βελτιστοποιητής ερωτημάτων μπορεί να εφαρμόσει δύο πολύ ισχυρούς ευριστικούς κανόνες μετασχηματισμού του λογικού πλάνου εκτέλεσης ελπίζοντας ότι το ισοδύναμο που θα προκύψει θα έχει μικρότερο κόστος εκτέλεσης. Οι κανόνες αυτοί ονομάζονται ευριστικοί επειδή δεν παίρνουν υπ' όψιν τις εκτιμήσεις κόστους.

1. Ώθηση των επιλογών προς τα κάτω: Μια επιλογή ωθείται στα φύλλα του δέντρου του λογικού πλάνου εκτέλεσης αξιοποιώντας τον κανόνα 2.3.4
2. Δημιουργία προβολών όπου αυτό γίνεται με την εφαρμογή των κανόνων 2.3.5 και 2.3.6.

Η λειτουργία των κανόνων αυτών φαίνεται στο παρακάτω παράδειγμα

Παράδειγμα

Έστω το ακόλουθο σχήμα μιας βάσης δεδομένων

```
S (Name, Level, Address)
C (Course, Instructor, Credits, Semester)
E (Name, Course, Semester)
```

και ένας χρήστης θέτει το παρακάτω ερώτημα

```
select   S.Name, S.Address, E.Course, C.Credits
from     S, E, C
where    S.Name = E.Name and E.Course = C.Course and
```

```
S.Level = “Junior” and S.Address like “%Athens%” and
C.Credits > 5
```

Αρχικά το ερώτημα αυτό μετατρέπεται στην ακόλουθη παράσταση σχεσιακής άλγεβρας

$$\pi_{\substack{S.Name,\\ S.Adress.\\ E.Course,\\ C.Credits}} \left(\sigma_{\substack{S.Level="Junior"\wedge\\ like(S.Address,"\%Athens\%")\wedge\\ C.Credits>5}} \left(S \triangleright\!\triangleleft E \triangleright\!\triangleleft C \right) \right)$$

Η εφαρμογή του μετασχηματισμού 1 δίνει την παρακάτω ισοδύναμη παράσταση σχεσιακής άλγεβρας

$$\pi_{\substack{S.Name,\\ S.Adress.\\ E.Course,\\ C.Credits}} \left(\sigma_{\substack{S.Level="Junior"\wedge\\ like(S.Address,"\%Athens\%")}} \left(S \right) \triangleright\!\triangleleft \sigma_{C.Credits} \left(E \right) \triangleright\!\triangleleft C \right)$$

και η εφαρμογή του μετασχηματισμού 2 δίνει την ακόλουθη παράσταση

$$\pi_{\substack{S.Name,\\ S.Adress.\\ E.Course,\\ C.Credits}} \left(\sigma_{\substack{S.Level="Junior"\wedge\\ like(S.Address,"\%Athens\%")}} \left(\pi_{\substack{S.Name,\\ S.Level,\\ S.Address}} \left(S \right) \right) \triangleright\!\triangleleft \sigma_{C.Credits} \left(\pi_{\substack{E.Course,\\ E.Name}} \left(E \right) \right) \triangleright\!\triangleleft \pi_{\substack{C.Course,\\ C.Credits}} \left(C \right) \right)$$

Μετά την εφαρμογή των κανόνων αυτών η βελτιστοποίηση ενός ερωτήματος ανάγεται στη λύση των ακόλουθων προβλημάτων.

1. Εύρεση της βέλτιστης σειράς για τις συζεύξεις επιλογών
2. Εύρεση της βέλτιστης σειράς των συνδέσμων
3. Επιλογή αλγορίθμου για κάθε σύνδεσμο

Από τα παραπάνω το πιο δύσκολο και αυτό που επηρεάζει περισσότερο το κόστος του ερωτήματος είναι το 2. Στην παρακάτω ενότητα παρουσιάζουμε μεθόδους που έχουν εμφανιστεί στη βιβλιογραφία για την αντιμετώπισή του.

2.5 Εύρεση της σειράς των συνδέσμων

Στην ενότητα αυτή παρουσιάζονται διάφορες μέθοδοι που έχουν προταθεί στην βιβλιογραφία για τη λύση του προβλήματος της βέλτιστης σειράς των συνδέσμων.

2.5.1 Ο χώρος των πλάνων

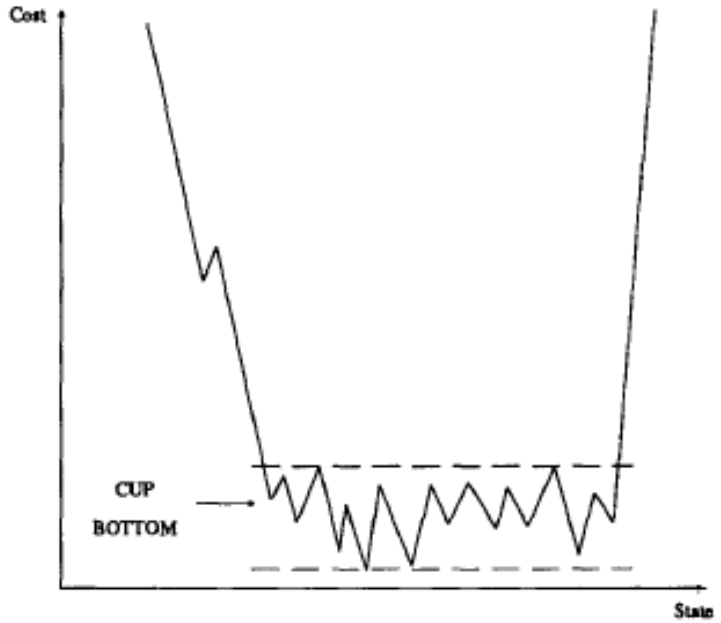
Η είσοδος ενός αλγορίθμου εύρεσης της σειράς των συνδέσμων είναι ο γράφος συνδέσμων, ο οποίος περιέχει ένα κόμβο για κάθε σχέση που εμφανίζεται στο ερώτημα και μια ακμή

ανάμεσα σε δύο κόμβους αν αυτές οι σχέσεις μπορούν να συνδεθούν. Από τη διάσχιση του γράφου μπορούν να προκύψουν πολλά δέντρα εκτέλεσης.

Παραδοσιακά, ο χώρος των αριστερά – βαθειών (left – deep) δέντρων είχε ιδιαίτερο ενδιαφέρον [SAC+79] επειδή σε αυτά καμία ενδιάμεση σχέση δεν χρειάζεται να αποθηκευθεί στο δίσκο. Σε ένα αριστερά – βαθύ δέντρο κάθε εσωτερική σχέση συνδέσμου είναι μια βασική σχέση (base relation). Δεν υπάρχει βαθμός ελευθερίας ως προς το σχήμα του αλλά υπάρχουν $n!$ τρόποι να τοποθετηθούν n σχέσεις στα φύλλα του. Ο πλήρης χώρος των δέντρων (που ονομάζονται bush trees) περιέχει $\binom{2(n-1)}{n-1}(n-1)!$ δυνατές λύσεις, οπότε το πρόβλημα είναι NP – hard. Έχει υποστηριχθεί βάσει πειραματικών ενδείξεων ([IK90], [IK91]) ότι η κατανομή του κόστους στο χώρο των πλάνων έχει τα ακόλουθα χαρακτηριστικά.

- το μέσο τοπικό ελάχιστο έχει μικρό κόστος σε σύγκριση με ένα τυχαίο πλάνο
- η μέση απόσταση από ένα τυχαίο πλάνο μέχρι ένα τοπικό ελάχιστο είναι μεγάλη
- υπάρχουν πολλά τοπικά ελάχιστα
- πολλά τοπικά ελάχιστα συνδέονται μέσω μονοπατιών που περνούν από πλάνα μικρού κόστους
- η αναζήτηση στα παραπάνω μονοπάτια παράγει καλύτερα αποτελέσματα από την αναζήτηση σε τυχαίες περιοχές

Τα παραπάνω σημεία οδηγούν στο ότι το σχήμα της συνάρτησης κόστους έχει το σχήμα ενός «πηγαδιού» (ή «ποτηριού») με ανωμαλίες στον πάτο του, όπως φαίνεται παρακάτω.



Σχήμα 2.2 (πηγή [IK90])

2.5.2 Αναζήτηση στον χώρο των αριστερά – βαθειών πλάνων με Δυναμικό Προγραμματισμό

Η μέθοδος αυτή έχει ενδιαφέρον επειδή είναι η πρώτη μέθοδος που προτάθηκε για βελτιστοποίηση ερωτημάτων ([SAC+79]). Ουσιαστικά πρόκειται για μια εξαντλητική αναζήτηση περιορισμένη στο χώρο των αριστερά – βαθειών πλάνων με αποθήκευση όμως των ενδιάμεσων αποτελεσμάτων με τη λογική του δυναμικού προγραμματισμού. Έχει ανάγκη από εκθετικά αυξανόμενο χώρο αποθήκευσης, δηλαδή υποφέρει από τη λεγόμενη "Curse of dimensionality". Πρακτικά αυτό σημαίνει ότι δεν μπορεί να υποστηρίξει ερωτήματα με περισσότερους από 18 συνδέσμους. Παρακάτω δίνεται ψευδοκώδικας. Ο αλγόριθμος διατηρεί ένα σύνολο από μερικές λύσεις (partial solutions) το οποία αρχικοποιείται με τις σχέσεις του ερωτήματος. Σε κάθε βήμα ο αλγόριθμος τοποθετεί στο σύνολο τα βέλτιστα πλάνα με 1, 2, …, n συνδέσμους, οπότε στο τέλος το σύνολο partial solutions θα περιέχει το βέλτιστο πλάνο.

Αλγόριθμος 2.1: Δυναμικός Προγραμματισμός

1. Αρχικοποίηση
   $partialsolutions \leftarrow \{R_1, R_2, \ldots, R_n\}$
2. for $i = 2$ to n do
   for all $pt \in partialsolutions$
   for all $R \notin pt$
   $pt \leftarrow (pt) \triangleright\triangleleft R$
   Αφαίρεσε από το *partialsolutions* όλα τα στοιχεία για τα οποία
   υπάρχει ισοδύναμο στοιχείο μικρότερου κόστους
3. return $pt \in partialsolutions$

2.5.3 Άπληστη αναζήτηση στο χώρο των αριστερά – βαθειών πλάνων

Μια άπληστη αναζήτηση ξεκινά με μια σχέση και χτίζει το δέντρο αυξητικά επιλέγοντας σε κάθε βήμα την σχέση για την οποία ισχύει μια ευριστική συνθήκη.

Αλγόριθμος 2.2: Άπληστη Αναζήτηση

1. Αρχικοποίηση
   $pt \leftarrow null$
   $rels \leftarrow \{R_1, \ldots, R_n\}$
2. do
      if $(pt = null)$ then
         $R_i = h(rels)$
         $pt \leftarrow R_i$
      else
         $R_i = h(rels)$
         $pt \leftarrow (pt) \rhd\lhd R_i$
      $rels \leftarrow rels - R_i$
3. return *pt*

Ένα γνωστό ευριστικό είναι αυτό της μικρότερης επιλεκτικότητας (minimum selectivity) όπου επιλέγεται κάθε φορά το ενδιάμεσο αποτέλεσμα με το μικρότερο κόστος.

$$h(\{R_i \mid i \in I\}) = \arg\min_{R_i} \left(\sigma_i = \frac{T\left(R_i \rhd\lhd \left(\rhd\lhd \{R_u \mid u \in pt\}\right)\right)}{T(R_i)\, T\left(\rhd\lhd \{R_u \mid u \in pt\}\right)} \right)$$

2.5.4 Αυξητική Βελτίωση

Η εύρεση της σειράς των συνδέσμων μπορεί να ιδωθεί ως ένα πρόβλημα αναζήτησης στον χώρο των πλάνων με τον ακόλουθο τρόπο. Ένα δέντρο εκτέλεσης συνιστά μια κατάσταση, η οποία έχει ένα κόστος. Δύο καταστάσεις είναι γειτονικές αν η μία μπορεί να προκύψει από την άλλη με εφαρμογή ενός κανόνα μετασχηματισμού. Ορίζουμε λοιπόν ένα σύνολο κανόνων μετασχηματισμού, για παράδειγμα.

1. $A \underset{m}{\rhd\lhd} B \rightarrow A \underset{m'}{\rhd\lhd} B$
2. $A \rhd\lhd B \rightarrow B \rhd\lhd A$
3. $(A \rhd\lhd B) \rhd\lhd C \rightarrow A \rhd\lhd (B \rhd\lhd C)$
4. $(A \rhd\lhd B) \rhd\lhd C \rightarrow (A \rhd\lhd C) \rhd\lhd B$
5. $A \rhd\lhd (B \rhd\lhd C) \rightarrow B \rhd\lhd (A \rhd\lhd C)$

Η εύρεση του βέλτιστου πλάνου ανάγεται στην επαναληπτική εφαρμογή μετασχηματισμών (αναζήτηση στο χώρο) με στόχο την εύρεση μιας κατάστασης με μικρό κόστος.

Η μέθοδος της Αυξητικής Βελτίωσης (Iterative Improvement [IK90]) είναι μια hill – climbing μέθοδος αναζήτησης στο χώρο των πλάνων η οποία βρίσκει πάντα ένα τοπικό ελάχιστο και σταματάει σε αυτό. Αρχίζοντας από μια τυχαία κατάσταση, εφαρμόζει ένα τυχαίο μετασχηματισμό για να λάβει μια γειτονική κατάσταση, η οποία γίνεται η τρέχουσα αν έχει μικρότερο κόστος από την προηγούμενη. Η παραπάνω διαδικασία μπορεί να επαναληφθεί πολλές φορές για τυχαία αρχικά σημεία, ελπίζοντας ότι ένα από αυτά θα οδηγήσει σε ένα γενικό ελάχιστο.

Αλγόριθμος 2.3: Αυξητική Βελτίωση

1. Αρχικοποίηση
   mincost ← ∞
2. do
      state ← random starting state
      cost ← $C(state)$
      do
         move ← random move
         newstate ← state after *move*
         newcost ← $C(newstate)$
         if $(newcost < cost)$ then
            state ← *newstate*
            cost ← *newcost*
      while (*state* is not a local minimum)
      if $(cost < mincost)$ then
         minstate ← *state*
         mincost ← *cost*
   while (termination criterion)
3. return *minstate*

2.5.5 Προσομοιωμένη Ανόπτηση

Η Προσομοιωμένη Ανόπτηση (Simulated Annealing) είναι μια ισχυρή στοχαστική μέθοδος γενικής βελτιστοποίησης η οποία προσπαθεί να μην παγιδευτεί σε τοπικά ελάχιστα. Πετυχαίνει το παραπάνω επιλέγοντας κατά καιρούς χειρότερες καταστάσεις (uphill κινήσεις) με μια πιθανότητα με σκοπό να εξερευνήσει το χώρο. Η πιθανότητα αυτή ελέγχεται από μια παράμετρο η οποία ονομάζεται θερμοκρασία και είναι μεγάλη στην αρχή της αναζήτησης ενώ μικραίνει ψύχοντας τη θερμοκρασία με την πάροδο του χρόνου. Η διαδικασία της μείωσης της θερμοκρασίας είναι μια σημαντική παράμετρος του αλγορίθμου και πολλές

εναλλακτικές έχουν προταθεί. Η μέθοδος έχει εφαρμοστεί με αξιόλογα αποτελέσματα ([IW87,][IK90],[IK91]) στην εύρεση της βέλτιστης σειράς των συνδέσμων.

Αλγόριθμος 2.4: Προσομοιωμένη Ανόπτηση

1. Αρχικοποίηση
   $state \leftarrow$ random starting state
   $cost \leftarrow C(state)$
   $minstate \leftarrow state$
   $mincost \leftarrow cost$
   $T \leftarrow$ starting temperature
2. do
      do
         $move \leftarrow$ random move
         $newstate \leftarrow$ state after $move$
         $newcost \leftarrow C(newstate)$
         if $(newcost \le cost)$ then
            $state \leftarrow newstate$
            $cost \leftarrow newcost$
         else with probability $e^{\frac{newcost-cost}{T}}$
            $state \leftarrow newstate$
            $cost \leftarrow newcost$
         if $(cost < mincost)$ then
            $minstate \leftarrow state$
            $mincost \leftarrow cost$
      while ($Equilibrium$ is not reached)
   $T \leftarrow reduce(T)$
   while (not $frozen$)
3. return $minstate$

Τέλος, λόγω της μορφής του χώρου των πλάνων (σχήμα 2.2) έχει προταθεί η μέθοδος 2 Phase Optimization ([IK90],[IK91]) η οποία χρησιμοποιεί την Αυξητική Βελτίωση για να βρει γρήγορα ένα τοπικό ελάχιστο του χώρου, και με αυτό ως αρχικό σημείο κάνει μια τοπική έρευνα με Προσομοιωμένη Ανόπτηση για να βελτιώσει τη λύση.

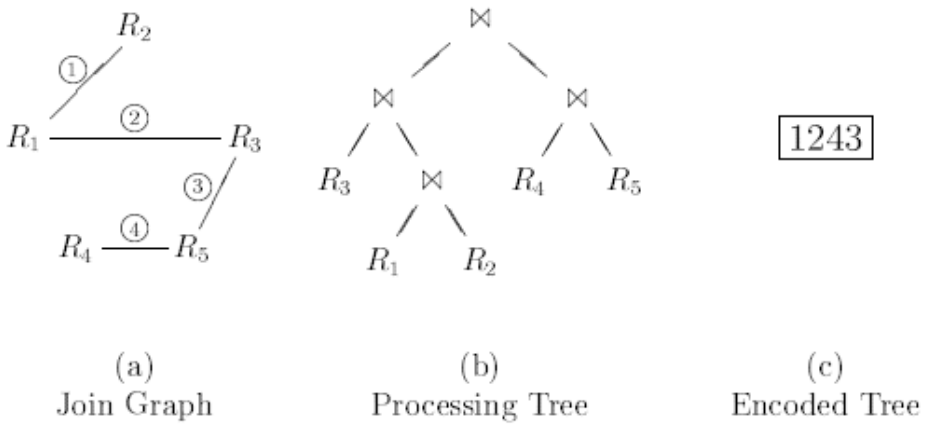
2.5.6 Γενετικοί Αλγόριθμοι

Οι γενετικοί αλγόριθμοι είναι μια πολύ ισχυρή μέθοδος βελτιστοποίησης γενικού σκοπού εμπνευσμένη από τη φυσική εξέλιξη. Για να μπορεί ο γενετικός αλγόριθμος να εφαρμοστεί, πρέπει ο χώρος των καταστάσεων να κωδικοποιηθεί σε μια γραμμική μορφή χρωμοσώματος. Ο γενετικός αλγόριθμος ξεκινά επιλέγοντας ένα αρχικό πληθυσμό από χρωμοσώματα και συνεχίζει εφαρμόζοντας τις πράξεις *επιλογή*, *διασταύρωση* και *μετάλλαξη* δημιουργώντας ένα καινούριο πληθυσμό (γενιά) ο οποίος γίνεται ο τρέχων. Η διαδικασία αυτή συνεχίζεται μέχρι ένα προκαθορισμένο αριθμό γενεών ή μέχρι η καλύτερη λύση που έχει βρεθεί να είναι αποδεκτή. Οι πράξεις που προαναφέρθηκαν είναι οι ακόλουθες.

1. Η επιλογή διαλέγει τα καλύτερα χρωμοσώματα τα οποία θα διασταυρωθούν
2. Η διασταύρωση σχηματίζει ζεύγη χρωμοσωμάτων τα οποία καλούνται γονείς και τους αντικαθιστά με νέα χρωμοσώματα τα οποία καλούνται παιδιά
3. Η μετάλλαξη αλλοιώνει τυχαία κάποια χρωμοσώματα του πληθυσμού

Ο γενετικός αλγόριθμος έχει το πλεονέκτημα ότι διατηρεί ένα σύνολο καταστάσεων σε κάθε χρονική στιγμή και όχι μόνο μία. Βελτιώνει τον πληθυσμό με τις πράξεις της επιλογής και της διασταύρωσης και εξερευνά ταυτόχρονα το χώρο με τις πράξεις της διασταύρωσης και της μετάλλαξης. Έτσι, έχει μικρή πιθανότητα να παγιδευτεί σε ένα τοπικό ελάχιστο. Για περισσότερες πληροφορίες πάνω στους γενετικούς αλγόριθμους παραπέμπουμε στο [Gol89].

Το κύριο βάρος για την εφαρμογή ενός γενετικού αλγόριθμου για την εύρεση της σειράς των συνδέσμων πέφτει στην κωδικοποίηση των καταστάσεων. Μεταξύ άλλων, έχει προταθεί στο [BFI91] η κωδικοποίηση διατεταγμένης λίστας. Το χρωμόσωμα αποτελείται από τους αριθμούς των συνδέσμων με τη σειρά που αυτοί εκτελούνται στο πλάνο, οι οποίοι προκύπτουν από μια αρίθμηση των ακμών στο γράφο συνδέσμων, όπως φαίνεται στο σχήμα 2.3.



Σχήμα 2.3 (πηγή [SMK97])

Εκτός από την κωδικοποίηση, υπάρχουν πολλές άλλες διαστάσεις και παράμετροι προς επιλογή πριν την εφαρμογή ενός γενετικού αλγορίθμου, η περιγραφή των οποίων ξεφεύγει από τα πλαίσια της παρούσας εργασίας. Για μια λεπτομερή παρουσίαση και σύγκριση των αλγορίθμων που αναφέρθηκαν παραπέμπουμε στο [SMK97].

3

Προσαρμοστική Επεξεργασία Ερωτημάτων με Eddies

3.1 Εισαγωγή στην Προσαρμοστική Επεξεργασία Ερωτημάτων

Η βασική προσφορά των ΣΔΒΔ είναι η ανεξαρτησία των εφαρμογών, που δίνονται σε κάποια δηλωτική γλώσσα από τα δεδομένα και τη διαδικασία εκτέλεσής τους. Το κεντρικό σημείο αυτής της ανεξαρτησίας είναι η βελτιστοποίηση ερωτημάτων, η οποία γεφυρώνει το χάσμα ανάμεσα στην περιγραφή και την υλοποίηση και απομονώνει τις στατικές εφαρμογές από το δυναμικό περιβάλλον πάνω στο οποίο τρέχουν. Στην διαδικασία αυτή τίθεται όλη η πίεση του εγχειρήματος. Ο λόγος της αναγκαιότητας της ανεξαρτησίας από τα δεδομένα είναι ότι αυτά αλλάζουν πολύ πιο γρήγορα από τις εφαρμογές, ή

$$\frac{dapp}{dt} \ll \frac{denv}{dt} \tag{3.1.1}$$

Ο τρόπος που ένα ΣΔΒΔ παίρνει υπ' όψιν του τις αλλαγές στο περιβάλλον είναι η *προσαρμοστικότητα* (adaptivity). Ονομάζουμε βρόχο προσαρμοστικότητας την παρακάτω διαδικασία που υπάρχει σε κάθε τύπου ΣΔΒΔ

1. Μετρήσεις και κατασκευή μοντέλου για τα δεδομένα
2. Κατασκευή πλάνου εκτέλεσης για το ερώτημα
3. Εκτέλεση ερωτήματος

Η συχνότητα προσαρμογής καθορίζεται από το βαθμό επικάλυψης των παραπάνω βημάτων.

Ο βελτιστοποιητής ερωτημάτων του System R [SAC+79], ο πρόγονος κάθε σύγχρονου σχεσιακού ΣΔΒΔ κρατούσε ένα σύνολο στατιστικών (μοντέλο) για τα δεδομένα το οποίο αν και δεν προσάρμοζε αυτόματα στις αλλαγές των δεδομένων, έδινε την δυνατότητα αυτή στον διαχειριστή του συστήματος με μια εντολή runstats. Η προσέγγιση αυτή αν και καταλήγει σε μια χαμηλή συχνότητα προσαρμογής έχει αισθητά αποτελέσματα. Ως αποτέλεσμα της ανανέωσης του μοντέλου, η βελτιστοποίηση μπορεί να καταλήξει σε εντελώς διαφορετικά αποτελέσματα για το ίδιο ερώτημα. Η φιλοσοφία αυτή διατηρείται ακόμα σχεδόν σε όλα τα εμπορικά συστήματα αν και η διαδικασία της βελτιστοποίησης και η φύση του μοντέλου έχουν εξελιχθεί. Η διαδικασία βελτιστοποίησης από της άλλη τους συστήματος Ingres [SWKH76] αν και αποδείχθηκε αρκετά αργή για τα περιβάλλοντα της εποχής της, είχε μια σαφώς μεγαλύτερη συχνότητα προσαρμογής (προσαρμογή στη διάρκεια του ερωτήματος).

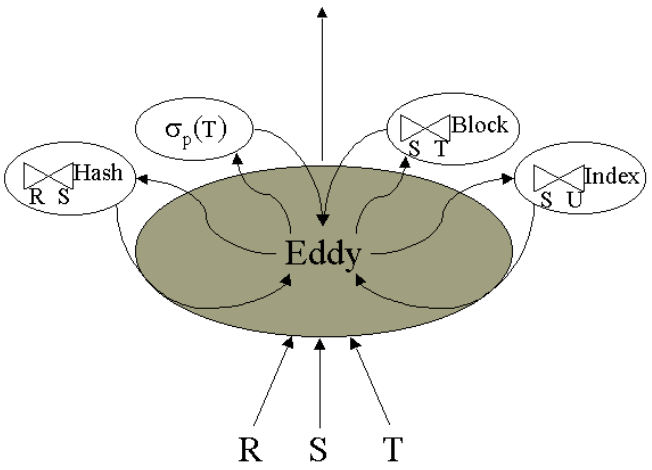
Τα επιχειρήματα ότι ο βαθμός προσαρμοστικότητας αυτών των συστημάτων δεν είναι αρκετός είναι πολλά και οφείλονται κυρίως σε δύο διαστάσεις.

- Πολυπλοκότητα Υλικού και Φορτίου: Σε συστήματα τοποθετημένα σε δίκτυα δεδομένων, συστήματα με πολλούς χρήστες και ΣΔΡΔ (Συστήματα Διαχείρισης Ρευμάτων Δεδομένων), η συμπεριφορά των πόρων (δίκτυο/εξυπηρετητές) και του φορτίου μπορεί είναι εντελώς απρόβλεπτη.
- Πολυπλοκότητα Δεδομένων: Νέου τύπου δεδομένα από μη σχεσιακές πηγές, όπως δεδομένα από το Internet, δεδομένα πολύπλοκων τύπων αλλά και πολύπλοκες συσχετίσεις ανάμεσα στα δεδομένα κάνουν την εκτίμηση επιλεκτικότητας μη ακριβή.

Ο κλάδος της Προσαρμοστικής Επεξεργασία Ερωτημάτων είναι ένα σχετικά νέο νήμα της έρευνας στις Βάσεις Δεδομένων που έχει όμως παράγει ενδιαφέροντες πρακτικές και αλγορίθμους και έχει καταλήξει στην κατασκευή ώριμων συστημάτων (NigaraCQ, TelegraphCQ, STREAM, Tukwilla κα). Η προσαρμογή του μοντέλου γενικά μπορεί να γίνει με αλλαγή του πλάνου στην πορεία της εκτέλεσης ή με διαφορετικές αποφάσεις ανά πλειάδα χωρίς καθολικό πλάνο εκτέλεσης και η συχνότητα της προσαρμογής διαφέρει πολύ ανάμεσα στα συστήματα. Για μια επισκόπηση του πεδίου παραπέμπουμε στα [BB05], [DHR06], [HFC+00]. Το πιο επιθετικό δείγμα της φιλοσοφίας αυτής είναι μάλλον όμως ο μηχανισμός

των Eddies [AH00] ο οποίος αποτελεί και το κύριο αντικείμενο της παρούσας εργασίας. Ο μηχανισμός των eddies (στρόβιλοι) και οι συμπληρωματικές τεχνολογίες των STAIRs και των SteMs αναπροσδιορίζουν την έννοια της επεξεργασίας ερωτημάτων σε Βάσεις Δεδομένων. Αποφασίζουν για την προσαρμογή του πλάνου εκτέλεσης σε επίπεδο πλειάδας αντιμετωπίζοντας ένα πλάνο εκτέλεσης σαν ένα σχήμα δρομολόγησης πλειάδων μεταξύ τελεστών. Την αποστολή της δρομολόγησης αναλαμβάνει ο τελεστής eddy. Τα eddies είναι ο πιο επιθετικός μηχανισμός προσαρμοστικής επεξεργασίας ερωτημάτων που έχει προτεθεί έως τώρα καθώς άρει την ανάγκη ενός πολύπλοκου βελτιστοποιητή ερωτημάτων και ενός συστήματος κράτησης στατιστικών. Οι μηχανισμοί των STAIRs και SteMs που αναπτύχθηκαν γύρω από τα eddies προχωρούν ένα βήμα ακόμη, «σπάζοντας» τους δυαδικούς τελεστές (συνδέσμους) σε μοναδιαίους και αφήνοντας τον eddy να προσομοιώνει, εκτός από τη σειρά εκτέλεσης των συνδέσμων και τους αλγόριθμους συνδέσμων με την πολιτική δρομολόγησης. Έτσι καταρρίπτεται ακόμα και η ανάγκη κώδικα για την εκτέλεση συνδέσμων και όλη η πολυπλοκότητα του συστήματος αποκρυσταλλώνεται στον τρόπο λειτουργίας του eddy και της πολιτικής δρομολόγησης που ακολουθεί. Ο eddy μπορεί να υλοποιηθεί ως ένας ακόμη τελεστής και μπορεί να συνδέεται σε ένα pipeline με άλλα eddies δημιουργώντας έτσι ένα ισχυρό μηχανισμό εκτέλεσης ερωτημάτων.

3.2 Eddies



Σχήμα 3.1 (πηγή [AH00])

Ο eddy [AH00] είναι ένα module που μεσολαβεί ανάμεσα σε ροές δεδομένων και μοναδιαίους ή δυαδικούς τελεστές (επιλογές και συνδέσμους). Σκοπός του είναι να δρομολογεί τις πλειάδες που δέχεται προς τους τελεστές και τα αποτελέσματα του ερωτήματος στην έξοδο με συνεπή τρόπο, δηλαδή η δρομολόγηση να ανταποκρίνεται σε μια σωστή σημασιολογικά εκτέλεση του ερωτήματος και να μην παράγονται διπλότυπα. Η δρομολόγηση συνίσταται από μια σειρά αποφάσεων, ζητούμενο της οποίας είναι να

αντιστοιχεί σε ένα καλό πλάνο εκτέλεσης. Ο eddy θεωρείται ένας τελεστής με ένα οποιονδήποτε αριθμό σχέσεων εισόδου, ένα αριθμό μοναδιαίων και δυαδικών τελεστών και μια μοναδική σχέση εξόδου. Διαθέτει ένα εσωτερικό tuple buffer στον οποίο βρίσκονται οι πλειάδες που έχουν εισέλθει και δεν έχουν φύγει προς την έξοδο σε κάθε δεδομένη στιγμή.

