Τετραγωνικό Πρόβλημα Ανάθεσης Ένας νέος αλγόριθμος επίλυσης

Θεόδωρος Γκεβεζές

Προσεγγιστικός Αλγόριθμος για την επίλυση του Τετραγωνικού Προβλήματος Ανάθεσης

Quadratic Assignment Problem - QAP
Τετραγωνικό Πρόβλημα Ανάθεσης

Διπλωματική εργασία:

Θεόδωρος Πασχάλη Γκεβεζές

Μεταπτυχιακό Πρόγραμμα Σπουδών «Θεωρητική Πληροφορική και Θεωρία Συστημάτων Ελέγχου»
Τμήμα Μαθηματικών
Σχολή Θετικών Επιστημών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Επιβλέπων καθηγητής:

Λεωνίδας Πιτσούλης

Επίκουρος Καθηγητής
Γενικό Τμήμα Φυσικών και Μαθηματικών Επιστημών
Πολυτεχνική Σχολή
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τριμελής εξεταστική επιτροπή:

Λεωνίδας Πιτσούλης

Νικόλαος Καραμπετάκης

Επίκουρος Καθηγητής
Τμήμα Μαθηματικών
Σχολή Θετικών Επιστημών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Γεώργιος Ραχώνης

Λέκτορας
Τμήμα Μαθηματικών
Σχολή Θετικών Επιστημών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Θεσσαλονίκη 2008

Περιεχόμενα

Ευχαριστώ τον κύριο Λεωνίδα Πιτσούλη, που μοιράστηκε κάποιες από τις γνώσεις του με μένα.

τον κύριο Νικόλαο Καραμπετάκη για την πολύτιμη βοήθειά του, ως υπεύθυνος για μένα καθηγητής.

τον κύριο Συμεών Μποζαπαλίδη, που με έπεισε να μπω στο μεταπτυχιακό πρόγραμμα σπουδών.

όλους τους καθηγητές του μεταπτυχιακού προγράμματος σπουδών για την πολύ καλή προσπάθειά τους και το σημαντικό έργο που προσφέρουν.

τη Ζωή μου για την υπομονή της.

Θεόδωρος Πασχάλη Γκεβεζές

The Quadratic Assignment Problem
Το Τετραγωνικό Πρόβλημα Ανάθεσης

1. Εισαγωγή

Το **Τετραγωνικό Πρόβλημα Ανάθεσης** (**Quadratic Assignment Problem – QAP**) είναι ένα από τα πιο γνωστά **προβλήματα συνδυαστικής βελτιστοποίησης** (**combinatorial optimization problems**). Δεδομένων του συνόλου $N = \{1,2,...,n\}$ και τριών δισδιάστατων πινάκων $F = (f_{ij})$, $D = (d_{kl})$ και $B = (b_{ik})$ μεγέθους $n \times n$ με πραγματικά στοιχεία, το Τετραγωνικό Πρόβλημα Ανάθεσης μπορεί να ορισθεί ως

$$\min_{p \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)} + \sum_{i=1}^{n} b_{ip(i)} \qquad (1)$$

όπου το S_n είναι το σύνολο όλων των μεταθέσεων (permutations) $p : N \to N$.

Μια από τις πιο σημαντικές και συνηθισμένες εφαρμογές του QAP είναι η τοποθέτηση ενός συνόλου εγκαταστάσεων σε ένα σύνολο τοποθεσιών, έτσι ώστε η τελική διάταξη να είναι όσο το δυνατόν πιο λειτουργική. Η αφηρημένη έννοια της λειτουργικότητας μπορεί να οριστεί επακριβώς από μια αντικειμενική συνάρτηση, η οποία θα υπολογίζει το κόστος της κάθε διάταξης. Το ολικό κόστος είναι μια συνάρτηση της ροής μεταξύ των εγκαταστάσεων, της απόστασης μεταξύ των τοποθεσιών και της δαπάνης τοποθέτησης κάθε εγκατάστασης σε μια συγκεκριμένη τοποθεσία. Το πρόβλημα απεικονίζεται με δύο γράφους, που έχουν πίνακες γειτνίασης τους F και D, αντίστοιχα.

Ο αντικειμενικός σκοπός είναι η τοποθέτηση (ανάθεση) κάθε εγκατάστασης σε μια τοποθεσία, έτσι ώστε το συνολικό τελικό κόστος ανάθεσης να ελαχιστοποιείται. Πρακτικά, αυτό σημαίνει ότι οι μεγάλες ροές θα αντιστοιχούν σε όσο το δυνατόν μικρότερες αποστάσεις και οι μεγάλες αποστάσεις θα αντιστοιχούν σε όσο το δυνατόν μικρότερες ροές, ενώ επιπλέον η κάθε εγκατάσταση θα τοποθετηθεί σε μια θέση με όσο το δυνατόν μικρότερο κόστος τοποθέτησης.

Το πρόβλημα αυτό μπορεί πολύ εύκολα να εκφραστεί ως ένα Τετραγωνικό Πρόβλημα Ανάθεσης, αν γίνουν οι παρακάτω αντιστοιχίες:

- οι ροές (flows) μεταξύ των εγκαταστάσεων αποτελούν τα στοιχεία του πίνακα $F = (f_{ij})$, με f_{ij} να είναι η τιμή της ροής από την εγκατάσταση i στην εγκατάσταση j
- οι αποστάσεις (distances) μεταξύ των τοποθεσιών αποτελούν τα στοιχεία του πίνακα $D = (d_{kl})$, με d_{kl} να είναι η απόσταση από την τοποθεσία k στην τοποθεσία l
- τα κόστη (costs) τοποθέτησης της κάθε εγκατάστασης σε συγκεκριμένη τοποθεσία αποτελούν τα στοιχεία του πίνακα $B = (b_{ik})$, με b_{ik} να είναι το κόστος τοποθέτησης της εγκατάστασης i στην τοποθεσία k
- το S_n περιέχει όλους τους δυνατούς συνδυασμούς αντιστοίχησης των εγκαταστάσεων σε τοποθεσίες
- η συνάρτηση (1) αποτελεί την περιγραφή του προβλήματος
- το πλήθος των εγκαταστάσεων και των τοποθεσιών είναι το μέγεθος (n) του προβλήματος

Το κάθε γινόμενο $f_{ij}d_{kl}$, ανεξάρτητο από τα υπόλοιπα, αντιπροσωπεύει το κόστος της ταυτόχρονης ανάθεσης της εγκατάστασης i στην τοποθεσία k και της εγκατάστασης j στην τοποθεσία l. Το QAP στα πλαίσια της θεωρίας τοποθέτησης (location theory), όπως περιγράφηκε παραπάνω για την τοποθέτηση n εγκαταστάσεων σε n τοποθεσίες, περιλαμβάνει πίνακες συμμετρικούς, μη αρνητικούς με μηδενικά στην κύρια διαγώνιο.

