ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΤΟΜΕΑΣ ΒΙΟΜΗΧΑΝΙΚΗΣ ΔΙΟΙΚΗΣΗΣ & ΕΠΙΧΕΙΡΗΣΙΑΚΗΣ ΕΡΕΥΝΑΣ

Αλγόριθμοι Πληροφοριών Χρονοδιαγράμματος & Διερεύνηση Υλοποίησής τους για τον Οργανισμό Σιδηροδρόμων Ελλάδος (ΟΣΕ)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΕΤΡΟΥ Κ. ΗΛΙΑΔΗ

Επιβλέπων: Νικόλαος Παναγιώτου
Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2016

Στον κυρ Κώστα και στην κυρά Ματίνα…

Περίληψη

Στο πλαίσιο της παρούσας Διπλωματικής Εργασίας, μελετώνται οι αλγόριθμοι που επιλύουν τα προβλήματα πληροφοριών χρονοδιαγράμματος του τύπου: "Δοθέντος ενός σταθμού αναχώρησης και ενός σταθμού προορισμού καθώς και μιας ώρας αναχώρησης, βρείτε τη βέλτιστη διαδρομή.", ως ένα ενιαίο πρόβλημα συντομότερης διαδρομής σε έναν κατάλληλα διαμορφωμένο στατικό γράφο.

Η ανάγκη για τη μελέτη των προβλημάτων αυτών προέκυψε από τον Οργανισμό Σιδηροδρόμων Ελλάδος (Ο.Σ.Ε.), ο οποίος αναζητά ένα σύγχρονο ηλεκτρονικό σύστημα πληροφοριών μέσω διαδικτύου, το οποίο θα είναι σε θέση να ανταποκρίνεται στα επερωτήματα των καταναλωτών σε αποδεκτό χρόνο.

Αρχικά παρέχουμε στον αναγνώστη όλες τις αναγκαίες γνώσεις για την κατανόηση του προβλήματος. Έπειτα ορίζουμε πρώτα ένα απλοποιημένο πρόβλημα πληροφοριών χρονοδιαγράμματος, το οποίο περιέχει ένα σύνολο παραδοχών ώστε να αμεληθούν προσωρινά οι παράγοντες που δεν επηρεάζουν τη μορφή της λύσης. Παρουσιάζουμε στη συνέχεια τις δύο βασικές προσεγγίσεις επίλυσης του προβλήματος αυτού: τη χρονικά-εξαρτώμενη και τη χρονικά-επεκτεινόμενη προσέγγιση. Σε καθεμιά από αυτές τις δύο προσεγγίσεις κατασκευάζεται ένας κατευθυνόμενος γράφος και εφαρμόζεται επάνω του ο κλασικός αλγόριθμος εύρεσης συντομότερης διαδρομής του Dijkstra. Ακολούθως, παραθέτουμε τις βασικότερες τεχνικές-επιτάχυνσης για τις συγκεκριμένες προσεγγίσεις.

Στο επόμενο στάδιο προχωράμε σε μια πιο ρεαλιστική μοντελοποίηση του προβλήματος, λαμβάνοντας υπόψη και του παράγοντες τους οποίους αμελήσαμε κατά τον ορισμό του απλοποιημένου προβλήματος. Ορίζουμε το ρεαλιστικό πρόβλημα νωρίτερης άφιξης και παρουσιάζουμε πώς προεκτείνονται τα μοντέλα του απλοποιημένου προβλήματος ώστε να μπορούν να αντιμετωπίσουν τα ρεαλιστικά χαρακτηριστικά, όπως τις μη στιγμιαίες μετεπιβιβάσεις και τις ημέρες κυκλοφορίας. Έπειτα παραθέτουμε και κάποια πιο ειδικά προβλήματα τα οποία είναι σε θέση να λύσουν τα παραπάνω μοντέλα, όπως το πρόβλημα του ελάχιστου αριθμού μετεπιβιβάσεων ή τις περιπτώσεις δύο κριτηρίων.

Για όλα τα προβλήματα που μελετάμε παρουσιάζουμε τις πειραματικές μελέτες από τη βιβλιογραφία που συγκρίνουν τις προσεγγίσεις επίλυσης με βάση τους χρόνους επεξεργασίας που απαιτούν.

Τέλος, παρουσιάζουμε τα στάδια της εφαρμογής που υλοποιήσαμε κατά τη μελέτη μας. Παρουσιάζουμε τον σχεδιασμό και τη δόμηση των δεδομένων, περιγράφουμε λεπτομερώς τις υλοποιήσεις στις οποίες προβήκαμε και προτείνουμε την προσέγγιση που αρμόζει καλύτερα στο δίκτυο και στις ανάγκες του Ο.Σ.Ε.

Abstract

In the course of this thesis, we study the algorithms that solve timetable information problems such as: “Given a departure station and a destination station as well as a departure time, find the optimal path.", as a single shortest path problem on a suitably formed static graph.

The need to study these problems came from the Hellenic Railways Organization (O.S.E.), who is currently in need of a modern web-based information system capable of responding to customer queries in a reasonable time.

Initially we provide the reader with all the necessary knowledge in order to understand the problem. After that we define a simplified timetable information problem, that includes a set of assumptions in order to temporarily ignore the factors that do not affect the form of the solution. We continue to present the two basic approaches of resolving this problem: the time-depended and time-expanded approach. In each of these approaches a directed graph is constructed and the classical shortest path finding Dijkstra algorithm is applied to it. Additionally, we present the most basic acceleration technics for the specific approaches.

In the next stage we move on to a more realistic modeling of the problem, taking into consideration the factors that we ignored during the definition of the simplified problem. We define the early arrival problem and show how the models of the simplified problem are expanded in order to be capable of handling the realistic characteristics, such as non-instant transfers and traffic days. Subsequently we present some special problems the above models can also solve, such as the minimum number of transfers problem or the bicriteria cases.

For all the problems we study, we additionally include the experimental studies from the bibliography which compare the resolution approaches based on the process time they require.

Lastly, we present the stages of the application we implemented during our study. We show the design and the construction of the data, describe the applications we implemented in detail, and suggest the approach that best fits the network and needs of O.S.E.

Σύνοψη

Η παρούσα διπλωματική εργασία μελετά τους αλγορίθμους που επιλύουν τα προβλήματα πληροφοριών χρονοδιαγράμματος των σιδηροδρομικών δικτύων. Με τον όρο προβλήματα πληροφοριών χρονοδιαγράμματος εννοούμε τα προβλήματα του τύπου : "Δοθέντος ενός σταθμού αναχώρησης και ενός σταθμού προορισμού, καθώς και μιας ώρας αναχώρησης, βρείτε τη βέλτιστη διαδρομή.".

Η ανάγκη για τη μελέτη των προβλημάτων αυτών προέκυψε από τον Οργανισμό Σιδηροδρόμων Ελλάδος (Ο.Σ.Ε.). Τα σύγχρονα δεδομένα στην εποχή της πληροφορίας απαιτούν από τις εταιρίες μεταφορών να εγκαταστήσουν τέτοια συστήματα ώστε να είναι σε θέση να παρέχουν οποιαδήποτε πληροφορία που αφορά στο χρονοδιάγραμμά τους αναζητεί ο καταναλωτής άμεσα μέσω διαδικτύου. Σε αυτή τη λογική και ο Ο.Σ.Ε. προχώρησε σε έρευνα αγοράς ώστε να βρει ένα σύστημα που να καλύπτει τις ανάγκες του. Να αναφέρουμε πως στα πλαίσια αυτής της έρευνας, έχει ήδη απορρίψει ένα πληροφοριακό σύστημα που του προτάθηκε από την εταιρία Singular-Logic, για τον λόγο πως το σύστημα δεν κάλυπτε τις απαιτήσεις σε χρόνους απόκρισης. Πιο συγκεκριμένα, το σύστημα ήταν πρακτικά αδύνατο να υπολογίσει τις βέλτιστες διαδρομές που συμπεριελάμβαναν μετεπιβιβάσεις, καθώς οι χρόνοι επεξεργασίας σε ορισμένες περιπτώσεις ξεπερνούσαν και τα 40 δευτερόλεπτα, γεγονός που καθιστά άμεσα ένα τέτοιο σύστημα απορριπτέο. Για τον λόγο αυτό ο οργανισμός στράφηκε στο Εθνικό Μετσόβιο Πολυτεχνείο ώστε να διεξάγει μια αρχική έρευνα πάνω στα εργαλεία και στους αλγορίθμους με τα οποία αντιμετωπίζονται σήμερα αποδοτικά αυτού του είδους τα προβλήματα πληροφοριών.

Στα πλαίσια αυτής της αρχικής έρευνας εντάσσεται και η παρούσα διπλωματική εργασία. Στόχο έχει να αναλύσει τις δύο βασικές προσεγγίσεις επίλυσης του προβλήματος πληροφοριών χρονοδιαγράμματος ως πρόβλημα συντομότερης διαδρομής σε έναν κατάλληλα διαμορφωμένο γράφο, και να παρουσιάσει τους χρόνους επεξεργασίας που απαιτούνται. Η ιδέα είναι να προκύψει ο κατάλληλος αλγόριθμος ο οποίος θα αποτελέσει τη βάση για τη δημιουργία ενός ολοκληρωμένου και αποδοτικού συστήματος πληροφοριών χρονοδιαγράμματος για τον Ο.Σ.Ε.

Οι απαιτήσεις μας από το σύστημα είναι η βελτιστότητα των προτεινόμενων διαδρομών αλλά και ο αποδεκτός χρόνος επεξεργασίας. Το σύστημα πρέπει είναι ικανό να μοντελοποιήσει μια σειρά από ρεαλιστικές λεπτομέρειες όπως μετεπιβιβάσεις, ημέρες λειτουργίας των συρμών, χρόνοι περπατήματος μεταξύ γειτονικών σταθμών κ.α.

Η μελέτη μας ξεκινά με μια σύντομη εισαγωγή στο θέμα στο 1ο Κεφάλαιο. Έπειτα, κατά το 2ο Κεφάλαιο, παραθέτουμε συνοπτικά όλο το αναγκαίο θεωρητικό υπόβαθρο για τη συνέχεια.

Στο 3ο Κεφάλαιο, ορίζουμε πρώτα το απλοποιημένο πρόβλημα πληροφοριών χρονοδιαγράμματος με το όνομα "Πρόβλημα Νωρίτερης Άφιξης" (ΠΝΑ), το οποίο περιέχει ένα σύνολο παραδοχών ώστε να αμεληθούν προσωρινά οι παράγοντες που δεν επηρεάζουν τη μορφή της λύσης. Παρουσιάζουμε έπειτα τις δύο βασικές προσεγγίσεις επίλυσης του προβλήματος αυτού: τη χρονικά-εξαρτώμενη και τη χρονικά-επεκτεινόμενη προσέγγιση. Σε καθεμιά από αυτές τις δύο προσεγγίσεις κατασκευάζεται ένας κατευθυνόμενος γράφος και εφαρμόζεται επάνω του ο κλασικός αλγόριθμος εύρεσης συντομότερης διαδρομής του Dijkstra (με κάποιες μικρές προσαρμογές για την εκάστοτε περίπτωση), ο οποίος αναλύεται και παρουσιάζεται με

λεπτομέρεια. Στη συνέχεια παραθέτουμε τις βασικότερες τεχνικές-επιτάχυνσης για τις συγκεκριμένες προσεγγίσεις. Στο τέλος του κεφαλαίου παρουσιάζουμε τις πειραματικές μελέτες από τη βιβλιογραφία που συγκρίνουν αυτές τις προσεγγίσεις.

Κατά το 4ο Κεφάλαιο της παρούσας εργασίας προχωράμε σε μια πιο ρεαλιστική μοντελοποίηση του προβλήματος, λαμβάνοντας υπόψη και του παράγοντες τους οποίους αμελήσαμε κατά τον ορισμό του απλοποιημένου προβλήματος. Ορίζουμε το ρεαλιστικό πρόβλημα νωρίτερης άφιξης. Παρουσιάζουμε πώς προεκτείνονται τα μοντέλα του απλοποιημένου προβλήματος ώστε να μπορούν να αντιμετωπίσουν τα ρεαλιστικά χαρακτηριστικά, όπως τις μη στιγμιαίες μετεπιβιβάσεις και τις ημέρες κυκλοφορίας. Έπειτα παραθέτουμε και κάποια πιο ειδικά προβλήματα τα οποία είναι σε θέση να λύσουν τα παραπάνω μοντέλα, όπως το πρόβλημα του ελάχιστου αριθμού μετεπιβιβάσεων καθώς και τη δικριτηριακή περίπτωση. Στο τέλος του κεφαλαίου παρουσιάζουμε τις πειραματικές μελέτες που συγκρίνουν τις δυο ρεαλιστικές προσεγγίσεις με βάση τους χρόνους επεξεργασίας που απαιτούν.

Τέλος, στο 5ο Κεφάλαιο, παρουσιάζουμε τα στάδια της εφαρμογής που υλοποιήσαμε κατά την προσπάθεια της μελέτης μας. Παρουσιάζουμε τον σχεδιασμό και τη δόμηση των δεδομένων καθώς και κάποια προγραμματιστικά εργαλεία που μας φάνηκαν χρήσιμα. Περιγράφουμε με λεπτομέρεια τις υλοποιήσεις στις οποίες προβήκαμε και προτείνουμε την προσέγγιση που αρμόζει καλύτερα στο δίκτυο και στις ανάγκες του Ο.Σ.Ε.

Περιεχόμενα

1. Εισαγωγή

Από όταν κατασκευάστηκαν οι πρώτες σιδηροδρομικές γραμμές, η δυνατότητα μεταφοράς απέκτησε σημασία στη σύγχρονη ζωή, και τα συστήματα μαζικής μεταφοράς εξελίχθηκαν σε τεράστια και σύνθετα δίκτυα. Μέχρι τα τέλη της δεκαετίας του ’80, οι διαδρομές έπρεπε να σχεδιάζονται με το χέρι χρησιμοποιώντας εκτυπωμένους σιδηροδρομικούς οδηγούς. Σε μικρά σιδηροδρομικά δίκτυα τα βέλτιστα δρομολόγια μπορούν να καθοριστούν εύκολα χειροκίνητα, αλλά σε μεγαλύτερα χρονοδιαγράμματα ο χειροκίνητος σχεδιασμός γίνεται δύσκολος και χρονοβόρος. Στα τέλη της δεκαετίας του ’80 του περασμένου αιώνα εγκαταστάθηκαν τα πρώτα ηλεκτρονικά συστήματα πληροφοριών χρονοδιαγράμματος στη Γερμανία. Σημερινά συστήματα είναι για παράδειγμα το HAFAS, το οποίο χρησιμοποιείται από πολλές ευρωπαϊκές σιδηροδρομικές εταιρείες, ή το EFA, το οποίο χρησιμοποιείται κυρίως για την τοπική κυκλοφορία και περιορίζεται σε μικρές περιοχές της Ευρώπης. Εμπειρικά, οι προκύπτουσες συνδέσεις είναι στην πλειονότητα των περιπτώσεων ικανοποιητικές. Υπάρχουν περιπτώσεις ωστόσο, για τις οποίες οι προτεινόμενες διαδρομές σαφώς και δεν είναι βέλτιστες. Ο κύριος λόγος για αυτές τις μη βέλτιστες συνδέσεις είναι ότι οι αλγόριθμοι πίσω από τα συστήματα χρησιμοποιούν ευρετικές μεθόδους για τη μείωση του χώρου αναζήτησης (προκειμένου να επιτευχθεί ένας αποδεκτός χρόνος απόκρισης) οι οποίες δεν εγγυώνται πάντα βέλτιστες λύσεις.

Στα 20 με 25 χρόνια που πέρασαν από τότε που αναπτύχθηκαν τα πρώτα συστήματα πληροφοριών χρονοδιαγράμματος, η απόδοση των υπολογιστών έχει βελτιωθεί δραστικά. Έχουν επιτευχθεί υπολογιστές -με πολλαπλούς πυρήνες και συχνότητες αρκετών gigahertz και πολλών gigabyte κύριας μνήμης- και έχουμε φτάσει στο σημείο όπου ο χρόνος εκτέλεσης των αλγορίθμων που εγγυώνται βέλτιστες λύσεις να είναι εφικτός για την εφαρμογή τους σε ένα παραγωγικό σύστημα πληροφοριών δρομολογίων.

Στην παρούσα εργασία μελετάμε τα δύο μοντέλα που χαρτογραφούν ένα επερώτημα πληροφοριών χρονοδιαγράμματος ως ένα *ενιαίο πρόβλημα συντομότερης-διαδρομής [single shortest-path problem]* πάνω σε έναν κατάλληλα ορισμένο γράφο (σε αντίθεση με τους περισσότερους αλγόριθμους πίσω από τα υπάρχοντα συστήματα που χωρίζουν το πρόβλημα σε δύο ή περισσότερα μέρη). Δύο ζητήματα είναι ζωτικής σημασίας για τη μοντελοποίηση: Από τη μία πλευρά, πρέπει να συμπεριληφθούν πολλές λεπτομέρειες που αφορούν στις εφικτές συνδέσεις, όπως για παράδειγμα οι κανόνες για τις μετεπιβιβάσεις. Από την άλλη πλευρά, πρέπει να καθοριστεί με σαφήνεια η έννοια του βέλτιστου. Τα διαφορετικά κριτήρια όπως η νωρίτερη άφιξη στον προορισμό ή η ελαχιστοποίηση των μετεπιβιβάσεων μοντελοποιούνται ως βάρη ακμών. Εκτός από τη βέλτιστη μοντελοποίηση, ζωτικής σημασίας είναι και ο μέσος χρόνος τρεξίματος των αλγορίθμων. Παρουσιάζουμε τα αποτελέσματα από πειραματικές μελέτες της βιβλιογραφίας για τη σύγκριση των χρόνων τρεξίματος.

Δεν είναι μόνο η δύναμη των σύγχρονων υπολογιστών υψηλής απόδοσης που μπορεί να χρησιμοποιηθεί για να υπολογίσει τις βέλτιστες συνδέσεις. Οι αλγοριθμικές βελτιώσεις των τυποποιημένων αλγορίθμων συντομότερης διαδρομής έχουν επίσης μειώσει σημαντικά το χρόνο

τρεξίματος. Από τη στιγμή που μεταφράσαμε το πρόβλημα πληροφορίων χρονοδιαγράμματος σε πρόβλημα συντομότερης διαδρομής, μπορούμε να χρησιμοποιήσουμε τις παραλλαγές του κλασικού αλγορίθμου συντομότερης διαδρομής του Dijkstra που θα αναλύσουμε παρακάτω. Ο χρόνος τρεξίματος αυτών των αλγορίθμων που εφαρμόζονται σε πραγματικές περιπτώσεις μπορεί να βελτιωθεί σημαντικά με τεχνικές επιτάχυνσης – ενώ την ίδια στιγμή, οι αλγόριθμοι μπορούν να εγγυόνται βέλτιστες λύσεις. Παρουσιάζουμε τέτοιες τεχνικές επιτάχυνσης και τα σημαντικά σημεία τους.

Όλη η ανάλυση μας κατά την παρούσα εργασία στοχεύει στην επιλογή του κατάλληλου αλγορίθμου, ο οποίος θα είναι σε θέση να αποτελέσει τη θεωρητική βάση για τη δημιουργία ενός σύγχρονου και αποδοτικού ηλεκτρονικού συστήματος πληροφοριών χρονοδιαγράμματος για τον Οργανισμό Σιδηροδρόμων Ελλάδος (Ο.Σ.Ε.).

2. Βασικές Αρχές

2.1. Αντικειμενοστραφής Προγραμματισμός

Οι πιο δημοφιλείς γλώσσες προγραμματισμού που αναπτύχθηκαν τα τελευταία 30 χρόνια είναι όλες τους *αντικειμενοστραφείς* [*object-oriented*]. Προφανώς λοιπόν το μοντέλο του *αντικειμενοστραφούς σχεδιασμού* [*object-oriented design*] υπερτερεί των υπολοίπων μοντέλων. Παρακάτω θα προσπαθήσουμε να περιγράψουμε συνοπτικά τις βασικές αρχές του σχεδιασμού αυτού και τα κυριότερα πλεονεκτήματα που προσφέρει, τα οποία και θα προσπαθήσουμε να εκμεταλλευτούμε στον δικό μας σχεδιασμό.

Παλαιότερα κυριαρχούσε η φιλοσοφία του *δομημένου προγραμματισμού* [*procedural programming*]. Σε γλώσσες όπως η COBOL ή η FORTRAN τα προγράμματα ουσιαστικά ήταν ένα μεγάλο κομμάτι κώδικα. Όλα τα δεδομένα και οι μεταβλητές ορίζονταν σε ένα μέρος και όλη η λογική σε ένα άλλο. Όσο τα προβλήματα μεγάλωναν, μεγάλωνε και η δυσκολία δημιουργίας και διαχείρισης τέτοιων προγραμμάτων. Έτσι άρχισαν να γίνονται γνωστές οι αντικειμενοστραφείς γλώσσες προγραμματισμού.

Η βασική ιδέα στις γλώσσες αυτές, είναι να χωρίσουμε το πρόγραμμα σε διάφορα ξεχωριστά μεταξύ τους *αντικείμενά* [*objects*], το καθένα από τα οποία αναπαριστά ένα διαφορετικό μέρος της εφαρμογής, και περιέχει τα δικά του δεδομένα και τη δική του λογική, τα οποία μπορούν όταν χρειάζεται να επικοινωνήσουν μεταξύ τους. Τα αντικείμενα αυτά δεν επιλέγονται τυχαία. Αντίθετα προκύπτουν από τον τρόπο με τον οποίο προσεγγίζουμε και μοντελοποιούμε το εκάστοτε πρόβλημα που καλούμαστε να λύσουμε. Για παράδειγμα, αντικείμενα θα μπορούσαν να είναι τα εξής: υπάλληλοι, τραπεζικοί λογαριασμοί, πλανήτες, αρχεία ήχου κ.τ.λ. Οτιδήποτε δηλαδή έχει νόημα για την εφαρμογή μας.

Μια άλλη βασική ιδέα είναι αυτή της *κλάσης* [*class*]. Η κλάση είναι το σχεδιάγραμμα ενός αντικειμένου, δηλαδή το εργαλείο με το οποίο δημιουργούμε ένα αντικείμενο. Η κλάση περιγράφει αυτό που θα είναι ένα αντικείμενο, αλλά δεν είναι το ίδιο τα αντικείμενο. Ένα κλασικό παράδειγμα είναι αυτό του τεχνικού σχεδίου. Πριν κατασκευάσουμε το κτήριο που θέλουμε, καταστρώνουμε πρώτα τα τεχνικά σχέδια που το περιγράφουν. Έπειτα χρησιμοποιούμε τα σχέδια αυτά για να κατασκευάσουμε το κτήριο ή όσα κτήρια θέλουμε. Δουλειά μας λοιπόν στον αντικειμενοστραφή σχεδιασμό είναι να καταστρώνουμε πρώτα τα σχέδια, που ονομάζονται κλάσεις, τα οποία θα χρησιμοποιούμε στη συνέχεια για να δημιουργήσουμε τα αντικείμενα μας.

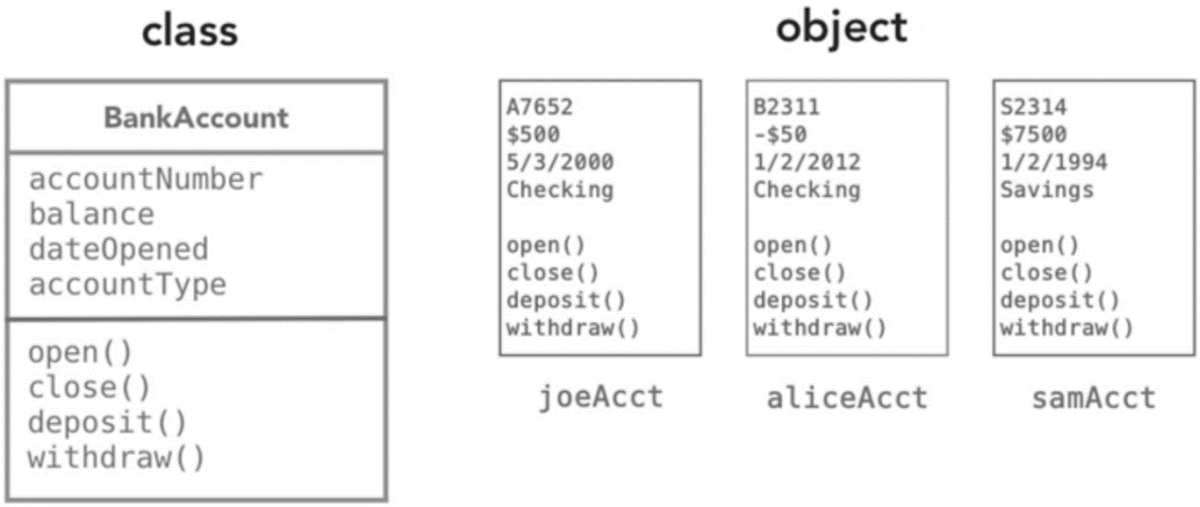
Οι έννοιες της κλάσης και του αντικειμένου γίνονται καλύτερα αντιληπτές με ένα παράδειγμα. Στην αριστερή πλευρά του Σχήματος 2.1 παρουσιάζεται ο ορισμός μιας κλάσης με το όνομα BankAccount. Το σχήμα αυτό είναι το βασικό στοιχείο της *Ενοποιημένης Γλώσσας*

Σχεδιασμού (Ε.Γ.Σ.) [*Unified Modeling Language (U.M.L.)*]. Η U.M.L. είναι ο πιο συνηθισμένος τρόπος περιγραφής κλάσεων, μιας και δίνουν τη δυνατότητα να περιγραφούν με σαφήνεια τα βασικά στοιχεία που τις αποτελούν μέσα σε λίγο χώρο. Το σχήμα αποτελείται από 3 πεδία:

- όνομα και είδος
- ιδιότητες
- μέθοδοι

Συνδυάζοντας αυτά τα σχήματα με βέλη και χρησιμοποιώντας και πρόσθετα ειδικά σύμβολα ώστε να πληροφορείται ο αναγνώστης για τις ιδιαίτερες ιδιότητες των μεταβλητών και των μεθόδων, προκύπτει εύκολα μια πλήρης εικόνα ολόκληρης της εφαρμογής.

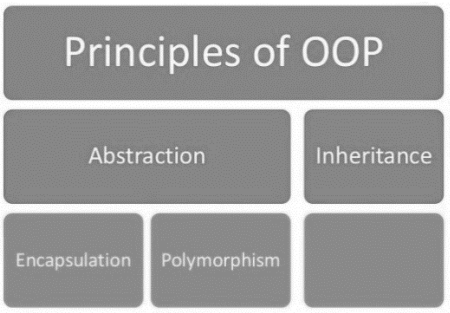
Στα δεξιά του Σχήματος 2.1 παρουσιάζονται σχηματικά 3 αντικείμενα που κατασκευάστηκαν με τη χρήση της κλάσης BankAccount. Τα αντικείμενα αυτά προέκυψαν κάνοντας μια απλή δήλωση όπως: BankAccount joeAcct; (δηλαδή με τον ίδιο ακριβώς τρόπο που θα δηλώναμε για παράδειγμα μια ακέραια μεταβλητή: int x;). Το πρόγραμμα ξέρει με αυτόν τον απλό τρόπο πως το αντικείμενο joeAcct είναι τύπου BankAccount. Έχει δεσμεύσει χώρο για 4 μεταβλητές, μια για κάθε ιδιότητα, οι οποίες προς το παρόν είναι κενές, και μπορούμε να προβούμε σε κάποια από τις ιδιότητες ή τις μεθόδους ενός αντικειμένου με τον χαρακτήρα της τελείας ".". Έτσι μπορούμε να πούμε απλά πως: joeAcct.accountNumber = "A7652"; ώστε να αναθέσουμε κωδικό στον συγκεκριμένο λογαριασμό ή να πούμε: joeAcct.withdraw(50); για να κάνουμε ανάληψη από τον λογαριασμό 50 ευρώ και να περιμένουμε πως θα μειωθεί το joeAcct.balance κατά 50.



Σχήμα 2.1 Το U.M.L. της κλάσης BankAccount και τρία αντικείμενα τύπου BankAccount: joeAcct, aliceAcct και samAcct.

Το αντικείμενο δηλαδή έχει δική του λογική, η οποία περιγράφεται στην κλάση με την οποία δημιουργήθηκε, την οποία σχεδιάσαμε εμείς ή σχεδίασαν κάποιοι άλλοι για μας. Οι σύγχρονες γλώσσες περιέχουν μια μεγάλη βιβλιοθήκη με έτοιμες κλάσεις που μπορούμε να χρησιμοποιήσουμε, χωρίς να γνωρίζουμε τον πραγματικό κώδικα που κρύβεται από πίσω τους. Το μόνο που χρειάζεται είναι να γνωρίζουμε τις ιδιότητες και τις μεθόδους τους. Να γνωρίζουμε

δηλαδή, τον τρόπο με τον οποίο τα αντικείμενα που θα φτιάξουμε με αυτήν την κλάση επικοινωνούν με το περιβάλλον τους, μιας και είναι αυτοτελείς οντότητες κατά κάποιο τρόπο.



Το ερώτημα που προκύπτει είναι γιατί να επιλέξουμε αυτόν τον τρόπο σχεδιασμού. Υπάρχουν 4 βασικές αρχές στον αντικειμενοστραφή σχεδιασμό, από τις οποίες προκύπτουν και τα πλεονεκτήματα του σε σχέση με τις παλιότερες προσεγγίσεις. Παρακάτω θα προσπαθήσουμε να τις περιγράψουμε συνοπτικά. Οι 4 βασικές αρχές είναι:

- *Αφαίρεση* [*Abstraction*]: Ένας τρόπος για να καταλάβουμε την ιδέα της αφαίρεσης, είναι να δανειστούμε κάποια στοιχεία από τη φιλοσοφία. Όταν μιλάμε για ένα ποτήρι ή ένα τραπέζι, όλοι ξέρουμε περί τίνος μιλάμε, τι σχετίζεται με αυτό, τι ιδιότητες έχει, χωρίς να ασχοληθούμε με περιττές πληροφορίες. Δεν πρόκειται ποτέ να πούμε πως το τραπέζι αυτό δεν έχει μηχανή, γιατί αυτό θα ήταν εντελώς περιττό, μιας και τα τραπέζια δεν έχουν μηχανές. Μιλώντας δηλαδή για την αφαίρεση "τραπέζι" έχουμε αυτόματα αποβάλει από το μυαλό μας καθετί άσχετο και μπορούμε να ασχοληθούμε με αυτά που έχουν πραγματικά νόημα.
- *Ενθυλάκωση* [*Encapsulation*]: Η ιδέα είναι να ξεχωρίσουμε τα πράγματα από το περιβάλλον τους, και να τα προστατέψουμε. Ένα απλό παράδειγμα είναι αυτό του παραπάνω σχήματος της κλάσης BankAccount. Δεν θέλουμε κάποιο άλλο μέρος του προγράμματος να μπορεί να μπει στην ιδιότητα balance και να την αλλάξει, χωρίς να χρησιμοποιήσει τη μέθοδο deposit() ή withdraw(). Το περιβάλλον δεν έχει άμεση πρόσβαση στα δεδομένα του αντικειμένου, αλλά έχει στις μεθόδους, οι οποίες έχουν πρόσβαση στα δεδομένα.
- *Κληρονομικότητα* [*Inheritance*]: Η ιδέα αυτή μας γλιτώνει από το να ξαναγράφουμε άχρηστο κώδικα. Ας υποθέσουμε ότι έχουμε δημιουργήσει μια κλάση Person η οποία έχει χαρακτηριστικά όπως name, email, phone και θέλουμε να δημιουργήσουμε μια κλάση Customers. Αντί να ξαναγράφουμε όλη την κλάση λέμε πως οι Customers κληρονομούν τους Persons, με την έννοια ότι και αυτοί θα έχουν name, email, phone, αλλά θα έχουν και customerNumber. Με τον ίδιο τρόπο φτιάχνουμε την κλάση Employees που κληρονομεί τους Persons και έχει και employeeID. Δημιουργούνται έτσι υπερκλάσεις και υποκλάσεις.
- *Πολυμορφισμός* [*Polymorphism*]: Είναι η ιδέα του να έχουμε τη δυνατότητα να περιμένουμε σωστά αποτελέσματα ακόμα και αν αλλάζουν οι μορφές αυτών που κάνουμε. Ένα απλό παράδειγμα είναι το σύμβολο +. Όταν το χρησιμοποιούμε με ακέραιους μας επιστρέφει το άθροισμα που είναι ακέραιος. Το ίδιο θα κάνει και με χαρακτήρες. Δηλαδή αν το χρησιμοποιήσουμε στο παράδειγμα "Hello"+"World" θα πάρουμε "HelloWorld". Ο πολυμορφισμός είναι η πιο περίπλοκη από τις τέσσερις αρχές και ίσως η πιο εξεζητημένη και γι' αυτό θα περιοριστούμε σε αυτό το παράδειγμα.

Αυτά είναι τα σημαντικότερα στοιχεία και οι αρχές πάνω στις οποίες θα αναπτύξουμε την εφαρμογή μας ώστε να επωφελείται από όλα αυτά τα χαρακτηριστικά, καθώς και να συμβαδίζει με τα σύγχρονα δεδομένα και τις πρακτικές που επικρατούν παγκοσμίως. Τις αναφέραμε

επιγραμματικά ώστε να μπορεί ο αναγνώστης να ανατρέξει γρηγορότερα στην βιβλιογραφία για ό,τι χρειαστεί.

2.2. Αφηρημένος Τύπος Δεδομένων ΑΤΔ

Αφού μιλήσαμε λίγο για τον αντικειμενοστραφή σχεδιασμό, μπορούμε πλέον να μιλάμε και για αντικείμενα με έναν πιο αφηρημένο τρόπο. Το δικαίωμα αυτό μας δόθηκε γιατί τα αντικείμενα είναι τώρα πια αυτόνομα. Έχουν ξεκάθαρα όρια μεταξύ τους, ξεκάθαρα χαρακτηριστικά και ξεκάθαρες λειτουργίες. Μας δίνεται η δυνατότητα να μπορούμε να μιλάμε λοιπόν πιο αφηρημένα και να απομακρυνθούμε από την εκάστοτε υλοποίηση, η οποία έχει σημαντικές διαφορές από γλώσσα σε γλώσσα.

Μπορούμε να πούμε πως οι οντότητες που ονομάζονται *Αφηρημένοι Τύποι Δεδομένων ΑΤΔ* [*Abstract Data Types ADT*], υπήρχαν πολύ πριν εξελιχθεί η επιστήμη της πληροφορικής. Απλά η θεώρησή τους έγινε με τη στροφή προς τον αντικειμενοστραφή σχεδιασμό. Τότε έγινε φανερό ότι μερικά αντικείμενα που δημιουργούσαν και χρησιμοποιούσαν οι προγραμματιστές, σε οποιαδήποτε γλώσσα προγραμματισμού κι να εργάζονταν, ήταν πρακτικά ίδια. Διέπονταν από τους ίδιους νόμους και περιλάμβαναν παρόμοια δεδομένα και μεθόδους. Έγινε προφανής λοιπόν η ανάγκη να οριστούν αυτές οι οντότητες σε ένα πιο αυθαίρετο επίπεδο, στο οποίο να οριστούν τα βασικά χαρακτηριστικά τους.

Ο όρος λοιπόν τύπος δεδομένων απορρέει από τον αντικειμενοστραφή σχεδιασμό και αναφέρεται σε δύο θέματα. Το πρώτο αφορά στο είδος δεδομένων: χαρακτήρας, ακέραιος, πραγματικός κλπ. και το δεύτερο στις ενέργειες που επιτρέπονται στα δεδομένα αυτά. Ο προσδιορισμός αφηρημένος γενικεύει τα προηγούμενα και σημαίνει τον τύπο δεδομένων πέρα από τις αναλυτικές προδιαγραφές του ή τον τρόπο υλοποίησής του. Ο ΑΤΔ αποτελεί βοηθητικό όρο στη σχεδίαση προγραμμάτων και μόνο ως βοηθητικό εργαλείο θεωρείται. Ο προγραμματιστής αναλογίζεται τις ενέργειες που θα υφίστανται τα δεδομένα, τις προσδιορίζει, τις αξιολογεί και τελικά επιλέγει τον κατάλληλο ΑΤΔ που θα αξιοποιήσει. Ύστερα συλλογίζεται τις λεπτομέρειες της αναπαράστασής του και τον προγραμματισμό των ενεργειών διαχείρισής τους.

Παραθέτουμε μια λίστα με τους πιο συνηθισμένους ΑΤΔ, οι οποίοι είναι χρήσιμοι σε ένα μεγάλο εύρος εφαρμογών:

Container	Multimap	Double-ended queue
List	Graph	Double-ended priority queue
Set	Stack	
Multiset	Queue	
Map	Priority queue	

Είναι σημαντικό να κατανοήσουμε τη διαφορά ανάμεσα στην *Αφαίρεση* [*Abstraction*] και στην *Υλοποίηση* [*Implementation*]. Η Αφαίρεση αποτελεί το θεωρητικό υπόβαθρο της εκάστοτε εφαρμογής. Μια σωστά δομημένη Αφαίρεση ενός προβλήματος δίνει όλες τις δυνατότητες για μια γρήγορη και αποδοτική Υλοποίηση. Παρόλα αυτά απαιτείται και η σωστή υλοποίηση του προβλήματος για ένα καλό αποτέλεσμα. Για παράδειγμα, δεν φτάνει απλώς να επιλεγούν οι ΑΤΔ

και οι αλγόριθμοι με τις καλύτερες θεωρητικές αποδώσεις, αλλά απαιτείται όλα αυτά να υλοποιηθούν με τις σωστές Δομές Δεδομένων, τις σωστές αλληλουχίες διαδικασιών και την αποφυγή των περιττών ενεργειών για να προκύψει το βέλτιστο αποτέλεσμα.

Στην παρούσα εργασία, δεν χρειάζεται να μιλήσουμε περαιτέρω για τις αφηρημένες οντότητες και με τους ορισμούς τους. Είναι πρακτικότερο να μιλάμε για το επόμενο επίπεδο, αυτό της Υλοποίησης. Σε επόμενο κεφάλαιο θα ασχοληθούμε με το πώς αυτές οι αφηρημένες οντότητες αποκτούν μορφή και γίνονται δεδομένα στον υπολογιστή καθώς και για τις διαφορές μεταξύ τους. Για να είμαστε σε θέση όμως να τις συγκρίνουμε μεταξύ τους, πρέπει πρώτα να ορίσουμε κάποια εργαλεία σύγκρισης.

2.3. Πολυπλοκότητα Αλγορίθμων

Στην επιστήμη της πληροφορικής μελετάμε τους αλγορίθμους που αντιμετωπίζουν τα διάφορα προβλήματα. Χρειαζόμαστε κάποια εργαλεία για να είμαστε σε θέση να τους συγκρίνουμε και να επιλέξουμε τον καλύτερο για την εκάστοτε εφαρμογή μας. Υπάρχουν μια σειρά από τέτοια θεωρητικά εργαλεία-δείκτες όπως: big O, big Θ, big Ω κ.α. Στην συνέχεια θα παρουσιάσουμε συνοπτικά το σημαντικότερο από αυτά τα εργαλεία, το οποίο θα μας χρειαστεί και στην συνέχεια της μελέτη μας.

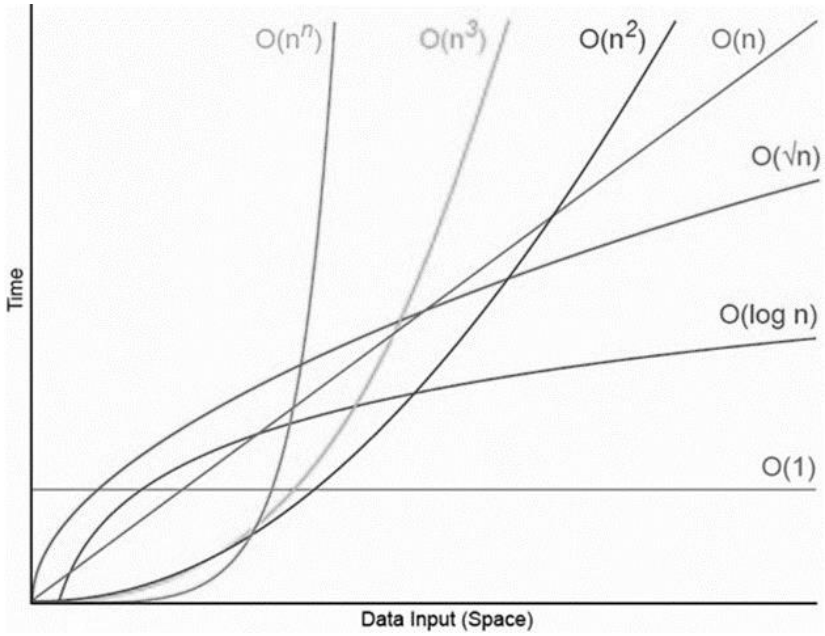
Το Σχήμα 2.2 παρουσιάζει μια γραφική παράσταση με τις πιο συνηθισμένες περιπλοκότητες του μεγάλου-Ο [Big-O complexity]. Ο πιο απλός ορισμός που μπορεί να δοθεί (καθώς όλα αυτά τα εργαλεία ανάλυσης προέρχονται από τα μαθηματικά και περιέχουν περίπλοκους ορισμούς και αποδείξεις) είναι ο εξής:

Ορισμός 2.1 Ο συμβολισμός του Big-O [Big-O notation] είναι μια σχετική αναπαράσταση της πολυπλοκότητας ενός αλγορίθμου.

Υπάρχουν μερικές σημαντικές και σκόπιμα επιλεγμένες λέξεις στον ορισμό αυτόν:

- σχετική: Μπορούμε να συγκρίνουμε μόνο μήλα με μήλα. Δεν μπορούμε να συγκρίνουμε έναν αλγόριθμο που κάνει αριθμητικό πολλαπλασιασμό με έναν που ταξινομεί μια λίστα ακεραίων για παράδειγμα. Αλλά μια σύγκριση των δύο αλγορίθμων που κάνουν και οι δύο αριθμητικές πράξεις (ένας πολλαπλασιασμό και ένας πρόσθεση) θα μας δώσει κάτι σημαντικό.
- αναπαράσταση: Το Big-O (στην απλούστερη μορφή του) μειώνει τη σύγκριση αλγορίθμων σε μια μόνο μεταβλητή. Αυτή η μεταβλητή επιλέγεται με βάση παρατηρήσεις ή παραδοχές. Για παράδειγμα, οι αλγόριθμοι ταξινόμησης συνήθως συγκρίνονται με βάση τις διαδικασίες σύγκρισης (σύγκριση δύο κόμβων ώστε να καθοριστεί η σχετική κατάταξή τους). Αυτό προϋποθέτει ότι η σύγκριση είναι "ακριβή" υπολογιστικά. Αλλά τι θα γινόταν αν η σύγκριση είναι "φθηνή", αλλά η εναλλαγή είναι "ακριβή"; Αλλάζει τη σύγκριση.

- πολυπλοκότητα: Αν μας παίρνει ένα δευτερόλεπτο για να ταξινομήσουμε 10.000 στοιχεία, πόσο θα μας πάρει για να ταξινομήσουμε ένα εκατομμύριο; Η πολυπλοκότητα σε αυτή την περίπτωση είναι ένα σχετικό μέτρο με κάτι άλλο



Σχήμα 2.2 Γραφική παράσταση στην οποία οι καμπύλες αναπαριστούν τις πιο συνηθισμένες πολυπλοκότητες αλγορίθμων big-O.

Θα αναφέρουμε ένα απλό παράδειγμα για να κατανοήσουμε τα παραπάνω. Το καλύτερο παράδειγμα για το Big-O είναι ίσως η εκτέλεση αριθμητικής. Ας πάρουμε δύο αριθμούς (123456 and 789012). Οι βασικές αριθμητικές διαδικασίες είναι οι εξής:

- πρόσθεση-αφαίρεση
- πολλαπλασιασμός-διαίρεση

Καθένα από τα παραπάνω είναι μια διαδικασία ή ένα πρόβλημα. Οι μέθοδοι επίλυσης των προβλημάτων αυτών ονομάζονται αλγόριθμοι.

Η πρόσθεση είναι η απλούστερη. Παρατάσσουμε τους αριθμούς (προς τα δεξιά) και προσθέτουμε τα ψηφία μια στήλης γράφοντας τον τελευταίο αριθμό του αποτελέσματος. Οι δεκάδες του εν λόγω αριθμού μεταφέρονται στην επόμενη στήλη. Ας υποθέσουμε ότι η πρόσθεση αυτών των αριθμών είναι η πιο ‘‘ακριβή’’ (υπολογιστικά) λειτουργία σε αυτόν τον αλγόριθμο. Είναι αυτονόητο πως για να προσθέσουμε αυτούς τους δύο αριθμούς θα πρέπει να προσθέσουμε συνολικά 6 ψηφία (και ενδεχομένως να μεταφέρουμε και ένα 7ο). Αν προσθέσουμε δύο αριθμούς 100 ψηφίων πρέπει να κάνουμε 100 προσθέσεις μεμονωμένων ψηφίων. Προφανώς η πολυπλοκότητα είναι ανάλογη με τον αριθμό των ψηφίων του μεγαλύτερου αριθμού n. Ονομάζουμε αυτή την περίπτωση γραμμική πολυπλοκότητα και την συμβολίζουμε με O(n). Η αφαίρεση είναι παρόμοιά (μόνο που μπορεί να χρειαστεί να δανειστούμε αντί να κουβαλήσουμε από μεγαλύτερη δεκάδα).

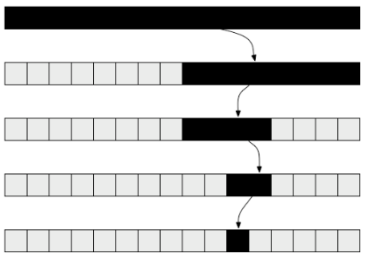
Ο πολλαπλασιασμός είναι διαφορετικός. Σε αυτόν παρατάσσουμε του αριθμούς όπως και πριν, παίρνουμε το $1^ο$ ψηφίο στον κάτω αριθμό και το πολλαπλασιάζουμε με καθένα από τα ψηφία του αριθμού από πάνω. Οπότε για να πολλαπλασιάσουμε τους δύο 6-ψηφιους αριθμούς μας πρέπει να πραγματοποιήσουμε 36 επιμέρους πολλαπλασιασμούς. Μετέπειτα μπορεί να χρειαστεί να προσθέσουμε 10 ή 11 στήλες για να λάβουμε το τελικό αποτέλεσμα. Αν είχαμε δυο 100-ψήφιους θα χρειαζόταν να πραγματοποιήσουμε 10000 πολλαπλασιασμούς και 200 προσθέσεις. Καθώς τα ψηφία γίνονται περισσότερα ο αλγόριθμος των επιμέρους πράξεων γίνεται ίσος με n-τετράγωνο, το οποίο ονομάζουμε τετραγωνική πολυπλοκότητα και το συμβολίζουμε με $O(n^2)$.

Να τονίσουμε εδώ πως μας ενδιαφέρει μόνο το πιο σημαντικό μέρος της πολυπλοκότητας. Για παράδειγμα στην τελευταία περίπτωση ενώ προέκυψαν n^2+2n επιμέρους πράξεις, αμελήσαμε εντελώς τον όρο 2n. Αυτό γιατί αν χρειαστεί για παράδειγμα να γίνει πολλαπλασιασμός δύο αριθμών με 1 εκατομμύριο ψηφία έκαστος, θα χρειαστούν 10^{12} επιμέρους πολλαπλασιασμοί και 2 εκατομμύρια προσθέσεις. Δηλαδή οι προσθέσεις αποτελούν το 0.0002% των συνολικών διαδικασιών.

Μια άλλη παρατήρηση είναι ότι υποθέσαμε τη χειρότερη περίπτωση. Υπάρχει για παράδειγμα η περίπτωση κατά την οποία πολλαπλασιάζεται ένας 6-ψηφιος με έναν 4-ψήφιο αριθμό. Τότε θα είχαμε μόνο 24 πολλαπλασιασμούς ψηφίων. Παρόλα αυτά εμείς υπολογίζουμε πάντα την περιπλοκότητα σύμφωνα με το χειρότερο σενάριο, δηλαδή όταν και οι δύο αριθμοί έχουν n ψηφία στην περίπτωσή μας. Δηλαδή ο συμβολισμός Big-O αφορά στη χειρότερη περίπτωση ενός αλγορίθμου.

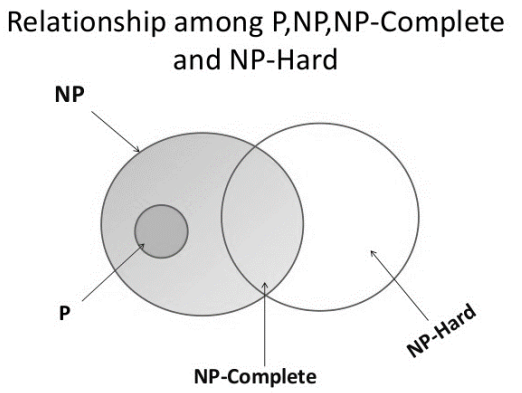
Μια άλλη περίπτωση από αυτές που αναφέραμε είναι αυτή της λογαριθμικής πολυπλοκότητας O(logn). Θα τη μελετήσουμε παρουσιάζοντας ένα ακόμα χαρακτηριστικό παράδειγμα, αυτό της *Δυαδικής Αναζήτησης* [*Binary Search*].

Η δυαδική αναζήτηση είναι μια ειδική περίπτωση αναζήτησης που απαιτεί τα δεδομένα να είναι *ταξινομημένα* [*sorted*]. Ο αλγόριθμος πίσω της είναι πολύ απλός. Ας υποθέσουμε ότι θέλουμε να βρούμε τη θέση στην οποία βρίσκεται η τιμή x σε έναν πίνακα που έχει ταξινομημένες τιμές από τη μικρότερη προς τη μεγαλύτερη. Με την απλή αναζήτηση θα εξετάζαμε μία-μία τις τιμές με το x, και με βάση το χειρότερο σενάριο (αυτό στο οποίο το x βρίσκεται στην τελευταία θέση του πίνακα), προκύπτει η γραμμική πολυπλοκότητα O(n).



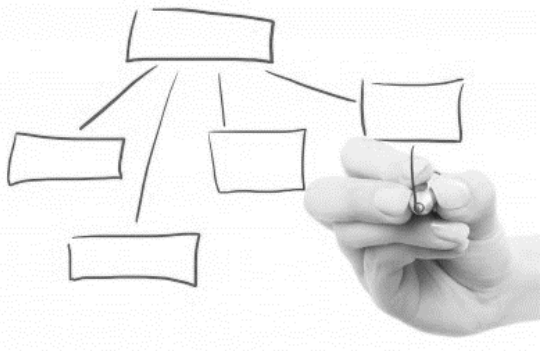
Μιας όμως και τα στοιχεία είναι ταξινομημένα μπορούμε απευθείας να συγκρίνουμε το x με την τιμή που περιέχεται στη μεσαία θέση. Ανάλογα με τη σύγκριση "διαγράφουμε" την πλευρά του πίνακα που δεν χρειαζόμαστε και επαναλαμβάνουμε τη διαδικασία μέχρι να βρούμε τη σωστή θέση. Αν είχαμε για παράδειγμα έναν πίνακα 16 θέσεων, θα χρειαζόμασταν 4 συγκρίσεις. Αν είχαμε όμως έναν πίνακα 1.000.000 θέσεων, θα χρειαζόμασταν μόλις 20 συγκρίσεις. Λέμε λοιπόν πως ο αλγόριθμος αυτός έχει λογαριθμική πολυπλοκότητα και γράφουμε O(logn). Με το logn εννοούμε πάντα $\log_2 n$ (π.χ. $\log_2 n(1000000) = 19.931569 \rightarrow 20$). Η δυαδική αναζήτηση είναι εξαιρετικά γρήγορη μιας και εκμεταλλευόμαστε την πληροφορία πως ο πίνακας είναι ταξινομημένος.

Ένα τελευταίο σημαντικό σημείο είναι η *πολυωνυμική περιπλοκότητα* [*polynomial complexity*] η οποία συμβολίζεται με $O(n^2)$, $O(n^3)$,… Η κλάση αυτών των προβλημάτων συμβολίζεται με το γράμμα P: polynomial. Προφανώς τα προβλήματα που έχουν τέτοια περιπλοκότητα, χαρακτηρίζονται ως δύσκολα. Υπάρχουν όμως και προβλήματα που δεν μπορούν να λυθούν ούτε σε πολυωνυμικό χρόνο. Τα προβλήματα αυτά ανήκουν στην κλάση NP: Non-deterministic polynomial. Η κατηγορία αυτών των προβλημάτων αποτελείται κυρίως από εκθετικά όπως $O(\alpha^n)$. Μια ακόμα πιο δύσκολη κατηγορία είναι αυτή των NP-hard προβλημάτων, και ο συγκεκριμένος όρος εκφράζει πως τα προβλήματα αυτά είναι τουλάχιστον NP (αλλά ίσως και ακόμα πιο δύσκολα). Τέλος υπάρχουν τα NP-complete, για τα οποία δεν έχει βρεθεί (πρακτικά) λύση και δεν γνωρίζουμε με βεβαιότητα αν είναι άλυτα. Αν όμως βρεθεί (ή αποδειχθεί ότι υπάρχει) λύση για ένα NP-complete πρόβλημα, τότε γνωρίζουμε πως θα υπάρχει για όλα. Αυτό το χαρακτηριστικό είναι που του δίνει και μεγάλο ερευνητικό ενδιαφέρον, μιας και αποτελεί πρόκληση. Το διπλανό σχήμα παρουσιάζει ένα διάγραμμα Venn το οποίο περιγράφει τη σχέση των προβλημάτων αυτών ως σύνολα.



2.4. Δομές Δεδομένων

Για την υλοποίηση των ΑΤΔ που αναφέραμε στην Ενότητα 2.2 χρησιμοποιούμε τις *Δομές Δεδομένων* [*Data Structures*]. Μια Δομή Δεδομένων είναι ένας συγκεκριμένος τρόπος οργάνωσης δεδομένων σε έναν υπολογιστή ώστε να μπορούν να χρησιμοποιηθούν αποδοτικά. Μια Δομή Δεδομένων μπορεί να χρησιμοποιηθεί για την υλοποίηση ενός ή περισσοτέρων Αφηρημένων Τύπων Δεδομένων.



Ένα ερώτημα είναι γιατί χρησιμοποιούμε δομές δεδομένων. Η απάντηση είναι γιατί βολεύει. Για την ακρίβεια αυτό κάνουμε συνέχεια και στον πραγματικό κόσμο. Η έννοια του τραπεζικού λογαριασμού για παράδειγμα, που περιέχει ένα ονοματεπώνυμο, έναν κωδικό, ένα νούμερο για το ποσό χρημάτων κ.τ.λ., υπήρχε πολύ πιο πριν μπουν στη ζωή μας οι ηλεκτρονικοί υπολογιστές. Γενικότερα, η πλειονότητα των δεδομένων γύρω μας παρουσιάζεται σε μια δομημένη μορφή, και είναι βολικό για μας να τη διαχειριζόμαστε και στον κόσμο της πληροφορικής με αυτή τη μορφή.