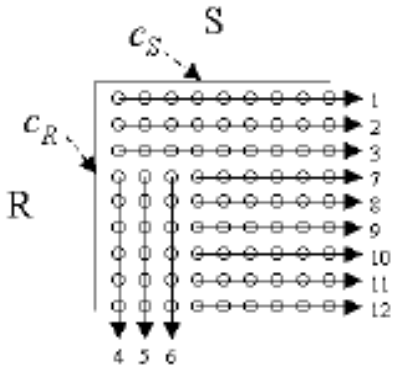
3.3 Εκτέλεση ερωτήματος επιλογών με Eddies

Η εκτέλεση ενός ερωτήματος σύζευξης επιλογών του τύπου $\sigma_{p_1 \wedge \cdots \wedge p_n}(R)$ είναι απλή. Οι τελεστές που αρχικοποιούνται είναι τελεστές επιλογής, ένας scan τελεστής πάνω στην R και ο eddy. Όταν ένας τελεστής επιλογή ς λαμβάνει μια πλειάδα από τον eddy την επιστρέφει αν αυτή ικανοποιεί το κατηγόρημα αλλιώς ειδοποιεί τον eddy με κάποιο τρόπο ότι η πλειάδα δεν πέρασε το κατηγόρημα. Στην περίπτωση αυτή η πλειάδα «διαγράφεται». Ένας σημασιολογικός περιορισμός είναι ότι μια πλειάδα δεν πρέπει να δρομολογηθεί δύο φορές σε ένα τελεστή. Για την ικανοποίησή του εισάγουμε στις πλειάδες μια βοηθητική δομή, τον Περιγραφέα Πλειάδας (Tuple Descriptor) ο οποίος περιέχει ένα bit string το οποίο ονομάζουμε done bits. Αρχικά όλα τα done bits είναι 0. Όταν ένας τελεστής επεξεργαστεί και επιστρέψει μια πλειάδα, «ανάβει» το αντίστοιχο με αυτόν done bit. Μια πλειάδα μπορεί να φύγει προς την έξοδο μόνο όταν όλα τα done bits της είναι 1. Στην γενική περίπτωση που ο eddy έχει είσοδο από πολλές σχέσεις χρειάζεται και άλλος ένας πίνακας με ready bits. Αρχικά ο πίνακας αυτός αντιπροσωπεύει τους σημασιολογικούς περιορισμούς του ερωτήματος (δηλαδή 1 είναι τα ready bits των τελεστών οι οποίοι μπορούν να δεχθούν την πλειάδα) και όταν ένας τελεστής τελειώσει την επεξεργασία μιας πλειάδας «σβήνει» το αντίστοιχο ready bit. Μια πλειάδα μπορεί να δρομολογηθεί σε ένα τελεστή μόνο αν το αντίστοιχο ready bit είναι 1. Στην παρούσα εργασία θα αναφερόμαστε στο προαναφερθέν μοντέλο υπολογισμού με τον όνομα EddySelections.

3.4 Εκτέλεση ερωτήματος συνδέσμων με Eddies και δυαδικούς τελεστές συνδέσμων

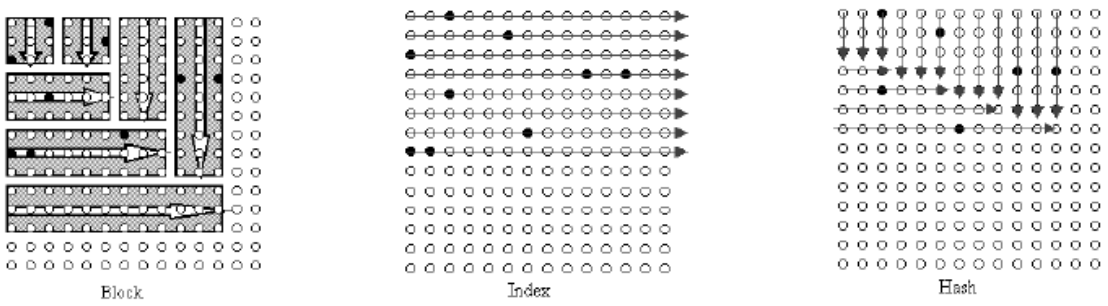
Η βασική πρόκληση που προκύπτει από τη δυναμική αλλαγή του πλάνου κατά το χρόνο εκτέλεσης του ερωτήματος είναι η ενταμιευμένη στους τελεστές συνδέσμων κατάσταση. Ενώ η σειρά των επιλογών μπορεί να αλλάζει αυθαίρετα χωρίς κάποιο επιπλέον κόστος υπολογισμού, οι αλγόριθμοι συνδέσμων συχνά διατηρούν μια κατάσταση (μια αλληλοσυσχέτιση μεταξύ των εισόδων τους, η οποία αντικατοπτρίζεται στην σειρά με την οποία ο τελεστής καταναλώνει πλειάδες από τις εισόδους του) η οποία για να αλλάξει ίσως χρειάζεται σημαντικό υπολογιστικό κόστος. Έστω για παράδειγμα ένα sort – merge join σε

δύο ταξινομημένες σχέσεις: Η σχέση slowlow έχει μικρό ρυθμό αποστολής δεδομένων και χαμηλές τιμές στο πεδίο συνδέσμου, ενώ η σχέση fasthigh έχει υψηλό ρυθμό και μεγάλες τιμές αντίστοιχα. Η κατανάλωση πλειάδων από τη fasthigh πρέπει να περιμένει αρκετό χρόνο μέχρι να καταναλωθούν αρκετές πλειάδες της slowlow. Ονομάζουμε το σημείο αυτό synchronization barrier. Η κατάσταση είναι ακόμα χειρότερη με μη – συμμετρικούς τελεστές όπως ο παραδοσιακός nested loop join. Μετά από κάθε πλειάδα ή μπλοκ πλειάδων που καταναλώνεται από την εξωτερική σχέση τίθεται ένα barrier μέχρι να ολοκληρωθεί μια πλήρης σάρωση της εσωτερικής σχέσης. Ονομάζουμε τα σημεία εκείνα σα οποία μπορεί να αλλάξει η σειρά των εισόδων ενός αλγορίθμου moments of symmetry (τα σημεία κατά τα οποία ο σύνδεσμος φτάνει σε ένα synchronization barrier).



Σχήμα 3.2 (πηγή [AH00])

Έστω για παράδειγμα ένας σύνδεσμος ένθετου βρόχου στις σχέσεις R και S με εξωτερική σχέση την R και εσωτερική της S. Για κάθε πλειάδα της R γίνεται μια πλήρης σάρωση της S και μόλις αυτή τελειώσει τίθεται ένα synchronization barrier (1,2,3 στο σχήμα 3.2). Μόνο σε ένα τέτοιο σημείο μπορεί να αλλάξει η σειρά των εισόδων και η S να γίνει εξωτερική. Αυτό φαίνεται στο σχήμα 3.2 όπου στο σημείο 3 αλλάζει η σειρά, γίνονται σαρώσεις της R και τίθενται τα barriers 4,5,6 όπου η σειρά αλλάζει ξανά. Τα παραπάνω υποθέτουν ότι ο iterator κάθε σχέσης «θυμάται» το σημείο που έμεινε και συνεχίζει από εκεί κάθε φορά που το πλάνο αλλάζει. Το παραπάνω σκεπτικό μπορεί να επεκταθεί σε ένα n-δικό τελεστή συνδέσμου αφού οι σύνδεσμοι προσεταιρίζονται. Ο eddy απλά παρεμβάλλεται ανάμεσα στις εισόδους δεδομένων και τον ιδεατό n-δικό τελεστή συνδέσμου και αλλάζει τη σειρά των εισόδων στα moments of symmetry. Υπάρχουν αλγόριθμοι συνδέσμων που έχουν πολύ συχνά moments of symmetry και σχεδόν ανύπαρκτα synchronization barriers (Ripple join family – [HH99] ή Symmetric Hash Join) οι οποίοι καθίστανται πολύ ελκυστικοί για χρησιμοποίηση με eddies. Στην παρούσα εργασία θα αναφερόμαστε στο προαναφερθέν μοντέλο υπολογισμού με τον όνομα EddyJoins.



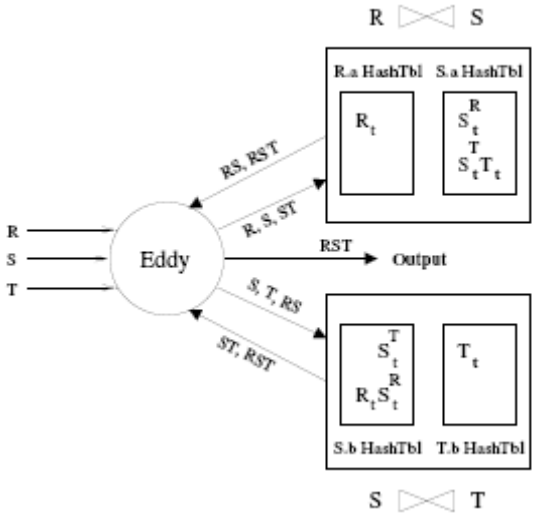
Σχήμα 3.3 (πηγή [AH00])

3.5 Το «βάρος της ιστορίας»

Ο eddy μπορεί να αλλάξει δυναμικά το πλάνο εκτέλεσης σε χρονικά σημεία που οι τελεστές συνδέσμων του δίνουν αυτή τη δυνατότητα (moments of symmetry). Ο μόνος περιορισμός στην απόφαση του eddy να στείλει μια πλειάδα σε κάποιον τελεστή είναι η σημασιολογία του ερωτήματος, πχ αν η πλειάδα έχει την ιδιότητα πάνω στην οποία ασκείται η επιλογή. Σε ερωτήματα όμως πολλών συνδέσμων, η δυνατότητα αλλαγής πλάνου εκτέλεσης εξαρτάται από προηγούμενες αποφάσεις δρομολόγησης και την κατάσταση που βρίσκεται ενταμιευμένη στους τελεστές συνδέσμων, για την οποία ο eddy δεν έχει πληροφόρηση (burden of routing history). Αυτό μπορεί να γίνει καθαρό με το παρακάτω παράδειγμα. Έστω το ερώτημα

```
select  *
from  R.S,T
where  R.a = S.a and S.b = T.b
```

και έστω ότι εκτελείται με δύο συνδέσμους τύπου pipelining hash join



Σχήμα 3.4 (πηγή [DH04])

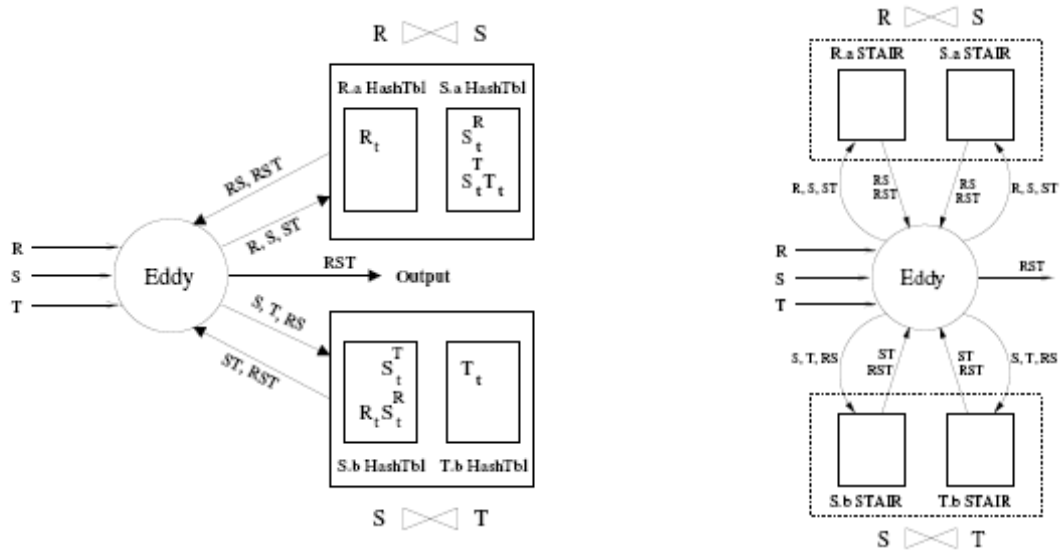
Στην αρχή της εκτέλεσης η σχέση R για κάποιο λόγο δεν παραδίδει πλειάδες στον eddy. Αυτό κάνει τον σύνδεσμο RS ένα πολύ ελκυστικό τελεστή για τις πλειάδες της S, (αφού ποτέ δεν επιστρέφει matches) οι οποίες αποθηκεύονται στον πίνακα κατακερματισμού του S.a. Το αποτέλεσμα είναι ο eddy να προσομοιώνει το στατικό πλάνο $\left(R \triangleright\triangleleft_a S\right) \triangleright\triangleleft_b T$. Αν αργότερα, όταν πλειάδες από την R αρχίσουν να φτάνουν, αποδειχθεί ότι αυτό ήταν λάθος (γιατί πχ. ο $\left(R \triangleright\triangleleft_a S\right)$ έχει μεγάλη επιλεκτικότητα) ο eddy μπορεί να αλλάξει την πολιτική δρομολόγησής του, άρα και το πλάνο εκτέλεσης δρομολογώντας τις πλειάδες της σχέσης S προς τον $S \triangleright\triangleleft_b T$. Δυστυχώς, η απόφαση αυτή έχει παρθεί πολύ αργά αφού όλες οι προηγούμενες πλειάδες της S έχουν αποθηκευτεί στο σύνδεσμο, οι πλειάδες της R πρέπει πρώτα να συνδεθούν με αυτές. Έτσι, αν και ο eddy έχει πληροφορία για το αντίθετο συνεχίζει να προσομοιώνει το αρχικό «κακό» πλάνο εκτέλεσης.

3.6 Εκτέλεση ερωτήματος συνδέσμων με Eddies και STAIRs

Ο τελεστής STAIR (Storage, Transformation and Access for Intermediate Results) που προτάθηκε στο [DH04] ουσιαστικά διαχωρίζει ένα σύνδεσμο σε δύο μοναδιαίους τελεστές και δίνει στον eddy πρόσβαση στην πληροφορία που χρειάζεται σχετικά με την ενταμιευμένη στους συνδέσμους κατάσταση. Ένας τελεστής STAIR στη σχέση R και το πεδίο a (συμβολίζεται με R.a STAIR) αποθηκεύει πλειάδες που περιέχουν όλα τα πεδία της R και ίσως περισσότερα, δηλαδή πλειάδες της R και πλειάδες που έχουν προκύψει από σύνδεσμο πλειάδων της R και πλειάδες άλλων σχέσεων και μπορεί να δεχθεί τις παρακάτω πράξεις:

- $insert\left(R.a, t\right)$: Αν η πλειάδα t περιέχει τα πεδία της σχέσης R, την αποθηκεύει στο STAIR
- $probe\left(R.a, val\right)$: Δεδομένης μιας τιμής val από το $dom\left(R.a\right)$, επιστρέφει όλες τις αποθηκευμένες πλειάδες που ταιριάζουν

Στο σχήμα 3.5 φαίνεται το προαναφερθέν ερώτημα όπως αυτό θα εκτελούνταν με κλασσικούς τελεστές συνδέσμων και με STAIRs.



Σχήμα 3.5 (πηγή [DH04])

Όπως φαίνεται, δημιουργείται ένα STAIR για κάθε στιγμιότυπο κάθε σχέσης που συμμετέχει σε σύνδεσμο (έτσι, για τη σχέση S που συμμετέχει σε δύο συνδέσμους δημιουργούνται δύο STAIRs, ένα στο πεδίο a και ένα στο πεδίο b). Επίσης, τα STAIRs ομαδοποιούνται σε ζεύγη σύμφωνα με τους συνδέσμους στους οποίους συμμετέχουν (και ονομάζονται δυαδικά). Αντί να δρομολογείται μια πλειάδα σε έναν τελεστή συνδέσμου, τώρα ο ίδιος ο eddy πραγματοποιεί ένα insert στο αντίστοιχο STAIR και ένα probe στο άλλο STAIR του ζευγαριού. Δεχόμαστε όπως και στο [DH04] ότι ισχύει η παρακάτω ιδιότητα.

Dual Routing Property: Όποτε μια πλειάδα δρομολογείται για probe σε ένα STAIR, δρομολογείται ταυτόχρονα για insert στο δυαδικό του

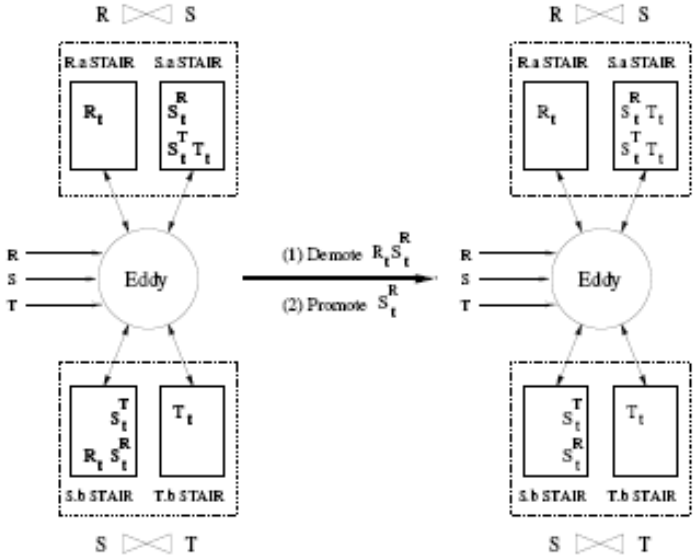
Η παραπάνω ιδιότητα εγγυάται την ορθότητα της εκτέλεσης του ερωτήματος, αλλά ουσιαστικά προσομοιώνει Symmetric Hash Joins σε ένα pipeline. Υπάρχει η δυνατότητα ο περιορισμός αυτός να χαλαρωθεί οπότε ο χώρος των αλγορίθμων να επεκταθεί, αλλά αυτό ξεφεύγει από τα πλαίσια της παρούσας εργασίας.

Εκτός των παραπάνω, ένα STAIR παρέχει πράξεις για τη διαχείριση της κατάστασής του, δηλαδή των αποθηκευμένων σε αυτό πλειάδες. Αυτές είναι οι ακόλουθες:

- $demotion(R.a, t, t')$: Αν η t είναι αποθηκευμένη στο STAIR R.a, η t' είναι υπο – πλειάδα της t, δηλαδή η t περιέχει όλα τα πεδία της t' και ίσως κάποια παραπάνω και η προβολή της t στην R ταυτίζεται με την t' η πράξη αντικαθιστά (υποβιβάζει) την t με την t' στο STAIR R.a
- $promotion(R.a, t, S.b)$: Αν η t είναι αποθηκευμένη στο STAIR R.a, η t περιέχει τα πεδία της σχέσης S και η t δεν περιέχει κανένα πεδίο της T, όπου T.b είναι το δυαδικό STAIR του S.b η πράξη αφαιρεί (διαγράφει) την t από το STAIR R.a, αποθηκεύει την t

στο STAIR S.b, κάνει probe την t στο STAIR T.b και αποθηκεύει τα matches του παραπάνω probe (αν υπάρχουν) στο STAIR R.a

Διαισθητικά, η πράξη demotion είναι ένα "undo" δουλειάς που έχει ήδη γίνει και θα πρέπει να ξαναγίνει και η πράξη Promotion είναι η εκτέλεση ενός join από πλειάδες που έχουν ήδη αποθηκευτεί στα STAIRs (ανήκουν στο παρελθόν). Χρησιμοποιώντας τις δύο παραπάνω πράξεις, μπορούμε να επανέλθουμε στο πρώτο παράδειγμα της ενότητας και να «σηκώσουμε το βάρος της ιστορίας της δρομολόγησης» όπως στο σχήμα 3.6. Στο [DH04] δίνεται απόδειξη ότι η οποιαδήποτε εκτέλεση ερωτήματος με STAIRs υπό την Dual Routing Property παράγει τα σωστά αποτελέσματα ανεξαρτήτως των promotion και demotion που εκτελούνται. Στην παρούσα εργασία θα αναφερόμαστε στο προαναφερθέν μοντέλο υπολογισμού με τον όνομα EddySTAIRs.

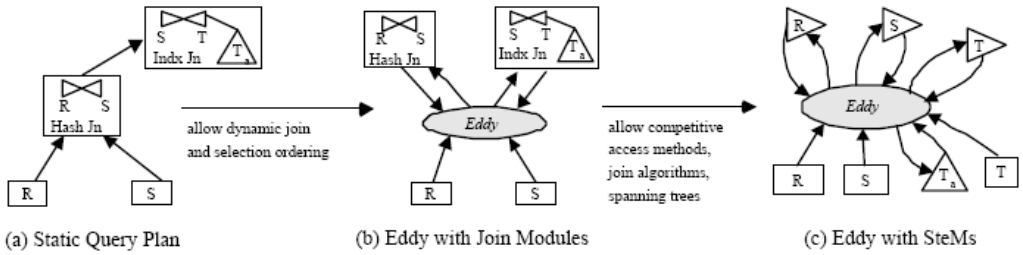


Σχήμα 3.6 (πηγή [DH04])

3.7 Εκτέλεση ερωτήματος συνδέσμων με Eddies και SteMs

Οι αλγόριθμοι συνδέσμων περιλαμβάνουν στην πραγματικότητα πολλούς φυσικούς τελεστές. Ο μηχανισμός των SteMs (State Modules) που προτάθηκε στο [RDH03] προτάθηκε για να ανεξαρτητοποιήσει τους τελεστές αυτούς και να τους κάνει ορατούς στον έλεγχο του eddy. Ένα SteM αντιστοιχεί σε μισό join και είναι ένα λεξικό ομοειδών πλειάδων. Η υλοποίηση του λεξικού αυτού είναι δυναμική και μπορεί να αλλάζει ανάλογα με το μέγεθος των δεδομένων που είναι αποθηκευμένα στο SteM (πχ από συνδεδεμένη λίστα σε πίνακα κατακερματισμού ή δενδρική δομή). Οι αλγόριθμοι συνδέσμων πια δεν προγραμματίζονται αλλά προσομοιώνονται από τα SteMs και τη δυνατότητά τους να προσαρμόζονται και τη πολιτική δρομολόγησης του eddy. Επίσης αίρεται η ανάγκη για κάποιον pre – optimizer καθώς πια το

σύστημα μπορεί να προσαρμόζει το spanning tree και τους αλγόριθμους συνδέσμων. Στο σχήμα 3.7 φαίνεται ένα ερώτημα $\left(R \triangleright\triangleleft_a S\right) \triangleright\triangleleft_b T$ αρχικοποιημένο προς εκτέλεση με τον κλασσικό μηχανισμό των eddies (EddyJoins) και με SteMs. Η εκτέλεση ενός ερωτήματος είναι πια απλή και συνίσταται στη δημιουργία των SteMs και των AMs (παρακάτω) και την δρομολόγηση των πλειάδων σε αυτούς.



Σχήμα 3.7 (πηγή [RDH03])

Οι καινούριοι τελεστές που αντικαθιστούν τους συνδέσμους είναι οι ακόλουθοι

- AMs (Access Modules): Εκφράζουν την έννοια της πρόσβασης σε μια πηγή δεδομένων και μπορεί να είναι scans ή ευρετήρια. Η λειτουργία τους είναι η ακόλουθη: Δέχονται ως είσοδο ένα probe tuple και επιστρέφουν τα matches του καθώς και το ίδιο. Όταν έχουν επιστρέψει όλα τα matches μεταδίδουν μια ειδική πλειάδα, την EOT (End Of Transmission). Τα scans δέχονται ως είσοδο μόνο μια ειδική πλειάδα, την seed tuple η οποία στέλνεται σε αυτά στην αρχή της εκτέλεσης του ερωτήματος, και ακόλουθα μεταδίδουν τα δεδομένα τους και την EOT μόλις αυτά τελειώσουν
- SteMs: Ένα SteM είναι ένα λεξικό ομοειδών πλειάδων το οποίο σχηματίζεται κατά τη διάρκεια εκτέλεσης του ερωτήματος και υποστηρίζει τις πράξεις build (εισαγωγή), probe και eviction (διαγραφή). Ένα SteM μπορεί να κάνει αναζήτησης σε κάθε πεδίο συνδέσμου στον οποίο συμμετέχει μια σχέση. Όταν ένα SteM δεχθεί μια build tuple την αποθηκεύει ανανεώνοντας τις δομές δεδομένων του και ίσως την επιστρέψει στον εδδυ ανάλογα με τους περιορισμούς δρομολόγησης. Όταν ένα SteM δεχθεί μια probe tuple επιστρέφει τα συνδεδεμένα matches της και την ίδια πλειάδα αν δεν έχει αποθηκευμένη μια EOT που να ταιριάζει με αυτήν.

Στον πίνακα 3.1 φαίνονται όλα τα προαναφερθέντα modules μαζί με τα Selection Modules και οι δυνατές δράσεις πάνω σε αυτά

Module	**Πλειάδα εισόδου**	**Πλειάδα εξόδου**	**Δράση**
SM	t	t ή τίποτα	Επιστρέφει την t αν αυτή επαληθεύει το κατηγόρημα

AM	t	t matches της t EOT	ασύγχρονα επιστρέφει την t ασύγχρονα επιστρέφει τα matches της t επιστρέφει την EOT αν όλα τα matches επεστράφησαν
SteM	build(t) EOT probe(t)	- - t ή τίποτα - αποτελέσματα συνδέσμου t ή τίποτα	αποθηκεύει την t στο SteM αποθηκεύει την EOT στο SteM ασύγχρονα επιστρέφει την t βρίσκει τα matches στο SteM ενώνει τα matches με την t και επιστρέφει τα αποτελέσματα ασύγχρονα επιστρέφει την t

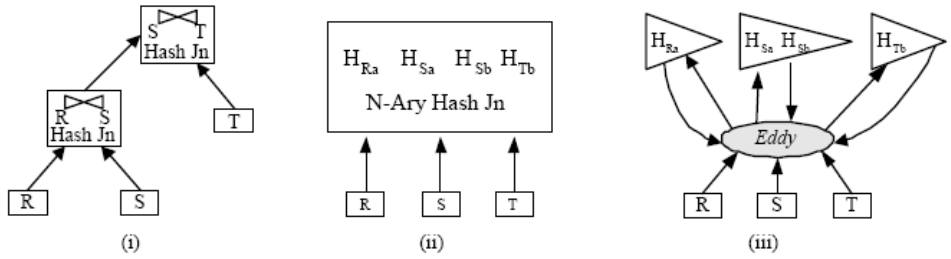
Πίνακας 3.1

Δυστυχώς, η τυχαία δρομολόγηση πλειάδων μέσα στα SteMs και τα AMs μπορεί να δημιουργήσει διπλότυπα και άπειρους βρόχους. Για το λόγο αυτό υπάρχει η ανάγκη εισαγωγής περιορισμών δρομολόγησης (routing constraints) καθώς και κανόνων στην υλοποίης των SteMs. Ακολουθώντας την πορεία του [RDH03] θα αρχίσουμε από την περίπτωση του Symmetric Hash Join και θα χτίσουμε πάνω σε αυτό χαλαρώνοντας σε κάθε βήμα τους περιορισμούς. Κατ' αρχήν ορίζουμε

- singleton tuple t είναι μια πλειάδα από μια σχέση εισόδου (base – table)
- μια πλειάδα που έχει παραχθεί κατά τη διάρκεια της εκτέλεσης του ερωτήματος με σύνδεσμο από τις σχέσεις $T_1,\ldots,T_k$ (οι οποίοι ονομάζονται base – table components) συμβολίζεται με $\langle t_1,\ldots,t_k \rangle$ όπου $t_i = \pi_{T_i}\left(\langle t_1,\ldots,t_k \rangle\right), i = 1,\ldots,k$
- prior prober ονομάζεται μια πλειάδα έστω t που έχει κάνει probe σε ένα SteM έστω το SteMS. Ο πίνακας S ονομάζεται probe completion table της πλειάδας αυτής και οι AMs που υπάρχουν για τον πίνακα αυτό ονομάζονται probe completion AMs

3.7.1 Προσομοίωση ενός n – δικού τελεστή Symmetric Hash Join

Ο τελεστής SHJ είναι ένας pipelined αλγόριθμος συνδέσμου ο οποίος συνεχώς κατασκευάζει δύο πίνακες κατακερματισμού στις δύο εισόδους του. Κάθε φορά που δέχεται μια πλειάδα από τη μια είσοδο του την αποθηκεύει στον πίνακα κατακερματισμού της, κάνει probe τον άλλο πίνακα με αυτή την πλειάδα και οδηγεί τα matches στην έξοδό του. Ο τελεστής αυτός μπορεί να γενικευτεί για $n \geq 3$ εισόδους με δύο τρόπους



Σχήμα 3.8 (πηγή [RDH03])

- Pipelining Symmetric Hash Joins: Ο αρχικός μηχανισμός των eddies (EddyJoins) καθώς και τα eddies με STAIRs (EddySTAIRs) δουλεύουν με αυτό τον τρόπο (Σχήμα 3.8(i)) τοποθετώντας ένα eddy ανάμεσα στους συνδέσμους ώστε να μπορεί να προσομοιώνει όλα τα δέντρα εκτέλεσης. Η ουσία είναι ότι μια πλειάδα της ενδιάμεσης σχέσης (S) αποθηκεύεται μόνο σε έναν από τους δύο πίνακες κατακερματισμού για την S, ανάλογα με το πλάνο εκτέλεσης ή την πολιτική δρομολόγησης που ακολουθείται
- n – way Symmetric Hash Join : Αυτός είναι ένας n-δικός τελεστής που έχει αποθηκευμένους όλους τους πίνακες κατακερματισμού. Όταν δέχεται μια πλειάδα της R (αντίστοιχα T) της αποθηκεύει στον πίνακα κατακερματισμού $H_{R.a}$ (αντίστοιχα $H_{T.b}$) και την κάνει probe στον πίνακα $H_{S.a}$ (αντίστοιχα $H_{S.b}$). Τα matches αν υπάρχουν γίνονται probe στον $H_{T.b}$ (αντίστοιχα $H_{R.a}$). Όταν δέχεται μια πλειάδα της S την αποθηκεύει *και στους δύο* πίνακες κατακερματισμού $H_{S.a}$ και $H_{S.b}$ και την κάνει probe στον $H_{R.a}$ ή στον $H_{T.b}$. Τα matches κάνουν probe στον $H_{T.b}$ ή στον $H_{R.a}$ αντίστοιχα. Τα SteMs ακολουθούν αυτό το μοντέλο, δηλαδή δεν αποθηκεύουν ενδιάμεσα αποτελέσματα και αποθηκεύουν τις πλειάδες των ενδιάμεσων σχέσεων σε όλους τους πίνακες κατακερματισμού (τυπικά κρατάνε μια πολύπλοκη δομή δεδομένων για κάθε σχέση). Για να προσομοιωθεί η παραπάνω εκτέλεση με τα SteMs χρειάζονται οι παρακάτω περιορισμοί

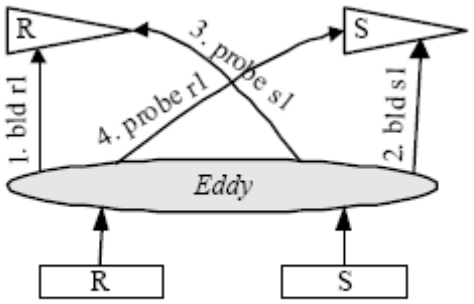
	Τα SteMs είναι υλοποιημένα ως πίνακες κατακερματισμού
BuidFirst	Κάθε singleton tuple (πλειάδα που ανήκει σε σχέση εισόδου) πρέπει πρώτα να γίνει build στο SteM της σχέσης
SteM Bounce Back	Τα SteMs επιστρέφουν μόνο πλειάδες που έχουν δεχθεί για build
Atomicity	Μετά από το build μιας πλειάδας σε ένα SteM, αυτή πρέπει αμέσως μετά να κάνει probe τα υπόλοιπα
BoudedRepetition	Καμιά πλειάδα δεν επιτρέπεται να δρομολογηθεί σε ένα SteM πάνω από μια φορά

Πίνακας 3.2

Οι πρώτοι τρεις περιορισμοί προσομοιώνουν το SHJ και ο τελευταίος εγγυάται τον τερματισμό της εκτέλεσης του ερωτήματος. Στην παρούσα εργασία θα αναφερόμαστε στο προαναφερθέν μοντέλο υπολογισμού με τον όνομα EddySteMs0.

3.7.2 Προσαρμογή των αλγορίθμων συνδέσμων

Για την επέκταση σε άλλους αλγόριθμους συνδέσμου κατ' αρχήν πρέπει να άρουμε τον πρώτο περιορισμό (υλοποίηση των SteMs). Οι δομή δεδομένων πια των SteMs μπορεί να είναι μια συνδεδεμένη λίστα, ένας πίνακας κατακερματισμού ή μια δενδρική δομή και να αλλάζει κατά τη διάρκεια εκτέλεσης του ερωτήματος. Το δεύτερο βήμα είναι να αποσυνδέσουμε τα builds και τα probes μιας πλειάδας, δηλαδή τον τρίτο περιορισμό. Αυτό όμως μπορεί να δημιουργήσει διπλότυπα, όπως στο επόμενο παράδειγμα



Σχήμα 3.9 (πηγή [RDH03])

Η σειρά των πράξεων είναι οι ακόλουθη

1. Η πλειάδα r κάνει build στο SteMR
2. Η πλειάδα s κάνει build στο SteMS
3. Η s κάνει probe Στο SteMR και παράγει την <r,s> η οποία φεύγει στην έξοδο
4. Η r κάνει probe Στο SteMS και παράγει την <r,s> η οποία φεύγει στην έξοδο

Η αντικατάσταση του περιορισμού Atomicity με τον περιορισμό TimeStamp άρει σφάλματα της παραπάνω μορφής. Το παρακάτω set περιορισμών εγγυάται σωστή εκτέλεση του ερωτήματος. Στην παρούσα εργασία θα αναφερόμαστε στο προαναφερθέν μοντέλο υπολογισμού με τον όνομα EddySteMs1.