Κάθε στιγμιότυπο του Τετραγωνικού Προβλήματος Ανάθεσης με πίνακες εισόδου F, D και B μπορεί να δηλωθεί ως $QAP(F,D,B)$, ενώ αν δεν υπάρχει ο γραμμικός όρος B $(B=0)$, τότε μπορεί να δηλωθεί ως $QAP(F,D)$. Πολλές μέθοδοι επίλυσης του QAP που έχουν προταθεί, δεν υπολογίζουν το γραμμικό όρο B, αφού η παρουσία του στην αντικειμενική συνάρτηση δεν προσδίδει κάποια επιπλέον πληροφορία στη διατύπωση του προβλήματος και η επίλυσή του ως ανεξάρτητο πρόβλημα δεν παρουσιάζει καμιά δυσκολία.

Τις τελευταίες τέσσερις δεκαετίες έχει γίνει εκτεταμένη έρευνα, σχετική με το Τετραγωνικό Πρόβλημα Ανάθεσης, για την ανάπτυξη τόσο αλγορίθμων εύρεσης της βέλτιστης λύσης (exact algorithms), όσο και προσεγγιστικών αλγορίθμων (approximation algorithms), που αποσκοπούν στην εύρεση μιας λύσης κοντά στη βέλτιστη (αν όχι της ίδιας της βέλτιστης) σε μικρό υπολογιστικό χρόνο. Παρόλη τη μεγάλη προσπάθεια, τόσο σε θεωρητικό όσο και σε πρακτικό (προγραμματιστικό) επίπεδο, το QAP παραμένει ένα από τα πιο δύσκολα και «σκληρά» υπολογιστικά προβλήματα βελτιστοποίησης και κανένας αλγόριθμος εύρεσης της βέλτιστης λύσης δεν μπορεί να λύση στιγμιότυπα του προβλήματος με μέγεθος μεγαλύτερο από 30 (n > 30).

Αντίθετα, για το ανάλογο γραμμικό πρόβλημα, το Γραμμικό Πρόβλημα Ανάθεσης (Linear Assignment Problem – LAP), έχει αναπτυχθεί ένας αλγόριθμος, που βρίσκει τη βέλτιστη λύση σε πολυωνυμικό υπολογιστικό χρόνο. Ο αλγόριθμος ονομάζεται **Hungarian algorithm** και αναπτύχθηκε από τον James Munkres [78] το 1957. Βασίζεται σε μια χρονικά προηγούμενη εργασία του Harold Kuhn [68] (1955) με όπου περιγράφεται η **Hungarian method**. Ο αλγόριθμος βρίσκει τη βέλτιστη λύση του LAP σε πολυωνυμικό χρόνο στην τάξη του ($O(n^3)$).

Το 1976 οι Sartaj Sahni και Teofilo Gonzalez [93] έδειξαν ότι το QAP είναι NP-hard πρόβλημα και ως εκ τούτου, δεν είναι δυνατό να βρεθεί σε πολυωνυμικό χρόνο ούτε η ακριβής λύση του, ούτε και κάποια προσεγγιστική λύση, της οποίας η τιμή θα απέχει από την τιμή της βέλτιστης κατά ένα σταθερό παράγοντα. Τα αποτελέσματα αυτά ισχύουν, εκτός και αν $P=NP$. Ο Maurice Queyranne [87] έδειξε ότι τα παραπάνω αποτελέσματα ισχύουν ακόμα και στην περίπτωση που οι συντελεστές των πινάκων του προβλήματος ικανοποιούν την τριγωνική ανισότητα.

Το Τετραγωνικό Πρόβλημα Ανάθεσης (QAP) παρουσιάστηκε για πρώτη φορά από τους Tjalling Koopmans και Martin Beckmann [66] το 1957, στη μορφή που φαίνεται στη συνάρτηση (1). Το χρησιμοποίησαν ως ένα μαθηματικό μοντέλο για τον εντοπισμό ενός συνόλου αδιαίρετων οικονομικών εργασιών. Από τότε έχει χρησιμοποιηθεί για τη μοντελοποίηση και τη λύση αρκετών άλλων προβλημάτων από διάφορες περιοχές έρευνας. Εκτός από την εφαρμογή του στο πρόβλημα τοποθέτησης εγκαταστάσεων, που αναφέρθηκε παραπάνω, το QAP απαντάται σε εφαρμογές όπως τον προγραμματισμό και χρονοπρογραμματισμό ενεργειών (scheduling) [39], την κατασκευή υπολογιστών (computer manufacturing), την ανάπτυξη παράλληλων και κατανεμημένων υπολογιστικών συστημάτων (parallel and distributed computer systems) [11], την

στατιστική ανάλυση δεδομένων (statistical data analysis) [59], την διαβάθμιση αρχαιολογικών δεδομένων (ranking of archeological data) [67], την επικοινωνία διαδικασιών (process communications), τη διευθέτηση των καλωδίων σε ηλεκτρονικά ταμπλό (backboard wiring) [97], τη ζυγοστάθμιση τουρμπίνων (turbine balancing), τη σχεδίαση πανεπιστημιουπόλεων, κατασκηνώσεων και στρατοπέδων (campus and camp planning) [26, 30], τη σχεδίαση πληκτρολογίων δακτυλογράφησης (typewriter keyboard designs) [18], την ανάλυση χημικών αντιδράσεων (analyzing chemical reactions) [100] και αρκετές άλλες [34, 75, 51].

2. Διατυπώσεις (formulations)

Σε πολλά προβλήματα συνδυαστικής βελτιστοποίησης υπάρχει η δυνατότητα διαφορετικών, αλλά ισοδύναμων διατυπώσεων. Καθεμιά από τις διατυπώσεις αυτές δίνει έμφαση σε διαφορετικά δομικά χαρακτηριστικά του προβλήματος και μπορεί να οδηγήσει σε ξεχωριστές προσεγγίσεις και τεχνικές επίλυσής του.

Ο Eugene Lawler [70] εισήγαγε μια πιο γενική εκδοχή του Τετραγωνικού Προβλήματος Ανάθεσης σε σχέση με αυτήν που αρχικά ανέπτυξαν οι Koopmans και Beckmann. Ο Lawler χρησιμοποίησε έναν πίνακα $C = (c_{ijkl})$ τεσσάρων διαστάσεων αντί των δύο πινάκων $F = (f_{ij})$ και $D = (d_{kl})$ δύο διαστάσεων. Έτσι το QAP μπορεί να ορισθεί ως:

$$\min_{p \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijp(i)p(j)} + \sum_{i=1}^{n} b_{ip(i)} \tag{2}$$

Κάθε στιγμιότυπο $QAP(F, D, B)$ του προβλήματος, με τη μορφή που πρότειναν οι Koopmans και Beckmann, μπορεί να διατυπωθεί με τη μορφή του Lawler, αν θέσουμε $c_{ijkl} = f_{ij} d_{kl}$ για όλα τα i, j, k, l με $i \neq j$ ή $k \neq l$ και $c_{iikk} = f_{ii} d_{kk} + b_{ik}$ για όλα τα i, j, k, l με $i = j$ και $k = l$. Ο πίνακας $C = (c_{ijkl})$ έχει μέγεθος $n \times n \times n \times n$.