Δεν είναι όμως μόνο αυτό. Όταν μιλάμε για δομές δεδομένων, ασχολούμαστε με πέντε βασικές λειτουργείες:

- Πώς θα έχουμε *πρόσβαση* στα δεδομένα [*access*]
- Πώς θα *εκχωρούμε* δεδομένα [*insert*]
- Πώς θα *αφαιρούμε* δεδομένα [*delete*]
- Πώς θα *βρίσκουμε* δεδομένα [*find*]
- Πώς θα *ταξινομήσουμε* τα δεδομένα [*sort*]

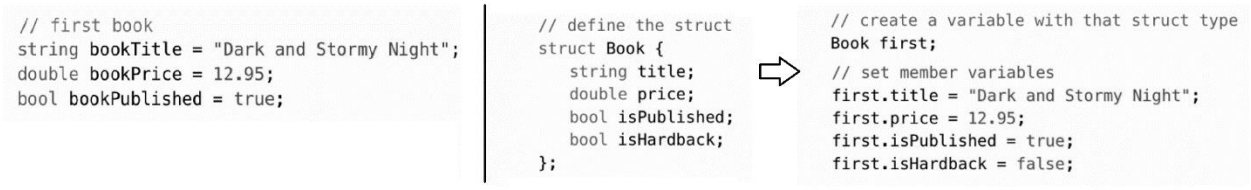
Πίσω από τη λέξη "πώς" εννοούμε "πόσο γρήγορα". Ανάλογα με τις απαιτήσεις απόδοσης μας σε κάποιες από τις παραπάνω λειτουργείες, επιλέγουμε και την κατάλληλή δομή. Προφανώς δε γίνεται να κατασκευάσουμε δομές που να ανταποκρίνονται καλά σε όλες τις παραπάνω λειτουργίες.

Παρακάτω, θα παρουσιάσουμε σε λίγες παραγράφους τις πιο θεμελιώδεις δομές δεδομένων και τα βασικά χαρακτηριστικά τους ώστε να είμαστε σε θέση να κατανοήσουμε πώς γεννήθηκαν όλα αυτά και τι θα μας χρησιμεύσει στην εφαρμογή μας στην συνέχεια της εργασίας μας. Πολλές δομές έχουν ήδη υλοποιηθεί και μετατραπεί σε κλάσεις από όλες σχεδόν τις σύγχρονες γλώσσες προγραμματισμού. Με άλλα λόγια είναι έτοιμες για χρήση από τον προγραμματιστή.

2.4.1. Δομή

Η πιο απλή δομή δεδομένων είναι η *δομή* [*struct*], η οποία εμφανίστηκε με τη γλώσσα C πριν 40 χρόνια και εξακολουθεί να υποστηρίζεται και στις σύγχρονες *βασισμένες-στη-C* [*C-based*] γλώσσες προγραμματισμού εξαιτίας της απλότητάς της.

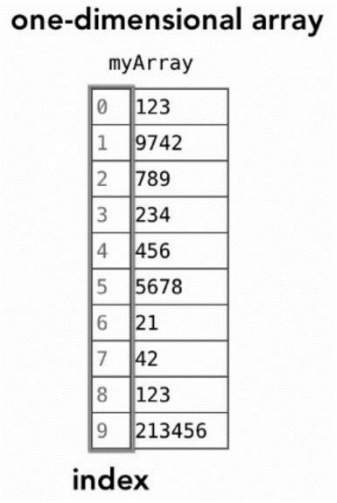
Η struct είναι απλά ένα πακέτο μεταβλητών. Η ιδέα είναι πως αντί να κατασκευάζουμε ξεχωριστά μεταβλητές όπως στο παράδειγμα που ακολουθεί, μπορούμε να τις πακετάρουμε και να τους δώσουμε ένα όνομα και να κατασκευάζουμε όσα αντικείμενα χρειαζόμαστε εύκολα. Έπειτα μπορούμε να προβαίνουμε σε κάθε μεταβλητή (ή ιδιότητα) κάθε αντικειμένου με τον χαρακτήρα τελεία ".".



Η struct διαφέρει πολύ από μια class που φτιάχνουμε εμείς. Ουσιαστικά αποτελεί την αρχική ιδέα της class. Υπάρχουν πολλές διαφορές μεταξύ τους αλλά η βασικότερη είναι πως με τη struct δεν μπορούμε να ορίσουμε καμιά συμπεριφορά που θα έχει το αντικείμενο που θα δημιουργηθεί, παρά μόνο data(μεταβλητές). Γενικότερα θα μπορούσαμε να πούμε πως η struck είναι η απλούστερη class που μπορούμε να δημιουργήσουμε.

2.4.2. Πίνακας

Μονοδιάστατος Πίνακας



Ο *πίνακας* [*array*] είναι η πιο θεμελιώδης και η πιο διαδεδομένη δομή ανάμεσα σε όλες τις γλώσσες. Μια array είναι μια διατεταγμένη συλλογή από αντικείμενα. Πολλές ανεξάρτητες αξίες κλείνονται μέσα σε ένα δοχείο με όνομα. Το δοχείο έχει όνομα, αλλά όχι τα αντικείμενα μέσα του. Πρόσβαση σε αυτά έχουμε με το όνομα του αντικειμένου και με μια μεταβλητή δείκτη που ξεκινάει από το 0 και φτάνει έως το (πλήθος των θέσεων-1) και αντιπροσωπεύει τη θέση του εκάστοτε αντικειμένου.

Η array έχει κάποιους βασικούς περιορισμούς:

- Είναι σταθερού μεγέθους: Τη στιγμή που δημιουργούμε μια array, το σύστημα δεσμεύει συγκεκριμένο αριθμό θέσεων μνήμης ανάλογα με το τι του ζητάμε, το οποίο δε γίνεται να αλλάξει. Μια array 10 θέσεων θα είναι πάντα 10 θέσεων.
- Είναι συγκεκριμένου τύπου δεδομένων: Τη στιγμή της δημιουργίας της, εκτός από το μέγεθός της, ορίζουμε και τον τύπο δεδομένων που θα περιέχουν οι θέσεις της και αυτός είναι μοναδικός και δεν μπορεί να αλλάξει κατά τη διάρκεια της ζωή της. Με απλά λόγια μια array μπορεί να περιέχει μόνο ακεραίους [int] για παράδειγμα, ή μόνο χαρακτήρες [char], ή μόνο αλφαριθμητικές εκφράσεις [string]. Επίσης μπορεί να περιέχει αντικείμενα δικής μας κλάσης, για παράδειγμα αν έχουμε κάπου φτιάξει μια class books; τότε μπορούμε να ζητήσουμε από την array να κρατήσει χώρο ώστε κάθε θέση αντικειμένου να χωράει ένα αντικείμενο τύπου books.

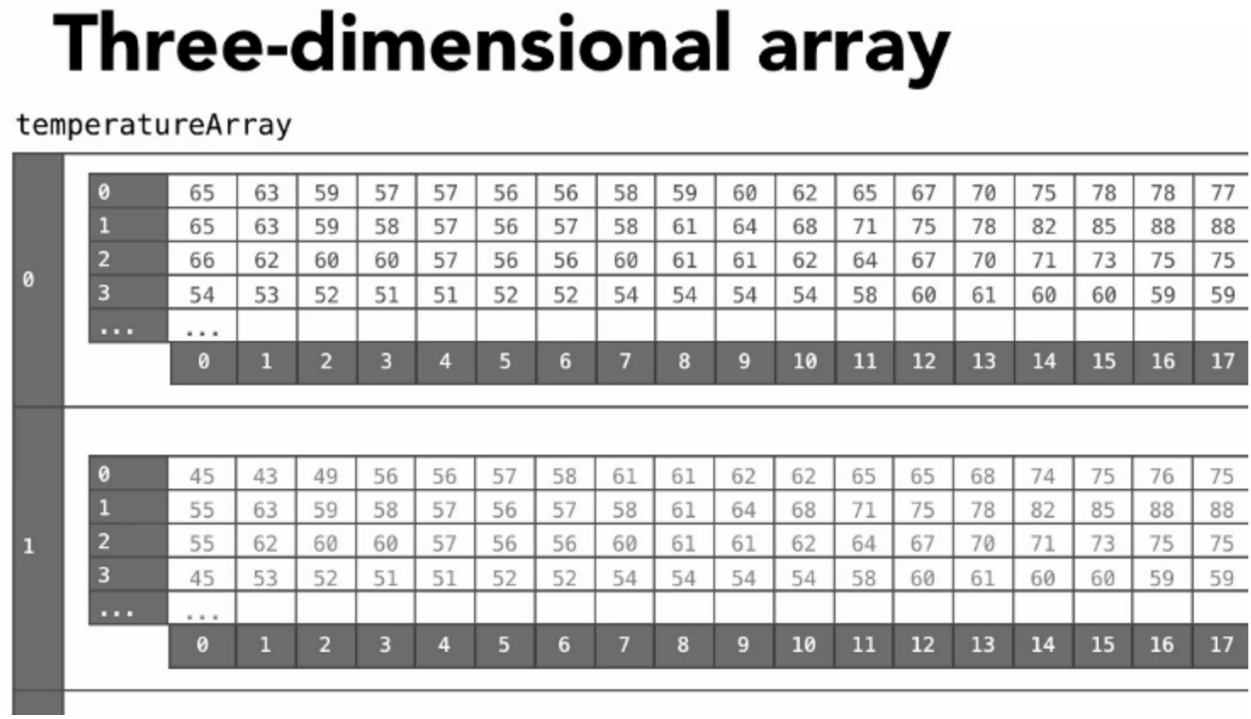
Είναι φανερό το πόσο περιορισμένες δυνατότητες προσφέρει μια τέτοια δομή, αλλά αυτό δεν είναι πάντα κακό. Γενικότερα σε όλες τις δομές δεδομένων, όσο περισσότερους νόμους βάζεις πάνω τους, δηλαδή όσο πιο πολύ τις περιορίζεις, τόσο γρηγορότερες και αποδοτικότερες θα είναι σε αυτά που θα σου προσφέρουν. Θα πούμε περισσότερα στη συνέχεια.

Πολυδιάστατος Πίνακας

Με τις ίδιες αρχές, μπορούμε να κατασκευάσουμε *πολυδιάστατους πίνακες* [*multidimensional arrays*]. Αυτοί θα έχουν τα ίδια χαρακτηρίστηκα με αυτούς της μιας διάστασης. Εκείνο που αξίζει να αναφέρουμε είναι η λογική πίσω από αυτή τη διάταξή.

Μιλώντας συνήθως για 3-διάστατους πίνακες, φανταζόμαστε κάποιο κύβο, χωρισμένο με τη λογική της μέτρησης όγκου m^3-cm^3-mm^3. Αυτή η ιδέα μας είναι έμφυτη από τα μαθηματικά, αλλά στην περίπτωσή μας είναι εντελώς άχρηστή. Αντίθετα, πρέπει να σκεφτόμαστε έναν 3-διάστατο πίνακα ως: έναν πίνακα που περιέχει πίνακες, όπως παρουσιάζεται και στο σχήμα. Ο λόγος είναι απλά γιατί εδώ εργαζόμαστε με δεδομένα, και δεν υπάρχει κανένας λόγος για πολύπλοκες αναπαραστάσεις. Ένα απλό παράδειγμα είναι αυτό του καταλόγου τηλεφώνων: ο

κατάλογος είναι ένας πίνακας(Χ θέσεων ανάλογα με τις πόλεις που αφορά) που περιέχει πίνακες(24 θέσεων Α-Ω) που περιέχει πίνακες(3 θέσεων: Ονοματεπώνυμο, Τηλέφωνο, Διεύθυνση). Γενικότερα είναι πιο προσιτή η ιδέα του δέντρου από αυτήν των πινάκων μιας και έχουμε να κάνουμε με στατικά δεδομένα.

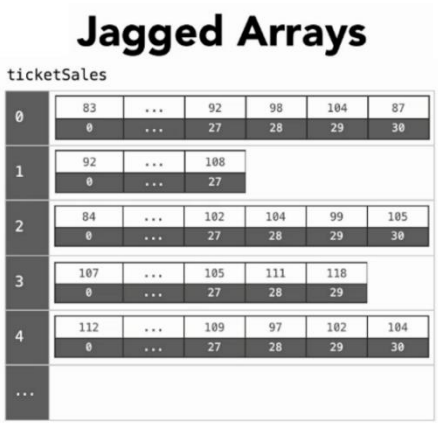


Τέλος, πρέπει να σημειώσουμε πως στον αντικειμενοστραφή σχεδιασμό αποφεύγουμε τους πολυδιάστατους πίνακες. Τους χρησιμοποιούμε μόνο σε ορισμένες περιπτώσεις που απαιτείται σαφήνεια και αμεσότητα. Αντί για αυτούς προτιμάμε να φτιάξουμε για παράδειγμα ένα μονοδιάστατο πίνακα με κάποια class. Για παράδειγμα θα μπορούσαμε να φτιάχναμε μια class grade, η οποία θα περιείχε μεταβλητές για κάθε μάθημα του εξαμήνου, και μετά θα μπορούσαμε να φτιάξουμε μια μονοδιάστατη array από grade, στην οποία κάθε στοιχείο θα αντιπροσώπευσε κάθε μαθητή. Έτσι αποφύγαμε τη δημιουργία πολυδιάστατου πίνακα καθώς είναι επιρρεπής σε λάθη.

Ακανόνιστος Πίνακας

Η μοναδική διαφορά ανάμεσα σε μια jagged και μια κανονική multidimensional array, είναι πως ο αριθμός του μεγέθους των πινάκων που αποτελούν αντικείμενα του βασικού πίνακα ποικίλλει από θέση σε θέση.

Η δομή αυτή δεν υποστηρίζεται συνήθως αυτόματα από τις γλώσσες, γιατί έχει νόημα μόνο όταν υπάρχει κάποια λογική από πίσω της, όπως για παράδειγμα ένα ημερολόγιο: Μήνες[Μέρες] ώστε να έχουμε διαφορετικό αριθμό μερών σε κάθε θέση.



Δυναμικός Πίνακας

Συνήθως στις υλοποιήσεις μας χρειαζόμαστε να έχουμε τη δυνατότητα αλλαγής μεγέθους ενός πίνακα που δημιουργούμε. Επίσης θέλουμε να έχουμε εύχρηστες και έτοιμες προς χρήση λειτουργίες όπως sort ή search. Έτσι χρησιμοποιούμε πίνακες με περισσότερη ευφυΐα και έτσι εμφανίζονται και οι παράγοντες απόδοσης.

Οι πίνακες που είδαμε μέχρι τώρα χαρακτηρίζονται ως στατικοί. Ο δυναμικός πίνακας, έχει παρόμοια συμπεριφορά με τον στατικό, με την επιπλέον ιδιότητα της δυνατότητας πρόσθεσης στοιχείου/ων. Χωρίς να ασχοληθούμε με την υλοποίηση που κρύβεται πίσω από τη δομή αυτή, μπορούμε να έχουμε στο μυαλό μας πως η δομή κρατά μια θέση με ένα μετρητή και έναν πίνακα με διευθύνσεις μνήμης. Κάθε φορά που του ζητούμε να προσθέσει ένα στοιχείο αυξάνει το μετρητή και τροποποιεί τις διευθύνσεις.

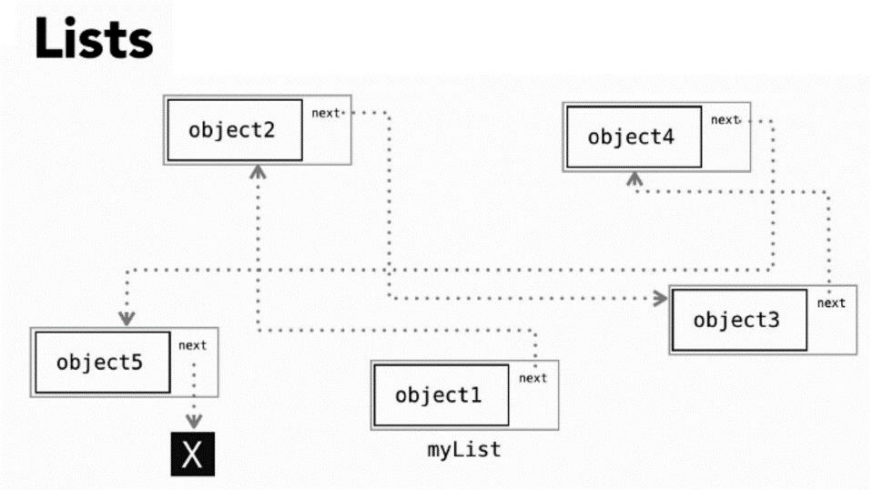
Πρέπει να σημειώσουμε πως έχει τεράστια διαφορά να ζητήσουμε από τη δομή να προσθέσει ένα στοιχείο στο τέλος της [append ή add] από το να της ζητήσουμε να το εκχωρήσει σε κάποια συγκεκριμένη θέση της [insert]. Τόσο που, αν διαπιστώσει ότι κατά την εισαγωγή έχει μπροστά της πολλές μετακινήσεις στοιχείων, ίσως προτιμήσει να δημιουργήσει έναν καινούριο πίνακα με ίδιο όνομα και να διαγράψει τον παλιό, χωρίς να το μάθει ποτέ ο προγραμματιστής. Παρόμοια συμπεριφορά παρουσιάζεται και με τη διαδικασία της αφαίρεσης [remove]. Όλα αυτά βέβαια έχουν να κάνουν με τις σύγχρονες ευφυείς υλοποιήσεις που βρίσκονται πίσω από τις σημερινές γλώσσες προγραμματισμού και τα πακέτα βιβλιοθηκών που προσφέρουν.

Το βασικότερο χαρακτηριστικό των arrays είναι ότι προσφέρουν *απευθείας πρόσβαση* [*direct access*] μιας και κάθε στοιχείο έχει μια συγκεκριμένη θέση με ένα συγκεκριμένο *δείκτη* [*index*].

2.4.3. Λίστα

Ο βασικός αντίπαλος της array είναι η *λίστα* [*list*]. Η αναπαράσταση μιας list φαίνεται στο σχήμα που ακολουθεί. Η ιδέα είναι να κρατάμε πίσω από κάθε αντικείμενο έναν δείκτη που να μας λέει τη διεύθυνση μνήμης του επόμενου αντικειμένου της λίστας, και για το τελευταίο αντικείμενο να μας στέλνει σε ένα αντικείμενο τερματισμού X [dummy object]. Για να είμαστε ακριβείς η list αυτή αποτελεί μια singly linked list, μιας και κάθε στοιχείο είναι συνδεδεμένο με το επόμενο. Αντίστοιχα υπάρχει και η δομή δεδομένων doubly liked list, στην οποία κάθε στοιχείο έχει ακόμα μια πληροφορία: την διεύθυνση του προηγούμενου στοιχείου του, αλλά δεν θα ασχοληθούμε περεταίρω με αυτή.

Η βασικότερη διαφορά ανάμεσά στις arrays και στις lists είναι στη λειτουργεία της πρόσβασης. Λέμε ότι οι arrays προσφέρουν *απευθείας πρόσβαση* [*direct access*], ενώ οι lists *διαδοχική πρόσβαση* [*sequential access*]. Πιο απλά, αν ζητήσουμε από την εφαρμογή να μας επιστρέψει για παράδειγμα το αντικείμενο object4 της παρακάτω λίστας και να το τυπώσει στην οθόνη, θα ξεκινήσει από το object1 και θα περιφερθεί μέχρι να το βρει. Στην χειρότερη περίπτωση το αντικείμενο θα βρίσκεται στο τέλος και οπότε προκύπτει πολυπλοκότητα O(n) σε αντίθεση με την απευθείας διαδικασία O(1) της array.



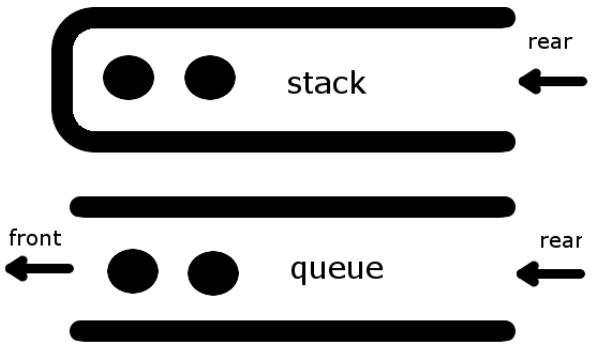
Η list όμως παρουσιάζει και κάποια σημαντικά προτερήματα. Στην περίπτωση εκχώρησης αντικειμένου [insert], το μόνο που έχει να κάνει είναι να "κόψει" τη σύνδεση που χρειάζεται, να εκχωρήσει το αντικείμενο που θέλουμε και να αλλάξει μερικούς δείκτες. Η διαδικασία της εισαγωγής όπως και της αφαίρεσης γίνεται σε σταθερό χρόνο O(1). Όσον αφορά για τη λειτουργία ταξινόμησης [sort], καμιά από τις δύο δομές δεν είναι καλή, λόγω της εξαιρετικής πολυπλοκότητας της λειτουργίας. Ο Πίνακας 2.1 συνοψίζει τα χαρακτηριστικά των δύο αυτών δομών δεδομένων.

	Arrays	Lists
Direct Access	**GOOD** fixed time O(1)	**POOR** linear time O(*n*)
Adding / Removing	**POOR** linear time O(*n*)	**GOOD** fixed time O(1)
Searching	O(*n*) linear search O(log *n*) binary search	O(*n*) - linear search

Πίνακας 2.1 Σύγκριση της πολυπλοκότητας βασικών διαδικασιών ανάμεσα στις δομές array και list.

Αξίζει να σημειώσουμε μια σημαντική διαφορά που παρουσιάζεται στον Πίνακα 2.1 στη διαδικασία εύρεσης. Η διαφορά είναι ότι στις arrays μπορούμε να τρέξουμε δυαδική αναζήτηση [binary search] ενώ στις lists όχι.

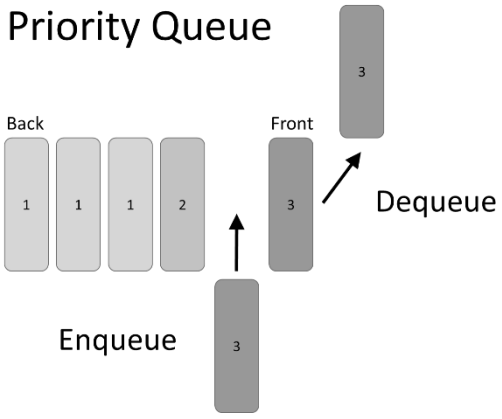
2.4.4. Στοίβα και Ουρά



Οι στοίβες και οι ουρές στον κόσμο της πληροφορικής έχουν ακριβώς την ίδια λογική με αυτήν του πραγματικού κόσμου. Η στοίβα, σαν μια στοίβα με πιάτα, υλοποιεί την ιδιότητα “LIFO-Last In First Out”. Η ουρά αντίστοιχα, σαν μια ουρά ανθρώπων σε κάποιο ταμείο, υλοποιεί την ιδιότητα “FIFO-First In First Out”.

Στις δομές αυτές δεν έχουμε δείκτες όπως στις array. Για την ακρίβεια δεν μας ενδιαφέρει τίποτα άλλο από το να εκμεταλλευόμαστε την παραπάνω ιδιότητα που τις διακρίνει. Το μόνο που μας ενδιαφέρει σε αυτές τις δομές είναι να βάζουμε και να βγάζουμε αντικείμενα. Αν χρειαζόμαστε παραπάνω λογική, μάλλον βρισκόμαστε σε λάθος δομή δεδομένων ή πρέπει να επανεξετάσουμε τις ανάγκες μας.

2.4.5. Ουρά Προτεραιότητας



Μια ειδική περίπτωση ουράς είναι αυτή της ουράς προτεραιότητας. Τη μελετάτε ως ξεχωριστή δομή δεδομένων και ΑΤΔ γιατί έχει μεγάλη χρησιμότητα στην εφαρμογή μας, και γιατί κρύβει από πίσω της περισσότερη λογική από μια απλή FIFO ουρά.

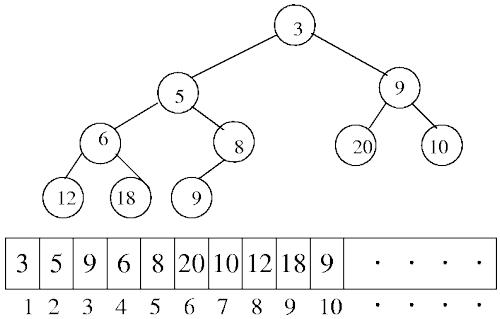
Η ουρά προτεραιότητας διατάσσει τα στοιχεία που περιέχει σύμφωνα με ένα χαρακτηριστικό. Χρειάζεται λοιπόν να δώσουμε στη δομή έναν τρόπο με τον οποίο θα τα συγκρίνει. Ο τρόπος αυτός συνήθως λέγεται comparator. Στην πιο απλή μορφή του θα είναι αν θα επιλέγει να "προχωρήσει" θέση στην ουρά τη μικρότερη ή τη μεγαλύτερη τιμή, αλλά μπορεί να γίνει όσο περίπλοκο χρειαζόμαστε για την εφαρμογή μας.

Η σημαντικότερη ιδιότητα μιας ουράς προτεραιότητας είναι πως τα στοιχεία της είναι ταξινομημένα ανά πάσα στιγμή. Μπορούμε δηλαδή να είμαστε πάντα βέβαιοι πως το πρώτο της στοιχείο είναι το βέλτιστο. Για να επιτευχθεί αυτό, κατά τη διαδικασία εκχώρησης ενός στοιχείου στην δομή αυτή, πραγματοποιείται αυτόματα και μια διαδικασία ταξινόμησης, ώστε το πρόγραμμα να μην προχωρήσει αν δεν επανέλθει η τάξη ως προς την προτεραιότητα των στοιχείων της δομής.

2.4.6. Σωρός

Γενικά οι σωροί είναι μια εξειδίκευση του ΑΤΔ Δέντρου. Ένα απλό παράδειγμα σωρού είναι ο *Δυαδικός Σωρός* [*Binary Heap*]. Όπως απεικονίζεται και στο διπλανό σχήμα, πρόκειται ουσιαστικά για ένα δέντρο με τις εξής βασικές ιδιότητες:

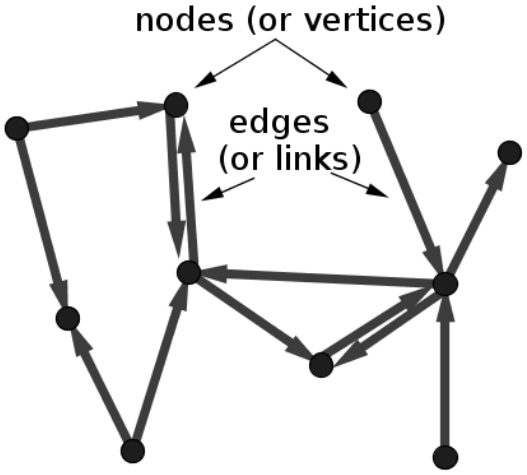
- Κάθε κόμβος-γονέας από τα δεξιά προς τα αριστερά πρέπει να έχει 2 κόμβους-παιδιά(εκτός από τον τελευταίο)
- Η τιμή του γονέα πρέπει να είναι μικρότερη(ή μεγαλύτερη) από αυτή των παιδιών



Η δεύτερη ιδιότητα μας δίνει την εξής δυνατότητα: όταν ο κόμβος-παιδί έχει μεγαλύτερη(ή μικρότερη) τιμή από το γονέα, αλλάζουμε μεταξύ τους τις τιμές. Με διαδοχικές τέτοιες εναλλαγές προκύπτει η ελάχιστη(ή μέγιστη αντίστοιχα) τιμή στην κορυφή του δέντρου χωρίς όμως να είναι όλες οι τιμές ταξινομημένες, όπως φαίνεται και στο διπλανό σχήμα. Έτσι χρησιμοποιώντας τη δομή δυαδικού σωρού εξοικονομούμε κάποιο χρόνο τρεξίματος, σε σύγκριση με τον απαιτούμενο χρόνο μιας ουράς προτεραιότητας.

2.4.7. Γράφος

Λόγω της φύσεως της παρούσας εργασίας, θα επικεντρωθούμε περισσότερο στην έννοια του γράφου. Στον τομέα των *Διακριτών Μαθηματικών* [*Discrete Mathematics*], και πιο συγκεκριμένα στη *Θεωρία Γράφων* [*Graph Theory*], ένας γράφος είναι μια αναπαράσταση ενός συνόλου αντικειμένων, όπου ορισμένα ζεύγη αντικειμένων συνδέονται μεταξύ τους με δεσμούς. Τα διασυνδεδεμένα αντικείμενα αντιπροσωπεύονται από τις μαθηματικές αφαιρέσεις που ονομάζονται κορυφές (που ονομάζονται επίσης κόμβοι ή σημεία), και οι δεσμοί που συνδέουν ορισμένα ζεύγη κορυφών ονομάζονται ακμές (ονομάζονται επίσης τόξα ή γραμμές). Συνήθως ένας γράφος απεικονίζεται σε διαγραμματική μορφή ως ένα σύνολο κουκκίδων για τις κορυφές, που ενώνονται με γραμμές ή καμπύλες για τις ακμές. Οι ακμές μπορεί να είναι κατευθυνόμενες ή μη κατευθυνόμενες. Έτσι προκύπτουν οι δύο τύποι γράφων: κατευθυνόμενος και μη-κατευθυνόμενος.



Πιο συγκεκριμένα ένας *κατευθυνόμενος γράφος* [*directed graph*] συμβολίζεται συνήθως με το γράμμα G, και αποτελείται από ένα σύνολο *κόμβων* [*nodes*] V (που ονομάζονται επίσης και *κορυφές* [*vertices*]) και ένα σύνολο προσανατολισμένων *ακμών* [*edges*] $E \subseteq V \times V$ (που ονομάζονται επίσης και *συνδέσεις* [*links*]). Ο αριθμός των κόμβων συμβολίζεται με n και ο αριθμός των ακμών με m, αντίστοιχα. Μια ακμή έχει τη μορφή (u, v), όπου u, v $\in$ V είναι δύο διακριτοί κόμβοι (δηλαδή, οι ακμές είναι διατεταγμένα ζεύγη κόμβων). Λιγότερο συχνά, χρησιμοποιούνται μη προσανατολισμένοι γράφοι, αν η κατεύθυνση των ακμών δεν είναι σημαντική. Σε αυτήν την περίπτωση, μία ακμή έχει τη μορφή {u, v} και είναι ένα υποσύνολο των κόμβων που περιέχει ακριβώς δύο κόμβους.

Σε κάθε ακμή έχει εκχωρηθεί ένα μήκος, το οποίο ως επί το πλείστον θα είναι ένας φυσικός αριθμός για την εφαρμογή μας. Μια διαδρομή είναι μια ακολουθία $v_1, e_1, v_2, ..., e_{k-1}, v_k$ κόμβων και ακμών, τέτοια ώστε για κάθε i ($1 \leq i < k$), η ακμή e_i να συνδέει τα v_I και v_{i+1}: $e_i = (v_i, v_{i+1})$. Για απλούς γράφους (δηλαδή, γράφους που δεν περιέχουν βρόγχους και παράλληλες ακμές), η διαδρομή έχει ήδη οριστεί μονοσήμαντα από την ακολουθία των κόμβων. Το μήκος μιας διαδρομής είναι το άθροισμα των μηκών όλων των ακμών της διαδρομής.

Στην επιστήμη της πληροφορικής ο Γράφος είναι ένας ΑΤΔ ο οποίος προορίζεται να υλοποιήσει τις έννοιες του κατευθυνόμενου και μη κατευθυνόμενου γράφου από τα μαθηματικά.

Ο ορισμός του γράφου είναι γενικός και περιλαμβάνει μόνο ελάχιστα χαρακτηριστικά. Σε αυτό έγκειται και η δυσκολία μελέτης του. Ο γράφος είναι αφηρημένος τύπος δεδομένων και οι προγραμματιστές προβαίνουν σε εξειδικευμένες υλοποιήσεις έτσι ώστε να κατασκευάσουν το γράφο που τους βολεύει για την επίλυση της εκάστοτε εφαρμογής. Λόγω της γενικότητας αυτής, οι γλώσσες προγραμματισμού δεν προσφέρουν έτοιμες δομές δεδομένων γράφου. Παρόλα αυτά

έχουν δημιουργηθεί κάποιες γενικές δομές δεδομένων γράφου και σε κάποιες γλώσσες προσφέρονται ως ελεύθερα πακέτα βιβλιοθήκης από open-course οργανισμούς, με το σκεπτικό πως ο προγραμματιστής θα προχωρήσει σε τροποποίησή ώστε να καλύψει τις συγκεκριμένες ανάγκες υλοποίησης.

Οι δομές δεδομένων που υλοποιούν τον ΑΤΔ γράφο, παρουσιάζουν μεγάλες διαφορές μεταξύ τους σε χρόνους απόκρισης στις διάφορες διαδικασίες. Για τον λόγο αυτό κρίθηκε αναγκαίο να αναφέρουμε σε αυτό το σημείο τις διαδικασίες και τις απεικονίσεις των δομών δεδομένων για τους γράφους, μιας και όλη η λύση του προβλήματος της παρούσας εργασίας βασίζεται στο συγκεκριμένο ΑΤΔ και η επιλογή της σωστής δομής είναι καίριας σημασίας.

Οι κυριότερες διαδικασίες που παρέχονται από μια δομή δεδομένων γράφου G συνήθως περιλαμβάνουν της εξής:

- γειτνίαση(G, x, y): έλεγξε αν υπάρχει ακμή μεταξύ των κόμβων x και y
- γείτονες(G, x): δώσε μου όλους τους κόμβους y με τους οποίους συνδέεται ο κόμβος x
- πρόσθεσε_κόμβο(G, x): πρόσθεσε τον κόμβο x, αν δεν υπάρχει ήδη
- αφαίρεσε_κόμβο(G, x): αφαίρεσε τον κόμβο x, αν υπάρχει ακόμα
- πρόσθεσε_ακμή(G, x, y): πρόσθεσε την ακμή μεταξύ των κόμβων x και y, αν δεν υπάρχει ήδη
- αφαίρεσε_ακμή (G, x, y): αφαίρεσε την ακμή μεταξύ των κόμβων x και y, αν υπάρχει ακόμα
- επέστρεψε_τιμή_κόμβου(G, x): επιστρέφει την τιμή που σχετίζεται με τον κόμβο x
- καθόρισε_τιμή_κόμβου(G, x, v): καθορίζει την τιμή που σχετίζεται με τον κόμβο x ίση με v

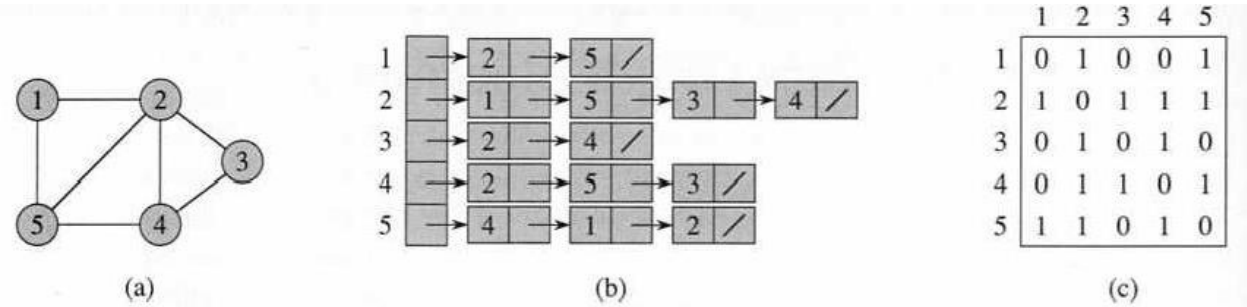
Οι δομές που σχετίζουν και τιμές με τις ακμές (βάρη) περιλαμβάνουν και:

- επέστρεψε_τιμή_ακμής(G, x, y): επιστρέφει την τιμή που σχετίζεται με την ακμή (x, y)
- καθόρισε_τιμή_ακμής (G, x, y, v): καθορίζει την τιμή που σχετίζεται με την ακμή (x, y) ίση με v

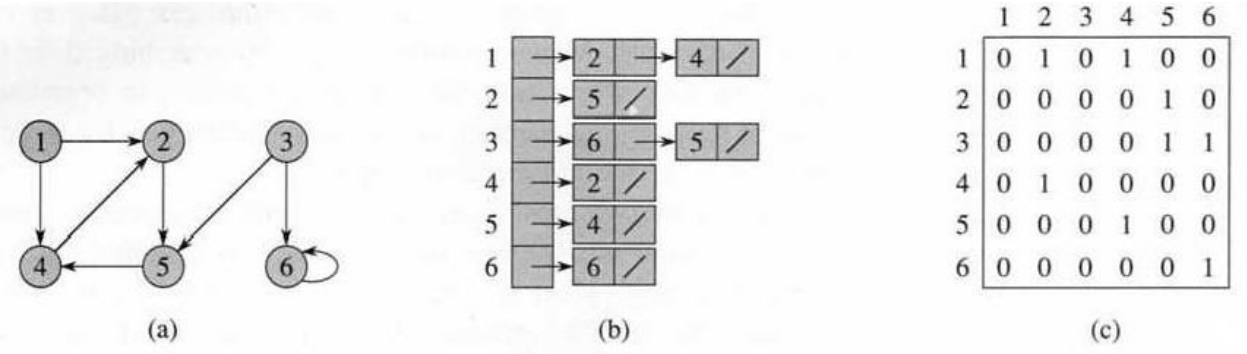
Οι δύο βασικότερες δομές δεδομένων γράφου είναι οι εξής:

- *Λίστα Γειτνίασης* [*Adjacency List*]: Οι κόμβοι αποθηκεύονται ως αντικείμενα, και κάθε κόμβος αποθηκεύει μια λίστα με τους γειτονικούς κόμβους. Αυτή η δομή δεδομένων επιτρέπει την αποθήκευση πρόσθετων δεδομένων σχετικά με τους κόμβους. Τα πρόσθετα αυτά δεδομένα μπορούν να αποθηκευτούν εάν οι ακμές είναι επίσης αποθηκευμένες ως αντικείμενα, στην περίπτωση αυτή κάθε κόμβος αποθηκεύει τις προσπίπτουσες ακμές του και κάθε ακμή αποθηκεύει τους προσπίπτων κόμβους της.

- *Πίνακας Γειτνίασης* [*Adjacency Matrix*]: Ένας δισδιάστατος πίνακας, στον οποίο οι σειρές αναπαριστούν κόμβους-πηγές και οι στήλες αναπαριστούν κόμβους-προορισμού. Τα δεδομένα σχετικά με τις ακμές και τις κορυφές πρέπει να αποθηκεύονται εξωτερικά. Μόνο το κόστος μιας ακμής μπορεί να αποθηκευτεί μεταξύ κάθε ζεύγους κόμβων.



Σχήμα 2.3 Δύο απεικονίσεις ενός μη κατευθυνόμενου γράφου. (a) Ένας μη κατευθυνόμενος γράφος G που περιέχει 5 κόμβους και 7 ακμές. (b) Μια απεικόνιση λίστα γειτνίασης του G. (c) Μια απεικόνιση πίνακα γειτνίασης του G.



Σχήμα 2.4 Δύο απεικονίσεις ενός κατευθυνόμενου γράφου. (a) Ένας κατευθυνόμενος γράφος G που περιέχει 6 κόμβους και 8 ακμές. (b) Μια απεικόνιση λίστα γειτνίασης του G. (c) Μια απεικόνιση πίνακα γειτνίασης του G.

Ο ακόλουθος πίνακας δίνει τα κόστη της πολυπλοκότητα χρόνου για την εκτέλεση διάφορων διαδικασιών, για κάθε μία από αυτές τις αναπαραστάσεις, με |V| συμβολίζεται ο αριθμός των κόμβων και με |E| ο αριθμός των ακμών. Τα κόστη ακμών που δεν υπάρχουν υποτίθεται ότι είναι ∞.

	Λίστα Γειτνίασης	**Πίνακας Γειτνίασης**
Αποθήκευσε γράφο	O(|V|+|E|)	O($\|V\|^2$)
Πρόσθεσε κόμβο	O(1)	O($\|V\|^2$)
Πρόσθεσε ακμή	O(1)	O(1)
Αφαίρεσε κόμβο	O(|E|)	O($\|V\|^2$)
Αφαίρεσε ακμή	O(|E|)	O(1)
Επερώτημα: είναι οι κόμβοι x και y γείτονες; (υποθέτοντας ότι οι θέσεις τους είναι γνωστές)	O(|V|)	O(1)
Παρατηρήσεις	Αργό για αφαίρεση κόμβων και ακμών, γιατί χρειάζεται να βρει όλους τους κόμβους και τις ακμές	Αργό για πρόσθεση ή αφαίρεση κόμβων, γιατί ο πίνακας πρέπει να αλλάξει διαστάσεις (αντιγραφτεί)

Γενικά προτιμώνται οι λίστες γειτνίασης, επειδή απεικονίζουν αποτελεσματικότερα τα αραιά γραφήματα. Ένας πίνακας γειτνίασης προτιμάται εάν ο γράφος είναι πυκνός, δηλαδή ο αριθμός των ακμών $|E|$ είναι κοντά στο τετράγωνο του αριθμού των κόμβων $|V|^2$, ή εάν χρειαζόμαστε να ελέγχουμε συχνά και γρήγορα αν υπάρχει μια ακμή που συνδέει δύο κόμβους.

2.5. Αλγόριθμοι και Ευρετικά

Ένας αλγόριθμος είναι η περιγραφή μιας αυτοματοποιημένης λύσης ενός προβλήματος. Τα βήματα του αλγορίθμου ορίζονται επακριβώς και με σαφήνεια. Η λύση μπορεί ή δεν μπορεί να είναι η βέλτιστη δυνατή, αλλά γνωρίζουμε από την αρχή τι είδους αποτελέσματα θα πάρουμε. Υλοποιούμε τον αλγόριθμο χρησιμοποιώντας κάποια γλώσσα προγραμματισμού και παίρνουμε ένα μέρος ενός προγράμματος.

Όπως αναφέραμε στην προηγούμενη ενότητα, ορισμένα προβλήματα είναι δύσκολα για τους υπολογιστές, και ίσως να μην μπορούμε να είμαστε σε θέση να πάρουμε μια αποδεκτή λύση σε ένα αποδεκτό χρονικό διάστημα. Σε τέτοιες περιπτώσεις μπορούμε να πάρουμε μια όχι και τόσο κακή λύση πολύ πιο γρήγορα, μέσω της εφαρμογής ορισμένων αυθαίρετων επιλογών (προσεγμένες μαντεψιές): αυτά είναι τα ευρετικά.

Ένα ευρετικό (ή πιο επίσημα μια ευρετική διαδικασία) εξακολουθεί να είναι ένα είδος ενός αλγορίθμου, αλλά είναι αυτό που δεν θα διερευνήσει όλες τις πιθανές καταστάσεις του προβλήματος, ή που θα ξεκινήσει με τις πιο πιθανές από αυτές.

Μερικά τυπικά παραδείγματα ευρετικών προέρχονται από τα παιχνίδια. Εάν γράψουμε ένα πρόγραμμα για ένα παιχνίδι σκακιού, μπορούμε να βάλουμε το πρόγραμμα να δοκιμάζει κάθε πιθανή κίνηση σε κάποιο επίπεδο βάθους, να εφαρμόσουμε κάποια λειτουργία αξιολόγησης ώστε να αξιολογήσουμε τις διάφορες επιλογές που θα προκύψουν και να επιλέξουμε τη βέλτιστη από αυτές. Ένα ευρετικό θα απέκλειε ολόκληρα τα "κλαδιά επιλογών" που ξεκινούν με προφανείς-κακές κινήσεις.

Σε ορισμένες περιπτώσεις δεν ψάχνουμε για τη βέλτιστη λύση, αλλά για οποιαδήποτε λύση ικανοποιεί κάποιους περιορισμούς. Ένα καλό ευρετικό θα βοηθήσει να βρεθεί μια λύση σε σύντομο χρονικό διάστημα, αλλά μπορεί επίσης να αποτύχει να βρει οποιαδήποτε λύση αν οι μόνες λύσεις είναι στις καταστάσεις που επέλεξε να αγνοήσει.

Στην παρούσα εργασία αναζητάμε πάντα τη βέλτιστη λύση του προβλήματος πληροφοριών χρονοδιαγράμματος. Γι' αυτό και θα αγνοήσουμε οποιαδήποτε μεθοδολογία περιλαμβάνει κάποιο ευρετικό το οποίο δεν εγγυάται τη βελτιστότητα του αποτελέσματος μας. Θα ασχοληθούμε μόνο με κάποια ευρετικά που εγγυόνται τη βελτιστότητα, τα οποία ανήκουν στην κατηγορία "ξεκίνα με τα πιο πιθανά" και γι' αυτό αναφέρονται συχνά και ως τεχνικές επιτάχυνσης (για διαφοροποίηση).

2.6. Αλγόριθμοι Συντομότερης Διαδρομής Μοναδικής-πηγής

Θα μοντελοποιήσουμε τα προβλήματα χρονοδιαγράμματος ως προβλήματα εύρεσης συντομότερης διαδρομής σε έναν κατάλληλα ορισμένο γράφο.

Μια *s-t-διαδρομή* [*source-target-path*] από έναν *κόμβο-πηγή s* σε ένα *κόμβο-στόχο t* είναι μια διαδρομή της οποίας ο πρώτος κόμβος είναι ο s και ο τελευταίος κόμβος είναι ο t. Εάν μεταξύ όλων αυτών των s-t-διαδρομών δεν υπάρχει κάποια με μικρότερο μήκος, η διαδρομή αυτή ονομάζεται συντομότερη διαδρομή από την πηγή s στον στόχο t.

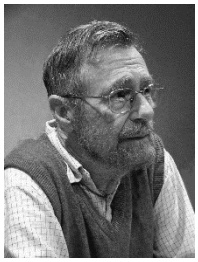
Δεδομένου ότι στα προβλήματα που εξετάζουμε, τα οποία αφορούν στις πληροφορίες χρονοδιαγράμματος σιδηροδρομικών μεταφορών, ο κόμβος-πηγή της συντομότερης διαδρομής θα είναι πάντα δεδομένος, ασχολούμαστε μόνο με τα προβλήματα συντομότερης διαδρομής μοναδικής-πηγής της μορφής: "Βρες τη συντομότερη διαδρομή από ένα δεδομένο κόμβο-πηγή προς όλους τους άλλους κόμβους". Σε μερικά από τα μοντέλα που θα εξετάσουμε, θα είναι δεδομένος και ο κόμβος-στόχος, και σε αυτήν την περίπτωση αναφερόμαστε στο πρόβλημα ως *μονού-ζεύγους* [*single-pair*]. Εμφανίζεται επίσης και η περίπτωση ενός περιορισμένου συνόλου κόμβων-στόχου, η οποία δεν σχετίζεται με την εφαρμογή μας και αναφέρεται ως πρόβλημα *μοναδικής-πηγής μερικών-στόχων* [*single-source some-targets*]: "Μεταξύ όλων των διαδρομών από την πηγή προς έναν από τους κόμβους-στόχου, θέλουμε τη διαδρομή με το ελάχιστο μήκος (να σημειωθεί ότι σε αυτήν την περίπτωση πρέπει να υπολογιστεί μόνο μια διαδρομή)".

Αυτοί οι περιορισμοί των προβλημάτων συντομότερης διαδρομής που πρέπει να επιλυθούν, περιορίζουν τις επιλογές των λογικών αλγορίθμων. Δεδομένου ότι ο στόχος είναι λίγο-πολύ σταθερός, ένας αλγόριθμος συντομότερης διαδρομής μπορεί να τερματιστεί μόλις βρεθεί η συντομότερη διαδρομή προς το στόχο.

Ένας τέτοιος τερματισμός μπορεί να γίνει εύκολα, αν ο αλγόριθμος είναι *ετικετοποίησης* [*label-setting*]. Ως label-setting χαρακτηρίζονται οι αλγόριθμοι που σε κάθε βήμα τους καθορίζεται σαφώς η συντομότερη διαδρομή προς έναν κόμβο. Αυτή η ιδιότητα μας δίνει τη δυνατότητα να ματαιώσουμε το αλγόριθμο μόλις υπολογιστεί η συντομότερη διαδρομή προς τον κόμβο που ζητάμε. Με αυτόν τον τρόπο η διαδικασία διακόπτεται ενώ έχουμε επεξεργαστεί μόνο ένα μικρό κλάσμα του συνόλου των κόμβων, που αναφέρεται επίσης ως ο χώρος αναζήτησης. Αρκετές τεχνικές επιτάχυνσης προσπαθούν να βελτιώσουν τον χρόνο τρεξίματος με περαιτέρω μείωση του χώρου αναζήτησης.

Σε αντίθεση με τους αλγορίθμους label-setting, οι αλγόριθμοι label-correcting βελτιώνουν σε κάθε βήμα τους την τρέχουσα συντομότερη διαδρομή προς τον κάθε άλλον κόμβο, και ο αλγόριθμος δεν μπορεί απλά να ματαιωθεί όπως στην περίπτωση των label-setting παραπάνω. Η διαφορά ταχύτητας μεταξύ τους είναι μεγάλη για την περίπτωσή μας και γι' αυτό θα επικεντρωθούμε στους label-setting αλγορίθμους.

2.6.1. Αλγόριθμος του Dijkstra



Ο κλασικός label-setting αλγόριθμος οφείλεται στον Ολλανδό Edsger W. Dijkstra, και παρουσιάστηκε στο δισέλιδο τεύχος με τίτλο "*Σημείωση για δύο προβλήματα συνδέσεων με γράφους* [*A Note on Two Problems in Connexion with Graphs*]" (Dijkstra 1959). Τα περισσότερα από τα προβλήματα πληροφοριών χρονοδιαγράμματος λύνονται με τον αλγόριθμο αυτό και τις παραλλαγές του. Βασική προϋπόθεση είναι πως τα δοσμένα μήκη ακμών απαιτείται να είναι μη αρνητικά.

Βασική Ιδέα και Αλγόριθμος

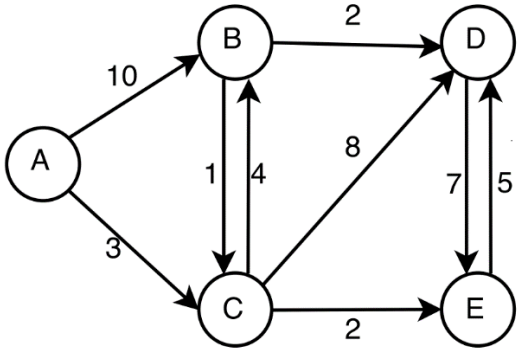
Σε κάθε κόμβο έχει εκχωρηθεί μια ετικέτα απόστασης και ένας δείκτης με τις καταστάσεις *unvisited, visited*, και *finished*. Ο αλγόριθμος αρχικοποιεί όλους τους κόμβους σε unvisited, θέτει την ετικέτα απόστασης του κόμβου πηγής ίση με 0, και μαρκάρει τον κόμβο-πηγή σε visited. Παράλληλα, ο αλγόριθμος διατηρεί μια ουρά προτεραιότητας που αποθηκεύει τους κόμβους visited, και προσθέτει τον κόμβο-πηγή στην αρχή της ουράς προτεραιότητας. Η προτεραιότητα σε αυτήν την ουρά είναι η ετικέτα απόστασης των κόμβων (όσο μικρότερη είναι τόσο καλύτερη). Το υπόλοιπο του αλγορίθμου είναι ένας βρόχος που τερματίζει όταν η ουρά προτεραιότητας αδειάσει: Ο κόμβος u με τη μικρότερη ετικέτα απόστασης αφαιρείται από την ουρά, και μαρκάρεται με finished. Στη συνέχεια, εξετάζονται όλες οι εξερχόμενες ακμές του u μία προς μία και "*χαλαρώνονται* [*relaxed*]"· δηλαδή πραγματοποιείται η εξής διαδικασία: Έστω V ο άλλος κόμβος μιας τέτοιας ακμής. Εάν, από τη μία πλευρά, ο v είναι ακόμα unvisited ή, από την άλλη πλευρά, ο u είναι ήδη visited αλλά η ετικέτα απόστασης του u συν το μήκος ακμής από τον u στο v είναι μικρότερα από την ετικέτα απόστασης v, η ετικέτα απόστασης του v ορίζεται ίση με την ετικέτα απόστασης του u συν το μήκος της ακμής από το u στο v. Ο κόμβος v μαρκάρεται ως visited και μπαίνει στην ουρά προτεραιότητας εκτός και αν ήταν ήδη σε αυτή την κατάσταση. Εάν ο V ήταν ήδη visited, ήταν ήδη στην ουρά και ίσως πρέπει να ενημερωθεί η προτεραιότητα.

Η έννοια της "χαλάρωσης" προέρχεται από μια αναλογία μεταξύ της εκτίμησης της μικρότερης διαδρομής και του μήκους ενός ελικοειδούς ελατηρίου. Αρχικά, το κόστος της συντομότερης διαδρομής είναι μια υπερεκτίμηση, και παρομοιάζεται με ένα τεντωμένο ελατήριο. Όσο υπολογίζονται οι συντομότερες διαδρομές, το εκτιμώμενο κόστος μειώνεται, και το ελατήριο χαλαρώνει. Τελικά, υπολογίζεται η συντομότερη διαδρομή και το ελατήριο χαλαρώνει στο μήκος ηρεμίας του.

Εάν ο αλγόριθμος εκτελεστεί περισσότερες από μία φορές πάνω στον ίδιο γράφο, αλλά για διαφορετικό κόμβο πηγή, δεν χρειάζεται να επαναλαμβάνεται η αρχικοποίηση του δείκτη. Αντ' αυτού, μπορεί να χρησιμοποιηθεί μια τεχνική χρονοσήμανσης: Ας υποθέσουμε ότι εκτελούμε τον αλγόριθμο για i-στη φορά. Ο δείκτης κωδικοποιείται από μια ακέραια τιμή m ως εξής: ένας κόμβος είναι unvisited αν $m < 2 \cdot i$, είναι visited εάν $m = 2 \cdot i$, και ο κόμβος μαρκάρεται ως finished εάν $m = 2 \cdot i + 1$.

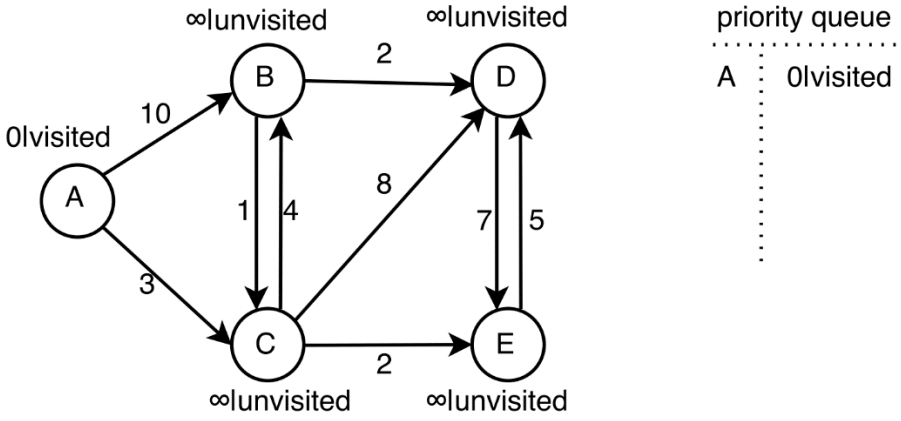
Παράδειγμα Αλγορίθμου Dijkstra

Για την πλήρη κατανόηση του αλγορίθμου παρουσιάζουμε την αναλυτική εφαρμογή πάνω στον παρακάτω γράφο. Θεωρούμε κόμβο έναρξης τον κόμβο Α. Ο αλγόριθμος υπολογίζει διαδοχικά όλες τις συντομότερες διαδρομές από τον κόμβο έναρξης προς κάθε έναν από τους υπόλοιπους κόμβους.



Σε κάθε κόμβο του αλγορίθμου τοποθετούμε μια ετικέτα απόστασης και ένα δείκτη κατάστασης που μπορεί να πάρει τις τιμές unvisited, visited, και finished. Κατά την αρχικοποίηση ο αλγόριθμος εκχωρεί σε όλες τις ετικέτες απόστασης την τιμή ∞ και μαρκάρει τους δείκτες ως unvisited. Στον κόμβο έναρξης θέτει την τιμή 0 και τον μαρκάρει ως visited.

Παράλληλα ο αλγόριθμος χρησιμοποιεί μια ουρά προτεραιότητας στην οποία διατηρούνται όλοι οι visited κόμβοι. Η προτεραιότητα σε αυτήν την ουρά είναι η ετικέτα απόστασης των κόμβων (όσο μικρότερη είναι τόσο καλύτερη). Από τη στιγμή που η ετικέτα του κόμβου έναρξης Α έγινε visited, ο κόμβος μπήκε στην ουρά προτεραιότητας όπως παρουσιάζεται στο επόμενο σχήμα.