BuidFirst	Κάθε singleton tuple (πλειάδα που ανήκει σε σχέση εισόδου) πρέπει πρώτα να γίνει build στο SteM της σχέσης
SteM Bounce Back	Τα SteMs επιστρέφουν μόνο πλειάδες που έχουν δεχθεί για build
TimeStamp	Σε κάθε singleton tuple αντιστοιχούμε μια τιμή $TS(t)$ (TimeStamp). Πριν αυτή γίνει build στο αντίστοιχο SteM το $TS(t) = \infty$ ενώ μόλις

	αυτή γίνει build το $TS(t)$ τίθεται στο χρόνο του συστήματος. Για κάθε σύνθετη πλειάδα ισχύει $TS\left(\left\langle t_1,\ldots,t_k\right\rangle\right)=\max\left\{TS(t_1),\ldots TS(t_k)\right\}$. Όταν μια πλειάδα r κάνει probe στο SteMS και βρίσκει κάποιο match s, το αποτέλεσμα $\left\langle r,s\right\rangle$ επιστρέφεται ανν $TS(r)>TS(s)$
BoudedRepetition	Καμιά πλειάδα δεν επιτρέπεται να δρομολογηθεί σε ένα SteM πάνω από μια φορά

Πίνακας 3.3

3.7.3 Ανταγωνιστικά Scans

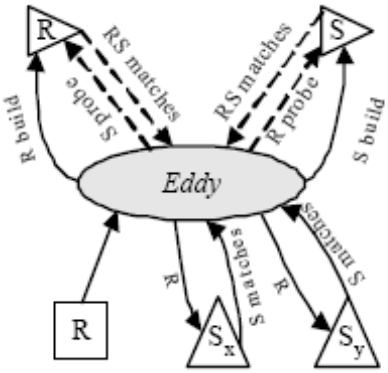
Όταν για μια σχέση έχουμε πολλά scans, διπλότυπα είναι δυνατόν να παραχθούν αφού η ίδια πλειάδα θα παράγεται από πολλά scans. Ο περιορισμός BuildFirst μαζί με τον παρακάτω τροποποιημένο SteM BounceBack όμως αφαιρεί τα διπλότυπα από το dataflow μόλις αυτά γίνουν build στο αντίστοιχο SteM τους. Το παρακάτω set περιορισμών εγγυάται σωστή εκτέλεση του ερωτήματος. Στην παρούσα εργασία θα αναφερόμαστε στο προαναφερθέν μοντέλο υπολογισμού με τον όνομα EddySteMs2.

BuidFirst	Κάθε singleton tuple (πλειάδα που ανήκει σε σχέση εισόδου) πρέπει πρώτα να γίνει build στο SteM της σχέσης
SteM Bounce Back	Τα SteMs επιστρέφουν μόνο πλειάδες που έχουν δεχθεί για build και αυτές μόνο αν δεν είναι διπλότυπα κάποιας πλειάδας που είναι ήδη αποθηκευμένη στο SteM
TimeStamp	Σε κάθε singleton tuple αντιστοιχούμε μια τιμή $TS(t)$ (TimeStamp). Πριν αυτή γίνει build στο αντίστοιχο SteM το $TS(t)=\infty$ ενώ μόλις αυτή γίνει build το $TS(t)$ τίθεται στο χρόνο του συστήματος. Για κάθε σύνθετη πλειάδα ισχύει $TS\left(\left\langle t_1,\ldots,t_k\right\rangle\right)=\max\left\{TS(t_1),\ldots TS(t_k)\right\}$. Όταν μια πλειάδα r κάνει probe στο SteMS και βρίσκει κάποιο match s, το αποτέλεσμα $\left\langle r,s\right\rangle$ επιστρέφεται ανν $TS(r)>TS(s)$
BoudedRepetition	Καμιά πλειάδα δεν επιτρέπεται να δρομολογηθεί σε ένα SteM πάνω από μια φορά

Πίνακας 3.4

3.7.4 Σχέσεις μόνο με indices

Τα AMs ευρετηρίων δημιουργούν ένα επιπρόσθετο πρόβλημα.



Σχήμα 3.10 (πηγή [RDH03])

Όταν δεν υπάρχει scan AM σε μια σχέση όπως παραπάνω στην S, μια πλειάδα της R αφού γίνει probe στο SteMS (έστω ότι το SteMS δεν περιέχει ακόμα όλα τα matches) πρέπει τώρα να επιστραφεί για να κάνει probe στα index AMs, αν από την πολιτική δρομολόγησης επιλέχθηκε να κάνει πρώτα probe στο SteMS. Οπότε χαλαρώνουμε τον περιορισμό SteM BounceBack όπως παρακάτω. Στην παρούσα εργασία θα αναφερόμαστε στο προαναφερθέν μοντέλο υπολογισμού με τον όνομα EddySteMs3.

BuidFirst	Κάθε singleton tuple (πλειάδα που ανήκει σε σχέση εισόδου) πρέπει πρώτα να γίνει build στο SteM της σχέσης
SteM Bounce Back	Τα SteMs επιστρέφουν πλειάδες που έχουν δεχθεί για build αν δεν είναι διπλότυπα κάποιας πλειάδας που είναι ήδη αποθηκευμένη στο SteM και πλειάδες που έχουν δεχθεί για probe εκτός αν το SteM περιέχει όλα τα matches τους ή η σχέση τους έχει scan AM
TimeStamp	Σε κάθε singleton tuple αντιστοιχούμε μια τιμή $TS(t)$ (TimeStamp). Πριν αυτή γίνει build στο αντίστοιχο SteM το $TS(t)=\infty$ ενώ μόλις αυτή γίνει build το $TS(t)$ τίθεται στο χρόνο του συστήματος. Για κάθε σύνθετη πλειάδα ισχύει $TS\left(\langle t_1,\ldots,t_k\rangle\right)=\max\left\{TS(t_1),\ldots TS(t_k)\right\}$. Όταν μια πλειάδα r κάνει probe στο SteMS και βρίσκει κάποιο match s, το αποτέλεσμα $\langle r,s\rangle$ επιστρέφεται ανν $TS(r)>TS(s)$
BoudedRepetition	Καμιά πλειάδα δεν επιτρέπεται να δρομολογηθεί σε ένα SteM πάνω από μια φορά

Πίνακας 3.5

3.7.5 Κυκλικά ερωτήματα

Αν θέλουμε το spanning tree του ερωτήματος να αλλάζει δυναμικά τα εναλλακτικά spanning trees παράγουν διπλότυπα, τα οποία όμως αντιμετωπίζονται με τον περιορισμό

ProbeCompletion. Το σύνολο των περιορισμών είναι το ακόλουθο. Στην παρούσα εργασία θα αναφερόμαστε σε αυτό το μοντέλο υπολογισμού με τον όνομα EddySteMs4.

BuidFirst	Κάθε singleton tuple (πλειάδα που ανήκει σε σχέση εισόδου) πρέπει πρώτα να γίνει build στο SteM της σχέσης
SteM Bounce Back	Τα SteMs επιστρέφουν πλειάδες που έχουν δεχθεί για build αν δεν είναι διπλότυπα κάποιας πλειάδας που είναι ήδη αποθηκευμένη στο SteM και πλειάδες που έχουν δεχθεί για probe εκτός αν το SteM περιέχει όλα τα matches τους ή η σχέση τους έχει scan AM
TimeStamp	Σε κάθε singleton tuple αντιστοιχούμε μια τιμή $TS(t)$ (TimeStamp). Πριν αυτή γίνει build στο αντίστοιχο SteM το $TS(t)=\infty$ ενώ μόλις αυτή γίνει build το $TS(t)$ τίθεται στο χρόνο του συστήματος. Για κάθε σύνθετη πλειάδα ισχύει $TS\left(\langle t_1,\ldots,t_k\rangle\right)=\max\{TS(t_1),\ldots TS(t_k)\}$. Όταν μια πλειάδα r κάνει probe στο SteMS και βρίσκει κάποιο match s, το αποτέλεσμα $\langle r,s\rangle$ επιστρέφεται ανν $TS(r)>TS(s)$
ProbeCompletion	Μια πλειάδα t prior prober δεν μπορεί να κάνει probe σε ένα SteM άλλο από αυτό του probe completion table του και παραμένει στο dataflow μέχρι να κάνει Probe σε ένα από τα probe completion AMs της
BoudedRepetition	Καμιά πλειάδα δεν επιτρέπεται να δρομολογηθεί σε ένα SteM πάνω από μια φορά

Πίνακας 3.6

3.7.6 Χαλάρωση του περιορισμού BuildFirst

Ο περιορισμός BuildFirst είναι πολύ αυστηρός ειδικά για πολύ μεγάλες σχέσεις, τις οποίες θα θέλαμε να κάνουμε probe κατ' ευθείαν σε άλλα SteMs και να μην αποθηκεύουμε τις πλειάδες τους, δηλαδή να κατασκευάζουμε ευρετήριο μόνο στη μια είσοδο του συνδέσμου. Μπορούμε να χαλαρώσουμε τον περιορισμό αυτό μόνο αν υπάρχει μόνο ένα scan AM πάνω στη σχέση (αλλιώς δημιουργούμε διπλότυπα). Επίσης, πρέπει να αφήσουμε τη δυνατότητα μια πλειάδα του dataflow να μπορεί να κάνει πολλαπλά probe στο ίδιο SteM κατά την εκτέλεση, αφού μπορεί το SteM να μην περιέχει την πρώτη φορά όλα τα matches, να χαλαρώσουμε λοιπόν τον περιορισμό BoundedRepetition. Το κάνουμε αυτό εισάγοντας ένα πεδίο LastMatchTimeStamp στα tuple descriptors, το οποίο κρατάει το μέγιστο TS των matches κάθε φορά που μια πλειάδα κάνει probe σε ένα SteM. Αποδεικνύεται στο [RDH03] ότι με

τους παρακάτω περιορισμούς δεν παράγονται διπλότυπα στη έξοδο του eddy. Αυτοί είναι και το πιο γενικό set περιορισμών (EddyJoins5)

Περιορισμοί στην πολιτική δρομολόγησης	
BoundedRepetition	Μια πλειάδα δεν μπορεί να δρομολογηθεί σε ένα τελεστή περισσότερες από ένα πεπασμένο αριθμό φορών
BuildFirst	Μια singleton tuple πρέπει αρχικά να γίνει build στο αντίστοιχο SteM ανν η σχέση έχει πολλά AMs ή έχει ένα AM ευρετηρίου
ProbeCompletion	Μια πλειάδα t prior prober δεν μπορεί να κάνει probe σε ένα SteM άλλο από αυτό του probe completion table του και παραμένει στο dataflow μέχρι να κάνει Probe σε ένα από τα probe completion AMs της
Περιορισμοί στην υλοποίηση των SteMs και των AMs	
SteM BounceBack	Τα SteMs επιστρέφουν πλειάδες που έχουν δεχθεί για build αν δεν είναι διπλότυπα κάποιας πλειάδας που είναι ήδη αποθηκευμένη στο SteM και πλειάδες που έχουν δεχθεί για probe εκτός αν το SteM περιέχει όλα τα matches τους ή η σχέση τους έχει scan AM
TimeStamp	Σε κάθε singleton tuple αντιστοιχούμε μια τιμή $TS(t)$ (TimeStamp). Πριν αυτή γίνει build στο αντίστοιχο SteM το $TS(t)=\infty$ ενώ μόλις αυτή γίνει build το $TS(t)$ τίθεται στο χρόνο του συστήματος. Για κάθε σύνθετη πλειάδα ισχύει $TS\left(\langle t_1,\ldots,t_k\rangle\right)=\max\left\{TS(t_1),\ldots TS(t_k)\right\}$. Όταν μια πλειάδα r κάνει probe στο SteMS και βρίσκει κάποιο match s, το αποτέλεσμα $\langle r,s\rangle$ επιστρέφεται ανν $TS(r)>TS(s)$

Πίνακας 3.7

3.8 STAIRs vs. SteMs

Οι προσέγγιση των STAIRs και των SteMs της είναι ισοδύναμες με την έννοια ότι προσπαθούν να αντιμετωπίσουν το ίδιο πρόβλημα εκτός ενός σημείου: Τα SteMs δεν αποθηκεύουν κανένα ενδιάμεσο αποτέλεσμα ενώ τα STAIRs αποθηκεύουν όλα τα ενδιάμεσα αποτελέσματα. Ουσιαστικά τα πρώτα αντιστοιχούν σε μια προσαρμοστική εκτέλεση ενός n – way SHJ ενώ τα δεύτερα σε προσαρμοστική επιλογή σε όλα τα δέντρα από pipelined binary SHJs. Η επέκταση των STAIRs για να χειριστούν πολλούς αλγορίθμους συνδέσμων είναι άμεση, θέτοντας παρόμοιους περιορισμούς όπως στην παραπάνω ενότητα. Ήδη στο [DH04]

αναφέρεται πώς τα STAIRs μπορούν να προσομοιώσουν τη λειτουργία των SteMs. Η διαφορά που παραμένει είναι ουσιαστικά ένα tradeoff ανάμεσα στη μνήμη που χρησιμοποιείται και στην επίδοση (χρόνο εκτέλεσης). Αντικείμενο τρέχουσας έρευνας είναι πώς μπορεί να υλοποιηθεί μια μέση λύση σε οποιαδήποτε από τις δύο αρχιτεκτονικές, στην οποία η αποθήκευση ενδιάμεσων αποτελεσμάτων θα γίνεται επιλεκτικά από τον eddy. Στην αρχιτεκτονική των SteMs αυτό μπορεί να γίνει χρησιμοποιώντας SteMs πάνω σε μη – singleton tuples, ενώ σε αυτή των STAIRs είναι να εφαρμόζεται επιλεκτικά η πράξη insert.

3.9 Προταθείσες πολιτικές δρομολόγησης

Στο μοντέλο επεξεργασίας ερωτημάτων με eddies και ειδικά με την εισαγωγή των STAIRs και των SteMs όλο το βάρος της απόδοσης του συστήματος πέφτει στην πολιτική δρομολόγησης του eddy. Η πολιτική αυτή πρέπει ανάλογα με το περιβάλλον και τις αλλαγές του να πλησιάζει ένα βέλτιστο πλάνο εκτέλεσης (ή περίπου βέλτιστο) αλλά να μπορεί γρήγορα να αλλάζει και να ανταποκρίνεται στη δυναμική του περιβάλλοντος. Επίσης πρέπει να είναι υπολογιστικά ελαφριά ώστε το βάρος του υπολογισμού να πέφτει στην πραγματική εκτέλεση του ερωτήματος και όχι στην απόφαση ή την αλλαγή της πολιτικής. Θα μπορούσαμε να αναλύσουμε την πολιτική δρομολόγησης σε δύο συνιστώσες

- Η πρώτη απόφαση που πρέπει να πάρει ο eddy είναι ποια πλειάδα από το dataflow θα επιλέξει προς επεξεργασία και αντικατοπτρίζεται από την υλοποίηση του tuple buffer. Αφού τελικά όλες οι πλειάδες θα εξεταστούν, η συνιστώσα αυτή παίζει ρόλο μόνο στο throughput του συστήματος και όχι στο συνολικό χρόνο εκτέλεσης, και μάλιστα μόνο σε παράλληλα περιβάλλοντα
- Η δεύτερη και πιο σημαντική συνιστώσα είναι η επιλογή του τελεστή στον οποίο θα σταλεί η επιλεχθείσα πλειάδα

Στο υπόλοιπο της ενότητας παρουσιάζουμε τις πολιτικές δρομολόγησης που έχουν προταθεί στη βιβλιογραφία.

3.9.1 Naive Routing Policy

Ο tuple buffer υλοποιείται σαν μια priority queue με το εξής σχήμα προτεραιοτήτων: Οι πλειάδες έχουν χαμηλή προτεραιότητα όταν εισέρχονται στον eddy και λαμβάνουν υψηλή προτεραιότητα όταν επιστρέφονται από κάποιον τελεστή. Το σχήμα αυτό εγγυάται ότι οι πλειάδες ρέουν σε όλους τους τελεστές πριν καινούριες πλειάδες καταναλωθούν από τις εισόδους. Η επιλογή του τελεστή γίνεται τυχαία ανάμεσα στους νόμιμου τελεστές που μπορούν να δεχθούν την πλειάδα. Αν και φαντάζει τετριμμένη, η πολιτική αυτή σε ένα παράλληλο περιβάλλον μπορεί να ανταποκριθεί σε αλλαγές του consumption rate (δηλαδή

του κόστους) των τελεστών. Τελεστές με μεγάλο κόστος είναι απασχολημένοι περισσότερη ώρα και παράγουν πλειάδες με μικρό ρυθμό, οπότε πλειάδες με μικρή προτεραιότητα έχουν μικρή πιθανότητα να δρομολογηθούν σε αυτούς (back – pressure effect) [AH00].

3.9.2 Deterministic Rank Ordering

Η πολιτική αυτή (μόνο για τελεστές επιλογής) κρατάει συνεχώς πληροφορία για τις επιλεκτικότητες και τα κόστη των τελεστών και εφαρμόζει περιοδικά τον αλγόριθμο KBZ [KBZ86] για να επανέλθει σε βέλτιστη σειρά επιλογών [DHR06].

3.9.3 Lottery Scheduling

Η πολιτική αυτή [WW94], [AH00] μπορεί να εφαρμοστεί σε συστήματα στα οποία κάθε τελεστής τρέχει σε ξεχωριστό thread (ή σε event – driven query executors). Διατηρεί δύο στοιχεί για κάθε τελεστή: Ένα bit busy/idle για το αν ο τελεστής είναι απασχολημένος ή όχι και ένα αριθμό από εισιτήρια (tickets). Ένας τελεστής μπορεί να λάβει μια πλειάδα μόνο αν είναι idle. Όταν ένας τελεστής λάβει μια πλειάδα παίρνει από τον eddy ένα ticket, το οποίο το επιστρέφει μόνο αν επιστρέψει και την πλειάδα. Ο eddy για να αποφασίσει σε ποιον τελεστή θα στείλει μια πλειάδα διοργανώνει μια λοταρία, οπότε η πιθανότητα ένας τελεστής αν λάβει μια πλειάδα είναι ανάλογη προς τον αριθμό των tickets που έχει. Η πολιτική αυτή παίρνει (έμμεσα) υπ' όψιν της τόσο τα κόστη όσο και της επιλεκτικότητες και παράλληλα κάνει μη – βέλτιστες επιλογές (εξερευνά το χώρο)

3.10 Ένα σύγχρονο περιβάλλον εκτέλεσης ερωτημάτων

Ο μηχανισμός των eddies μπορεί να δουλέψει σε ένα ευρύ φάσμα συστημάτων, από παραδοσιακά ΣΔΒΔ μέχρι δυναμικά ΣΔΡΔ. Έχει υλοποιηθεί τόσο σε ένα στατικό περιβάλλον στα πλαίσια της PostgreSQL [Des04] όσο και σε ένα δυναμικό ασύγχρονο περιβάλλον στα πλαίσια του TelegraphCQ. Στην ενότητα αυτή δίνουμε μια εικόνα για το πώς ένα ερώτημα με eddies μπορεί να εκτελεστεί σε ένα σύγχρονο iterator – based περιβάλλον, ενώ στην επόμενη ενότητα δίνουμε τη δεύτερη περίπτωση.

Υποθέτουμε ότι έχουμε ένα single – threaded σύστημα (η επέκταση σε multi – threaded είναι άμεση) όπου κάθε τελεστής έχει μια μέθοδο get_next(), η οποία όταν καλεστεί επιστρέφει την επόμενη πλειάδα του τελεστή μετά την επεξεργασία της. Υποθέτουμε ότι έχουμε τους τελεστές Scan, Selection, Join και Eddy, δηλαδή το μοντέλο υπολογισμού είναι το EddySelections σε συνδυασμό με ένα από τα EddyJoins, EddySTAIRs και EddySteMs0.

Για την εκτέλεση ενός ερωτήματος, ο eddy κρατάει μια στοίβα από τελεστές join που έχουν και άλλες πλειάδες να επιστρέψουν (active operators). Όταν ένας τελεστής σε μεγαλύτερο επίπεδο καλέσει την get_next() του eddy, αυτός κάνει τα ακόλουθα

1. Αν η στοίβα των τελεστών δεν είναι άδεια καλεί την get_next() του τελεστή που είναι στην κορυφή της στοίβας. Αν αυτή επιστρέψει null, αφαιρεί τον τελεστή από τη στοίβα και επαναλαμβάνει το βήμα 1, αλλιώς αν ο τελεστής επιστρέψει μια πλειάδα η εκτέλεση συνεχίζεται από το βήμα 3.
2. Αν η στοίβα είναι άδεια ο eddy διαλέγει μια πηγή δεδομένων (scan) και καλεί σε αυτή την get_next() φέρνοντας μια καινούρια πλειάδα στο dataflow. Αν καμμία πηγή δεδομένων δεν έχει άλλες πλειάδες, η εκτέλεση του ερωτήματος έχει τελειώσει και ο eddy επιστρέφει null.
3. Ο eddy ελέγχει αν η τρέχουσα πλειάδα μπορεί να φύγει στην έξοδο. Αν μπορεί, την επιστρέφει στον καλούντα και η εκτέλεση τελειώνει. Αλλιώς, επιλέγει έναν τελεστή και δρομολογεί την πλειάδα στον τελεστή αυτό. Αν ο τελεστής είναι join, τον τοποθετεί στην κορυφή της στοίβας και επιστρέφει στο βήμα 1.

3.11 Ένα ασύγχρονο περιβάλλον εκτέλεσης ερωτημάτων

Ένα άλλο δυνατό περιβάλλον υλοποίησης είναι ένα εντελώς ασύγχρονο multi – threaded περιβάλλον. Σε αυτό, κάθε τελεστής τρέχει σε ξεχωριστό thread και δεν υπάρχει συγχρονισμός μεταξύ των τελεστών (κεντρικό «ρολόι» συστήματος). Οι πηγές δεδομένων (scans και indices) παράγουν πλειάδες κατά βούληση τις οποίες στέλνουν με ένα ρυθμό στον eddy. Ο eddy είναι υπεύθυνος να καταναλώνει τις πλειάδες με αρκετά μεγάλο ρυθμό ώστε ο tuple buffer του να μην υπερχειλίζει. Ο eddy επίσης στέλνει και λαμβάνει πλειάδες από τους τελεστές ασύγχρονα, δηλαδή δεν περιμένει ένας τελεστής να τελειώσει την επεξεργασία μιας πλειάδας και να επιστρέψει κάτι στον eddy για να συνεχίσει. Η ορθότητα μιας τέτοιας εκτέλεσης δεν είναι προφανής. Ένα τέτοιο σύστημα έχει προταθεί για την υλοποίηση των SteMs [RDH03] και χρησιμοποιεί ειδικές πλειάδες EOT και περιορισμούς δρομολόγησης για την εγγύηση της ορθότητας εκτέλεσης. Το θετικό της φιλοσοφίας αυτής εκτός από την άμεση παραλληλοποίηση είναι ότι ο eddy δεν χρειάζεται να υπολογίζει τα κόστη των τελεστών λόγω του φαινόμενου back – pressure ([AH00]).

3.12 Σύνοψη

Συνδέοντας ένα τελεστή eddy στο πλάνο εκτέλεσης ανάμεσα στις πηγές δεδομένων και τους τελεστές επεξεργασίας τους, δίνεται η δυνατότητα της προσαρμογής του πλάνου σε επίπεδο πλειάδας. Η εκτέλεση του ερωτήματος μετασχηματίζεται σε μια διαδικασία δρομολόγησης

πλειάδων στους τελεστές, με τον eddy στο ρόλο του δρομολογητή. Η προσαρμοστικότητα βέβαια συνεπάγεται και ένα κόστος αφού ο eddy πρέπει να κρατά επιπλέον πληροφορία για τις πλειάδες ώστε να μπορεί να κάνει μια σωστή δρομολόγηση. Ενώ ερωτήματα με τελεστές επιλογής μπορούν να αναπροσαρμόζονται σε κάθε χρονική στιγμή, δεν ισχύει το ίδιο με τους τελεστές συνδέσμου, οι οποίοι διατηρούν μια ενταμιευμένη κατάσταση και επιτρέπουν στο πλάνο να προσαρμόζεται μόνο σε καλά ορισμένες χρονικές στιγμές. Για το λόγο αυτό, συνίσταται η χρησιμοποίηση αλγορίθμων συνδέσμων με συχνές τέτοιες στιγμές (moments of symmetry) όπως η οικογένεια Ripple Joins. Ένα επιπρόσθετο πρόβλημα των συνδέσμων είναι ότι η κατάσταση τους επηρεάζει τις μελλοντικές αποφάσεις δρομολόγησης αναγκάζοντας τον eddy να ακολουθεί ένα πλάνο εκτέλεσης ενώ γνωρίζει ότι υπάρχει κάποιο καλύτερο. Δύο λύσεις έχουν προταθεί: ο τελεστής STAIR και ο τελεστής SteM, οι οποίες αποτελούν δύο ακραίες περιπτώσεις. Στην πρώτη, όλα τα ενδιάμεσα αποτελέσματα των συνδέσμων αποθηκεύονται, αλλά στον eddy παρέχονται πράξεις αλλαγής της κατάστασης, ενώ στη δεύτερη τα ενδιάμεσα αποτελέσματα πρέπει να επαναϋπολογιστούν. Ένα πλήθος προσεγγίσεων δρομολόγησης έχουν προταθεί στη βιβλιογραφία που παίρνουν υπ' όψιν τα κόστη καθώς και τις επιλεκτικότητες των τελεστών. Τα eddies έχουν υλοποιηθεί στο περιβάλλον του TelegraphCQ καθώς και της PostgreSQL με υποσχόμενα πειραματικά αποτελέσματα.

4

Εισαγωγή στην Ενισχυτική Μάθηση

4.1 Εισαγωγή

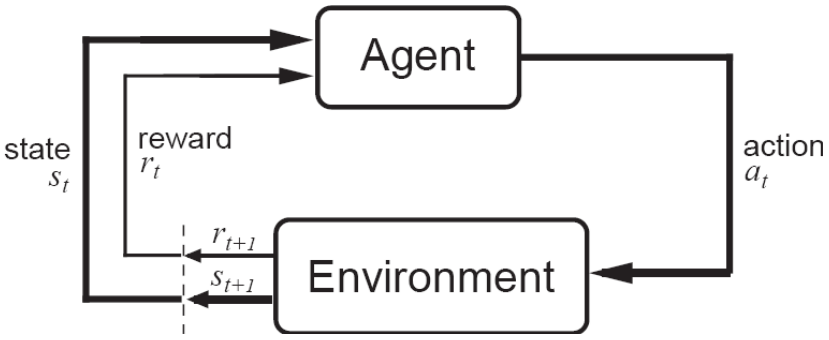
Η Ενισχυτική Μάθηση (Reinforcement Learning) [SB98],[KLM96] είναι ο κλάδος της Τεχνητής Νοημοσύνης και της Μηχανικής Μάθησης που ασχολείται με τη μάθηση μιας συμπεριφοράς (δηλαδή μιας αντιστοίχησης καταστάσεων σε δράσεις) με στόχο την μεγιστοποίηση ενός ποσοτικού σήματος. Ο «μαθητευόμενος» (πράκτορας) δεν έχει πληροφορία για το ποια δράση είναι η πιο κατάλληλη, αλλά πρέπει να βγάλει τα συμπεράσματά του από την ενίσχυση που λαμβάνει για τις δράσεις του. Μια δράση δεν επηρεάζει στη γενική κατάσταση μόνο την ενίσχυση που θα ληφθεί, αλλά και την επόμενη κατάσταση και μέσω αυτής την επόμενη ενίσχυση κοκ. Αυτά τα δύο σημεία, δηλαδή η έρευνα τύπου δοκιμής και λάθους και η καθυστερημένη ενίσχυση (delayed reward) είναι τα πιο χαρακτηριστικά σημεία της Ενισχυτικής Μάθησης. Η Ενισχυτική Μάθηση διαφέρει από την επιβλεπόμενη μάθηση (supervised learning) στο ότι ο πράκτορας δεν διαθέτει παραδείγματα σωστής συμπεριφοράς από ένα εξωτερικό δάσκαλο (training set). Μια από τις μεγαλύτερες προκλήσεις που παρουσιάζονται είναι το πρόβλημα της ισορροπίας ανάμεσα στην αναζήτηση για γνώση και την εκμετάλλευση της γνώσης (exploration – exploitation tradeoff). Ο πράκτορας μαθαίνει στον ίδιο χρόνο με τον οποίο δρα, πρέπει να προτιμά δράσεις οι οποίες σύμφωνα με τη γνώση του επιστρέφουν μεγάλη ενίσχυση αλλά πρέπει

ταυτόχρονα να εξερευνά το χώρο των δράσεων προσπαθώντας να αυξήσει τη γνώση του για άλλες δράσεις.

Το πεδίο της Ενισχυτικής Μάθησης προέρχεται από τη σύγκλιση της έρευνας πάνω σε δύο τομείς. Ο πρώτος τομές αφορά μάθηση δοκιμής και σφάλματος που άρχισε στην ψυχολογία και στη μάθηση στο ζωικό βασίλειο και επανήλθε με την Τεχνητή Νοημοσύνη τη δεκαετία του 1980. Ο δεύτερος, προέρχεται από το πεδίο του Βέλτιστου Ελέγχου και της έρευνας στο Δυναμικό Προγραμματισμό, που ασχολείται με τη σχεδίαση ενός ελεγκτή για ένα σύστημα που θα ελαχιστοποιεί μια μετρική της δυναμικής συμπεριφοράς του συστήματος στο χρόνο.

4.2 Το πρόβλημα της Ενισχυτικής Μάθησης και η Ιδιότητα Markov

Το πρόβλημα της Ενισχυτικής Μάθησης (Reinforcement Learning) μπορεί να διατυπωθεί ως εξής [SB98]:



Σχήμα 4.1 (πηγή [SB98])

Ένας *πράκτορας* (agent) επικοινωνεί με σαφή τρόπο με το *περιβάλλον* του (environment). Σε κάθε χρονική στιγμή, ο πράκτορας δέχεται ένα σήμα από το περιβάλλον του, το οποίο ονομάζουμε *κατάσταση* (state). Σύμφωνα με την πληροφορία αυτή, ο πράκτορας διαλέγει μια *δράση* (action) από ένα σύνολο δυνατών δράσεων που μπορεί να κάνει στην παρούσα κατάσταση και την πραγματοποιεί. Η δράση αυτή αλλάζει την κατάσταση του περιβάλλοντός του, οπότε ο πράκτορας λαμβάνει από το περιβάλλον τη νέα κατάσταση καθώς και ένα *σήμα ενίσχυσης* (reinforcement signal). Στόχος του πράκτορα είναι να μάθει μια *πολιτική*, δηλαδή μια απεικόνιση από το σύνολο των καταστάσεων στο σύνολο των δράσεων η οποία μεγιστοποιεί κάποια συνάρτηση των σημάτων ενίσχυσης στο μέλλον. Ο πράκτορας πρέπει να εκμεταλλεύεται τη γνώση του για να μεγιστοποιήσει την ενίσχυσή του αλλά παράλληλα και να εξερευνά το χώρο των καταστάσεων και των δράσεων (δηλαδή αν κάνει μη – βέλτιστες κινήσεις) για την απόκτηση νέας γνώσης. Όσο η γνώση του περιβάλλοντος γίνεται καλύτερη, τόσο ο πράκτορας πρέπει να δίνει περισσότερη πιθανότητα στην εκμετάλλευση από την εξερεύνηση.