2.1 Τετραγωνικός Ακέραιος Προγραμματισμός (Quadratic Integer Programming)

Κάθε μετάθεση (permutation) $p : N \to N$ του συνόλου $N = \{1,2,...,n\}$ μπορεί να αναπαρασταθεί από ένα δισδιάστατο πίνακα $X = (x_{ij})$ μεγέθους $n \times n$, οι τιμές των στοιχείων του οποίου καθορίζονται ως εξής:

$$x_{ij} = \begin{cases} 1 \text{ αν } p(i) = j \\ 0 \text{ διαφορετικά} \end{cases} \tag{3}$$

Ο πίνακας $X = (x_{ij})$ ονομάζεται **πίνακας μετάθεσης** (**permutation matrix**). Το σύνολο όλων των πινάκων μετάθεσης μεγέθους n δηλώνεται ως X_n. Κάθε πίνακας μετάθεσης ικανοποιεί τους εξής περιορισμούς

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,...,n \tag{4}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,...,n \tag{5}$$

$$x_{ij} \in \{0,1\} \qquad i, j = 1,2,...,n \tag{6}$$

Ο περιορισμός της εξίσωσης (4) εξασφαλίζει την ύπαρξη ακριβώς ενός άσσου σε κάθε γραμμή του πίνακα μετάθεσης και στην περίπτωση του QAP ότι κάθε εγκατάσταση θα τοποθετηθεί σε μία ακριβώς θέση. Αντίστοιχα ο περιορισμός της εξίσωσης (5) εξασφαλίζει την ύπαρξη ακριβώς ενός άσσου σε κάθε στήλη του πίνακα μετάθεσης και στην περίπτωση του QAP ότι κάθε θέση θα καταληφθεί από μία ακριβώς εγκατάσταση.

Μπορεί να παρατηρηθεί ότι το στοιχείο $d_{p(i)p(j)}$ στον ορισμό (1) του QAP με τη χρήση πινάκων μετάθεσης γράφεται ως

$$d_{p(i)p(j)} = \sum_{k=1}^{n}\sum_{l=1}^{n} d_{kl} x_{ik} x_{jl}$$

Σύμφωνα με τις δύο αυτές παρατηρήσεις και χρησιμοποιώντας τους πίνακες μετάθεσης το QAP μπορεί να διατυπωθεί ως ένα πρόβλημα Ακέραιου Προγραμματισμού με τετραγωνική αντικειμενική συνάρτηση

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl} x_{ik} x_{jl} + \sum_{i,j=1}^{n} b_{ij} x_{ij} \tag{7}$$

έτσι ώστε

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,...,n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,...,n$$

$$x_{ij} \in \{0,1\} \qquad i, j = 1,2,...,n$$

Η τετραγωνική αντικειμενική συνάρτηση στη διατύπωση αυτή είναι και ο λόγος για τον οποίο το πρόβλημα χαρακτηρίστηκε ως «Τετραγωνικό» (“Quadratic”) από τους Koopmans και Beckmann [66]. Στη συνέχεια της εργασίας αυτής, όποτε αναφέρεται ότι $X = (x_{ij}) \in X_n$, θα συνεπάγεται ότι ο X είναι πίνακας μετάθεσης και ότι τα x_{ij} ικανοποιούν τους περιορισμούς ανάθεσης (4), (5) και (6).

2.2 Διατύπωση εσωτερικού γινομένου πινάκων

Το εσωτερικό γινόμενο (inner product) δύο πραγματικών, δισδιάστατων πινάκων $A = (a_{ij})$ και $B = (b_{ij})$ μεγέθους $n \times n$ ορίζεται από τη σχέση

$$\langle A, B \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{ij} \tag{8}$$

Με τη χρήση του εσωτερικού γινομένου πινάκων το QAP, όπως το παρουσίασαν οι Koopmans και Beckmann, μπορεί να γραφεί σε μια πιο συμπαγή μορφή.

Δεδομένων ενός πίνακα A μεγέθους $n \times n$, μιας μετάθεσης $p \in S_n$ και του αντίστοιχου πίνακα μετάθεσης $X \in X_n$

- το XA μεταθέτει τις γραμμές του πίνακα A σύμφωνα με τη μετάθεση p
- το AX^T μεταθέτει τις στήλες του πίνακα A σύμφωνα με τη μετάθεση p.

Έτσι ισχύει η ισότητα

$$XAX^T = (a_{p(i)p(j)}) \tag{9}$$

σύμφωνα με την οποία μπορούμε να διατυπώσουμε το πρόβλημα που περιγράφεται στη συνάρτηση (1) ως

$$\min \langle F, XDX^T \rangle + \langle B, X \rangle \tag{10}$$

υπό τον περιορισμό

$$X \in X_n$$

2.3 Διατύπωση Κοίλης Τετραγωνικής Αντικειμενικής Συνάρτησης (Concave Quadratic Formulation)

Στη διατύπωση του Τετραγωνικού Προγράμματος Ακεραίων, αλλά και στη εκδοχή του Lawler, χρησιμοποιήθηκαν οι συντελεστές c_{ijkl} στην αντικειμενική συνάρτηση. Οι συντελεστές αυτοί μπορούν να θεωρηθούν ως στοιχεία ενός δισδιάστατου πίνακα S μεγέθους $n^2 \times n^2$, έτσι ώστε ο συντελεστής c_{ijkl} να βρίσκεται στη γραμμή $(i-1)n + j$ και στη στήλη $(k-1)n + l$ του πίνακα S. Η αφαίρεση μιας σταθεράς από τα στοιχεία της κύριας διαγωνίου του πίνακα S δεν αλλάζει τη βέλτιστη λύση (μετάθεση) του αντίστοιχου QAP. Η αλλαγή αυτή απλά προσθέτει μια σταθερή τιμή στην αντικειμενική συνάρτηση που υπολογίζει την τιμή της λύσης.

Σύμφωνα με την παρατήρηση αυτή η επίλυση ενός στιγμιότυπου του QAP με πίνακα συντελεστών S ισοδυναμεί με την επίλυση ενός στιγμιότυπου του QAP με πίνακα συντελεστών $Q = S - aI$, όπου I είναι ο μοναδιαίος πίνακας (unit matrix) μεγέθους $n^2 \times n^2$ και η τιμή της σταθεράς a είναι μεγαλύτερη από τη νόρμα $\|S\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |s_{ij}|$ του πίνακα S.