Το υπόλοιπο του αλγορίθμου είναι μια επαναληπτική διαδικασία η οποία τερματίζεται όταν αδειάσει η ουρά προτεραιότητας. Η διαδικασία περιλαμβάνει τα εξής βήματα:

I. Βγάλε από την ουρά προτεραιότητας τον 1ο κόμβο (δηλαδή αυτό με τη μικρότερη ετικέτα απόστασης) και μάρκαρέ τον ως finished.

II. Εξέτασε διαδοχικά τους γειτονικούς κόμβους στους οποίους καταλήγουν οι ακμές που ξεκινούν από τον συγκεκριμένο κόμβο (out-edges), ενημέρωσε την ετικέτα απόστασής τους (διαδικασία ”χαλάρωσης“), μάρκαρέ τους ως visited και τοποθέτησέ τους στην ουρά προτεραιότητας.

Έστω v ο κόμβος από τον οποίο ξεκινά και u αυτός στον οποίο καταλήγει η ακμή. Η διαδικασία ”χαλάρωσης“ αποτελείται από την εξής συνάρτηση. Αν:

- ο v είναι ακόμα unvisited
  ή
- ο u είναι ήδη visited αλλά η ετικέτα απόστασης του u συν το μήκος ακμής από τον u στον v είναι μικρότερα από την ετικέτα απόσταση v

όρισε την ετικέτα απόστασης του v ίση με την ετικέτα απόστασης του u συν το μήκος της ακμής από το u στον v.

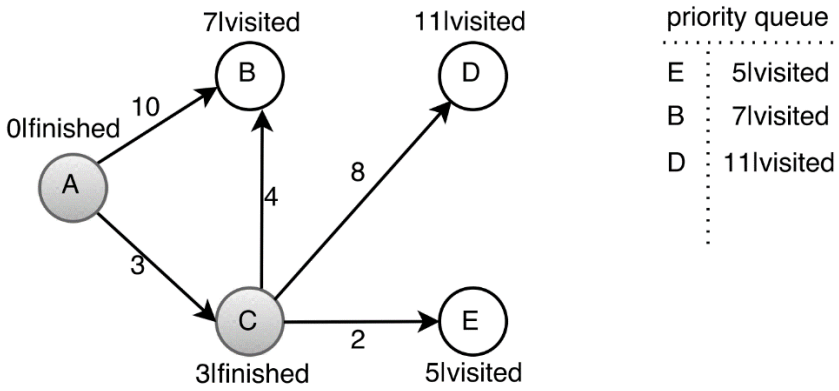
Παρακάτω παρουσιάζουμε αναλυτικά την κάθε μία επανάληψη.

<u>1η επανάληψη</u>

- Βγαίνει ο κόμβος A από την ουρά και γίνεται finished.
- Εξετάζεται ο κόμβος B. 0+10<∞ άρα ενημερώνεται η ετικέτα του, μαρκάρεται ως visited και εισέρχεται στην ουρά προτεραιότητας.
- Εξετάζεται ο κόμβος C. 0+3<∞ άρα ενημερώνεται η ετικέτα του, μαρκάρεται ως visited και εισέρχεται στην ουρά προτεραιότητας.
- Υπάρχουν άλλες out-edges; Όχι

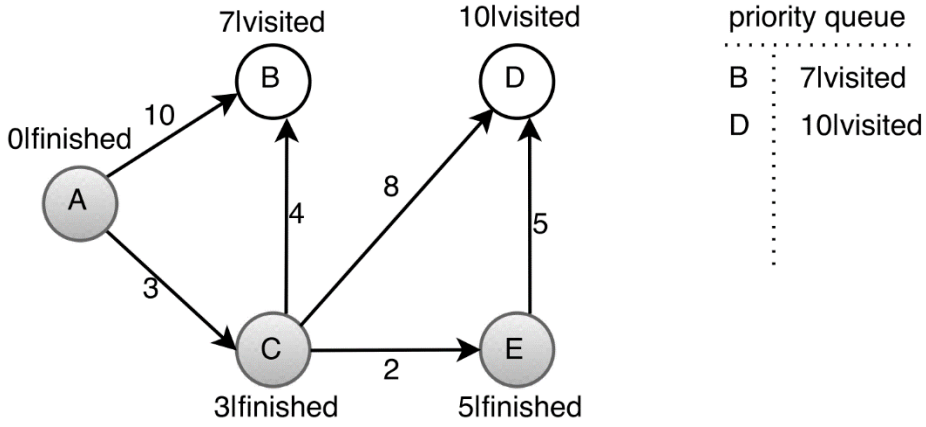
<u>2η επανάληψη</u>

- Βγαίνει ο κόμβος C από την ουρά και γίνεται finished.
- Εξετάζεται ο κόμβος B. 3+4<10 άρα ενημερώνεται η ετικέτα του. Είναι ήδη visited και στην ουρά προτεραιότητας.
- Εξετάζεται ο κόμβος D. 3+8<∞ άρα ενημερώνεται η ετικέτα του, μαρκάρετε ως visited και εισέρχεται στην ουρά προτεραιότητας.
- Εξετάζεται ο κόμβος E. 3+2<∞ άρα ενημερώνεται η ετικέτα του, μαρκάρετε ως visited και εισέρχεται στην ουρά προτεραιότητας.
- Υπάρχουν άλλες out-edges; Όχι



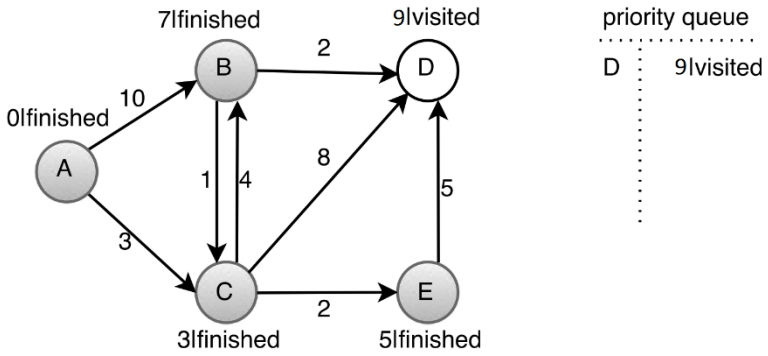
3η επανάληψη

- Βγαίνει ο κόμβος Ε από την ουρά και γίνεται finished.
- Εξετάζεται ο κόμβος D. 5+5<11 άρα ενημερώνεται η ετικέτα του. Είναι ήδη visited και στην ουρά προτεραιότητας.
- Υπάρχουν άλλες out-edges; Όχι



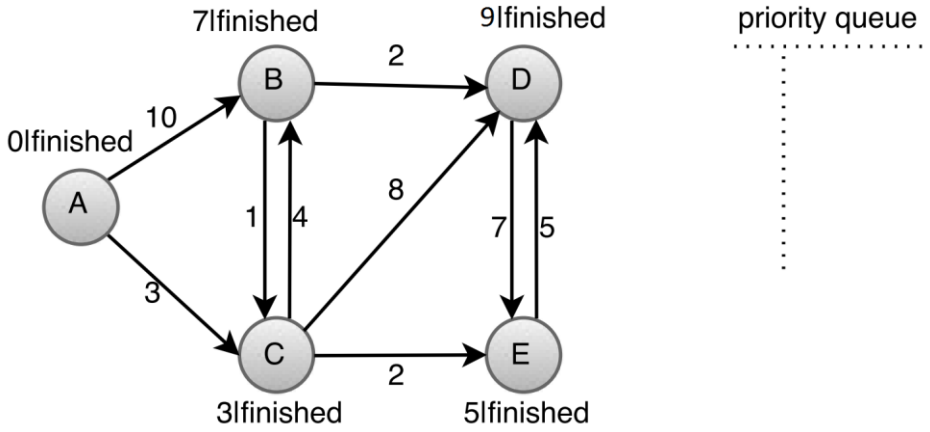
4η επανάληψη

- Βγαίνει ο κόμβος Β από την ουρά και γίνεται finished.
- Εξετάζεται ο κόμβος D. 7+2<10 άρα ενημερώνεται η ετικέτα του. Είναι ήδη visited και στην ουρά προτεραιότητας.
- Ο κόμβος C δεν εξετάζεται γιατί είναι ήδη finished.
- Υπάρχουν άλλες out-edges; Όχι



5η επανάληψη

- Βγαίνει ο κόμβος D από την ουρά και γίνεται finished.
- Ο κόμβος E δεν εξετάζεται γιατί είναι ήδη finished.
- Υπάρχουν άλλες out-edges; Όχι



Σε αυτό το σημείο ο αλγόριθμος σταματά καθώς δεν υπάρχει κανείς κόμβος στην ουρά προτεραιότητας.

Ο αλγόριθμος έχει υπολογίσει τις ελάχιστες αποστάσεις του κάθε κόμβου από τον κόμβο έναρξης. Αν θέλουμε μια μόνο συντομότερη διαδρομή προς έναν συγκεκριμένο κόμβο, το μόνο που έχουμε να κάνουμε είναι να ματαιώσουμε την επαναληπτική διαδικασία μόλις γίνει αυτός ο κόμβος finished.

Αν επιθυμούμε να γνωρίζουμε και τη διαδρομή, προσθέτουμε απλά έναν πίνακα στον οποίο αποθηκεύουμε τους κόμβους ”γονείς” του κάθε κόμβου. Ο πίνακας αυτός ενημερώνεται μαζί με την ετικέτα απόστασης ώστε στο τέλος να μας πληροφορεί για τον κόμβο από τον οποίο προήλθαμε και είμαστε σε θέση να βρούμε τη συντομότερη διαδρομή με μια προς τα πίσω διαδικασία από ”γονέα” σε ”γονέα” έως την πηγή.

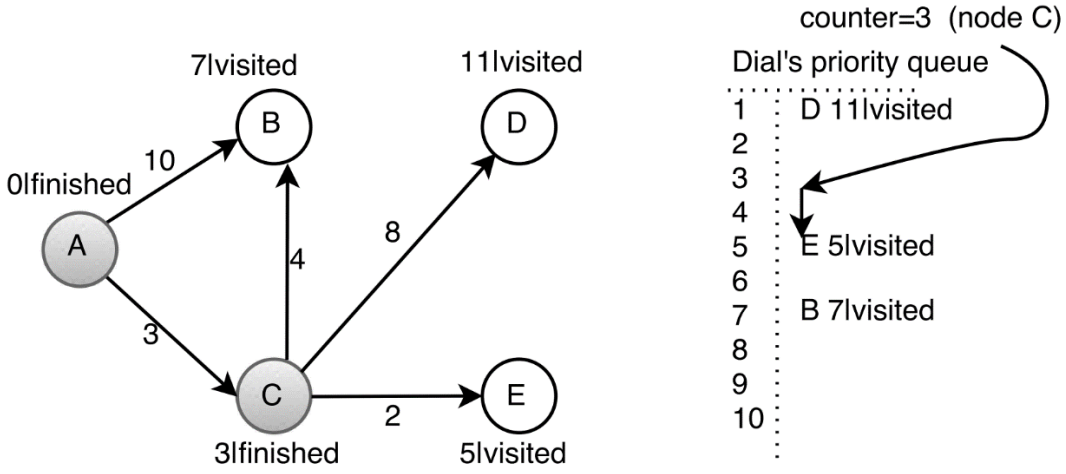
2.6.2. Ουρές Προτεραιότητας για τον Αλγόριθμο του Dijkstra

Στην αρχική περιγραφή του αλγορίθμου του Dijkstra (Dijkstra 1959), δεν αναφέρεται πώς να επιλέξουμε τον κόμβο με τη μικρότερη ετικέτα απόστασης μεταξύ όλων των visited κόμβων που βρίσκονται μέσα στην ουρά προτεραιότητας. Η προσέγγιση να σαρώσουμε απλά όλους τους visited κόμβους οδηγεί σε χρόνο τρεξίματος ανάλογο του τετραγώνου του αριθμού των κόμβων. Για τους γράφους που έχουν αριθμό ακμών ανάλογο του τετραγώνου του αριθμού κόμβων, αυτός ο χρόνος τρεξίματος είναι βέλτιστος, δεδομένου ότι ο αλγόριθμος επεξεργάζεται κάθε ακμή. Ωστόσο, για "αραιούς" γράφους με λιγότερες ακμές, ο χρόνος τρεξίματος μπορεί να βελτιωθεί με την εφαρμογή μιας καλύτερης ουράς προτεραιότητας.

Η υλοποίηση του Dial

Εάν τα μήκη των ακμών είναι φυσικοί αριθμοί φραγμένοι από μία μικρή σταθερά C, μια τεχνική κάδων που προτάθηκε από τον Dial (Dial 1969) οδηγεί σε ένα πολύ αποδοτικό αλγόριθμο. Με τον όρο κάδος εννοούμε τον ΑΤΔ *δοχείου* [*container*]. Διατηρούμε μια σειρά από C κάδους, ώστε κάθε ένας από τους κάδους να είναι σε θέση να αποθηκεύσει ένα σύνολο από κόμβους. Η εισαγωγή ενός κόμβου με απόσταση d γίνεται με την προσθήκη του κόμβου στον (d mod C)-ου κάδου. Περαιτέρω, έχουμε έναν μετρητή που αντανακλά τον κάδο που περιείχε τον τελευταίο κόμβο που έχει αφαιρεθεί. Η αφαίρεση ενός κόμβου είναι απλή: αυξάνουμε τον μετρητή κάδου μέχρι να φτάσουμε σε έναν μη κενό κάδο (αν φτάσουμε τον τελευταίο κάδο συνεχίζουμε στον πρώτο κάδο με μια κυκλική διαδρομή), και παίρνουμε έναν κόμβο από αυτόν τον κάδο. Αν η απόσταση ενός κόμβου πρέπει να ενημερωθεί, αφαιρούμε τον κόμβο από τον κάδο που φυλάσσεται προς το παρόν, και τον τοποθετούμε σε έναν άλλο κάδο με τη νέα απόσταση.

Για παράδειγμα, η υλοποίηση του Dial στην παραπάνω εφαρμογή κατά τη 2η επανάληψη θα μας έδινε την κατάσταση που παρουσιάζεται στο παρακάτω σχήμα.



Βλέπουμε πως υπάρχουν 10 κάδοι καθώς αυτή είναι η μέγιστη τιμή βάρους ακμής στο συγκεκριμένο παράδειγμα. Ο μετρητής έχει το αριθμό του κάδου από τον οποίο αφαιρέθηκε τελευταία φορά κόμβος, δηλαδή ο C από τον κάδο 3. Αφού εισέλθουν και οι νέοι visited κόμβοι αν υπάρξουν, για ν βρεθεί ο κόμβος που θα βγει στην επόμενη επανάληψη, αυξάνουμε το μετρητή

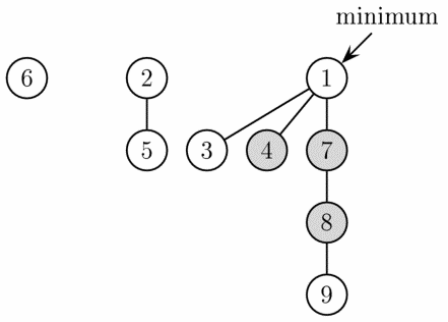
μέχρι να βρει μη κενό κάδο. Στην περίπτωσή μας μέχρι να φτάσει στον κάδο 5 που περιέχει τον κόμβο Ε με όντως τη μικρότερη ετικέτα ακμής.

Ο χρόνος τρεξίματος της προσθήκης του Dial στον αλγόριθμο του Dijkstra είναι O(m+nC), επειδή είναι πιθανό ότι για κάθε αφαίρεση ενός κόμβου από την ουρά, ο μετρητής κάδων να πρέπει να αυξηθεί C - 1 φορές. Στην πράξη, ειδικά αν ο C είναι μικρός, η υλοποίηση του Dial είναι μία από τις πιο αποτελεσματικές υλοποιήσεις του αλγορίθμου του Dijkstra με ενσωματωμένα μήκη ακμών (Hung and Divoky 1988).

Οι Σωροί Binary, Fibonacci και Σύζευξης

Αντί για τον αφηρημένο τύπο δεδομένων ουράς προτεραιότητας μπορούμε να χρησιμοποιήσουμε τον ΑΤΔ Σωρού. Η ιδέα είναι να εκμεταλλευτούμε το γεγονός ότι δεν απαιτούμε από την αφαίρεση να παραθέτει τα αντικείμενα που περιέχει σε πλήρη ταξινόμηση, αλλά να είμαστε βέβαιοι ότι το πρώτο αντικείμενο της ταξινόμησης είναι το σωστό. Μπορούμε δηλαδή να αποφύγουμε κάποιες ενέργειες οι οποίες συμβαίνουν στις ουρές προτεραιότητας. Στην πληροφορική υπάρχουν διάφοροι σωροί με μεγάλες διαφορές μεταξύ τους.

Η χρήση ενός σωρού Fibonacci ως ουρά προτεραιότητας δίνει το θεωρητικά καλύτερο φράγμα για τον αλγόριθμο του Dijkstra O(m+nlogn). Η ονομασία του σωρού αυτού προέρχεται από τους αριθμούς Fibonacci, οι οποίοι χρησιμοποιούνται στην ανάλυση χρόνου τρεξίματος [running time analysis] της συγκεκριμένης δομής. Με αυτό το σωρό, οι λειτουργίες εισαγωγής και οι αφαίρεσης μπορούν να γίνουν σε χρόνο O(logn), και η λειτουργία μείωσης κλειδιού (ενημέρωση προτεραιότητας) είναι δυνατή σε αποσβεσμένο σταθερό χρόνο.



Ο σωρός σύζευξης είναι συγγενής του σωρού Fibonacci. Το επίκεντρο του σωρού σύζευξης είναι μια εύκολη και αποτελεσματική εφαρμογή. Το μειονέκτημα σε σχέση με τους σωρούς Fibonacci είναι ότι για τους σωρούς σύζευξης, η λειτουργία μείωσης-κλειδιού δεν μπορεί να είναι εγγυημένο ότι θα τρέξει σε σταθερό χρόνο. Ωστόσο, πειραματικές μελέτες δείχνουν ότι οι σωροί σύζευξης είναι στην πράξη πολύ αποτελεσματικοί, και μια καλή επιλογή για να χρησιμοποιηθούν σε αλγόριθμο Dijkstra, ειδικά στη γενική περίπτωση των μη ακέραιων μηκών ακμών.

Ένας σωρός σύζευξης υλοποιείται ως ένα δέντρο, όπου κάθε κόμβος του δέντρου περιέχει ένα στοιχείο του σωρού. Σέβεται τη σειρά του σωρού, δηλαδή, η προτεραιότητα των κόμβων-παιδιών δεν είναι μικρότερη από την προτεραιότητα των κόμβων-γονέων. Το στοιχείο με τη μικρότερη προτεραιότητα είναι αποθηκευμένο στη ρίζα του δένδρου. Οι λειτουργίας εισαγωγής, αφαίρεσης, και μείωσης κλειδιού υλοποιούνται ως μια σειρά συνδεδεμένων λειτουργειών.

2.6.3. Άλλοι Αλγόριθμοι Συντομότερης Διαδρομής Μοναδικής-πηγής

Όντας ένα θεμελιώδες θέμα στην επιστήμη των υπολογιστών, υπάρχουν πολλοί διαφορετικοί αλγόριθμοι συντομότερης διαδρομής για μια ποικιλία από διαφορετικές εκδοχές του προβλήματος. Ειδικότερα όμως οι label-setting αλγόριθμοι συντομότερης διαδρομής είναι εξαιρετικά περιορισμένοι.

Ένα σημαντικό ζήτημα στην πρόσφατη έρευνα στους αλγορίθμους συντομότερης διαδρομής μοναδικής πηγής είναι ο σχεδιασμός των αλγορίθμων με χρόνο τρεξίματος καλύτερο από O(m+nlogn). Ορισμένοι από αυτούς τους νέους αλγόριθμους έχουν περισσότερο θεωρητικό ενδιαφέρον, όπως ο αλγόριθμος γραμμικού χρόνου του Thorup για μη κατευθυνόμενους γράφους (Thorup 1999), αλλά και από πρακτική άποψη, υπάρχουν ενδιαφέροντα αποτελέσματα, όπως οι αλγόριθμοι γραμμικού χρόνου μέσης περίπτωσης που πρότεινε ο Goldberg (Goldberg 2001). Οι τελευταίοι αυτοί αλγόριθμοι χρησιμοποιούν ιεραρχικά οργανωμένους κάδους ως δομές δεδομένων για την αποθήκευση των ήδη επισκεφθέντων κόμβων.

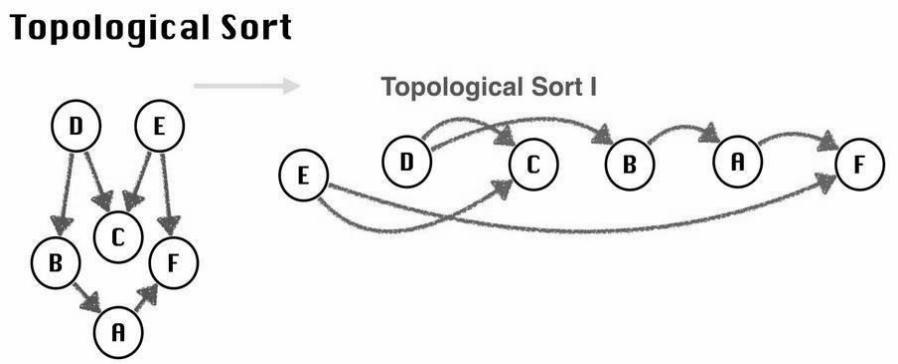
Μια άλλη κατεύθυνση της πρόσφατης δουλειάς είναι η ανάπτυξη τεχνικών επιτάχυνσης για αλγόριθμους συντομότερης διαδρομής μονού ζεύγους που εφαρμόζονται σε μεγάλους και αραιούς γράφους. Αυτές οι τεχνικές υπολογίζουν πληροφορίες σχετικά με τις συντομότερες διαδρομές σε ένα στάδιο προεπεξεργασίας. Η κατανάλωση χώρου για την αποθήκευση αυτών των πληροφοριών είναι συνήθως ανάλογη του αριθμού των κόμβων (πρέπει να είναι μικρότερη από τετραγωνική, διότι διαφορετικά θα μπορούσε να αποθηκευτεί ολόκληρος ο πίνακας αποστάσεων). Στη συνέχεια, για να απαντηθεί ένα επερώτημα, οι προϋπολογισμένες πληροφορίες μπορούν να χρησιμοποιηθούν για να περιορίσουν τον χώρο αναζήτησης στον αλγόριθμο του Dijkstra ή σε άλλους αλγόριθμους συντομότερης διαδρομής.

Τέλος, ολοκληρώνουμε τη συζήτηση των αλγορίθμων συντομότερης διαδρομής μοναδικής-πηγής με έναν αλγόριθμο γραμμικού χρόνου για μια πολύ ειδική κατηγορία γράφων, αυτή των κατευθυνόμενων μη κυκλικών γράφων.

Συντομότερες διαδρομές σε D.A.G.s

Ένας κατευθυνόμενος γράφος G που δεν περιέχει κανέναν κατευθυνόμενο κύκλο ονομάζεται Κατευθυνόμενος *Μη Κυκλικός Γράφος (Κ.Μ.Κ.Γ.)* [*Directed Acyclic Graph (D.A.G.)*]. Στα D.A.G.s, το πρόβλημα συντομότερης διαδρομής μοναδικής πηγής μπορεί να επιλυθεί σε γραμμικό χρόνο.

Μια από τις πολλές ιδιότητες των D.A.G.s είναι πως οι κόμβοι του μπορούν να μπουν σε *τοπολογική σειρά* [*topological sort*]. Η ιδέα της τοπολογικής ταξινόμησης είναι να μπουν οι κόμβοι με τέτοιο τρόπο ώστε οι ακμές να έχουν μια συγκεκριμένη κατεύθυνση, για παράδειγμα από τα αριστερά προς τα δεξιά, όπως φαίνεται και στο παρακάτω σχήμα.



Με άλλα λόγια, μια τέτοια διάταξη απαιτεί από κάθε ακμή να δείχνει προς τα εμπρός (δηλαδή, όλοι οι γείτονες του κάθε κόμβου να εμφανίζονται πάντα αργότερα στην διάταξη). Αυτή η διάταξη δεν είναι μοναδική για κάθε γράφο.

Υπάρχουν διάφοροι τρόποι για να βρούμε μια τέτοια διάταξη, αλλά ο πιο συνηθισμένος τρόπος είναι χρησιμοποιώντας τον αλγόριθμο *Αναζήτησης Πρώτα-Βάθους* [*Depth-First Search*] που μας δίνει αποτέλεσμα σε γραμμικό χρόνο. Επιλέγουμε να μην παραθέσουμε τον μηχανισμό λειτουργίας του, για να επικεντρωθούμε στο δικό μας πρόβλημα.

Η ιδέα λοιπόν είναι να εκμεταλλευτούμε αυτήν την ιδιότητα των DAGs ώστε να λύσουμε το πρόβλημα συντομότερης διαδρομής γρηγορότερα. Αρχικά τρέχουμε μια Depth-First Search πάνω στον γράφο και έτσι προκύπτει μια από τις πολλές τοπολογικές σειρές. Στη συνέχεια, η συντομότερη διαδρομή μπορεί να βρεθεί με την επεξεργασία των κόμβων του γραφήματος με τοπολογική σειρά και τη χαλάρωση των εξερχόμενων ακμών όπως και στον αλγόριθμο του Dijkstra. Η μεγάλη όμως διαφορά στην ειδική αυτήν περίπτωση είναι πως δεν απαιτείται πια η διατήρηση μιας ουράς προτεραιότητας και πως κάθε ακμή αγγίζεται το πολύ μία φορά. Χωρίς να μπούμε σε λεπτομέρειες, μπορούμε κοιτώντας το παραπάνω σχήμα να φανταστούμε τον αλγόριθμο του Dijkstra να "χαλαρώνει" έναν-έναν με την σειρά τους τοπολογικά ταξινομημένους κόμβους και γίνεται αμέσως αντιληπτός ο λόγος για τον οποίο δεν χρειάζεται πια μια ουρά προτεραιότητας. Έτσι ο συνολικός χρόνος τρεξίματος είναι γραμμικός ως προς το μέγεθος του γράφου G.

2.7. Συντομότερες Διαδρομές Πολλαπλών Κριτηρίων

Πρέπει να ασχοληθούμε επίσης και με την περίπτωση στην οποία παρουσιάζονται παραπάνω από διαφορετικά μήκη ακμής, τα οποία ονομάζονται επίσης και διαφορετικά *κριτήρια βελτιστοποίησης* [*optimization criteria*] (για παράδειγμα ο χρόνος ταξιδιού και ο αριθμός μετεπιβιβάσεων σε έναν γράφο που αναπαριστά ένα σιδηροδρομικό δίκτυο): Έστω $l_1(e)$, ..., $l_k(e)$ τα k διαφορετικά μήκη ακμών. Το μήκος μιας διαδρομής P ορίζεται ως το άθροισμα των πλειάδων $(l_1(e), ..., l_k(e))$ όλων των ακμών e της διαδρομής P. Δεν είναι πλέον σαφές το πώς ορίζεται η μοναδική "συντομότερη διαδρομή", ειδικά όταν όλα τα εκάστοτε μήκη θεωρηθεί πως έχουν την ίδια βαρύτητα. Εδώ μπορεί κανείς να εισάγει την έννοια των *επικρατούντων διαδρομών* [*dominating paths*] ως εξής: Μια διαδρομή P επικρατεί μιας άλλης διαδρομής Q εάν για ένα μήκος l_i η διαδρομή P είναι καλύτερη από την Q, και για όλα τα υπόλοιπα μήκη l_j η διαδρομή P είναι

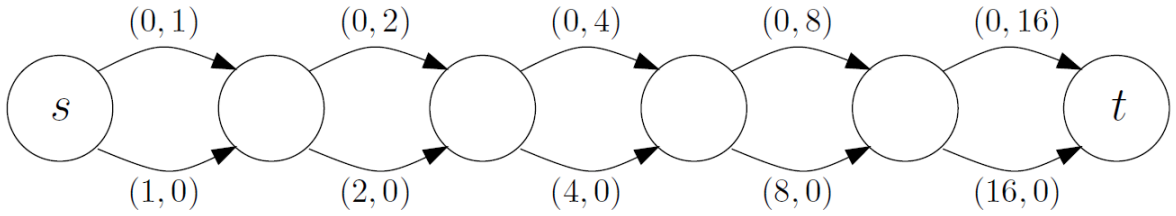
τουλάχιστον εξίσου καλή με τη διαδρομή Q. Πιο επίσημα, ορίζουμε την επικρατούσα διαδρομή ως εξής:

Ορισμός 2.2 Μια s-t-διαδρομή P με μήκος $(p_1, ..., p_k)$ επικρατεί μιας άλλης s-t-διαδρομής Q με μήκος $(q_1, ..., q_k)$ αν $p_i < q_i$ για κάποιο i $(1 \leq i \leq k)$, και $p_j \leq q_j$ για όλα τα υπόλοιπα j $(1 \leq j \leq k, i \neq j)$.

Οι διαδρομές που δεν επικράτησαν δεν είναι βέλτιστες, με την έννοια ότι υπάρχει μια καλύτερη διαδρομή η οποία βελτιώνει τουλάχιστον ένα κριτήριο, ενώ όλα τα υπόλοιπα κριτήρια παραμένουν τουλάχιστον εξίσου καλά. Έτσι, ενδιαφερόμαστε για όλες τις διαδρομές επί των οποίων δεν επικρατεί κάποια άλλη:

Ορισμός 2.3 Μία s-t-διαδρομή ονομάζεται *Pareto-βέλτιστη* [*Pareto-optimal*] εάν δεν επικρατεί επί αυτής κάποια άλλη s-t-διαδρομή.

Το πρόβλημα του καθορισμού όλων των Pareto-βέλτιστων διαδρομών είναι γενικά NP-hard. Συμβολίζουμε με p(v) τον αριθμό των Pareto-βέλτιστων s-v-διαδρομών. Γενικά, το p(v) μπορεί να είναι εκθετικό ως προς τον αριθμό των κόμβων του γράφου, ακόμα και αν k=2, όπως δείχνει το Σχήμα 2.5. Στις παραγράφους που ακολουθούν θα περιγράψουμε κάποιους αλγόριθμους για τον υπολογισμό των Pareto-βέλτιστων διαδρομών, οι οποίοι τρέχουν σε ιδιαίτερες περιπτώσεις σε πολυωνυμικό χρόνο, και στις γενικές περιπτώσεις σε εκθετικό χρόνο. Όπως θα δούμε αργότερα, ακόμα και οι αλγόριθμοι με εκθετικούς χρόνους απόκρισης μπορούν να είναι αρκετά αποτελεσματικοί στην πράξη, καθώς πολύ συχνά ο αριθμός των Pareto-βέλτιστων διαδρομών είναι αρκετά μικρός.



Σχήμα 2.5 Παράδειγμα γράφου με $p(t)=2^{n-1}$ Pareto-βέλτιστες s-t-διαδρομές (εδώ n=6 και p(t) =32).

2.7.1. Λεξικογραφικά Πρώτη Λύση

Για κάθε διάταξη των κριτηρίων (υποθέτουμε ότι τα κριτήρια είναι με τη διάταξη $l_1, ..., l_k$), υπάρχει μια ιδιαίτερη Pareto-βέλτιστη διαδρομή: αυτή η οποία είναι *λεξικογραφικά βέλτιστη* [*lexicographical optimal*] (δηλαδή η βέλτιστη ως προς τη λεξικογραφική σειρά που ορίζουν τα μήκη των διαδρομών).

Σε αυτήν την περίπτωση δηλαδή ψάχνουμε τη βέλτιστη διαδρομή με βάση τη σειρά των κριτηρίων $l_1, ..., l_k$ ακριβώς όπως η διάταξη ενός τηλεφωνικού καταλόγου (δηλαδή πρώτα με το πρώτο γράμμα του ονόματος, μετά με το δεύτερο κτλ.). Για παράδειγμα θα μπορούσαμε να

αναζητούμε τη βέλτιστη διαδρομή πρώτα ως προς τον χρόνο άφιξης, έπειτα ως προς τον αριθμό μετεπιβιβάσεων και τέλος ως προς την τιμή εισιτηρίου (σε όλες αυτές τις τιμές ζητάμε τη μικρότερη).

Η λεξικογραφικά βέλτιστη διαδρομή μπορεί να υπολογιστεί με τον αλγόριθμο του Dijkstra διατηρώντας τις πλειάδες $(d_1, ..., d_k)$ ως ετικέτες κόμβων και χρησιμοποιώντας τη λεξικογραφική διάταξη όταν οι ακμές χαλαρώνονται. Προκύπτει πως η λεξικογραφικά βέλτιστη διαδρομή είναι Pareto-βέλτιστη, καθώς μια επικρατούσα διαδρομή θα ήταν λεξικογραφικά μικρότερη.

2.7.2. Όλες οι Pareto-Βέλτιστες Διαδρομές

Όλες οι Pareto-βέλτιστες διαδρομές από έναν κόμβο s προς όλους τους άλλους κόμβους μπορούν να καθοριστούν από έναν labelling αλγόριθμο παρόμοιο με αυτόν του Dijkstra. Σε αυτόν, αντί να διατηρείται μία μόνο ετικέτα ανά κόμβο, διατηρείται ένα σύνολο ετικετών για κάθε κόμβο. Μια ετικέτα ενός κόμβου v είναι, όπως προηγουμένως για τη λεξικογραφικά πρώτη λύση, μια πλειάδα της μορφής $(d_1, ..., d_k)_v$. Το σύνολο των ετικετών ενός κόμβου v σε κάποιο στάδιο του αλγορίθμου αναπαριστά τις διαδρομές επί των οποίων δεν επικρατεί κάποια άλλη, από τον s μέχρι τον v, που έχουν βρεθεί έως τώρα. Οι διαφορές με τον αλγόριθμο του Dijkstra είναι οι εξής:

- Αρχικά, το σύνολο των ετικετών για κάθε κόμβο είναι κενό, και η ετικέτα του s ορίζεται ως 0^k (k=αριθμός κριτηρίων).
- Αντί να αποθηκεύονται κόμβοι στην ουρά προτεραιότητας, αποθηκεύονται οι ετικέτες $(d_1, ..., d_k)_v$ (μαζί με τον αντίστοιχο κόμβο v) και μπαίνουν σε λεξικογραφική διάταξη.
- Αντί απλά να ενημερώνονται οι ετικέτες των εξερχόμενων ακμών e = (u, v), προστίθεται μια νέα ετικέτα ίση με $(d_1+l_1(e), \ldots, d_k+l_k(e))_v$ στις ετικέτες του v, και οι ετικέτες επί των οποίων δεν επικρατεί κάποια άλλη αφαιρούνται από το σύνολο των ετικετών του v.

Ο αλγόριθμος επεξεργάζεται τις εξερχόμενες ακμές του τελικού κόμβου v για κάθε Pareto-βέλτιστη s-v-διαδρομή, που καταστρέφει το χρόνο τρεξίματος του αλγορίθμου. Έτσι ίσως χρειαστεί εκθετικός χρόνος τρεξίματος ως προς τον αριθμό των κόμβων, αλλά αν ο αριθμός των Pareto-βέλτιστων διαδρομών ανά κόμβο είναι φραγμένος από κάποια σταθερά, ο χρόνος τρεξίματος είναι ο ίδιος με αυτόν του αλγορίθμου του Dijkstra.

Στην περίπτωση του μονού ζεύγους (δηλαδή όταν ο κόμβος προορισμού t είναι γνωστός όπως και στην εφαρμογή μας), ο αλγόριθμος μπορεί να τερματιστεί όταν βρεθούν όλες οι βέλτιστες κατά Pareto διαδρομές στον t. Πιο συγκεκριμένα, στη δικριτητιριακή περίπτωση (δηλαδή k=2), η συντομότερη s-t-διαδρομή P' ως προς το δεύτερο κριτήριο l_2, μπορεί να υπολογιστεί εκ των προτέρων χρησιμοποιώντας τον αλγόριθμο του Dijkstra, και ο labelling αλγόριθμος μπορεί να τερματιστεί όταν βρεθεί μια Pareto-βέλτιστη s-t-διαδρομή μήκους (d_1, d_2) με $d_2 = l_2(P')$.

Παρόμοιες τεχνικές χρησιμοποιούνται επίσης σε προσεγγίσεις ετικέτας για το *πρόβλημα συντομότερης διαδρομής περιορισμένων πόρων* [*(resource) constrained shortest path problem*], το οποίο είναι επίσης NP-hard. Όπως και στο κανονικό πρόβλημα συντομότερης διαδρομής, υπάρχει ένα μήκος ακμής l ανά ακμή. Επιπρόσθετα, στις ακμές εκχωρούνται k πόροι, οι οποίοι προσθέτονται επίσης κατά μήκος μιας διαδρομής, και υπάρχει ένα όριο πόρων για κάθε πόρο. Ο στόχος είναι να βρούμε μια συντομότερη διαδρομή σε σχέση με το l που ικανοποιεί όλους τους

περιορισμούς πόρων. Κατά τη διάρκεια του αλγορίθμου ετικέτας, εξετάζονται μόνο οι *υποσχόμενες διαδρομές* [*promising paths*]: μια διαδρομή P είναι μη υποσχόμενη αν ένας πόρος ξεπερνάει το δοσμένο όριο, ή αν το μήκος l(P) ξεπερνάει έναν άνω φραγμό μιας βέλτιστης λύσης.

2.8. Γράφοι και Χρόνος

Ο χρόνος είναι προφανώς ένα σημαντικό ζήτημα στις πληροφορίες χρονοδιαγράμματος. Πρώτα, ορίζουμε τις διακριτές τιμές του χρόνου που χρησιμοποιούνται στα επόμενα κεφάλαια, και έπειτα μελετάμε τα χρονικά-εξαρτώμενα μοντέλα συντομότερων διαδρομών τα όποια προτάθηκαν από τους Orda και Rom. Επίσης ρίχνουμε μια πρώτη ματιά στην έννοια της χρονικής-επέκτασης.

2.8.1. Αξίες του Χρόνου

Ο χρόνος θα μοντελοποιηθεί σε αυτήν την εργασία ως διακριτές τιμές που αναπαριστούν τα λεπτά. Μερικές φορές, μας ενδιαφέρει μόνο ο χρόνος μέσα σε μία ημέρα, και οι τιμές του χρόνου θα είναι στο διάστημα [0,1439]΄ σε άλλες περιπτώσεις είναι κωδικοποιημένη και η ημέρα μέσα στην τιμή του χρόνου μέσω της πρόσθεσης του 1440 επί τον αριθμό ημερών μεταξύ μιας δεδομένης ημέρας αναφοράς και της πραγματικής ημέρας. Ο επίσημος ορισμός είναι αυτός που ακολουθεί.

Ορισμός 2.4 Μια τιμή χρόνου t∈IN αναπαριστά λεπτά που έχουν περάσει από τα μεσάνυχτα μιας συγκεκριμένης ημέρας αναφοράς. Μια τέτοια τιμή χρόνου είναι της μορφής t = a1440 + b, όπου a∈IN και b∈[0, 1439]. Ως εκ τούτου, ο πραγματικός χρόνος μέσα σε μία ημέρα είναι t(mod1440) και η πραγματική ημέρα είναι ⌊t/1440⌋.

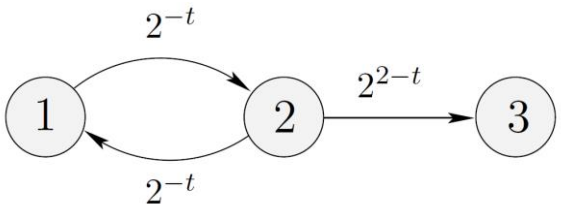
Ο συμβολισμός ⌊x⌋ αναφέρεται ως “το πάτωμα του x”, και εκφράζει τον μέγιστο ακέραιο που είναι μικρότερος από ή ίσος με το x.

Με δεδομένες δύο τιμές χρόνου t και t’, χρειάζεται συχνά να αναφερθούμε στον χρόνο που περνάει ξεκινώντας από τον χρόνο της μέρας που αναπαρίσταται με t μέχρι τον χρόνο της μέρας που αναπαρίσταται με t’ (ανεξάρτητα από την πραγματική μέρα που αναπαριστούν τα t και t’). Αναφερόμαστε σε αυτήν τη διαφορά χρόνου ως το day-diff(t,t’). Για παράδειγμα, αν το t αναπαριστά τις 2 η ώρα το μεσημέρι και το t’ τις 10 η ώρα το πρωί, το day-diff(t,t’) είναι 20 ώρες. Να σημειωθεί ότι αυτή η διαφορά ώρας δεν είναι συμμετρική, για παράδειγμα, χρησιμοποιώντας τις προηγούμενες τιμές το day-diff(t’,t) είναι 4 ώρες.

Ορισμός 2.5 Με δεδομένες δύο τιμές χρόνου t και t’, το day-diff(t, t’) είναι ο μικρότερος μη αρνητικός ακέραιος αριθμός l έτσι ώστε l ≡ t’- t(mod 1440).

2.8.2. Χρονικά-Εξαρτώμενες Συντομότερες Διαδρομές

Οι Orda και Rom (Orda 1991) ερεύνησαν μια προσέγγιση μοντελοποίησης της συντομότερης διαδρομής σε γράφους των οποίων τα βάρη ακμών εξαρτώνται από τον χρόνο. Έστω T το σύνολο που αναπαριστά τον χρόνο (στην περίπτωση των πληροφοριών χρονοδιαγράμματος είναι συνήθως T = IN). Κάθε φορά που προσπερνιέται μια ακμή, η *καθυστέρηση* [*delay*] σε αυτή την ακμή εξαρτάται από τον χρόνο και μοντελοποιείται ως μια συνάρτηση d_e: T → T. Περαιτέρω, τα βάρη ακμών είναι επίσης συναρτήσει του χρόνου, για παράδειγμα w_e: T → IR. Γενικά, επιτρέπεται η αναμονή στους κόμβους, αλλά υπάρχουν επίσης και κόστη αναμονής στους κόμβους, τα οποία μπορούν να χρησιμοποιηθούν για να αποτρέψουν την αναμονή χρησιμοποιώντας υψηλά κόστη αναμονής. Δεδομένης μιας διαδρομής και των χρόνων αναμονής στους ενδιάμεσους κόμβους, το βάρος της διαδρομής μπορεί να υπολογιστεί χρησιμοποιώντας τις συναρτήσεις καθυστέρησης και βάρους. Ο στόχος είναι να βρούμε, δεδομένου ενός κόμβου και ενός χρόνου αναχώρησης, μια διαδρομή (μαζί με τους χρόνους αναμονής στους κόμβους) με το μικρότερο βάρος. Προκύπτει ότι, στη γενική περίπτωση, το πρόβλημα είναι δύσκολο να λυθεί, αφού ακόμα και άπειρες συντομότερες διαδρομές είναι πιθανές, όπως φαίνεται στο Σχήμα 2.6.



Σχήμα 2.6 Δείγμα χρονικά-εξαρτώμενου γράφου με μια άπειρη συντομότερη 1-3-διαδρομή. Οι ετικέτες των ακμών υποδηλώνουν τα χρονικά-εξαρτώμενα βάρη, ενώ η καθυστέρηση είναι ίση με 1 για κάθε ακμή.

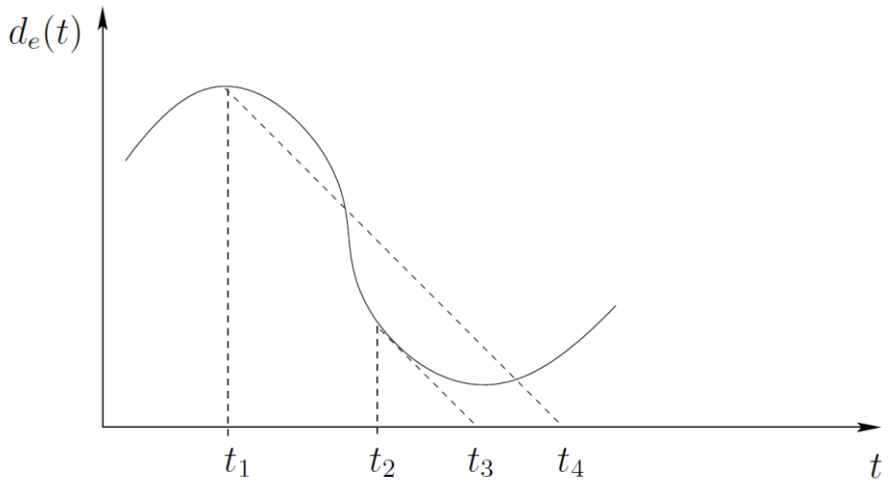
Ωστόσο, οι Orda και Rom έχουν δείξει επίσης, ότι σε περιορισμένες περιπτώσεις, ένας αλγόριθμος σαν του Dijkstra μπορεί να χρησιμοποιηθεί για να προσδιορίσει χρονικά-εξαρτώμενες συντομότερες διαδρομές. Σε αυτόν τον αλγόριθμο, δεν υπάρχουν πλέον βάρη, και ο στόχος είναι να ελαχιστοποιηθεί ο χρόνος άφιξης (δηλαδή, το πρόβλημα αντιστοιχεί στην εύρεση μιας ταχύτερης διαδρομής). Αν η αναμονή απαγορεύεται, το πρόβλημα μπορεί να αποδειχθεί πως είναι NP-hard. Όταν όμως επιτρέπεται η απεριόριστη αναμονή και όλες οι καθυστερήσεις είναι μη-αρνητικές (δηλαδή, για κάθε t, $d_e(t) \geq 0$), μια τροποποίηση του αλγόριθμου του Dijkstra λύνει το πρόβλημα: Οι ετικέτες απόστασης αντιπροσωπεύουν τους χρόνους άφιξης στους κόμβους, και κάθε φορά που μια ακμή e = (u, v) πρόκειται να χαλαρωθεί και t_u είναι ο χρόνος άφιξης στον κόμβο u, ο βέλτιστος χρόνος αναμονής τ_e για την e μπορεί να καθοριστεί με την ελαχιστοποίηση του $t_u + d_e(t_u + \tau_e)$. Η τελευταία τιμή χρόνου είναι η μη επιβεβαιωμένη ώρα άφιξης στον κόμβο v στον αλγόριθμο του Dijkstra.

Η αναμονή σε κόμβους μπορεί να αποδώσει: Σκεφτείτε για παράδειγμα τη συνάρτηση καθυστέρησης d_e της ακμής e=(u, v) που φαίνεται στο Σχήμα 2.7. Σε αυτήν την περίπτωση, ακόμη και αν η νωρίτερη άφιξη στο u είναι t_1, ένας χρόνος αναμονής του $\tau_e=t_2-t_1$ αποδίδει τη νωρίτερη ώρα άφιξης στο v. Η πιθανότητα αναμονής στους κόμβους συνεπάγεται και την ανάγκη για υπολογισμό των βέλτιστων χρόνων αναμονής, που μπορεί να μην είναι πολύ αποτελεσματικό. Ωστόσο, για μια περιορισμένη κατηγορία συναρτήσεων καθυστέρησης, τις FIFO (first in first out) καθυστερήσεις, η αναμονή δεν αποδίδει ποτέ και ο υπολογισμός των χρόνων αναμονής είναι εύκολος. Μια συνάρτηση καθυστέρησης είναι FIFO, αν πληροί τις ακόλουθες προϋποθέσεις:

$$t_1 \leq t_2 \Rightarrow t_1 + d_e(t_1) \leq t_2 + d_e(t_2).$$

Για μια συνάρτηση καθυστέρησης FIFO το d_e μιας ακμής e = (u, v), ο βέλτιστος χρόνος άφιξης t_v στον v μπορεί να προσδιοριστεί απλά με αξιολόγηση του $d_e(t_u)$, όπου t_u είναι η ώρα άφιξης στον u. Η παραπάνω συζήτηση συνοψίζεται με το ακόλουθο λήμμα.

Λήμμα 2.1 Με δεδομένο ένα χρονικά-εξαρτώμενο δίκτυο με καθυστερήσεις ακμών d(e), εάν το d είναι μη αρνητικό και FIFO, μία συντομότερη διαδρομή όσον αφορά το d (που ονομάζεται επίσης και ταχύτερη διαδρομή) μπορεί να υπολογιστεί αποτελεσματικά από την προαναφερθείσα παραλλαγή του αλγορίθμου του Dijkstra.



Σχήμα 2.7 Η απόδοση της αναμονής για μια ακμή e = (u, v) και η εικονιζόμενη συνάρτηση αναμονής d_e. Αν t_1 είναι η ώρα άφιξης στο u, η μη αναμονή συνεπάγεται ώρα άφιξης t_4 στον κόμβο v, ενώ το να περιμένουμε μέχρι t_2 αποδίδει άφιξή στο $t_3 < t_4$.

.

2.8.3. Χρονική-Επέκταση

Αντί να μοντελοποιήσουμε τον χρόνο χρησιμοποιώντας συναρτήσεις καθυστέρησης και βάρους που εξαρτώνται από τον χρόνο, όπως περιεγράφηκε στην προηγούμενη παράγραφο, η προσέγγιση της χρονικής-επέκτασης μεταφράζει τις καθυστερήσεις και τα βάρη που εξαρτώνται από τον χρόνο σε έναν στατικό *χρονικά-επεκτεινόμενο γράφο* [*time-expanded graph*]. Αυτή η προσέγγιση απαιτεί διακριτές τιμές χρόνου, και στη γενική περίπτωση για κάθε τιμή του χρόνου διατηρείται ένα αντίγραφο του κόμβου στη χρονικά-επεκτεινόμενη εκδοχή του γράφου. Ένα σημαντικό πρόβλημα με τη χρονική-επέκταση είναι το τεράστιο μέγεθος του γράφου που προκύπτει· μπορεί να είναι ακόμη και κυριολεκτικά αδύνατο να αποθηκεύσουμε έναν τέτοιο γράφο.

Δεδομένου ότι στις πληροφορίες δρομολογίων οι τιμές χρόνου είναι διακριτές, η χρονική-επέκταση μπορεί να εφαρμοστεί άμεσα. Στην πραγματικότητα, μόνο για ορισμένες τιμές του χρόνου που σχετίζονται με ένα γεγονός του χρονοδιαγράμματος πρέπει να διατηρείται στον χρονικά-επεκταμένο γράφο ένα αντίγραφο του κόμβου (π.χ., μια αναχώρηση ή άφιξη μιας αμαξοστοιχίας). Ως εκ τούτου, ο γράφος που προκύπτει δεν είναι τόσο μεγάλος όσο στη γενική περίπτωση. Θα συζητήσουμε την εφαρμογή της χρονικής-επέκτασης στις πληροφορίες χρονοδιαγράμματος με περισσότερες λεπτομέρειες στο επόμενο κεφάλαιο.

3. Το Βασικό Πρόβλημα Πληροφοριών Χρονοδιαγράμματος

3.1. Προσδιορισμός Προβλήματος

Σε αυτήν την ενότητα, παρέχουμε μια μαθηματική διατύπωση των βασικών *δεδομένων χρονοδιαγράμματος* [*timetable data*] που χρησιμοποιούμε σε αυτό το κεφάλαιο, και έπειτα ορίζουμε το πρώτο και πιο θεμελιώδες πρόβλημα χρονοδιαγράμματος, το *Πρόβλημα Νωρίτερης Άφιξης (ΠΝΑ)* [*Earliest Arrival Problem (EAP)*]. Χρησιμοποιούμε ορολογία από σιδηροδρομικές μεταφορές λόγω της επικεντρωμένης φύσεως της παρούσας εργασίας, αλλά φυσικά οι ορισμοί του προβλήματος, των μοντέλων και των αλγορίθμων μπορούν να εφαρμοστούν επίσης και σε άλλους τομείς με παρόμοια χαρακτηριστικά.

3.1.1. Δεδομένα

Ένα *χρονοδιάγραμμα* [*timetable*] αποτελείται από δεδομένα που αφορούν: σταθμούς (ή στάσεις λεωφορείων, λιμάνια, κλπ.), τρένα (ή λεωφορεία, πλοία, κλπ.), σταθμούς ανταπόκρισης, ώρες αναχώρησης και άφιξης των τρένων στους σταθμούς, και τις ημέρες κυκλοφορίας. Πιο επίσημα, μας δίνεται ένα σύνολο τρένων Z, ένα σύνολο σταθμών B, καθώς και μια σειρά στοιχειωδών συνδέσεων C, της οποία τα στοιχεία c είναι διανύσματα της μορφής $c=(Z, S_1, S_2, t_d, t_a)$. Μια τέτοια πλειάδα (δηλαδή μια στοιχειώδης σύνδεσης) ερμηνεύεται ως το γεγονός ότι το τρένο Z αφήνει το σταθμό S_1 στο χρόνο t_d, και η επόμενη στάση του τρένου Z είναι ο σταθμός S_2 στο χρόνο t_a. Εάν το x συμβολίζει ένα πεδίο πλειάδας, τότε το x(c) καθορίζει την τιμή του x στη στοιχειώδη σύνδεση c.

Οι ώρες αναχώρησης και άφιξης $t_d(c)$ και $t_a(c)$ μιας στοιχειώδους σύνδεσης $c \in C$ μέσα σε μια ημέρα είναι ακέραιοι και ανήκουν στο διάστημα [0,1439], το οποίο αντιπροσωπεύει τον χρόνο σε λεπτά μετά τα μεσάνυχτα. Το μήκος μιας στοιχειώδους σύνδεσης c, που συμβολίζεται ως length(c), είναι το day-diff($t_d(c)$,$t_a(c)$), και παριστά τον χρόνο που περνά μεταξύ της αναχώρησης και της άφιξης (βλ. επίσης και τον ορισμό των day-diff που παρατίθεται στην παράγραφο 2.10.1).

3.1.2. Το Πρόβλημα Νωρίτερης Άφιξης

Έστω $P=(c_1, ..., c_k)$ μια ακολουθία στοιχειωδών συνδέσεων μαζί με τους χρόνους αναχώρησης $dep_i(P)$ και τους χρόνους άφιξης $arr_i(P)$ για κάθε στοιχειώδη σύνδεση c_i, $1 \leq i \leq k$. Υποθέτουμε ότι οι χρόνοι $dep_i(P)$ και $arr_i(P)$, περιλαμβάνουν δεδομένα που αφορούν επίσης και στην ημέρα αναχώρησης/άφιξης μετρώντας τον χρόνο σε λεπτά από την "πρώτη" ημέρα του χρονοδιαγράμματος. Μια τιμή χρόνου t είναι της μορφής $t=a\cdot1440+b$, όπου $a\in[0, 364]$ και $\beta\in[0, 1439]$. Ως εκ τούτου, ο πραγματικός χρόνος μέσα σε μια ημέρα είναι $t \pmod{1440}$ και η πραγματική ημέρα είναι $[t/1440]$.

Μια τέτοια ακολουθία P ονομάζεται *συνεπής σύνδεση* [*consistent connection*] από τον σταθμό $A=S_1(c_1)$ στον σταθμό $B=S_2(c_k)$ εφόσον πληροί ορισμένες προϋποθέσεις συνοχής: ο σταθμός αναχώρησης του c_{i+1} είναι ο σταθμός άφιξης του c_i και οι τιμές χρόνου $dep_i(P)$ και $arr_i(P)$ αντιστοιχούν στις τιμές χρόνου t_d και t_a, αντίστοιχα των στοιχειωδών συνδέσεων (modulo 1440), και οι αναχωρήσεις από κάθε στοιχειώδη σύνδεση (εκτός από την πρώτη) γίνονται αργότερα από τις προηγούμενες αφίξεις. Πιο επίσημα, η P είναι μια συνεπής σύνδεση εάν πληρούνται οι ακόλουθες προϋποθέσεις:

- $S_2(c_i) = S_2(c_{i+1})$
- $dep_i(P) \equiv t_d(c_i) \pmod{1440}$
- $arr_i(P) = length(c_i) + dep_i(P)$
- $dep_{i+1}(P) - arr_i(P) \geq 0$

Να σημειωθεί ότι με αυτόν τον ορισμό υποθέτουμε ότι είναι δυνατή η μετεπιβίβαση σε οποιονδήποτε σταθμό σε οποιοδήποτε τρένο αναχωρεί αργότερα ή στον ίδιο χρόνο με τον χρόνο άφιξης στον συγκεκριμένου σταθμό. Υποθέτουμε περαιτέρω ότι όλα τα τρένα λειτουργούν καθημερινά. Εξαιτίας αυτών των απλοποιήσεων, αναφερόμαστε στο πρόβλημα που ορίζεται εδώ και ως *απλοποιημένο πρόβλημα νωρίτερης άφιξης* [*simplified earliest arrival problem*]. Στο επόμενο κεφάλαιο θα συζητήσουμε πώς μπορούν να συμπεριληφθούν πιο ρεαλιστικοί κανόνες μετεπιβίβασης καθώς και οι ημέρες κυκλοφορίας, οι οποίες προσδιορίζουν τις ημέρες λειτουργίας των τρένων.