Στη γενική του περίπτωση, το πρόβλημα της ενισχυτικής μάθησης μπορεί να μοντελοποιηθεί ως εξής. Έστω:

$t \in \{0,1,2\ldots\}$ ο χρόνος

$s_t \in S$ η κατάσταση του περιβάλλοντος τη χρονική στιγμή t

$a_t \in A(s_t)$ η δράση που κάνει ο πράκτορας τη χρονική στιγμή t (με $A(s) \subseteq A$ συμβολίζουμε το σύνολο των δυνατών δράσεων για την κατάσταση s. Επίσης συμβολίζουμε $A = \bigcup_{s \in S} A(s)$)

$r_t \in \mathbb{R}$ το σήμα ενίσχυσης τη χρονική στιγμή t

Το σχήμα μάθησης που ακολουθεί ο πράκτορας προσπαθεί να μεγιστοποιήσει μια αθροιστική συνάρτηση των μελλοντικών σημάτων ενίσχυσης R_t. Συναρτήσεις που απαντώνται στη βιβλιογραφία είναι οι παρακάτω:

$$R_t = \sum_{i=0}^{T} r_{t+i+1} \tag{4.2.1}$$

Στην περίπτωση αυτή ο πράκτορας ενδιαφέρεται μόνο για τις μελλοντικές ενισχύσεις μέχρι ένα χρονικό ορίζοντα. Η αθροιστική αυτή ενίσχυση ταιριάζει πολύ σε επεισοδικά προβλήματα (βλέπε ενότητα 4.3)

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \mid \gamma \in [0,1] \tag{4.2.2}$$

Στην περίπτωση αυτή ο πράκτορας ενδιαφέρεται για όλο το μελλοντικό άθροισμα των ενισχύσεων, βεβαρημένο όμως με έναν παράγοντα $\gamma < 1$ (ρυθμός έκπτωσης). Μια ενίσχυση που λαμβάνεται k βήματα στο μέλλον ενδιαφέρει τον πράκτορα μόνο κατά γ^k. Το μοντέλο αυτό είναι και το πιο εύκολο προς ανάλυση και το πιο μελετημένο στη βιβλιογραφία. Όταν $\gamma = 0$ ο πράκτορας γίνεται «μυωπικός», δηλαδή τον ενδιαφέρει μόνο η ενίσχυσή του στο τρέχον βήμα, ενώ για $\gamma \to 1$ ο πράκτορας βλέπει πιο βαθειά στο μέλλον.

$$R_t = \lim_{h \to \infty} \left(\frac{1}{h} \sum_{i=0}^{h} r_{t+i+1} \right) \tag{4.2.3}$$

Στην περίπτωση αυτή ο πράκτορας ενδιαφέρεται για δράσεις που μεγιστοποιούν τη μέση ενίσχυση σε κάθε βήμα και δεν τον ενδιαφέρει η χρονική κατανομή της ενίσχυσης.

Η πολιτική που ακολουθείται συμβολίζεται με

$\pi : S \times A \to [0,1]$ όπου $\pi(s,a)$ είναι η πιθανότητα σύμφωνα με την πολιτική π, η δράση στην κατάσταση s να είναι η α

Το σύστημα έχει την ιδιότητα Markov ανν

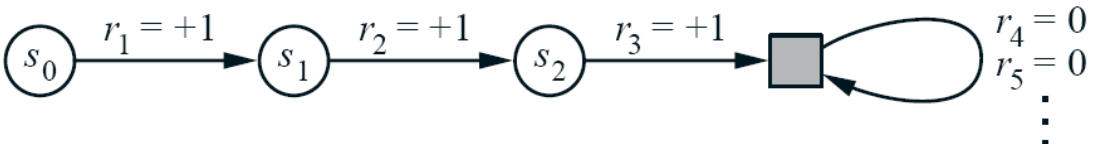
$$\begin{aligned}&\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, \ldots, s_0, a_0\}\\ &\qquad = \Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\}\end{aligned} \tag{4.2.4}$$

δηλαδή αν η επόμενη κατάσταση και το επόμενο σήμα ενίσχυσης εξαρτώνται μόνο από την παρούσα κατάσταση και τη δράση που επιλέχθηκε. Ουσιαστικά, η παρούσα κατάσταση πρέπει να συμπυκνώνει όλη την πληροφορία που χρειάζεται ο πράκτορας για την ιστορία καταστάσεων – δράσεων μέχρι τώρα, οπότε οι αποφάσεις να μπορούν να παρθούν με βάση μόνο την παρούσα κατάσταση. Αν ισχύει ή προσεγγίζεται ικανοποιητικά η παραπάνω ιδιότητα μπορούμε να κατασκευάσουμε μοντέλο Markov του προβλήματος (finite Markov Decision Process). Αυτό αποτελείται από τις πιθανότητες μετάβασης (transition probabilities) και τις προσδωκόμενες τιμές των ενισχύσεων.

$$\begin{aligned}P^a_{s,s'} &= \Pr\{s_{t+1}=s' \mid s_t=s, a_t=a\}\\ R^a_{s,s'} &= E\{r_{t+1} \mid s_t=s, a_t=a, s_{t+1}=s'\}\end{aligned} \tag{4.2.5}$$

4.3 Επεισοδικά και συνεχή προβλήματα

Υπάρχουν προβλήματα στα οποία η ακολουθία της επικοινωνίας πράκτορα – περιβάλλοντος μπορεί λογικά να ομαδοποιηθεί σε μια σειρά από ανεξάρτητα επεισόδια (*επεισοδικά προβλήματα* – episodic tasks) και προβλήματα στα οποία η αλληλουχία της αλληλεπίδρασης πράκτορα – περιβάλλοντος είναι συνεχής (*συνεχή προβλήματα* – continuing tasks). Ένα επεισοδικό πρόβλημα τελειώνει σε μια ειδική τερματική κατάσταση και επιστρέφει σε μια μοναδική αρχική κατάσταση ή σε μια τυχαία κατάσταση από ένα σετ αρχικών καταστάσεων. Στα πρώτα η συνάρτηση προσδωκόμενης ενίσχυσης 4.2.1 ταιριάζει καλύτερα, ενώ για συνεχή προβλήματα πιο φυσική φαίνεται η 4.2.2. Εισάγοντας στα επεισοδικά προβλήματα μια απορροφητική κατάσταση όπως παρακάτω μπορούμε να αναπτύξουμε ενιαίο φορμαλισμό και για τις δύο περιπτώσεις χρησιμοποιώντας την 4.2.2 με $\gamma = 1$ ή $T = \infty$



Σχήμα 4.2 (πηγή [SB98])

4.4 Συναρτήσεις τιμών

Ένα ακόμα προσδιοριστικό χαρακτηριστικό του πεδίου της Ενισχυτικής Μάθησης, είναι ότι όλες οι μέθοδοι χρησιμοποιούν συναρτήσεις τιμών για τη λύση του. Ο πράκτορας κάνει ένα

είδος book – keeping κρατώντας εκτιμήσεις των μελλοντικών ενισχύσεων που λαμβάνει και μέσω αυτών λαμβάνει τις αποφάσεις του. Ο στόχος είναι ο πράκτορας να αποθηκεύει τη γνώση του για το πόσο καλή είναι μια κατάσταση ή πόσο καλό είναι να κάνει μια δράση όταν βρίσκεται σε μια κατάσταση. Ο στόχος είναι οι τιμές αυτές να προσεγγίσουν τις συναρτήσεις που ορίζονται παρακάτω.

Ορίζουμε ως $V^{\pi}(s)$ την τιμή της κατάστασης s όταν ακολουθείται η πολιτική π ως την προσδωκόμενη ενίσχυση που θα λάβει ο πράκτορας αν αρχίσει από την s και ακολουθήσει την π (state – value function). Τυπικά:

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} \tag{4.4.1}$$

και παρόμοια ορίζουμε ως $Q^{\pi}(s,a)$ την τιμή του ζεύγος (s,α), δηλαδή την προσδοκώμενη ενίσχυση που θα λάβει ο πράκτορας αν στην κατάσταση s κάνει τη δράση και μετά ακολουθήσει την πολιτική π (action – value function).

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} \tag{4.4.2}$$

Δεδομένου ενός μοντέλου Markov, είναι δυνατόν να διατυπωθούν αναδρομικές εξισώσεις για τις συναρτήσεις τιμών, γνωστές ως εξισώσεις Bellman.

$$V^{\pi}(s) = \sum_{a}\pi(s,a)\sum_{s'}P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma V^{\pi}(s')\right] \tag{4.4.3}$$

$$Q^{\pi}\left(s,a\right) = \sum_{s'}P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma\sum_{s'}\pi\left(s',a'\right)Q^{\pi}\left(s',a'\right)\right] \tag{4.4.4}$$

Η λύση του παραπάνω συστήματος εξισώσεων ισοδυναμεί με την εύρεση της μελλοντικής ενίσχυσης για κάθε κατάσταση δεδομένης μιας πολιτικής (αξιολόγηση πολιτικής).

4.5 Η έννοια της βέλτιστης πολιτικής

Όπως έχει ήδη αναφερθεί, ο πράκτορας ενδιαφέρεται να βρει την πολιτική που του δίνει τη μέγιστη μελλοντική ενίσχυση. Τυπικά, αυτό περιγράφεται όπως παρακάτω

Ορίζουμε μια σχέση μερικής διάταξης επί των πολιτικών $\leq$:

$$\pi \leq \pi' \Leftrightarrow V^{\pi}(s) \leq V^{\pi'}(s), \forall s \tag{4.5.1}$$

Ονομάζουμε βέλτιστη πολιτική μια πολιτική π^* που ικανοποιεί την παρακάτω σχέση

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \forall s \tag{4.5.2}$$

Για τις τιμές της βέλτιστης πολιτικής ισχύουν οι εξισώσεις βελτίστου Bellmann

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} \left\{ \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right] \right\}
\end{aligned}
\tag{4.5.3}
$$

$$
\begin{aligned}
Q^*(s,a) &= E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\} \\
&= E\left\{ r_{t+1} + \gamma \max_{a' \in A(s_{t+1})} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a \right\} \\
&= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a' \in A(s')} Q^*(s', a') \right]
\end{aligned}
\tag{4.5.4}
$$

Η αξιολόγηση της $V^*(s)$ (η λύση δηλαδή του συστήματος $|S| \times |S|$ εξισώσεων 4.5.3) σε συνδυασμό με μια έρευνα ενός βήματος (one step search) από την $\pi^*(s) = \arg\max_{a \in A(s)} \left(\sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right] \right)$ ισοδυναμεί με τη λύση του προβλήματος Ενισχυτικής Μάθησης. Η λύση από την άλλη του συστήματος $|S||A| \times |S||A|$ εξισώσεων 4.5.4 ισοδυναμεί πλήρως με τη λύση του προβλήματος Ενισχυτικής Μάθησης αφού $\pi^*(s) = \arg\max_{a \in A(s)} \left(Q(s,a) \right)$. Βέβαια, η λύση των παραπάνω εξισώσεων και η εύρεση μιας βέλτιστης πολιτικής δεν είναι πάντα δυνατή και ίσως σε κάποιες περιπτώσεις δεν είναι και επιθυμητή. Μια πλήρης γνώση του μοντέλου Markov, η ύπαρξη ιδιότητας Markov και το υπολογιστικό κόστος λύσης των παραπάνω συστημάτων μειώνουν κατά πολύ το χώρο των προβλημάτων που μπορούν να λυθούν με μια τέτοια μέθοδο. Επίσης, ενώ τα παραπάνω αφορούν ντετερμινιστικές πολιτικές, τα ενδιαφέροντα και δύσκολα προβλήματα συνήθως απαιτούν ένα βαθμό εξερεύνησης αφού η δυναμική τους αλλάζει με το χρόνο, άρα και η βέλτιστη πολιτική σταματά να είναι βέλτιστη.

4.6 Η ισορροπία μεταξύ Εκμετάλλευσης και Εξερεύνησης

Ίσως το πιο απλό πρόβλημα που εντάσσεται στο πεδίο της ενισχυτικής μάθησης είναι το πρόβλημα k – armed bandit. Ένας παίκτης βρίσκεται μπροστά από k κουλοχέρηδες, οι οποίοι έχουν διαφορετικούς μέσους όρους επιστρεφόμενων χρημάτων, και έχει h ευκαιρίες να παίξει. Δεν χρειάζεται να βάλει νόμισμα αλλά η ποινή που πληρώνει είναι ότι ίσως έπαιξε σε μια μη βέλτιστη μηχανή. Όταν ο παίκτης παίξει σε μια μηχανή, αυτή επιστρέφει 0 ή r_i σύμφωνα με μια άγνωστη πιθανότητα. Τι πρέπει να παίξει ο παίκτης; Το πρόβλημα αυτό δείχνει ξεκάθαρα το tradeoff ανάμεσα στην εκμετάλλευση της γνώσης και την εξερεύνηση για απόκτηση νέας γνώσης. Το πρόβλημα έχει μελετηθεί πολύ βαθιά στη Στατιστική και έχει λυθεί με διάφορους τρόπους [BF86].

Υποθέτουμε ότι ο πράκτορας αποθηκεύει εκτιμήσεις των συναρτήσεων τιμών, και ιδιαίτερα της Q. Συμβολίζουμε ως $Q_t(s,a)$ την εκτίμηση του πράκτορα τη χρονική στιγμή t για την τιμή ενός ζεύγους κατάστασης και δράσης. Ίσως η επιλογή της $a = \arg\max_{b \in A(s)} \left(Q_t(s,b) \right)$ να μην είναι πάντα η καλύτερη. Η επιλογή της δράσης με βάση τις αποθηκευμένες τιμές μπορεί να γίνει με πολλούς τρόπους, οι οποίοι καθρεφτίζουν λύσεις στο πρόβλημα της ισορροπίας εκμετάλλευσης – εξερεύνησης.

- Η πιο απλή λύση είναι φυσικά να διαλέγουμε την δράση που έχει κάθε φορά τη μέγιστη τιμή. Η προσέγγιση αυτή δεν εξερευνά καθόλου και θα την ονομάσουμε greedy, δηλαδή

$$a_t = greedy\left(Q_t(s,a) \right) = \arg\max_{b \in A} \left(Q_t(s,b) \right) \tag{4.6.1}$$

- Η λύση που μόνο εξερευνά και δεν εκμεταλλεύεται καθόλου την αποκτηθείσα πληροφορία αντιστοιχεί σε μια τυχαία κίνηση στο χώρο των δράσεων, και επιλέγει κάθε δράση με ίδια πιθανότητα, δηλαδή

$$a_t = \underset{a \in A(s)}{random}\left(Q_t(s,a) \right) \Leftrightarrow \Pr\{a_t = a\} = \frac{1}{|A(s)|} \tag{4.6.2}$$

- Άλλη προφανής λύση είναι να δίνουμε σε κάθε δράση πιθανότητα ίση με τη σχετική μέση ενίσχυσή της, δηλαδή

$$a_t = \underset{a \in A(s)}{uniform}\left(Q_t(s,a) \right) \Leftrightarrow \Pr\{a_t = a\} = \frac{Q_t(s,a)}{\sum_{b \in A(s)} Q_t(s,b)} \tag{4.6.3}$$

- Η προσέγγιση ε-greedy διαλέγει τη βέλτιστη κίνηση με πιθανότητα 1-ε και κάποια άλλη με πιθανότητα ε

$$a_t = \underset{a \in A(s)}{\varepsilon - greedy}\left(Q_t(s,a) \right) \Leftrightarrow \Pr\{a_t = a\} = \begin{cases} 1-\varepsilon, a = \arg\max_{b \in A(s)} \left(Q_t(s,b) \right) \\ \varepsilon, a \neq \arg\max_{b \in A(s)} \left(Q_t(s,b) \right) \end{cases} \tag{4.6.4}$$

- Μπορούμε να χρησιμοποιήσουμε κάποια κατανομή πιθανοτήτων εκτός της ομοιόμορφης για να μην δίνουμε ίσες πιθανότητες σε δράσεις με πολύ διαφορετικές τιμές. Η κατανομή Boltzmann/Gibbs δίνει

$$a_t = \underset{a \in A(s)}{boltzmann}\left(Q_t(s,a) \right) \Leftrightarrow \Pr\{a_t = a\} = \frac{e^{p_t(s,a)}}{\sum_{b \in A(s)} e^{p_t(s,b)}} \tag{4.6.5}$$

όπου $p_t(s,a)$ η «προτίμηση» για τη δράση a, η οποία βασίζεται με κάποιο τρόπο στην αποθηκευμένη τιμή.

- Μια παρόμοια ιδέα είναι η Προσομοιωμένη Ανόπτηση (Simulated Annealing)

$$a_t = \underset{a \in A(s)}{annealing}\left(Q_t(s,a)\right) \Leftrightarrow \Pr\{a_t = a\} = \frac{e^{Q_t(s,a)/T}}{\sum_{b \in A(s)} e^{Q_t(s,b)/T}} \tag{4.6.6}$$

όπου T είναι μια θετική παράμετρος που ονομάζεται θερμοκρασία. Μεγάλες θερμοκρασίες ευνοούν τη εξερεύνηση ενώ μικρές εκμεταλλεύονται τη γνώση. Αν ο πράκτορας ξεκινά από με μια μεγάλη θερμοκρασία και να την ψύχει καθώς αποκτά περισσότερη γνώση, εξερευνά στην αρχή περισσότερο το χώρο των δράσεων, ενώ όταν η γνώση του για το πρόβλημα είναι «καλή», την εκμεταλλεύεται. Η διαδικασία ψύξης προφανώς χρειάζεται ιδιαίτερη προσοχή ώστε να ανταποκρίνεται στη φύση του εκάστοτε προβλήματος.

Καλούμε τέλος μια πολιτική soft αν $\pi(s,a) > 0, \forall s$ και ε-soft αν $\pi(s,a) > \varepsilon / |A(s)|, \forall s$.

4.7 Generalized Policy Iteration

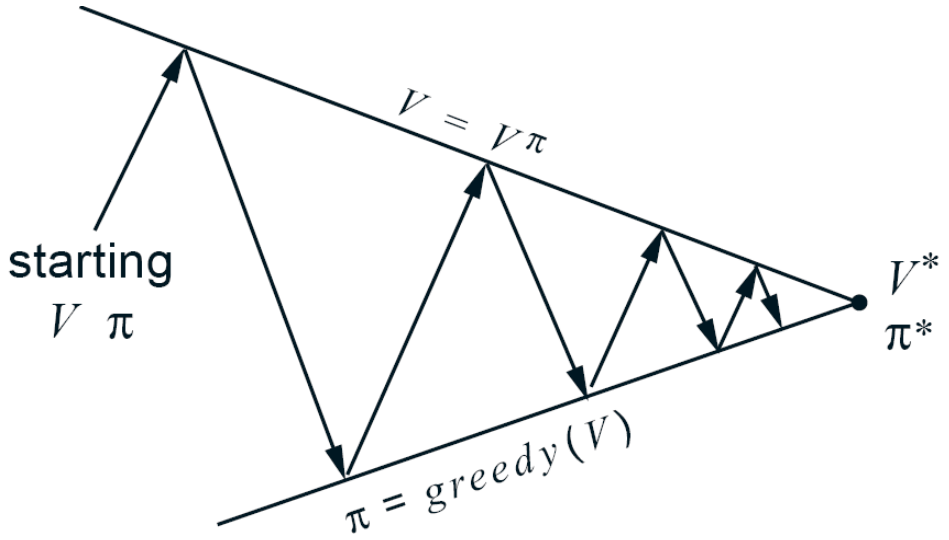
Η ιδέα της χρησιμοποίησης συναρτήσεων τιμών για τη λύση του προβλήματος Ενισχυτικής Μάθησης μπορεί να δώσει ένα γενικό (generic) σχήμα το οποίο σχεδόν κάθε μέθοδος λύσης ικανοποιεί (Generalized Policy Iteration). Ο πράκτορας ενώ δρα ακολουθώντας μια πολιτική π αυξάνει τη γνώση του Q ή V (φάση evaluation). Παράλληλα όμως, χρησιμοποιεί τη γνώση αυτή που αποκτά για να βελτιώσει την πολιτική που ακολουθεί (φάση improvement).

- Ονομάζουμε evaluation μιας πολιτικής μια επαναληπτική διαδικασία (αλγόριθμο), που στόχο έχει να μετακινήσει τις αποθηκευμένες τιμές V_t, Q_t προς τις πραγματικές τιμές V^π, Q^π. Η διαδικασία αυτή αντικατοπτρίζει τη φύση του αλγόριθμου μάθησης και μπορεί να γίνεται χρησιμοποιώντας το μοντέλο Markov, την πραγματική εμπειρία του πράκτορα ή με προσομοίωση δράσεων (simulated experience). Η συνάρτηση τιμών που χρησιμοποιείται μπορεί επίσης να διαφέρει ανάλογα με τον αλγόριθμο.
- Ονομάζουμε improvement μιας πολιτικής μια διαδικασία που στόχο έχει την μετακίνηση της πολιτικής που ακολουθεί ο πράκτορας προς την ήδη αποθηκευμένη γνώση του V_t, Q_t. Η φάση αυτή αντικατοπτρίζει την προσέγγιση της λύσης στο exploration – exploitation tradeoff, δηλαδή η πολιτική μπορεί να συμμορφώνεται πλήρως με τη συνάρτηση τιμών ή να αφήνει με διάφορους τρόπους που έχουν αναφερθεί χώρο για αναζήτηση. Το θεώρημα βελτίωσης πολιτικής (policy improvement theorem) παρέχει μια θεωρητική εγγύηση για την παραπάνω διαδικασία

$$Q^\pi\left(s, \pi'(s)\right) \geq V^\pi(s) \Rightarrow \pi'(s) \geq \pi(s) \tag{4.7.1}$$

όπου $Q^{\pi}\left(s,\pi'(s)\right)=\sum_{a}\pi(s,a)Q^{\pi}(s,a)$.

Ονομάζουμε τέλος έλεγχο (control) τον συνδυασμό των δύο φάσεων, δηλαδή την επίλυση του προβλήματος ενισχυτικής μάθησης. Ο τρόπος που γίνονται οι δύο αυτές φάσεις αλλά και ο βαθμός επικάλυψής τους μπορούν φυσικά να διαφέρουν ανάμεσα στους αλγόριθμους μάθησης. Η διαδικασία της αλληλεπίδρασης τελειώνει μόλος φτάσει σε ένα βαθμό ευστάθειας (κριτήριο τερματισμού), δηλαδή η πολιτική συμμορφώνεται με τη γνώση και το αντίθετο. Το σχήμα αυτό απεικονίζεται παρακάτω για την απλοποιημένη περίπτωση που χρησιμοποιείται η συνάρτηση τιμών V η οποία συμμορφώνεται πλήρως με την πολιτική (λύση των εξισώσεων Bellman) και δεν υπάρχει καθόλου εξερεύνηση.



Σχήμα 4.3 (πηγή [SB98])

4.8 Λύση με Δυναμικό Προγραμματισμό

Ο Δυναμικός Προγραμματισμός (Dynamic Programming) αναφέρεται σε μια κλάση αλγορίθμων που υπολογίζουν τη βέλτιστη πολιτική δεδομένου ενός μοντέλου Markov για το πρόβλημα. Ουσιαστικά είναι μια επαναληπτική διαδικασία επίλυσης των εξισώσεων Bellman. Η ιδέα είναι να μετατραπούν οι παραπάνω εξισώσεις σε επαναληπτικές αναθέσεις και οι ενημερώσεις των τιμών να γίνονται με βάση τις τρέχουσες εκτιμήσεις των άλλων τιμών (bootstrapping). Η εφαρμοσιμότητα των μεθόδων αυτών είναι σχετικά περιορισμένη λόγω της ανάγκης γνώσης του μοντέλου Markov που προϋποθέτουν και του μεγάλου υπολογιστικού κόστους των (ο δυναμικός προγραμματισμός υποφέρει από τη λεγόμενη curse of dimensionality). Παρ' όλ' αυτά η απόδειξη σύγκλισης στο βέλτιστο κάνει κάποιες παραλλαγές ελκυστικές σε κάποιες περιπτώσεις, ενώ η θεωρητική σημασία τους είναι μεγάλη. Πάνω στο Δυναμικό Προγραμματισμό είναι που «χτίζουν» οι πιο προχωρημένες τεχνικές Ενισχυτικής Μάθησης.

4.8.1 Αξιολόγηση πολιτικής με Δυναμικό Προγραμματισμό

Η αξιολόγηση (evaluation) μιας πολιτικής γίνεται με την παρακάτω επαναληπτική σχέση, η οποία είναι απλά η μετατροπή της εξίσωσης Bellman σε επαναληπτική ανάθεση. Ο αλγόριθμος σταματά αν οι τιμές δεν αλλάζουν πολύ ανάμεσα στις επαναλήψεις ή αν έχουν φτάσει σε ένα επιθυμητό επίπεδο ακρίβειας.

$$\begin{aligned} V_{k+1}(s) &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right], \forall s \\ V_0(s) &= 0, \forall s \end{aligned} \tag{4.8.1}$$

Η ανανέωση μιας τιμής γίνεται με βάση την υπάρχουσα γνώση για όλες τις άλλες τιμές (full backup) και όχι μόνο από την τιμή της επόμενης κατάστασης. Η παραπάνω ακολουθία αποδεδειγμένα συγκλίνει στη V^π. Ψευδοκώδικας για την προσέγγιση αυτή φαίνεται παρακάτω

Αλγόριθμος 4.1 : Dynamic Programming Policy Evaluation

είσοδος: π
έξοδος: V
1. Αρχικοποίηση
   $V(s) = 0, \forall s \in S$
2. Αξιολόγηση πολιτικής
   repeat
      $\Delta \leftarrow 0$
      for each $s \in S$
         $u \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s'} P^{\pi(s)}_{ss'} \left[R^{\pi(s)}_{ss'} + \gamma V(s') \right]$
         $\Delta \leftarrow \max\left(\Delta, |u - V(s)|\right)$
   until $\Delta < \theta$

4.8.2 Έλεγχος με Δυναμικό Προγραμματισμό

Υπάρχουν δύο προσεγγίσεις για την εύρεση της βέλτιστης πολιτικής με Δυναμικό Προγραμματισμό. Η πρώτη, βασίζεται στο σχήμα GPI. Μια αρχική πολιτική επιλέγεται και αφού αυτή αξιολογηθεί χρησιμοποιώντας δυναμικό προγραμματισμό, βελτιώνεται με κάποια μέθοδο της ενότητας 4.6 (συνήθως greedy). Αυτό συμβαίνει επαναληπτικά, μέχρι η αλληλεπίδραση των δύο φάσεων να γίνει ευσταθής σύμφωνα με το παρακάτω σχήμα.

$$\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*$$

Ψευδοκώδικας για την προσέγγιση αυτή φαίνεται παρακάτω.

Αλγόριθμος 4.2 : Dynamic Programming Policy Iteration

έξοδος: π

1. Αρχικοποίηση

$V(s) \in \mathbb{R}, \pi(s) \in A(s), \forall s \in S$

2. Αξιολόγηση πολιτικής

repeat

$\Delta \leftarrow 0$

for each $s \in S$

$u \leftarrow V(s)$

$V(s) \leftarrow \sum_{s'} P_{ss'}^{\pi(s)} \left[R_{ss'}^{\pi(s)} + \gamma V(s') \right]$

$\Delta \leftarrow \max\left(\Delta, \left|u - V(s)\right|\right)$

until $\Delta < \theta$

3. Βελτίωση πολιτικής

stable $\leftarrow$ *false*

for each $s \in S$

$b \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a \left(\sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V(s') \right] \right)$

if $(stable)$ then stop else goto 2

Η δεύτερη προσέγγιση προσπαθεί να εξαλείψει το φαινόμενο ότι μια επανάληψη του αλγόριθμου 4.2 περιλαμβάνει μια πλήρη αξιολόγηση πολιτικής, η οποία μπορεί να είναι μια πολύ χρονοβόρα διαδικασία. Αν και εξακολουθεί να υπακούει στο σχήμα GPI, η μέθοδος value iteration κάνει μόνο μια ανανέωση τιμών σε κάθε επανάληψή της. Ουσιαστικά η μέθοδος αυτή μετατρέπει την εξίσωση βέλτιστου Bellman σε επαναληπτική ανάθεση.

Αλγόριθμος 4.3 : Dynamic Programming Value Iteration

έξοδος: π

1. Αρχικοποίηση

$V(s) \in \mathbb{R}, \forall s \in S$

2. Value iteration

repeat

$\Delta \leftarrow 0$

for each $s \in S$

$u \leftarrow V(s)$

$V(s) \leftarrow \max_s \left(\sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s') \right] \right)$

$\Delta \leftarrow \max\left(\Delta, \left|u - V(s)\right|\right)$

until $\Delta < \theta$

3. return $\arg\max_a \left(\sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s') \right] \right)$

4.8.3 Ασύγχρονος Δυναμικός Προγραμματισμός

Οι βρόχοι που διαπερνούν όλο το χώρο καταστάσεων στις προηγούμενες περιπτώσεις μπορούν να αποφευχθούν. Αλγόριθμοι στην κλάση του Ασύγχρονου Δυναμικού Προγραμματισμού [Ber82] αποθηκεύουν ότι τιμές καταστάσεων έχουν διαθέσιμες και τις ανανεώνουν με ότι τιμές έχουν διαθέσιμες. Έτσι, η εύρεση βέλτιστης πολιτικής μπορεί να γίνει παράλληλα με τη διαδικασία της δράσης του πράκτορα, ο οποίος χρησιμοποιεί κάθε φορά την τελευταία έκδοση της πολιτικής που του υπέδειξε ο Δυναμικός Προγραμματισμός.

4.9 Μάθηση Monte Carlo

Οι μέθοδοι Monte Carlo αντίθετα με τους αλγορίθμους Δυναμικού Προγραμματισμού δεν υποθέτουν γνώση του περιβάλλοντος Markov. Αντίθετα, προσπαθούν να εκτιμήσουν τις τιμές των $V^\pi(s)$ ή $Q^\pi(s,a)$ κρατώντας μέσους όρους των ενισχύσεων από την εμπειρία του πράκτορα. Η εμπειρία αυτή μπορεί να είναι on – line (η πραγματική εμπειρία του πράκτορα όταν ακολουθεί μια πολιτική) ή simulated (ο πράκτορας προσομοιώνει μεταβάσεις σύμφωνα με μια πολιτική η οποία δεν είναι κατ' ανάγκη αυτή που ακολουθεί). Οι μέθοδοι Monte Carlo εφαρμόζονται σε επεισοδικά προβλήματα και ανανεώνουν τις τιμές τους οπότε βελτιώνουν και την πολιτική μετά το τέλος κάθε επεισοδίου. Η ανανέωση των τιμών δεν χρησιμοποιεί

καθόλου τις τρέχουσες εκτιμήσεις του πράκτορα για τις τιμές των άλλων καταστάσεων. Υπό αυτήν την έννοια (bootstrapping) οι μέθοδοι Monte Carlo βρίσκονται στο άλλο άκρο από το δυναμικό προγραμματισμό. Οι first – visit Monte Carlo μέθοδοι χρησιμοποιούν μόνο την ενίσχυση που ακολουθεί την πρώτη επίσκεψη σε δεδομένη κατάσταση για κάθε επεισόδιο και έχουν μελετηθεί διεξοδικότερα από τις every – visit Monte Carlo μεθόδους που χρησιμοποιούν κάθε επίσκεψη στο επεισόδιο.

4.9.1 Αξιολόγηση πολιτικής Monte Carlo

Η αξιολόγηση πολιτικής Monte Carlo είναι απλή. Ο πράκτορας κρατά για κάθε κατάσταση μια λίστα με τις ενισχύσεις που λαμβάνει για κάθε κατάσταση. Η εκτίμησή του για την τιμή μιας κατάστασης είναι ο μέσος όρος των ενισχύσεων αυτών και ανανεώνεται μετά το τέλος κάθε επεισοδίου.