Επιπλέον, έστω ότι το x είναι ένα διάνυσμα μεγέθους n με τα στοιχεία του να είναι οι γραμμές του πίνακα μετάθεσης X

$$x = ((x_{11}, x_{12}, ..., x_{1n}), ..., (x_{n1}, x_{n2}, ..., x_{nn}))^T = (x_1, ..., x_n)^T$$

Άρα ισχύει ότι

$$\begin{aligned} x^T Q x &= \sum_{i=1}^{n^2} q_{ii} x_i^{\,2} + 2 \sum_{i=1}^{n^2-1} \sum_{j=i+1}^{n^2} q_{ij} x_i x_j \\ &= \sum_{i=1}^{n^2} (q_{ii} + \sum_{\substack{j=1 \\ j \neq i}}^{n^2} q_{ij}) x_i^{\,2} - \sum_{i=1}^{n^2-1} \sum_{j=i+1}^{n^2} q_{ij} (x_i - x_j)^2 \\ &= \sum_{i=1}^{n^2} (-a + \sum_{j=1}^{n^2} s_{ij}) x_i^{\,2} - \sum_{i=1}^{n^2-1} \sum_{j=i+1}^{n^2} s_{ij} (x_i - x_j)^2 \\ &= \sum_{i=1}^{n^2} (-a + \sum_{j=1}^{n^2} s_{ij}) x_i^{\,2} \end{aligned}$$

Επίσης, επειδή ισχύει ότι

$$x^T \left[\frac{1}{2} \left(Q + Q^T \right) \right] x = \frac{1}{2} x^T Q x$$

μπορούμε να αξιώσουμε ότι ο πίνακας Q είναι συμμετρικός και αρνητικά ορισμένος.

Οι Mokhtar Barazaa και Hanif Sherali [9] βασίστηκαν στα παραπάνω για να εισάγουν μια νέα διατύπωση του QAP το 1982. Πρόκειται για ένα πρόβλημα ελαχιστοποίησης κοίλης, τετραγωνικής, αντικειμενικής συνάρτησης (quadratic concave minimization problem), που ορίζεται ως εξής:

$$\min x^T Q x \tag{11}$$

έτσι ώστε

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,...,n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,...,n$$

$$x_{ij} \geq 0 \qquad i, j = 1,2,...,n$$

Οι εφευρέτες της διατύπωσης αυτής την χρησιμοποίησαν ώστε να αναπτύξουν διαδικασίες τομής επιφανειών (cutting plane procedures). Οι μέθοδοι εύρεσης της βέλτιστης λύσης στιγμιότυπων του QAP, που βασίζονται στις διαδικασίες αυτές δεν είναι υπολογιστικά αποδοτικές. Αντίθετα, οι ευρηστικές μέθοδοι, που μπορούν να προκύψουν από αυτές τις διαδικασίες, παράγουν λύσεις πολύ καλής ποιότητας με τιμή κοντά σε αυτήν της βέλτιστης λύσης.

Αν στην παραπάνω διαδικασία, για τον ορισμό των συντελεστών του πίνακα Q, δεν αφαιρούνταν ο όρος aI, αλλά προσθέτονταν στον πίνακα S, τότε από την αντικειμενική συνάρτηση του αντίστοιχου QAP θα αφαιρούνταν μια σταθερή τιμή και το

πρόβλημα θα μπορούσε να διατυπωθεί ως ένα πρόβλημα ελαχιστοποίησης κυρτής, τετραγωνικής αντικειμενικής συνάρτησης (quadratic convex minimization problem).

2.4 Ίχνος πίνακα

Το ίχνος ενός δισδιάστατου πίνακα $A = (a_{ii})$ διαστάσεων $n \times n$ ορίζεται ως το άθροισμα των στοιχείων της κύριας διαγωνίου του.

$$trace(A) = a_{11} + a_{22} + ... + a_{nn} = \sum_{i=1}^{n} a_{ii} \tag{12}$$

Ιδιότητα γινομένου πινάκων: Έστω ότι έχουμε δύο πίνακες $A = (a_{ij})$ και $B = (b_{ij})$ μεγέθους $n \times n$. Τότε ισχύει

$$trace(AB) = \sum_{i=1}^{n} (AB)_{ii} = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} B_{ji} \tag{13}$$

Αν στη διατύπωση του QAP με τη βοήθεια του εσωτερικού γινομένου πινάκων και συγκεκριμένα στον τύπο (10) θέσουμε

$$\overline{D} = XD^T X^T$$

τότε

$$trace(F\,\overline{D}) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}\, \bar{d}_{ji} = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)}$$

αφού

$$\bar{d}_{ji} = d_{p(i)p(j)}$$

όπου $p \in S_n$ είναι η μετάθεση (permutation) που αντιστοιχεί μοναδικά στον πίνακα μετάθεσης (permutation matrix) $X \in X_n$ του τύπου (10).

Επίσης παρατηρούμε ότι ισχύει

$$trace(BX^T) = \sum_{i=1}^{n} b_{ip(i)} \tag{14}$$

Χρησιμοποιώντας όλα τα παραπάνω, ο Carol Edwards [28, 29] διατύπωσε το QAP με τη βοήθεια του ίχνους πίνακα ως

$$\min trace(FXD^T + B)X^T \tag{15}$$

έτσι ώστε

$$X \in X_n$$

Αν σε ένα στιγμιότυπο του QAP μόνο ένας από τους δύο πίνακες είναι συμμετρικός, ενώ ο άλλος όχι, τότε υπάρχει ένας μετασχηματισμός, ο οποίος μετατρέπει το QAP σε ένα ισοδύναμό του με συμμετρικούς και τους δύο πίνακες. Χωρίς βλάβη της γενικότητας, ας θεωρήσουμε ότι ο πίνακας F είναι συμμετρικός, ενώ ο D όχι.

Ιδιότητες ίχνους πίνακα:

$$trace(AB) = trace(BA)$$
$$trace(A) = trace(A^T)$$

Ιδιότητες ανάστροφων πινάκων:

$$(AB)^T = B^T A^T$$

Αφού ο πίνακας F είναι συμμετρικός, ισχύει $F = F^T$. Στην περιγραφή του QAP, μέσω του ίχνους πίνακα (15), ο τετραγωνικός όρος της αντικειμενικής συνάρτησης είναι ο $trace(FXD^T X^T)$. Σύμφωνα με τις παραπάνω ιδιότητες έχουμε:

$$trace(FXD^T X^T) =$$
$$trace((FXD^T X^T)^T) =$$
$$trace(XDX^T F^T) =$$
$$trace(FXDX^T)$$

Άρα ορίζοντας

$$E = \frac{1}{2}(D + D^T) \tag{16}$$

ισχύει

$$trace(FXE^T X^T) = \frac{1}{2} trace(FXD^T X^T + FXDX^T) = trace(FXD^T X^T) \tag{17}$$

Τα παραπάνω δείχνουν ότι για τη μετατροπή ενός QAP, στο οποίο μόνο ένας πίνακας είναι συμμετρικός (έστω ο F) σε ένα QAP με συμμετρικούς και τους δύο πίνακες (F και D), αρκεί η αντικατάσταση τού μη συμμετρικού πίνακα D από ένα νέο πίνακα E, τα στοιχεία του οποίου υπολογίζονται από την εξίσωση (16). Αν και οι δύο πίνακες F και D είναι μη συμμετρικοί, τότε το πρόβλημα ονομάζεται ασύμμετρο. Με τη μετατροπή τέτοιων στιγμιότυπων σε συμμετρικά ασχολήθηκαν οι Scott Handley, Franz Rendl και Henry Wolkowicz [45].