Παρουσιάζεται επίσης μια μεγάλη, on-line σειρά *επερωτημάτων* [*queries*]. Ένα επερώτημα γενικά καθορίζει ένα σύνολο έγκυρων συνδέσεων, και ένα κριτήριο (ή κριτήρια) βελτιστοποίησης σε αυτό το σύνολο των συνδέσεων. Το θέμα είναι να βρεθεί η βέλτιστη σύνδεση (ή ένα σύνολο βέλτιστων συνδέσεων) η οποία να σέβεται το συγκεκριμένο κριτήριο (ή κριτήρια). Για το πρόβλημα νωρίτερης άφιξης, ένα επερώτημα (A,B,t_0) αποτελείται από έναν σταθμό αναχώρησης A, έναν σταθμό άφιξης B, και την ώρα αναχώρησης t_0. Η ώρα αναχώρησης βρίσκεται στο [0,1439] για την απλουστευμένη εκδοχή. Οι συνδέσεις είναι έγκυρες αν δεν αναχωρούν πριν από τον δοσμένο χρόνο αναχώρησης t_0, και το κριτήριο βελτιστοποίησης είναι η ελαχιστοποίηση της διαφοράς μεταξύ του χρόνου άφιξης και του δεδομένου χρόνου αναχώρησης.

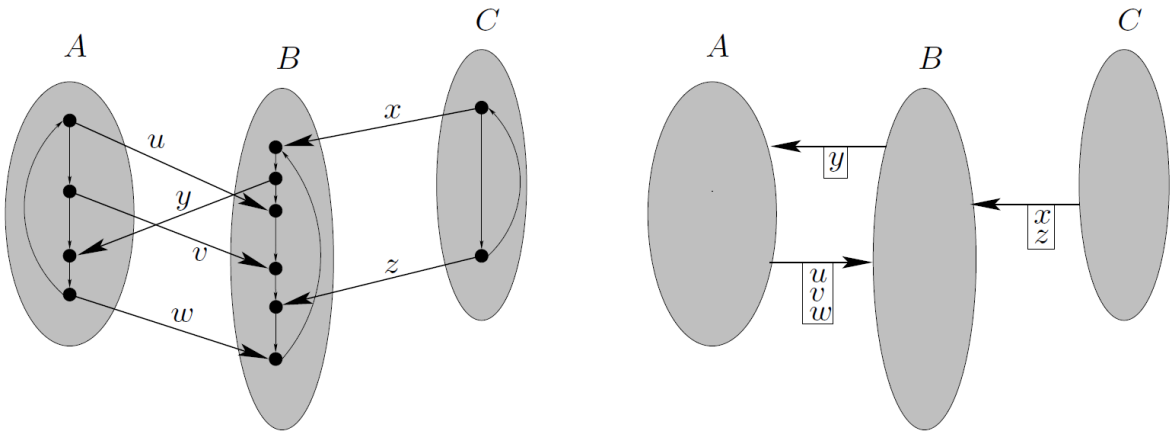
3.2. Βασικές Προσεγγίσεις Μοντελοποίησης

Εξετάζουμε τη μοντελοποίηση της απλοποιημένης εκδοχής του ΠΝΑ για δύο προσεγγίσεις: τη *χρονικά-επεκτεινόμενη* [*time-expanded*], καθώς και τη *χρονικά-εξαρτώμενη* [*time-dependent*] προσέγγιση. Επιπλέον, συζητάμε κάποιες ακόμη ιδέες και ομοιότητες των δυο προσεγγίσεων.

3.2.1. Χρονικά-Επεκτεινόμενο Μοντέλο

Το χρονικά-επεκτεινόμενο μοντέλο (Schulz, Wanger and Weihe 2000), βασίζεται στις αρχές της χρονικής-επέκτασης όπως αναφέρθηκαν στην Παράγραφο 2.8.3, και βασίζεται στον χρονικά-επεκτεινόμενο γράφο ο οποίος κατασκευάζεται ως εξής. Υπάρχει ένας κόμβος για κάθε χρονικό συμβάν σε έναν σταθμό (αναχώρηση ή άφιξη), και υπάρχουν δύο τύποι ακμών. Για κάθε στοιχειώδη σύνδεση (Z, S_1, S_2, t_d, t_a) στο χρονοδιάγραμμα, υπάρχει μια *ακμή-τρένου* [*train-edge*] στον γράφο που συνδέει έναν κόμβο αναχώρησης, που ανήκει στον σταθμό S_1 και σχετίζεται με τον χρόνο t_d, με έναν κόμβο άφιξης, που ανήκει στο σταθμό S_2 και σχετίζεται με το χρόνο t_a. Με άλλα λόγια, τα άκρα των ακμών-τρένου αποτελούν το σύνολο των κόμβων του γράφου. Για κάθε σταθμό S, όλοι οι κόμβοι που ανήκουν στο S διατάσσονται σύμφωνα με τις τιμές των χρόνων τους. Έστω v_1, ..., v_κ οι κόμβοι του S σε αυτή τη σειρά. Στη συνέχεια, υπάρχει μια σειρά από *ακμές-αναμονής* [*stay-edges*] (v_i, v_{i+1}), $1 \leq i \leq k\text{-}1$, και (v_k, v_1) που συνδέουν τα χρονικά γεγονότα μέσα σε έναν σταθμό και αντιπροσωπεύουν την αναμονή σε αυτόν τον σταθμό. Το μήκος της ακμής (u, v) είναι το day-diff(t_v, t_u), όπου t_u και t_u είναι οι τιμές του χρόνου που σχετίζεται με υ και ν αντίστοιχα. Το Σχήμα 3.1 απεικονίζει τον ορισμό αυτό. Το ακόλουθο θεώρημα δηλώνει ότι ένας κατάλληλος υπολογισμός συντομότερης διαδρομής στον παραπάνω γράφο λύνει την απλοποιημένη εκδοχή του ΠΝΑ.

Θεώρημα 3.1 Μια συντομότερη διαδρομή σε έναν χρονικά-επεκτεινόμενο γράφο από τον πρώτο κόμβο αναχώρησης s στο σταθμό αναχώρησης A, με ώρα αναχώρησης αργότερη ή ίση με τον δοσμένο χρόνο εκκίνησης t_0, προς έναν από τους κόμβους άφιξης του σταθμού προορισμού B αποτελεί μια λύση για την απλοποιημένη εκδοχή του ΠΝΑ στο χρονικά-επεκτεινόμενο μοντέλο.



Σχήμα 3.1 Ο χρονικά-επεκτεινόμενος γράφος (αριστερά) και ο χρονικά-εξαρτώμενος γράφος (δεξιά) ενός χρονοδιαγράμματος με τρεις σταθμούς A, B, C. Υπάρχουν τρία τρένα που συνδέουν το A με το B (στοιχειώδης συνδέσεις U, V, W), ένα τρένο από το C μέσω του B στο A(x, y) και ένα τρένο από το C στο B(z).

Ο Πλήρως Χρονικά-Επεκτεινόμενος Γράφος

Αντί για την εισαγωγή ακμών-παραμονής από τον τελευταίο κόμβο ως τον πρώτο κόμβο σε κάθε σταθμό, ο χρονικά-επεκτεινόμενος γράφος μπορεί επίσης να επεκταθεί πλήρως διατηρώντας ένα αντίγραφο κάθε γεγονότος για κάθε ημέρα κατά τη διάρκεια της περιόδου του χρονοδιαγράμματος. Αυτό αντικατοπτρίζει πιο άμεσα τη γενική ιδέα της χρονικής-επέκτασης που περιγράφεται στην Παράγραφο 2.8.3. Μια πραγματική κατασκευή ενός τέτοιου γράφου συνήθως δεν είναι δυνατή (θα ήταν πάρα πολύ μεγάλο για πρακτικές εφαρμογές), αλλά έχει την καλή ιδιότητα ότι υπάρχει μια αντιστοιχία ένα-προς-ένα μεταξύ όλων των συνδέσεων -όχι μόνο αυτών με χρόνους παραμονής λιγότερο μια ημέρας- και των διαδρομών.

Εάν το χρονοδιάγραμμα είναι έγκυρο για N μέρες, ο *πλήρως χρονικά-επεκτεινόμενος γράφος* [*fully time-expanded graph*] είναι βασισμένος σε N αντίγραφα του χρονικά-επεκτεινόμενου γράφου. Κάθε φορά που υπάρχει μια ακμή που ξεπερνά τα μεσάνυχτα στο i-στο αντίγραφο, η ακμή ανακατευθύνεται στον αντίστοιχο κόμβο του (i+1)-στου αντιγράφου, για i<n· στο N-στο αντίγραφο, οι ακμές που ξεπερνούν τα μεσάνυχτα διαγράφονται. Στη συνέχεια, υποθέτουμε ότι κάθε κόμβος στον πλήρως χρονικά-επεκτεινόμενου γράφου δεν είναι μόνο αντιστοιχισμένος με τον χρόνο t_d στο διάστημα [0,1439], αλλά και με τον απόλυτο χρόνο μέσα στο χρονοδιάγραμμα, δηλαδή στον κόμβο του αντιγράφου που αντιστοιχεί στην ημέρα i έχει ανατεθεί ο χρόνος $t = t_d + i \cdot 1440$.

Υπάρχει μια αντιστοιχία ένα-προς-ένα μεταξύ (έγκυρων και συνεπών) συνδέσεων και διαδρομών στον πλήρως χρονικά-επεκτεινόμενο γράφο. Έτσι, μια κατάλληλη συντομότερη διαδρομή στον πλήρως χρονικά-επεκτεινόμενο γράφο παρέχει μια λύση στο πρόβλημα της νωρίτερης άφιξης. Τέλος, θα αναφέρουμε τις εξής δύο σημαντικές ιδιότητες του πλήρως χρονικά-επεκτεινόμενου γράφου:

- Η πρώτη ιδιότητα είναι ότι δεν περιέχει καθόλου κατευθυνόμενους κύκλους και κατά συνέπεια ανήκει στην κατηγορία γράφων Κ.Μ.Κ.Γ. (Κατευθυνόμενος Μη Κυκλικός Γράφος) [D.A.G.].
- Η δεύτερη ιδιότητα είναι ότι οποιεσδήποτε δύο s-t-διαδρομές που συνδέουν τους δυο ίδιους κόμβους s και t έχουν το ίδιο μήκος.

Αλγόριθμος

Αυτές οι δύο ιδιότητες μας υποδεικνύουν ότι η συντομότερη διαδρομή μπορεί να βρεθεί σε γραμμικό χρόνο: Στα D.A.G.s, μπορεί να χρησιμοποιηθεί μια τοπολογική ταξινόμηση των κόμβων όπως περιγράφεται στην Ενότητα 2.8. Περαιτέρω, εάν δύο s-t-διαδρομές έχουν πάντοτε το ίδιο μήκος, κάθε αλγόριθμος αναζήτησης στον πλήρως χρονικά-επεκτεινόμενο γράφο, όπως για παράδειγμα η *Αναζήτηση Πρώτα-Πλάτους* [Breadth-First Search], θα αρκούσε για να καθορίσουμε τον πρώτο εφικτό κόμβο του σταθμού στόχου μας.

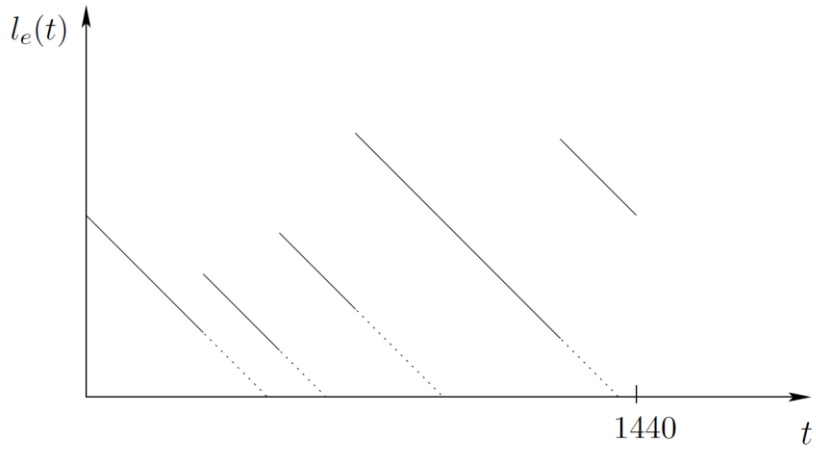
Ωστόσο, η άμεση εφαρμογή τέτοιων διαδικασιών θα απαιτούσε τον τεράστιο πλήρως χρονικά-επεκτεινόμενο γράφο. Αντ' αυτού, προτείνεται η εφαρμογή της υλοποίησης του Dial στον αλγόριθμου του Dijkstra στον χρονικά-επεκτεινόμενο γράφο που επωφελείται από τις εξής ειδικές ιδιότητες:

- Εφόσον τα μήκη ακμών είναι μη αρνητικά, η πραγματική συντομότερη διαδρομή στον χρονικά-επεκτεινόμενο γράφο μπορεί να βρεθεί με τον αλγόριθμο του Dijkstra.
- Όλα τα μήκη ακμών είναι φυσικοί αριθμοί μικρότεροι ή ίσοι του 1440, οπότε μπορούμε να χρησιμοποιήσουμε την υλοποίηση κάδων του Dial για ουράς προτεραιότητας και κατά συνέπεια ο χρόνος τρεξίματος είναι γραμμικός ως προς τον αριθμό των κόμβων.
- Στην πραγματικότητα, επεξεργαζόμαστε μόνο ένα κλάσμα των κόμβων, διότι ο κύριος βρόχος μπορεί να ματαιωθεί όταν επιτευχθεί ένας κόμβος στον σταθμό προορισμού.

Ο αλγόριθμος του Dijkstra επεξεργάζεται τους κόμβους κατά σειρά απόστασης από τον κόμβο πηγή. Στην πραγματικότητα, επεξεργάζονται όλοι οι προσβάσιμοι κόμβοι από τον κόμβο εκκίνησης διατεταγμένοι κατά τον χρόνο. Από τη μία πλευρά, η διάταξη όλων των κόμβων του πλήρως χρονικά-επεκτεινόμενου γράφου κατά χρόνο είναι μια τοπολογική ταξινόμηση, αφού δεν υπάρχει ακμή προς τα πίσω στο χρόνο. Έτσι, ο αλγόριθμος του Dijkstra επεξεργάζεται τους κόμβους του πλήρως χρονικά-επεκτεινόμενου γράφου με τοπολογική ταξινόμηση. Από την άλλη πλευρά, μια πολύ καλή ιδέα είναι η αναζήτηση των προσβάσιμων κόμβων του γράφου με αύξηση χρόνου, αφού μπορούμε να ματαιώσουμε την αναζήτηση με το που φτάσουμε στον πρώτο κόμβο του σταθμού προορισμού, κάτι το οποίο είναι ακριβώς αυτό που κάνει ο αλγόριθμος του Dijkstra όταν εφαρμόζεται στον χρονικά-επεκτεινόμενο γράφο. Ως εκ τούτου, όταν εφαρμόζεται η υλοποίηση του Dial για τον αλγόριθμο του Dijkstra στον πλήρως χρονικά-επεκτεινόμενο γράφο με την ειδική δομή της, είναι μια αναζήτηση γραμμικού χρόνου σε έναν γράφο τοπολογικά ταξινομημένο, η οποία μπορεί να τερματιστεί όταν επιτευχθεί ο πρώτος κόμβος στον σταθμό προορισμού.

3.2.2. Χρονικά-Εξαρτώμενο Μοντέλο

Το χρονικά-εξαρτώμενο μοντέλο (Brodal and Jacob 2004) χρησιμοποιεί τη χρονικά-εξαρτώμενη συντομότερη διαδρομή που ερευνήθηκε από τους Orda και Rom. Είναι επίσης βασισμένο σε έναν κατευθυνόμενο γράφο, που ονομάζεται χρονικά-εξαρτώμενος γράφος. Σε αυτόν τον γράφο, υπάρχει μόνο ένας κόμβος ανά σταθμό, και υπάρχει μια ακμή e από τον σταθμό A στον σταθμό B εάν υπάρχει μια στοιχειώδης σύνδεση από τον A στον B. Το σύνολο των στοιχειωδών συνδέσεων από τον A στον B συμβολίζεται με C(e). Ο ορισμός απεικονίζεται στο Σχήμα 3.2. Το μήκος μιας ακμής e=(v, w) εξαρτάται από τον χρόνο κατά τον οποίο αυτή η συγκεκριμένη ακμή θα χρησιμοποιηθεί κατά τη διάρκεια του αλγορίθμου που λύνει το ΠΝΑ. Με άλλα λόγια, εάν το T είναι ένα σύνολο που δηλώνει χρόνο, τότε το μήκος μιας ακμής e=(v,w) δίνεται από $l_e(t)=f_{(v,w)}(t) - t$, όπου t είναι ο χρόνος αναχώρησης από τον v, η f_e: T → T είναι μια συνάρτηση τέτοια ώστε $f_e(t) =t'$, και $t'\geq t$ είναι ο συντομότερος δυνατός χρόνος άφιξης στον w. Το μήκος της χρονικά-εξαρτώμενης ακμής l_e είναι μια συνάρτηση καθυστέρησης στην ορολογία των χρονικά-εξαρτώμενων συντομότερων διαδρομών που παρουσιάστηκε στην Παράγραφο 2.8.2. Ένα δείγμα συνάρτησης καθυστέρησης l_e όπως ορίζεται παραπάνω, απεικονίζεται στο Σχήμα 3.2.



Σχήμα 3.2 Δείγμα συνάρτησης καθυστέρησης l_e που χρησιμοποιείται στο χρονικά-εξαρτώμενο μοντέλο για το πρόβλημα νωρίτερης άφιξης. Οι διακεκομμένες γραμμές που τέμνονται με την τετμημένη υποδεικνύουν τους χρόνους άφιξης $f_e(t)$ στους σταθμούς άφιξης και άλματα προκύπτουν όταν ένα τρένο αναχωρεί. Μετά από μία ημέρα (1440 λεπτά), η συνάρτηση επαναλαμβάνεται, δηλαδή $l_e(t+1.440) = l_e(t)$.

Όπως έχουμε ήδη πει και στο Λήμμα 2.1, μια χρονικά-εξαρτώμενη συντομότερη διαδρομή μπορεί να βρεθεί εύκολα όταν οι συναρτήσεις καθυστέρησης είναι μη-αρνητικές και FIFO: μια τροποποίηση του αλγορίθμου του Dijkstra μπορεί να χρησιμοποιηθεί, όπου οι συναρτήσεις καθυστέρησης πρέπει να αξιολογήθηκαν. Στην περίπτωσή μας, κάθε μήκος l_e χρονικά-εξαρτώμενης ακμής είναι μη-αρνητικό από κατασκευής· εμείς πρέπει να διασφαλίσουμε ότι είναι FIFO:

$$t \leq t' \Rightarrow t + l_e(t) \leq t' + l_e(t')$$

Χρησιμοποιώντας τις συναρτήσεις f_e της νωρίτερης άφιξης, μια απλούστερη και ίση προϋπόθεση είναι ότι το f_e είναι μη φθίνον:

$$t \leq t' \Rightarrow f_e(t) \leq f_e(t')$$

Το νόημα της προϋπόθεσης είναι ότι δεν επιτρέπεται η προσπέραση των τρένων σε μια ακμή, και από τώρα και στο εξής, θα υποθέτουμε ότι εκπληρώνεται πάντα η προϋπόθεση, κάτι το οποίο ισχύει για τα δεδομένα μας. Είναι επίσης εύλογο, εφόσον αν ποτέ ένα τρένο προσπεράσει ένα άλλο τρένο κάπου μεταξύ δύο σταθμών, το πρώτο τρένο είναι πιθανόν γρήγορο τρένο υψηλής ταχύτητας και το δεύτερο τρένο είναι ένα πιο αργό τρένο· αυτά τα δύο τρένα συνήθως δεν σταματούν στους ίδιους σταθμούς (το πιο αργό τρένο σταματά πιο συχνά), και ως εκ τούτου δεν ανήκουν στην ίδια ακμή.

Παραδοχή 3.1 Για οποιουσδήποτε δύο δοσμένους σταθμούς A και B, δεν υπάρχουν δύο τρένα που αναχωρούν από τον A και φθάνουν στον B έτσι ώστε το τρένο που αναχωρεί από τον A δεύτερο φθάνει στον B πρώτο.

Αλγόριθμος

Στη συνέχεια, θα εξηγήσουμε την τροποποίηση του αλγόριθμου του Dijkstra που μπορεί να χρησιμοποιηθεί για την επίλυση του προβλήματος νωρίτερης άφιξης στο χρονικά-εξαρτώμενο μοντέλο με περισσότερες λεπτομέρειες. Έστω D ο σταθμός αναχώρησης και t_0, ο νωρίτερος χρόνος αναχώρησης. Οι διαφορές, σε σχέση με τον αλγόριθμο του Dijkstra, είναι:

Θέτουμε την ετικέτα απόστασης δ(D) του κόμβου εκκίνησης ο οποίος αντιστοιχεί στον σταθμό αναχώρησης D σε t_0 (και όχι σε μηδέν), και υπολογίζουμε τα μήκη ακμών κατά τη διάρκεια. Τα μήκη ακμών υπολογίζονται ως εξής: Εφόσον ο αλγόριθμος του Dijkstra είναι ένας label-setting αλγόριθμος συντομότερης διαδρομής, κάθε φορά που εξετάζεται μια ακμή e=(A,B), η ετικέτα απόστασης δ(A) του κόμβου A είναι η βέλτιστη. Στο χρονικά-εξαρτώμενο μοντέλο, το δ(A) υποδηλώνει τον νωρίτερο χρόνο άφιξης στον σταθμό A. Με άλλα λόγια, μπορούμε πράγματι να γνωρίζουμε τον νωρίτερο χρόνο άφιξης στον σταθμό A, όποτε εξετάζεται η ακμή e=(A,B), και επομένως γνωρίζουμε σε αυτό το στάδιο του αλγορίθμου (με βάση την Παραδοχή 3.1) ποιο τρένο πρέπει να χρησιμοποιηθεί για να φτάσουμε στο σταθμό B μέσω του σταθμού A όσο το δυνατόν νωρίτερα: το πρώτο τρένο που αναχωρεί αργότερα ή τον ίδιον χρόνο με τον χρόνο άφιξης στο A. Έστω $t=\delta(A)$ και το $c^* \in C(e)$ να είναι η σύνδεση που ελαχιστοποιεί $\{\text{day-diff}(t,t_d(c))|c \in C(e)\}$. Η συγκεκριμένη σύνδεση c^* μπορεί να βρεθεί εύκολα μέσω μιας δυαδικής αναζήτησης, αν οι στοιχειώδεις συνδέσεις C(e) διατηρούνται σε ταξινομημένο πίνακα. Το μήκος ακμής του e, $l_e(t)$, ορίζεται τότε ως $l_e(t)=\text{day-diff}(t, t_d(c^*)) + \text{length}(c^*)$. Κατά συνέπεια, $f_e(t) = t + l_e(t)$. Να σημειωθεί ότι στην πραγματικότητα τα μήκη $l_e(t)$ δεν πρέπει να υπολογίζονται ρητά, δεδομένου ότι στον αλγόριθμο του Dijkstra χρειάζεται μόνο η απόσταση (αντίστοιχα. ο χρόνος άφιξης) $f_\varepsilon(t)$ στον κόμβο στόχο (αντίστοιχα. στο σταθμός B), τα οποία μπορούμε να πάρουμε άμεσα από τον χρόνο άφιξης $t_\alpha(c^*)$ της αντίστοιχης σύνδεσης.

Το ακόλουθο θεώρημα αποδεικνύεται και έχει προκύψει επίσης και από τα γενικά αποτελέσματα σε χρονικά-εξαρτώμενες συντομότερες διαδρομές. Η ορθότητα του βασίζεται στους ακόλουθους δύο λόγους:

- τα μήκη των ακμών είναι μη αρνητικά, πράγμα που σημαίνει με άλλα λόγια ότι όλα τα f_e έχουν μη αρνητική καθυστέρηση (για όλα τα t, $f_e(t) \geq t$), και είναι εξ ορισμού αληθές
- τα μήκη των ακμών είναι FIFO, πράγμα που σημαίνει ότι στην περίπτωσή μας όλα τα f_e είναι μη φθίνοντα ($t \leq t' \Rightarrow f_e(t) \leq f_e(t')$) και ακολουθεί άμεσα από την Παραδοχή 3.1.

Λόγω της φύσης της υπό έρευνα εφαρμογής, μπορούμε με ασφάλεια να υποθέσουμε ότι όλες οι συναρτήσεις που θα εξετάζουμε στην συνέχεια έχουν επίσης μη-αρνητική καθυστέρηση.

Θεώρημα 3.2 Ο παραπάνω τροποποιημένος αλγόριθμος του Dijkstra λύνει την απλοποιημένη εκδοχή του ΠΝΑ στο χρονικά-εξαρτώμενο μοντέλο, με την προϋπόθεση ότι ισχύει η Παραδοχή 3.1.

3.2.3. Σύγκριση των Προσεγγίσεων

Στο απλοποιημένο σενάριο που εξετάζουμε σε αυτό το κεφάλαιο, οι γράφοι που χρησιμοποιούνται στις δύο προσεγγίσεις είναι ισχυρά συνδεδεμένοι αφού: συγχωνεύοντας όλους τους κόμβους που ανήκουν στον ίδιο σταθμό του χρονικά-επεκτεινόμενου γράφου και έπειτα διαγράφοντας τις παράλληλες ακμές, προκύπτει ο χρονικά-εξαρτώμενος γράφος. Περαιτέρω, ο αλγόριθμος που χρησιμοποιείται στη χρονικά-εξαρτώμενη προσέγγιση μπορεί να θεωρηθεί ως μία βελτιωμένη υλοποίηση της απλής αναζήτησης συντομότερης διαδρομής του αλγορίθμου του Dijkstra στη χρονικά-επεκτεινόμενη προσέγγιση στην οποία: αν η πρώτη ακμή από κάποιο σταθμό A προς έναν άλλο σταθμό B έχει ήδη επεξεργαστεί από τον αλγόριθμο του Dijkstra στον χρονικά-επεκτεινόμενο γράφο, όλες οι άλλες ακμές e'_{AB} από τον σταθμό A προς τον σταθμό B δεν χρειάζεται πλέον να εξεταστούν. Ο λόγος είναι ότι μια τέτοια ακμή δεν παρέχει κάποια βελτίωση αφού η διαδρομή μέσω της πρώτης ακμής επεκταμένη κατά κάποιες stay-edges έως την άκρη της ακμής e'_{AB} έχει το ίδιο μήκος. Κατά μία έννοια, ο χρονικά-εξαρτώμενος αλγόριθμος υλοποιεί αυτή την παρατήρηση.

Ωστόσο, δεν είναι άμεσα σαφές αν η χρονικά-εξαρτώμενη προσέγγιση είναι ανώτερη. Από τη μια πλευρά τα μήκη των ακμών πρέπει ακόμη να υπολογιστούν στον χρονικά-εξαρτώμενο αλγόριθμο, το οποίο όμως καταναλώνει επίσης χρόνο τρεξίματος. Υποστηρίζεται θεωρητικά ότι στον χρονικά-εξαρτώμενο αλγόριθμο πρέπει να εκτελεστούν λιγότερες στοιχειώδεις λειτουργίες. Από την άλλη πλευρά, η ομοιότητα των γράφων και των αλγορίθμων στις δύο προσεγγίσεις διαταράσσεται, όταν ενσωματώνονται στα μοντέλα οι ρεαλιστικές προδιαγραφές που περιγράφονται στο επόμενο κεφάλαιο, οι οποίες έχουν επίσης επιπτώσεις και στον χρόνο τρεξίματος των αλγορίθμων: ο χρονικά-εξαρτώμενος αλγόριθμος δεν είναι πια τόσο γρηγορότερος από τον χρονικά-επεκτεινόμενο αλγόριθμο.

3.3. Βασικές Τεχνικές Επιτάχυνσης

Τώρα που έχουμε δει πώς μπορεί να μοντελοποιηθεί το βασικό πρόβλημα πληροφοριών χρονοδιαγράμματος, επικεντρώνουμε την προσοχή μας στο πώς μπορεί να βελτιωθεί ο χρόνος τρεξίματος των αλγορίθμων χρησιμοποιώντας βασικές τεχνικές επιτάχυνσης. Να σημειωθεί ότι η έννοια της *τεχνικής επιτάχυνσης* [*speedup technique*] αναφέρεται πάντοτε σε μια στρατηγική για τη βελτίωση του χρόνου τρεξίματος ενός αλγορίθμου συντομότερης διαδρομής που εγγυάται πάντοτε ότι η λύση είναι η βέλτιστη. Στις παραγράφους που ακολουθούν παρουσιάζουμε πώς εφαρμόζεται μια ευρέως γνωστή τεχνική επιτάχυνσης για τους αλγορίθμους συντομότερης διαδρομής που ονομάζεται *στόχο-κατευθυνόμενη αναζήτηση* [*goal-directed search*] και στα δύο μοντέλα.

3.3.1. Χρονικά-Επεκτεινόμενο Μοντέλο

Στόχο-κατευθυνόμενη Αναζήτηση

Μια τεχνική επιτάχυνσης που χρησιμοποιείται συχνά είναι η στόχο-κατευθυνόμενη αναζήτηση, που επίσης ονομάζεται αλγόριθμος A*, που εισάχθηκε αρχικά από τους Hart, Nilsson και Raphael (Hart, Nilsson and Raphael 1968) στο πεδίο της τεχνητής νοημοσύνης. Η ιδέα πίσω από αυτήν τη μέθοδο είναι να κατευθύνουμε την αναζήτηση που διεξάγεται από τον αλγόριθμο του Dijkstra προς τον προορισμό με τη βοήθεια ενός ευρετικού (το οποίο όμως δεν επηρεάζει τη βελτιστότητα του αποτελέσματος). Το μήκος κάθε ακμής τροποποιείται με τέτοιο τρόπο, ώστε εάν η ακμή δείχνει προς τον προορισμό το μήκος της γίνεται μικρότερο, ενώ εάν η ακμή δείχνει προς την αντίθετη κατεύθυνση από αυτήν του κόμβου προορισμού, τότε το μήκος της γίνεται μεγαλύτερο.

Στη γενική περίπτωση του αλγορίθμου A*, κάθε φορά που εξετάζεται ένας κόμβος (όπως ακριβώς και στον αλγόριθμο του Dijkstra), η ετικέτα απόστασης του υπολογίζεται από τη συνάρτηση:

$$f(n) = g(n) + h(n)$$

Στη συνάρτηση αυτή, το g(n) είναι η τρέχουσα απόσταση από τον κόμβο εκκίνησης και ταυτίζεται με την απόσταση που υπολογίζει ο αλγόριθμος του Dijkstra. Αντί όμως να μείνουμε με αυτήν και μόνο την τιμή, προσπαθούμε να εκμεταλλευτούμε και κάποια άλλη πληροφορία που ίσως να έχουμε διαθέσιμη, ώστε να οδηγηθούμε πιο γρήγορα στο αποτέλεσμά μας. Αυτή η πληροφορία μας δίνεται μέσω μιας συνάρτησης δυναμικού h(n), η οποία εξαρτάται από την εκάστοτε εφαρμογή.

Με άλλα λόγια η συνάρτηση δυναμικού h(n) είναι το ευρετικό με το οποίο επηρεάζουμε τις "εικονικές" αποστάσεις μεταξύ των κόμβων (δηλαδή τις αποστάσεις που χρησιμοποιεί ο αλγόριθμος εσωτερικά για να οδηγηθεί στο αποτέλεσμα) και όχι τις πραγματικές αποστάσεις (δηλαδή αυτές που θα παρουσιαστούν ως αποτέλεσμα στον χρήστη). Έτσι μπορούμε να πούμε πως ο αλγόριθμος του Dijkstra είναι υποπερίπτωση του A* στην οποία δεν έχουμε ευρετικά, δηλαδή h(n) = 0.

Αν το ευρετικό h ικανοποιεί την επιπρόσθετη προϋπόθεση $h(u) \leq d(u, v) + h(v)$ για κάθε ακμή (u, v) του γράφου, τότε το h λέγεται μονότονο ή συνεπές. Σε αυτή την υποπερίπτωση, στην

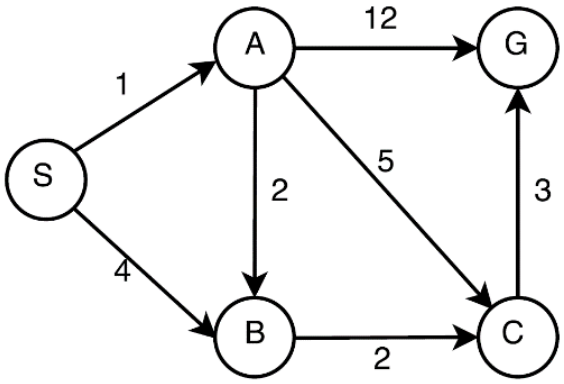
οποία ανήκει και η εφαρμογή μας, ο αλγόριθμος A* ισοδυναμεί με τον αλγόριθμο του Dijkstra με "μειωμένα κόστη" και μπορούμε να τον περιγράψουμε και ως εξής: για μια ακμή (u, v) με μήκος l(u, v), το νέο της μήκος l'(u, v) γίνεται l'(u, v)=l(u, v)-p(u)+p(v), όπου p[·] είναι μια συνάρτηση δυναμικού που σχετίζεται με τους κόμβους του γράφου. Μια συνάρτηση δυναμικού p[·] ονομάζεται έγκυρη αν το l'(u, v) είναι μη-αρνητικός. Η βελτιστότητα της στόχο-κατευθυνόμενης αναζήτησης με έγκυρες δυναμικές συναρτήσεις προκύπτει από το παρακάτω λήμμα.

Λήμμα 3.1 Έστω p[·] ένα έγκυρο δυναμικό. Μια s-t-διαδρομή P είναι μια συντομότερη διαδρομή σε σχέση με τα μήκη ακμών l'(u, v) αν και μόνο αν η P είναι μια συντομότερη διαδρομή στον αρχικό γράφο με μήκη ακμών l(u, v). Ο αλγόριθμος του Dijkstra εφαρμοσμένος στον γράφο με μήκη ακμών l'(u, v) μπορεί να χρησιμοποιηθεί για τον υπολογισμό του P.

Συνήθως, χρησιμοποιούνται χαμηλότερα φράγματα για τον σταθμό προορισμού με σκοπό να ληφθούν έγκυρα δυναμικά. Στην περίπτωσή μας, εκμεταλλευόμαστε τις γεωγραφικές πληροφορίες σχετικά με τους σταθμούς, που είναι διαθέσιμες στα δεδομένα μας. Χρησιμοποιώντας τις συντεταγμένες των σταθμών ορίζουμε ως D(u) την Ευκλείδεια απόσταση από το σταθμού u έως το σταθμό προορισμού B. Θεωρούμε επίσης και τη μέγιστη ταχύτητα του χρονοδιαγράμματος v_{max} ως εξής: Η ταχύτητα μιας στοιχειώδους σύνδεσης ορίζεται ως η Ευκλείδεια απόσταση μεταξύ των δύο σταθμών προς τη χρονική διάρκεια. Η v_{max} είναι η μέγιστη μεταξύ όλων αυτών τον ταχυτήτων. Η συνάρτηση δυναμικού στο χρονικά-επεκτεινόμενο μοντέλο ορίζεται ως $p[u]= D(u)/v_{max}$. Αυτή η προσαρμογή κλίμακας των Ευκλείδειων αποστάσεων εγγυάται εξ 'ορισμού έγκυρα (δηλαδή, μη αρνητικά) δυναμικά.

Παράδειγμα Στόχο-κατευθυνόμενης Αναζήτησης

Για να κατανοήσουμε καλύτερα τη στόχο-κατευθυνόμενη αναζήτηση, αλλά και τη συγγένειά της με τον αλγόριθμο του Dijkstra, την εφαρμόζουμε αναλυτικά στο παρακάτω παράδειγμα.



node	h
S	7
A	6
B	2
C	1
G	0

Ας υποθέσουμε ότι αναζητούμε τη βέλτιστη διαδρομή από τον κόμβο S στον G στον παραπάνω γράφο. Ας υποθέσουμε επίσης πως γνωρίζουμε εκ των προτέρων τα δυναμικά h για κάθε κόμβο για την περίπτωσή μας. Τα δυναμικά εξαρτώνται από την εκάστοτε αναζήτηση, αφού

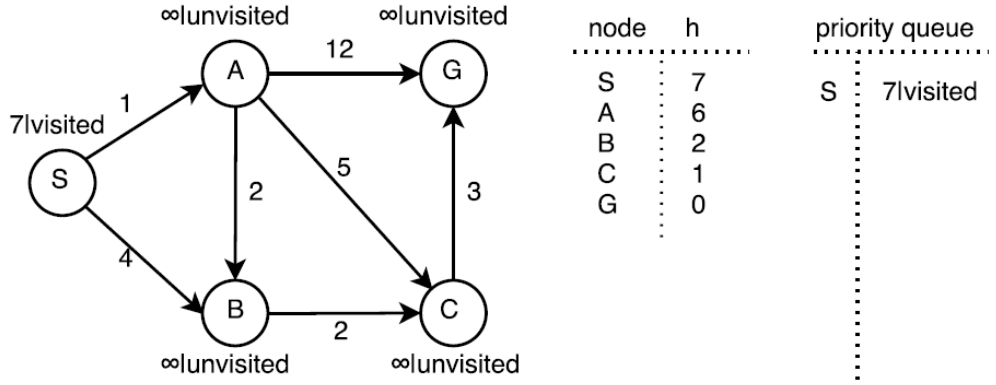
εξαρτώνται από τον εκάστοτε κόμβο προορισμού. Μιας όμως και δεν έχουμε την αναλυτική μορφή της συνάρτησης δυναμικού, ώστε να μπορούμε να υπολογίζουμε τα δυναμικά του κάθε κόμβου on-the-fly, υποθέτουμε ότι έχουν προϋπολογιστεί για τη συγκεκριμένη αναζήτηση. Βλέπουμε μάλιστα και την αναμενόμενη συμπεριφορά που παρουσιάζουν, και μερικά σημαντικά χαρακτηρίστηκα:

- Καθώς πλησιάζουν οι κόμβοι στον κόμβο τερματισμού, τα δυναμικά μειώνονται
- Το δυναμικό του τερματισμού είναι 0
- Είναι μη αρνητικά

Αυτά τα χαρακτηριστικά πρέπει να καλύπτονται εξ ορισμού από τις συναρτήσεις δυναμικών που χρησιμοποιούνται, και τα οποία καλύπτονται επίσης και από την συνάρτηση δυναμικού που ορίσαμε στην προηγούμενη παράγραφο.

Μια άλλη σημαντική διαφορά με τον κλασικό αλγόριθμο του Dijkstra είναι πως δεν γίνεται πια να χρησιμοποιούμε ετικέτα κατάστασης "finished". Αυτό συμβαίνει επειδή τώρα τα βάρη ακμών τροποποιούνται απρόβλεπτα (λόγω των ευρετικών) κατά τη διάρκεια του αλγορίθμου. Ο αλγόριθμος δεν μπορεί να είναι πια βέβαιος, πως ο κόμβος v που θα βγει από την ουρά προτεραιότητας στην αρχή κάθε επανάληψης θα είναι κόμβος μιας διαδρομής …(u, v) η οποία έχει χαλαρωθεί πλήρως. Μπορεί δηλαδή να υπάρξει μια άλλη διαδρομή …(u, v) που να χαλαρώσει και άλλο τη συγκεκριμένη ακμή. Η ιδιαιτερότητα αυτή μας αναγκάζει να τροποποιήσουμε λίγο την ουρά προτεραιότητας ώστε να αποθηκεύουμε ολόκληρη τη διαδρομή, αντί μόνο του κόμβου κατάληξης.

Επίσης δεν μπορούμε πια να τερματίσουμε τον αλγόριθμο μόλις ακουμπήσουμε τον σταθμό προορισμού. Ο αλγόριθμος τώρα τερματίζεται μόλις μια διαδρομή η οποία φτάνει στον σταθμό προορισμού πάει να βγει (ως πρώτη) από την ουρά προτεραιότητας. Όλες αυτές οι διαφοροποιήσεις θα φανούν καλύτερα και στο παράδειγμα που ακολουθεί.

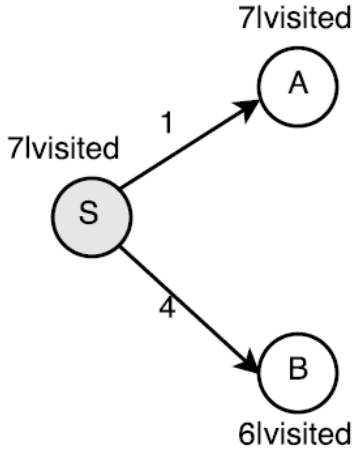


Κατά την αρχικοποίηση του αλγορίθμου έχουμε:

- η ετικέτα του κόμβου εκκίνησης S προκύπτει σύμφωνα με τη συνάρτηση f(n) = g(n) + h(n) ίση με 7,
- μπαίνει ο S στην ουρά προτεραιότητας
- θέτονται οι τιμές των υπολοίπων ετικετών ίσες με ∞

Στις εικόνες που ακολουθούν παραθέτουμε την οπτικοποίηση των βημάτων του αλγορίθμου.

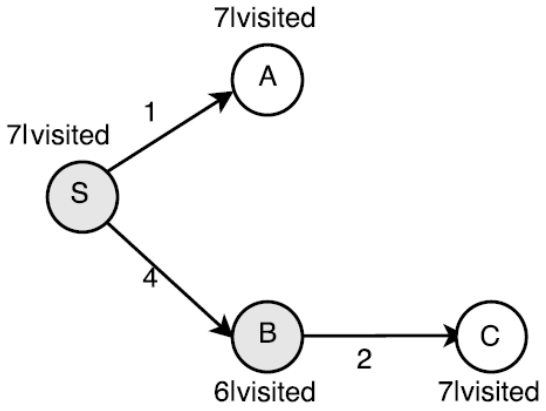
1η Επανάληψη



node	h
S	7
A	6
B	2
C	1
G	0

priority queue	
SB	6Ivisited
SA	7Ivisited

2η Επανάληψη

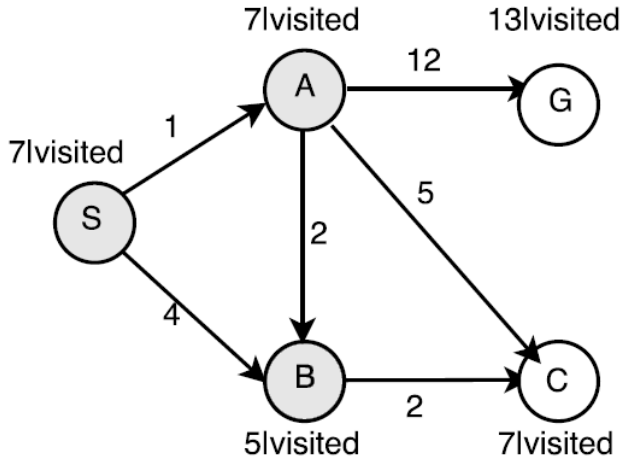


node	h
S	7
A	6
B	2
C	1
G	0

priority queue	
SA	7Ivisited
SBC	7Ivisited

Σε περίπτωση ισοβαθμίας η προτεραιότητα εξαρτάται από όποιο κριτήριο που έχουμε θέσει. Στην περίπτωσή μας θεωρούμε αλφαβητική ανατροπή ισοπαλίας. Άρα: SA > SBC

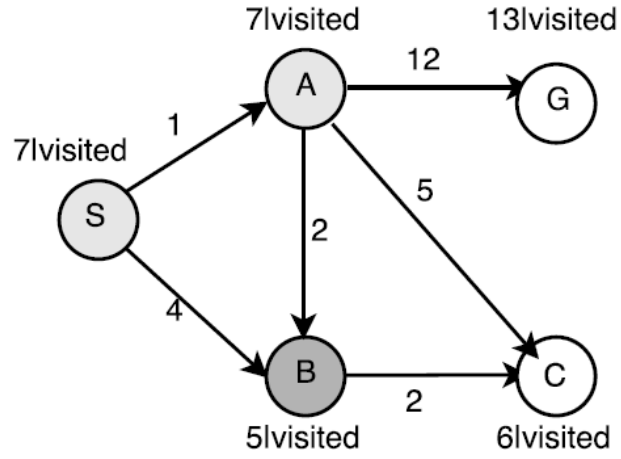
3η Επανάληψη



node	h
S	7
A	6
B	2
C	1
G	0

priority queue	
SAB	5|visited
SAC	7|visited
SBC	7|visited
SAG	13|visited

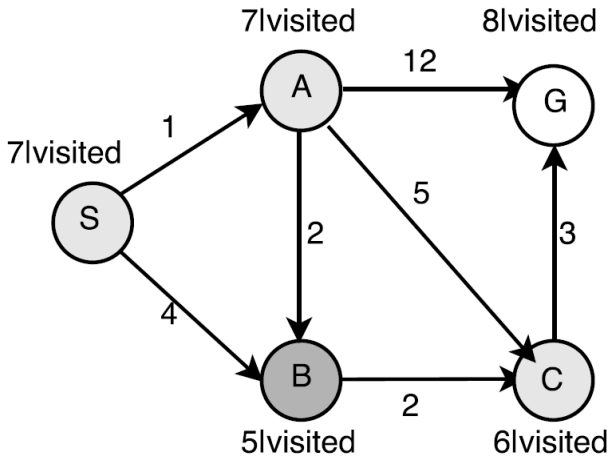
4η Επανάληψη



node	h
S	7
A	6
B	2
C	1
G	0

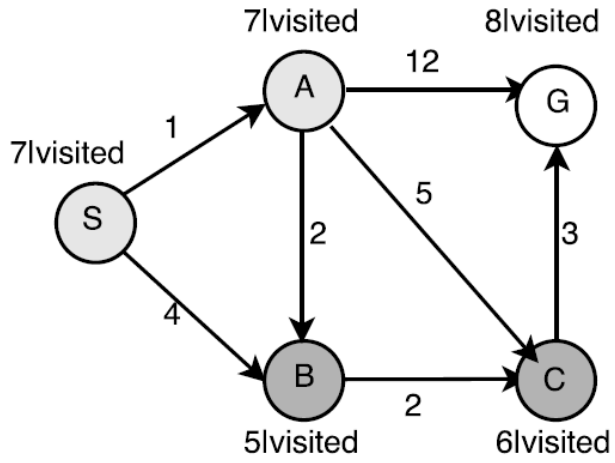
priority queue	
SABC	6|visited
SAC	7|visited
SBC	7|visited
SAG	13|visited

5η Επανάληψη

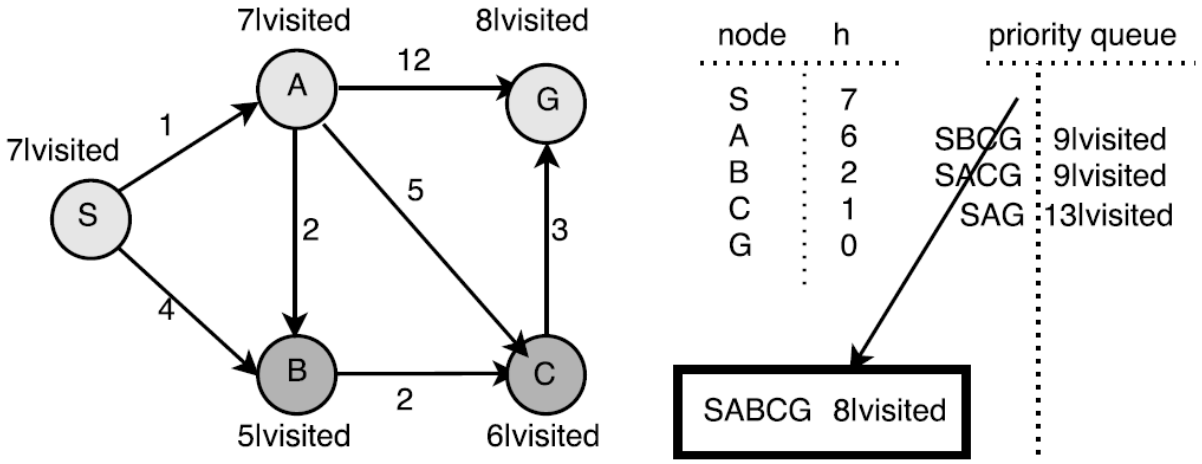


node	h	priority queue	
S	7	SBC	7|visited
A	6	SABCG	8|visited
B	2	SACG	9|visited
C	1	SAG	13|visited
G	0		

6η Επανάληψη



node	h	priority queue	
S	7	SABCG	8|visited
A	6	SBCG	9|visited
B	2	SACG	9|visited
C	1	SAG	13|visited
G	0		

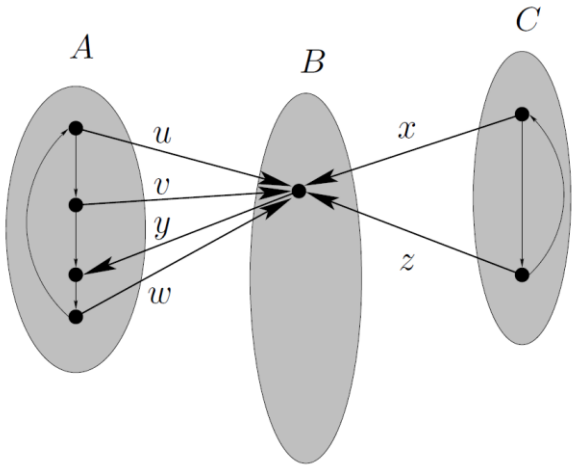


Παραλείποντας Κόμβους

Υπάρχει ένας απλός τρόπος για να μειωθεί το μέγεθος των γράφων στο χρονικά-επεκτεινόμενο μοντέλο, με βάση το γεγονός ότι υπάρχουν πάρα πολλοί κόμβοι με out-degree ίσο με ένα (δηλαδή έχουν μόνο μια εξερχόμενη ακμή). Κάθε διαδρομή μέσω ενός τέτοιου κόμβου πρέπει να συνεχίσει προς τα μπροστά στη μοναδική εξερχόμενη ακμή. Έτσι, μπορούμε με ασφάλεια να διαγράψουμε τέτοιους κόμβους από τον γράφο και να ανακατευθύνουμε τις εισερχόμενες ακμές στην αρχή της μοναδικής εξερχόμενης ακμής. Το βάρος μιας ανακατευθυνόμενης ακμής είναι το άθροισμα των εισερχόμενων ακμών συν το βάρος της εξερχόμενης ακμής.

Στον χρονικά-επεκτεινόμενο γράφο, παραλείπουμε κόμβους-άφιξης, εκτός από εκείνους που ανήκουν στον σταθμό προορισμού. Οι κόμβοι-άφιξης στον σταθμό προορισμού είναι σημαντικοί, καθώς ο αλγόριθμος πρέπει να τερματιστεί μόλις επεξεργαστεί ένας τέτοιος κόμβος από τον κύριο βρόχο του αλγορίθμου του Dijkstra. Δεδομένου ότι ο προορισμός γίνεται γνωστός μόνο όταν δοθεί ένα επερώτημα, οι κόμβοι αφίξεως στον σταθμό προορισμού εισάγονται δυναμικά κατά ζήτηση κατά τη διάρκεια του αλγορίθμου.

Η τεχνική αυτή παράγει έναν γράφο περίπου ίσο με το μισό του αρχικού μεγέθους, δεδομένου ότι όλοι οι κόμβοι-αφίξεως έχουν out-degree ίσο με ένα. Το Σχήμα 3.3 απεικονίζει αυτή την κατασκευή.



Σχήμα 3.3 Το δείγμα χρονικά-επεκτεινόμενου γράφου του Σχήματος 3.1, όπου όμως έχουν παραλειφθεί οι περιττοί κόμβοι. Ο σταθμός Α υποτίθεται ότι είναι ο σταθμός προορισμού, δηλαδή στον σταθμό Α κανένας από τους κόμβους δεν παραλείφθηκε.

3.3.2. Χρονικά-Εξαρτώμενο Μοντέλο

Στόχο-Κατευθυνόμενη Αναζήτηση

Το ευρετικό της στόχο-κατευθυνόμενης αναζήτησης που περιεγράφηκε στην προηγούμενη παράγραφο μπορεί επίσης να εφαρμοστεί και στο χρονικά-εξαρτώμενο μοντέλο. Ωστόσο, τα πειράματα με ακριβώς αυτή την τεχνική δεν έδειξαν καμία βελτίωση του χρόνου τρεξίματος στο χρονικά-εξαρτώμενο μοντέλο. Ως εκ τούτου, αναπτύχθηκε μια παραλλαγή της μεθόδου στόχο-κατευθυνόμενης αναζήτησης. Η παραλλαγή αυτή χρησιμοποιεί:

- μια διαφορετική συνάρτηση δυναμικού, η οποία εξαρτάται από τον σταθμό προορισμού
- της Μανχάταν, εκτός από τις Ευκλείδειες, αποστάσεις μεταξύ σταθμών
- ακέραια δυναμικά για να αποφευχθούν οι δαπανηρές διαδικασίες κινητής υποδιαστολής

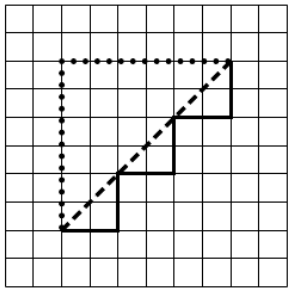
Η Ευκλείδεια απόσταση μεταξύ δύο σημείων (x_1, y_1) και (x_2, y_2) στο επίπεδο ορίζεται κατά τα γνωστά ως:

$$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$$

Η απόσταση Μανχάταν είναι η απόσταση των προβολών των σημείων αυτών στους άξονες. Ορίζεται ως:

$$d_M = |x_1 - x_2| + |y_1 - y_2|$$

και πήρε το όνομά της από την πραγματική απόσταση που διένυαν τα ταξί στο νησί του Μανχάταν λόγο της αυστηρής πολεοδομίας.