Αλγόριθμος 4.4 : Monte Carlo Policy Evaluation

είσοδος: π
έξοδος: V
1. Αρχικοποίηση
   $V(s), \pi(s), returns(s) \leftarrow [], \forall s \in S$
2. Αξιολόγηση πολιτικής
   repeat
      εκτέλεση επεισοδίου E χρησιμοποιώντας την π
      for each $s \in$ E
         $R \leftarrow$ ενίσχυση που ακολούθησε την πρώτη εμφάνιση της s
         $returns(s) \leftarrow [R \mid returns(s)]$
         $V(s) \leftarrow average(returns(s))$
   forever

4.9.2 Έλεγχος Monte Carlo

Ο έλεγχος Monte Carlo ακολουθεί το σχήμα GPI, εκτιμώντας όμως τις τιμές Q αφού χωρίς γνώση του μοντέλου Markov, οι τιμές V δεν αρκούν για τη βελτίωση της πολιτικής. Η εκτίμηση των τιμών γίνεται με τη διαδικασία αξιολόγησης πολιτικής και ανανεώνονται μετά από κάθε επεισόδιο. Τότε γίνεται και η βελτίωση της πολιτικής με κάποιο τρόπο από αυτούς που αναφέρονται στην ενότητα 4.6. Παρακάτω φαίνεται ένας βασικός αλγόριθμος σε ψευδοκώδικα για έλεγχο Monte Carlo, αλλά πρέπει να αναφερθεί ότι αυτός έρχεται σε πολλές παραλλαγές. Για περισσότερες πληροφορίες παραπέμπουμε στο [SB98]

Αλγόριθμος 4.4 : Monte Carlo on - policy Control

είσοδος: π

έξοδος: Q

1. Αρχικοποίηση

$Q(s,a), \pi(s,a), returns(s,a) \leftarrow [], \forall s,a \in S \times A$

2. Έλεγχος

repeat

εκτέλεση επεισοδίου Ε χρησιμοποιώντας την π

for each $s \in$ E

$R \leftarrow$ ενίσχυση που ακολούθησε την πρώτη εμφάνιση της s,a

$returns(s,a) \leftarrow \left[R \mid returns(s,a) \right]$

$Q(s,a) \leftarrow average\left(returns(s,a)\right)$

$\pi(s,a) \leftarrow somehow\left(Q(s,a)\right)$

forever

4.10 Μάθηση Temporal Difference

Η νέα ιδέα που παρουσιάζει το πεδίο της Ενισχυτικής Μάθησης είναι αυτή της Temporal Difference Learning, η οποία μπορεί να νοηθεί ως μια μίξη του Δυναμικού Προγραμματισμού και των μεθόδων Monte Carlo. Όπως και οι μέθοδοι Monte Carlo, οι μέθοδοι TD ενώ δεν υποθέτουν γνώση ενός περιβάλλοντος Markov για το πρόβλημα ανανεώνουν την εκτίμηση του πράκτορα για τις συναρτήσεις τιμών χρησιμοποιώντας τόσο την εμπειρία του πράκτορα (όπως οι μέθοδοι Monte Carlo), όσο και τις τρέχουσες εκτιμήσεις τους για τη συνάρτηση τιμών (όπως οι μέθοδοι Δυναμικού Προγραμματισμού). Επίσης, ενώ οι μέθοδοι Monte Carlo περιμένουν μέχρι το τέλος ενός επεισοδίου για να ανανεώσουν τις εκτιμήσεις τους, οι μέθοδοι TD κάνουν το ίδιο μετά από κάθε δράση του πράκτορα (on – line).

4.10.1 Αξιολόγηση πολιτικής TD(0)

Ενώ οι μέθοδοι Monte Carlo προσπαθούν να εκτιμήσουν κατ' ευθείαν το R_t κρατώντας το μέσο όρο των ενισχύσεων που λαμβάνει ο πράκτορας και μετακινώντας τη συνάρτηση τιμών προς αυτήν την κατεύθυνση με μια ανανέωση του τύπου $V(s_t) \leftarrow V(s_t) + a\left[R_t - V(s_t)\right]$, η ιδέα πίσω από τις TD μεθόδους είναι η συνάρτηση τιμών να μετακινείται προς τον ίδιο στόχο χρησιμοποιώντας μόνο την άμεση ενίσχυση και τις ήδη αποθηκευμένες τιμές ως

$V(s_t) \leftarrow V(s_t) + a\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$. Η πιο απλή μέθοδος της οικογένειας αυτής, TD(0) αξιολογεί την πολιτική όπως φαίνεται παρακάτω.

Αλγόριθμος 4.5 : TD(0) Policy Evaluation

είσοδος: π
έξοδος: V
1. Αρχικοποίηση
   $V(s), \forall s \in S$
2. Αξιολόγηση πολιτικής
   repeat for each episode
      repeat for each episode step
         $a \leftarrow$ δράση σύμφωνα με την π
         κάνε την a; δες τα s', r
         $V(s) \leftarrow V(s) + a\left[r + \gamma V(s') - V(s)\right]$
         $s \leftarrow s'$
      until (s τελική)

4.10.2 Έλεγχος TD

Οι μέθοδοι ελέγχου TD υπακούουν στο σχήμα GPI. Υπάρχει και εδώ το πρόβλημα της συνέχισης της εξερεύνησης όπως κα στις μεθόδους Monte Carlo, οπότε γενικά χρησιμοποιούνται soft πολιτικές και η αξιολόγηση γίνεται για την συνάρτηση Q για τους ίδιους λόγους που αυτό συμβαίνει στις μεθόδους Monte Carlo. Ο παρακάτω αλγόριθμος που καλείται Sarsa είναι μια on – policy TD(0) μέθοδος που ουσιαστικά αξιολογεί την Q και κατευθύνει την πολιτική προς την τελευταία αξιολόγηση.

Αλγόριθμος 4.6 : Sarsa

έξοδος: π

1. Αρχικοποίηση

$Q(s,a), \forall s \in S$

2. Έλεγχος

repeat for each episode

$\alpha \leftarrow somehow\left(Q(s,a)\right)$

repeat for each episode step

κάνε την a; δες τα s', r

$\alpha' \leftarrow somehow\left(Q(s',a)\right)$

$Q(s,a) \leftarrow Q(s,a) + a\left[r + \gamma Q(s',a') - Q(s,a)\right]$

$s \leftarrow s'; a \leftarrow a'$

until $(s$ τελική$)$

Η μέθοδος Q – learning από την άλλη προσεγγίζει κατ' ευθείαν το Q^* ανεξάρτητα από την πολιτική που ακολουθεί ο πράκτορας (off – policy μέθοδος). Η πρόταση της μεθόδου αυτής από τον Watkins αυτή ήταν ένα από τα σημαντικότερα σημεία της ιστορίας της ενισχυτικής μάθησης.

Αλγόριθμος 4.7 : Q-learning

έξοδος: π

1. Αρχικοποίηση

$Q(s,a), \forall s \in S$

2. Έλεγχος

repeat for each episode

repeat for each episode step

$\alpha \leftarrow somehow\left(Q(s,a)\right)$

κάνε την a; δες τα s', r

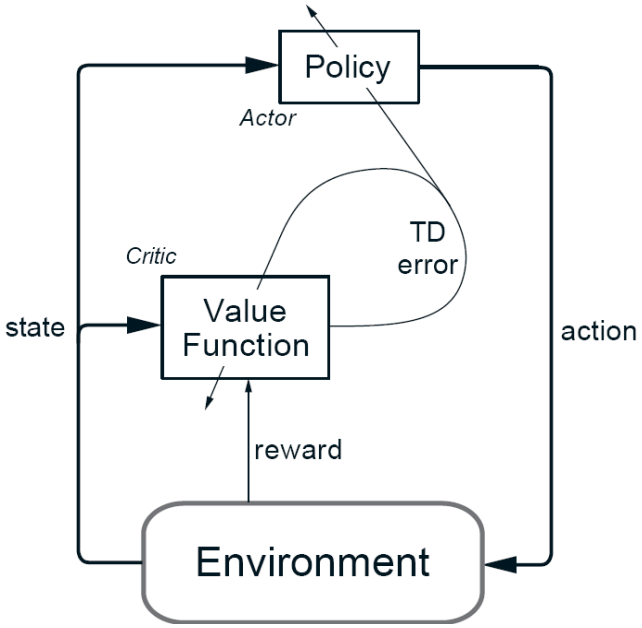
$Q(s,a) \leftarrow Q(s,a) + a\left[r + \gamma \max_{a'}\left(Q(s',a')\right) - Q(s,a)\right]$

$s \leftarrow s'$

until $(s$ τελική$)$

Μια άλλη προσέγγιση στον έλεγχο TD είναι οι μέθοδοι Actor – Critic, οι οποίες χρησιμοποιούν τη συνάρτηση V. Κρατάνε την πολιτική του πράκτορα ως μια ξεχωριστή δομή δεδομένων (actor, αφού διαλέγει τη δράση που θα γίνει) και ονομάζουν τις υπολογισμένες τιμές critic, αφού κριτικάρουν τις δράσεις που διαλέγει ο actor. Η κριτική έχει

τη μορφή ενός TD σφάλματος. Η αρχιτεκτονική ενός τέτοιου συστήματος φαίνεται παρακάτω.



Σχήμα 4.4 (πηγή [SB98])

Ο critic κρατά τις «προτιμήσεις» για τις δράσεις $p(s,a)$ και τις ανανεώνει σύμφωνα με τις τιμές V σχηματίζοντας το TD – σφάλμα δ. Ο actor διαλέγει και πραγματοποιεί δράσεις σύμφωνα με τις προτιμήσεις $p(s,a)$. Η λειτουργία του συστήματος μπορεί να περιγραφεί σαν ένας «διάλογος» ανάμεσα στα δύο υποσυστήματα actor και critic.

Αλγόριθμος 4.8 : Actor - Critic Control

A : βλέπω την s

επιλογή $\alpha \leftarrow somehow\left(p(s,a)\right)$

C : βλέπω τα s', r

υπολογισμός σφάλματος $\delta \leftarrow \rho + \gamma V(s') - V(s)$

ανανέωση προτιμήσεων $p(s,a) \leftarrow update\left(p(s,a), \delta\right)$

Η επιλογή των δράσεων γίνεται τυπικά με μια soft μέθοδο (πχ Boltzmann/Gibbs) της ενότητας 4.6, ενώ η ανανέωση των προτιμήσεων με μια διαδικασία του τύπου $p(s,a) \leftarrow p(s,a) + \beta\delta$.

4.11 Eligibility Traces

Τα eligibility traces είναι ένας από τους πιο ισχυρούς μηχανισμούς που έχουν εισαχθεί στο πεδίο της ενισχυτικής μάθησης. Μπορούν να ειδωθούν ως ένας τρόπος υλοποίησης των Monte Carlo και Temporal Difference μεθόδων ή σαν ένας μηχανισμός που ελέγχει κατά πόσον μια τιμή θα αλλάξει όταν υπάρξει ένα TD σφάλμα. Αναλύουμε τις δύο αυτές όψεις παρακάτω.

4.11.1 λ – return

Ο παρών τρόπος ανάλυσης παρουσιάζει τα eligibility traces σαν μια γέφυρα ανάμεσα στις Monte Carlo και Temporal Difference μεθόδους. Ονομάζουμε στόχο n βημάτων την έκφραση

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}), n \in \{1, \ldots, T-t\} \tag{4.11.1}$$

και ερμηνεύουμε τις μεθόδους αξιολόγησης πολιτικής σαν μια προσπάθειας μετακίνησης των τιμών ως προς ένα στόχο. Μια Monte Carlo μέθοδος χρησιμοποιεί ανανεώσεις του τύπου

$$V(s) \leftarrow V(s) + a\left[R - V(s)\right] \tag{4.11.2}$$

δηλαδή έχει στόχο το $R_t = R_t^{(T-t)}$, δηλαδή το στόχο T-t βημάτων ενώ η TD(0) αξιολόγηση κάνει ανανεώσεις του τύπου

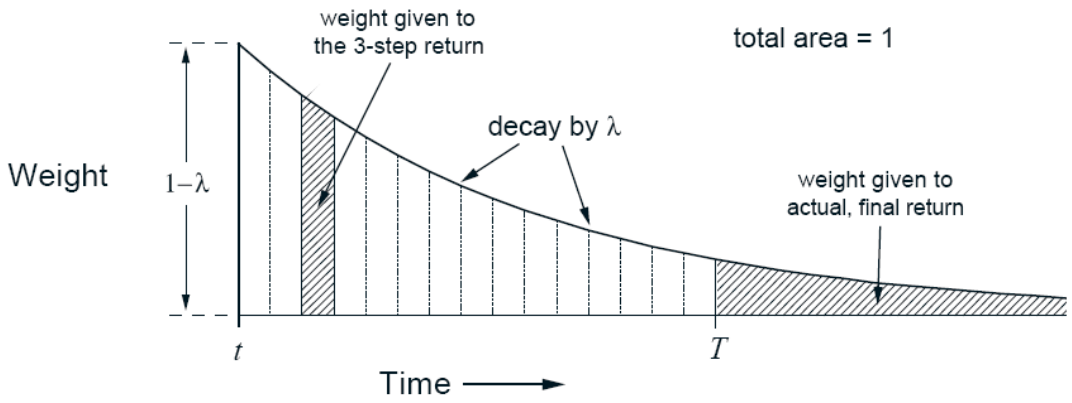
$$V(s) \leftarrow V(s) + a\left[r + \gamma V(s') - V(s)\right] \tag{4.11.3}$$

δηλαδή έχει στόχο το $r_{t+1} + \gamma V_t(s_{t+1}) = R_t^{(1)}$ δηλαδή το στόχο 1 βήματος. Ανάμεσα στις δύο αυτές μεθόδους βρίσκονται μέθοδοι που χρησιμοποιούν τους στόχους 2, 3, … βημάτων.

Ονομάζουμε επίσης λ-return το βεβαρημένο μέσο όρο όλων των στόχων n – βημάτων, δηλαδή

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t \tag{4.11.4}$$

Είναι εύκολο να δούμε ότι $R_t^1 = R_t, R_t^0 = R_t^{(0)}$, δηλαδή μια μέθοδος με στόχο το λ-return περιλαμβάνει όλο το φάσμα των μεθόδων από τις MC μέχρι τις TD, δίνοντας βάρη στους στόχους διαφόρων βημάτων όπως παρακάτω.



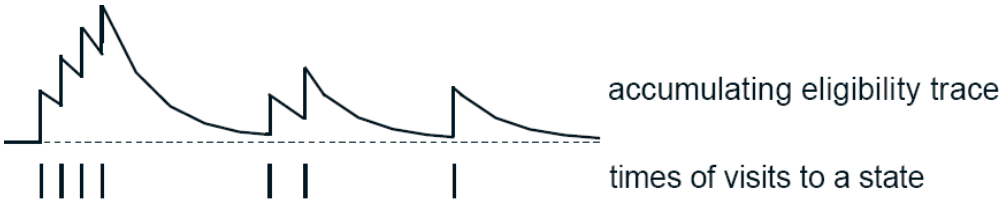
Σχήμα 4.5 (πηγή [SB98])

4.11.2 Αξιολόγηση πολιτικής TD(λ)

Ένας αλγόριθμος που αξιολογεί μια πολιτική με στόχο το λ – return μπορεί να κατασκευαστεί με την εισαγωγή των eligibility traces. Ορίζουμε

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s), s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1, s = s_t \end{cases}, \lambda \in [0,1], s \in S \qquad (4.11.5)$$

Το eligibility trace μιας κατάστασης μειώνεται με το χρόνο εκτός και αν συμβεί μια επίσκεψη στην κατάσταση αυτή, οπότε αυξάνει κατά 1. Τα eligibility traces κρατάνε λοιπόν ποιες καταστάσεις έχουν επισκεφθεί πρόσφατα, όπως παρακάτω.



Σχήμα 4.6 (πηγή [SB98])

Αυτού του είδους τα eligibility traces λέγονται συσωρευτικά. Εναλλακτικά μπορούν να χρησιμοποιηθούν eligibility traces αντικατάστασης

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s), s \neq s_t \\ 1, s = s_t \end{cases}, \lambda \in [0,1], s \in S \qquad (4.11.6)$$

Το πνεύμα της αξιολόγησης TD(λ) είναι ότι η τιμή μιας κατάστασης ανανεώνεται ανάλογα με το πόσο πρόσφατα επισκέφθηκε αλλά οι τιμές όλων των καταστάσεων ανανεώνονται. Η ανανέωση των τιμών είναι

$$V_{t+1}(s_t) = V_t(s_t) + a\left[r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\right] e_t(s_t) \qquad (4.11.7)$$

Μπορεί να αποδειχθεί ότι ο παρακάτω αλγόριθμος αξιολογεί μια πολιτική με στόχο το R_t^λ ([SB98]).

Αλγόριθμος 4.5 : TD(λ) Policy Evaluation

είσοδος: π

έξοδος: V

1. Αρχικοποίηση

$V(s), e(s) = 0, \forall s \in S$

2. Αξιολόγηση πολιτικής

repeat for each episode

repeat for each episode step

$a \leftarrow$ δράση σύμφωνα με την π

κάνε την a; δες τα s', r

$\delta \leftarrow r + \gamma V(s') - V(s)$

$e(s) \leftarrow e(s) + 1$

for each $s \in S$

$V(s) \leftarrow V(s) + a\delta e(s)$

$s \leftarrow s'$

until (s τελική)

4.11.3 Έλεγχος TD(λ)

Για την κατασκευή μεθόδων ελέγχου TD(λ) με eligibility traces σύμφωνα με τη φιλοσοφία GPI είναι αναγκαίο να γενικευτούν τα τελευταία για ζεύγη καταστάσεων – δράσεων. Αυτό μπορεί να γίνει όπως παρακάτω.

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1, s = s_t, a = a_t \\ \gamma\lambda e_{t-1}(s,a), \text{αλλιώς} \end{cases}, \lambda \in [0,1], s \in S, a \in A \qquad (4.11.8)$$

ή

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a), s \neq s_t \\ 0, s = s_t, a \neq a_t \\ 1, s = s_t, a = a_t \end{cases}, \lambda \in [0,1], s \in S, a \in A \qquad (4.11.9)$$

για eligibility traces αντικατάστασης.

Οι μέθοδοι Sarsa, Q – learning και Actor – Critic γενικεύονται έτσι εύκολα σε μεθόδους που χρησιμοποιούν eligibility traces.

Αλγόριθμος 4.10 : Sarsa(λ)

έξοδος: π

1. Αρχικοποίηση

$Q(s,a), e(s,a) = 0 \forall s \in S, a \in A$

2. Έλεγχος

repeat for each episode

$\alpha \leftarrow somehow(Q(s,a))$

repeat for each episode step

κάνε την a; δες τα s', r

$\alpha' \leftarrow somehow(Q(s',a))$

$\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$

$e(s,a) \leftarrow e(s,a) + 1$

for all s, a

$Q(s,a) \leftarrow Q(s,a) + a\delta e(s,a)$

$e(s,a) \leftarrow \gamma\lambda e(s,a)$

$s \leftarrow s'; a \leftarrow a'$

until $(s$ τελική$)$

Αλγόριθμος 4.10 : Watkins - Q(λ)

έξοδος: π

1. Αρχικοποίηση

$Q(s,a), e(s,a) = 0 \forall s \in S, a \in A$

2. Έλεγχος

repeat for each episode

$\alpha \leftarrow somehow\left(Q(s,a)\right)$

repeat for each episode step

κάνε την a; δες τα s', r

$\alpha' \leftarrow somehow\left(Q(s',a)\right)$

$a^* \leftarrow \arg\max_b \left(Q(s',b)\right)$

$\delta \leftarrow r + \gamma Q(s',a^*) - Q(s,a)$

$e(s,a) \leftarrow e(s,a) + 1$

for all s, a

$Q(s,a) \leftarrow Q(s,a) + a\delta e(s,a)$

if $(a^* = a')$ then $e(s,a) \leftarrow \gamma\lambda e(s,a)$

else $e(s,a) \leftarrow 0$

$s \leftarrow s'; a \leftarrow a'$

until $(s$ τελική$)$

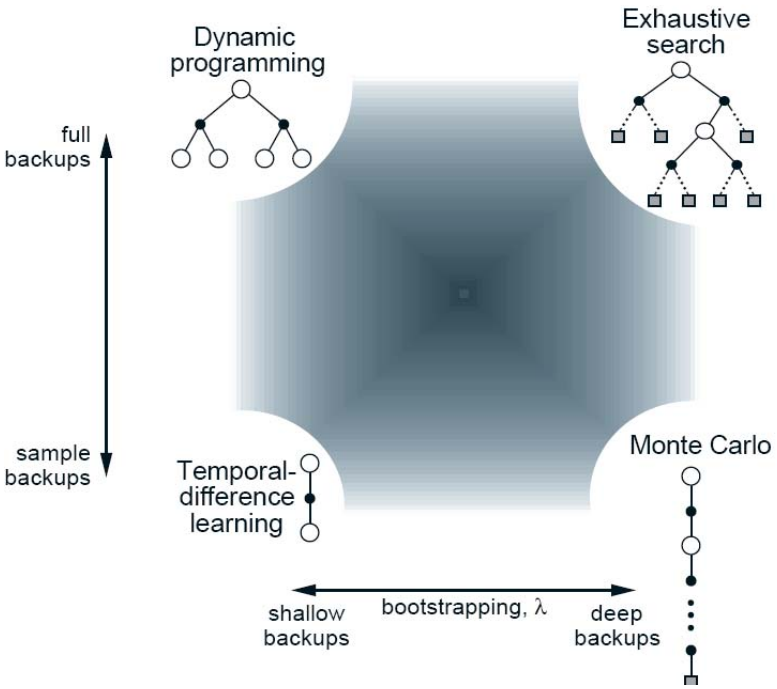
4.12 State aggregation

Στην παρουσίαση των μεθόδων ενισχυτικής μάθησης υποθέσαμε ότι η αποθήκευση των τιμών $V(s)$ ή $Q(s,a)$ γίνεται σε πινακοειδή (tabular) μορφή. Αυτό μπορεί να κάνει τον αναγκαίο χώρο αποθήκευσης που χρησιμοποιεί ο αλγόριθμος εκθετικά μεγάλο ως προς τις διαστάσεις του προβλήματος, αφού τυπικά το ίδιο συμβαίνει και για το χώρο των καταστάσεων. Η υπόθεση αυτή διευκολύνει την παρουσίαση και την ανάλυση των αλγορίθμων αλλά δεν είναι αναγκαία για την εφαρμογή τους σε δύσκολα προβλήματα. Η λύση είναι να χωρίσουμε το χώρο των καταστάσεων σε κατηγορίες (clusters) με μια ανιστοίχιση $c : S \times X \rightarrow [0,1]$ όπου X ο χώρος των clusters και $|X| \ll |S|$. Λέμε ότι η c είναι μια αντιστοίχιση hard – clustering αν κάθε κατάσταση μπορεί να ανήκει σε ένα μόνο cluster, δηλαδή $c(s,x) \in \{0,1\}, \forall s, x$, αλλιώς η c είναι μια soft – clustering αντιστοίχιση. Στο μοντέλο του προβλήματος ενισχυτικής μάθησης πια, ο πράκτορας μπορεί να δει την κατάσταση στην οποία βρίσκεται το περιβάλλον αλλά μπορεί να ανανεώνει τις συναρτήσεις

τιμών μόνο για τα clusters, δηλαδή $V : X \to \mathbb{R}, Q : X \times A \to \mathbb{R}$. Η υλοποίηση των clusters και της c είναι άλλο ένα ενδιαφέρον πρόβλημα, αφού μπορούν να χρησιμοποιηθούν ακόμα και προσαρμοστικές μέθοδοι ή νευρωνικά δίκτυα. Υπάρχουν τέλος, αποδείξεις σύγκλισης για τους πιο διαδεδομένους αλγόριθμους όταν χρησιμοποιείται soft clustering ([SJJ94]).

4.13 Διαστάσεις της Ενισχυτικής Μάθησης

Για τη λύση του προβλήματος ενισχυτικής μάθησης υπάρχει όπως είδαμε μια πληθώρα προσεγγίσεων. Στο παρακάτω σχήμα φαίνεται μια απόπειρα κατηγοριοποίησης κάποιων μεθόδων σε δύο διαστάσεις: πόσο μακριά στο μέλλον μέχρι το τέλος του επεισοδίου βλέπουν για να ανανεώσουν μια τιμή και πόσες εναλλακτικές μεταβάσεις χρησιμοποιούν.



Σχήμα 4.7 (πηγή [SB98])

Άλλες διαστάσεις είναι το επεισοδικό ή μη του προβλήματος, η συνάρτηση τιμών που χρησιμοποιεί ο αλγόριθμος, ο βαθμός της εξερεύνησης, αν ο αλγόριθμος λειτουργεί σύγχρονα ή ασύγχρονα, αν χρησιμοποιεί eligibility traces και τι είδους είναι αυτά κα. Η σχεδίαση λοιπόν ενός αλγόριθμου για ένα συγκεκριμένο πρόβλημα και η επιλογή των σχετικών παραμέτρων αποτελεί ένα πολύ σύνθετο πρόβλημα σχεδίασης.

5

Η Βελτιστοποίηση Ερωτημάτων ως Πρόβλημα Ενισχυτικής Μάθησης

Τα eddies είναι ένας πολύ ισχυρός μηχανισμός προσαρμογής του πλάνου εκτέλεσης κατά τη διάρκεια εκτέλεσης του ερωτήματος σε επίπεδο πλειάδας. Η βάρος της κατεύθυνσης της προσαρμογής και της βελτιστότητας του πλάνου πέφτει στην πολιτική δρομολόγησης. Αν και αρκετοί αλγόριθμοι δρομολόγησης έχουν προταθεί, το ζήτημα της εύρεσης ενός βέλτιστου πλάνου εκτέλεσης κατά τη διάρκεια του ερωτήματος δεν έχει αντιμετωπιστεί συνολικά. Στο κεφάλαιο αυτό δείχνουμε πώς η εκτέλεση ενός ερωτήματος με eddies μπορεί να μοντελοποιηθεί ως πρόβλημα Ενισχυτικής Μάθησης (ισοδύναμα ως Markov Decision Process) και η βελτιστοποίηση ισοδυναμεί με τη μάθηση μιας βέλτιστης πολιτικής δρομολόγησης. Ο eddy αναλαμβάνει το ρόλο του πράκτορα (agent) και το περιβάλλον είναι οι τελεστές με τους οποίους επικοινωνεί. Η μοντελοποίηση αυτή ανοίγει το δρόμο για πολλούς αλγόριθμους μάθησης με διαφορετικά χαρακτηριστικά συμπεριλαμβανομένων και των αλγορίθμων που έχουν προταθεί στη βιβλιογραφία. Ακολουθώντας το μοντέλο της ενισχυτικής μάθησης, η έρευνα για το βέλτιστο πλάνο εκτέλεσης μετακινείται από το χώρο των πλάνων στο χώρο των καταστάσεων που θα οριστεί, επιτρέποντας έτσι στην επεξεργασία του ερωτήματος να συμβαίνει παράλληλα με τη βελτιστοποίηση.

5.1 Τα προβλήματα βελτιστοποίησης στην εκτέλεση ερωτημάτων με Eddies

Κατά την εκτέλεση ενός ερωτήματος στα μοντέλα EddySelections, EddyJoins, EddySTAIRs, EddySteMs0 – EddySteMs5 προκύπτουν αρκετά ανεξάρτητα προβλήματα βελτιστοποίησης. Η βέλτιστη εκτέλεση ενός ερωτήματος συνίσταται στην λύση κάποιων ή όλων από τα εν λόγω προβλήματα. Στην ενότητα αυτή επιχειρούμε μια καταγραφή τους.

5.1.1 Σειρά των επιλογών

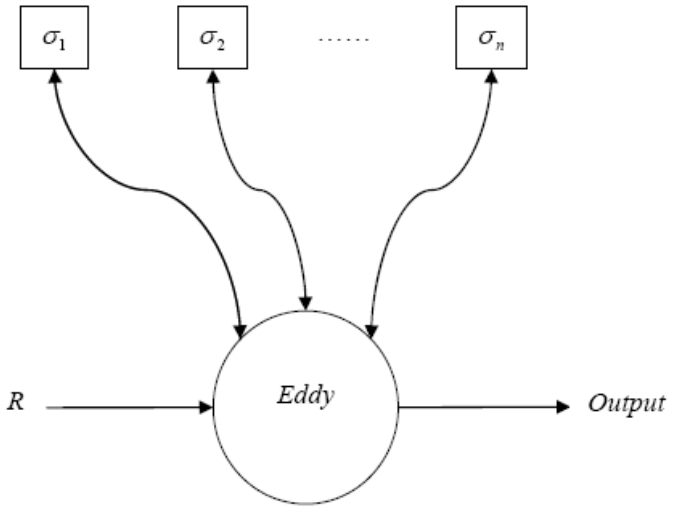
Δεδομένου ενός ερωτήματος σύζευξης επιλογών σε μία σχέση ποια είναι η σειρά εφαρμογής των επιλογών που ελαχιστοποιεί το κόστος του ερωτήματος. Το ερώτημα στο οποίο παρουσιάζεται το πρόβλημα αυτό απομονωμένο είναι το ακόλουθο.

select *
from R
where $p_1 \wedge p_2 \wedge \cdots \wedge p_n$

που μετασχηματίζεται στην παρακάτω έκφραση σχεσιακής άλγεβρας

$$\sigma_{i_1}\left(\sigma_{i_2}\left(\ldots\sigma_{i_n}\left(R\right)\right)\right)\,|\,[i_1,\ldots,i_n] = permutation\left(\{1,2,\ldots,n\}\right)$$

και αρχικοποιείται για εκτέλεση με eddies όπως παρακάτω



Σχήμα 5.1

Το πρόβλημα αυτό είναι σχετικά εύκολο, έχει μελετηθεί διεξοδικά στη βιβλιογραφία και πολλές καλές λύσεις έχουν προταθεί. Οι επιλογές είναι τυπικά φτηνοί τελεστές, οπότε μια πολύπλοκη διαδικασία μάθησης με μεγάλο υπολογιστικό κόστος δεν ταιριάζει παρά μόνο σε

λίγες περιπτώσεις. Παρ' όλ' αυτά ερωτήματα που περιλαμβάνουν συνδέσμους ειδικού τύπου (πχ ερωτήσεις σε βάσεις δεομένων με σχήματα αστέρα) μπορούν να αναχθούν σε αυτό το πρόβλημα. Στην ενότητα 5.3 δίνουμε την προσέγγιση ενισχυτικής μάθησης στην εκτέλεση ενός τέτοιου ερωτήματος με eddies.

5.1.2 Σειρά των συνδέσμων

Δεδομένου ενός ερωτήματος φυσικών συνδέσμων πάνω σε n σχέσεις, ποια είναι η καλύτερη σειρά με την οποία μπορούν να εκτελεστούν οι σύνδεσμοι σε σχέση με το κόστος εκτέλεσης. Στην παρούσα ενότητα καθώς και στην ενότητα 5.4 περιοριζόμαστε στην μελέτη του παραπάνω προβλήματος σε μη κυκλικά ερωτήματα στην περίπτωση που όλοι οι σύνδεσμοι υλοποιούνται με κάποιον αλγόριθμο ο οποίος επιτρέπει στο πλάνο εκτέλεσης να αλλάζει ανά πάσα στιγμή, τύπου Symmetric Hash Join. Οι περιορισμοί αυτοί θα απομακρυνθούν κατά τη μελέτη των SteMs στις ενότητες 5.6, 5.7. Επίσης, στην εκτέλεση ενός ερωτήματος με eddies δεν υπάρχουν materialization points, αν και μια τέτοια επέκταση δεν είναι δύσκολη. Κατά τα άλλα, η βελτιστοποίηση θεωρεί όλη την κλάση των bushy δέντρων. Υπό τους περιορισμούς αυτούς, το πρόβλημα συναντάται σε τρεις ισοδύναμες αρχιτεκτονικές.