2.5 Γινόμενο Kronecker (Kronecker product)

Έστω $A = (a_{ij})$ πίνακας μεγέθους $m \times n$ και $B = (b_{ij})$ πίνακας μεγέθους $p \times q$. Το γινόμενο Kronecker (Kronecker product) των δύο πινάκων A και B ορίζεται ως

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \tag{18}$$

Το αποτέλεσμα του γινομένου Kronecker είναι ένας πίνακας μεγέθους $mp \times nq$, το περιεχόμενο του οποίου αποτελείται από όλα τα δυνατά γινόμενα των στοιχείων του πίνακα A με τα στοιχεία του πίνακα B. Οι ιδιότητες του γινομένου Kronecker, καθώς και υπολογιστικές διαδικασίες που το χρησιμοποιούν, παρουσιάστηκαν από τον Alexander Graham [44]. Ένα από τα αποτελέσματά του θα χρησιμοποιηθεί στην παρακάτω διατύπωση.

Έστω $vec(X) \in \Re^{n^2}$ είναι το διάνυσμα που αποτελείται από τις στήλες του πίνακα μετάθεσης X. Χρησιμοποιώντας αυτόν το συμβολισμό το QAP μπορεί να εκφραστεί ως

$$\min \quad vec(X)^T (F \otimes D) vec(X) + vec(B)^T vec(X) \tag{19}$$

έτσι ώστε

$$X \in X_n$$

2.6 Το QAP ως LAP

Βασιζόμενος στην παραπάνω διατύπωση, ο Lawler [70] διατύπωσε το Τετραγωνικό Πρόβλημα Ανάθεσης (Quadratic Assignment Problem – QAP) ως Γραμμικό Πρόβλημα Ανάθεσης (Linear Assignment Problem – LAP) με την εισαγωγή ενός επιπλέον περιορισμού. Σύμφωνα με τον περιορισμό αυτό, μόνο οι πίνακες μετάθεσης μεγέθους $n^2 \times n^2$, που αποτελούν το γινόμενο Kronecker επίσης πινάκων μετάθεσης μεγέθους $n \times n$, μπορούν να λογίζονται ως εφικτή λύση του προβλήματος.

Όπως και στην πρώτη διατύπωση του Lawler, που παρουσιάστηκε παραπάνω, θεωρούμε τον πίνακα C, ο οποίος τώρα δομείται σε δισδιάστατη μορφή και περιέχει n^4 στοιχεία. Το στοιχείο c_{ijkl} αντιστοιχεί στην $(i-1)n + j$ γραμμή και στην $(k-1)n + l$ στήλη, ενώ η τιμή του είναι $c_{ijkl} = f_{ij} d_{kl}$. Έτσι, σύμφωνα με τον Lawler, το QAP μπορεί να διατυπωθεί ως

$$\min \quad \langle C, Y \rangle \tag{20}$$

έτσι ώστε

$$Y = X \otimes X$$
$$X \in X_n$$

Όπως αναφέρθηκε, το LAP μπορεί να λυθεί επακριβώς και σε πολυωνυμικό χρόνο ($O(n^3)$) από τον αλγόριθμος Hungarian. Παρόλο που η παραπάνω διατύπωση του Lawler περιγράφει ένα LAP, ο επιπλέον περιορισμός, που πρέπει να ικανοποιείται, έχει ως αποτέλεσμα να μην μπορεί να λυθεί αποτελεσματικά.

3. Υπολογιστική πολυπλοκότητα (computational complexity)

Από θεωρητικής πλευράς, το Τετραγωνικό Πρόβλημα Ανάθεσης αποδεικνύεται ένα πολύ «σκληρό» πρόβλημα. Αυτό πρακτικά έχει αντίκτυπο στη δυσκολία ανάπτυξης χρονικά αποτελεσματικών αλγορίθμων, που να βρίσκουν τη βέλτιστη λύση του προβλήματος. Οι Sartaj Sahni και Teofilo Gonzales [93] υπολόγισαν και απέδειξαν την υπολογιστική πολυπλοκότητα του QAP. Για την παρουσίαση του αποτελέσματος της δουλειάς τους θα χρησιμοποιήσουμε το συμβολισμό

$$f(F, D, p)$$

για να δηλώσουμε την τιμή της αντικειμενικής συνάρτησης του προβλήματος με δεδομένα τον πίνακα ροών F , τον πίνακα αποστάσεων D και λύση τη μετάθεση p .

Ορισμός: *Δεδομένου ενός πραγματικού αριθμού $e > 0$, ένας αλγόριθμος Y επίλυσης του QAP ονομάζεται e-προσεγγιστικός (e-approximation algorithm) αν η ανισότητα*

$$\left| \frac{f(F, D, \pi_Y) - f(F, D, \pi_{opt})}{f(F, D, \pi_{opt})} \right| \leq e$$

ισχύει για κάθε στιγμιότυπο $QAP(F, D)$, όπου π_Y είναι η λύση που υπολογίστηκε από τον αλγόριθμο Y και π_{opt} η βέλτιστη λύση του προβλήματος. Η λύση του $QAP(F, D)$, που παράγεται από έναν e-προσεγγιστικός αλγόριθμο (e-approximation algorithm), ονομάζεται e-προσεγγιστική λύση (e-approximate solution).

■

Θεώρημα (Sahni και Gonzalez) [93]: *Το Τετραγωνικό Πρόβλημα Ανάθεσης είναι ισχυρά NP-hard. Για κάθε αυθαίρετο $e > 0$, η ύπαρξη ενός e-προσεγγιστικού αλγορίθμου για την επίλυση του QAP, που εκτελείται σε πολυωνυμικό χρόνο, συνεπάγεται $P = NP$.*

■

Η απόδειξη του θεωρήματος έγινε με αναγωγή στο πρόβλημα του κύκλου του Hamilton, το οποίο περιγράφεται από την εύρεση ή την απόδειξη μη ύπαρξης ενός κύκλου που να περιέχεται σε δεδομένο γράφο και να επισκέπτεται κάθε του κορυφή ακριβώς μία φορά.