Όπως και στην Παράγραφο 3.3.1, για μια ακμή $e = (u, v) \in E$ με χρονικά-εξαρτώμενο μήκος $l_e(t)$, η νέα συνάρτηση μήκους $l'_e(t)$ ορίζεται ως $l'_e(t) = l_e(t) - p[u] + p[v]$, όπου $p[\cdot]$ είναι η συνάρτηση δυναμικού. Έστω Β ο σταθμός προορισμού. Τότε, ορίζεται:

$$p[u] = D(u, B)\lambda_B, \qquad u \in V, \lambda_B \geq 0$$

όπου D(u, B) είναι η Ευκλείδεια απόσταση ή η απόσταση Μανχάταν μεταξύ του σταθμού του κόμβου u και του σταθμού προορισμού B. Η παράμετρος λ_B ονομάζεται παράγοντας κλίμακας και είναι ένας μη αρνητικός αριθμός (ο οποίος μπορεί να διαφέρει για κάθε σταθμό προορισμού B) που χρησιμοποιείται για να προσαρμόσει την κλίμακα των αποστάσεων, έτσι ώστε το $l'_e(t)$ να είναι μη-αρνητικό. Ο συντελεστής κλίμακας ορίζεται ως

$$\lambda_B = \min_{(u,v)\in E, D(u,B)-D(v,B)>0} \frac{\min_t l_{(u,v)}(t)}{D(u, B) - D(v, B)}$$

Να σημειωθεί ότι το λ_B αντιστοιχεί στο αντίστροφο της μέγιστης ταχύτητας που αναφέρθηκε στην Παράγραφο 3.3.1· ωστόσο, στην Παράγραφο 3.3.1 η μέγιστη ταχύτητα είναι η ίδια για όλους τους προορισμούς, ενώ το λ_B εξαρτάται από τον σταθμό προορισμού. Η βασική ιδέα πίσω από την τροποποίηση αυτή είναι ότι η μέγιστη ταχύτητα υπολογίζεται μόνο από όλες τις ακμές που δείχνουν γεωγραφικά προς τον σταθμό προορισμού. Έτσι, για τις ακμές που δείχνουν προς τον προορισμό, η προϋπόθεση $l'_e \geq 0$ εκπληρώνεται εξαιτίας της προσαρμογής κλίμακας. Για τις υπόλοιπες ακμές, που δείχνουν προς την αντίθετη κατεύθυνση από αυτήν του προορισμού, το μήκος γίνεται ούτως ή άλλως μεγαλύτερο και επίσης πληρείται η προϋπόθεση: $l'_e \geq l_e \geq 0$. Το επόμενο λήμμα δείχνει πιο επίσημα ότι τα δυναμικά p[·], όπως ορίζονται παραπάνω είναι έγκυρα.

Λήμμα 3.2 Η συνάρτηση p[·] είναι μια συνάρτηση έγκυρων δυναμικών για τη στόχο-κατευθυνόμενη αναζήτηση, δηλαδή, το l'_e είναι μη αρνητικό για κάθε ακμή e.

Ακόμα και αν όλα τα βάρη ακμών είναι ακέραιοι, η χρήση των Ευκλείδειων ή των Μανχάταν αποστάσεων ως δυναμικά μας αναγκάζει να χρησιμοποιήσουμε αριθμούς κινητής υποδιαστολής στην ουρά προτεραιότητας, και αυτό μπορεί να οδηγήσει σε πρόσθετη επιβάρυνση του χρόνου τρεξίματος. Για να αποφευχθεί αυτό, μπορούμε να μετατρέψουμε τα δυναμικά κινητής υποδιαστολής σε ακέραια χωρίς να επηρεάζεται η εγκυρότητά τους όπως δείχνει το παρακάτω λήμμα.

Λήμμα 3.3 Αν για τους μη αρνητικούς αριθμούς $w \in \mathbb{N}$, $\alpha, b \in \mathbb{R}^+$ ισχύει ότι $w - \alpha + \beta \geq 0$, τότε ισχύει και ότι $w - \lfloor\alpha\rfloor + \lfloor\beta\rfloor \geq 0$.

Το να ληφθούν υπόψη τα ακέραια τμήματα των δυναμικών κινητής υποδιαστολής είναι αρκετά ωφέλιμο σε ορισμένες περιπτώσεις, όπως δείχνουν τα πειράματα στις επόμενες ενότητες.

Αποφυγή Δυαδικής Αναζήτησης

Παραθέτουμε επίσης και το ευρετικό το οποίο αποφεύγει τη δυαδική αναζήτηση (Brodal and Jacob 2001). Η ιδέα είναι ως εξής. Έστω k ο out-degree ενός κόμβου v στον χρονικά-εξαρτώμενο γράφο και έστω (v, u_1), ..., (v, u_k) οι εξερχόμενες ακμές του. Κατασκευάζουμε έναν πίνακα D_v ταξινομώντας όλα τα γεγονότα των εξερχόμενων ακμών του v με βάση τους χρόνους αναχώρησης τους. Τοποθετούμε τα πρώτα k τέτοια γεγονότα στις πρώτες k θέσεις (πρωτεύον τομέας) του D_v, αφήνουμε τις επόμενες k θέσεις κενές (δευτερεύον τομέας), τοποθετούμε τα επόμενα k γεγονότα στις επόμενες k θέσεις του D_v, και ούτω καθεξής. Έστω t_0 το τελευταίο

γεγονός στον D_v πριν από κάποιο δευτερεύον τομέα. Για κάθε (v, u_i), $1 \leq i \leq k$, βρίσκουμε το πρώτο γεγονός με χρόνο αναχώρησης $t \geq t_0$ και το τοποθετούμε στην i-στη θέση του επόμενου δευτερογενούς τομέα. Για μια ακμή (w, v), έστω t_1 ο χρόνος άφιξης ενός πρωτεύοντος γεγονότος $P^w \in D_w$ (γεγονός το οποίο ανήκει σε κάποιον πρωτογενή τομέα του D_w). Δημιουργούμε έναν δείκτη από το P^w στο αμέσως επόμενο πρωτεύον γεγονός $P^v \in D_v$ με χρονοσφραγίδα $t_2 \geq t_1$. Η παραπάνω κατασκευή αποφεύγει τη δυαδική αναζήτηση, διότι όταν αφαιρείται ένας κόμβος v από την ουρά προτεραιότητας, απλά ακολουθούμε τον δείκτη του γεγονότος P^w που προκάλεσε την αφαίρεση του v από την ουρά προτεραιότητας, και ο οποίος μας οδηγεί σε μια πρωτογενή εγγραφή $P^v \in D_v$. Ως εκ τούτου, για να βρούμε το επόμενο εξερχόμενο γεγονός από τον κόμβο v αρκεί να σαρώσουμε το υπόλοιπο του πρωτεύοντος τομέα που περιέχει το P^v και τον επόμενό δευτερεύον του τομέα.

3.4. Πειραματική Σύγκριση των Μοντέλων

Σε αυτό το κεφάλαιο παρουσιάζουμε τις πειραματικές μελέτες από τη βιβλιογραφία (Schulz 2005) που συγκρίνουν στην πράξη την απόδοση της χρονικά-επεκτεινόμενης και της χρονικά-εξαρτώμενης προσέγγισης καθώς και των τεχνικών επιτάχυνσης που παρουσιάστηκαν στην προηγούμενη ενότητα.

Δεδομένων δύο διαφορετικών υλοποιήσεων και ενός χρονοδιαγράμματος, ορίζεται η *σχετική απόδοση* [*relative performance*] ή *επιτάχυνση* [*speedup*] με βάση μία μετρηθείσα παράμετρο απόδοσης ως ο λόγος της τιμής που λαμβάνεται από την πρώτη εκτέλεση προς την τιμή που λαμβάνεται από τη δεύτερη. Όταν συγκρίνονται μία χρονικά-επεκτεινόμενη και μια χρονικά-εξαρτώμενη υλοποίηση, διαιρείται πάντα η χρονικά-επεκταμένη τιμή από τη χρονικά-εξαρτώμενη τιμή, δηλαδή, θεωρείται ότι η επιτάχυνση επιτεύχθηκε όταν χρησιμοποιείται η χρονικά-εξαρτώμενη αντί της χρονικά-επεκτεινόμενης προσέγγισης.

Για τα πειράματα που περιγράφονται σε αυτό το κεφάλαιο, ο κώδικας γράφτηκε σε C++ και μεταγλωττίστηκε με τον GNU C++ compiler έκδοσης 3.2. Τα πειράματα διεξάχθηκαν σε ένα PC με επεξεργαστή AMD Athlon XP 1500+ στα 1.3 GHz και μνήμη 512MB με λειτουργικό σύστημα Linux (έκδοσης kernel 2.4.19). Η υλοποίηση του χρονικά-εξαρτώμενου μοντέλου για το πρόβλημα νωρίτερης άφιξης χρησιμοποιεί τη δομή δεδομένων γράφου LEDA έκδοσης 4.4 της εταιρίας Algorithmic Solutions Software GmbH.

3.4.1. Δεδομένα

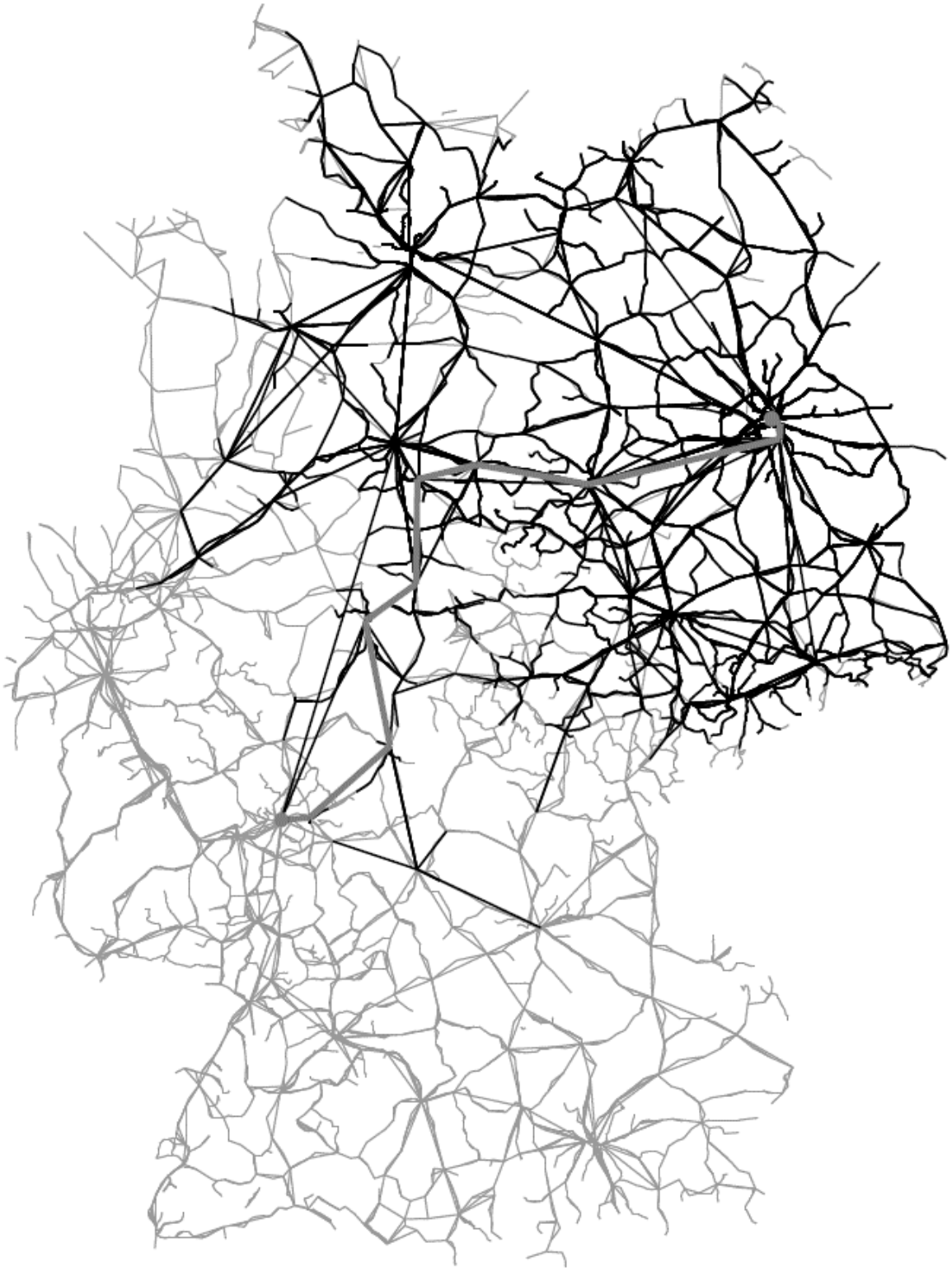
Τα δεδομένα που χρησιμοποιήθηκαν στις πειραματικές μελέτες που παρουσιάζουμε παρακάτω είναι δεδομένα πραγματικού κόσμου από τους γερμανικούς και γαλλικούς σιδηροδρόμους. Πιο συγκεκριμένα, έχουν χρησιμοποιηθεί τα ακόλουθα πέντε χρονοδιαγράμματα (κατά σειρά μεγέθους). Το πρώτο χρονοδιάγραμμα περιέχει την κυκλοφορία του γαλλικού σιδηροδρόμου μεγάλων αποστάσεων (france) από τη χειμερινή περίοδο 1996-1997. Τα υπόλοιπα τέσσερα είναι γερμανικά χρονοδιαγράμματα από τη χειμερινή περίοδο 2000/01· ένα χρονοδιάγραμμα αναπαριστά τη σιδηροδρομική κυκλοφορία μεγάλων αποστάσεων στη Γερμανία (GER-longdist), δύο περιέχουν την τοπική κυκλοφορία στο Βερολίνο/Βρανδεμβούργο (GER-

Local1) και στην περιοχή Rhein/Main (GER-Local2), και το τελευταίο είναι η ένωση των τριών αυτών χρονοδιαγραμμάτων (GER-all). Θα θέλαμε να σημειώσουμε ότι το HAFAS, το σύστημα πληροφοριών χρονοδιαγράμματος που χρησιμοποιείται από τη γερμανική εταιρεία σιδηροδρόμων Deutsche Bahn, βασίζεται σε δεδομένα ίδιας μορφής. Ο Πίνακας 3.1 δείχνει τα χαρακτηριστικά των γράφων που χρησιμοποιήθηκαν σε αυτά τα μοντέλα για τα χρονοδιαγράμματα που αναφέρονται παραπάνω.

	Time-Expanded		**Time-Dependent**			
Timetable	Nodes	Edges	Nodes	Edges	Elem. conn. per node	Elem. conn. per edge
`france`	166085	332170	4578	14791	36	11
`ger-longdist`	480173	960346	6817	18812	70	26
`ger-local1`	691541	1383082	13460	37315	51	19
`ger-local2`	1124824	2249648	13073	36621	86	31
`ger-all`	2295930	4591860	32253	92507	71	25

Πίνακας 3.1 Παράμετροι των εξεταζόμενων γράφων για καθένα από τα χρονοδιαγράμματα. Οι δύο στήλες στα αριστερά δείχνουν το μέγεθος του γράφου που χρησιμοποιήθηκε στο χρονικά-επεκτεινόμενο μοντέλο. Ο αριθμός των κόμβων ισούται με τον συνολικό αριθμό των στοιχειωδών συνδέσεων του χρονοδιαγράμματος. Οι υπόλοιπες στήλες δείχνουν τις παραμέτρους του γράφου που χρησιμοποιήθηκε στο χρονικά-εξαρτώμενο μοντέλο.

Επερωτήματα πραγματικού κόσμου ήταν διαθέσιμα μόνο για τα χρονοδιαγράμματα ger-longdist και ger-all: πάρθηκαν 50.000 επερωτήματα από ένα στιγμιότυπο του κεντρικού HAFAS server στη Γερμανία, το οποίο αποτελείται από περισσότερα από μισό εκατομμύριο επερωτήματα πραγματικού κόσμου. Επιπρόσθετα δημιουργήθηκαν τυχαία ερωτήματα για κάθε χρονοδιάγραμμα. Κάθε σύνολο επερωτημάτων αποτελείται από 50.000 επερωτήματα της μορφής: σταθμός αναχώρησης, σταθμός προορισμού και χρόνος νωρίτερης αναχώρησης.



Εικόνα 3.1 Μια οπτικοποίηση των δεδομένων χρονοδιαγράμματος ger-longdist, ο χώρος αναζήτησης που αντιστοιχεί σε ένα δείγμα επερωτήματος από το Βερολίνο προς τη Φρανκφούρτη, και η βέλτιστη σιδηροδρομική σύνδεση.

3.4.2. Περιβάλλον Υλοποίησης και Παράμετροι Απόδοσης

Για το χρονικά-επεκτεινόμενο μοντέλο η υλοποίηση περιλαμβάνει την τεχνική βελτιστοποίησης που παραλείπει τους κόμβους-άφιξης. Για το χρονικά-εξαρτώμενο μοντέλο, έχει υλοποιηθεί και η απλή εκδοχή που χρησιμοποιεί δυαδική αναζήτηση, αλλά και η τεχνική "αποφυγής δυαδικής αναζήτησης". Και για τα δύο μοντέλα χρησιμοποιήθηκε επίσης η στόχο-κατευθυνόμενη αναζήτηση. Έτσι, για το χρονικά-επεκτεινόμενο μοντέλο έχουμε δύο διαφορετικές εκτελέσεις (στόχο-κατευθυνόμενη αναζήτηση με ή χωρίς Ευκλείδειες αποστάσεις), ενώ για το χρονικά-εξαρτώμενο μοντέλο έχουμε αρκετές εκτελέσεις ανάλογα με τη χρήση: δυαδικής αναζήτησης ή της εκδοχής "αποφυγής δυαδικής αναζήτησης", το ευρετικό στόχο-κατευθυνόμενης αναζήτησης με Ευκλείδειες ή Μανχάταν αποστάσεις, και εάν χρησιμοποιούνται δυναμικά κινητής υποδιαστολής ή ακέραια.

Για κάθε πιθανό συνδυασμό χρονοδιαγράμματος και υλοποίησης διεξάχθηκε το αντίστοιχο σύνολο τυχαίων επερωτημάτων (για ger-longdist και ger-all πραγματοποιήθηκαν επιπλέον τα αντίστοιχα επερωτήματα πραγματικού κόσμου) και μετρήθηκαν οι ακόλουθες παράμετροι απόδοσης ως μέσες τιμές του συνόλου επερωτημάτων: του χρόνου CPU σε χιλιοστά του δευτερολέπτου, του αριθμού των κόμβων, του αριθμού των ακμών, και του αριθμού των στοιχειωδών συνδέσεων που αγγίζονται από τον αλγόριθμο. Για το χρονικά-επεκτεινόμενο μοντέλο, ο αριθμός των στοιχειωδών συνδέσεων που έχουν αγγιχθεί είναι ο αριθμός των σιδηροδρομικών-ακμών που έχουν αγγιχθεί από τον αλγόριθμο, ενώ για το χρονικά-εξαρτώμενο μοντέλο είναι ο συνολικός αριθμός των στοιχειωδών που έχουν χρησιμοποιηθεί για τον υπολογισμό των μηκών ακμών. Ακριβέστερα, όταν χρησιμοποιείται δυαδική αναζήτηση στο χρονικά-εξαρτώμενο μοντέλο, ο αριθμός των βημάτων που απαιτούνται για μια ακμή από τη δυαδική αναζήτηση είναι ο αριθμός των στοιχειωδών συνδέσεων που έχουν αγγιχθεί.

3.4.3. Αποτελέσματα και Συζήτηση

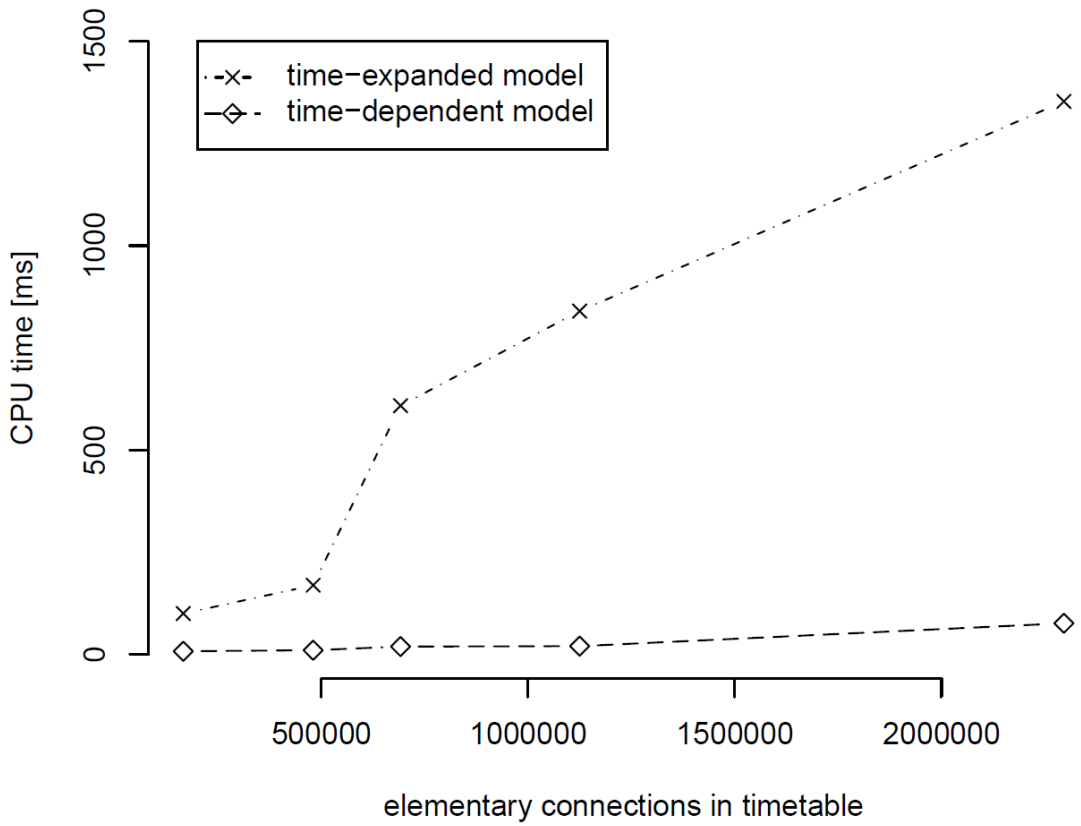
Τα πειραματικά αποτελέσματα παρουσιάζονται στα Σχήματα 3.4 και 3.5, και στους Πίνακες 3.2, 3.3 και 3.4. Τα αποτελέσματα δείχνουν με σαφήνεια ότι το χρονικά-εξαρτώμενο μοντέλο λύνει το απλοποιημένο πρόβλημα νωρίτερης άφιξης αρκετά γρηγορότερα από το χρονικά-επεκτεινόμενο μοντέλο, για κάθε σύνολο δεδομένων που μελετήθηκε. Όσον αφορά στον χρόνο CPU, η επιτάχυνση κυμαίνεται μεταξύ 12 (france) και 40 (GER-Local2), όταν χρησιμοποιούνται οι βασικές υλοποιήσεις (βλ. Σχήμα 3.4 και Πίνακα 3.2), και μεταξύ 17(france) και 57(GER-Local2) όταν η σύγκριση αφορά τις βέλτιστες υλοποιήσεις (συμπεριλαμβανομένων των ευρετικών) και στα δύο μοντέλα (βλ. Πίνακες 3.3 και 3.4).

Όσον αφορά στο χρονικά-εξαρτώμενο μοντέλο, παρατηρούμε ότι είναι καλύτερο να χρησιμοποιούμε την τεχνική "αποφυγής δυαδικής αναζήτησης" (βλ. Πίνακες 3.3 και 3.4). Σε σύγκριση με την εκτέλεση δυαδικής αναζήτησης η επιτάχυνση ήταν μεταξύ 1.39 (GER- Local1) και 1.86 (ger-all με επερωτήματα πραγματικού κόσμου). Η τεχνική στόχο-κατευθυνόμενης αναζήτησης μειώνει πάντα το χώρο αναζήτησης του αλγόριθμου του Dijkstra, δηλαδή, τον αριθμό των κόμβων και των ακμών που αγγίζονται. Ωστόσο, η μείωση αυτή απέδωσε μόνο σε λίγες περιπτώσεις, με την έννοια ότι δεν μπόρεσε επίσης να μειώσει και τον χρόνο CPU. Στις περισσότερες περιπτώσεις, ο χρόνος CPU αυξήθηκε εξαιτίας των επιπρόσθετων υπολογισμών που

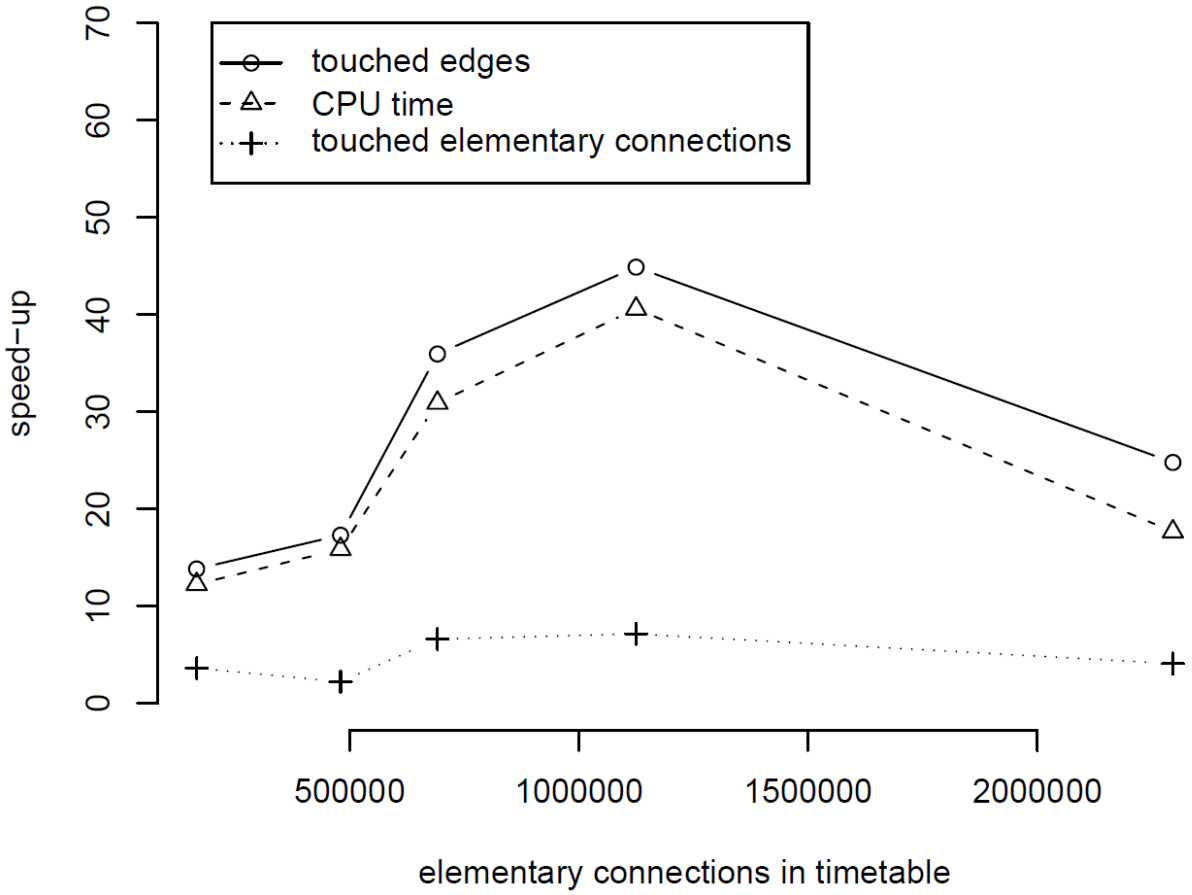
απαιτούνταν για τον υπολογισμό των μηκών ακμών, και αυτός είναι ο κύριος λόγος για τον οποίο αυτή η τεχνική παρουσιάζεται ως βραδύτερη στα αποτελέσματα. Ένας άλλος λόγος είναι ότι στα χρονοδιαγράμματα που χρησιμοποιούνται στα πειράματά αυτά, η μέγιστη ταχύτητα των τρένων ανάμεσα σε όλες τις στοιχειώδεις συνδέσεις είναι υψηλή. Μια υψηλή μέγιστη ταχύτητα αποδίδει μικρές συναρτήσεις δυναμικών, και κατά συνέπεια κακή απόδοση για την τεχνική στόχο-κατευθυνόμενης αναζήτησης.

	Timetable	Real	Time [ms]	El. Conn.	Nodes	Edges
Expanded	france		100.4	30824	33391	61649
	ger-longdist		169.6	44334	48094	88668
	ger-local1		608.7	176720	182717	353443
	ger-local2		840.1	226027	232511	452056
	ger-all		1352.8	326186	342917	652378
	ger-longdist	×	66.7	18891	20853	37783
	ger-all	×	392.1	96943	104369	193888
Dependent	france		8.2	8539	2269	4463
	ger-longdist		10.7	20066	3396	5129
	ger-local1		19.7	26792	6535	9835
	ger-local2		20.7	31698	6524	10075
	ger-all		76.6	79981	16145	26333
	ger-longdist	×	5.5	11173	1711	2682
	ger-all	×	37.3	40808	6926	11647

Πίνακας 3.2 Μέσος όρος χρόνων CPU για την επίλυση ενός επερωτήματος για το χρονικά-επεκτεινόμενο (άνω μέρος) και το χρονικά-εξαρτώμενο μοντέλο (κάτω μέρος). Οι κόμβοι-άφιξης παραλείπονται στο χρονικά-επεκτεινόμενο μοντέλο (βλ. Παράγραφο 3.4.1), και στο χρονικά-εξαρτώμενο μοντέλο χρησιμοποιήθηκε δυαδική αναζήτηση. Και στις δύο περιπτώσεις δεν εφαρμόστηκε στόχο-κατευθυνόμενη αναζήτηση. Η στήλη Real δείχνει αν χρησιμοποιήθηκαν επερωτήματα πραγματικού κόσμου ή τυχαία επερωτήματα.



Σχήμα 3.4 Απόδοση της βασικής υλοποίησης του χρονικά-επεκτεινόμενο και του χρονικά-εξαρτώμενου μοντέλου για το απλοποιημένο πρόβλημα νωρίτερης άφιξης (χωρίς στόχο-κατευθυνόμενη αναζήτηση, δυαδική αναζήτηση για το χρονικά-εξαρτώμενο μοντέλο) όσον αφορά στα πέντε χρονοδιαγράμματα και για τυχαία επερωτήματα. Κάθε σημείο αντιπροσωπεύει τον μέσο όρο μιας μετρούμενης παραμέτρου απόδοσης σε αυτά τα επερωτήματα. Στην τετμημένη φαίνεται το μέγεθος του χρονοδιαγράμματος ως αριθμός στοιχειωδών συνδέσεων, και η τεταγμένη αντιπροσωπεύει τον μέσο όρο του χρόνου CPU κατά την απάντηση ενός επερωτήματος στο χρονικά-επεκτεινόμενο μοντέλο (άνω καμπύλη) και στο χρονικά-εξαρτώμενο μοντέλο (κάτω καμπύλη), αντίστοιχα.



Σχήμα 3.5 Όπως στο Σχήμα 3.4, αλλά τώρα η τεταγμένη αντιπροσωπεύει την επιτάχυνση από τη χρήση του χρονικά-εξαρτώμενου μοντέλου αντί της χρήσης του χρονικά-επεκτεινόμενο μοντέλου, σε σχέση με τον αριθμό των ακμών που έχουν αγγιχθεί, τον χρόνο CPU, και τον αριθμό στοιχειωδών συνδέσεων που έχουν αγγιχθεί (από πάνω προς τα κάτω).

Time-Dependent Model, Binary Search

	Timetable	Real	Time [ms]	El. conn.	Nodes	Edges
Goal Eucl. int	france		9.4	7072	1593	3415
	ger-longdist		13.5	16597	2737	4217
	ger-local1		28.6	26008	6257	9434
	ger-local2		30.4	31196	6398	9895
	ger-all		100.3	74525	14568	24030
	ger-longdist	×	6.3	8349	1238	1991
	ger-all	×	43.1	33676	5551	9420
Goal Eucl. float	france		9.4	7062	1590	3410
	ger-longdist		13.6	16560	2730	4208
	ger-local1		28.9	25975	6249	9422
	ger-local2		30.7	31152	6389	9882
	ger-all		103.6	74394	14538	23983
	ger-longdist	×	6.4	8318	1233	1984
	ger-all	×	44.5	33565	5532	9388
Goal Manh. int	france		7.9	7225	1647	3511
	ger-longdist		11.2	16975	2807	4316
	ger-local1		23.2	26086	6284	9473
	ger-local2		24.7	31235	6407	9908
	ger-all		86.4	74822	14639	24138
	ger-longdist	×	5.3	8555	1272	2041
	ger-all	×	38.0	33994	5615	9524
Goal Manh. float	france		7.7	7214	1644	3505
	ger-longdist		11.1	16938	2800	4306
	ger-local1		23.1	26053	6276	9461
	ger-local2		24.6	31189	6398	9894
	ger-all		88.9	74689	14608	24091
	ger-longdist	×	5.2	8524	1267	2034
	ger-all	×	39.0	33880	5594	9491

Πίνακας 3.3 Σύγκριση των χρονικά-εξαρτώμενων εκτελέσεων που χρησιμοποιούν δυαδική αναζήτηση και τεσσάρων διαφορετικών εκδοχών στόχο-κατευθυνόμενης αναζήτησης: Ευκλείδεια απόσταση με ακέραια και κινητής υποδιαστολής δυναμικά, και απόσταση Μανχάταν με ακέραια και κινητής υποδιαστολής δυναμικά. Οι στήλες είναι όπως στον Πίνακα 3.2.

Time-Expanded Model

	Timetable	Real	Time [ms]	El. Conn.	Nodes	Edges
Goal Eucl. int	france		84.0	22259	24179	44517
	ger-longdist		175.0	34259	37453	68517
	ger-local1		684.3	170369	176243	340741
	ger-local2		953.0	219992	226386	439986
	ger-all		1392.6	285440	300788	570885
	ger-longdist	×	54.3	13384	14931	26768
	ger-all	×	341.9	74069	80229	148140

Time-Dependent Model, Avoid Binary Search

Plain Dijkstra	france		5.9	8942	2262	4386
	ger-longdist		7.5	9216	3396	5129
	ger-local1		14.2	18312	6541	9814
	ger-local2		14.6	18435	6524	10075
	ger-all		47.4	48520	16146	26333
	ger-longdist	×	3.8	4773	1711	2682
	ger-all	×	20.1	20993	6927	11648
Goal Eucl. int	france		6.6	6711	1614	3406
	ger-longdist		9.2	7553	2737	4217
	ger-local1		20.5	17656	6301	9499
	ger-local2		21.5	18088	6398	9895
	ger-all		63.0	44010	14568	24030
	ger-longdist	×	4.2	3527	1238	1991
	ger-all	×	23.7	16898	5552	9421
Goal Manh. int	france		5.1	6926	1669	3505
	ger-longdist		7.1	7733	2807	4316
	ger-local1		15.3	17621	6289	9480
	ger-local2		16.1	18113	6407	9908
	ger-all		50.5	44214	14639	24138
	ger-longdist	×	3.2	3618	1272	2041
	ger-all	×	19.1	17088	5615	9524

Πίνακας 3.4 Σύγκριση της στόχο-κατευθυνόμενης αναζήτησης στη χρονικά-επεκτεινόμενη περίπτωση (άνω μέρος) και της τεχνικής αποφυγής δυαδικής αναζήτησης στο χρονικά-εξαρτώμενο μοντέλο (κάτω μέρος). Στη χρονικά-εξαρτώμενη περίπτωση καταγράφονται δύο διαφορετικά μέτρα απόστασης για τη στόχο-κατευθυνόμενη αναζήτηση, η Ευκλείδεια απόσταση και η απόσταση Μανχάταν με ακέραια δυναμικά, που ήταν τα πιο γρήγορα. Οι στήλες είναι όπως στον Πίνακα 3.2.

4. Προς Ρεαλιστικές Πληροφορίες Χρονοδιαγράμματος

Το προηγούμενο κεφάλαιο παρουσίασε τις βασικές τεχνικές μοντελοποίησης που είναι υποδειγματικές για μια εκδοχή του προβλήματος νωρίτερης άφιξης, η οποία είναι υπερβολικά περιοριστική για εφαρμογή σε ένα πραγματικό σύστημα πληροφοριών χρονοδιαγράμματος. Στην πραγματικότητα, πρέπει να αντιμετωπιστούν αρκετά ζητήματα που αφορούν στον ρεαλισμό των προβλημάτων και των μοντέλων:

- Πρώτον, πρέπει να αντιμετωπίσουμε τους ρεαλιστικούς κανόνες μετεπιβίβασης, και να εγκαταλείψουμε την παραδοχή ότι οι μετεπιβιβάσεις εντός ενός σταθμού έχουν αμελητέα διάρκεια.
- Επιπλέον, στην πραγματικότητα τα τρένα λειτουργούν μόνο σε συγκεκριμένες ημέρες κυκλοφορίας , έτσι δεν μπορούμε να υποθέσουμε -όπως κάναμε μέχρι τώρα- ότι κάθε μέρα του χρονοδιαγράμματος είναι ίδια.
- Ένα άλλο ζήτημα είναι να λάβουμε υπόψη διαφορετικά κριτήρια βελτιστοποίησης, εκτός από τη νωρίτερη άφιξη: Στο πρόβλημα ελάχιστου αριθμού μεταβιβάσεων, ο στόχος είναι να βρεθεί μια σύνδεση που ελαχιστοποιεί τον αριθμό των μεταβιβάσεων, όταν εξετάζουμε μια διαδρομή από το A στο B. Στο πρόβλημα αργότερης αναχώρησης, ανάμεσα σε όλες τις βέλτιστες λύσεις για μια περίπτωση του προβλήματος νωρίτερης άφιξης, είναι επιθυμητή αυτή που αναχωρεί το αργότερο.
- Τέλος, ένα σύστημα πληροφοριών χρονοδιαγράμματος πρέπει επίσης να επιτρέπει συνδυασμούς διαφορετικών κριτηρίων.

Θα δείξουμε πώς μπορούν να επεκταθούν και οι δύο προσεγγίσεις για τη μοντελοποίηση πληροφοριών χρονοδιαγράμματος που παρουσιάστηκαν προηγουμένως ώστε να αντιμετωπίσουν τις ρεαλιστικές προδιαγραφές προβλημάτων που περιεγράφηκαν παραπάνω. Παρουσιάζουμε επίσης εκτεταμένα πειράματα από τη βιβλιογραφία συγκρίνοντας τις εκτεταμένες προσεγγίσεις.

4.1. Ρεαλιστικοί Κανόνες Μετεπιβίβασης

Σε αντίθεση με το προηγούμενο κεφάλαιο, όπου οι μετεπιβιβάσεις επιτρέπονταν πάντοτε, παρουσιάζουμε τώρα πώς μπορούν να συμπεριληφθούν μη μηδενικοί χρόνοι μετεπιβίβασης στα μοντέλα που παρουσιάστηκαν. Επίσης, εξετάζουμε πιο εξεζητημένους κανόνες μετεπιβίβασης. Αναφερόμαστε στο πρόβλημα νωρίτερης άφιξης με ρεαλιστικούς κανόνες μετεπιβιβάσεων ως το *ρεαλιστικό πρόβλημα νωρίτερης άφιξης* [*realistic earliest arrival problem*], ή το *ρεαλιστικό ΠΝΑ* [*realistic EAP*].

4.1.1. Προδιαγραφές

Σε έναν σταθμό $S \in B$ είναι δυνατή η μετεπιβίβαση από το ένα τρένο σε ένα άλλο μόνο αν ο χρόνος μεταξύ της άφιξης και της αναχώρησης στον σταθμό S είναι μεγαλύτερος από ή ίσος με έναν δεδομένο ελάχιστο χρόνο μετεπιβίβασης, συγκεκριμένο για κάθε σταθμό, που συμβολίζεται ως transfer(S). Για να συμπεριλάβουμε αυτόν τον κανόνα μετεπιβιβάσεων στις επίσημες προδιαγραφές μας, επεκτείνουμε τον ορισμό μιας συνεπούς σύνδεσης της Παραγράφου 3.1.2 με την ακόλουθη προϋπόθεση:

- $\mathrm{dep}_{i+1}(P) - \mathrm{arr}_i(P) \geq \begin{cases} 0 & \text{αν } Z(c_{i+1}) = Z(c_i) \\ \mathrm{transfer}(S_2(c_i)) & \text{αλλιώς} \end{cases}$

Μπορεί επίσης να υπάρχουν πιο λεπτομερείς κανόνες μετεπιβιβάσεων, για παράδειγμα, μπορεί ο χρόνος μετεπιβίβασης να είναι μικρότερος για τρένα που αναχωρούν από την ίδια πλατφόρμα. Οι χρόνοι μετεπιβιβάσεων μπορεί να είναι δοσμένοι για κάθε σιδηροδρομική διαδρομή (που αναφέρεται επίσης ως σιδηροδρομική γραμμή), αντί να προσδιορίζεται ένας χρόνος μετεπιβίβασης για κάθε έναν σταθμό ξεχωριστά. Ένας άλλος πιο γενικός τρόπος είναι να καθορίσουμε έναν ελάχιστο χρόνο μετεπιβίβασης συγκεκριμένο για κάθε σταθμό, καθώς και κάποιες εξαιρέσεις με τη μορφή ενός συνόλου επιπλέον εφικτών μετεπιβιβάσεων για κάθε άφιξη ενός τρένου σε έναν σταθμό.

4.1.2. Χρονικά-Επεκτεινόμενο Μοντέλο

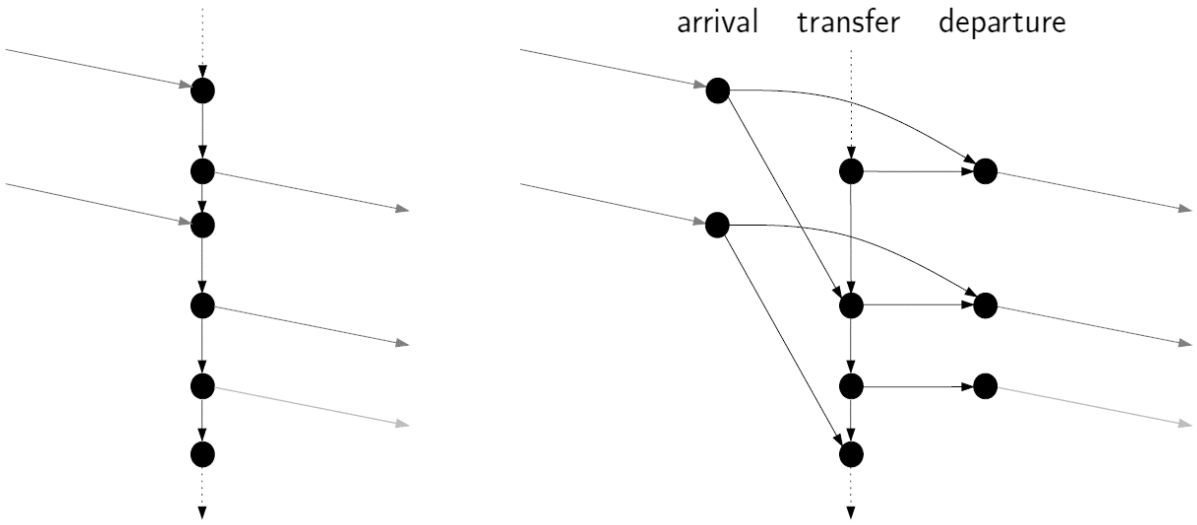
Για να λύσουμε το ρεαλιστικό πρόβλημα νωρίτερης άφιξης, προεκτείνουμε το χρονικά-επεκτεινόμενο μοντέλο κατασκευάζοντας τον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο ως εξής. Με βάση τον χρονικά-επεκτεινόμενο γράφο, κρατάμε για κάθε σταθμό ένα αντίγραφο όλων των κόμβων αναχώρησης -και άφιξης- στο σταθμό τους οποίους αποκαλούμε *κόμβους-μετεπιβίβασης* [*transfer-nodes*]· βλ. Σχήμα 4.1. Οι ακμές-παραμονής εισάγονται τώρα μεταξύ των κόμβων-μετεπιβίβασης. Για κάθε κόμβο άφιξης υπάρχουν δύο επιπλέον εξερχόμενες ακμές: μια ακμή στον κόμβο αναχώρησης του ιδίου συρμού· και μια δεύτερη ακμή στον κόμβο-μετεπιβίβασης με χρόνο μεγαλύτερο από ή ίσο με τον χρόνο του κόμβου-άφιξης συν το transfer(S). Τα μήκη ακμών ορίζονται όπως και στον ορισμό του αρχικού μοντέλου στην Ενότητα 3.3.

Πιο λεπτομερείς κανόνες μετεπιβίβασης μπορούν εύκολα να ενσωματωθούν και στον ρεαλιστικό χρονικά-επεκταμένο γράφο. Κάθε μετεπιβίβαση που επιτρέπεται κατ' εξαίρεση από το τρένο Z_1 στο τρένο Z_2 μπορεί να μοντελοποιηθεί με μια επιπλέον ακμή από τον κόμβο-άφιξης του Z_1 στον συγκεκριμένο σταθμό στον κόμβο-αναχώρησης του Z_2.

Η ορθότητα της μοντελοποίησης βασίζεται στο ακόλουθο θεώρημα.

Θεώρημα 4.1 Μια συντομότερη διαδρομή στον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο από τον πρώτο κόμβο-αναχώρησης στον σταθμό αναχώρησης Α με χρόνο αναχώρησης αργότερο από ή ίσο με τον δεδομένο χρόνο-εκκίνησης t_0, προς έναν από τους κόμβους-άφιξης του σταθμού προορισμού Β, αποτελεί μια λύση για το ρεαλιστικό ΠΝΑ στο εκτεταμένο χρονικά-επεκτεινόμενο μοντέλο. Η πραγματική διαδρομή μπορεί να βρεθεί με τον αλγόριθμο του Dijkstra.

Η τεχνική της παράλειψης των περιττών κόμβων που περιγράφεται στην Παράγραφο 3.3.1 μπορεί επίσης να εφαρμοστεί στον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο. Ωστόσο, εδώ ο γράφος μπορεί μόνο να μειωθεί κατά περίπου ένα τρίτο (αντί του μισού της απλής εκδοχή του γράφου), δεδομένου ότι μόνο οι κόμβοι-αναχώρησης έχουν out-degree ίσο με ένα.



Σχήμα 4.1 Μοντελοποίηση μετεπιβιβάσεων τρένων στο χρονικά-επεκτεινόμενο μοντέλο για έναν σταθμό δείγμα. Στα αριστερά παρουσιάζεται η αρχική μοντελοποίηση από το προηγούμενο κεφάλαιο, και στα δεξιά ο ρεαλιστικός χρονικά-επεκτεινόμενος γράφος με τρεις τύπους κόμβων: κόμβοι άφιξης, μετεπιβίβασης και αναχώρησης.

4.1.3. Χρονικά-Εξαρτώμενο Μοντέλο

Για να μοντελοποιήσουμε μη-μηδενικές μετεπιβιβάσεις τρένων, προεκτείνουμε το αρχικό χρονικά-εξαρτώμενο μοντέλο χρησιμοποιώντας πληροφορίες για τις διαδρομές που μπορεί να ακολουθούν τα τρένα. Ως εκ τούτου, μπορούμε να υποθέσουμε ότι μας δίνεται ένα σύνολο από *διαδρομές τρένων* [*train routes*] και τα αντίστοιχά τους *χρονικά προγράμματα* [*time schedules*]. Εξετάζουμε τόσο την περίπτωση σταθερού χρόνου μετεπιβίβασης ανά σταθμό, όσο και την περίπτωση των λεπτομερών κανόνων μετεπιβίβασης, τους οποίους επίσης αποκαλούμε και ως μεταβλητούς χρόνους μετεπιβίβασης. Στη συνέχεια, περιγράφουμε την κατασκευή ενός γράφου G = (V, E) που μοντελοποιεί αυτές τις δύο περιπτώσεις. Θα αναφερόμαστε στον G ως *γράφος διαδρομών-τρένων* [*train-route graph*] .

Έστω S το σύνολο των κόμβων οι οποίοι αναπαριστούν σταθμούς. Για u∈S, συμβολίζουμε με station(u) τον πραγματικό σταθμό που αναπαριστά το u. Λέμε ότι οι κόμβοι $s_0, s_1, ..., s_{k-1}$, k>0, σχηματίζουν μια διαδρομή τρένου αν υπάρχει κάποιο τρένο που ξεκινά το ταξίδι του από το station(s_0) και επισκέπτεται διαδοχικά τα station(s_1), ..., station(s_{k-1}) με τη συγκεκριμένη σειρά. Εάν υπάρχουν περισσότερα από ένα τρένα που ακολουθούν το ίδιο πρόγραμμα (δηλαδή τη σειρά με την οποία επισκέπτονται τους παραπάνω κόμβους), τότε λέμε ότι όλα αυτά ανήκουν στην ίδια διαδρομή-τρένου P. Να σημειωθεί ότι μπορεί να είναι station(s_i)=station(s_j), $i \neq j$, $s_i \neq s_j$, $0 \leq i, j \leq k-1$, για παράδειγμα όταν ένα τρένο πραγματοποιεί κυκλική διαδρομή.

Για u∈S, έστω Σ_u το σύνολο των διάφορων διαδρομών τρένων που σταματούν στο station(u), και έστω P_u ένα σύνολο που περιέχει ακριβώς έναν κόμβο για κάθε P∈Σ_u η οποία περνά μέσα από το station(u). Επίσης, έστω $P_u = |P_u|$, και $P = \cup_{u \in S} P_u$. Τότε, το σύνολο κόμβων V του G ορίζεται ως $V = S \cup P$. Για u∈S συμβολίζουμε με p_i^u, $0 \leq I < P_u$, τον κόμβο που αναπαριστά την i-στή διαδρομή τρένου που σταματάει στον u.

Το σύνολο ακμών E = A∪D∪$\overline{D}$∪R του G αποτελείται από τέσσερις τύπους ακμών, οι οποίοι ορίζονται ως εξής.

- $A = \bigcup_{u \in S} A_u$, όπου $A_u = \bigcup_{0 \le i < P_u} \{(p_i^u, u)\}$.
- $D = \bigcup_{u \in S} D_u$, όπου $D_u = \bigcup_{0 \le i < P_u} \{(u, p_i^u)\}$.
- $\overline{D} = \bigcup_{u \in S} \overline{D}_u$, όπου $\overline{D} = \emptyset$, αν ο χρόνος που απαιτείται για μια μετεπιβίβαση είναι ο ίδιος για όλα τα τρένα που σταματούν στο station(u)· και $\overline{D}_u = \bigcup_{0 \le i < P_u, i \ne j} \{(p_i^u, p_j^u)\}$ για το αντίθετο
- $R = \bigcup_{u,v \in S, 0 \le i < P_u, 0 \le j < P_v} \{(p_i^u, p_j^v)\}$: station(u) και station(v) επισκέπτονται διαδοχικά από την ίδια διαδρομή τρένου και p_i^u, p_j^v είναι οι αντίστοιχοι κόμβοι διαδρομής

Μια ακμή e ονομάζεται *διαδρομή* [*route*] ή *ακμή χρονοδιαγράμματος* [*timetable edge*] αν e∈R, και *ακμή μετεπιβίβασης* [*transfer edge*] αν e∈A∪D∪$\overline{D}$. Η μοντελοποίηση των διαδρομών τρένων στηρίζεται σε δύο επιπλέων παραδοχές.

Παραδοχή 4.1 Έστω u,v δύο οποιοιδήποτε κόμβοι στο S και $p_i^u \in P_u$, $p_j^v \in P_v$ έτσι ώστε $(p_i^u, p_j^v) \in R$. Αν d_1, d_2 είναι οi χρόνοι αναχώρησης από το p_i^u και α_1, α_2 είναι οι αντίστοιχοι χρόνοι άφιξης στο p_j^v, τότε $d_1 \le d_2 \Rightarrow \alpha_1 \le \alpha_2$.

Η παραδοχή αυτή είναι το ισοδύναμο της Παραδοχής 3.1 στην απλοποιημένη χρονικά-εξαρτώμενη προσέγγιση, και δηλώνει ότι δεν μπορεί να υπάρχουν δύο τρένα που ανήκουν στην ίδια διαδρομή τρένου, έτσι ώστε το πρώτο από αυτά που αφήνει το σταθμό να είναι ένα αργό τρένο, ενώ αυτό που ακολουθεί να είναι ένα γρήγορο τρένο και να φθάνει στον επόμενο σταθμό πριν από το πρώτο. Όταν αυτή η παραδοχή παραβιάζεται, μπορούμε να την επιβάλουμε διαχωρίζοντας τα τρένα που ανήκουν στην ίδια διαδρομή τρένου σε διαφορετικές κατηγορίες ταχύτητας, και ως εκ τούτου εισάγουμε νέες διαδρομές τρένων, μια για κάθε κατηγορία ταχύτητας, ενώ όλα τα τρένα ακολουθούν το ίδιο πρόγραμμα με πριν.

Παραδοχή 4.2 Για κάθε $u \in S$ και $p_i^u \in P_u$ τέτοια ώστε $(x, p_i^u) \in R$ για κάποιο x∈V, έστω $\delta_x^{u_i}$ το μικρότερο χρονικό διάστημα μεταξύ δύο διαδοχικών αφίξεων στο p_i^u από $(x, p_i^u) \in R$ και τ_u ο μέγιστος χρόνος που απαιτείται για μια μετεπιβίβαση στο station(u). Τότε, θα πρέπει να ισχύει ότι $\delta_x^{u_i} \ge \tau_u$.

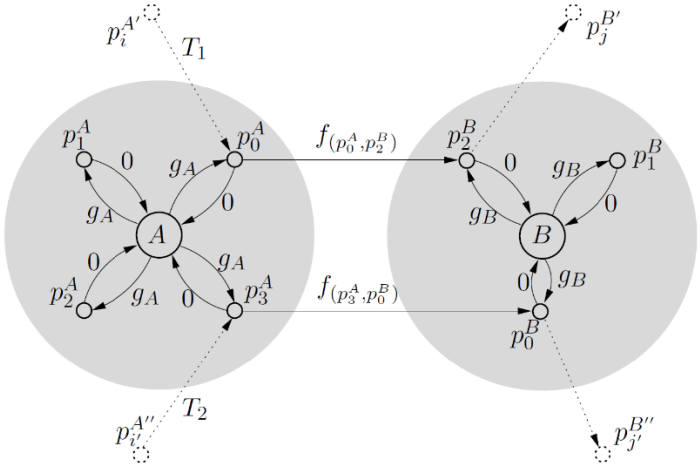
Η Παραδοχή 4.2 έχει σκοπό να διασφαλίζει ότι δεν είναι δυνατόν να βολεύει να περιμένουμε στους σταθμούς για να πάρουμε το επόμενο τρένο της ίδιας διαδρομής τρένου. Με άλλα λόγια, δεδομένου ότι η ισχύει η Παραδοχή 4.1, το να πάρουμε το πρώτο δυνατόν τρένο από κάποιον σταθμό A προς κάποιον σταθμό B δεν θα οδηγήσει στο να χάσουμε κάποια σύνδεση από τον B που θα μπορούσε να είχε χρησιμοποιηθεί, αν είχαμε χρησιμοποιήσει κάποιο τρένο (της ίδιας διαδρομής τρένου), που αναχώρησε αργότερα από αυτό που χρησιμοποιήσαμε. Θα δούμε αργότερα ότι η Παραδοχή 4.2 θα χρειάζεται μόνο στην περίπτωση όπου λαμβάνουμε υπόψη μεταβλητούς χρόνους μετεπιβίβασης εντός του ίδιου σταθμού.

Στη συνέχεια, θα υποθέτουμε ότι u,v∈S, $0 \le i, i' < P_u$, $0 \le j < P_v$, και ότι το T είναι ένα σύνολο που αντιπροσωπεύει το χρόνο.

Σταθερός Χρόνος Μετεπιβίβασης

Στην περίπτωση αυτή, τα κόστη ακμών του γράφου διαδρομών-τρένων ορίζονται ως εξής (βλ. Σχήμα 4.2). Έστω ότι το transfer(u)=transfer(station(u)) δηλώνει τον χρόνο μετεπιβίβασης του σταθμού όπου ανήκει ένας κόμβος u.

- Μια ακμή $(p_i^u, u) \in A$ έχει μηδενικό κόστος.
- Μια ακμή $(u, p_i^u) \in D_u$ έχει κόστος που καθορίζεται από μια συνάρτηση g_u: T→T, τέτοια ώστε $g_u(t) = t + transfer(u)$.
- Μία ακμή $(p_i^u, p_j^u) \in R$ έχει κόστος που καθορίζεται από μια συνάρτηση $f_{(p_i^u, p_j^u)}$: T → T τέτοια ώστε $f_{(p_i^u, p_j^u)}(t)$ είναι ο χρόνος κατά τον οποίο θα φτάσουμε στο p_j^u χρησιμοποιώντας την ακμή (p_i^u, p_j^u), Δεδομένου ότι φτάσαμε στο p_i^u στον χρόνο t.