1. Ερωτήματα με δυαδικούς συνδέσμους τύπου SHJ (μοντέλο EddyJoins)
2. Ερωτήματα με STAIRs στα οποία ισχύει η Dual Routing Property (μοντέλο EddySTAIRs)
3. Ερωτήματα με SteMs τα οποία προσομοιώνουν ένα n – way SHJ (μοντέλο EddySteMs0)

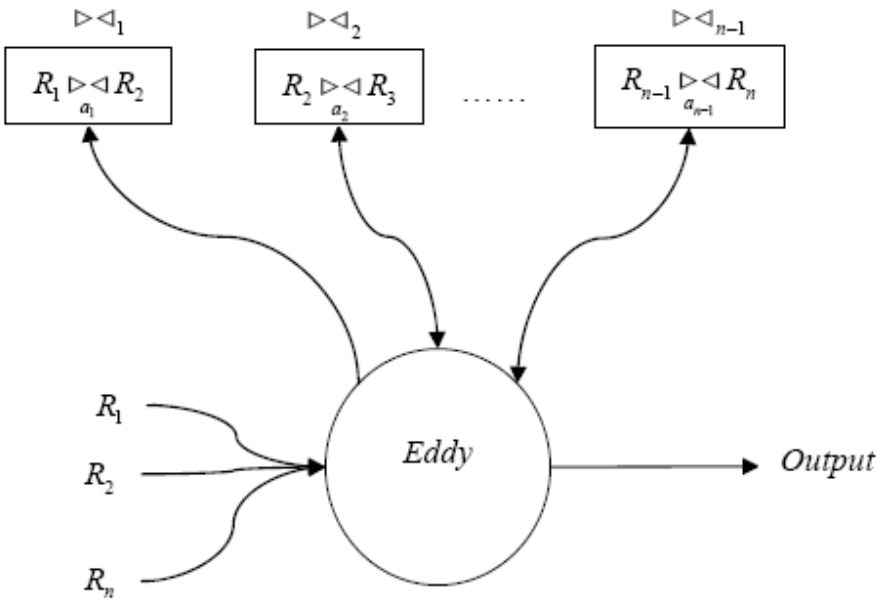
Το ερώτημα που παρουσιάζει το πρόβλημα αυτό απομονωμένο είναι το

select *
from $R_1, R_2, \ldots, R_n$
where $R_1.a_1 = R_2.a_1$ AND … AND $R_{n-1}.a_{n-1} = R_n.a_{n-1}$

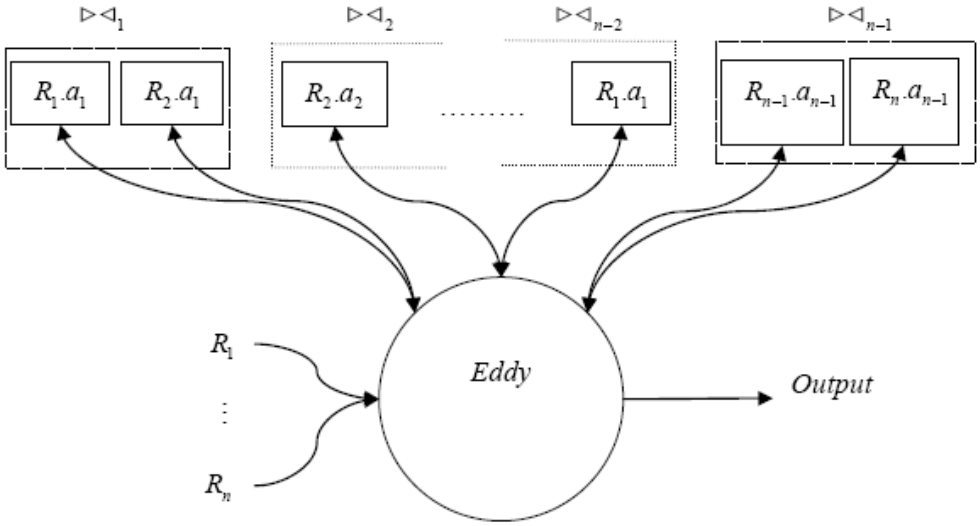
που μετασχηματίζεται στην ακόλουθη παράσταση σχεσιακής άλγεβρας

$$R_1 \underset{a_1}{\triangleright\!\triangleleft} R_2 \underset{a_2}{\triangleright\!\triangleleft} R_3 \underset{a_3}{\triangleright\!\triangleleft} \cdots \underset{a_{n-2}}{\triangleright\!\triangleleft} R_{n-1} \underset{a_{n-1}}{\triangleright\!\triangleleft} R_n$$

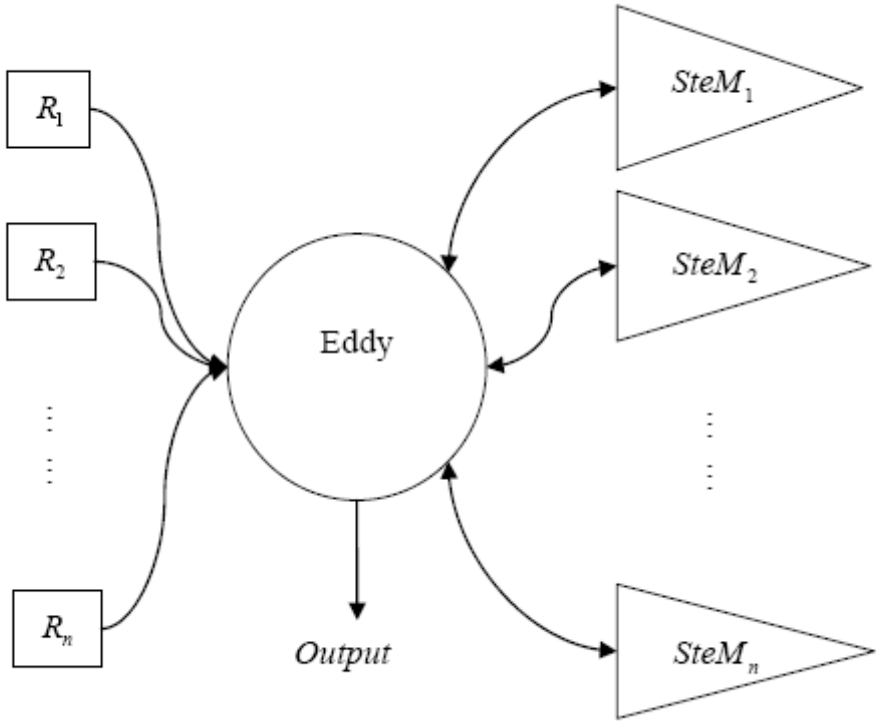
Το ερώτημα αυτό αρχικοποιείται προς εκτέλεση στις τρεις προαναφερθείσες αρχιτεκτονικές όπως παρακάτω.



Σχήμα 5.2



Σχήμα 5.3



Σχήμα 5.4

5.1.3 Άρση του «βάρους της ιστορίας»

Στο μοντέλο επεξεργασίας EddySTAIRs, ο eddy μπορεί να μετακινήσει πλειάδες από ένα STAIR σε ένα άλλο αν ο τρέχων τρόπος αποθήκευσης δεν συνάδει με την τρέχουσα πολιτική δρομολόγησης. Η συχνότητα με την οποία γίνεται αυτό καθώς και το ποιες πλειάδες θα μετακινηθούν είναι άλλο ένα πρόβλημα που μπορεί να αντιμετωπιστεί με μεθόδους ενισχυτικής μάθησης.

5.1.4 Επιλογή αλγορίθμων συνδέσμων

Στο μοντέλο επεξεργασίας ερωτημάτων με SteMs EddySteMs1, ο eddy μπορεί να αλλάζει δυναμικά τον αλγόριθμο υλοποίησης συνδέσμου ανάμεσα σε δύο σχέσεις. Ερευνούμε με τι κριτήρια και πώς μπορεί να γίνει η αλλαγή αυτή ώστε να κατευθύνεται προς μια βέλτιστη εκτέλεση του ερωτήματος.

5.1.5 Ανταγωνισμός των Access Methods

Στο μοντέλο επεξεργασίας ερωτημάτων με SteMs, ο eddy μπορεί να επιλέγει τον τρόπο που αποκτά πρόσβαση σε πλειάδες των σχέσεων, χρησιμοποιώντας εναλλακτικά scans ή ευρετήρια.

5.2 Στόχοι της μοντελοποίησης και σχεδιαστικές επιλογές

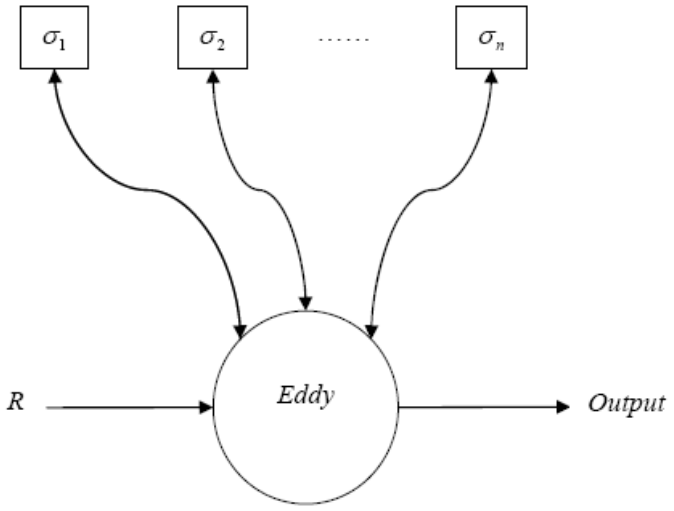
Η μοντελοποίηση ενός πραγματικού δύσκολου προβλήματος στα πλαίσια της ενισχυτικής μάθησης περιλαμβάνει πολλές επιμέρους προκλήσεις και επιλογές. Στα πλαίσια της εργασίας αυτής, κατ' αρχήν επιχειρούμε μια διάσπαση του προβλήματος σε επιμέρους, όπως αυτά αναφέρθηκαν στην ενότητα 5.1. Ο συνδυασμός των αλγόριθμων μάθησης των επιμέρους προβλημάτων συνιστά και τη λύση του συνολικού προβλήματος, όμως ο βαθμός χρονικής επικάλυψης των αλγορίθμων είναι ένα ακόμα δύσκολο πρόβλημα.

Σαν μια σχεδιαστική επιλογή που διαπερνά όλη την ανάλυση, είναι ότι θεωρούμε την επεξεργασία των πλειάδων ως ένα επεισοδικό πρόβλημα, με την επεξεργασία μιας πλειάδας να συνιστά ένα επεισόδιο. Επίσης, θεωρούμε ότι δεν υπάρχει κάποιο μοντέλο για τα δεδομένα (ιστογράμματα, εκτιμήσεις μεγέθους κλπ) αλλά ο eddy πρέπει να μάθει ότι μπορεί για τα δεδομένα στη διάρκεια εκτέλεσης του ερωτήματος. Ο στόχος είναι η επεξεργασία να ανταποκρίνεται σε δεδομένα που αλλάζουν γρήγορα ή που δεν έχουμε πληροφορίες γι' αυτά. Οι πιθανές εκτιμήσεις που θα υπάρχουν όμως σε ένα πραγματικό σύστημα, μπορούν να χρησιμοποιηθούν στην αρχικοποίηση των συναρτήσεων τιμών για τους αλγόριθμους.

Κατά τη διάρκεια του κεφαλαίου αυτού, υιοθετούμε ένα σύγχρονο περιβάλλον επεξεργασίας ερωτημάτων, παρόμοιο με αυτό της ενότητας 3.10, το οποίο μας επιτρέπει να χειριστούμε το χρόνο σαν μια ακολουθία διακριτών βημάτων σε κάθε ένα από τα οποία ο eddy λαμβάνει μια απόφαση.

Η πρώτη φάση της μοντελοποίησης που είναι και ο κύριος όγκος της παρούσας εργασίας παρουσιάζεται σε αυτό το κεφάλαιο. Στόχος της είναι ο ορισμός των συνόλων καταστάσεων και δράσεων και ο ορισμός των συναρτήσεων ενισχύσεων για τα προβλήματα βελτιστοποίησης. Τα παραπάνω πρέπει να γίνουν με ένα τρόπο κατά τον οποίο ο ανταγωνισμός των δράσεων θα οδηγεί σε μια βέλτιστη εκτέλεση του ερωτήματος με βάση τα μοντέλα κόστους. Η δεύτερη φάση που περιγράφεται στο κεφάλαιο 6 συνίσταται στην επιλογή μιας συνάρτησης μελλοντικών ενισχύσεων και επιλογή αλγόριθμων μάθησης και των σχετικών παραμέτρων.

5.3 Μάθηση της σειράς των επιλογών



Σχήμα 5.5

Στην ενότητα αυτή ασχολούμαστε με το πρόβλημα selection order. Ένας τελεστής eddy συνδέεται με ένα scan σε μία σχέση και σε n τελεστές επιλογής. Σε κάθε χρονική στιγμή t συμβαίνει ένα από τα παρακάτω.

1. Ο eddy καλεί την get_next() του scan το οποίο εισάγει μια καινούρια πλειάδα στο dataflow, ή η ειδική πλειάδα null που επιστρέφεται δείχνει στον eddy ότι δεν υπάρχουν άλλες πλειάδες στη σχέση και η εκτέλεση τερματίζεται.
2. Ο eddy δρομολογεί μια πλειάδα από το dataflow σε ένα τελεστή επιλογής, ο οποίος την επιστρέφει αν το κατηγόρημα ήταν αληθές ή επιστρέφει null αν αυτό ήταν ψευδές για αυτή την πλειάδα.
3. Ο eddy προωθεί μια πλειάδα στην έξοδο.

Υποθέτουμε, τέλος ότι ο eddy διατηρεί ένα πίνακα από bits που ονομάζεται donebits όπως αυτός περιγράφεται στην ενότητα 3.3.

Ο πράκτορας του προβλήματος ενισχυτικής μάθησης είναι ο eddy και το περιβάλλον οι τελεστές επιλογής και scan. Το σύνολο καταστάσεων είναι το ακόλουθο

$$S = \left\{ s_i \mid \left\langle donebits \right\rangle_2 = i \right\} \cup \left\{ null \right\}, \left| S \right| = 2^n + 1$$

δηλαδή κατάσταση είναι η υπογραφή της τρέχουσας πλειάδας. Το σύνολο των δράσεων είναι το

$$A = \left\{ \sigma_i \mid i = 1, \ldots, n \right\} \cup \left\{ get\left(R\right) \right\} \cup \left\{ output \right\}, \left| A \right| = n + 2$$

δηλαδή οι δράσεις που μπορεί να κάνει ο πράκτορας είναι να δρομολογήσει μια πλειάδα στον τελεστή επιλογής i, να προωθήσει την πλειάδα στην έξοδο ή να ζητήσει μια καινούρια πλειάδα. Η επόμενη κατάσταση από μια δράση σ_i είναι ένας ανανεωμένος πίνακας από done bits ή η κατάσταση null. Η επόμενη κατάσταση από μια δράση scan είναι μια καινούρια πλειάδα ή η κατάσταση null, και η επόμενη κατάσταση από μια δράση output είναι η κατάσταση null.

Παρακάτω φαίνονται τα σύνολα $A(s)$ τα οποία ενσωματώνουν τους σημασιολογικούς περιορισμούς του ερωτήματος.

$A(s_{n-1}) = \{output\}$

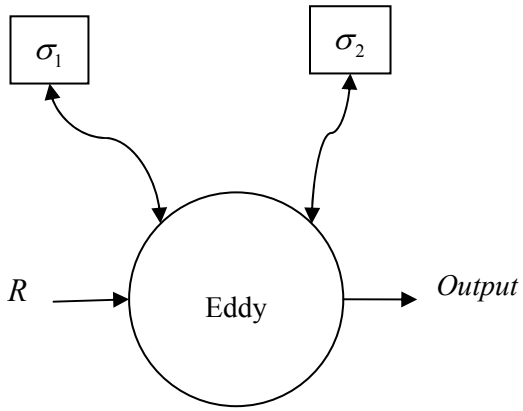
$A(null) = \{get(R) \mid \text{οι πλειάδες της } R \text{ δεν έχουν τελειώσει}\}$

$A(s_i) = \{\sigma_j \mid \text{το } j \text{ done bit είναι } 0\}, i \neq n-1$

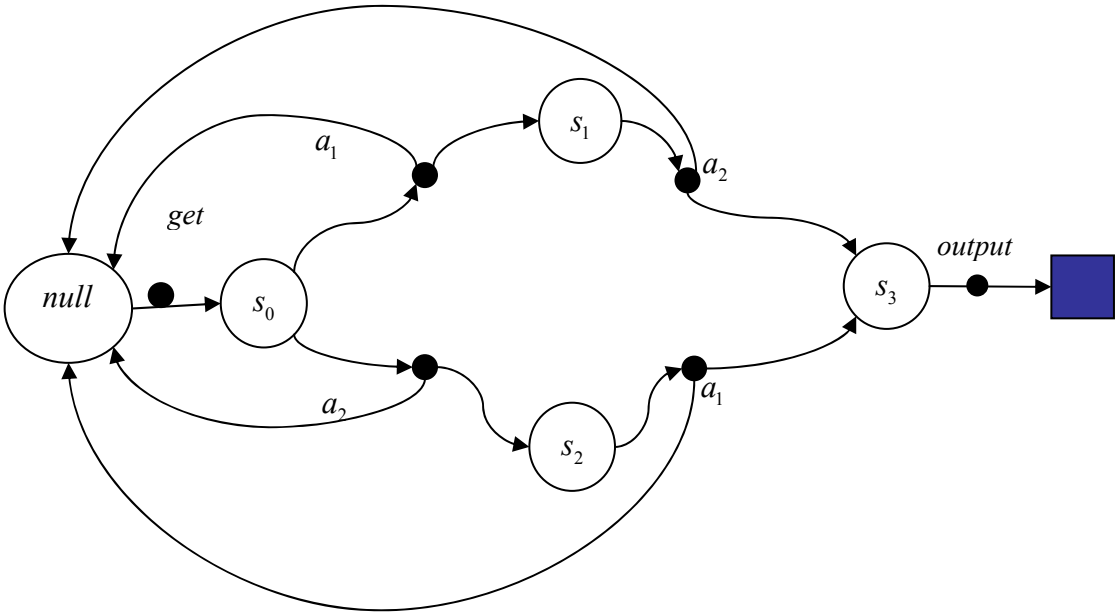
Το πρόβλημα είναι προφανώς επεισοδικό με αρχική κατάσταση την *null* και με ένα επεισόδιο να συνιστά την επεξεργασία μιας πλειάδας.

Παράδειγμα

Έστω ένα ερώτημα με δύο επιλογές. Η μοντελοποίησή του φαίνεται παρακάτω.



s	*done bits*	$A(s)$
s_0	00	{1,2}
s_1	01	{2}
s_2	10	{1}
s_3	11	*output*



Μπορούμε να κατασκευάσουμε το μοντέλο Markov (πιθανότητες μεταβάσεων) όπως παρακάτω. Συμβολίζουμε με $sel(i)$ την επιλεκτικότητα της επιλογής σ_i και αγνοούμε την κατάσταση null και τις μεταβάσεις από αυτή.

$$P_{ss'}^{a_1} = \begin{bmatrix} 1-sel(1) & sel(1) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1-sel(1) & 0 & 0 & sel(1) \\ 0 & 0 & 0 & 0 \end{bmatrix}, P_{ss'}^{a_2} = \begin{bmatrix} 1-sel(2) & 0 & sel(2) & 0 \\ 1-sel(2) & 0 & 0 & sel(2) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, P_{ss'}^{output} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Σε μια τόσο απλή περίπτωση μπορούμε ακόμα και να λύσουμε τις εξισώσεις βέλτιστου Bellman υποθέτοντας ότι $V^*(s_3) = 0$

$$V^*(s_3) = 0$$

$$V^*(s_1) = P_{s_1s_3}^{a_2}\left[R_{s_1s_3}^{a_2} + \gamma V^*(s_3)\right] + P_{s_1s_0}^{a_2}\left[R_{s_1s_0}^{a_2} + \gamma V^*(s_0)\right]$$

$$V^*(s_2) = P_{s_2s_3}^{a_1}\left[R_{s_2s_3}^{a_1} + \gamma V^*(s_3)\right] + P_{s_2s_0}^{a_1}\left[R_{s_2s_0}^{a_1} + \gamma V^*(s_0)\right]$$

$$V^*(s_0) = \max_{a\in\{a_1,a_2\}}\left\{P_{s_0s_1}^{a}\left[R_{s_0s_1}^{a} + \gamma V^*(s_1)\right] + P_{s_0s_0}^{a}\left[R_{s_0s_0}^{a} + \gamma V^*(s_0)\right] + P_{s_0s_2}^{a}\left[R_{s_0s_2}^{a} + \gamma V^*(s_2)\right]\right\}$$

$$\Rightarrow V^*(s_0) = -\left(1 + \frac{1}{\max(sel_1, sel_2)}\right)$$

οπότε επιλέγεται ο τελεστής με την μικρότερη επιλεκτικότητα αν χρησιμοποιηθεί η συνάρτηση ενίσχυσης που δεν λαμβάνει υπ' όψιν το κόστος (βλέπε επόμενη υπο – ενότητα).

5.3.1 Ενισχύσεις των δράσεων

Η επιλογή της συνάρτησης ενίσχυσης πρέπει να είναι τέτοια ώστε να οδηγεί τον ανταγωνισμό των επιλογών προς τη κατεύθυνση της βέλτιστης εκτέλεσης του ερωτήματος. Υιοθετούμε γενικά αρνητικές ενισχύσεις. Ένας τελεστής επιλογής θεωρείται καλός αν το κατηγόρημά του δεν είναι επιλεκτικό και έχει μικρό χρόνο επεξεργασίας. Αγνοώντας το κόστος επεξεργασίας, ορίζουμε

$$r\left(s,\sigma_i\right)=\begin{cases}0,\ s_{t+1}=null\\-1,\ s_{t+1}\neq null\end{cases}$$

δηλαδή η ενίσχυση είναι -1 αν η επιλογή επιστρέψει την πλειάδα στον eddy και 0 αν επιστρέψει null. Η εκτίμηση τέτοιων τιμών πχ από ένα Monte Carlo αλγόριθμο προσεγγίζει την επιλεκτικότητα της επιλογής. Η παραπάνω ενίσχυση που δεν λαμβάνει υπ' όψιν το κόστος ταιριάζει σε ένα multi – threaded περιβάλλον επεξεργασίας λόγω του back – pressure effect ([AH00]). Το κόστος μπορεί να ενσωματωθεί στην ενίσχυση «χρεώνοντας» τον τελεστή λίγο ακόμα κι αν της δεν επέστρεψε την πλειάδα. Το κόστος από εδώ και πέρα έχει την έννοια του χρόνου που πέρασε μέχρι ο τελεστής να επεξεργαστεί την πλειάδα.

$$r\left(s,\sigma_i\right)=\begin{cases}-\varepsilon\cdot cost,\ s_{t+1}=null\\-cost,\ s_{t+1}\neq null\end{cases},\varepsilon\simeq 0$$

Η εκτίμηση των παραπάνω ενισχύσεων προσεγγίζει το συνολικό κόστος επεξεργασίας της σχέσης από μια επιλογή. Η ενίσχυση της προώθησης μιας πλειάδας στην έξοδο πρέπει να είναι 0 (ώστε να γίνεται όποτε αυτή είναι δυνατή), ενώ η ενίσχυση της ανάγνωσης από τη σχέση μπορεί να είναι 0 ή να λαμβάνει υπ' όψιν τον χρόνο ανάγνωσης (αυτό είναι χρήσιμο όταν υπάρχουν ανταγωνιστικά scans στη σχέση και περιγράφεται στην ενότητα 5.6)

$$r\left(output\right)=0$$

$$r\left(get\left(R\right)\right)=0\ \text{ή}\ r\left(get\left(R\right)\right)=-\varepsilon\cdot cost$$

5.3.2 State aggregation

Η παραπάνω μοντελοποίηση αν και ανταποκρίνεται διαισθητικά της σχεδιαστικές απαιτήσεις υποφέρει από ένα σημαντικό πρόβλημα. Ο χώρος των καταστάσεων με την παραπάνω μοντελοποίηση αυξάνει εκθετικά ως της το πλήθος των επιλογών. Της, το selection order είναι διαισθητικά ένα πρόβλημα χωρίς καταστάσεις (stateless), δηλαδή η έννοια του delayed reward δεν έχει έννοια εδώ. Η λύση έρχεται με την τεχνική του state aggregation (ενότητα 4.12). Ενοποιούμε παρόμοιες καταστάσεις σε clusters $x\in X$ τω $\left|X\right|\ll\left|S\right|$ μέσω μιας αντιστοιχίας $m:S\to X$ (hard clustering). Ο eddy μπορεί να βλέπει την κατάσταση της

πλειάδας, η οποία χρησιμεύει για τον ορισμό του εκάστοτε $A(s)$, αλλά μπορεί να κρατάει τιμές $V(x)$ ή $Q(x,a)$ μόνο για τα clusters. Υπάρχουν τουλάχιστον δύο καλοί διαισθητικά τρόποι για να γίνει αυτό.

Ο πρώτος τρόπος ακολουθεί την λογική της Lottery Scheduling [WW94] και αντιμετωπίζει το πρόβλημα ως πρόβλημα χωρίς καταστάσεις, το ανάγει δηλαδή σε ένα n – armed bandit πρόβλημα. Ορίζουμε λοιπόν

$X = \{x_0\}$ και $m(s) = x_0, \forall s$,

δηλαδή ο eddy αποθηκεύει μία μόνο τιμή για κάθε τελεστή $Q(s,a) = Q(a)$. Με τον τρόπο αυτό, ο eddy δεν μαθαίνει την καλύτερη σειρά των επιλογών αλλά μάλλον την καλύτερη επιλογή. Για να χρησιμοποιηθεί κάτι τέτοιο, η πολιτική πρέπει να είναι γενικά soft, ώστε όταν η καλύτερη δράση δεν είναι δυνατή, ο eddy να δρομολογεί την πλειάδα σε ένα άλλο μη – βέλτιστο τελεστή.

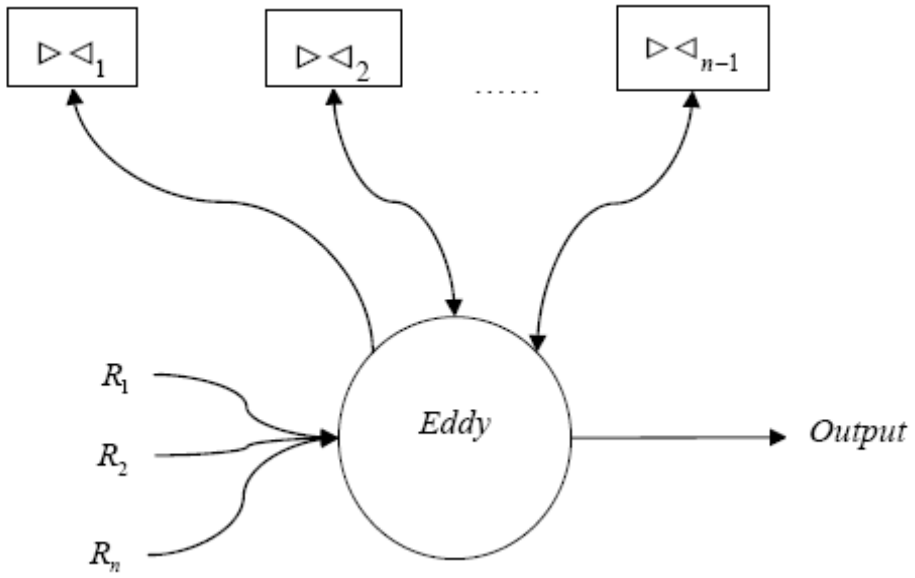
Μια προσέγγιση που βρίσκεται ενδιάμεσα από το πλήρες σύνολο καταστάσεων και την μετατροπή του προβλήματος σε stateless είναι να ορίσουμε

$X = \{x_i \mid i = 1, \ldots, n\}$ και

$m(s) = x_i$ ανν i done bits της πλειάδας = 1

Με τον παραπάνω τρόπο, επιχειρούμε μια «κάθετη» ομαδοποίηση των καταστάσεων, έτσι ώστε της οι πλειάδες που έχουν περάσει από ένα τελεστή ανήκουν στην κατάσταση x_1, αυτές που έχουν περάσει από δύο τελεστές ανήκουν στη x_2 κοκ. Ένα πλεονέκτημα της προσέγγισης της είναι ότι η έννοια της πολιτικής $\pi : S \times A \to [0,1]$ του πράκτορα έχει πολύ καλό διαισθητικό νόημα αφού μπορεί να περιγραφεί ως «πρώτα μια πλειάδα στέλνεται στην επιλογή 1, μετά στην επιλογή 3 κλπ». Η πολιτική πρέπει για τους ίδιους λόγους με την πρώτη προσέγγιση να είναι soft.

5.4 Μάθηση της σειράς των συνδέσμων



Σχήμα 5.6

Στην ενότητα αυτή ασχολούμαστε με το πρόβλημα join order με της περιορισμούς που αναφέρθηκαν στην ενότητα 5.1.2. Υποθέτουμε το αφαιρετικό μοντέλο επεξεργασίας που φαίνεται στο σχήμα 5.6 όπου κάθε σύνδεσμος λειτουργεί ως SHJ. Το σύνολο των δράσεων που μπορεί να κάνει ο eddy είναι

$$A = \left\{ \triangleright\triangleleft_i \mid i = 1,\ldots,n-1 \right\} \bigcup \left\{ get\left(R_i\right) \mid i = 1,\ldots,n \right\}$$

όπου $get\left(R_i\right)$ είναι η κλήση της get_next() στο scan της σχέσης R_i, ενώ $\triangleright\triangleleft_i$ σημαίνει δρομολόγηση της τρέχουσας πλειάδας στον σύνδεσμο $R_i \underset{a_i}{\triangleright\triangleleft} R_{i+1}$. Όπως έχει αναφερθεί, το πρόβλημα της εύρεσης της σειράς των συνδέσμων εμφανίζεται στις τρεις αρχιτεκτονικές EddyJoins, EddySTAIRs και EddySteMs0 με ισοδύναμο τρόπο. Η δράση $\triangleright\triangleleft_i$ ανάγεται της ακόλουθες πράξεις ανάλογα με την αρχιτεκτονική.

1. EddyJoins: Το $R_i \underset{a_i}{\triangleright\triangleleft} R_{i+1}$ είναι ένα SHJ. Η δράση $\triangleright\triangleleft_i$ αν εφαρμοστεί σε μια πλειάδα που περιέχει την R_i, αποθηκεύει την πλειάδα στον αντίστοιχο πίνακα κατακερματισμού του συνδέσμου και αναζητά τα matches της, τα οποία και επιστρέφει στον eddy αν υπάρχουν αλλιώς επιστρέφει null. Αντίστοιχα και για μια πλειάδα που περιέχει την R_{i+1}.
2. EddySTAIRs. Το $R_i \underset{a_i}{\triangleright\triangleleft} R_{i+1}$ αποτελείται από δύο δυαδικά STAIRs, το $R_i.a_i$ και το $R_{i+1}.a_i$. Η δράση $\triangleright\triangleleft_i$ αν εφαρμοστεί σε μια πλειάδα που περιέχει την R_i (R_{i+1})

σημαίνει ότι αυτή γίνεται build στο $R_i.a_i$ ($R_{i+1}.a_i$) και κάνει probe στο $R_{i+1}.a_i$ ($R_i.a_i$). Στον eddy επιστρέφονται τα matches ή null αν αυτά δεν υπάρχουν, δηλαδή αν $\triangleright\triangleleft_i .STAIR_1 \equiv R_i.a_i$ και $\triangleright\triangleleft_i .STAIR_2 \equiv R_{i+1}.a_i$

$\triangleright\triangleleft_i \equiv build\left(\triangleright\triangleleft_i .STAIR_1, t\right) probe\left(\triangleright\triangleleft_i .STAIR_2, t\right)$ ή

$\triangleright\triangleleft_i \equiv build\left(\triangleright\triangleleft_i .STAIR_2, t\right) probe\left(\triangleright\triangleleft_i .STAIR_1, t\right)$

Υπενθυμίζουμε ότι ισχύει η Dual Routing Constraint της ενότητας 3.6.

3. EddySteMs0: Το $R_i \underset{a_i}{\triangleright\triangleleft} R_{i+1}$ εκτελείται με τη βοήθεια των $SteM_{R_i}$ και $SteM_{R_{i+1}}$. Η μόνη διαφορά με την αρχιτεκτονική των STAIRs είναι ότι μόνο singleton πλειάδες αποθηκεύονται. Δηλαδή αν η πλειάδα ανήκει σε μία από της σχέσεις R_i, R_{i+1}

$\triangleright\triangleleft_i \equiv build\left(SteM_{R_i}, t\right) probe\left(SteM_{R_{i+1}}, t\right)$ ή

$\triangleright\triangleleft_i \equiv build\left(SteM_{R_{i+1}}, t\right) probe\left(SteM_{R_i}, t\right)$

ενώ αν η πλειάδα περιέχει μια από της σχέσεις ως base – table component,

$\triangleright\triangleleft_i \equiv probe\left(SteM_{R_i}, t\right)$ ή

$\triangleright\triangleleft_i \equiv probe\left(SteM_{R_{i+1}}, t\right)$

Υπενθυμίζουμε ότι ισχύει ο περιορισμός Atomicity της ενότητας 3.7.1.

Ορίζουμε ως σύνολο καταστάσεων το

$S = \{\text{σχέσεις της τρέχουσας πλειάδας}\} \cup \{null\}$

Η επόμενη κατάσταση μιας δράσης $\triangleright\triangleleft_i$ είναι οι σχέσεις των matches αν αυτά υπάρχουν ή $null$ στην αντίθετη περίπτωση. Η επόμενη κατάσταση μιας δράσης $get\left(R_i\right)$ είναι R_i αν το scan επιστρέψει μια πλειάδα, ή $null$ στην περίπτωση που όλη η σχέση έχει σαρωθεί. Πρέπει να σημειωθεί βέβαια, ότι το προαναφερθέν σήμα κατάστασης μπορεί να προκύψει πλήρως από τους πίνακες ready bits και done bits, απλά η παραπάνω περιγραφή κάνει την ανάλυση πιο εύκολη.