4. Κατώτερα φράγματα (lower bounds)

Για κάθε πρόβλημα συνδυαστικής βελτιστοποίησης, άρα και για το Τετραγωνικό Πρόβλημα Ανάθεσης, είναι πολύ σημαντική η εύρεση κατώτερων φραγμάτων για την τιμή της αντικειμενικής τους συνάρτησης. Ένα καλό κατώτερο φράγμα είναι το κλειδί της επιτυχίας των αλγορίθμων διακλάδωσης και περιορισμού (branch and bound algorithms), ελαττώνοντας την περιοχή αναζήτησης για την εύρεση της βέλτιστης λύσης του προβλήματος. Μια άλλη χρήση των κατώτερων φραγμάτων είναι η αξιολόγηση της ποιότητας των προσεγγιστικών αλγορίθμων. Η καλύτερη λύση που βρίσκει ένας προσεγγιστικός αλγόριθμος για ένα στιγμιότυπο του προβλήματος, συγκρίνεται με το κατώτερο φράγμα για το στιγμιότυπο αυτό και έτσι μπορεί να υπολογιστεί ένα μέτρο για την εγγύτητά της προσεγγιστικής λύσης στη βέλτιστη. Στην ευχάριστη περίπτωση που προσεγγιστική λύση και κατώτερο φράγμα έχουν την ίδια τιμή, η βέλτιστη λύση για το συγκεκριμένο στιγμιότυπο έχει βρεθεί.

Για τους παραπάνω λόγους, είναι επιθυμητό οι τεχνικές υπολογισμού των κατώτερων φραγμάτων να δίνουν όσο το δυνατόν πιο «σφιχτά» φράγματα, δηλαδή φράγματα οι τιμές των οποίων να είναι πολύ κοντά στην τιμή της βέλτιστης λύσης. Ένα άλλο κριτήριο αξιολόγησης των τεχνικών υπολογισμού κατώτερων φραγμάτων είναι η υπολογιστική πολυπλοκότητά τους. Μια τεχνική εκτός από το να βρίσκει ένα όσο το δυνατόν μεγαλύτερο κατώτερο φράγμα, πρέπει να είναι και γρήγορη στον υπολογισμό της. Τέλος, πολύ σημαντική για την πρακτική της εφαρμογή στους αλγορίθμους διακλάδωσης και περιορισμού, είναι και η αποδοτικότητα αυτών των τεχνικών στον υπολογισμό κατώτερων φραγμάτων για υποσύνολα του αρχικού συνόλου του προβλήματος.

Για το QAP, οι τεχνικές υπολογισμού κατώτερων φραγμάτων μπορούν να χωριστούν ανεπίσημα σε τρεις κατηγορίες. Η πρώτη κατηγορία περιλαμβάνει το κλασικό πλέον φράγμα των Paul Gilmore και Eugene Lawler (GLB) [40, 70] και άλλες σχετικές με αυτό τεχνικές [14, 36, 92, 32]. Στη δεύτερη κατηγορία ανήκουν τεχνικές, οι οποίες βασίζονται στις ιδιοτιμές των πινάκων για τον υπολογισμό του κατώτερου φράγματος [45, 48, 49, 50, 88, 89]. Οι υπόλοιπες τεχνικές, που συγκροτούν την τρίτη κατηγορία, βασίζονται κυρίως σε αναδιατυπώσεις (reformulations), γραμμικοποιήσεις (linearization) και απλοποιήσεις (relaxations) του προβλήματος [1, 3, 20, 35, 46, 47] για να παράγουν λύσεις, που ενώ δεν ικανοποιούν όλους τους περιορισμούς, η τιμή τους μπορεί να αποτελέσει μια πολλή καλή προσέγγιση της βέλτιστης τιμής ενός στιγμιότυπου του QAP. Σε γενικές γραμμές, οι τεχνικές ιδιοτιμών είναι οι πιο αποτελεσματικές αναφορικά με την τιμή του φράγματος, αλλά και οι πιο δαπανηρές σε υπολογιστικό χρόνο.

5. Μέθοδοι εύρεσης της ακριβούς λύσης (exact solution methods)

Μια **μέθοδος εύρεσης της ακριβούς λύσης** (**exact solution method**) ενός προβλήματος συνδυαστικής βελτιστοποίησης χρησιμοποιείται για την εύρεση της βέλτιστης λύσης του προβλήματος. Στην περίπτωση του Τετραγωνικού Προβλήματος Ανάθεσης μια τέτοια μέθοδος βρίσκει το ολικό ελάχιστο της αντικειμενικής συνάρτησης και τη μετάθεση (permutation), με την οποία επιτυγχάνεται αυτό το ελάχιστο. Τρεις είναι οι βασικές κατηγορίες στις οποίες μπορούν να χωριστούν οι μέθοδοι εύρεσης της ακριβούς λύσης του QAP. Διαδικασίες διακλάδωσης και περιορισμού (branch and bound

procedures), δυναμικός προγραμματισμός (dynamic programming) και τεχνικές τομής επιφανειών (cutting plane techniques). Η μέχρι τώρα έρευνα έχει δείξει ότι οι μέθοδοι διακλάδωσης και περιορισμού είναι οι πιο αποτελεσματικές για τη λύση του QAP. Λόγω της εγγενούς δυσκολίας που παρουσιάζει όμως το QAP, στιγμιότυπα με μέγεθος πάνω από 30 ($n > 30$) δεν είναι δυνατό να λυθούν επακριβώς με οποιαδήποτε από αυτές τις τεχνικές.

5.1 Διακλάδωση και περιορισμός (branch and bound)

Οι μέθοδοι διακλάδωσης και περιορισμού έχουν εφαρμοστεί με επιτυχία για την εύρεση της βέλτιστης λύσης σε αρκετά προβλήματα συνδυαστικής βελτιστοποίησης. Το ίδιο ισχύει και για την εφαρμογή τους στο QAP, αφού είναι οι πιο αποδοτικές μέθοδοι για την εύρεση του ολικού ελαχίστου της συνάρτησης (1).

Τα βασικά χαρακτηριστικά ενός αλγορίθμου της κατηγορίας αυτής είναι η τεχνική διακλάδωσης και η τεχνική περιορισμού. Η τεχνική διακλάδωσης δημιουργεί ένα δέντρο αναζήτησης, οι εσωτερικοί κόμβοι του οποίου αντιπροσωπεύουν υποσύνολα της λύσης του προβλήματος, ενώ τα φύλλα του αντιστοιχούν σε ολοκληρωμένες λύσεις. Η αναζήτηση για τη βέλτιστη λύση του προβλήματος γίνεται από μία τεχνική διάσχισης του δέντρου αναζήτησης, όπως η αναζήτηση πρώτα σε βάθος (depth first search – DFS) ή η αναζήτηση πρώτα σε πλάτος (breadth first search – BFS). Μόλις η αναζήτηση φτάσει σε ένα φύλλο του δέντρου, μια λύση του προβλήματος έχει βρεθεί και υπολογίζεται η τιμή της αντικειμενικής συνάρτησης για τη λύση αυτή. Η τεχνική περιορισμού χρησιμοποιεί την καλύτερη τιμή που έχει βρεθεί, ώστε να αποκόπτει κλάδους του δέντρου που δίνουν σίγουρα μεγαλύτερη τιμή στο αποτέλεσμα. Έτσι, η αναζήτηση δε προσπελάζει αυτά τα μέρη του δέντρου και η όλη διαδικασία γίνεται συντομότερη.