Σχήμα 4.2 Ένα παράδειγμα μοντελοποίησης με διαδρομές τρένων και σταθερό χρόνο μετεπιβίβασης ανά σταθμό.

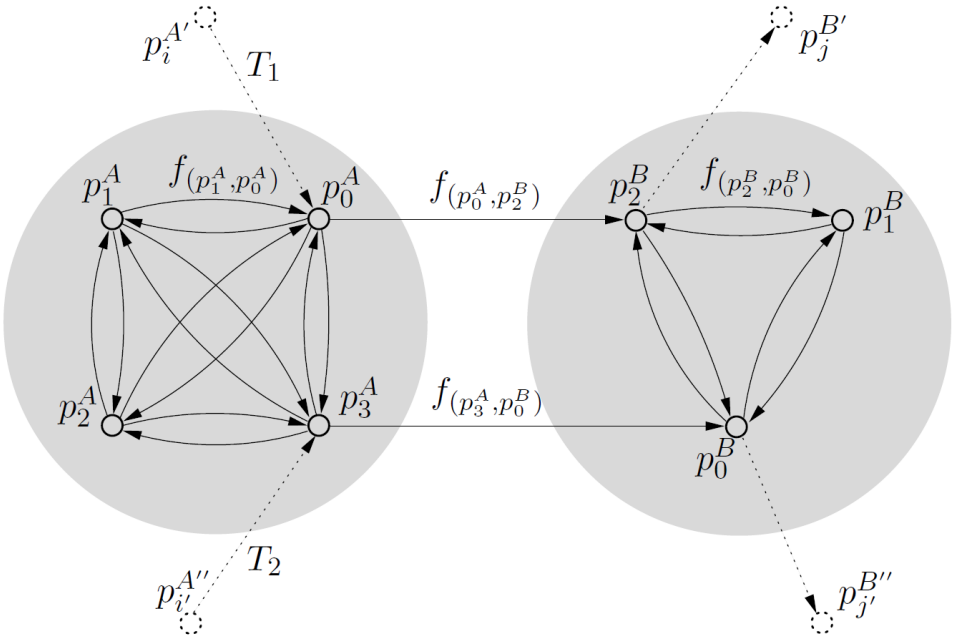
Για να λυθεί το ρεαλιστικό ΠΝΑ για ένα δεδομένο επερώτημα (α, β, t_0), πρέπει να βρούμε μια συντομότερη διαδρομή από τον σταθμό α στον σταθμό β χρησιμοποιώντας τον τροποποιημένο αλγόριθμο του Dijkstra που εισάχθηκε στην Παράγραφο 3.3.2, όπου για κάθε κόμβο χρησιμοποιούμε τη συνάρτηση κόστους που συνδέεται με αυτόν, όπως περιεγράφηκε παραπάνω. Θα πρέπει να βρούμε τη συντομότερη διαδρομή στον γράφο $G=(V, E_{\alpha,\beta})$ από το s_α στο s_β ξεκινώντας στον χρόνο t_0, όπου $\alpha=station(s_\alpha)$, $\beta = station(s_\beta)$, και $E_{\alpha,\beta}$ = AUDUR (βλέπε Σχήμα 4.2). Να σημειωθεί ότι όλες οι ακμές $e \in D_{s\alpha}$ πρέπει να έχουν μηδενικό κόστος.

Θεώρημα 4.2 Ο παραπάνω αλγόριθμος λύνει το ρεαλιστικό ΠΝΑ με σταθερούς χρόνους μετεπιβίβασης στους σταθμούς στο εκτεταμένο χρονικά-επεκτεινόμενο μοντέλο, υπό την προϋπόθεση ότι ισχύει η Παραδοχή 4.1.

Μεταβλητός Χρόνοι Μετεπιβίβασης

Τα κόστη ακμών του γράφου διαδρομών-τρένων σε αυτήν την περίπτωση ορίζονται ως εξής (βλ. Σχήμα 4.3).

- Μια ακμή $(p_i^u, u) \in A$ έχει μηδενικό κόστος.
- Μια ακμή $(u, p_i^u) \in D_u$ έχει μηδενικό κόστος. Μια ακμή $(p_i^u, p_{i'}^u) \in D_u$ έχει κόστος που καθορίζεται από μια συνάρτηση $f_{(p_i^u, p_{i'}^u)}: T \rightarrow T$ τέτοια ώστε $f_{(p_i^u, p_{i'}^u)}(t)$ να αναπαριστά τον χρόνο που απαιτείται από έναν επιβάτη ο οποίος έφτασε στο station(u) στον χρόνο t με ένα τρένο της διαδρομής τρένου p_i^u, να μετεπιβιβαστεί στο πρώτο δυνατό τρένο της διαδρομής $p_{i'}^u$. Ειδικότερα, $f_{(p_i^u, p_{i'}^u)}(t) = t + \tau^u_{change(i,i')}(t)$, όπου $\tau^u_{change(i,i')}(t)$ είναι η συνάρτηση που, για κάθε χρόνο άφιξης, επιστρέφει τον αντίστοιχο χρόνο μετεπιβίβασης.
- Μια ακμή $(p_i^u, p_j^u) \in R$ έχει κόστος που καθορίζεται από μια συνάρτηση $f_{(p_i^u, p_j^u)}: T \rightarrow T$ τέτοια ώστε $f_{(p_i^u, p_j^u)}(t)$ να είναι ο χρόνος κατά τον οποίο θα φτάσουμε στο p_j^u χρησιμοποιώντας την ακμή (p_i^u, p_j^u), δεδομένου ότι φτάσαμε στο p_i^u στον χρόνο t.



Σχήμα 4.3 Ένα παράδειγμα μοντελοποίησης με διαδρομές τρένων με μεταβλητούς χρόνους μετεπιβίβασης στους σταθμούς.

Όπως και πριν, δοθέντος ενός επερωτήματος ρεαλιστικού ΠΝΑ (α, β, t_0), το μόνο που χρειάζεται είναι να βρεθεί μια συντομότερη διαδρομή από τον σταθμό α στον σταθμό β στον παραπάνω γράφο, όπου για κάθε ακμή χρησιμοποιείται η συνάρτηση σχετικού κόστους, όπως περιεγράφηκε παραπάνω. Αυτό επιτυγχάνεται και πάλι με τον τροποποιημένο αλγόριθμο του Dijkstra (βλ. Παράγραφο 3.3.2). Στην πραγματικότητα, χρειάζεται να βρεθεί η συντομότερη διαδρομή στον γράφο $G = (V, E_{\alpha,\beta})$ από το s_α στο s_β ξεκινώντας στον χρόνο t_0, όπου α =

station(s_α), β=station(s_β), αλλά τώρα $E_{\alpha,\beta}= D_{s\alpha} \cup A_{s\beta} \cup \overline{D} \cup R$ (βλ. Σχήμα 4.3). Να σημειωθεί ότι όλες οι ακμές $e \in D_{s\alpha}$ πρέπει να έχουν μηδενικό κόστος.

Έστω οι κόμβοι p_i^u , $p_j^u \in P_u$, $0 \leq i, j < P_u$, $i \neq j$, τέτοιοι ώστε $(p_i^u , p_j^u) \in \overline{D}$. Επιπρόσθετα έστω ο κόμβος $p_{i'}^u \in P_v$, $0 \leq i' < P_v$ τέτοιος ώστε $(p_{i'}^u, p_i^u) \in R$. Για να είμαστε ικανοί να εφαρμόσουμε τον παραπάνω αλγόριθμο και να λύσουμε το ΠΝΑ σε αυτή την περίπτωση, πρέπει να διασφαλίσουμε ότι οι συναρτήσεις είναι μη φθίνουσες. Η Παραδοχή 4.1 διασφαλίζει ότι η $f_{(p_{i'}^u, p_i^u)}(t)$ είναι μη φθίνουσα. Αυτό που πρέπει να αποδείξουμε είναι ότι η $f_{(p_i^u, p_j^u)}(t)$ είναι επίσης μη φθίνουσα, όταν $t \in T_{u_i}$, όπου το T_{u_i} συμβολίζει το σύνολο των χρονικών τιμών αφίξεων του p_i^u από την ακμή $(p_{i'}^u, p_i^u) \in R$. (Να σημειωθεί ότι ο $p_{i'}^u$ είναι ο μόνος γειτνιάζον κόμβος, δηλαδή, το πέρας μιας εισερχόμενης ακμής, του p_i^u που δεν είναι στο P_u.)

Λήμμα 4.1 Η συνάρτηση $f_{(p_i^u, p_j^u)}(t)$ είναι μη φθίνουσα όταν $t \in T_{u_i}$, όπου $T_{u_i} = \{ t | t$ είναι ο χρόνος άφιξης στο p_i^u από την ακμή $(p_{i'}^u, p_i^u) \in R\}$.

Το λήμμα 4.1 και η παραπάνω συζήτηση καθόρισαν το ακόλουθο θεώρημα.

Θεώρημα 4.3 Ο παραπάνω αλγόριθμος λύνει το ρεαλιστικό ΠΝΑ με μεταβλητούς χρόνους μετεπιβίβασης στους σταθμούς στο εκτεταμένο χρονικά-εξαρτώμενο μοντέλο, υπό την προϋπόθεση ότι ισχύουν οι Παραδοχές 4.1 και 4.2.

4.2. Ημέρες Κυκλοφορίας

Για όλα τα μοντέλα που περιεγράφηκαν μέχρι τώρα έχουμε υποθέσει ότι κάθε τρένο λειτουργεί καθημερινά, δηλαδή ότι το χρονοδιάγραμμα είναι πανομοιότυπο κάθε ημέρα. Σε αυτή την ενότητα συζητάμε για το πώς διαφορετικές ημέρες κυκλοφορίας μπορούν να ενσωματωθούν σε αυτά τα μοντέλα. Για κάθε τρένο μας δίνεται η πληροφορία για το ποια ημέρα του χρονοδιαγράμματος λειτουργεί.

4.2.1. Προδιαγραφές

Ένα χρονοδιάγραμμα είναι έγκυρο για έναν αριθμό N *ημερών κυκλοφορίας* [*traffic days*], και σε κάθε τρένο εκχωρείται ένα *πεδίο-bit* [*bit-field*] N bits που προσδιορίζει σε ποιες ημέρες κυκλοφορίας λειτουργεί το τρένο (για μεταμεσονύκτια τρένα μετράει η αναχώρηση της πρώτης στοιχειώδους σύνδεσης). Μια σύνδεση ορίστηκε στην Παράγραφο 3.1.2 ως η ακολουθία των στοιχειωδών συνδέσεων c_i μαζί με τους χρόνους αναχώρησης dep_i και τους χρόνους άφιξης arr_i. Δεδομένου ότι σε κάθε στοιχειώδης σύνδεση c_i έχει εκχωρηθεί ένα τρένο, η εγκυρότητα του c_i σε μια δεδομένη μέρα μπορεί να επαληθευτεί μέσω των ημερών κυκλοφορίας του αντίστοιχου τρένου. Η ημέρα εντός του χρονοδιαγράμματος μίας από τις στοιχειώδεις συνδέσεις c_i κωδικοποιείται σε μια τιμή χρόνου dep_i και μπορεί να υπολογιστεί από $[dep_i(P)/1440]$· βλ. επίσης Παράγραφο 2.2.1. Ως εκ τούτου, πρέπει να επεκτείνουμε τον ορισμό της συνεπούς σύνδεσης με τον ακόλουθο όρο:

- το c_i είναι έγκυρο την ημέρα $[dep_i(P)/1440]$

4.2.2. Χρονικά-Επεκτεινόμενο Μοντέλο

Όταν χρησιμοποιούνται οι ημέρες κυκλοφορίας, μια βέλτιστη σύνδεση μπορεί να μείνει για περισσότερο από μία ημέρα σε έναν ενδιάμεσο σταθμό (π.χ., ας υποθέσουμε ότι σε μια αργία δεν λειτουργούν καθόλου τρένα). Τέτοιες συνδέσεις δεν αντιστοιχούν σε απλές διαδρομές στον χρονικά-επεκτεινόμενο γράφο, και το πρόβλημα δεν μπορεί να επιλυθεί άμεσα χρησιμοποιώντας τον γράφο αυτό. Ως εκ τούτου, κάνουμε χρήση του πλήρως χρονικά-επεκτεινόμενου γράφου που παρουσιάστηκε στην Παράγραφο 3.2.1 για να αντιμετωπιστεί το πρόβλημα στη χρονικά-επεκτεινόμενη προσέγγιση. Όπως θα δούμε παρακάτω, δεν χρειάζεται να διατηρήσουμε ρητά αυτόν τον γράφο. Η εκτέλεση ενός αλγορίθμου στον πλήρως χρονικά- επεκτεινόμενο γράφο μπορεί να προσομοιωθεί είτε πάνω στον αρχικό (Παράγραφος 3.2.1) είτε πάνω στον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο (Παράγραφος 4.1.2).

Όπως περιεγράφηκε νωρίτερα για την απλοποιημένη περίπτωση του προβλήματος νωρίτερης άφιξης, ο πλήρως χρονικά-επεκτεινόμενος γράφος βασίζεται σε N αντίγραφα του χρονικά-επεκτεινόμενου γράφου, αν η περίοδος του χρονοδιαγράμματος αποτελείται από N ημέρες. Η κατασκευή μπορεί να γίνει κατ' αναλογία και για τη ρεαλιστική εκδοχή, συμπεριλαμβανομένης της μοντελοποίησης των ρεαλιστικών κανόνων μετεπιβιβάσεων. Για να ενσωματώσουμε τις πληροφορίες ημέρων κυκλοφορίας, διαγράφονται από τον γράφο όλες οι ακμές-τρένων που αντιστοιχούν σε στοιχειώδεις συνδέσεις οι οποίες δεν είναι έγκυρες την ημέρα i στο i-στο αντίγραφο. Και πάλι, υπάρχει μια ένα-προς-ένα αντιστοιχία μεταξύ των συνδέσεων (σύμφωνα, και με τις ημέρες κυκλοφορίας) και των διαδρομών στον πλήρως χρονικά-επεκτεινόμενο γράφο, η οποία δίνει άμεσα το ακόλουθο θεώρημα.

Θεώρημα 4.4 Μια συντομότερη διαδρομή στον αρχικό (ή στον ρεαλιστικό) πλήρως χρονικά-επεκτεινόμενο γράφο αποτελεί λύση για την απλοποιημένη (αντίστοιχα ρεαλιστική) εκδοχή του ΠΝΑ όταν οι στοιχειώδεις συνδέσεις είναι έγκυρες μόνο σε συγκεκριμένες ημέρες κυκλοφορίας.

Δεδομένου ότι το μέγεθος του πλήρως χρονικά-επεκτεινόμενου γράφου είναι τεράστιο (N φορές το μέγεθος του χρονικά-επεκτεινόμενου γράφου), μελετάμε τώρα πώς μπορούμε να αποφύγουμε τη διατήρηση αυτού του τεράστιου γράφου. Από κατασκευής, όλα τα μήκη ακμών στον πλήρως χρονικά-επεκτεινόμενο γράφο είναι μικρότερα της μιας ημέρας. Έστω μια εφαρμογή του αλγορίθμου του Dijkstra στον πλήρως χρονικά-επεκτεινόμενο γράφο, και έστω t ο χρόνος που σχετίζεται με τον πρώτο κόμβο στην ουρά προτεραιότητας. Όλοι οι άλλοι κόμβοι στην ουρά προτεραιότητας έχουν έναν σχετικό χρόνο μεγαλύτερο από ή ίσο με t και μικρότερο από t+1440. Αυτή η παρατήρηση επιτρέπει την προσομοίωση του αλγόριθμου εφαρμόζοντας μια τροποποίηση του αλγορίθμου του Dijkstra στον χρονικά-επεκτεινόμενο γράφο: ο χρονικά-επεκτεινόμενος γράφος απεικονίζει τον υπογράφο του πλήρως χρονικά-επεκτεινόμενου γράφου που προκύπτει από τους κόμβους με χρόνους στο χρονικό διάστημα [t, t+1440], και αγνοώντας όλες τις ακμές-τρένων που αντιστοιχούν στα τρένα τα οποία δεν λειτουργούν στην εξεταζόμενη ημέρα. Στην προσομοίωση, πρέπει να καθοριστεί η εξεταζόμενη ημέρα διαιρώντας τον τρέχον απόλυτο χρόνο αναχώρησης με το 1440 (ο τρέχον απόλυτος χρόνος αναχώρησης είναι ίσος με τον απόλυτο χρόνο που σχετίζεται με τον κόμβο πηγή s συν την τρέχουσα απόσταση του υπό μελέτη κόμβου). Κάθε φορά που παγιώνεται ένας κόμβος, δεν μαρκάρεται μόνιμα στον αλγόριθμο του Dijkstra. Μαρκάρεται και πάλι ως ένας κόμβος που δεν έχει αγγιχθεί, θέτοντας την ετικέτα απόστασής του άπειρη, επειδή από αυτό το σημείο και πέρα θεωρείται πως είναι το αντίγραφο του κόμβου της επόμενης ημέρας. Αυτό μπορεί να γίνει με ασφάλεια, δεδομένου ότι ο πλήρως χρονικά-

επεκτεινόμενος γράφος είναι D.A.G. και πως ο αλγόριθμος του Dijkstra επεξεργάζεται τους κόμβους με τοπολογική σειρά: καμία ακμή δεν δείχνει προς τα πίσω, και έτσι ο κόμβος μπορεί να "χρησιμοποιηθεί εκ νέου".

4.2.3. Χρονικά-Εξαρτώμενο Μοντέλο

Το μοντέλο, όπως ορίζεται στην Παράγραφο 3.2.2 ή στην Παράγραφο 4.1.3 είναι εγγενώς ικανό να χειριστεί ημέρες κυκλοφορίας. Για ακμές διαδρομών, το μήκος καθορίζεται από την πρώτη στοιχειώδη σύνδεση σε αυτή την ακμή που αναχωρεί αργότερα από τον χρόνο άφιξης. Τώρα, το μήκος καθορίζεται από την πρώτη ακμή που αναχωρεί αργότερα από τον χρόνο άφιξης που είναι έγκυρος κατά την υπό εξέταση ημέρα (η ημέρα υπολογίζεται διαιρώντας τον χρόνο άφιξης με το 1440).

Να σημειωθεί ωστόσο, ότι η τεχνική επιτάχυνσης για την αποφυγή δυαδικής αναζήτησης στον υπολογισμό των μηκών ακμών που περιγράφεται στην Παράγραφο 3.4.2 δεν μπορεί πλέον να εφαρμοστεί άμεσα.

4.3. Το Πρόβλημα Ελάχιστου Αριθμού Μετεπιβιβάσεων

Εκτός από τη Νωρίτερη Άφιξη (ΝΑ), ο Ελάχιστος Αριθμός Μετεπιβιβάσεων (ΕΑΜ) είναι επίσης ένα σημαντικό κριτήριο στις πληροφορίες χρονοδιαγράμματος. Παρακάτω ορίζεται επίσημα το πρόβλημα της ελαχιστοποίησης του αριθμού μετεπιβιβάσεων και παρουσιάζεται πώς μπορεί να επιλυθεί χρησιμοποιώντας τους γράφους που κατασκευάστηκαν τόσο στο χρονικά-επεκτεινόμενο όσο και στο χρονικά-εξαρτώμενα μοντέλο για το πρόβλημα νωρίτερης άφιξης.

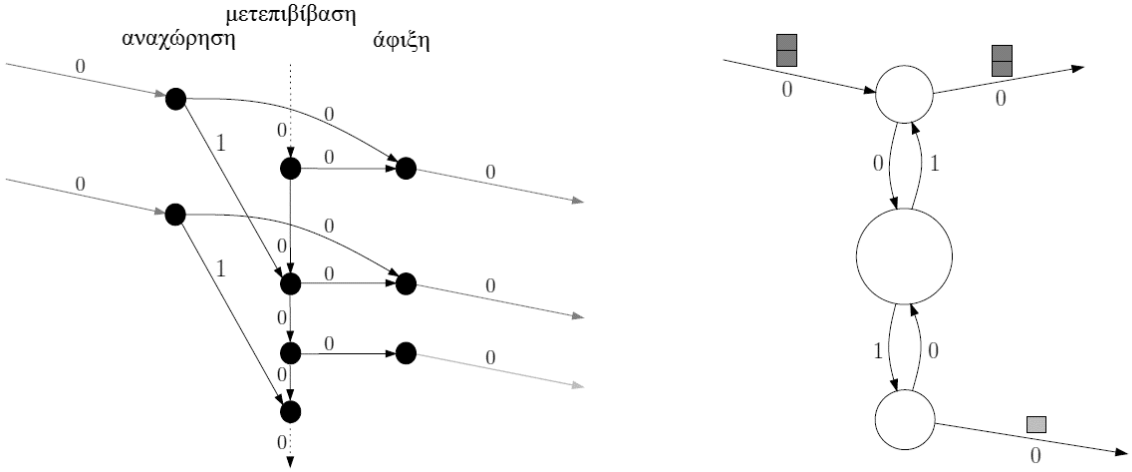
4.3.1. Προδιαγραφές Προβλήματος

Στο *Πρόβλημα Ελάχιστου Αριθμού Μετεπιβιβάσεων (ΠΕΑΜ)* [*Minimum Number of Transfers Problem (MNTP)*], ένα επερώτημα αποτελείται μόνο από έναν σταθμό αναχώρησης Α και έναν σταθμό άφιξης Β. Τα τρένα υποτίθεται ότι λειτουργούν καθημερινά, και δεν υπάρχει κανένας περιορισμός ως προς τον αριθμό των ημερών που ένα χρονοδιάγραμμα είναι έγκυρο. Όλες οι συνδέσεις από το Α στο Β είναι έγκυρες, και το κριτήριο βελτιστοποίησης είναι να ελαχιστοποιηθεί ο αριθμός των μετεπιβιβάσεων. Πιο συγκεκριμένα, έστω $P = (c_1, ..., c_k)$ μια σύνδεση από το Α στο Β και έστω $trans_i(P) \in \{0,1\}$ μια μεταβλητή που συμβολίζει εάν μια μετεπιβίβαση είναι απαραίτητη από μια στοιχειώδη σύνδεση c_i στο c_{i+1}, $1 \leq i < k$. Τότε, $trans_i(P)=1$ αν $Z(c_{i+1}) \neq Z(c_i)$, και $trans_i(P)=0$ αλλιώς. Κατά συνέπεια, ο στόχος του ΠΕΑΜ είναι να ελαχιστοποιήσει, μεταξύ όλων των P, την ποσότητα $\sum_{i=1}^{k-1} trans_i(P)$.

4.3.2. Μοντελοποίηση

Οι γράφοι που ορίστηκαν για το ρεαλιστικό πρόβλημα νωρίτερης άφιξης τόσο στη χρονικά-επεκτεινόμενη (Παράγραφος 4.1.2) όσο και στη χρονικά-εξαρτώμενη (Παράγραφος 4.1.3) προσέγγιση μπορούν να χρησιμοποιηθούν για την επίλυση του προβλήματος ελάχιστου αριθμού μεταβιβάσεων με μια παρόμοια μέθοδο: στις ακμές που μοντελοποιούν μετεπιβιβάσεις αντιστοιχείται βάρος ίσο με ένα, και σε όλες τις άλλες ακμές αντιστοιχείται βάρος μηδέν, όπως παρουσιάζεται στο παράδειγμα του Σχήματος 4.4. Στην περίπτωση της χρονικής-επέκτασης όλες οι ακμές από κόμβους-άφιξης σε κόμβους-μετεπιβίβασης έχουν βάρος ένα, ενώ στη χρονικά-εξαρτώμενη περίπτωση οι ακμές στο σύνολο D (δηλαδή, ακμές από κόμβους που αναπαριστούν σταθμούς στους κόμβους-διαδρομών), εκτός από αυτές που ανήκουν στον σταθμό αναχώρησης, εκχωρείται βάρος ενός, και όλες οι άλλες ακμές έχουν μηδενικό βάρος. Να σημειωθεί ότι όλα τα κόστη ακμών στον χρονικά-εξαρτώμενο γράφο είναι τα στατικά εδώ. Η ορθότητα της μοντελοποίησης αυτής για την εύρεση των ΕΑΜ-βέλτιστων συνδέσεων ως συντομότερες διαδρομές. Η ορθότητα της μοντελοποίησης αυτής αποδεικνύεται και προκύπτει το ακόλουθο θεώρημα.

Θεώρημα 4.5 Μια συντομότερη διαδρομή στον ρεαλιστικό χρονικά-επεκτεινόμενο (ή διαδρομών-τρένων) γράφο, μαζί με τα κόστη ακμών που δίνονται παραπάνω, από έναν κόμβο που ανήκει στον (ή που αναπαριστά αντίστοιχα τον) σταθμό αναχώρησης σε έναν κόμβο που ανήκει στον (ή που αναπαριστά αντίστοιχα τον) σταθμό άφιξης είναι μια λύση στο ΠΕΑΜ.



Σχήμα 4.4 Βάρη ακμών για την επίλυση του προβλήματος ελάχιστου αριθμού μεταβιβάσεων. Στα αριστερά φαίνονται τα βάρη ακμών του χρονικά-επεκτεινόμενου δείγματος από το Σχήμα 4.1 που χρησιμοποιούνται για τη μοντελοποίηση του ΠΕΑΜ, και στα δεξιά ο γράφος διαδρομών-τρένων για το ίδιο παράδειγμα, υποθέτοντας ότι τα δύο πάνω τρένα σχηματίζουν μια διαδρομή τρένου, και το κάτω τρένο μια άλλη διαδρομή τρένου.

4.4. Δικριτηριακά Προβλήματα

Εξετάζουμε επίσης τα δικριτηριακά προβλήματα με τη νωρίτερη άφιξη (ΝΑ) και τον ελάχιστο αριθμό μετεπιβιβάσεων (ΕΑΜ) ως τα δύο κριτήρια. Στη συνέχεια, θα παραμετροποιήσουμε το δικριτηριακό πρόβλημα που εξετάζουμε με (X, Y), όπου X (και Y) είναι το πρώτο (αντίστοιχα δεύτερο) κριτήριο που θέλουμε να βελτιστοποιήσουμε και X, Y ∈ {ΝΑ, ΕΑΜ}. Το γενικό πρόβλημα και οι αλγόριθμοι συντομότερων διαδρομών πολλαπλών κριτηρίων έχουν περιγραφεί στην Ενότητα 2.7. Όσον αφορά στις πληροφορίες χρονοδιαγράμματος, ο Möhring (Möhring 1999) παρέχει μια επισκόπηση των μεθόδων που χρησιμοποιούνται όταν εμπλέκονται πολλαπλά κριτήρια. Οι Muller-Hannemann και Weihe (Muller-Hannemann and Weihe 2001) διεξήγαγαν πειράματα υπολογισμού όλων των Pareto-βέλτιστων διαδρομών σε έναν χρονικά-επεκτεινόμενο γράφο για πληροφορίες χρονοδιαγράμματος (εξέτασαν επίσης και άλλους γράφους). Βρήκαν ότι, για διάφορα προβλήματα πολλαπλών κριτηρίων σε γράφους πληροφοριών χρονοδιαγράμματος, υπάρχουν κατά μέσο όρο πολύ λίγες Pareto-βέλτιστες διαδρομές ανά κόμβο, γεγονός που υποδηλώνει ότι ο αλγόριθμος labelling είναι εφικτός παρά το ότι γενικά έχει εκθετικό χρόνο τρεξίματος. Επίσης οι Muller-Hannemann και Schnee εφάρμοσαν έναν αλγόριθμο ετικέτας για την επίλυση προβλημάτων πολλαπλών κριτηρίων στη χρονικά-επεκτεινόμενη προσέγγιση (Muller-Hannemann and Weihe 2001). Σε αυτήν την εργασία, διαπίστωσαν ότι η έννοια της Pareto βελτιστότητας είναι μερικές φορές πολύ αυστηρή, και προτείνουν να γίνει χρήση μιας πιο χαλαρής παραλλαγής της Pareto βελτιστότητας προκειμένου να βρεθούν όλες οι ελκυστικές συνδέσεις τρένων.

Δεδομένου ενός σταθμού αναχώρησης A, ενός σταθμού προορισμού B, και ενός χρόνου αναχώρησης t, μας ενδιαφέρουν τρείς παραλλαγές του προβλήματος:

- Πρώτα και πιο γενικά, θέλουμε να βρούμε όλες τις Pareto-βέλτιστες λύσεις (δηλαδή, το σύνολο των εφικτών λύσεων όπου ο πίνακας-γνωρισμάτων [attribute-vector] της μιας λύσης δεν κυριαρχείται από τον πίνακας-γνωρισμάτων μιας άλλης λύσης· βλ. Ορισμό 2.7).
- Δεύτερον, να υπολογιστεί μια λύση περιορισμένων πόρων για το (X, Y): μια λύση που ελαχιστοποιεί το πρώτο κριτήριο X ενώ διατηρεί το δεύτερο κριτήριο (το οποίο θεωρείται επίσης πόρος) Y κάτω από ένα συγκεκριμένο όριο . Μας ενδιαφέρει κυρίως εδώ η (ΝΑ, ΕΑΜ) περίπτωση και αναφέρονται σε αυτό το πρόβλημα ως το πρόβλημα νωρίτερης άφιξης με φραγμένο αριθμό μεταβιβάσεων (ΝΑΦΜ), το οποίο ορίζεται ως το πρόβλημα της εύρεσης μια έγκυρης και συνεπούς σύνδεση από A στο B τέτοια ώστε ο χρόνος άφιξης στο B να είναι ο νωρίτερος δυνατός, και να μπορεί να υποστεί τον επιπλέον περιορισμό ώστε ο συνολικός αριθμός μετεπιβιβάσεων που εκτελούνται κατά τη διαδρομή να μην είναι μεγαλύτερος από ένα επιπλέον δεδομένο όριο k.
- Τέλος, το τρίτο πρόβλημα ασχολείται με τη λεξικογραφικά πρώτη λύση: δηλαδή ανάμεσα σε όλες τις λύσεις που ελαχιστοποιούν το X αυτή με το ελάχιστο Y.

4.4.1. Χρονικά-Επεκτεινόμενο Μοντέλο

Χρησιμοποιούμε τον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο (βλ. Παράγραφο 4.1.2) για να βρούμε τη λεξικογραφικά πρώτη Pareto-βέλτιστη καθώς και όλες τις Pareto-βέλτιστες λύσεις. Παραδόξως, θα προκύψει ότι μπορούν επίσης να υπολογιστούν όλες οι Pareto-βέλτιστες λύσεις από την κανονική εκδοχή του αλγόριθμου του Dijkstra στη δικριτηριακή περίπτωση που μελετάμε εδώ. Να σημειωθεί ότι στη γενική περίπτωση (δηλαδή, είτε παραπάνω από δύο κριτήρια είτε δύο

κριτήρια στα οποία δεν περιλαμβάνεται ο χρόνος ταξιδιού), η μέθοδος αυτή δεν λειτουργεί και πρέπει να χρησιμοποιηθεί ένας γενικός αλγόριθμος πολλαπλών κριτηρίων.

Λεξικογραφικά Πρώτη Pareto-Βέλτιστη

Μελετάμε πρώτα την περίπτωση (NA, EAM). Κάθε ακμή e του ρεαλιστικού χρονικά-επεκτεινόμενου γράφου συνδέεται τώρα με ένα ζεύγος από κόστη (α, b) = (NA(e), EAM(e)), όπου NA(e) είναι το κόστος του e κατά την επίλυση του ρεαλιστικού ΠΝΑ (Παράγραφος 4.1.2) και EAM(e) είναι το κόστος του e κατά την επίλυση του ΠΕΑΜ (Ενότητα 4.3). Καθορίζουμε σε αυτά τα ζεύγη κόστους (a, b) την κανονική προσθήκη, δηλαδή,(α, b)+(α', b') = (α+α', b+b'), και τη λεξικογραφική σύγκριση, δηλαδή, (α, b)<(α', b') ⇔ (α<α') ή (α=α' και b+b'). Για να βρούμε τη λεξικογραφικά πρώτη (NA, EAM) Pareto-βέλτιστη λύση, όπως συζητήσαμε ήδη στην Παράγραφο 2.2.1, αρκεί να τρέξουμε τον αλγόριθμο του Dijkstra διατηρώντας ετικέτες απόστασης ως ζεύγη απόστασης και εκκινώντας την ετικέτα απόστασης του κόμβου-εκκίνησης s ως (0, 0). Η βέλτιστη λύση βρίσκεται όταν εξεταστεί ο κόμβος σε έναν σταθμό προορισμού για πρώτη φορά κατά την εκτέλεση του αλγορίθμου. Η (EAM, NA) περίπτωση είναι συμμετρική με τα παραπάνω και μπορεί να λυθεί παρόμοια.

Θεώρημα 4.6 Μια συντομότερη διαδρομή στον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο χρησιμοποιώντας ζεύγη-κόστους που συνδέονται με τις ακμές του, όπως ορίζεται παραπάνω αποτελεί μια λύση στο πρόβλημα εύρεσης της λεξικογραφικά πρώτης Pareto-βέλτιστης σύνδεσης (μεταξύ όλων των συνδέσεων που ελαχιστοποιούν το πρώτο κριτήριο, εκείνη με την ελάχιστη τιμή στο δεύτερο κριτήριο).

Να σημειωθεί ότι μπορεί να λυθεί κατά τον ίδιο τρόπο το *πρόβλημα αργότερης αναχώρησης* [latest-departure problem] με την ελαχιστοποίηση της διαφοράς μεταξύ του χρόνου άφιξης και του χρόνου αναχώρησης ως δεύτερο κριτήριο.

Όλες οι Pareto-Βέλτιστες

Όπως έχει ήδη αναφερθεί, η εύρεση όλων των Pareto-βέλτιστων λύσεων είναι γενικά ένα δύσκολο πρόβλημα, δεδομένου ότι ο αριθμός τους μπορεί να είναι εκθετικός. Ωστόσο, ο ρεαλιστικός πλήρως χρονικά-επεκτεινόμενος γράφος που χρησιμοποιήθηκε στην Ενότητα 4.2 έχει μια πολύ ενδιαφέρουσα και χρήσιμη ιδιότητα: Δεδομένου ενός κόμβου εκκίνησης s, για κάθε κόμβο v στον χρονικά-επεκτεινόμενο γράφο υπάρχει το πολύ μια Pareto-βέλτιστη s-v-διαδρομή. Αφού η λεξικογραφικά πρώτη s-v-διαδρομή είναι μια Pareto-βέλτιστη διαδρομή (βλ. Παράγραφο 2.2.1), αυτή η λεξικογραφικά πρώτη s-v-διαδρομή είναι η επιθυμητή Pareto-βέλτιστη s-v-διαδρομή. Ως εκ τούτου, μπορούμε να υπολογίσουμε για κάθε κόμβο του σταθμού προορισμού τη συντομότερη διαδρομή σύμφωνα με τα ζεύγη-κόστους (a, b) = (NA(e), EAM(e)) με την κανονική προσθήκη και τη λεξικογραφική σύγκριση. Όταν ο πρώτος κόμβος του σταθμού προορισμού έχει παγιωθεί, έχουμε βρει την πρώτη (NA, EAM) -Pareto-βέλτιστη σύνδεση. Επιτρέπουμε στον αλγόριθμο του Dijkstra να συνεχίσει· κάθε φορά που ένας κόμβος του σταθμού προορισμού παγιώνεται με έναν μικρότερο αριθμό μεταβιβάσεων σε σχέση με οποιαδήποτε από τις Pareto-βέλτιστες λύσεις που έχουν ήδη βρεθεί, βρίσκεται μια νέα Pareto-βέλτιστη σύνδεση (η οποία αντιστοιχεί στη συντομότερη διαδρομή για αυτόν τον κόμβο). Ο αλγόριθμος μπορεί να διακοπεί όταν βρεθεί η Pareto-βέλτιστη λύση με τον μικρότερο δυνατό αριθμό μετεπιβιβάσεων (δηλαδή, η λύση του ΠΕΑΜ).

Θεώρημα 4.7 Όλες οι Pareto-βέλτιστές λύσεις για τα δύο κριτήρια ΝΑ και ΕΑΜ μπορούν να απαριθμηθούν κατά τη διάρκεια ενός τρεξίματος του αλγόριθμου του Dijkstra στον πλήρως χρονικά-επεκτεινόμενο γράφο χρησιμοποιώντας ζεύγη-κόστους που συνδέονται με τις ακμές του όπως ορίζεται ανωτέρω.

Τώρα, όπως και στην Παράγραφο 4.2.2, ο ρεαλιστικός πλήρως χρονικά-επεκτεινόμενος γράφος δεν χρειάζεται να διατηρηθεί ρητά. Ο παραπάνω αλγόριθμος μπορεί και πάλι να προσομοιωθεί πάνω στον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο. Η προσομοίωση είναι πανομοιότυπη με εκείνη που περιγράφεται στην Παράγραφο 4.2.2.

Νωρίτερη Άφιξη με Φραγμένο Αριθμό Μετεπιβιβάσεων

Ο παραπάνω αλγόριθμος για την εύρεση όλων Pareto-βέλτιστων λύσεων μπορεί να τερματιστεί νωρίτερα αν είναι επιθυμητή μόνο μια λύση που ελαχιστοποιεί τη νωρίτερη άφιξη (ΝΑ), ενώ διατηρεί τον αριθμό μετεπιβιβάσεων (ΕΑΜ) κάτω από ένα δεδομένο όριο k. Από το Θεώρημα 4.7, όλες οι Pareto-βέλτιστες λύσεις απαριθμούνται με λεξικογραφική (ΝΑ, ΕΑΜ) σειρά κατά τη διάρκεια του αλγορίθμου, και ως εκ τούτου η λύση στο πρόβλημα νωρίτερης άφιξης με φραγμένες μετεπιβιβάσεις (ΝΑΦΜ) βρίσκεται όταν επεξεργαστεί ο πρώτος κόμβος στον προορισμό από τον αλγόριθμο που έχει έναν αριθμό μετεπιβιβάσεων μικρότερο ή ίσο με k. Ο χρόνος τρεξίματος μπορεί να βελτιωθεί περαιτέρω κατά τη διάρκεια του αλγορίθμου, εξετάζοντας μόνο υποσχόμενες διαδρομές με έναν αριθμό μετεπιβιβάσεων μικρότερο ή ίσο με k: αγνοούνται όλες οι ετικέτες κόμβων με αριθμό μετεπιβιβάσεων μεγαλύτερο από k. Διαμορφώνουμε αυτό το αποτέλεσμα ως ένα Πόρισμα του Θεωρήματος 4.7:

Πόρισμα 4.1 Το πρόβλημα ΝΑΦΜ μπορεί να λυθεί στη χρονικά-επεκτεινόμενη προσέγγιση εφαρμόζοντας μια τροποποίηση του αλγορίθμου που χρησιμοποιείται στο Θεώρημα 4.7. Ο τροποποιημένος αλγόριθμος εξετάζει μόνο υποσχόμενες διαδρομές και σταματά όταν καθοριστεί η πρώτη εφικτή λύση.

Να σημειωθεί ότι στην περίπτωση της ελαχιστοποίησης ΕΑΜ και διατήρησης της ΝΑ μικρότερης ή ίσης με t, το πρόβλημα μπορεί να αντιμετωπιστεί με παρόμοιο τρόπο. Και πάλι, ο αλγόριθμος που περιγράφεται παραπάνω μπορεί να χρησιμοποιηθεί για όλες τις Pareto-βέλτιστες λύσεις. Τώρα, ο αλγόριθμος τερματίζεται όταν επεξεργαστεί ένας κόμβος στον προορισμό ο οποίος έχει έναν χρόνο άφιξης μεγαλύτερο από t. Η λύση λαμβάνεται από τον τελευταίο κόμβο με χρόνο άφιξης μικρότερο ή ίσο με t.

4.4.2. Χρονικά-Εξαρτώμενο Μοντέλο

Χρησιμοποιούμε τον γράφο διαδρομών-τρένων που μοντελοποιεί το ρεαλιστικό ΠΝΑ (Παράγραφος 4.1.3) για την επίλυση των τριών παραλλαγών των προβλημάτων δικριτηριακής βελτιστοποίησης.

Λεξικογραφικά Πρώτη Pareto-Βέλτιστη

Παρουσιάζουμε μια προσέγγιση παρόμοια με αυτήν που περιεγράφηκε για τη χρονικά-επεκταμένη περίπτωση, που βρίσκει όμως μόνο τη λεξικογραφικά πρώτη (ΕΑΜ, ΝΑ) Pareto-βέλτιστη λύση. Αργότερα, θα εξηγήσουμε γιατί αποτυγχάνει να κάνει το ίδιο και για την (ΝΑ, ΕΑΜ) περίπτωση.

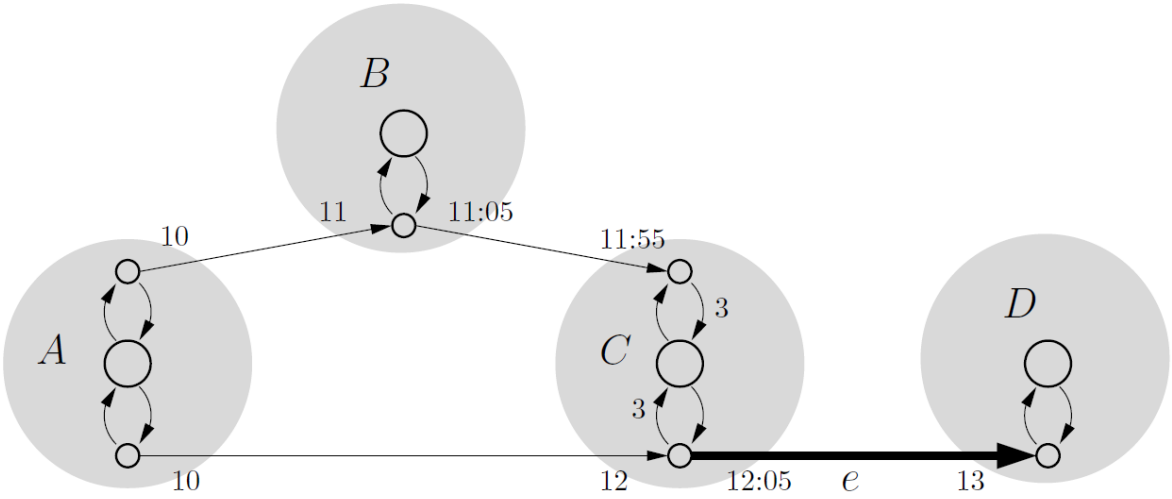
Μια λύση στο προβλήματα της λεξικογραφικά πρώτης (EAM, NA) Pareto-βέλτιστης μπορεί να επιτευχθεί εύκολα, αν αντί να χρησιμοποιήσουμε μια μόνο τιμή για το κόστος μιας ακμής, χρησιμοποιήσουμε ένα ζεύγος τιμών (a, b), και καθορίσουμε την κανονική προσθήκη και τη λεξικογραφική σύγκριση σε αυτά τα ζεύγη (ακριβώς όπως στην Παράγραφο 4.4.1). Τα χαρακτηριστικά τιμών a, b των ζευγών ενημερώνονται ξεχωριστά. Για την περίπτωση (EAM, NA), το a είναι το κόστος EAM και ορίζεται στο IN*(μη-αρνητικοί ακέραιοι), και το b είναι το κόστος NA που ορίζεται στο T (το σύνολο που αναπαριστά το χρόνο). Μια ακμή e ∈ E έχει κόστος που καθορίζεται από μια συνάρτηση h_e: (IN*, T) → (IN*, T), τέτοια ώστε κάθε τιμή χαρακτηριστικού να προσδιορίζεται από την αντίστοιχη συνάρτηση ακμής.

Θεώρημα 4.8 Μια συντομότερη διαδρομή στον γράφο διαδρομών-τρένων που χρησιμοποιεί ζεύγη-κόστους που συνδέονται με της ακμές τις, όπως ορίζεται παραπάνω αποτελεί λύση στο πρόβλημα της εύρεσης της λεξικογραφικά πρώτης (EAM, NA) Pareto-βέλτιστης σύνδεσης.

Σε αντίθεση με τη χρονικά-επεκταμένη περίπτωση, το συμμετρικό πρόβλημα εύρεσης μιας λεξικογραφικά πρώτης (NA, EAM) Pareto-βέλτιστης λύσης δεν μπορεί να λυθεί μόνο με τη χρήση κατάλληλων ζευγών-κόστους ακμών στον γράφο διαδρομών-τρένων, όπως στην Παράγραφο 4.4.1. Για να δείξουμε ότι μια τέτοια προσέγγιση μπορεί να αποτύχει, εξετάζουμε τον γράφο διαδρομών-τρένων που φαίνεται στο Σχήμα 4.5 με το επερώτημα για να βρούμε μια (NA, EAM) Pareto-βέλτιστη σύνδεση από το A έως D. Ας υποθέσουμε ότι υπάρχει ένα τρένο T_1 στην πρώτη διαδρομή από το A στο C μέσω του B, και ένα άλλο τρένο T_2 στη δεύτερη διαδρομή από το A στο D μέσω του C. Ας υποθέσουμε επίσης ότι και το T_1 και το T_2 αναχωρούν από το A αργότερα από τον δεδομένο χρόνο αναχώρησης, και έστω ότι το τρένο T_1 φτάνει νωρίτερα από το τρένο T_2 στον σταθμό C (έστω $C_1<C_2$ οι αντίστοιχοι χρόνοι άφιξης στο C και έστω D_2 ο χρόνος άφιξης του τρένου T_2 στο σταθμό D). Εξετάζουμε την ακμή e από τον σταθμό C στον σταθμό D. Ο χρονικά-εξαρτώμενος αλγόριθμος συντομότερης διαδρομής με ζεύγη-κόστους ακμής (a, b), όπου a είναι το κόστος της NA και b είναι το κόστος του EAM, θα αποτύχει να βρει μια (NA, EAM) Pareto-βέλτιστη λύση, επειδή η χρονικά-εξαρτώμενη συνάρτηση h_e είναι φθίνουσα σε αυτήν την περίπτωση. Ισχύει ότι $(C_1, 1) < (C_2, 0)$, και

$$h_e(C_1, 1) = (D_2, 1) > (D_2, 0) = h_e(C_2, 0).$$

Ως εκ τούτου, όταν ο αλγόριθμος χαλαρώνει την ακμή e αναθέτει την ετικέτα $(D_2, 1)$ στην αρχή του e, και κατά συνέπεια εξέρχεται μια σύνδεση με αυτήν την ετικέτα προορισμού, δηλαδή, μια σύνδεση που χρησιμοποιεί τα τρένο T_1 και T_2 με μια μετεπιβίβαση στον σταθμό C. Ωστόσο, αυτή δεν είναι η λεξικογραφικά πρώτη (NA, EAM) Pareto-βέλτιστη σύνδεση, δεδομένου ότι η σύνδεση που χρησιμοποιεί μόνο το τρένο T_2 δίδει μια λεξικογραφικά μικρότερη ετικέτα προορισμού $(D_2, 0)$.



Σχήμα 4.5 Οι λεξικογραφικά πρώτες (NA, EAM) συνδέσεις δεν μπορούν να βρεθούν απλώς με τη χρήση ζευγών ως κόστη ακμών στον γράφο διαδρομών-τρένων: Ας υποθέσουμε ότι μια σύνδεση από το A μέσω του B φτάνει νωρίτερα στο C (συμπεριλαμβανομένου και του χρόνου μετεπιβίβασης) από ότι ένα τρένο από το A στο C, και οι δύο συνδέσεις τελειώνουν με το ίδιο τρένο από το C στο D. Η ετικέτα της πηγής του e γίνεται (11:58, 1), και ο χρονικά-εξαρτώμενος αλγόριθμος εξάγει τη σύνδεση με μία μετεπιβίβαση μέσω του B, ενώ η βέλτιστη σύνδεση (A - C - D) δεν περιλαμβάνει καμία μεταβίβαση με τον ίδιο χρόνο άφιξης.

Νωρίτερη Άφιξη με Φραγμένο Αριθμό Μετεπιβιβάσεων

Στην παράγραφο αυτήν, περιγράφουμε δύο αλγόριθμους για την επίλυση του προβλήματος ΝΑΦΜ. Ο πρώτος είναι μια προσαρμογή της μεθόδου αντιγραφής-γράφων στο ρεαλιστικό χρονικά-εξαρτώμενο μοντέλο (γράφος διαδρομών-τρένων). Ο δεύτερος είναι μια προσαρμογή της προσέγγισης ετικέτας που περιγράφεται στην Παράγραφο 2.8.2 για την επίλυση συντομότερων διαδρομών περιορισμένων πόρων στο δικό μας ρεαλιστικό χρονικά-εξαρτώμενο μοντέλο. Έστω A ο σταθμός πηγή, B ο σταθμός προορισμού, t ο χρόνος αναχώρησης, και k ο φραγμός στον αριθμό μεταβιβάσεων.

Η ιδέα είναι ως εξής. Κατασκευάζουμε έναν νέο γράφο G'=(V', E') που αποτελείται από k+1 επίπεδα. Κάθε επίπεδο περιέχει ένα αντίγραφο του γράφου διαδρομών-τρένων G = (V, E), όπου E = AUDU$\overline{D}$UR (βλ. Παράγραφο 4.1.3). Για τον κόμβο u ∈ V, συμβολίζουμε το i-στο του αντίγραφο, το οποίο είναι τοποθετημένο στο i-στο επίπεδο, ως u_i, $0 \leq i \leq k$. Για κάθε ακμή (u, v) ∈ AUR, τοποθετούμε στο E' τις ακμές (u_i, v_i), για όλα τα i με $0 \leq i \leq k$. Για κάθε ακμή (u, v) ∈ D$\overline{D}$ τοποθετούμε στο E' τις ακμές (u_i, v_{i+1}), για όλα τα i με $0 \leq i \leq k$. Αυτές οι ακμές, που ενώνουν διαδοχικά επίπεδα, δείχνουν μετεπιβιβάσεις. Με την παραπάνω κατασκευή, είναι εύκολο να επαληθευθεί ότι μια διαδρομή από κάποιον κόμβο s_0 (αντίγραφο του s στο 0-στο επίπεδο) σε έναν κόμβο x_l (αντίγραφο του x στο l-στο επίπεδο) αντιπροσωπεύει μια διαδρομή από τον σταθμό του s στο σταθμό του x με l μετεπιβιβάσεις. Με άλλα λόγια, το πρόβλημα ΝΑΘΜ μπορεί να λυθεί εκτελώντας έναν υπολογισμό για χρονικά-εξαρτώμενη συντομότερη διαδρομή σε G' με στόχο να βρεθεί μια συντομότερη διαδρομή για τον κόμβο s_0 μηδέν στο επίπεδο 0, όπου το s αναπαριστά

τον σταθμό πηγή A στο G, στον πρώτο δυνατό u_i στο επίπεδο i, $0 \leq i \leq k$, όπου u είναι ο κόμβος του γράφου διαδρομών-τρένων που αναπαριστά τον σταθμό προορισμού B. Έστω n = |V| και m = |E|. Εφόσον το G' αποτελείται από k+1 αντίγραφα του G, μια εφαρμογή του αλγόριθμου του Dijkstra στο G' για την επίλυση του ΝΑΘΜ παίρνει χρόνο O(mk + nk • log(nk)) (υποθέτοντας ότι ο υπολογισμός του μήκους της χρονικά-εξαρτώμενης ακμής μπορεί να είναι γίνει σε σταθερό χρόνο).

Η προσαρμογή της προσέγγισης ετικέτας στον γράφο διαδρομών-τρένων G = (V, E) έχει ως εξής. Χρησιμοποιούμε τη χρονικά-εξαρτώμενη τροποποίηση του αλγόριθμου του Dijkstra (βλ. Παράγραφο 4.1.3), όπου τώρα διατηρούμε έως και k+1 (αντί της μιας) ετικέτες. Κάθε ετικέτα είναι της μορφής $(t_i, l_i)_u$, $0 \leq i \leq k$, που αντιπροσωπεύει τον τρέχον καλύτερο χρόνο t_i για να φθάσουμε τον κόμβο u πραγματοποιώντας ακριβώς l_i μετεπιβιβάσεις.

Έστω s ο κόμβος που αναπαριστά τον σταθμό πηγή A στον γράφο διαδρομών-τρένων. Αρχικά, εισάγουμε στην ουρά προτεραιότητας την ετικέτα $(t, 0)_s$. Η ουρά προτεραιότητας έχει διαταχθεί σύμφωνα με το χρόνο, με στόχο τον υπολογισμό της διαδρομής νωρίτερης άφιξης. Στον κύριο βρόχο του αλγορίθμου, μια ετικέτα $(t_l, l)_u$ εξάγεται από την ουρά προτεραιότητας. Αν το u αντιπροσωπεύει τον σταθμό προορισμού (δηλαδή, station(u) = B), ο αλγόριθμος τερματίζεται και t_l είναι η πρώτη άφιξη στον B με λιγότερες ή ίσες με k μετεπιβιβάσεις. Διαφορετικά, χαλαρώνουμε τις εξερχόμενες ακμές του u θεωρώντας ότι φτάνουμε στο u σε χρόνο t_l με l μετεπιβιβάσεις. Επιπλέον, εάν $(t_{l'}, l')$ ήταν η τελευταία ετικέτα του u που έχει εξαχθεί, τότε διαγράφουμε από την ουρά προτεραιότητας όλες τις ετικέτες της μορφής $(t_r, r)_u$ για $l<r<l'$, θέτοντας $l'= k$ στην περίπτωση όπου $(t_l, l)_u$ ήταν η πρώτη από τις ετικέτες του u που είχαν εξαχθεί. Με αυτόν τον τρόπο, έχουμε απορρίψει τις ετικέτες που κυριαρχούνται από $(t_l, l)_u$ από την ουρά προτεραιότητας, δεδομένου για όλα τα $(t_r, r)_u$ ισχύει ότι $t_l \leq t_r$ (καθώς $(t_l, l)_u$ εξάχθηκε πριν από το $(t_r, r)_u$) και $l<r$. Σαφώς, τέτοιες ετικέτες δεν είναι πλέον χρήσιμες καθώς $(t_l, l)_u$ αντιστοιχεί σε μια s-u-διαδρομή τουλάχιστον εξίσου γρήγορη με αυτήν που προτάθηκε από $(t_r, r)_u$, και με λιγότερες μετεπιβιβάσεις από την τελευταία. Ακριβώς για τους ίδιους λόγους, όταν χαλαρώσουμε μια ακμή $(u, v) \in E$ έχοντας βρει μια νέα ετικέτα $(t_{l1}, l_1)_v$ για v, στην πραγματικότητα ενημερώνουμε την ετικέτα του v μόνο αν μέχρι τώρα καμία ετικέτα του v σε έχει εξαχθεί από την ουρά προτεραιότητας, ή εάν η τελευταία ετικέτα του v που εξάχθηκε είχε αριθμό μετεπιβιβάσεων μεγαλύτερο από l_1. Η ακμή (u, v) αγνοείται αν $l_1>k$, και έτσι λαμβάνονται υπόψη μόνο υποσχόμενες διαδρομές (δηλαδή, διαδρομές με μικρότερες από ή ίσες με k μετεπιβιβάσεις) κατά τη διάρκεια του αλγορίθμου.