Με το φορμαλισμό αυτό έχουμε $|S| = n + (n-1) + (n-2) + \cdots + 1 = \frac{n(n+1)}{2} = O\left(n^2\right)$ καταστάσεις και $|A| = n = O\left(n\right)$ δράσεις (όπου n είναι ο αριθμός των σχέσεων) οπότε ο

χώρος αποθήκευσης της tabular αλγορίθμου μάθησης θα είναι $O\left(n^2\right)$ αν της χρησιμοποιεί της τιμές $V\left(s\right)$ και $O\left(n^3\right)$ αν χρησιμοποιεί της $Q\left(s,a\right)$.

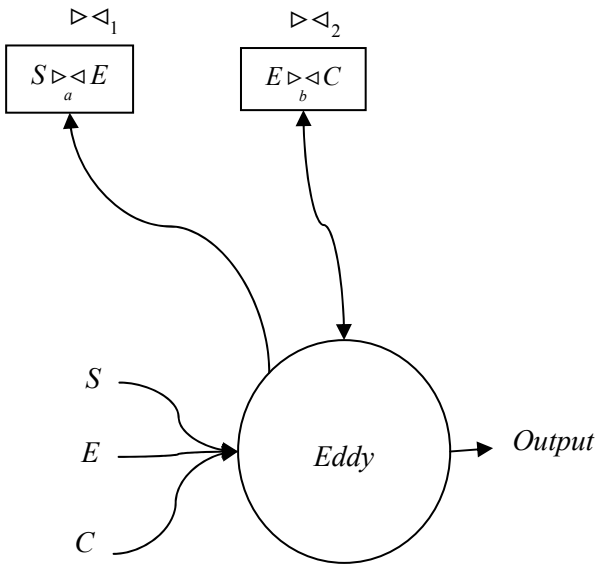
Τα σύνολα $A\left(s\right)$ μπορούν να υπολογιστούν πλήρως από την κατάσταση, της παρακάτω

$$A\left(null\right)=\left\{get\left(R_i\right)\mid i=1,\ldots,n\wedge \text{οι πλειάδες της } R_i \text{ δεν έχουν τελειώσει}\right\}$$
$$A\left(s\right)=\left\{\triangleright\triangleleft_i\mid \text{η } s \text{ περιέχει την } R_i \text{ ή την } R_{i+1}\right\}$$

Το πρόβλημα είναι επεισοδικό, με ένα επεισόδιο να περιλαμβάνει την επεξεργασία μιας πλειάδας (σύνθετης ή singleton). Η βασική παραδοχή που κάνουμε, είναι ότι η επόμενη κατάσταση μετά από μια πράξη $\triangleright\triangleleft_i$ είναι η κατάσταση που προκύπτει από τα matches ή null, κάτι που συμμορφώνεται με το σύγχρονο μοντέλο της ενότητας 3.10. Στην περίπτωση του join order, ένα σχήμα state aggregation μάλλον θα ζημίωνε το μοντέλο, αφού σε ένα ερώτημα συνδέσμων το τίμημα (μελλοντική ενίσχυση) μιας απόφασης μπορεί να φανεί πολύ μακροπρόθεσμα.

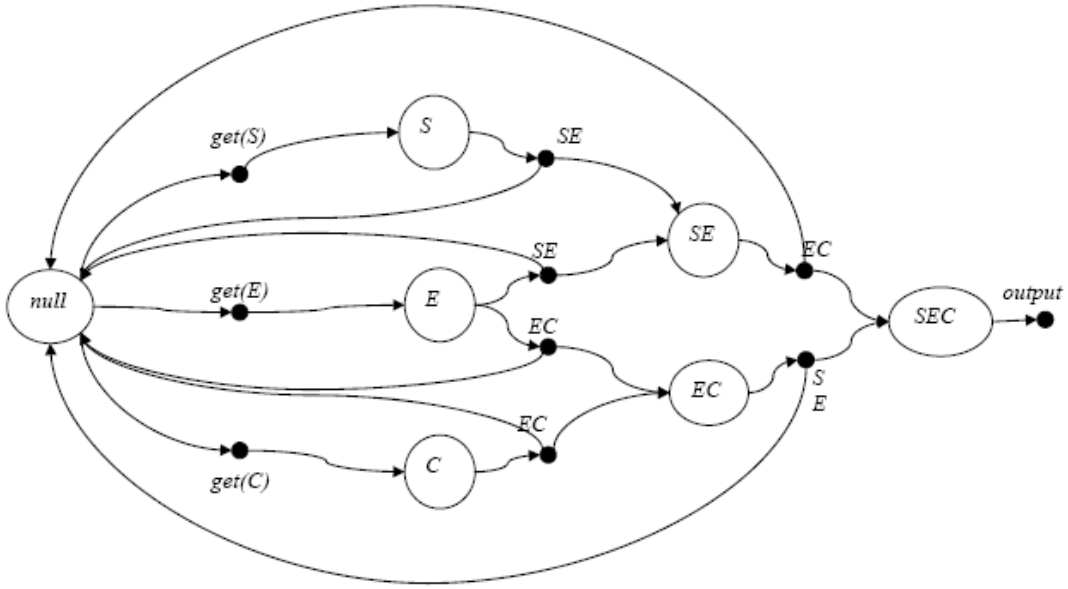
Παράδειγμα

Ένα ερώτημα με δύο συνδέσμους μοντελοποιείται όπως φαίνεται παρακάτω.



s	*tables*	$A(s)$
s_1	S	$\{\triangleright\triangleleft_1\}$
s_2	E	$\{\triangleright\triangleleft_1,\triangleright\triangleleft_2\}$
s_3	C	$\{\triangleright\triangleleft_2\}$
s_4	SE	$\{\triangleright\triangleleft_2\}$
s_5	EC	$\{\triangleright\triangleleft_1\}$
s_6	SEC	*output*

Το πρόβλημα είναι επεισοδικό και το διάγραμμα της διαδικασίας Markov φαίνεται παρακάτω.



5.4.1 Ενισχύσεις των δράσεων

Η συνάρτηση ενίσχυσης των συνδέσμων πρέπει να κατευθύνει τις επιλογές προς μια βέλτιστη εκτέλεση του ερωτήματος. Σε ένα σύνδεσμο ενδιαφέρει το κόστος επεξεργασίας και το μέγεθος της σχέσης που θα παραχθεί. Αγνοώντας προσωρινά το κόστος, ορίζουμε

$r\left(s, \triangleright\triangleleft_i\right) = -$αριθμός των matches για την πλειάδα

Η εκτίμηση των τιμών θα καταλήγει λοιπόν σε μια προσέγγιση του fanout του συνδέσμου (μέσος αριθμός των matches που) . Το κόστος μπορεί εύκολα να ενσωματωθεί στις ενισχύσεις όπως παρακάτω.

$$r\left(s_t, \triangleright\triangleleft_i\right) = \begin{cases} -\varepsilon \cdot cost, s_{t+1} = null \\ -m \cdot cost, s_{t+1} \neq null \end{cases}$$

$$r\left(R_1 R_2 \cdots R_n, output\right) = 0$$

$$r\left(null, get\left(R\right)\right) = 0 \text{ ή } r\left(null, get\left(R\right)\right) = -\varepsilon \cdot cost$$

5.5 Άρση του «βάρους της ιστορίας»

Η διαφορά της εκτέλεσης ενός ερωτήματος με STAIRs αντί δυαδικών Symmetric Hash Joins είναι ότι ο eddy έχει στην διάθεσή του δύο πράξεις χειρισμού της κατάστασης των ήδη αποθηκευμένων αποτελεσμάτων, τις πράξεις $demotion\left(STAIR, t, t'\right)$, η οποία εκφυλίζει μια πλειάδα σε μια singleton υπο – πλειάδα της και $promotion\left(STAIR_1, t, STAIR_2\right)$, η οποία μεταφέρει μια πλειάδα από ένα STAIR σε ένα άλλο, την συνδέει με το δυαδικό του

καινούριου STAIR και αποθηκεύει τα matches στο αρχικό STAIR. Μπορούμε να αφήσουμε τον eddy να χρησιμοποιεί τις πράξεις αυτές κατά βούληση, χωρίς να υπάρχουν λάθη στο αποτέλεσμα του ερωτήματος ([DH04]). Εδώ θα περιοριστούμε στο να χρησιμοποιήσουμε τις πράξεις αυτές για να αλλάξουμε την κατάσταση των συνδέσμων, δηλαδή να μεταφέρουμε τα αποθηκευμένα tuples του παρελθόντος από ένα σύνδεσμο σε αυτόν που ανταποκρίνεται στην τωρινή πολιτική δρομολόγησης. Θα ονομάσουμε την πράξη αυτή Migrate και θα την ορίσουμε ως

$$Migrate\left(\triangleright\triangleleft_i \mapsto \triangleright\triangleleft_{i+1}\right) \equiv demotion\left(\triangleright\triangleleft_{i+1}.STAIR_1, T, T'\right)$$
$$promotion\left(\triangleright\triangleleft_{i+1}.STAIR_1, T', \triangleright\triangleleft_i.STAIR_2\right)$$

όπου οι πλειάδες Τ ανήκουν στη σχέση $R_i R_{i+1}$ και οι Τ' στη σχέση R_{i+1} ή

$$Migrate\left(\triangleright\triangleleft_{i+1} \mapsto \triangleright\triangleleft_i\right) \equiv demotion\left(\triangleright\triangleleft_i.STAIR_2, T, T'\right)$$
$$promotion\left(\triangleright\triangleleft_i.STAIR_2, T', \triangleright\triangleleft_{i+1}.STAIR_1\right)$$

όπου οι πλειάδες Τ ανήκουν στη σχέση $R_i R_{i+1}$ και οι Τ' στη σχέση R_i

Ο eddy θα πρέπει να έχει πληροφορία για το πού είναι αποθηκευμένες οι singleton πλειάδες (δηλαδή στο STAIR ποιού συνδέσμου), ώστε να είναι σε θέση να πάρει την απόφαση για μεταφορά τους. Θα το εκφράσουμε αυτό δίνοντας του το σήμα

$$s = \left[\left(R_1, \triangleright\triangleleft_{i_1}\right), \left(R_2, \triangleright\triangleleft_{i_2}\right), \ldots, \left(R_n, \triangleright\triangleleft_{i_n}\right)\right]$$
$$i_k \in \{k-1, k\} \mid k \in \{1, \ldots, n-1\}, i_1 = 1, i_n = n-1$$

Το ζεύγος $\left(R_k, \triangleright\triangleleft_{i_k}\right)$ δεν σημαίνει ότι όλες οι πλειάδες της R_k είναι αποθηκευμένες στο αντίστοιχο STAIR του $\triangleright\triangleleft_{i_k}$ αλλά ότι το τελευταίο migration είχε γίνει με το $\triangleright\triangleleft_{i_k}$ ως στόχο. Η παραπάνω κατάσταση πρέπει να αρχικοποιείται σύμφωνα με τις αρχικές επιλογές δρομολόγησης του eddy, και αλλάζει με τις εφαρμογές της πράξης Migrate, δηλαδή σε κάθε βήμα ο eddy έχει να αποφασίσει ανάμεσα στις πράξεις

$$A(s) = \left\{Migrate\left(\triangleright\triangleleft_i \mapsto \triangleright\triangleleft_{i+1}\right) \vee Migrate\left(\triangleright\triangleleft_{i+1} \mapsto \triangleright\triangleleft_i\right) \mid i = 1, \ldots, n-2\right\} \cup \{nothing\}$$

και η πράξη πχ $Migrate\left(\triangleright\triangleleft_1 \mapsto \triangleright\triangleleft_2\right)$ αλλάζει το ζεύγος $\left(R_2, \triangleright\triangleleft_1\right)$ σε $\left(R_2, \triangleright\triangleleft_2\right)$. Επειδή το κόστος μιας τέτοιας λειτουργίας μπορεί να είναι μεγάλο, πρέπει να εφαρμόζεται με προσοχή, δηλαδή το κόστος της να μπαίνει στην συνάρτηση ενίσχυσης, και η πράξη nothing να έχει και αυτή μια σχετικά μεγάλη ενίσχυση. Ένα σήμα ενίσχυσης που θα μπορούσε να οριστεί είναι

$$r\left(\left[\ldots,\left(R_i,\triangleright\triangleleft_{i-1}\right),\ldots\right],Migration\left(\triangleright\triangleleft_{i-1}\mapsto\triangleright\triangleleft_i\right)\right)=$$
$$-cost\cdot\left[Q\left(Route\left(R_i,\triangleright\triangleleft_i\right)\right)-Q\left(Route\left(R_i,\triangleright\triangleleft_{i-1}\right)\right)\right]$$

$$r\left(\left[\ldots,\left(R_i,\triangleright\triangleleft_{i-1}\right),\ldots\right],nothing\right)=0$$

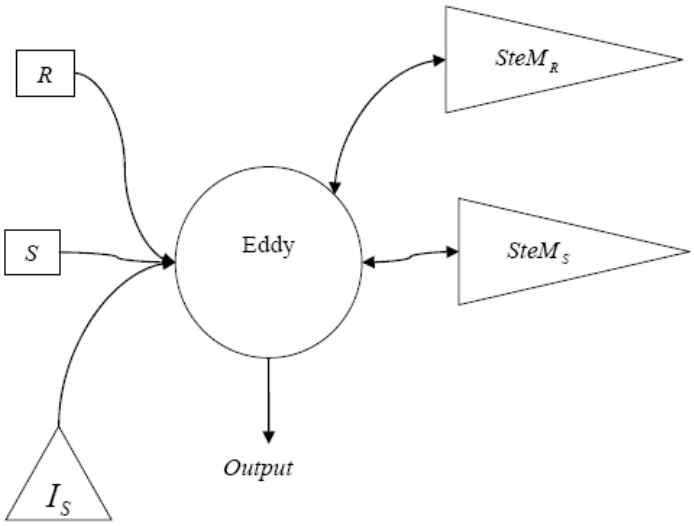
Όπως φαίνεται, μπορεί τα δύο προβλήματα μάθησης (εύρεση της σειράς των συνδέσμων και άρση του βάρους της ιστορίας) να τρέχουν ανεξάρτητα ακόμα και σε διαφορετικές κλίμακες χρόνου αλλά η απόφαση για state migration επηρεάζεται σαφώς από τη γνώση που έχουμε ως προς την τιμή των συνδέσμων (την μελλοντική ενίσχυση).

5.6 Μάθηση των αλγορίθμων συνδέσμων και ανταγωνισμός πολλαπλών AMs

Επεκτεινόμαστε στην παρούσα ενότητα στις αρχιτεκτονικές EddySteMs1, EddySteMs2 και EddySteMs3. Οι περιορισμοί που ισχύουν είναι αυτοί του πίνακα 3.5, από τους οποίους την πολιτική δρομολόγησης ενδιαφέρουν μόνο οι BoundedRepetition και BuildFirst. Οι υπόλοιποι περιορισμοί είναι θέματα υλοποίησης των ίδιων των SteMs, και στα παρακάτω υποθέτουμε ότι τα SteMs έχουν υλοποιηθεί σύμφωνα με αυτούς. Ο περιορισμός Atomicity που απλοποιούσε την εκτέλεση ενός ερωτήματος σε μια προσομοίωση ενός n – way SHJ δεν ισχύει πια, οπότε τα builds και τα probes αντιμετωπίζονται ως ξεχωριστές δράσεις. Λόγω του περιορισμού BuildFirst, ο eddy πρέπει να είναι σε θέση να ξέρει αν μια singleton πλειάδα είναι καινούρια στο ή έχει ήδη εισαχθεί στο αντίστοιχο SteM. Πρέπει λοιπόν να προστεθεί στο σήμα κατάστασης ένα bit *isNew* το οποίο τίθεται 1 αρχικά και 0 όταν η πλειάδα εισαχθεί στο αντίστοιχο SteM, δηλαδή. Και στην περίπτωση αυτή βέβαια, το σήμα κατάστασης μπορεί να προκύψει πλήρως από τα ready και done bits.

$$S=\left(\left\{\text{σχέσεις της τρέχουσας πλειάδας}\right\}\times\left\{true,false\right\}\right)\cup\left\{null\right\}$$

Το πρόβλημα του ανταγωνισμού εναλλακτικών scans ή indices μπορεί εύκολα να λυθεί εισάγοντας τις δράσεις $probe\left(AM_S\right)$ για τα indices όπου αυτές είναι επιτρεπτές και ορίζοντας τις κατάλληλες ενισχύσεις. Το πρόβλημα της μάθησης του βέλτιστου αλγόριθμου συνδέσμου είναι λίγο πιο δύσκολο. Όπως έχει αναφερθεί, ο eddy μπορεί να «προσομοιώνει» τον αλγόριθμο συνδέσμου μεταξύ δύο σχέσεων με την πολιτική δρομολόγησης. Για παράδειγμα, έστω το ερώτημα $R\triangleright\triangleleft S$ με την S να διαθέτει ένα ευρετήριο, το οποίο αρχικοποιείται προς εκτέλεση όπως στο παρακάτω σχήμα



Σχήμα 5.7

Ο eddy μπορεί να αποφασίσει να μην χρησιμοποιήσει το ευρετήριο προσομοιώνοντας ένα αλγόριθμο SHJ, εκτελώντας το πλάνο

$$\left(get\left(R\right)build\left(SteM_R\right)probe\left(SteM_S\right)\middle|get\left(S\right)build\left(SteM_S\right)probe\left(SteM_R\right)\right)^*$$

ή ακόμα και να εκτελέσει ένα non – pipelined αλγόριθμο Hybrid Hash Join εκτελώντας το πλάνο

$$\left(get\left(R\right)build\left(SteM_R\right)\right)^*\left(get\left(S\right)build\left(SteM_S\right)\right)^*probe\left(SteM_R\right)$$

Μπορεί όμως να χρησιμοποιήσει το ευρετήριο υλοποιώντας ένα Index Join

$$\left(get\left(R\right)build\left(SteM_R\right)probe\left(I_S\right)\right)^*\left|\left(build\left(SteM_S\right)probe\left(SteM_R\right)\right)^*\right.$$

Όπως είναι φανερό, ο eddy μπορεί να αλλάξει δυναμικά τον αλγόριθμο συνδέσμου πχ από SHJ σε HHJ επιλέγοντας να μην προωθήσει μια R πλειάδα στο $SteM_S$ αλλά να φέρει μια καινούρια R πλειάδα. Μπορούμε να μοντελοποιήσουμε αυτή την αλλαγή επιτρέποντας στις δράσεις $get\left(R_i\right)$ να εκτελούνται σε κάθε κατάσταση και όχι μόνο στην κατάσταση $null$. Η αλλαγή από ένα hash join σε ένα index join γίνεται μέσω του ανταγωνισμού των probes σε SteMs ή ευρετήρια. Το καινούριο σύνολο καταστάσεων είναι

$$A=\left\{build\left(SteM_{R_i}\right)\middle|\,i=1,\ldots,n\right\}\cup\left\{probe\left(SteM_{R_i}\right)\middle|\,i=1,\ldots,n\right\}\cup$$
$$\left\{build\left(AM_{R_i}\right)\middle|\,i=1,\ldots,n\right\}\cup\left\{get\left(R_i\right)\middle|\,i=1,\ldots,n\right\}$$

και οι περιορισμοί δρομολόγησης εκφράζονται μέσω των συνόλων $A\left(s\right)$ όπως παρακάτω

$$A(null) = \{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

$$A((R_i, true)) = \{build(SteM_{R_i})\} \qquad [\text{περιορισμός BuildFirst}]$$

$$A((R_i, false)) = \left\{ \begin{array}{l} probe(SteM_{R_j}) \mid j \neq i \wedge \\ \text{το } R_i \rhd\lhd R_j \text{ δεν είναι καρτεσιανό γινόμενο} \end{array} \right\} \cup$$

$$\left\{ \begin{array}{l} probe(AM_{R_j}) \mid j \neq i \wedge \\ \text{οι } R_i, R_j \text{ έχουν κοινή ιδιότητα} \end{array} \right\} \cup$$

$$\{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

$$A((R_{i_1} R_{i_2} \cdots R_{i_k}, false)) = \left\{ \begin{array}{l} probe(SteM_{R_j}) \mid j \notin \{i_1, \ldots, i_k\} \wedge \\ \text{το } R_{i_1} R_{i_2} \cdots R_{i_k} \rhd\lhd R_j \text{ δεν είναι καρτεσιανό γινόμενο} \end{array} \right\} \cup$$

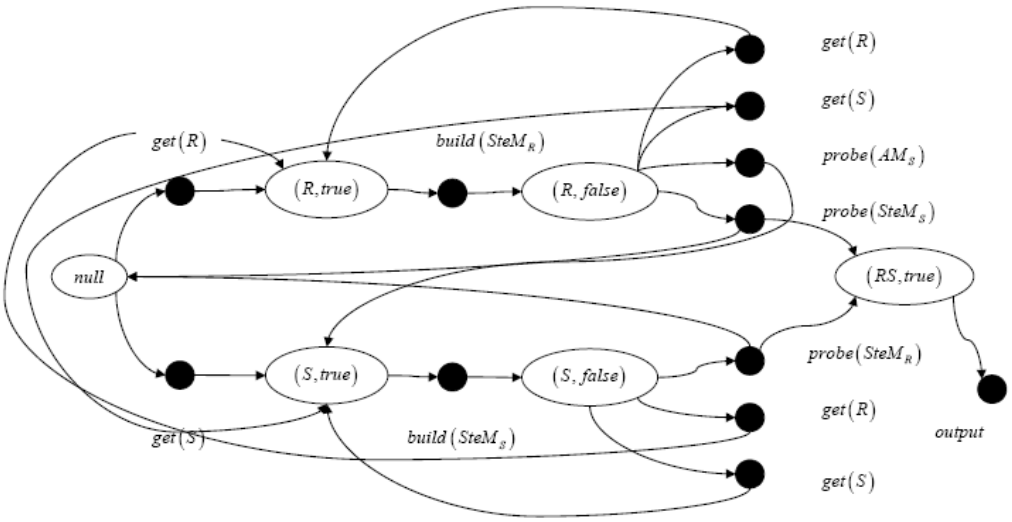
$$\left\{ \begin{array}{l} probe(AM_{R_j}) \mid j \notin \{i_1, \ldots, i_k\} \wedge \\ \text{οι } R_{i_1} R_{i_2} \cdots R_{i_k}, R_j \text{ έχουν κοινή ιδιότητα} \end{array} \right\} \cup$$

$$\{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

$$A((R_1 R_2 \cdots R_n, false)) = \{output\} \cup$$

$$\{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

Παρακάτω φαίνεται η διαδικασία Markov που δημιουργείται για το προηγούμενο παράδειγμα. Όπως φαίνεται, πια ο eddy μπορεί να προσομοιώσει και να μάθει όλους τους αλγόριθμους συνδέσμων.



Σχήμα 5.8

Για να μπορεί ο eddy να συγκρίνει αλγόριθμους συνδέσμων, πρέπει υποχρεωτικά πια οι ενισχύσεις των δράσεων να περιέχουν το κόστος. Έτσι, έχουμε

$$r\left(s, build\left(SteM_T\right)\right) = -\varepsilon_{build} \cdot cost$$

$$r\left(s, probe\left(SteM_T\right)\right) = \begin{cases} -\varepsilon_{probe} \cdot cost, & s' = null \\ -m \cdot cost, & s' \neq null \wedge m \text{ matches returned} \end{cases}$$

$$r\left(s, probe\left(AM_T\right)\right) = \begin{cases} -\varepsilon_{index} \cdot cost, & s' = null \\ -m \cdot cost, & s' \neq null \wedge m \text{ matches returned} \end{cases}$$

$$p\left(s, get\left(R\right)\right) = -\varepsilon_{get} \cdot cost$$

$$p\left(s, output\right) = 0$$

Τα βάρη ε των ενισχύσεων μπορούν να τεθούν ίσα ώστε ο ανταγωνισμός να είναι δίκαιος ή να ευνοούν κάποια πράξη.

5.7 Κυκλικά ερωτήματα και χαλάρωση του BuildFirst

Για την εκτέλεση γενικών ερωτημάτων με την αρχιτεκτονική EddySteMs5 αρκεί να ενσωματωθούν οι περιορισμοί δρομολόγησης του πίνακα 3.7 στα σύνολα $A(s)$ καθώς και η παραπάνω πληροφορία που χρειάζεται ο eddy στο σύνολο των καταστάσεων S. Το σήμα κατάστασης γίνεται

$$s = \begin{bmatrix} \text{tablesSpanned} \\ \text{isNew} \\ \text{hasMultipleAMs} \\ \text{hasIndexAM} \\ \text{isPriorProber} \\ \text{probeCompletionTable} \\ \text{probeCompletionAMs} \end{bmatrix}$$

με τους περιορισμούς όμως

$$isNew \Rightarrow \neg isPriorProber$$
$$\neg isPriorProber \Rightarrow probeCompletionTable = null$$
$$\neg isPriorProber \Rightarrow probeCompletionAMs = null$$

Το παραπάνω σήμα κατάστασης δεν μπορεί να προκύψει άμεσα από τα ready και done bits της πλειάδας, αλλά στον tuple descriptor πρέπει να προστεθούν τα πεδία 3 – 7. Το σύνολο των καταστάσεων είναι το

$$S = \begin{pmatrix} \{\text{tables}\} \times \{true, false\} \times \{true, false\} \times \{true, false\} \times \\ \{true, false\} \times \{\text{table}\} \times \{\text{Access Methods}\} \end{pmatrix} \cup \{null\}$$

Ο χώρος των καταστάσεων έχει πια μέγεθος $|S| = n^2 + 3n = O(n^2)$ ενώ ο χώρος των δράσεων έχει μέγεθος επίσης $|A| = O(n^2)$. Οι περιορισμοί δρομολόγησης εκφράζονται όπως παρακάτω, υποθέτοντας ότι μόλις μια πλειάδα είναι έτοιμη για έξοδο το σύνολο probeCompletionAMs γίνεται null (εναλλακτικά μπορούμε να χρησιμοποιήσουμε ένα ακόμα bit στο σήμα κατάστασης)

$$A(null) = \{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

$$A((R_i, true, true, \_, false, null, null)) = \{build(SteM_{R_i})\} \qquad [\text{περιορισμός BuildFirst}]$$

$$A((R_i, true, \_, true, false, null, null)) = \{build(SteM_{R_i})\} \qquad [\text{περιορισμός BuildFirst}]$$

$$A((R_i, true, false, false, false, null, null)) = \{build(SteM_{R_i})\} \cup \{probe(SteM_{R_j}) \mid j \neq i \wedge \text{το } R_i \rhd\lhd R_j \text{ δεν είναι καρτεσιανό γινόμενο}\} \cup \{probe(AM_{R_j}) \mid j \neq i \wedge \text{οι } R_i, R_j \text{ έχουν κοινή ιδιότητα}\} \cup \{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

$$A((R_i, false, \_, \_, false, \_, \_)) = \{probe(SteM_{R_j}) \mid j \neq i \wedge \text{το } R_i \rhd\lhd R_j \text{ δεν είναι καρτεσιανό γινόμενο}\} \cup \{probe(AM_{R_j}) \mid j \neq i \wedge \text{οι } R_i, R_j \text{ έχουν κοινή ιδιότητα}\} \cup \{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

$$A((R_{i_1} R_{i_2} \cdots R_{i_k}, false, \_, \_, false, \_, \_)) = \{probe(SteM_{R_j}) \mid j \notin \{i_1, \ldots, i_k\} \wedge \text{το } R_{i_1} R_{i_2} \cdots R_{i_k} \rhd\lhd R_j \text{ δεν είναι καρτεσιανό γινόμενο}\} \cup \{probe(AM_{R_j}) \mid j \notin \{i_1, \ldots, i_k\} \wedge \text{οι } R_{i_1} R_{i_2} \cdots R_{i_k}, R_j \text{ έχουν κοινή ιδιότητα}\} \cup \{get(R_j) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\}$$

$$A\left(\left(R_1 R_2 \cdots R_n, false, \_, \_, false, \_, \_\right)\right) = \{output\} \cup$$
$$\left\{get\left(R_j\right) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\right\}$$

$$A\left(\left(R_i, false, \_, \_, true, PCT, \_\right)\right) = \left\{probe\left(SteM_{PCT}\right)\right\} \cup$$
$$\left\{\begin{array}{l} probe\left(AM_{R_j}\right) \mid j \neq i \wedge \\ \text{οι } R_i, R_j \text{ έχουν κοινή ιδιότητα} \end{array}\right\} \cup$$
$$\left\{get\left(R_j\right) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\right\}$$

$$A\left(\left(R_{i_1} R_{i_2} \cdots R_{i_k}, false, \_, \_, true, PCT, \_\right)\right) = \left\{probe\left(SteM_{PCT}\right)\right\} \cup$$
$$\left\{\begin{array}{l} probe\left(AM_{R_j}\right) \mid j \notin \{i_1, \ldots, i_k\} \wedge \\ \text{οι } R_{i_1} R_{i_2} \cdots R_{i_k}, R_j \text{ έχουν κοινή ιδιότητα} \end{array}\right\} \cup$$
$$\left\{get\left(R_j\right) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\right\}$$

$$A\left(\left(R_1 R_2 \cdots R_n, false, \_, \_, true, \_, CAMs\right)\right) = \left\{probe\left(AM_{R_j}\right) \mid AM_{R_j} \in CAMs\right\} \cup$$
$$\left\{get\left(R_j\right) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\right\}$$

$$A\left(\left(R_1 R_2 \cdots R_n, false, \_, \_, true, \_, null\right)\right) = \{output\} \cup$$
$$\left\{get\left(R_j\right) \mid \text{η } R_j \text{ έχει ακόμα πλειάδες}\right\}$$

6

Αλγόριθμοι Ενισχυτικής Μάθησης για Βελτιστοποίηση Ερωτημάτων

Στο παρόν κεφάλαιο προσπαθούμε να δώσουμε μια ενιαία εικόνα της διαδικασίας της βελτιστοποίησης ενός ερωτήματος ως λύση προβλημάτων ενισχυτικής μάθησης χρησιμοποιώντας τη μοντελοποίηση του κεφαλαίου 5. Αρχίζουμε με μια αρχιτεκτονική μόνο με επιλογές, και προχωρούμε στην περιγραφή της βελτιστοποίησης στις τρεις γενικές αρχιτεκτονικές που έχουν προταθεί για τα eddies – δυαδικοί σύνδεσμοι, STAIRs και SteMs.

6.1 Εκτέλεση ερωτημάτων επιλογής

Περιοριζόμαστε στην περίπτωση του clustering με $X = \{x_0\}$. Παρακάτω φαίνεται ένας γενικός (generic) αλγόριθμος στο πνεύμα του Q – learning που λύνει το πρόβλημα εύρεσης της σειράς των επιλογών. Ένας Monte Carlo αλγόριθμος απλά θα έκανε την ανανέωση των τιμών (βήμα $Q(i) \leftarrow update(Q(i), r)$) μετά τη λήξη του επεισοδίου. Υποθέτουμε ένα integrator – based περιβάλλον όπως αυτό που περιγράφεται στην ενότητα 3.10.

Αλγόριθμος 6.1 : EddySelections learning

1. Αρχικοποίηση

$s', r, s, a, Q[i], i = 0, \ldots, n+1$

2. Eddy.*get _ next*()

$s \leftarrow -1$

repeat

$i \leftarrow somehow\left(Q, actions\left(s\right)\right)$

$take\left(i\right)$

if $\left(data\_is\_over\right)$ return *null*

$s' \leftarrow next\_state$

$r \leftarrow reward$

$Q\left(i\right) \leftarrow update\left(Q\left(i\right), r\right)$

$s \leftarrow s'$

$t \leftarrow t+1$

until $\left(s = n-1\right)$

return t

Ο eddy κρατά εκτός από τα done bits κάθε πλειάδας, και μια δομή $Q\left(a\right)$ που αντιστοιχίζει κάθε δράση στην τιμή της. Η υλοποίηση της δομής αυτής μπορεί να γίνεται με ένα πίνακα, οπότε χρειαζόμαστε μια συνάρτηση $i : A \to \mathbb{N}$. Έστω για παράδειγμα η παρακάτω αντιστοίχιση.