Αρκετοί ερευνητές έχουν ασχοληθεί με την ανάπτυξη αλγορίθμων που βασίζονται σε διακλάδωση και περιορισμό για την εύρεση της βέλτιστης λύσης στιγμιότυπων του QAP. Οι πιο αποδοτικοί είναι αυτοί που βασίζονται στο κατώτερο φράγμα των Paul Gilmore και Eugene Lawler (GLB). Οι Peter Hahn, Thomas Grant και Nat Hall [46, 47] ανέπτυξαν μια τεχνική βασισμένη στη μέθοδο υπολογισμού κατώτερου ορίου των δύο πρώτων. Η τεχνική τους κάνει χρήση της μεθόδου Hungarian για να λύσει το QAP. Οι Πάνος Παρδαλός, K Ramakrishnan, Mauricio Resende και Yau Li [86] βασίστηκαν σε ένα διαφορετικό φράγμα για να αναπτύξουν μια τεχνική, η οποία έλυσε αρκετά, άλυτα μέχρι τότε, στιγμιότυπα του QAP. Σημαντικά αποτελέσματα υπάρχουν και στις εργασίες των Paul Gilmore [40], Eugene Lawler [70], Joseph Gavett και Norman Plyter [38], A Land [69], Christopher Nugent [80], P Mirchandani και T Obata [77] και Catherine Roucairol [91].

5.2 Δυναμικός προγραμματισμός (dynamic programming)

Οι Νίκος Χρηστοφίδης και Enrique Benavent [22] χρησιμοποιώντας μια μέθοδο δυναμικού προγραμματισμού έλυσαν επακριβώς κάποιες ειδικές περιπτώσεις στιγμιότυπων του QAP. Στα στιγμιότυπα αυτά ο πίνακας F αποτελεί τον πίνακα γειτνίασης ενός δέντρου. Τα στοιχεία του ισούνται με τα βάρη των ακμών μεταξύ των κόμβων του δέντρου. Με την τεχνική αυτή και για αυτήν την ειδική κατηγορία προβλημάτων, λύθηκαν στιγμιότυπα με μέγεθος μεγαλύτερο του 30 ($n > 30$).

Η τεχνική του δυναμικού προγραμματισμού βασίζεται στη διάσπαση του αρχικού προβλήματος σε υποπροβλήματα μικρότερου μεγέθους, κάθε ένα από τα οποία λύνεται αναδρομικά. Άρα, μπορεί να χρειάζεται η περαιτέρω διάσπασή τους. Τα υποπροβλήματα μπορεί να μην είναι ανεξάρτητα μεταξύ τους. Υπάρχει το ενδεχόμενο να μοιράζονται κάποια κοινά μικρότερα μέρη. Ένα χαρακτηριστικό του δυναμικού προγραμματισμού είναι η άπαξ επίλυση κάθ' ενός υποπροβλήματος, όσες φορές και αν εμφανιστεί αυτό κατά την αναδρομική διάσπαση και επίλυση του αρχικού προβλήματος. Η βέλτιστη ολική λύση προκύπτει από το συνδυασμό βέλτιστων λύσεων υποπροβλημάτων (αρχή του βέλτιστου – principle of optimality).

Ο δυναμικός προγραμματισμός εμφανίζεται σε δύο εκδοχές:

- Προσέγγιση από πάνω προς τα κάτω (top – down approach): Το αρχικό πρόβλημα διασπάται σε υποπροβλήματα, τα οποία επιλύονται αναδρομικά και η λύση τους αποθηκεύεται για την περίπτωση που χρειαστεί να λυθούν ξανά. Στην προσέγγιση αυτή λύνονται ακριβώς τα υποπροβλήματα που χρειάζονται για την εύρεση της λύσης του ολικού προβλήματος και κανένα παραπάνω.
- Προσέγγιση από κάτω προς τα πάνω (bottom – up approach): Όλα τα υποπροβλήματα, που μπορεί να χρειαστούν, λύνονται και οι λύσεις τους χρησιμοποιούνται για την κατασκευή λύσεων ακόμη μεγαλύτερων υποπροβλημάτων μέχρι να φτάσουμε στο αρχικό πρόβλημα. Η προσέγγιση αυτή δεν έχει αναδρομικές κλήσεις και έτσι χρησιμοποιεί λιγότερο χώρο στη στοίβα κλήσης υποπρογραμμάτων. Από την άλλη μεριά όμως δεν είναι πάντα τόσο «έξυπνη» ώστε να εντοπίζει και να λύνει ακριβώς τα υποπροβλήματα που είναι απαραίτητα για τη λύση κάποιου συγκεκριμένου προβλήματος.

5.3 Τεχνικές τομής επιφανειών (cutting plane techniques)

Η διάσπαση του Benders (Benders' decomposition) μετασχηματίζει το QAP σε μια μεικτή διαδικασία γραμμικού, ακέραιου προγραμματισμού (mixed integer linear programming – MILP). Η διατύπωση του MILP χωρίζεται σε ένα κύριο πρόβλημα (master problem) και σε ένα δευτερεύον υποπρόβλημα το οποίο καλείται και πρόβλημα σκλάβος (slave problem). Το κύριο πρόβλημα περιέχει τις αρχικές μεταβλητές και σταθερές ανάθεσης. Το δευτερεύον πρόβλημα είναι ένα γραμμικό πρόβλημα και έτσι μπορεί να λυθεί σε πολυωνυμικό χρόνο. Το κύριο πρόβλημα είναι γραμμικό μόνο αν διατυπωθεί σε σχέση με τις αρχικές μεταβλητές ανάθεσης και με τις διπλές μεταβλητές του δευτερεύοντος προβλήματος. Για το λόγο αυτό είναι δυνατόν να λυθεί σε πολυωνυμικό χρόνο μόνο για δεδομένες τιμές των διπλών αυτών μεταβλητών.