Όσον αφορά τώρα στην περιπλοκότητα του αλγορίθμου ετικέτας, χρειάζεται να δούμε ότι για κάθε κόμβο ο συνολικός αριθμός των ετικετών που σαρώνεται με σκοπό να βρεθούν εκείνες που είναι στην ουρά προτεραιότητας και μπορούν να διαγραφούν με ασφάλεια είναι O(k), ενώ ο συνολικός αριθμός των διαγραφών είναι O(nk), όπου n = |V|. Αυτό οφείλεται στο γεγονός ότι ελέγχουμε μόνο τις ετικέτες από τον τελευταίο γνωστό (από την λειτουργία delete-min) αριθμό μετεπιβιβάσεων, μέχρι τον προηγούμενο. Με τον τρόπο αυτό, κάθε ετικέτα ελέγχεται το πολύ μία φορά σε όλη τη διάρκεια της εκτέλεσης του αλγορίθμου. Καθώς κάθε ακμή θα χαλαρωθεί το πολύ k+1 φορές, ο συνολικός αριθμός των χαλαρώσεων θα είναι O(mk), όπου m = |E|. Μπορούμε επίσης να δούμε ότι ο συνολικός αριθμός των ετικετών που βρίσκονται στην ουρά προτεραιότητας είναι το πολύ O(nk). Εξαιτίας αυτού, ο χρόνος μιας delete-min ή μιας delete λειτουργίας είναι O(log(nk)). Αυτό σημαίνει ότι ο συνολικός χρόνος που απαιτείται για τον αλγόριθμο είναι O(mk+nk•log(nk)), που ασυμπτωτικά είναι ο ίδιος με τον προηγούμενο. Η συζήτηση σε αυτή την ενότητα παγιώνει τα εξής.

Θεώρημα 4.9 Το πρόβλημα νωρίτερης άφιξης με φραγμένο αριθμό μετεπιβιβάσεων μπορεί να επιλυθεί στο εκτεταμένο χρονικά-εξαρτώμενο μοντέλο σε χρόνο O(mk+nk•log(nk)), όπου n(και m) είναι ο αριθμός των κόμβων (αντίστοιχα. ακμών) του γράφου διαδρομών-τρένων, και k είναι ο φραγμός στον αριθμό των μετεπιβιβάσεων.

Όλες οι Pareto-Βέλτιστες

Όλες οι Pareto-βέλτιστες λύσεις μπορούν να υπολογιστούν από τον standard αλγόριθμο ετικέτας που εφαρμόζεται στον γράφο διαδρομών-τρένων. Προκειμένου να εξετάσουμε λιγότερες διαδρομές και για να είμαστε ικανοί να τερματίσουμε τον αλγόριθμο όταν βρεθούν όλες οι Pareto-βέλτιστες λύσεις στον προορισμό, προτείνουμε την εφαρμογή του τροποποιημένου αλγόριθμου ετικέτας που παρουσιάστηκε παραπάνω για το πρόβλημα ΝΑΦΜ με προϋπολογισμένους φραγμούς στον αριθμό των μετεπιβιβάσεων: Πρώτα, λύνουμε το ΠΝΑ και μετράμε τον αριθμό των μετεπιβιβάσεων που βρήκαμε, έστω M. Στη συνέχεια, τρέχουμε τον αλγόριθμο ετικέτας που είδαμε παραπάνω για την επίλυση του ΝΑΦΜ. Υπενθυμίζεται ότι ο αλγόριθμος διατηρεί για κάθε κόμβο M+1 ετικέτες, όπου η ετικέτα i, για $0 \leq i \leq M$, αποθηκεύει την καλύτερη λύση ΝΑ πραγματοποιώντας ακριβώς i μετεπιβιβάσεις (με την προϋπόθεση ότι υπάρχει τέτοια λύση), και απορρίπτει διαδρομές που έχουν κυριαρχηθεί. Ως εκ τούτου, αντί να σταματήσουμε τον αλγόριθμο όταν επεξεργαστεί μια ετικέτα που ανήκει στον σταθμό προορισμού στον κύριο βρόχο του αλγορίθμου, μπορούμε απλά να συνεχίσουμε με την εκτέλεση του αλγορίθμου για να παράγουμε την επόμενη λύση με το πολύ M-1 μετεπιβιβάσεις, θεωρώντας ότι ο φραγμός k είναι ίσος με M-1, και ούτω καθεξής, μέχρι να βρεθεί μια διαδρομή που λύνει το ΠΕΑΜ (ή μέχρι να μην γίνεται να βρεθεί καινούρια διαδρομή). Η παραπάνω συζήτηση συνοψίζεται ως πόρισμα του προηγούμενου θεωρήματος.

Πόρισμα 4.2 Ο τροποποιημένος αλγόριθμος ετικέτας του Θεωρήματος 4.9 μπορεί να απαριθμήσει όλες τις Pareto-βέλτιστες λύσεις για τα δύο κριτήρια ΝΑ και ΕΑΜ στο εκτεταμένο χρονικά-εξαρτώμενο μοντέλο.

4.5. Πειραματική Σύγκριση των Μοντέλων

Σε αυτήν την ενότητα παρουσιάζουμε το πώς οι χρονικά-επεκτεινόμενες και οι χρονικά-εξαρτώμενες προσεγγίσεις συγκρίνονται όταν λαμβάνονται υπόψη οι ρεαλιστικές εκδοχές των προβλημάτων και των μοντέλων (Schulz 2005). Η γενική πειραματική διάταξη είναι η ίδια με αυτήν της Ενότητας 3.5. Ως δεδομένα εισόδου χρησιμοποιείται μια τροποποίηση του χρονοδιαγράμματος ger-longdist που ονομάζεται ger-longdist1. Δεδομένου ότι εδώ λαμβάνονται υπόψη οι μετεπιβιβάσεις τρένων, χρησιμοποιούνται μόνο τα τρένα του χρονοδιαγράμματος που λειτουργούν σε μια συγκεκριμένη ημέρα, ενώ στην απλοποιημένη περίπτωση που συζητήθηκε πριν υπήρχε η παραδοχή ότι όλα τα τρένα λειτουργούν καθημερινά. Χρησιμοποιούνται επερωτήματα πραγματικού κόσμου και τυχαία επερωτήματα, όπως περιεγράφηκε στην Παράγραφο 3.5.1.

	Time-Expanded		Time-Dependent			
			Station	Route	Timetable	Transfer
Graph	Nodes	Edges	Nodes	Nodes	Edges	Edges
Simplified	289432	578864	6685	–	17577	–
Realistic	578864	1131164	6685	79784	72779	159568

Πίνακας 4.1 Παράμετροι γράφων για τα ρεαλιστικά μοντέλα, που εφαρμόστηκαν στο ίδιο χρονοδιάγραμμα-είσοδο ger-longdist1, και συγκρίθηκαν με τα απλοποιημένα μοντέλα που χρησιμοποιήθηκαν στο προηγούμενο κεφάλαιο.

4.5.1. Περιβάλλον Υλοποίησης

Και για τις δύο προσεγγίσεις υλοποιήθηκαν οι λύσεις που περιεγράφηκαν για το ρεαλιστικό πρόβλημα νωρίτερης άφιξης (EA-realistic) (Ενότητα 4.1), για το πρόβλημα ελάχιστου αριθμού μετεπιβιβάσεων (MNT) (Ενότητα 4.3), για το πρόβλημα όλων των Pareto-βέλτιστων (All Pareto-Opt) που περιέχει τη NA και το EAM ως τα δύο κριτήρια (Ενότητα 4.4), για τη λεξικογραφικά πρώτη (MNT, EA) Pareto-βέλτιστη (Lex-F(MNT, EA)), και για τη λεξικογραφικά πρώτη (EA, MNT) Pareto-βέλτιστη (Lex-F(EA, MNT)) -η τελευταία μόνο για το χρονικά-επεκτεινόμενο μοντέλο· βλ. Παράγραφο 4.4.2. Επιπλέον, για το χρονικά-εξαρτώμενο μοντέλο, εξετάστηκαν και οι δύο αλγόριθμοι για την επίλυση του προβλήματος νωρίτερης άφιξης με φραγμένο αριθμό μετεπιβιβάσεων (Παράγραφος 4.4.2): ο ένας βασίζεται στην προσέγγιση αντιγραφής-γράφων (EABT-BJ) και ο άλλος βασίζεται στην προσέγγιση ετικέτας (EABT-L).

Στις χρονικά-επεκτεινόμενες υλοποιήσεις μειώθηκε και πάλι το σύνολο των κόμβων παραλείποντας τους κόμβους-αναχώρησης στον ρεαλιστικό χρονικά-επεκτεινόμενο γράφο (βλ. Παράγραφο 3.4.1). Επίσης στις ρεαλιστικές χρονικά-εξαρτώμενες υλοποιήσεις εφαρμόστηκαν ευρετικά παρόμοια με τη μέθοδο "αποφυγής δυαδικής αναζήτηση" που περιεγράφηκε στην Παράγραφο 3.4.2.

4.5.2. Αποτελέσματα

Ο Πίνακας 4.2 παρουσιάζει τη σύγκριση μεταξύ των προβλημάτων που επιλύθηκαν στις ρεαλιστικές εκδοχές, και των χρονικά-επεκτεινόμενων, και των χρονικά-εξαρτώμενων προσεγγίσεων. Παρουσιάζονται οι παράμετροι κλειδιά -αριθμός κόμβων που έχουν αγγιχθεί, οι ακμές, και ο μέσος χρόνος τρεξίματος- για τα επερωτήματα πραγματικού-κόσμου. Περισσότερες λεπτομέρειες, καθώς και αποτελέσματα για τυχαία επερωτήματα και άλλα προβλήματα, παρουσιάζονται στους Πίνακες 4.3 και 4.4.

	Time-Expanded			**Time-Dependent**		
Problem	Nodes	Edges	Time [ms]	Nodes	Edges	Time [ms]
EA-realistic	40624	73104	78	44731	83662	50
MNT	101731	138417	125	26680	83173	38
Lex-F(MNT,EA)	99061	137075	161	28272	83363	83
All Pareto-Opt	123943	236887	287	78412	145444	181

Πίνακας 4.2 Κύρια αποτελέσματα για τα ρεαλιστικά μοντέλα, που εφαρμόστηκαν στο χρονοδιάγραμμα εισόδου ger-longdist1: ο μέσος αριθμός κόμβων και ακμών που αγγίζονται από τους αλγόριθμους, και ο χρόνος CPU σε millisecond για τα διάφορα προβλήματα που επιλύθηκαν.

Όσον αφορά τον χρόνο CPU, τα αποτελέσματα δείχνουν ότι η χρονικά-εξαρτώμενη προσέγγιση εξακολουθεί να αποδίδει καλύτερα από τη χρονικά-επεκτεινόμενη σε όλες τις περιπτώσεις που εξετάστηκαν. Ωστόσο, η διαφορά για το ρεαλιστικό ΠΝΑ δεν είναι τόσο μεγάλη όσο ήταν για το απλοποιημένο ΠΝΑ· στην πραγματικότητα, για τα επερωτήματα πραγματικού-κόσμου η επιτάχυνση είναι τώρα μόνο 1,5. Επιπλέον, αν αναλογιστεί κανείς τον αριθμό των ακμών που έχουν αγγιχθεί, τότε για επερωτήματα πραγματικού-κόσμου η επιτάχυνση είναι ακόμη μικρότερη και από 1 (δηλαδή, η χρονικά-επεκταμένη προσέγγιση είναι καλύτερη). Για τα άλλα προβλήματα που εξετάστηκαν ο χρόνος επιτάχυνσης κυμαίνεται ανάμεσα σε 1.5 και 3.3, ενώ η επιτάχυνση για τις ακμές που έχουν αγγιχθεί είναι 1,6 σε όλες τις περιπτώσεις.

Ειδικότερα, για το πρόβλημα ΕΑΜ, το εκτεταμένο χρονικά-εξαρτώμενο μοντέλο (γράφος διαδρομών-τρένων) είναι σαφώς ανώτερο από το εκτεταμένο χρονικά-επεκτεινόμενο μοντέλο (ρεαλιστικός χρονικά-επεκτεινόμενος γράφος)· η επιτάχυνση ως προς τον χρόνο CPU είναι 3,3 για επερωτήματα πραγματικού-κόσμου και 4,5 για τυχαία επερωτήματα. Αυτό είναι σαφές εφόσον και οι δύο γράφοι είναι στατικοί, και ο ρεαλιστικός χρονικά-επεκτεινόμενος γράφος είναι πολύ μεγαλύτερος και περιέχει πολλές περιττές πληροφορίες που δεν είναι αναγκαίες για την επίλυση του προβλήματος ΕΑΜ. Μια παρόμοια παρατήρηση ισχύει και για το λεξικογραφικά πρώτο (ΕΑΜ, ΝΑ) Pareto-βέλτιστο πρόβλημα, όπου η επιτάχυνση CPU είναι 1,9 για επερωτήματα πραγματικού κόσμου και 2 για τυχαία επερωτήματα, και για όλα τα Pareto-βέλτιστα προβλήματα, όπου η επιτάχυνση CPU είναι 1,5 για επερωτήματα πραγματικού-κόσμου και 1,8 για τυχαία επερωτήματα.

Να σημειώσουμε ότι η λύση στο πρόβλημα εύρεσης όλων Pareto-βέλτιστων παρουσιάζει μια αρκετά σταθερή συμπεριφορά σε σύγκριση με το ΠΝΑ και στις δύο προσεγγίσεις. Είναι 3,6 φορές πιο αργό από το ΠΝΑ για επερωτήματα πραγματικού-κόσμου και στα δύο μοντέλα, και

περίπου κατά τον ίδιο παράγοντα πιο αργό όταν εξετάζονται τυχαία επερωτήματα (3,3 στο χρονικά-επεκτεινόμενο μοντέλο και 4 φορές πιο αργό στο χρονικά-εξαρτώμενο).

Time-Expanded Model

Problem	Real Queries	Time [ms]	Average Nodes	Average Edges
EA-simplified	×	70	20760	41519
EA-realistic	×	78	40624	73104
MNT	×	125	101731	138417
Lex-F(EA,MNT)	×	82	40628	73123
Lex-F(MNT,EA)	×	161	99061	137075
All Pareto-Opt	×	287	123943	236887
EA-simplified		106	34469	61955
EA-realistic		122	61159	111301
MNT		212	169299	239841
Lex-F(EA,MNT)		129	61195	111386
Lex-F(MNT,EA)		259	163438	234297
All Pareto-Opt		405	170946	330150

Time-Dependent Model

Problem	Real Queries	Time [ms]	Average Touched	Average Edges
EA-simplified	×	10	2967	4365
EA-realistic	×	50	44731	83662
MNT	×	38	26680	83173
Lex-F(MNT,EA)	×	83	28272	83363
EABT-L	×	79	39070	71127
EABT-BJ	×	77	46127	82749
All Pareto-Opt	×	181	78412	145444
EA-simplified		11	3315	
EA-realistic		54	48200	89953
MNT		47	33455	96646
Lex-F(MNT,EA)		106	35262	97833
EABT-L		104	49179	90406
EABT-BJ		107	60493	109298
All Pareto-Opt		219	92378	172514

Πίνακας 4.3 Αναλυτικά αποτελέσματα για τα ρεαλιστικά προβλήματα. Ο άνω πίνακας αφορά στις χρονικά- επεκτεινόμενες υλοποιήσεις, και ο κάτω πίνακας στις χρονικά-εξαρτώμενες. Όλα τα αποτελέσματα βασίζονται στο χρονοδιάγραμμα ger-longdist1. Για σύγκριση με τα αρχικά

μοντέλα, συμπεριλαμβάνονται τα αποτελέσματα για την απλοποιημένη εκδοχή του προβλήματος νωρίτερης άφιξης (ΝΑ-απλοποιημένο). Ο Πίνακας 4.4 δείχνει πιο αναλυτικά αποτελέσματα για τη χρονικά-εξαρτώμενη εφαρμογή.

Problem	Real Queries	Timetable Edges touched	Transfer Edges touched	El. conn. per timetable edge
EA-simplified	×	4365	–	3.126
EA-realistic	×	38168	45494	0.319
MNT	×	21558	61615	–
Lex-F(MNT,EA)	×	22901	60462	0.178
EABT-L	×	32093	39034	0.247
EABT-BJ	×	37499	45250	0.261
All Pareto-Opt	×	65753	79691	0.211
EA-simplified		4811	–	3.005
EA-realistic		41011	48942	0.311
MNT		27235	69411	–
Lex-F(MNT,EA)		28779	69054	0.172
EABT-L		40638	49768	0.242
EABT-BJ		49392	59906	0.254
All Pareto-Opt		77610	94904	0.204

Πίνακας 4.4 Πιο αναλυτικά αποτελέσματα για τη χρονικά-εξαρτώμενη υλοποίηση, όπου διακρίνεται ο αριθμός των ακμών μεταξύ των ακμών-χρονοδιαγράμματος (οι οποίες έχουν χρονικά-εξαρτώμενα μήκη) και των ακμών-μετεπιβιβάσεων (οι οποίες έχουν στατικά μήκη). Επίσης, παρουσιάζονται οι μέσοι αριθμοί των στοιχειωδών συνδέσεων που αγγίζονται κατά τη διάρκεια του υπολογισμού των μηκών ακμών. Να σημειωθεί ότι η τιμή που είναι λιγότερο από ένα έχει νόημα εδώ, δεδομένου ότι για όλες τις ακμές-μετεπιβιβάσεων αυτός ο αριθμός είναι μηδέν: δεν χρειάζεται να αγγιχθεί καμία στοιχειώδης σύνδεση.

Περαιτέρω παρατηρήσεις σχετικά με κάθε μοντέλο ξεχωριστά είναι οι εξής.

Χρονικά-Επεκτεινόμενο Μοντέλο

Ο γράφος που χρησιμοποιήθηκε στο ρεαλιστικό ΠΝΑ (Πίνακας 4.3) έχει λιγότερους από διπλάσιους κόμβους και ακμές από τον γράφο που χρησιμοποιήθηκε στο απλοποιημένο ΠΝΑ, και έχει παρόμοια δομή. Έτσι, χρειάζεται μόνο λίγο περισσότερο χρόνο για να λυθεί το ρεαλιστικό ΠΝΑ από ότι να λυθεί το απλοποιημένο ΠΝΑ (11% περισσότερο χρόνο χρησιμοποιώντας επερωτήματα πραγματικού-κόσμου και 16% περισσότερο χρησιμοποιώντας τυχαία επερωτήματα). Το λεξικογραφικά πρώτο (ΝΑ, ΕΑΜ) Pareto-βέλτιστο πρόβλημα λύνεται με έναν πολύ παρόμοιο τρόπο με αυτόν του ρεαλιστικού ΠΝΑ. Είναι ενδιαφέρον να παρατηρήσουμε ότι ο χρόνος CPU καθώς ο μέσος αριθμός των κόμβων και των ακμών που αγγίχθηκαν είναι σχεδόν ταυτόσημοι. Σε αντίθεση, το ΕΑΜ, το λεξικογραφικά πρώτο (ΕΑΜ, ΝΑ) Pareto-βέλτιστο, και

όλα τα προβλήματα Pareto-βέλτιστων απαιτούν περισσότερο χρόνο CPU από το ρεαλιστικό ΠΝΑ, δεδομένου ότι πρέπει να εξερευνηθεί ένα πολύ μεγαλύτερο τμήμα του γράφου.

Χρονικά-Εξαρτώμενο Μοντέλο

Αν και ο γράφος διαδρομών-τρένων έχει περίπου 13 φορές περισσότερους κόμβους και ακμές από τον γράφο του αρχικού χρονικά-εξαρτώμενου μοντέλου (βλ. Πίνακα 4.1), τα πειραματικά αποτελέσματα που παρουσιάζονται στον Πίνακα 4.3 δείχνουν ότι ο χρόνος που απαιτείται για την επίλυση του ρεαλιστικού ΠΝΑ είναι μόνο 5 φορές πιο αργότερος από αυτόν του απλοποιημένου ΠΝΑ. Αυτό οφείλεται στο γεγονός ότι η δομή των δύο γράφων είναι πολύ διαφορετική. Ενώ ο αρχικός χρονικά-εξαρτώμενος γράφος έχει μόνο ακμές χρονοδιαγράμματος, σχεδόν το 70% των ακμών του γράφου διαδρομής-τρένου έχουν σταθερό κόστος και τα χρονοδιαγράμματα των άλλων ακμών έχουν πολύ λιγότερα γεγονότα από τις αντίστοιχες ακμές στον αρχικό χρονικά-εξαρτώμενο γράφο.

Το πρόβλημα ΕΑΜ λύνεται γρηγορότερα από το ρεαλιστικό ΠΝΑ, επειδή σε αυτή την περίπτωση όλα τα μήκη ακμών στον γράφο διαδρομής-τρένου είναι στατικά και ως εκ τούτου δεν χρειάζεται δυαδική αναζήτηση.

Η λύση του λεξικογραφικά πρώτου (ΕΑΜ, ΝΑ) Pareto-βέλτιστου προβλήματος περιλαμβάνει (και πάλι) χρονικά-εξαρτώμενα μήκη ακμών και είναι πιο αργή και από το ΠΕΑΜ και από το ρεαλιστικό ΠΝΑ, παρότι οι παράμετροι απόδοσης του (αριθμός κόμβων και ακμών που έχουν αγγιχθεί) είναι παρόμοιες με εκείνες του ΠΕΑΜ. Αυτό οφείλεται στο γεγονός ότι σε αυτό το πρόβλημα χρειάζονται περισσότεροι υπολογισμοί προκειμένου να διατηρηθεί ο σωρός προτεραιότητας. Είναι ενδιαφέρον να παρατηρήσουμε ότι ο αλγόριθμος για την επίλυση του λεξικογραφικά πρώτου (ΕΑΜ, ΝΑ) Pareto-βέλτιστου προβλήματος αγγίζει κατά μέσο σχεδόν το ήμισυ των στοιχειωδών συνδέσεων από ότι όταν επιλύεται το ρεαλιστικό ΠΝΑ. Αυτό μπορεί να εξηγηθεί ως εξής. Ενώ επιλύουμε το πρόβλημα Lex-F(ΕΑΜ, ΝΑ), οι πρώτες συνδέσεις που πρέπει να εξεταστούν είναι αυτές χωρίς μετεπιβιβάσεις (δηλαδή, εκείνες που ανήκουν στις διαδρομές τρένου που διέρχονται από το σταθμό αναχώρησης), οι επόμενες συνδέσεις θα είναι εκείνες με μόνο μία μετεπιβίβαση, και ούτω καθεξής, αποφεύγοντας έτσι πραγματοποιήσουμε μια μετεπιβίβαση όσο αυτό δυνατόν. Τώρα, η εφαρμογή δεν χρησιμοποιεί δυαδική αναζήτηση για ακμή χρονοδιαγράμματος, εκτός εάν έχουμε φτάσει στην πηγή της ακμής μέσω μιας ακμής μετεπιβίβασης του ίδιου σταθμού. Επιπλέον, ο χρόνος αυτού του προβλήματος είναι μεγαλύτερος από τον χρόνο του ΠΝΑ, δεδομένου ότι, προκειμένου να συγκρίνουμε δύο ετικέτες κόμβων θα προκύπτει αρκετά συχνά η ανάγκη να συγκρίνουμε τη δεύτερη παράμετρο κόστος (χρόνος), όπως την πρώτη (αριθμός μετεπιβιβάσεων) καθώς η πρώτη είναι η ίδια για πολύ περισσότερους κόμβους.

Και οι δύο αλγόριθμοι για την επίλυση του προβλήματος ΝΑΦΜ απαιτούν πρακτικά τον ίδιο χρόνο. Είναι ενδιαφέρον να παρατηρήσουμε όμως ότι η προσέγγιση ετικέτας (ΝΑΜΦ-L) αγγίζει πολύ λιγότερους κόμβους και ακμές από την προσέγγιση αντιγραφής-γράφου (ΝΑΦΜ-BJ).

Ο αλγόριθμος για την απαρίθμηση όλων των Pareto-βέλτιστων λύσεων χρησιμοποιεί τον υπολογισμό του προβλήματος ΝΑΦΜ ως υποδιαδικασία. Χρειάζεται μόνο να διπλασιαστεί ο χρόνος που απαιτείται για την επίλυση του ΝΑΦΜ. Το ίδιο ισχύει και όταν συγκρίνεται με τον

αλγόριθμο για την επίλυση του λεξικογραφικά πρώτου (EAM, NA) Pareto-βέλτιστου προβλήματος.

4.5.3. Συζήτηση

Έχουμε συζητήσει για τα χρονικά-επεκτεινόμενα και τα χρονικά-εξαρτώμενα μοντέλα για διάφορα είδη προβλημάτων μονοκριτητριακής και δικριτηριακής βελτιστοποίησης πάνω σε συστήματα πληροφοριών χρονοδιαγράμματος. Στη χρονικά-επεκτεινόμενη περίπτωση, οι προεκτάσεις που μοντελοποιούν τις πιο ρεαλιστικές απαιτήσεις (όπως τη μοντελοποίηση μετεπιβιβάσεων) θα μπορούσαν να ενσωματωθούν με έναν περισσότερο ή λιγότερο άμεσο τρόπο και το κεντρικό χαρακτηριστικό της προσέγγισης είναι ότι μια λύση θα μπορούσε να παρασχεθεί σε ένα δεδομένο πρόβλημα βελτιστοποίησης με την επίλυση ενός προβλήματος συντομότερης διαδρομής σε έναν στατικό γράφο, ακόμα και για την εύρεση όλων των Pareto-βέλτιστων λύσεων στα υπό εξέταση δικριτηριακά προβλήματα βελτιστοποίησης. Στη χρονικά-εξαρτώμενη περίπτωση, το κεντρικό χαρακτηριστικό της ύπαρξης ενός κόμβου ανά σταθμό θα έπρεπε να παραβιαστεί όταν λαμβάνονται υπόψη πιο ρεαλιστικές απαιτήσεις (όπως η ενσωμάτωση των ελάχιστων χρόνων μετεπιβίβασης στους σταθμούς), και πιο εξεζητημένες τεχνικές πρέπει να χρησιμοποιηθούν στα δικριτηριακά προβλήματα βελτιστοποίησης για την αποτελεσματική λύση τους. Παρ' όλα αυτά, όλα τα υπό εξέταση προβλήματα θα μπορούσαν να απεικονιστούν αποτελεσματικά με την επέκταση του χρονικά-εξαρτώμενου μοντέλου που παρουσιάστηκε. Επιπλέον, προέκυψε ότι μια τέτοια μοντελοποίηση ήταν πιο συμπαγής σε σχέση με το εκτεταμένο χρονικά-επεκτεινόμενο μοντέλο, οδηγώντας έτσι σε καλύτερη επίδοση στην πράξη.

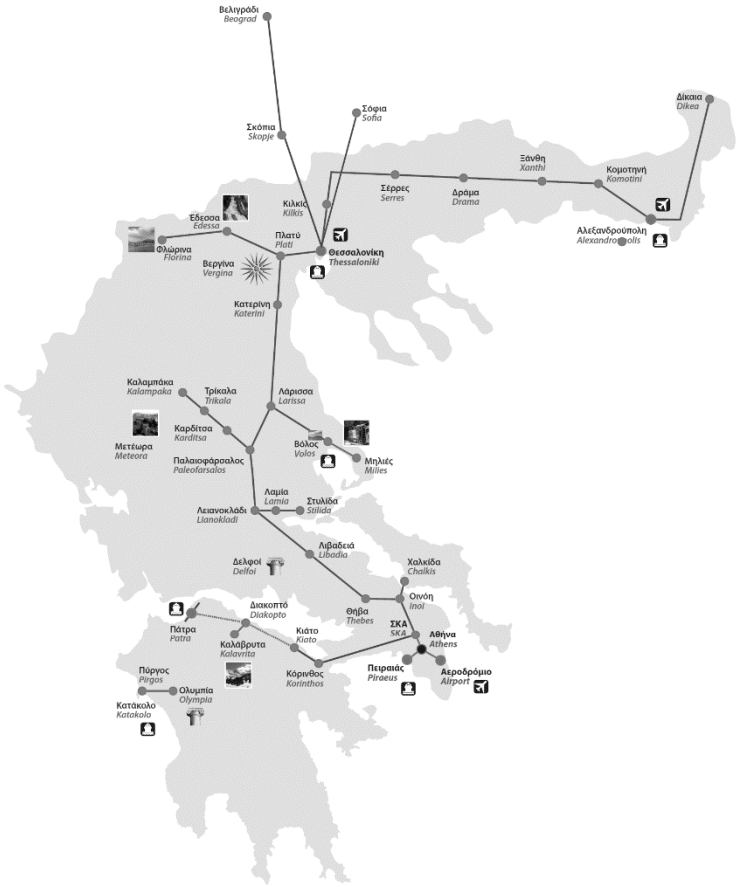
Η πειραματική μελέτη που παρουσιάσαμε έδειξε ότι η χρονικά-εξαρτώμενη προσέγγιση είναι σαφώς ανώτερη ως προς την απόδοση όταν εξετάζεται η αρχική εκδοχή των μοντέλων, και παρατηρούνται παράγοντες επιτάχυνσης στην περιοχή 10-40. Κατά την εξέταση των προεκτάσεων των μοντέλων για την επίλυση ρεαλιστικών εκδοχών των προβλημάτων βελτιστοποίησης, η χρονικά-εξαρτώμενη προσέγγιση εξακολουθεί να αποδίδει καλύτερα, αλλά όμως με πολύ μικρότερη διαφορά (η επιτάχυνση μειώνεται τώρα στο εύρος του 1,5 έως 3.3). Η χρονικά-επεκτεινόμενη προσέγγιση επωφελείται σε αυτήν την περίπτωση από την ευθεία μοντελοποίηση που επιτρέπει πιο άμεσες προεκτάσεις και πιο αποτελεσματικές λύσεις. Όταν πρέπει να ενσωματωθούν άλλα κριτήρια βελτιστοποίησης, είναι πιο πιθανό ότι μπορεί να μοντελοποιηθεί πιο άμεσα ως μήκη ακμών στο χρονικά-επεκτεινόμενο μοντέλο σε σχέση με το χρονικά-εξαρτώμενο μοντέλο. Σε περίπτωση που το κριτήριο μπορεί να εκφραστεί ως επιπρόσθετο κόστος για στοιχειώδης συνδέσεις, αυτό το κόστος προκαλεί μήκη ακμών στον χρονικά-επεκτεινόμενο γράφο, ενώ δεν είναι σαφές εάν το κόστος μπορεί να χαρτογραφηθεί σε εφικτά μήκη ακμών στη χρονικά-εξαρτώμενη προσέγγιση, καθώς εξετάζεται μόνο η πρώτη στοιχειώδης σύνδεσης ανά ακμή.

5. Υλοποίηση για το Χρονοδιάγραμμα του Ο.Σ.Ε.

5.1. Δεδομένα

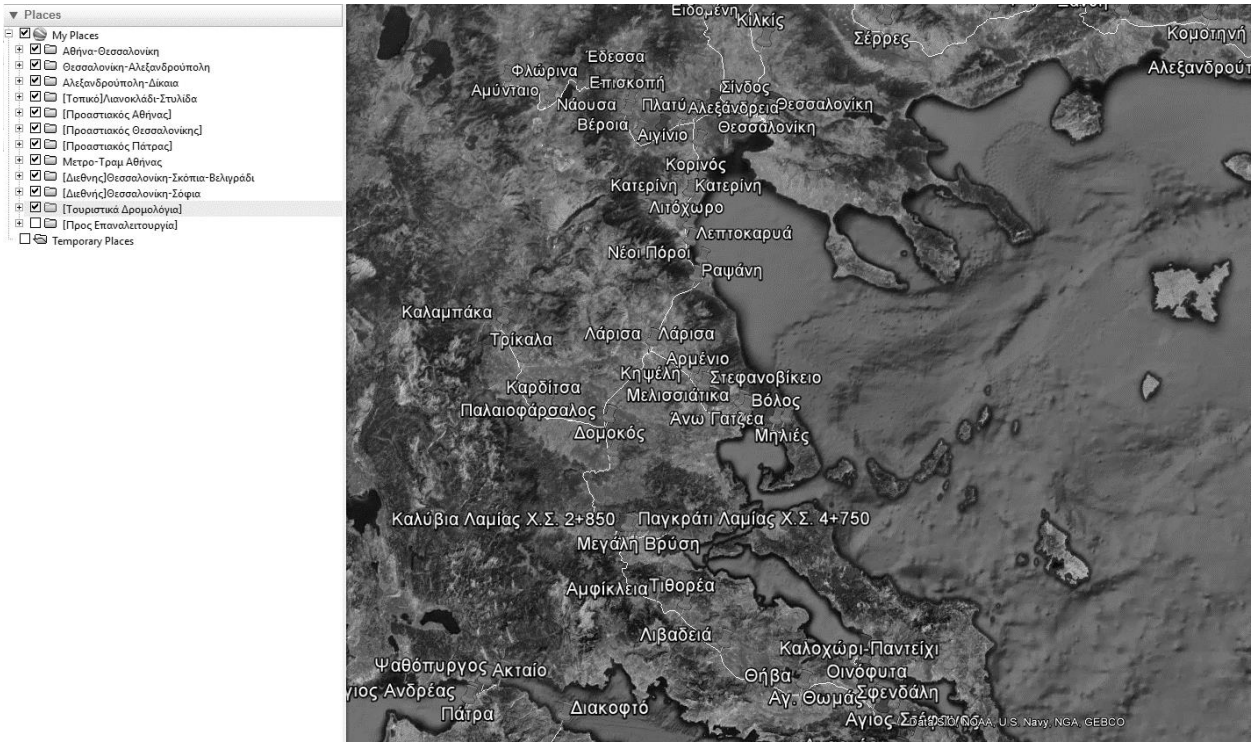
Για να είμαστε σε θέση να προχωρήσουμε σε οποιαδήποτε απόπειρα υλοποίησης, πρέπει πρώτα να έχουμε συγκεντρωμένα και δομημένα όλα τα δεδομένα μας. Ήταν ανάγκη λοιπόν να οργανώσουμε τα δεδομένα του χρονοδιαγράμματος του Ο.Σ.Ε. με κάποιο τρόπο ώστε να αποτελέσουν την είσοδο στις διάφορες υλοποιήσεις μας.

Το δίκτυο του Ο.Σ.Ε αποτελείται πρακτικά από μια βασική αρτηρία η οποία διανύει ολόκληρη την ηπειρωτική Ελλάδα. Η αρτηρία αυτή, καθώς και οι διακλαδώσεις της, παρουσιάζονται στην Εικόνα 5.1.



Εικόνα 5.1 Γενική απεικόνιση του ενεργού ελληνικού σιδηροδρομικού δικτύου.

Αρχικά μελετήσαμε λεπτομερώς το ενεργό αλλά και το ανενεργό δίκτυο, και συγκεντρώσαμε όλες τις αναγκαίες πληροφορίες, δηλαδή τις συντεταγμένες και τα ονόματα, σε μορφή .kml ώστε να μπορούμε να τα διαχειριστούμε εύκολα και να ανατρέχουμε σε αυτά όταν χρειάζεται. Επίσης, μιας και η μορφή αυτή είναι η επικρατέστερη σήμερα (ουσιαστικά πρόκειται για μορφή .xml με την ιδιαίτερη ικανότητα να μπορεί να οπτικοποιηθεί πάνω σε χάρτες), η ιδέα είναι να υπάρχουν διαθέσιμες αυτές οι πληροφορίες, για να ενσωματωθούν στο σύστημα αν χρειαστεί, ώστε να υπάρχει η δυνατότητα οπτικοποίησης της προτεινόμενης στον χρήστη διαδρομής. Στην παρακάτω εικόνα βλέπουμε ένα στιγμιότυπο των πληροφοριών αυτών, καθώς και τη δεντρική οργάνωσή τους.



Εικόνα 5.2 Στιγμιότυπο κατά την επεξεργασία των δεδομένων του δικτύου. Στα αριστερά φαίνεται η δεντρική ομαδοποίηση των αντικειμένων.

Μόλις οργανώσαμε τα στοιχεία δικτύου έπρεπε να οργανώσουμε τα δεδομένα χρονοδιαγράμματος. Αρχικά συγκεντρώσαμε όλους τους πίνακες με τα υφιστάμενα δρομολόγια του Ο.Σ.Ε., από το επίσημο website του οργανισμού. Έπειτα ερευνήσαμε τον πιο σύγχρονο και εύχρηστο τρόπο για να δομήσουμε τα δεδομένα αυτά, ώστε να μπορούμε να πειραματιστούμε πάνω τους. Απώτερος και ουσιαστικότερος στόχος μας είναι η πλήρης ανεξαρτητοποίηση της εφαρμογής από τις διάφορες δομές δεδομένων που υπάρχουν και η δυνατότητα εύκολης ολικής ή μερικής μεταβολής δεδομένων . Έτσι καταλήξαμε στη μορφή GTFS.

Το πρότυπο General Transit Feed Specification (GTFS) δημιουργήθηκε από την Google το 2005, και καθόρισε μια μορφή [format] για την εύκολη ανταλλαγή δεδομένων που αφορούν στις δημόσιες συγκοινωνίες. Επιτρέπει στις εταιρίες μαζικών συγκοινωνιών να δημοσιεύουν τους πίνακες χρονοδιαγράμματος μαζί με τις σχετικές γεωγραφικές πληροφορίες, αλλά και στους προγραμματιστές να μπορούν να αναπτύξουν τις εφαρμογές τους πάνω στη συγκεκριμένη δομή δεδομένων.

Το πρότυπο χωρίζει τα δεδομένα σε διάφορα επιμέρους αρχεία. Μιας και είναι περίπλοκο να ασχοληθούμε με όλες τις ειδικές περιπτώσεις ενός δικτύου μεταφορών, θα δώσουμε μια σύντομη περιγραφή της βασικής ιδέας.

Ένας σταθμός, ή γενικότερα μια ανεξάρτητη περιοχή στην οποία τα οχήματα παίρνουν και αφήνουν κόσμο, ονομάζεται stop. Μια ακολουθία από δύο ή περισσότερα stops που συμβαίνουν σε συγκεκριμένες ώρες σχηματίζουν ένα trip. Τα trips που παρουσιάζονται στους επιβάτες ως μια ενιαία υπηρεσία ομαδοποιούνται σε routes. Για κάθε trip δίνονται οι χρόνοι κατά τους οποίους το όχημα φτάνει και αναχωρεί από τα εκάστοτε stops. Τέλος, καθορίζονται και οι μέρες της εβδομάδας κατά τις οποίες τα trips λειτουργούν.

Τα ελληνικά μέσα μαζικής μεταφοράς προσαρμόζονται σταδιακά στο πρότυπο αυτό. Χαρακτηριστικό είναι πως ο Οργανισμός Αστικών Συγκοινωνιών Αθήνας (ΟΑΣΑ) παρέχει όλα τα δεδομένα των συγκοινωνιών (μετρό, λεωφορεία, τραμ, τρόλεϊ, προαστιακός σιδηρόδρομος) που διαχειρίζεται σε μορφή GTFS μέσω διαδικτύου.

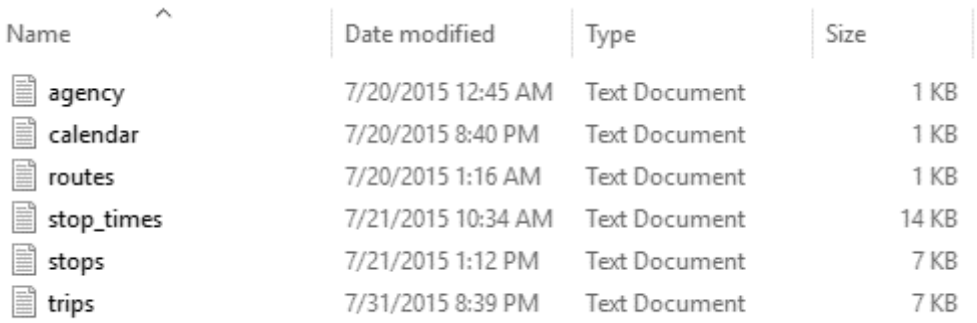
Επιλέξαμε αυτήν τη δομή δεδομένων καθώς έχει καθιερωθεί και χρησιμοποιείται από πολλές μεγάλες εταιρίες μεταφορών. Η δομή αυτή μας δίνει το πλεονέκτημα της άμεσης συμπερίληψης επιπλέον δικτύων. Τα δίκτυα αυτά μπορεί να είναι δίκτυα αεροπορικά, λεωφορείων, ferries κτλ. Η συμπερίληψη τους είναι εξαιρετικά εύκολη και η παραμετροποίηση του υφιστάμενου προγράμματος, που είναι αναγκαία ώστε να μπορεί να ξεχωρίζει τα διάφορα μέσα μεταξύ τους, έγκειται απλά και μόνο στην αναγνώριση ενός δείκτη από 0 έως 7 του πεδίου route_type όπως φαίνεται και στην παρακάτω εικόνα.

route_type	**Required**	The **route_type** field describes the type of transportation used on a route. Valid values for this field are: • **0** - Tram, Streetcar, Light rail. Any light rail or street level system within a metropolitan area. • **1** - Subway, Metro. Any underground rail system within a metropolitan area. • **2** - Rail. Used for intercity or long-distance travel. • **3** - Bus. Used for short- and long-distance bus routes. • **4** - Ferry. Used for short- and long-distance boat service. • **5** - Cable car. Used for street-level cable cars where the cable runs beneath the car. • **6** - Gondola, Suspended cable car. Typically used for aerial cable cars where the car is suspended from the cable. • **7** - Funicular. Any rail system designed for steep inclines. See a Google Maps screenshot highlighting the route_type.

Εικόνα 5.3 Οι 8 δυνατές τιμές που μπορεί να λάβει το πεδίο route_type καθώς και τα μέσα μεταφοράς που συμβολίζουν, σύμφωνα με το online εγχειρίδιο του προτύπου GTFS.

Είναι προφανής η ευκολία με την οποία μπορούμε να ενώσουμε διαφορετικά χρονοδιαγράμματα ώστε να σαρώνονται από το ίδιο σύστημα. Αυτή εξάλλου ήταν και η βασική ιδέα της Google: να είναι σε θέση να συγκεντρώσει όλα τα χρονοδιαγράμματα ώστε να μπορεί να υπολογίζει βέλτιστες διαδρομές με οποιοδήποτε μέσο μέσα από το Google Maps.

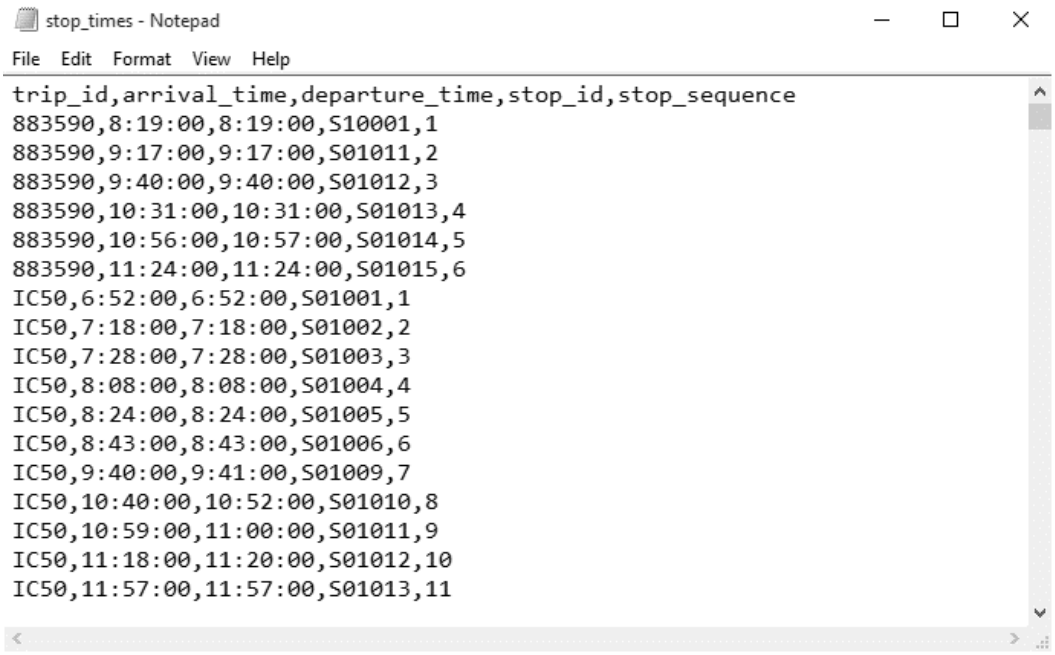
Παρουσιάζουμε ένα μέρος των αρχείων που δημιουργήσαμε και κάνουμε μια συνοπτική περιγραφή τους. Το πρότυπο GTFS αποτελείται από τουλάχιστον 6 αρχεία κειμένου .txt. Το πρότυπο δίνει πολλές δυνατότητες για παραπάνω αρχεία και παραπάνω πεδία αλλά δεν θα ασχοληθούμε στην παρούσα εργασία. Τα 6 βασικά αρχεία παρουσιάζονται στην παρακάτω εικόνα.



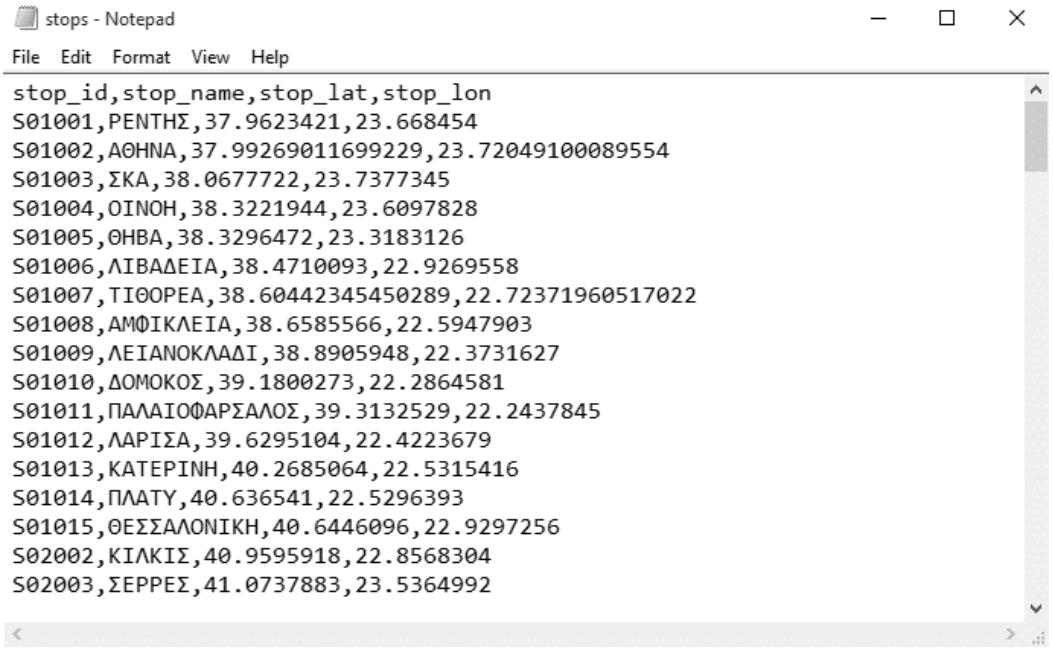
Name	Date modified	Type	Size
agency	7/20/2015 12:45 AM	Text Document	1 KB
calendar	7/20/2015 8:40 PM	Text Document	1 KB
routes	7/20/2015 1:16 AM	Text Document	1 KB
stop_times	7/21/2015 10:34 AM	Text Document	14 KB
stops	7/21/2015 1:12 PM	Text Document	7 KB
trips	7/31/2015 8:39 PM	Text Document	7 KB

Εικόνα 5.4 Τα 6 αρχεία .csv, στα οποία συγκεντρώσαμε τα δεδομένα του χρονοδιαγράμματος του Ο.Σ.Ε.

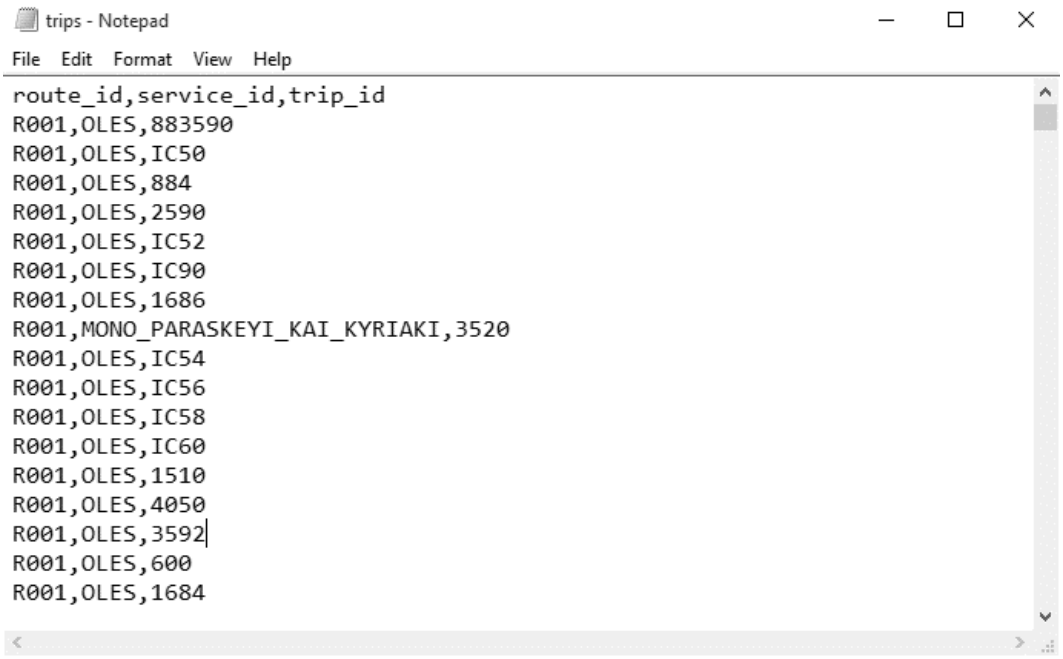
Η δομή των δεδομένων είναι ίδια με αυτή των αρχείων .csv [Comma Separated Values]. Κάθε αρχείο αποτελείται από τη γραμμή κεφαλίδα που καθορίζει την πλειάδα γνωρισμάτων που περιλαμβάνει το κάθε αρχείο τα οποία διαχωρίζονται μεταξύ τους με κόμμα. Στην υλοποίησή μας θα χρησιμοποιήσουμε μόνο τα 4 από τα 6 αρχεία δεδομένων: *stops.txt, stop_times.txt, trips.txt* και *calendar.txt.* Παραθέτουμε τις Εικόνες 5.5, 5.6, 5.7 και 5.8, για την κατανόηση της μορφής των δεδομένων. Στην πρώτη γραμμή παρουσιάζονται τα ονόματα των πεδίων και ακολουθούν οι πρώτες εγγραφές.



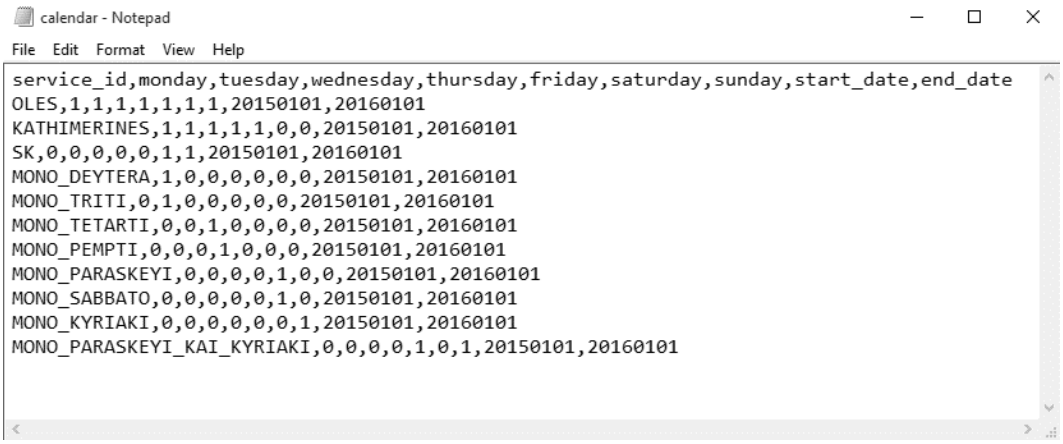
Εικόνα 5.5 Αρχείο stop_times.csv.



Εικόνα 5.6 Αρχείο stops.csv.



Εικόνα 5.7 Αρχείο trips.csv.



Εικόνα 5.8 Αρχείο calendar.csv.

Το πρότυπο επεκτείνεται ώστε να είναι σε θέση να περιλαμβάνει οποιαδήποτε λεπτομέρεια προκύψει. Για περισσότερες λεπτομέρειες σχετικά με τους κανόνες και τις δυνατότητες του προτύπου αλλά και για ειδικά προγραμματιστικά εργαλεία δημιουργίας και μετατροπής παραπέμπουμε τον αναγνώστη στη σελίδα:

https://developers.google.com/transit/gtfs/

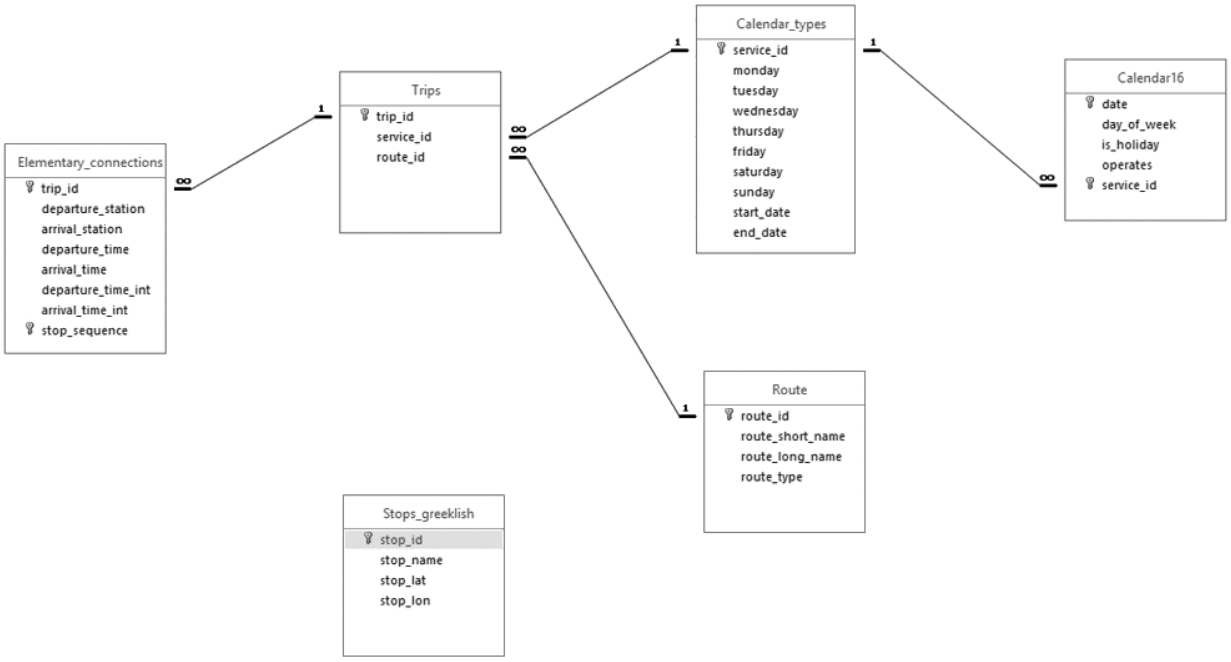
Όσο οι προσπάθειες υλοποίησης της εφαρμογής μας βελτιώνονταν έγινε αντιληπτό πως μια τέτοια προσέγγιση είναι λανθασμένη. Παρότι υπάρχουν εξειδικευμένα προγράμματα για την επεξεργασία και την τροποποίηση των δεδομένων GTFS, είναι λάθος να τα έχουμε ανάγκη κάθε φορά που συμβαίνει μια τροποποίηση του χρονοδιαγράμματος. Χωρίς τα προγράμματα αυτά δε, είναι εξαιρετικά δύσκολη η χειροκίνητη ενημέρωση των csv αρχείων.

Ένα άλλο σημαντικό μειονέκτημα που έγινε αντιληπτό βρίσκεται κατά τη φόρτωση των δεδομένων στην εφαρμογή. Χρειαζόμαστε περισσότερη "ευφυΐα" στο πρόγραμμα μας, ώστε να είναι σε θέση να φορτώνει αυτόματα τα δεδομένα, και να μην έχει την ανάγκη ειδικών συναρτήσεων σάρωσης δεδομένων (οι οποίες κοστίζουν υπολογιστικά).