$i\left(get\left(R\right)\right) = 0$

$i\left(\sigma_i\right) = i \mid i = 1, \ldots, n$

$i\left(output\right) = n+1$

Φυσικά, η υλοποίηση της δομής μπορεί να γίνει με πιο αποδοτικό τρόπο (πχ hash map). Επίσης χρειαζόμαστε μια αναπαράσταση για τις καταστάσεις. Πάλι μια αναπαράσταση ακεραίου αρκεί, ενώ πιο εξεζητημένες λύσεις είναι δυνατές.

$null \mapsto -1$

$s_i \mapsto i, i = 0, \ldots, n-1$

Η ρουτίνα *take* εκτελεί την δράση που παίρνει σαν όρισμα και θέτει τις μεταβλητές $next\_state$ και *reward*, δηλαδή υλοποιεί ουσιαστικά το περιβάλλον. Σε ένα σύγχρονο σύστημα θα ήταν κάτι σαν τον παρακάτω ψευδοκώδικα

Αλγόριθμος 6.2 : take

είσοδος : i

if $(i = 0)$ then

  if $(t \leftarrow Scan(R).get\_next() = null)$ then

    $data\_is\_over \leftarrow true$

    $next\_state \leftarrow -1$

    $reward \leftarrow reward(0, \_)$

  else

    $next\_state = t.donebits$

    $reward \leftarrow reward(0, \_)$

else if $(i = n+1)$

  $next\_state \leftarrow -1$

  $reward \leftarrow reward(n+1, \_)$

else

  if $(t \leftarrow Selection(i).get\_next() = null)$

    $next\_state \leftarrow -1$

    $reward \leftarrow reward(i, false)$

  else

    $next\_state \leftarrow turn\ on\ bit\ i\ of\ t$

    $reward \leftarrow reward(i, true)$

Η συνάρτηση $reward$ υλοποιεί την συνάρτηση ενισχύσεων. Για τις περιπτώσεις που αναφέρονται στην ενότητα 5.3 έχουμε.

$$r(n+1, \_) = 0$$

$$(1)\begin{vmatrix} r(0,\_) = 0 \\ r(i,true) = -1 \\ r(i,false) = 0 \end{vmatrix} \quad (2)\begin{vmatrix} r(0,\_) = -\varepsilon \cdot cost \\ r(i,true) = -cost \\ r(i,false) = \varepsilon \cdot cost \end{vmatrix}$$

Η συνάρτηση $actions(s)$ επιστρέφει απλά τις νόμιμες δράσεις για την κατάσταση που παίρνει σαν όρισμα όπως αυτές ορίζονται στην παράγραφο 5.3. Οι διαφοροποίηση των αλγόριθμων γίνεται με τον ορισμό των ρουτίνων *somehow* που αντανακλά την προσέγγιση στο δίλλημα εκμετάλλευσης – εξερεύνησης και *update* που αντανακλά τη φύση στης μάθησης (μάθηση από εμπειρία ή από μοντέλο κλπ). Μπορούμε να πάρουμε την μέθοδο Lottery Scheduling από τον παραπάνω γενικό αλγόριθμο επιλέγοντας την πρώτη συνάρτηση *reward* και θέτοντας.

$$somehow\left(Q, A(s)\right) = \underset{a \in A(s)}{uniform}\left(Q(a)\right) \Leftrightarrow \Pr\{select\ a\} = \frac{Q(a)}{\sum_{b \in A(s)} Q(b)}$$ και

$$update\left(Q(i), r\right) = Q(i) + r$$

δηλαδή αθροίζοντας απλά τις τιμές των ενισχύσεων για κάθε δράση και επιλέγοντας τις δράσεις με ομοιόμορφη πιθανότητα. Πολλές βελτιώσεις μπορούν να προταθούν στο παραπάνω σχήμα.

Στη διάσταση της εξερεύνησης, η ομοιόμορφη πιθανότητα ίσως εξερευνά πάρα πολύ το χώρο, ενώ μια προσέγγιση $\varepsilon - greedy$ θα ήταν καλύτερη για συστήματα στα οποία υπάρχει καλή αρχική γνώση για τις επιλεκτικότητες, οι οποίες όμως ίσως αλλάζουν στο χρόνο εκτέλεσης.

$$somehow\left(Q, A(s)\right) = \underset{a \in A(s)}{\varepsilon - greedy}\left(Q(a)\right) \Leftrightarrow \Pr\{select\ a\} = \begin{cases} 1-\varepsilon, a = \arg\max_{b \in A(s)}\left(Q(b)\right) \\ \varepsilon, a \neq \arg\max_{b \in A(s)}\left(Q(b)\right) \end{cases}$$

Τέλος, υποσχόμενη φαντάζει η πιθανότητα της Προσομοιωμένης Ανόπτησης αφού έχει εφαρμοστεί με επιτυχία σε πολλά προβλήματα συμπεριλαμβανομένης της βελτιστοποίησης ερωτημάτων ([IW87], [IK90]).

$$somehow\left(Q, A(s)\right) = \underset{a \in A(s)}{annealing}\left(Q(a)\right) \Leftrightarrow \Pr\{select\ a\} = \frac{e^{Q(s,a)/T}}{\sum_{b \in A(s)} e^{Q(s,b)/T}}$$

Για τη διαδικασία ψύξης της θερμοκρασίας έχουν προταθεί διάφορες προσεγγίσεις. Μια εύκολα υλοποιήσιμη και ισχυρή μέθοδος [IW87] είναι $T_{new} = f\left(T_{old}\right)T_{old}$ όπου f είναι μια συνάρτηση που παίρνει τιμές στο $[0,1]$ και είναι φθίνει με το χρόνο.

Στη διάσταση της ανανέωσης των τιμών, μια πρώτη προσέγγιση είναι η αποθήκευση των μέσων ενισχύσεων, η οποία συγκλίνει στις επιλεκτικότητες των επιλογών αλλά είναι ισοδύναμη με την κράτηση αθροισμάτων.

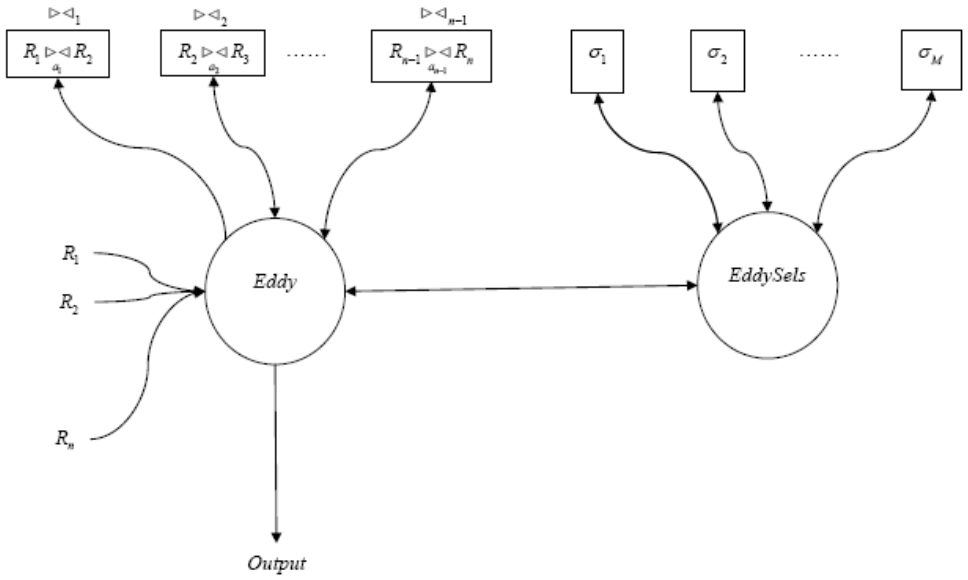
$$update\left(Q(i), r\right) = Q(i) + \frac{1}{t+1}\left(r - Q(i)\right)$$

Θα μπορούσαμε τέλος να κρατάμε ένα μέσο όρο με βάρη, δίνοντας περισσότερη σημασία στις άμεσες ενισχύσεις.

$$update\left(Q(i), r\right) = Q(i) + a\left(r - Q(i)\right)$$

6.2 Εκτέλεση γενικών ερωτημάτων με επιλογές και τελεστές δυαδικών συνδέσμων

Στην ενότητα αυτή ακολουθώντας την σύγχρονη αρχιτεκτονική της ενότητας 3.10 και το μοντέλο επεξεργασίας EddyJoins δείχνουμε πώς μπορεί να γίνει η επεξεργασία ενός γενικού ερωτήματος με επιλογές και συνδέσμους. Όπως έχει προαναφερθεί, θα διαχωρίσουμε τα προβλήματα της σειράς των συνδέσμων και της σειράς των επιλογών, χρησιμοποιώντας το δημοφιλές ευριστικό τέχνασμα ώθησης των επιλογών στα φύλλα του δέντρου. Μοντελοποιούμε την τεχνική αυτή αναγκάζοντας κάθε singleton πλειάδα να περνά πρώτα από όλα τα κατηγορήματα και μετά από τους συνδέσμους, όπως παρακάτω. Βέβαια, δεν χρειάζεται να υλοποιήσουμε δύο eddies για την εκτέλεση του ερωτήματος αφού ο παραπάνω περιορισμός μπορεί εύκολα να ενσωματωθεί σε ένα πραγματικό σύστημα, απλά η περιγραφή που απεικονίζεται διευκολύνει την ανάλυση.



Σχήμα 6.1

Ένας γενικός αλγόριθμος στη φιλοσοφία του Q – learning που λύνει το πρόβλημα φαίνεται παρακάτω. Ο EddySels υλοποιεί ένας αλγόριθμο μάθησης επιλογών στο πνεύμα του αλγόριθμου 6.1 με τις παρακάτω τροποποιήσεις.

- Δεν ζητά πλειάδες από κάποια σχέση αλλά δέχεται μια πλειάδα ως είσοδο.
- Αφού μπορεί πια να δέχεται πλειάδες διαφόρων σχέσεων, πρέπει να κρατά δύο πίνακες ready bits και done bits. Μόλις μια πλειάδα εισέρχεται στον EddySels, αυτός θέτει τα ready bits ανάλογα με τις επιλογές που αναφέρονται στη σχέση της πλειάδας. Τα done bits αρχικά είναι $donebits = \neg readybits$.

Οι περισσότεροι αλγόριθμοι μάθησης χρησιμοποιούν την συνάρτηση τιμών $Q(s,a)$, οπότε χρειάζεται ένας τρόπος κωδικοποίησης των καταστάσεων και των δράσεων. Ασχολούμαστε πάλι με την απλή περίπτωση της πινακοειδούς αναπαράστασης της συνάρτησης τιμών. Η κωδικοποίηση των δράσεων μπορεί να γίνει με τον ίδιο τρόπο που παρουσιάστηκε στην προηγούμενη ενότητα.

$$j\left(get\left(R_i\right)\right) = -1$$
$$j\left(\sigma_i\right) = i \mid i = 1,\ldots,n$$
$$j\left(output\right) = 0$$

ενώ η κατάσταση μπορεί να είναι ο δυαδικός αριθμός n – bits που προκύπτει αν θέσουμε 1 όλα τα bits που αντιστοιχούν σε σχέσεις που εμφανίζονται στην πλειάδα και 0 τα bits που αντιστοιχούν σε σχέσεις που δεν εμφανίζονται στην πλειάδα. Πιο συμπαγείς τρόποι αναπαράστασης είναι δυνατοί, σε κάθε περίπτωση πάντως έχουμε μια συνάρτηση $i : S \to \mathbb{N}$ οπότε η $Q(s,a)$ μπορεί να αναπαρασταθεί με ένα πίνακα $Q\left[i(s), j(a)\right]$. Κατά το μοντέλο της ενότητας 3.10, ο eddy διατηρεί μια στοίβα «ενεργών» τελεστών την οποία ονομάζουμε active. Παρακάτω φαίνεται σε ψευδοκώδικα πώς μια διαδικασία μάθησης τύπου temporal difference μπορεί να ενσωματωθεί στην επεξεργασία των πλειάδων.

Αλγόριθμος 6.3 : EddyJoins learning

1. Αρχικοποίηση

   $s', r, s, a, Q[i, j]$

2. Eddy.$get\_next()$

   2a. if $(active \neq empty)$

   $a \leftarrow top(active)$

   $take(a)$

   if $(next\_state = null)$ $pop(top(active))$; $goto$ 2a

   else

   $s \leftarrow null$;

   $a \leftarrow somehow(Q, A(s))$

   if $(data\_is\_over)$ return $null$

   $s \leftarrow s'$

   if $(\text{EddySels}(t) = null)$ $s \leftarrow null$; $goto$ 2a

   $a \leftarrow somehow(Q, A(s))$

   $take(a)$

   $s' \leftarrow next\_state$

   $r \leftarrow reward$

   $Q \leftarrow update(Q, A(s))$

   if $(a = output)$ return t else $push(active, a)$

ενώ περιμένουμε από την $take$ να λειτουργεί όπως παρακάτω

Αλγόριθμος 6.4 : take

είσοδος : a

if $\left(a = get\left(R_i\right)\right)$ then

  if $\left(t \leftarrow Scan(R).get\_next() = null\right)$ then

    $data\_is\_over_i \leftarrow true$

    $next\_state \leftarrow null$

    $reward \leftarrow reward\left(get\left(R_i\right), false\right)$

  else

    $next\_state = t.schema$

    $reward \leftarrow reward\left(get\left(R_i\right), true\right)$

else if $\left(i = output\right)$

  $next\_state \leftarrow null$

  $reward \leftarrow reward\left(output, \_\right)$

else

  if $\left(t \leftarrow \triangleright\triangleleft_i .get\_next() = null\right)$

    $next\_state \leftarrow null$

    $reward \leftarrow reward\left(\triangleright\triangleleft_i, false\right)$

  else

    $next\_state \leftarrow matches.schema$

    $reward \leftarrow reward\left(\triangleright\triangleleft_i, true\right)$

Η ενίσχυση μπορεί να είναι μία από αυτές που αναφέρονται στην ενότητα 5.4.1 με τη διαφορά ότι αφού τα matches επιστρέφονται ένα τη φορά, αρκεί κάθε φορά να επιστρέφεται μια ενίσχυση -1 και όχι –m:

$r\left(\triangleright\triangleleft, false\right) = -\varepsilon \cdot cost$

$r\left(\triangleright\triangleleft, false\right) = -cost$

ή

$r\left(\triangleright\triangleleft, false\right) = 0$

$r\left(\triangleright\triangleleft, false\right) = -1$

Η συνάρτηση των μελλοντικών μπορεί να είναι απλά αθροιστική $\left(\gamma = 1\right)$ ή με βάρη. Η φύση της μάθησης καθορίζεται και πάλι από την ρουτίνα $Q \leftarrow update\left(Q, A\left(s\right)\right)$ και ο βαθμός εξερεύνησης από την $a \leftarrow somehow\left(Q, A\left(s\right)\right)$. Για τον βαθμό εξερεύνησης μπορεί να

χρησιμοποιηθεί οποιαδήποτε μέθοδος της ενότητας 4.6, ενώ για τον τρόπο μάθησης μπορεί να εφαρμοστεί οποιαδήποτε μέθοδος TD ή MC. Η διαφορά που θα παρουσίαζε μια μάθηση MC στον παραπάνω ψευδοκώδικα είναι ότι θα χρησιμοποιούσε κάποιο βοηθητικό πίνακα για την ανανέωση των τιμών Q και θα αποθήκευε στον πραγματικό πίνακα μόνο στο τέλος του επεισοδίου. Για παράδειγμα, από τον παραπάνω γενικό αλγόριθμο μπορούμε να πάρουμε τον Q – learning με

$$update\left(Q[s,a],A(s)\right)\leftarrow Q[s,a]+\eta\left\{r+\gamma\max_{a'}\left(Q[s',a']-Q[s,a]\right)\right\}$$

μόνο για την δράση που επιλέχθηκε, ενώ θα μπορούσαμε να χρησιμοποιήσουμε eligibility traces αν

$update(Q,A)=$
  for all (s,a)
    $Q(s,a)=Q(s,a)+\gamma\lambda e(s,a)$
    $e(s,a)=\gamma\lambda e(s,a)$

και προσθέταμε τον κώδικα αύξησης του eligibility trace της δράσης που επιλέχθηκε.

6.3 Εκτέλεση γενικών ερωτημάτων με επιλογές και τελεστές STAIRs

Η εκτέλεση ερωτημάτων επιλογών και συνδέσμων γίνεται με τον ίδιο τρόπο με την αρχιτεκτονική EddyJoins που περιγράφηκε στην προηγούμενη ενότητα, με την προσθήκη της δυνατότητας αλλαγής της κατάστασης. Το πρόβλημα της άρσης του «βάρους της ιστορίας» μπορεί όμως να διαχωριστεί πλήρως από αυτά της εύρεσης της σειράς των συνδέσμων και των επιλογών, και να αντιμετωπιστεί ως ένα αυτόνομο πρόβλημα ενισχυτικής μάθησης. Βέβαια, για τη λύση του χρησιμοποιούνται οι συναρτήσεις τιμών που κρατά ο eddy για τους τελεστές συνδέσμων. Όπως έχει αναπτυχθεί στην ενότητα 5.5, το σήμα κατάστασης που αναφέρεται στο state migration είναι το

$$s=\left[\left(R_1,\triangleright\triangleleft_{i_1}\right),\left(R_2,\triangleright\triangleleft_{i_2}\right),\ldots,\left(R_n,\triangleright\triangleleft_{i_n}\right)\right]$$

και οι δράσεις

$$A(s)=\left\{Migrate\left(\triangleright\triangleleft_i\mapsto\triangleright\triangleleft_{i+1}\right)\vee Migrate\left(\triangleright\triangleleft_{i+1}\mapsto\triangleright\triangleleft_i\right)\mid i=1,\ldots,n-2\right\}\cup\left\{nothing\right\}$$

με ενισχύσεις

$$r\left(\left[\ldots,\left(R_i,\triangleright\triangleleft_{i-1}\right),\ldots\right],Migration\left(\triangleright\triangleleft_{i-1}\mapsto\triangleright\triangleleft_i\right)\right)=$$
$$-cost\cdot\left[Q\left(R_i,\triangleright\triangleleft_i\right)-Q\left(R_i,\triangleright\triangleleft_{i-1}\right)\right]$$

$$r\left(\left[\ldots,\left(R_i,\triangleright\triangleleft_{i-1}\right),\ldots\right],nothing\right)=0$$

Η πράξη Migration αλλάζει την κατάσταση ενώ η πράξη nothing την αφήνει απαράλλαχτη. Η καλύτερη προσέγγιση είναι μάλλον να αντιμετωπιστεί ως πρόβλημα χωρίς καταστάσεις, δηλαδή ο eddy να κρατά μόνο τιμές $q(a)$ για τις παραπάνω δράσεις. Ένας αλγόριθμος επεξεργασίας ερωτημάτων θα ήταν πανομοιότυπος με τον αλγόριθμο 6.4 και περιοδικά θα «πάγωνε» την επεξεργασία και θα έλυνε το πρόβλημα της άρσης του βάρους της ιστορίας.

Αλγόριθμος 6.3 : EddySTAIRs learning

1. Λύση του EddyJoins
2. if $(criterion)$
   $a \leftarrow somehow\left(Q, A(s)\right)$
   $take(a)$
   $s' \leftarrow next\_state$
   $r \leftarrow reward$
   $q \leftarrow update\left(q, A(s)\right)$
   $s \leftarrow s'$

Ίσως η καλύτερη πρακτική για την εξερεύνηση είναι μια συντηρητική πρακτική τύπου uniform ή ε – greedy αφού η δράση Migrate θα έχει σημαντικό υπολογιστικό κόστος. Η ανανέωση των τιμών q γίνεται με έναν τρόπο όπως

$$update\left(q(a),r\right)=q(i)+\eta\left(r-q(a)\right)$$

δίνοντας μεγάλο βάρος στις βραχυπρόθεσμες ενισχύσεις. Το πότε θα γίνεται η διερεύνηση του state migration (η συνθήκη του if) είναι επίσης άλλη μία σχεδιαστική επιλογή. Θα μπορούσε να γίνεται κάθε Ν βήματα ή όταν υπάρχει διαφωνία ανάμεσα στην κατάσταση αποθήκευσης και τη δρομολόγηση μεγαλύτερη από ένα όριο.

6.4 Εκτέλεση γενικών ερωτημάτων με επιλογές και τελεστές SteMs

Η εκτέλεση ερωτημάτων στην αρχιτεκτονική EddySTeMs5 μπορεί να γίνει με έναν από τους παραπάνω τρόπους, χρησιμοποιώντας δηλαδή μια στοίβα για τους τελεστές (SteMs και AMs) που δεν έχουν επιστρέψει όλα τα matches τους. Στην δημοσίευση όμως των SteMs [RDH03] χρησιμοποιείται ένα ασύγχρονο μοντέλο επεξεργασίας, υποβοηθούμενο από τις πλειάδες EOT (End Of Transmission). Μπορούμε να αντιμετωπίσουμε τις πλειάδες αυτές σαν άλλη μια κατάσταση, οι οποίες μπορούν μόνο να γίνουν build στο αντίστοιχο SteM. Στο σύνολο καταστάσεων λοιπόν της ενότητας 5.7 προστίθεται το $\{EOT\}$ και στα σύνολα $A(s)$ προστίθεται το

$A(EOT) = \{build(SteM_R) | SteM_R \text{ νόμιμο}\}$.

Ένα πρόβλημα μιας ασύγχρονης υλοποίησης είναι ότι ο πράκτορας ίσως δεν μπορεί να δει την επόμενη κατάσταση και την ενίσχυση από μια πράξη παρά πολύ μετά, και τότε πρέπει να ανανεώσει την συνάρτηση τιμών. Το πρόβλημα είναι ότι ενώ οι πλειάδες ταξιδεύουν ασύγχρονα, το μοντέλο της ενισχυτικής μάθησης προϋποθέτει στη βασική του μορφή σύγχρονη επικοινωνία πράκτορα και περιβάλλοντος. Μπορούμε να παρακάμψουμε το πρόβλημα αυτό χρησιμοποιώντας το ίδιο το dataflow για την κυκλοφορία των σημάτων κατάστασης και ενίσχυσης. Θα εισάγουμε το γενικό κανόνα ότι κάθε πλειάδα έχει προκύψει από μια δράση πάνω σε μια προηγούμενη κατάσταση, και η ενίσχυση της δράσης αυτής καθώς και η προηγούμενη κατάσταση και δράση ταξιδεύουν μαζί με την πλειάδα, δηλαδή τα σήματα που κυκλοφορούν στο dataflow είναι της μορφής $sig = (t, s_{prev}, a_{prev}, r)$. Αυτό είναι αληθές εκτός από τις περιπτώσεις που μια δράση έχει αποτέλεσμα null αλλά μπορεί εύκολα να γενικευθεί εισάγοντας ειδικές πλειάδες *null* οπότε σήματα του τύπου $(null, s_{prev}, a_{prev,} r)$.

Ένας απλοποιημένος γενικός αλγόριθμος που συνδυάζει την επεξεργασία του ερωτήματος με τη μάθηση φαίνεται παρακάτω. Οι δράσεις των επιλογών μπορούν πάλι να γίνονται στην αρχή, με ένα «εξωτερικό» eddy όπως περιγράφεται στην ενότητα 6.2.

Αλγόριθμος 6.4 : EddySteMs learning

1. Αρχικοποίηση

   $s', r, s, a, Q(s,a), i = 0, \ldots, n+1$

2. Επεξεργασία

   repeat

      δες το τρέχον σήμα $\left(t, s_{prev}, a_{prev}, r\right)$

      $s \leftarrow stateOf\left(t.TD\right)$

      $Q\left(s_{prev}, a_{prev}\right) \leftarrow update\left(Q, r\right)$

      $a \leftarrow somehow\left(Q, A\left(s\right)\right)$

      $take\left(a\right)$

   until $\left(\text{end of query}\right)$

Πρέπει να σημειωθεί ότι η ασύγχρονη ενισχυτική μάθηση είναι ένα πολύ ενεργό πεδίο έρευνας και υπάρχουν αποδείξεις σύγκλισης για τους περισσότερους αλγόριθμους ενισχυτικής μάθησης στην ασύγχρονή τους μορφή.

7

Επίλογος

7.1 Σύνοψη και συμπεράσματα

Η βελτιστοποίηση ερωτημάτων Βάσεων Δεδομένων βασισμένη στο κόστος είναι ένα πολύ δύσκολο πρόβλημα και η έρευνα έχει να επιδείξει ένα ευρύ φάσμα προσεγγίσεων και αλγόριθμων για τη λύση του. Νέου τύπου περιβάλλοντα επεξεργασίας δεδομένων αλλά και πολυπλοκότητες και συσχετίσεις στα ίδια τα δεδομένα απαιτούν τη διάσπαση του βρόχου κατασκευής μοντέλου για τα δεδομένα, βελτιστοποίησης και εκτέλεσης ενός ερωτήματος με τον τρόπο που αυτός γίνεται στα υπάρχοντα ΣΔΒΔ. Το πεδίο της Προσαρμοστικής Επεξεργασίας Ερωτημάτων που προέκυψε από την ανάγκη αυτή έχει να επιδείξει σημαντικά αποτελέσματα, με τα eddies να είναι ίσως το πιο ενδημικό δείγμα της φιλοσοφίας του. Ένα σημαντικό κενό στη βιβλιογραφία είναι πώς ο πλούτος των μεθόδων βελτιστοποίησης μπορεί να ενταχθεί στο πλαίσιο των eddies.

Το κύριο αποτέλεσμα της εργασίας αυτής είναι ότι έδειξε πώς η έννοια της βελτιστοποίησης ερωτήματος μπορεί να ταιριάξει με σαφή τρόπο στο μοντέλο εκτέλεσης ερωτημάτων με eddies χρησιμοποιώντας το μοντέλο της Ενισχυτικής Μάθησης. Η αναζήτηση του βέλτιστου πλάνου δεν γίνεται στο χώρο των πολιτικών (πλάνων), κάτι που θα κατέστρεφε την έννοια της προσαρμοστικότητας ανά πλειάδα αλλά μάλλον σε ένα χώρο καταστάσεων ώστε να γίνεται ταυτόχρονα με την εκτέλεση του ερωτήματος. Η εργασία αυτή είναι ένα μικρό

κομμάτι μόνο μιας έρευνας για το κατά πόσον ο μηχανισμός των eddies μπορεί να γίνει ένας καθολικός μηχανισμός επεξεργασίας ερωτημάτων σε βάσεις δεδομένων.

Αρχικά δώσαμε μια βιβλιογραφική εισαγωγή στα πεδία της Βελτιστοποίησης Ερωτημάτων και Ενισχυτικής Μάθησης στα οποία βασίζεται η παρούσα εργασία καθώς και μια συστηματική παρουσίαση του μηχανισμού των eddies, στο επίπεδο ανάλυσης που βρίσκεται σήμερα. Ακολούθως δείξαμε πώς τα ζητήματα βελτιστοποίησης που εγείρονται μπορούν να διαχωριστούν και να αντιμετωπιστούν όσο το δυνατόν ξεχωριστά. Δείξαμε πώς μια μοντελοποίηση μπορεί να μετατρέψει τα ζητήματα αυτά σε προβλήματα ενισχυτικής μάθησης, και τέλος δείξαμε πώς διάφοροι αλγόριθμοι μάθησης μπορούν να συνδυαστούν με την επεξεργασία του ερωτήματος.

7.2 Μελλοντικές επεκτάσεις

Η προφανής και πρώτη κατεύθυνση μελλοντικής δουλειάς είναι η υλοποίηση των παραπάνω σε ένα πραγματικό σύστημα και η υποβολή τους σε διεξοδικές δοκιμές με πραγματικά δεδομένα, με σκοπό την σύγκριση αποδοτικότητας αλγορίθμων και εύρεση των παραμέτρων που εισέρχονται. Άλλες μελλοντικές κατευθύνσεις έρευνας σημειώνονται παρακάτω.

Στο πεδίο των eddies ίσως το πιο σημαντικό βήμα θα ήταν να βρεθεί ένας τρόπος ευέλικτης αποθήκευσης των ενδιάμεσων αποτελεσμάτων είτε στο περιβάλλον των STeMs είτε αυτό των STAIRs. Στο πεδίο της μοντελοποίησης του eddy ως πράκτορα σε πρόβλημα ενισχυτικής μάθησης, υπάρχουν πολλές προχωρημένες τεχνικές που δεν εξετάστηκαν, και κυρίως οι έννοιες των POMDPs, Hierarchical Models και Generalization που μπορούν να επιτρέψουν μεγάλους χώρους καταστάσεων και ελλιπή γνώση του περιβάλλοντος καθώς και νεότεροι και πιο προχωρημένοι αλγόριθμοι μάθησης. Τέλος, ένα θέμα είναι κατά πόσον τεχνικές content – based routing [BBDW05] μπορούν να εισαχθούν στη μάθηση και πώς θα άλλαζε αυτό τον χώρο καταστάσεων.

8

Βιβλιογραφία

[AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 261-272. ACM, 2000.

[BB05] Shivnath Babu and Pedro Bizarro. Adaptive query processing in the looking glass. In *CIDR*, pages 238-249, 2005.

[BBDW05] Pedro Bizarro, Shivnath Babu, David J. DeWitt, and Jennifer Widom. Content-based routing: Different plans for different data. In *VLDB*, pages 757-768, 2005.

[Ber82] Dimitri P. Bertsekas. Distributed dynamic programming. *IEEE Transactions on Automatic Control*, 27:610-616, 1982.

[BF86] Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments.* Routledge, 1986.

[BFI91] Kristin P. Bennett, Michael C. Ferris, and Yannis E. Ioannidis. A genetic algorithm for database query optimization. In *ICGA*, pages 400-407, 1991.

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden,

Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[Des04] Amol Deshpande. An initial study of overheads of eddies. *SIGMOD Rec.*, 33(1):44-49, 2004.

[DH04] Amol Deshpande and Joseph M. Hellerstein. Lifting the burden of history from adaptive query processing. In *VLDB*, pages 948-959, 2004.

[DHR06] Amol Deshpande, Joseph M. Hellerstein, and Vijayshankar Raman. Adaptive query processing: why, how, when, what next. In *SIGMOD Conference*, pages 806-807, 2006.

[GMUW02] Hector Garcia-Molina, Jeffrey D. Ullman, and Jeniffer Widom. *Database Systems: The Complete Book.* Prentice Hall, 2002.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[HFC+00] Joseph M. Hellerstein, Michael J. Franklin, Sirish Chandrasekaran, Amol Deshpande, Kris Hildrum, Samuel Madden, Vijayshankar Raman, and Mehul A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Eng. Bull.*, 23(2):7-18, 2000.

[HH99] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 287-298. ACM Press, 1999.

[IK90] Yannis E. Ioannidis and Younkyung Cha Kang. Randomized algorithms for optimizing large join queries. In *SIGMOD Conference*, pages 312-321, 1990.

[IK91] Yannis E. Ioannidis and Younkyung Cha Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *SIGMOD Conference*, pages 168-177, 1991.

[Ioa96] Yannis E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1):121-123, 1996.

[IW87] Yannis E. Ioannidis and Eugene Wong. Query optimization by simulated annealing. In *SIGMOD Conference*, pages 9-22, 1987.

[KBZ86] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of nonrecursive queries. In *VLDB*, pages 128-137, 1986.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res. (JAIR)*, 4:237-285, 1996.

[Mit97] Tom Mitchel. *Machine Learning.* Mc Graw Hill., 1997.

[RDH03] Vijayshankar Raman, Amol Deshpande, and Joseph M. Hellerstein. Using state modules for adaptive query processing. In *ICDE*, pages 353-, 2003.

[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *SIGMOD Conference*, pages 23-34. ACM, 1979.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press Cambridge, 1998.

[SJJ94] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In *NIPS*, pages 361-368, 1994.

[SMK97] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB J.*, 6(3):191-208, 1997.

[SWKH76] Michael Stonebraker, Eugene Wong, Peter Kreps, and Gerald Held. The design and implementation of ingres. *ACM Trans. Database Syst.*, 1(3):189-222, 1976.

[TD03] Feng Tian and David J. DeWitt. Tuple routing strategies for distributed eddies. In *VLDB*, pages 333-344, 2003.

[WW94] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *OSDI*, pages 1-11, 1994.