Έτσι προχωρήσαμε στη σχεδίαση και ανάπτυξη μιας σχεσιακής βάσης δεδομένων, ώστε να μπορούμε να:

- έχουμε συγκεντρωμένα σε ένα αρχείο όλα μας τα δεδομένα
- έχουμε δομημένα όλα μας τα δεδομένα
- μην περιοριζόμαστε σε θέματα μεγέθους, μιας και μια βάση μπορεί να επεκταθεί όσο χρειαζόμαστε
- έχουμε δυνατότητα να μπορούμε εύκολα να ενημερώνουμε τα δεδομένα χρονοδιαγράμματος, χρησιμοποιώντας για παράδειγμα μια ειδικά σχεδιασμένη φόρμα
- έχουμε τη δυνατότητα, χρησιμοποιώντας τη γλώσσα SQL, να φέρουμε τα δεδομένα σε οποιαδήποτε μορφή θέλουμε μέσα στην εφαρμογή μας
- έχουμε τη δυνατότητα, χρησιμοποιώντας βιβλιοθήκες, να συνδέουμε αυτόματα τη βάση με την εφαρμογή μας
- έχουμε τη δυνατότητα αποθήκευσης της βάσης μας σε κάποια cloud πλατφόρμα ή σε κάποιον server

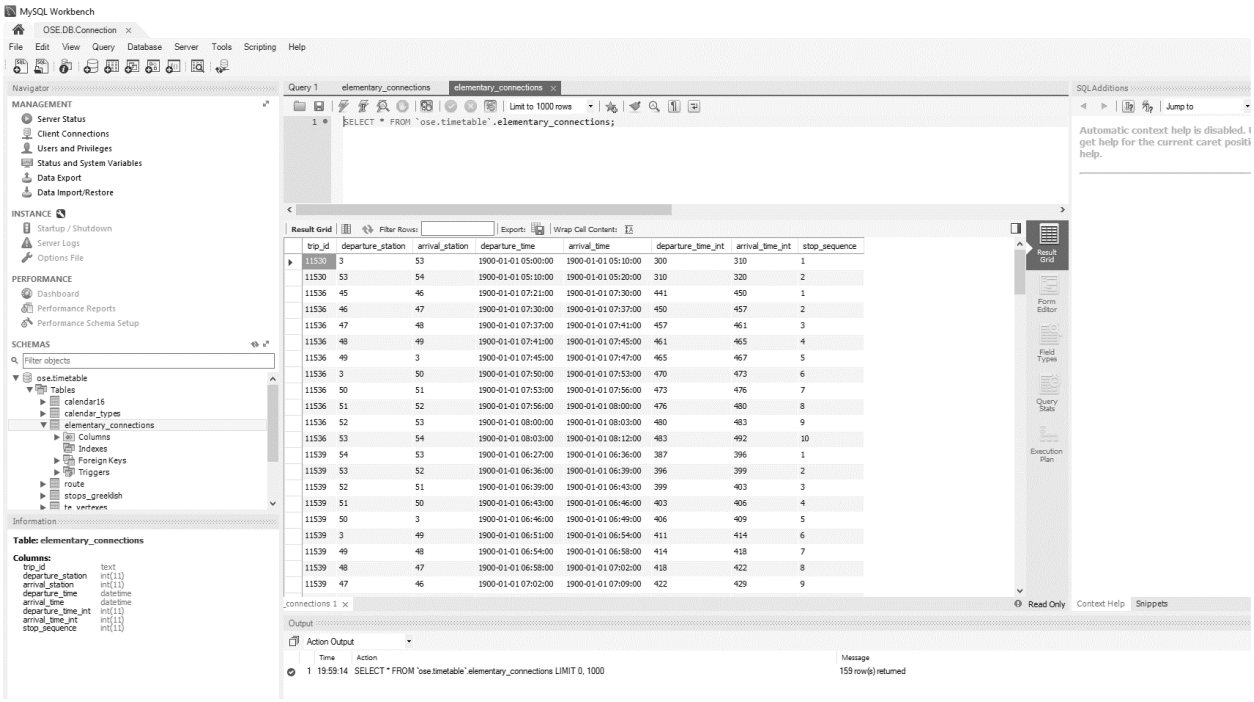
Με βάση λοιπόν αυτές τις ιδέες σχεδιάσαμε αρχικά ένα απλό σχεσιακό μοντέλο και έπειτα το υλοποιήσαμε στην Access για να δοκιμάσουμε. Το μοντέλο παρουσιάζεται στο παρακάτω σχήμα. Η μετάβαση ήταν απλή, μιας και η μορφή GTFS αποτελεί πρακτικά ένα στιγμιότυπο μιας σχεσιακής βάσης δεδομένων.



Εικόνα 5.8 Το σχεσιακό μοντέλο που κατασκευάσαμε περνώντας τα αρχεία GTFS σε Access DB.

Μόλις βεβαιωθήκαμε για τη λειτουργικότητα της βάσης, προχωρήσαμε στη μεταφορά της από την Access σε μια MySQL και την τοποθέτησή της στην cloud πλατφόρμα Microsoft Azure, σε προσωπικό λογαριασμό Microsoft, με φοιτητική άδεια χρήσης. Η Microsoft προσφέρει παγκόσμια δωρεάν φοιτητικές άδειες για τέτοιους σκοπούς. Επεξεργαζόμαστε τη βάση μας χρησιμοποιώντας το πρόγραμμα MySQL Workbench της Oracle. Να σημειώσουμε πως η MySQL είναι ένα πρόγραμμα διαχείρισης σχεσιακών βάσεων δεδομένων ελεύθερης άδειας, παρότι έχει αγοραστεί από την Oracle.

Έτσι, ανεξαρτητοποιηθήκαμε από διάφορους περιορισμούς που προέκυπταν, τους οποίους δεν έχει νόημα να αναφέρουμε εδώ, και καταλήξαμε στη μορφή που παρουσιάζεται στην παρακάτω εικόνα.



Εικόνα 5.9 Στιγμιότυπο κατά την επεξεργασία της online βάσης δεδομένων MySQL χρησιμοποιώντας το MySQL Workbench.

Κατά τη συλλογή και δόμηση αυτών των δεδομένων, δεν επικεντρωθήκαμε τόσο στην εγκυρότητά τους όσο στην εξέλιξή τους, ώστε να φτάσουν σε αυτήν τη μορφή. Για παράδειγμα, επιλέξαμε να μη συμπεριλάβουμε τον προαστιακό Πάτρας, μιας και δεν θα μας χρησίμευε στις δοκιμές των εφαρμογών μας, καθώς είναι απομονωμένος από το δίκτυο. Επίσης δεν προσθέσαμε τα διεθνή δρομολόγια. Με άλλα λόγια το χρονοδιάγραμμα που περιέχει αυτή η βάση είναι βέβαιο ότι έχει παραλήψεις, αλλά είναι περισσότερο από αρκετό για να δοκιμάσουμε επάνω του κάθε εφαρμογή μας.

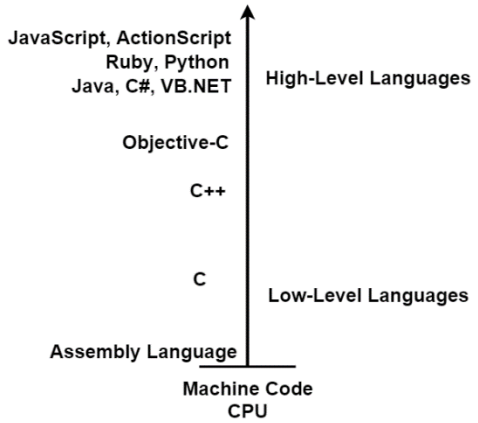
5.2. Επιλογή Γλώσσας Προγραμματισμού

Το επόμενο βήμα είναι η επιλογή μιας γλώσσας προγραμματισμού με την οποία θα προσπαθήσουμε να υλοποιήσουμε τη θεωρία που περιγράψαμε στα προηγούμενα κεφάλαια.

Το πρόβλημα της επιλογής γλώσσας προγραμματισμού είναι εξαιρετικά πολύπλοκο, ειδικότερα για την περίπτωσή μας, στην οποία μας ενδιαφέρουν κυρίως οι χρόνοι απόκρισης του συστήματος που θα προκύψει.

Έγινε από τις πρώτες μέρες αντιληπτό ότι πρέπει να ασχοληθούμε με αντικειμενοστραφείς γλώσσες προγραμματισμού για τους λόγους που περιγράψαμε στην αρχή της παρούσας εργασίας. Καθώς δεν είχαμε κάποια άλλη εμπειρία με αντικειμενοστραφείς γλώσσες προγραμματισμού, έκανα μια συνοπτική έρευνα στα βασικά χαρακτηριστικά των πιο δημοφιλών γλωσσών προγραμματισμού σήμερα.

Το βασικότερο χαρακτηριστικό κάθε γλώσσας είναι το πόσο κοντά βρίσκεται στη γλώσσα μηχανής (close to the metal). Το διπλανό σχήμα μας παρουσιάζει τις πιο δημοφιλείς γλώσσες προγραμματισμού, καθώς και την "απόσταση" της καθεμιάς από τη γλώσσα μηχανής. Ο βασικός κανόνας είναι πως όσο πιο πολύ πλησιάζει κανείς τη γλώσσα μηχανής, τόσο πιο δύσκολη γίνεται μια γλώσσα προγραμματισμού (καθώς απομακρύνεται από τη γλώσσα των ανθρώπων), αλλά και τόσο καλύτερα και αμεσότερα μπορεί να κατευθύνει τη μηχανή.



Καθώς η C δεν είναι αντικειμενοστραφής γλώσσα, επιλέξαμε αρχικά να πειραματιστούμε με τη C++, η οποία αρχικά ονομαζόταν "C με κλάσεις", και ήταν όπως φανερώνει και το αρχικό της όνομα μια αντικειμενοστραφής επέκταση της C. Χρησιμοποιώντας τη C++, παρότι θεωρείται μια από τις δυσκολότερες γλώσσες, θα μπορούσαμε να πετύχουμε μια καλή υλοποίηση και να καταφέρουμε καλούς χρόνους τρεξίματος. Εξάλλου, τα πειράματα της βιβλιογραφίας που παρουσιάσαμε είναι όλα γραμμένα με τη συγκεκριμένη γλώσσα.

Πράγματι, καταφέραμε να αναπτύξουμε μερικά αρχικά προγράμματα κονσόλας [console applications] χρησιμοποιώντας (στην πραγματικότητα εξερευνώντας) την C++, και τα αποτελέσματα ήταν εξαιρετικά ενθαρρυντικά. Ο κώδικας των προγραμμάτων αυτών ήταν εξαιρετικά κακογραμμένος. Περιείχαν πολλές επαναληπτικές διαδικασίες και θα μπορούσαμε να τα χαρακτηρίσουμε ως αυτοσχέδια. Παρόλα αυτά οι χρόνοι τρεξίματος πάνω σε μερικά δείγματα δεδομένων εισαγωγής ήταν της τάξεως των χιλιοστών του δευτερολέπτου.

Το επόμενο βήμα ήταν η αναζήτηση των εργαλείων με τα οποία αντιμετωπίζονται τα προβλήματα αυτού του τύπου. Όπως έχουμε ήδη αναφέρει το πρόβλημα πληροφοριών χρονοδιαγράμματος ανήκει στη μεγάλη οικογένεια των προβλημάτων της Θεωρίας Γράφων. Λόγω της φύσεως της C++, χρησιμοποιείται όχι από εμπορικούς αλλά και από ερευνητικούς και ακαδημαϊκούς οργανισμούς. Πάνω λοιπόν στη C++, έχουν δημιουργηθεί βιβλιοθήκες εργαλείων

που αφορούν στα προβλήματα της θεωρίας γράφων. Οι βασικότερες και πληρέστερες βιβλιοθήκες είναι οι εξής:

- Boost Graph Library
- LEDA
- Stanford
- LEMON
- SNAP

Μετά από πολλούς πειραματισμούς με αυτά τα εργαλεία, έγινε αντιληπτή η δυσχρηστία της συγκεκριμένης γλώσσας σε θέματα όπως: δύσκολη ανάγνωση, προσθήκη περιβάλλοντος χρήστη [user interface], επικοινωνία με βάσεις δεδομένων, διαχείριση μνήμης [memory management] και διάφορα άλλα.

Μετά από διάφορα στάδια αποφασίσαμε να απομακρυνθούμε κι' άλλο από τη γλώσσα μηχανής (μιας και γνωρίζαμε ότι υπήρχε αυτή η δυνατότητα με βάση τους χρόνους τρεξίματος που είχαμε ήδη) και να μεταφερθούμε σε μια σύγχρονη γλώσσα προγραμματισμού. Επιλέξαμε να συνεχίσουμε τους πειραματισμούς μας στην C#.

Η C# είναι και αυτή προφανώς μια C-like language που παρουσιάστηκε το 2000, την οποία ανέπτυξε η Microsoft και το βασικότερο χαρακτηριστικό της είναι η βιβλιοθήκη της. Η πραγματική γλώσσα είναι ελάχιστη σε έκταση ώστε να μαθαίνεται εύκολα. Η βιβλιοθήκη .NET framework, με την οποία σχετίζεται η C# (χωρίς να σημαίνει ότι είναι διαθέσιμη μόνο για τη συγκεκριμένη γλώσσα) είναι μια από τις μεγαλύτερες βιβλιοθήκες προγραμματισμού. Περιέχει χιλιάδες έτοιμες κλάσεις γραμμένες από τη Microsoft, οι οποίες είναι βέλτιστα υλοποιημένες.

Με βάση λοιπόν τη λογική του να μην "ανακαλύψουμε τον τροχό" και του "όσο πιο καλός είναι ένας προγραμματιστής τόσο λιγότερο κώδικα γράφει" οδηγηθήκαμε στη C# και στο .NET framework. Η βιβλιοθήκη αυτή περιέχει πάνω από 13000 κλάσεις. Υπάρχει δηλαδή ήδη έτοιμος κώδικας για μια σειρά από προβλήματα που έχει να αντιμετωπίσει κανείς κατά την ανάπτυξη ενός πληροφοριακού συστήματος. Υπάρχουν έτοιμες κλάσεις για επικοινωνία με οποιαδήποτε βάση δεδομένων, για σύνδεση του προγράμματος με συστήματα ERP όπως το SAP, για ανάπτυξη desktop εφαρμογών, για ανάπτυξη universal εφαρμογών οι οποίες θα τρέχουν σε οποιαδήποτε συσκευή (desktop, κινητό, tablet) τρέχει windows, για εφαρμογές σε άλλα λειτουργικά συστήματα όπως IOS, Android κτλ. Επίσης η Microsoft προέκτεινε τη βιβλιοθήκη αυτή με την ASP.NET, η οποία περιέχει όλα τα ειδικά εργαλεία για να κατασκευάσει κανείς web applications. Όλα αυτά χρησιμοποιώντας μια μόνο γλώσσα. Για την ακρίβεια όποια γλώσσα θέλουμε. Χρησιμοποιώντας το Microsoft Visual Studio, το περιβάλλον ανάπτυξης προγραμμάτων της Microsoft, ο προγραμματιστής έχει τη δυνατότητα να γράψει σε οποιαδήποτε γλώσσα θέλει και να χρησιμοποιήσει και πάλι όλα αυτά τα εργαλεία. Δεν πρόκειται να αναλύσουμε περεταίρω αυτές τις δυνατότητες καθώς πίσω από αυτές κρύβεται μια ολόκληρη φιλοσοφία.

Παρόλα αυτά στο framework αυτό δεν περιέχεται κλάση γράφου. Με άλλα λόγια η Microsoft έχει επιλέξει, τουλάχιστον προς το παρόν, να μην υλοποιήσει κάποια γενική δομή

δεδομένων γράφου. Αυτό είναι ένα απλό δείγμα της περιπλοκότητας της εφαρμογής που μελετάμε στην παρούσα εργασία.

Υπάρχει βέβαια μια τεράστια ενεργή κοινότητα προγραμματιστών και έχουν αναπτυχθεί πολλές βιβλιοθήκες. Εμείς ασχοληθήκαμε με τη βασικότερη από αυτές, την QuickGraph. Η βιβλιοθήκη αυτή έχει δομές γράφων που καλύπτουν πολλά σενάρια και κατά τη γνώμη μας αξίζει να μελετήσει κανείς σε βάθος. Αποτελεί μια μεταφορά της C++ Boost Graph Library στη C#. Να σημειωθεί πως η βιβλιοθήκη Boost Library θεωρείται μια από τις βασικότερες καθώς περιέχει εξαιρετικές υλοποιήσεις, πολλές από τις οποίες συμπεριλαμβάνονται κατά καιρούς στα πρότυπα ISO C++.

Χρησιμοποιώντας λοιπόν τη C# με το .NET framework και την QuickGraph, έχει κανείς όλα τα εργαλεία που χρειάζεται για να κατασκευάσει ένα πραγματικά σύγχρονο σύστημα πληροφοριών χρονοδιαγράμματος το οποίο να μπορεί να παρέχει στον χρήστη (σε οποιαδήποτε συσκευή) την πληροφορία που χρειάζεται σε μόλις μερικά χιλιοστά του δευτερολέπτου.

5.3. Περιβάλλον Υλοποίησης

Έπειτα από τη θεωρητική μελέτη του προβλήματος και την οργάνωση των δεδομένων μας προβήκαμε σε κάποιες προσπάθειες υλοποίησης. Στόχος μας ήταν τόσο η αποσαφήνιση κάποιων δύσκολων σημείων της θεωρίας στην πράξη, όσο και η λήψη των πραγματικών αποτελεσμάτων που αφορούν στους χρόνους τρεξίματος των μοντέλων πάνω στο χρονοδιάγραμμα του Ο.Σ.Ε.

Στις επόμενες ενότητες θα περιγράψουμε τις πιο επιτυχημένες υλοποιήσεις μας. Η ποιότητά τους βελτιωνόταν παράλληλα με την εξέλιξη της παρούσας εργασίας και με την απόκτηση οικειότητας των εργαλείων και των τεχνικών προγραμματισμού. Πιστεύουμε πως οι υλοποιήσεις αυτές είναι παρουσιάσιμες και πως θα βοηθήσουν σε μεγάλο βαθμό κάποιον που θα ασχοληθεί με το συγκεκριμένο ή με παρόμοιο πρόβλημα στο μέλλον.

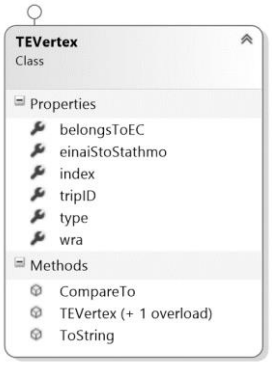
Όλες οι υλοποιήσεις έγιναν σε έναν desktop υπολογιστή με:

- Intel Core i5 3350P 3.1GHz CPU
- 8 GB DDR3 1600MHz RAM
- Windows 10 Education x64
- Microsoft Visual Studio Community 2015 Version 14.0.2572.00 Update 1

Προτιμήσαμε να ασχοληθούμε με την κατασκευή console και desktop εφαρμογών, για να αποφύγουμε τις τεχνικές ιδιαιτερότητες ενός web application, οι οποίες αφορούν σε θέματα επικοινωνίας server-client. Με την επιλογή αυτή εστιάσαμε μόνο στη δημιουργία της ίδιας της εφαρμογής, η οποία έπειτα μπορεί να πάρει εύκολα τη μορφή ενός web application με κατάλληλες μετατροπές. Στη συνέχεια θα περιγράψουμε συνοπτικά τη δομή του κώδικα της κάθε υλοποίησης, θεωρώντας πως ο αναγνώστης έχει γνώσεις προγραμματισμού με χρήση της γλώσσας C# και του .NET framework.

5.4. Χρονικά-Επεκτεινόμενες Υλοποιήσεις

Όπως αναλύσαμε στα προηγούμενα κεφάλαια, στη χρονικά-επεκτεινόμενη προσέγγιση μοντελοποιούμε γεγονότα. Κατασκευάσαμε την κλάση TEVertex με την οποία θα αναπαριστούμε τα γεγονότα αυτά ως κόμβους στον χρονικά-επεκτεινόμενο γράφο.



Στη διπλανή εικόνα παρουσιάζεται το Unified Modeling Language (U.M.L.) σχήμα της κλάσης TEVertex. Η κλάση TEVertex είναι μια ειδίκευση της γενικής στοιχειώδους σύνδεσης. Όπως περιγράψαμε στην Παράγραφο 3.1.1, μια στοιχειώδης σύνδεση είναι μια πλειάδα πέντε τιμών της μορφής $c=(Z, S_1, S_2, t_d, t_a)$. Για το χρονικά-επεκτεινόμενο μοντέλο χωρίζουμε την κάθε στοιχειώδη σύνδεση σε δύο TEVertex. Δηλαδή, ξεχωρίζουμε τα δύο διαφορετικά γεγονότα της κάθε στοιχειώδους σύνδεσης c και τα συμβολίζουμε με τη μεταβλητή type, η οποία παίρνει την τιμή 1 αν είναι γεγονός αναχώρησης και 2 αν είναι άφιξης.

Η παραπάνω μετατροπή των στοιχειωδών συνδέσεων σε ένα σύνολο TEVertex είναι στατική. Αυτό σημαίνει πως έχει γίνει ήδη ή γίνεται αυτόματα από τη βάση δεδομένων. Η συγκεκριμένη μετατροπή μπορεί να γίνει χρησιμοποιώντας SQL είτε αποθηκεύοντας τα δεδομένα σε έναν νέο πίνακα, είτε κρατώντας τα ως ένα αποτέλεσμα query και καλώντας το αυτό για εισαγωγή στο πρόγραμμα. Γενικότερα, λόγω της στατικής φύσης των δεδομένων στις εφαρμογές μας, μπορούμε να υποθέτουμε με ασφάλεια πως τα δεδομένα εισέρχονται στις υλοποιήσεις μας σε οποιαδήποτε μορφή μας βολεύει.

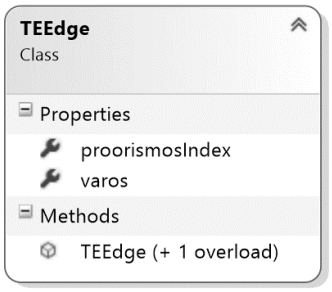
Αφού ορίσαμε τους κόμβους, μπορούμε να περάσουμε στην εισαγωγή των δεδομένων στην εφαρμογή. Επικοινωνούμε με τη βάση δεδομένων και της ζητούμε να μας επιστρέψει τις πληροφορίες για να γεμίσουμε τις τρεις βασικές λίστες:

- List<string> stopsList
- List<string> tripsList
- List<TEVertex> TEVertexesList

Έπειτα διακόπτουμε τη σύνδεση με τη βάση μιας και έχουμε όλα τα απαραίτητα δεδομένα στη RAM μας.

Επόμενο βήμα είναι κατασκευή του χρονικά-επεκτεινόμενου γράφου. Για να γίνει η κατασκευή αυτή πρέπει πρώτα να ομαδοποιήσουμε τα γεγονότα σύμφωνα με τον σταθμό στον οποίον συμβαίνουν. Ο λόγος για τον οποίον συμβαίνει αυτό θα γίνει αντιληπτός στη συνέχεια. Κατασκευάζουμε έτσι πρώτα τον List<List<TEVertex>> TEpreGraph, ο οποίος είναι ένας δισδιάστατος πίνακας με 102 γραμμές, όσοι δηλαδή και οι σταθμοί του δικτύου, όπου κάθε γραμμή περιέχει τα διάφορα γεγονότα TEVertex που συμβαίνουν στον συγκεκριμένο σταθμό. Στην συνέχει ταξινομούμε τα γεγονότα της κάθε γραμμής σύμφωνα με την ώρα τους, από τη μικρότερη προς τη μεγαλύτερη.

Έπειτα περνάμε στην κατασκευή του κανονικού χρονικά-επεκτεινόμενου γράφου List<List<TEEdge>> TEGraph. Ο γράφος αυτός έχει τη μορφή λίστας γειτνίασης, όπως περιγράψαμε στην Παράγραφο 2.4.10. Η κλάση TEEdge, περιέχει μόνο 2 ιδιότητες: proorismosIndex, varos. Ο λόγος για τον οποίον χρησιμοποιούμε την TEEdge είναι για να ανεξαρτητοποιήσουμε τον γράφο από όλα τα "περιττά" (προς το παρών) δεδομένα, όπως για παράδειγμα το όνομα του τρένου tripID. Για τον γράφο το κάθε γεγονός TEVertex είναι τώρα απλά ένας ακέραιος.



Στη συνέχεια γεμίζουμε τον γράφο TEGraph προσθέτοντας ακμές TEEdge. Γνωρίζουμε εξ αρχής τον αριθμό των γραμμών του γράφου, ο οποίος θα είναι ίσος με τον αριθμό των γεγονότων TEVertex, και συγκεκριμένα για το απλοποιημένο μοντέλο στο οποίο όλες οι συνδέσεις του χρονοδιαγράμματος συμβαίνουν μέσα σε μια ημέρα, ίσος με 7676. Η διαδικασία γεμίσματος συμβαίνει μέσα σε έναν επαναληπτικό βρόχο for-loop, στον οποίον εξετάζουμε σε κάθε επανάληψη κάθε ένα από τα 7676 γεγονότα και υλοποιούμε την εξής λογική:

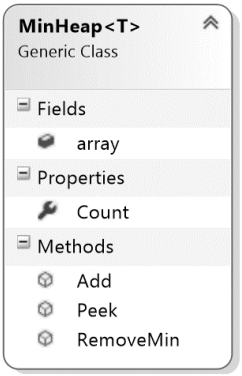
- Αν το γεγονός είναι γεγονός αναχώρησης, βάλε δίπλα του μια ακμή με τον κατάλληλο γεγονός άφιξης (δηλαδή προσθέτουμε τις train-edges).
- Αν το γεγονός είναι γεγονός άφιξης, ανέτρεξε στον TEpreGraph και πρόσθεσε δίπλα του μια ακμή για κάθε γεγονός αναχώρησης το οποίο συμβαίνει μετά από το εν λόγω γεγονός άφιξης. Εδώ γίνεται αντιληπτή η χρησιμότητα του TEpreGraph καθώς χρειαζόμαστε τα γεγονότα στον συγκεκριμένο σταθμό ταξινομημένα ως προς τον χρόνο. Η ανάγκη αυτή δεν είναι τόσο εμφανής όταν υλοποιούμε το απλοποιημένο μοντέλο, μιας και θα μπορούσαμε να εξετάζουμε σε κάθε επανάληψη και τα 7676 γεγονότα, αλλά όταν προχωρούμε σε εβδομαδιαία ή και ετήσια χρονική-επέκταση, όπου ο αριθμός των γεγονότων γίνεται πολύ μεγάλος.

Με τον παραπάνω τρόπο προκύπτει ένας σχετικά μικρός γράφος, μιας και αποτελείται από δείκτες γεγονότων αντί των ίδιων των γεγονότων, μαζί με τις ετικέτες απόστασης.

Στο σημείο αυτό πρέπει να αποσαφηνίσουμε πως όλη η υλοποίηση μέχρι στιγμής είναι και αυτή στατική, όπως και τα δεδομένα χρονοδιαγράμματος. Δηλαδή η κατασκευή του γράφου μπορεί να γίνει μια φορά και να αποθηκευτεί κατάλληλα, ώστε κατά την εισαγωγή των δεδομένων να εισέρχεται και ο έτοιμος χρονικά-επεκτεινόμενος γράφος.

Εκτός από έναν κατάλληλα διαμορφωμένο γράφο, χρειαζόμαστε επίσης έναν κατάλληλα προσαρμοσμένο αλγόριθμο του Dijkstra, ο οποίος χρειάζεται με τη σειρά του και μια ουρά προτεραιότητας.

Το .NET framework δεν διαθέτει δομή δεδομένων ουράς προτεραιότητας. Χρησιμοποιήσαμε μια δική μας υλοποίηση ουράς, με το όνομα MinHeap, προεκτείνοντας τη δομή δεδομένων List με τις αναγκαίες μεθόδους και τη λογική της ουράς προτεραιότητας. Το UML της κλάσης αυτής παρουσιάζεται στη διπλανή εικόνα. Αξίζει να αναφέρουμε τις ιδιαίτερες μεθόδους που προσθέσαμε:



- Η μέθοδος Add προσθέτει ένα στοιχείο μέσα στην ουρά και αν χρειάζεται επαναταξινομεί τα στοιχεία μέσα της ώστε να βρίσκεται στην κορυφή αυτό με τη μικρότερη τιμή.
- Η μέθοδος Peek επιστέφει το πρώτο στοιχείο, χωρίς να το αφαιρέσει.
- Η μέθοδος RemoveMin αφαιρεί το πρώτο στοιχείο, που είναι το μικρότερο.

Πριν περάσουμε στην υλοποίηση του αλγορίθμου του Dijkstra, παίρνουμε από τον χρήστη τα 3 δεδομένα εισαγωγής του βασικού επερωτήματος, δηλαδή: σταθμός αναχώρησης s_1, σταθμός προορισμού s_2 και νωρίτερος χρόνος αναχώρησης t. Εδώ πρέπει να αναφέρουμε μια ιδιαιτερότητα του χρονικά-επεκτεινόμενου μοντέλου που αντιμετωπίσαμε, η οποία δεν παρουσιάζεται στους επίσημους ορισμούς του. Η ιδιαιτερότητα είναι πως ο γράφος, στην παρούσα στιγμή, δεν περιέχει κόμβο εκκίνησης και κόμβο προορισμού. Δηλαδή, αφού ο χρονικά-επεκτεινόμενος γράφος αποτελείται από γεγονότα άφιξης και αναχώρησης τρένων, δεν περιέχει εξ ορισμού γεγονότα εκκίνησης και τερματισμού. Με απλά λόγια προέκυψε το ερώτημα "από πού ξεκινάω και πού τελειώνω". Παρακάτω περιγράφουμε τη λανθασμένη μοντελοποίηση ώστε να αποφευχθούν τέτοια λάθη.

Αρχικά προσπαθήσαμε να αποφύγουμε την εισαγωγή ενός νέου γεγονότος στον χρονικά-επεκτεινόμενο γράφο, γιατί θα έπρεπε να μπει ανάμεσα στα ήδη υπάρχοντα γεγονότα και να προστεθούν και όλες οι αναγκαίες stay-edges, οπότε θα ήταν υπολογιστικά χρονοβόρο. Επίσης, θα έπρεπε να αφαιρεθούν μετά τον υπολογισμό της συντομότερης διαδρομής, ώστε ο γράφος να είναι έτοιμος να τρέξει ένα νέο επερώτημα. Έτσι εξετάσαμε αν μπορούμε να κάνουμε τον αλγόριθμο να βρει το κατάλληλο γεγονός εκκίνησης. Ανατρέξαμε στον TEpreGraph, που έχει ταξινομημένα τα γεγονότα για κάθε σταθμό, στη γραμμή που αντιπροσωπεύει τον σταθμό s_1 και επιλέξαμε το 1ο γεγονός άφιξης στον s_1 πριν τον χρόνο t. Με άλλα λόγια προσπαθήσαμε να μοντελοποιήσουμε το γεγονός εκκίνησης σα να φτάσαμε μόλις στον s_1 με κάποιο άλλο τρένο. Η προσέγγιση αυτή δεν είναι έγκυρη, γιατί υπάρχει η πιθανότητα να υπάρχει γεγονός αναχώρησης ανάμεσα στο γεγονός που επιλέχθηκε και στον χρόνο t, με αποτέλεσμα να δίνεται η δυνατότητα στον αλγόριθμο να υπολογίσει βέλτιστη διαδρομή με χρόνο εκκίνησης (λίγο) πριν τον t. Ως γεγονός τερματισμού μπορούμε να θεωρήσουμε το τελευταίο γεγονός αναχώρησης (δηλαδή πάλι το ανάποδο) στον σταθμό προορισμού s_2. Έτσι, η συντομότερη διαδρομή που θα προέκυπτε θα είχε απλά κάποιες περιττές stay-edges στο τέλος της (τις οποίες θα αποκρύβαμε από τον χρήστη), μιας και αφού θα είχε ήδη φτάσει στον s_2 θα περίμενε μέχρι το γεγονός τερματισμού.

Αντί για την παραπάνω περίπλοκη προσέγγιση καταλήξαμε σε μια πιο σαφή και εύχρηστη. Για τον σταθμό εκκίνησης προσθέτουμε στον γράφο TEGraph μια ακόμα γραμμή στο τέλος, μιας και προσθέτουμε ένα ακόμα γεγονός. Το γεγονός αυτό είναι το γεγονός εκκίνησης και συμβαίνει στον χρόνο t. Δίπλα του προσθέτουμε τις απαιτούμενες TEEdges. Δηλαδή προσθέτουμε τις stay-edges χρησιμοποιώντας τον TEpreGraph σαν το γεγονός εκκίνησης να ήταν γεγονός άφιξης. Με άλλα λόγια, ο αλγόριθμος του Dijkstra στον χρονικά-επεκτεινόμενο γράφο θα ξεκινάει πάντα από

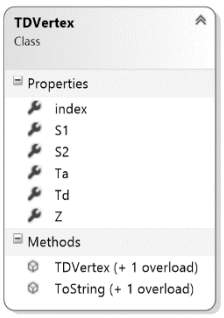
την τελευταία γραμμή, στην οποία κάθε φορά θα προσθέτουμε τις αναγκαίες ακμές πριν εκκινήσουμε και θα την καθαρίζουμε αφού τερματίσουμε. Όσον αφορά το γεγονός τερματισμού δεν το μοντελοποιούμε ρητά. Αντ' αυτού ορίζουμε μια ακεραία μεταβλητή με το όνομα targetVertexIndex, και τη στιγμή που ο αλγόριθμος του Dijkstra πάει να τερματιστεί γιατί έφτασε στο s_2, εκχωρούμε στην targetVertexIndex το index αυτού του τελευταίου γεγονότος άφιξης. Η προσέγγιση αυτή εγγυάται σωστά αποτελέσματα.

Τέλος, υλοποιούμε τον αλγόριθμο του Dijkstra με τη χρήση της ουράς MinHeap πάνω στον χρονικά-επεκτεινόμενο γράφο TEGraph όπως αναλύθηκε στην Παράγραφο 2.3.1. χωρίς κάποια ιδιαιτερότητα, με σταθμό εκκίνησης και συνθήκη τερματισμού όπως περιγράψαμε παραπάνω.

Η περαιτέρω χρονική-επέκταση του μοντέλου, ώστε να δουλεύει με εβδομαδιαίους ή και ετήσιους γράφους είναι απλή. Τα μόνα σημεία διαφοροποίησης είναι:

- στις τιμές των ωρών οι οποίες θα περιέχουν μέσα τους και την ημερομηνία ως πολλαπλάσιο του 1440, όπως περιγράψαμε στην Παράγραφο 2.8.1,
- και στην προσθήκη ενός ακόμα κριτηρίου κατά την εισαγωγή stay-edges, έτσι ώστε να μην προσθέτουμε stay-edges διάρκειας παραπάνω τις μιας μέρας.

5.5. Χρονικά-Εξαρτώμενες Υλοποιήσεις



Παρουσιάζουμε την υλοποίηση της χρονικά-εξαρτώμενης προσέγγισης. Αρχικά κατασκευάσαμε την κλάση TDVertex, η οποία παρουσιάζεται στο διπλανό σχήμα UML. Η κλάση αυτή ταυτίζεται με τον ορισμό της στοιχειώδους σύνδεσης, αλλά την ονομάσαμε έτσι για λόγους σαφήνειας. Στη συνέχεια μπορούμε να εισάγουμε τα δεδομένα από τη βάση δεδομένων στην εφαρμογή μας και να γεμίζουμε τις 3 λίστες όπως και στη χρονικά-επεκτεινόμενη υλοποίηση.

Έπειτα περνάμε στην κατασκευή του χρονικά-εξαρτώμενου γράφου, μιας και δεν χρειαζόμαστε κάποιο είδους προ-γράφου. Η υλοποίηση του TDGraph στην οποία προβήκαμε έχει ως εξής. Κατασκευάζουμε μια 3-διάστατη λίστα μεγέθους 102x102x0, όπου 102 είναι ο αριθμός των σταθμών του δικτύου μας. Πιο απλά κατασκευάζουμε έναν 2-διάστατο πίνακα, όπου κάθε θέση του περιέχει μια λίστα. Προσθέτουμε διαδοχικά κάθε κόμβο TDVertex στη λίστα που βρίσκεται στη θέση $[s_1][s_2]$ του 2-διάστατου πίνακα, έτσι ώστε να ομαδοποιήσουμε όλες τις συνδέσεις από τον κόμβο x στον y στη λίστα TDGraph[x][y]. Η διαγώνιος αυτού του πίνακα είναι προφανώς κενή. Επίσης ο πίνακας αυτός προκύπτει πολύ αραιός, γεγονός όμως που δεν μας επηρεάζει καθώς έχουμε απευθείας πρόσβαση στις λίστες που χρειαζόμαστε όπως περιγράψαμε.

Μπορούμε να φανταστούμε πως η τρίτη διάσταση του γράφου αυτού αναπαριστά τον χρόνο. Ενώ στη χρονικά-επεκτεινόμενη προσέγγιση είχαμε προβάλει αυτή τη διάσταση στις δύο διαστάσεις, με αποτέλεσμα να πολλαπλασιαστούν οι κόμβοι, στη χρονικά-εξαρτώμενη προσέγγιση διατηρούμε αυτή τη διάσταση και τη χρησιμοποιούμε ως δυνατά "ζεύγη τιμών" (t_a, t_d) ανάμεσα σε δύο σταθμούς (x, y).

Στη συνέχεια παίρνουμε από τον χρήστη τα δεδομένα του επερωτήματος. Σε αντίθεση με τη χρονικά-επεκτεινόμενη προσέγγιση, ο σταθμός εκκίνησης και προορισμού ταυτίζεται με τα δεδομένα εισαγωγής s_1 και s_2 αντίστοιχα.

Χρησιμοποιώντας την ίδια δομή ουράς προτεραιότητας MinHeap υλοποιούμε την εκδοχή του αλγορίθμου του Dijkstra για την χρονικά-εξαρτώμενη προσέγγιση. Λόγω της περιπλοκότητας του συγκεκριμένου αλγορίθμου, θεωρούμε αναγκαίο να παραθέσουμε τον ψευδοκώδικά του, ώστε να είμαστε σε θέση μετά να σχολιάσουμε τις ιδιαιτερότητές του. Έχουμε λοιπόν τον εξής αλγόριθμο:

Αλγόριθμος του Dijkstra για το χρονικά-εξαρτώμενο μοντέλο

```
1:  Θέτουμε τις ετικέτες των κόμβων d[v] := ∞ για κάθε κόμβο v (δηλαδή για κάθε σταθμό).
2:  Θέτουμε την ετικέτα του d[s] := t, όπως περιγράφει και η θεωρία.
3:  Δημιουργούμε ένα κενό σύνολο S στο οποίο θα αποθηκεύουμε του χαλαρωμένους κόμβους.
4:  Προσθέτουμε στην ουρά προτεραιότητας το ζεύγος τιμών {(s, d[s])}.
5:  Όσο η ουρά δεν είναι κενή:
6:     Αφαίρεσε από την ουρά το ζεύγος (u, t'') με την ελάχιστη ετικέτα απόστασης.
7:     Πρόσθεσε το u στο σύνολο S.
8:     Αν u = d διάκοψε την επανάληψη (συνθήκη τερματισμού).
9:     Για κάθε (u, v) τέτοιο ώστε το u να μην ανήκει ήδη στο S:
10:       Θέσε στο t' το αποτέλεσμα της συνάρτησης f_uv(d[u]).
11:       Αν d[v] > t' τότε:
12:          Αν d[v] = ∞ τότε:
13:             Πρόσθεσε στην ουρά το ζεύγος τιμών (v, t').
14:          Αλλιώς:
15:             Ενημέρωσε το ήδη υπάρχον ζεύγος τιμών (v,t').
16:          Τέλος αν
17:          Θέσε στο d[v] την τιμή t'.
18:          Θέσε στο parent[v] το u .
19:       Τέλος αν
20:    Τέλος για κάθε
21:Τέλος όσο
```

Στη συνέχεια του αλγορίθμου ακολουθεί μια προς-τα-πίσω επαναληπτική διαδικασία ώστε να οδηγηθούμε από τον σταθμό προορισμού στον σταθμό αναχώρησης μέσω του πίνακα parent[v].

Η συνάρτηση $f_{uv}(t)$ ορίζεται ως εξής: Ανάμεσα σε όλα τα ζεύγη χρόνων (t', t'') που ορίζουν οι TDVertex στη λίστα TDGraph[u][v], διάλεξε το ελάχιστο t'' έτσι ώστε να ισχύει και $t \leq t''$ παράλληλα. Η συνάρτηση αυτή είναι μια συνάρτηση ελαχιστοποίησης, την οποία υλοποιήσαμε ελέγχοντας κάθε TDVertex της λίστας TDGraph[u][v].

5.6. Χρόνοι Τρεξίματος

Τόσο στις χρονικά-επεκτεινόμενες, όσο και στις χρονικά-εξαρτώμενες απλοποιημένες υλοποιήσεις στις οποίες προβήκαμε, τα αποτελέσματα ήταν εξαιρετικά ενθαρρυντικά.

Πιο συγκεκριμένα, στο απλοποιημένο χρονικά-επεκτεινόμενο μοντέλο, ακόμα και για περίπλοκα επερωτήματα με τρεις και τέσσερις μετεπιβιβάσεις, οι χρόνοι τρεξίματος ήταν μόλις μερικές δεκάδες milliseconds. Η ίδια συμπεριφορά διατηρήθηκε και όταν υλοποιήσαμε τον εβδομαδιαίο χρονικά-επεκτεινόμενο γράφο, με κάποια άφιξη της τάξεως των 10ms. Ακόμα μικρότεροι ήταν οι χρόνοι τρεξίματος του χρονικά-εξαρτώμενου μοντέλου, όπως ήταν αναμενόμενο και από τη θεωρία, οι οποίοι ακόμα και για περίπλοκες διαδρομές δεν ξεπερνούσαν τα 15ms.

Να σημειώσουμε πως οι χρόνοι αυτοί επιτεύχθηκαν με τις δικές μας υλοποιήσεις, οι οποίες σαφώς και δεν είναι οι βέλτιστες. Για παράδειγμα, η ουρά προτεραιότητας που χρησιμοποιήσαμε στους αλγορίθμους του Dijkstra ταξινομεί τα στοιχεία του σε άσχημο χρόνο ίσο με $O(n^2)$. Παρόλα αυτά, ο αριθμός των στοιχείων είναι τόσο μικρός στην εφαρμογή μας που οι διαφορές ανάμεσα στις κακές και καλές υλοποιήσεις εκφυλίζονται.

Θεωρούμε πως η χρήση των βέλτιστων υλοποιήσεων, με τη βοήθεια μιας βιβλιοθήκης όπως η QuickGraph, και η προσαρμογή της στη δική μας εφαρμογή θα εξαλείψει πρακτικά τους χρόνους τρεξίματος ακόμα και στα περίπλοκα πολυκριτηριακά μοντέλα που περιγράψαμε κατά την διάρκεια της παρούσας εργασίας.

Όλα αυτά τα θετικά αποτελέσματα μας βεβαιώνουν πως είναι δυνατή η κατασκευή ενός σύγχρονου υπολογιστικού συστήματος πληροφοριών χρονοδιαγράμματος με πολύ περισσότερες δυνατότητες από την εύρεση συντομότερης διαδρομής.

Τέλος, να σημειώσουμε πως όλες οι υλοποιήσεις μας περιλαμβάνονται σε ηλεκτρονική μορφή μαζί με την παρούσα εργασία ως αρχεία .sln (και όχι απλά .exe), ώστε να υπάρχει η πλήρης δυνατότητα μελέτης και τρεξίματος του κώδικα με τη χρήση του Visual Studio.

5.7. Επιλογή Προσέγγισης Μοντελοποίησης

Μετά από όλες τις προσπάθειες υλοποίησης που επιχειρήσαμε, θεωρούμε πως για το δίκτυο και τις ανάγκες του Ο.Σ.Ε. ταιριάζει καλύτερα η χρονικά-επεκτεινόμενη προσέγγιση. Οι λόγοι που μας έκαναν να καταλήξουμε σε αυτήν την επιλογή είναι οι εξής:

- Από τη μελέτη των πειραματικών αποτελεσμάτων της βιβλιογραφίας είδαμε πως για το απλοποιημένο πρόβλημα η χρονικά-εξαρτώμενη προσέγγιση είναι σαφώς ανώτερη της χρονικά-επεκτεινόμενης. Όταν όμως προεκτάθηκαν τα μοντέλα για να συμπεριλάβουν τις ρεαλιστικές παραμέτρους η διαφορά έγινε εξαιρετικά μικρή. Μιας και ο Ο.Σ.Ε. έχει ανάγκη από όσο το δυνατόν πιο ρεαλιστικό σύστημα πληροφοριών οι διαφορές των μοντέλων γίνονται πολύ μικρές.
- Λόγω του μεγέθους του ελληνικού σιδηροδρομικού δικτύου, αναιρείται πρακτικά το μεγαλύτερο μειονέκτημα του χρονικά-επεκτεινόμενου μοντέλου έναντι του χρονικά-εξαρτώμενου, το οποίο είναι το μεγάλο μέγεθος του γράφου που προκύπτει. Όπως αναλύσαμε και στην προηγούμενη ενότητα, προβήκαμε σε κατασκευή πολλών διαφορετικών γράφων ώστε να βεβαιωθούμε πως το μέγεθός τους είναι διαχειρίσιμο. Καταλήξαμε στο συμπέρασμα πως έχουμε τη δυνατότητα κατασκευής ενός ετήσιου (μεγαλύτερος δεν έχει πρακτικό νόημα) πλήρως χρονικά-επεκτεινόμενου γράφου, ο οποίος δεν θα απαιτεί παραπάνω από 4GB μνήμης, μιας και ο εβδομαδιαίος απαιτεί περίπου 62MB.
- Ένα μεγάλο πλεονέκτημα του χρονικά-επεκτεινόμενου μοντέλου είναι η ευκολία προέκτασής του. Είδαμε στην πράξη πως τις περισσότερες φορές το μοντέλο είναι έτοιμο να προεκταθεί ώστε να αντιμετωπίσει μια ρεαλιστική λεπτομέρεια ή κάποιο άλλο επερώτημα, χωρίς να απαιτεί ιδιαίτερες τροποποιήσεις στη δομή του γράφου.
- Το χρονικά-επεκτεινόμενο μοντέλο υπερτερεί στα θέματα σαφήνειας. Ο ορισμός του είναι απλούστερος από αυτόν του χρονικά-εξαρτώμενου και αυτό θα είναι ένα μεγάλο πλεονέκτημα κατά την περίοδο ανάπτυξης αλλά και εντοπισμού σφαλμάτων του συστήματος. Με άλλα λόγια είναι σχεδόν βέβαιο πως η ανάπτυξη του συστήματος με αυτό το μοντέλο θα χρειαστεί πολύ λιγότερο χρόνο (και γενικότερα πόρους) από ότι με το χρονικά-εξαρτώμενο μοντέλο.

Για όλους τους παραπάνω λόγους θεωρούμε πως στην περίπτωση του συστήματος πληροφοριών για το χρονοδιάγραμμα του Ο.Σ.Ε. ταιριάζει καλύτερα η ρεαλιστική χρονικά-επεκτεινόμενη προσέγγιση.

6. Συμπεράσματα

Κατά τη γνώμη μας η παρούσα διπλωματική εργασία παρουσίασε το σύνολο των αναγκαίων γνώσεων για την αντιμετώπιση των προβλημάτων πληροφοριών χρονοδιαγράμματος. Στοχεύαμε εξ αρχής στην συγκέντρωση όλου του θεωρητικού υπόβαθρου για την πλήρη κατανόηση του προβλήματος και την επίλυσή του, ώστε η εργασία μας να λάβει χαρακτήρα εγχειριδίου για κάποιον που θα ασχοληθεί με το συγκεκριμένο (ή με παρόμοιο) πρόβλημα στο μέλλον.

Εκτός από τις θεωρητικές αναλύσεις, προχωρήσαμε και σε πραγματικές υλοποιήσεις, ώστε να είμαστε βέβαιοι πως τα συγκεκριμένα μοντέλα είναι εφαρμόσιμα στην περίπτωσή του Ο.Σ.Ε., και πως αποφέρουν τα απαιτούμενα αποτελέσματα. Είναι σημαντικό το ότι έχουμε στα χέρια μας κάποια πρόχειρα πειραματικά αποτελέσματα υλοποιήσεων πάνω στο χρονοδιάγραμμα του Ο.Σ.Ε. Πάνω σε αυτά θα βασιστεί η επιλογή προσέγγισης, καθώς και οι τεχνικές επιτάχυνσης της, για την δημιουργία ενός ολοκληρωμένου και σύγχρονου πληροφοριακού συστήματος.

Ακόμα περισσότερα ήταν τα προσωπικά οφέλη που αποκομίσαμε από την παρούσα εργασία. Κατά τη διάρκεια της μελέτης μας δόθηκε η δυνατότητα να ασχοληθούμε σε βάθος με τον τομέα του προγραμματισμού και της μηχανικής αλγορίθμων. Παρότι το συγκεκριμένο πρόβλημα μας είχε φανεί εύκολο με μια πρώτη ματιά, αποδείχθηκε εξαιρετικά πολύπλοκο. Είναι δύσκολο να αντιληφθεί κανείς ποιο πρόβλημα είναι δύσκολο για έναν υπολογιστή, μιας και είναι εντελώς διαφορετικό από αυτά που χαρακτηρίζουν οι άνθρωποι ως δύσκολα προβλήματα. Μια άλλη λανθασμένη αντίληψη που είχαμε είναι πως οι άνθρωποι, όπως και οι υπολογιστές, χρησιμοποιούν αλγορίθμους για να λύσουν προβλήματα. Κατά την μελέτη μας μάθαμε πως αυτό συμβαίνει μόνο σε λίγες περιπτώσεις και συγκεκριμένα μόνο στις πιο απλές. Στην πραγματικότητα οι άνθρωποι δεν χρησιμοποιούμε ποτέ αλγορίθμους για να λύσουμε τα πιο εξεζητημένα προβλήματα που προκύπτουν, χρησιμοποιούμε ευρετικά. Το πιο συνηθισμένο από τα ευρετικά αυτά, τα οποία δεν εγγυόνται βέλτιστες λύσεις, είναι η μέθοδος δοκιμής-σφάλματος [trial and error]. Αυτό συμβαίνει συχνά χωρίς καν να το αντιληφθούμε. Με την παρούσα εργασία καταλάβαμε την σπουδαιότητα μιας εγγυημένα βέλτιστης λύσης καθώς και τη θεωρητική ανάλυση και υπολογιστική δύναμη που απαιτεί.

Ένα άλλο σημαντικό θέμα με το οποίο καταφέραμε να αποκτήσουμε μια αρχική σχέση είναι αυτό του αντικειμενοστραφούς σχεδιασμού. Μπορέσαμε να εμπεδώσουμε τα πλεονεκτήματα που προσφέρει αυτή η προσέγγιση σε σχέση με αυτή του δομημένου προγραμματισμού. Κατανοήσαμε δηλαδή γιατί κυριαρχεί αυτή η προσέγγιση παγκόσμια τα τελευταία 30 χρόνια.

Επίσης ασχοληθήκαμε με τις C-like γλώσσες προγραμματισμού. Οι γλώσσες αυτές μοιράζονται το ίδιο (σχεδόν) συντακτικό, πράγμα που σημαίνει ότι αν μάθει κάνεις μια από αυτές θα μπορεί τουλάχιστον να διαβάσει και να καταλάβει τη γενική εικόνα μιας εφαρμογής γραμμένης σε οποιαδήποτε από τις άλλες. Το συντακτικό της C έχει επικρατήσει γενικά στον κόσμο του προγραμματισμού και χαρακτηρίζεται ως εύχρηστο και σαφές σε σχέση με κάποιων άλλων γλωσσών. Μαθαίνοντας τα βασικά από αυτές τις γλώσσες είμαστε σε θέση να ασχοληθούμε με

οποιαδήποτε εφαρμογή είναι γραμμένη σε μια C-like γλώσσα προγραμματισμού, όπως: Java, C#, C++, Objective-C, JavaScript. Με άλλα λόγια αποκτήσαμε μια ικανότητα, την οποία είναι βέβαιο πως θα χρησιμοποιήσουμε στο μέλλον.

Γνωρίζαμε ήδη από την διάρκεια των σπουδών μας ότι οι σύγχρονες εφαρμογές των επιχειρήσεων απαιτούν την οργάνωση των δεδομένων τους σε μια βάση δεδομένων. Το πρόγραμμα σπουδών μας αφιέρωνε μάλιστα ένα ολόκληρο μάθημα εξαμήνου στη μελέτη και τον σχεδιασμό των μοντέλων οντοτήτων-συσχετίσεων και τη μετατροπή τους σε σχεσιακές βάσεις δεδομένων. Παρόλα αυτά δεν είχαμε καταλάβει τα πλεονεκτήματα που προσφέρει αυτή η οργάνωση των δεδομένων. Μόνο μέσα από την εφαρμογή αυτών των ιδεών μπορεί κανείς να κατανοήσει γιατί είναι τόσο πολύτιμα τα δεδομένα και η αυστηρή οργάνωσή τους σε μια εφαρμογή. Αυτός είναι και ο λόγος για τον οποίον υπάρχουν σήμερα επιτυχημένες εταιρίες οι οποίες πρακτικά το μόνο που προσφέρουν είναι η πρόσβαση μέσω μιας εφαρμογής (συνήθως διαδικτυακής) στη βάση δεδομένων τους.

Όσον αφορά στη διαδικασία εκπόνησης μιας μελέτης πραγματικού κόσμου, πιστεύουμε πως γνωρίσαμε όλα τα στάδιά της και εκπληρώσαμε τον βαθύτερο σκοπό της ανάθεσης διπλωματικής εργασίας από το πολυτεχνείο. Θέλουμε να πιστεύουμε πως είμαστε προετοιμασμένοι να αντιμετωπίσουμε το επόμενο τεχνολογικό πρόβλημα που θα παρουσιαστεί, αυτή τη φορά όμως έξω από τα ακαδημαϊκά πλαίσια. Αυτή είναι εξάλλου και η φύση του μηχανικού: να λύνει προβλήματα.

Βιβλιογραφία

- Daniel Delling, Thomas Pajor, and Dorothea Wagner. *Engineering Time-Expanded Graphs for Faster Timetable Information.* Future and Emerging Technologies Unit of EC (IST priority – 6th FP). 2005.
- Edsger W. Dijkstra. *A note on two problems in connexion with graphs.* Numerische Mathematik, 1959.
- Andrew V. Goldberg. *A simple shortest path algorithm with linear average time.* In Proceedigns of the 9th European Symposium on Algorithms (ESA 2001), volume 2161 of LNCS, Springer, 2001.
- M. S. Hung and J. Divoky. *A computational study of effcient shortest path algorithms.* Computers and Operations Research, 1988.
- Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. *Efficient Models for Timetable Information in Public Transportation Systems*. ACM Journal of Experimental Algorithmics, Vol. 12, Article No. 2.4. 2008.
- Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. *Using multilevel graphs for timetable information in railway systems*. In Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX 2002), volume 2409 of LNCS, pages 43{59. Springer, 2002.
- Mikkel Thorup. *Undirected single source shortest paths with positive integer weights in linear time*. Journal of the ACM, 1999.
- Matthias Muller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. *Timetable Information: Models and Algorithms*, Springer. 2007.
- R. Jakob, M. Marathe, and K. Nagel. *A computational study of routing algorithms for realistic transportation networks*. The ACM Journal of Experimental Algorithmics, 4, 1999.
- A. Orda and R. Rom. *Minimum weight paths in time-dependent networks.* Networks, 21, 1991.
- Schulz Frank. *Timetable Information and Shortest Paths*. PhD thesis, Karlsruhe Institute of Technology, 2005.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill, 2001.
- Thomas Willhalm and Dorothea Wagner. *Shortest path speedup techniques*. In Algorithmic Methods for Railway Optimization, LNCS. Springer. 2005.

ΗΛΙΑΔΗΣ Κ. ΠΕΤΡΟΣ
Διπλωματούχος Μηχανολόγος Μηχανικός Ε.Μ.Π.