Ο αλγόριθμος των τεχνικών τομής επιφανειών μπορεί να περιγραφεί με την παρακάτω διαδικασία. Αρχικά, ένας ευρετικός μηχανισμός χρησιμοποιείται για την παραγωγή μιας πρώτης μετάθεσης. Στη συνέχεια, το δευτερεύον πρόβλημα επιλύεται για δεδομένες τιμές, που καθορίζονται από αυτή τη μετάθεση και έτσι υπολογίζονται βέλτιστες τιμές για τις διπλές μεταβλητές του. Αν η διπλή αυτή λύση του δευτερεύοντος προβλήματος ικανοποιεί όλους τους περιορισμούς του κύριου προβλήματος, τότε μια ολική βέλτιστη λύση έχει βρεθεί για την αρχική MILP διατύπωση του QAP. Σε αντίθετη περίπτωση, τουλάχιστον ένας περιορισμός του κύριου προβλήματος παραβιάζεται. Το κύριο πρόβλημα επιλύεται για συγκεκριμένες τιμές των διπλών μεταβλητών και το αποτέλεσμα του δίνεται ως είσοδος στο δευτερεύον πρόβλημα. Η διαδικασία

επαναλαμβάνεται, μέχρις ότου η λύση του δευτερεύοντος προβλήματος να ικανοποιεί όλους τους περιορισμούς που επιβάλει το κύριο πρόβλημα.

Κάθε λύση του κύριου προβλήματος, με δεδομένες τιμές των διπλών μεταβλητών του δευτερεύοντος προβλήματος, αποτελεί ένα κατώτερο φράγμα για την τιμή του QAP. Από την άλλη πλευρά, η τιμή της αντικειμενικής συνάρτησης του QAP, που αντιστοιχεί σε οποιοδήποτε σύνολο εφικτών μεταβλητών ανάθεσης, αποτελεί ένα ανώτατο φράγμα για την τιμή του QAP. Ο αλγόριθμος τερματίζεται όταν το ανώτερο και το κατώτερο φράγμα συμπέσουν. Γενικά, η επιθυμητή αυτή σύγκλιση χρειάζεται μεγάλο υπολογιστικό χρόνο, με αποτέλεσμα οι τεχνικές τομής επιφανειών να χρησιμοποιούνται μόνο για την επίλυση στιγμιότυπων του QAP με μικρό μέγεθος. Ωστόσο τα ερευνητικά αποτελέσματα δείχνουν ότι οι τεχνικές αυτές δίνουν, σε αρκετά πρώιμο στάδιο της αναζήτησης, κάποιες πολύ καλές λύσεις του προβλήματος. Επίσης, αρκετές ευρετικές μέθοδοι, που βασίζονται στην τεχνική τομής επιφανειών, έχουν πολύ καλά αποτελέσματα.

Σημαντικές εργασίες, που αφορούν τις τεχνικές αυτές, έχουν γραφεί από τους Mokhtar Barazaa και Hanif Sherali [8, 9], Egon Balas και Joseph Mazzola [5, 6, 7], L Kaufmann και F Broeckx [64] και Rainer Burkard και Tilman Bonniger [15].

6. Ευρετικές μέθοδοι (heuristics)

Παρ' όλη τη σημαντική μελέτη και πρόοδο στην ανάπτυξη αλγορίθμων εύρεσης της βέλτιστης λύσης του QAP, μόνο μικρά σε μέγεθος προβλήματα ($n \leq 30$) είναι δυνατό να λυθούν με αυτούς. Το γεγονός αυτό οφείλεται στην εγγενή δυσκολία που παρουσιάζει η επίλυση του QAP και δικαιολογείται πλήρως από την κλάση των NP-hard προβλημάτων, στην οποία ανήκει. Για το λόγο αυτό δημιουργήθηκε η ανάγκη ανάπτυξης ευρετικών μεθόδων, οι οποίες μπορούν να βρίσκουν καλές λύσεις του προβλήματος (όχι απαραίτητα βέλτιστες αλλά πολύ κοντά σε αυτές) σε μικρό σχετικά χρονικό διάστημα. Είναι επιθυμητό μια τέτοια ευρετική μέθοδος να έχει καλά αποτελέσματα τόσο στο χρόνο επίλυσης όσο και στην ποιότητα της λύσης. Αρκετή έρευνα έχει γίνει στον τομέα αυτό, τα αποτελέσματα της οποίας μπορούν να ομαδοποιηθούν στις εξής κατηγορίες:

- Κατασκευαστικές μέθοδοι (construction methods – CM)
- Μέθοδοι περιορισμένης απαρίθμησης (limited enumeration methods – LEM)
- Μέθοδοι βελτίωσης (improvement methods – IM)
- Ταμπού αναζήτηση (tabu search – TS)
- Εξομοίωση ισχυροποίησης (simulated annealing – SA)
- Γενετικοί αλγόριθμοι (genetic algorithms – GA)
- Συστήματα μυρμηγκιών (ant systems – AS)
- Νευρωνικά δίκτυα (Neural Networks – NN)
- Χαοτική βελτιστοποίηση (Chaotic Optimization – CO)
- Μέθοδος άπληστης τυχαίας προσαρμόσιμης αναζήτησης (greedy randomized adaptive search procedure – GRASP)

6.1 Κατασκευαστικές μέθοδοι (construction methods – CM)

Όπως υποδηλώνει και το όνομά τους, οι κατασκευαστικές μέθοδοι (construction methods – CM) κατασκευάζουν μια λύση του προβλήματος βήμα προς βήμα. Η βασική

ιδέα, που εισήχθηκε αρχικά από τον Paul Gilmore [40], περιλαμβάνει την κατασκευή της τελικής, ολοκληρωμένης μετάθεσης από μια μερική μετάθεση (partial permutation). Αρχικά η μερική μετάθεση, έστω p, είναι κενή. Επαναληπτικά το p επεκτείνεται με την προσθήκη μιας εκχώρησης τη φορά. Η κάθε εκχώρηση είναι ένα ζεύγος μιας εγκατάστασης, που δεν έχει τοποθετηθεί ακόμη και μιας κενής τοποθεσίας. Έστω M το σύνολο των δεικτών θέσης, για τις οποίες η αντίστοιχη ανάθεση έχει γίνει, δηλαδή η μερική μετάθεση p έχει πάρει τιμή για τη θέση αυτή. Έστω $p(M)$ είναι το σύνολο $\{p(i) | i \in M\}$. Με αυτούς τους όρους, σε κάθε βήμα της επανάληψης η μετάθεση p επεκτείνεται με την επιλογή ενός ζεύγους (i, j), έτσι ώστε $i \notin M$ και $j \notin p(M)$. Η διαδικασία επαναλαμβάνεται έως ότου το p γίνει πλήρης μετάθεση.

Σε κάθε βήμα της επαναληπτικής διαδικασίας, η επιλογή του j, της εγκατάστασης που θα αντιστοιχηθεί σε μια κενή τοποθεσία i, γίνεται με βάση μια ευρετική μέθοδο. Μια ευρετική μέθοδος, που χρησιμοποιήθηκε πρώτη στην πράξη για το λόγο αυτό, είναι η CRAFT (Computerized Relative Allocation of Facilities Technique), η οποία αναπτύχθηκε από τους Elwood Buffa, G Armour και T Vollmann [13]. Πρόκειται για μια ευρέως χρησιμοποιούμενη ευρετική μέθοδο για τη διάταξη εγκαταστάσεων, η οποία χρησιμοποιούνταν για περισσότερα από 25 χρόνια. Η είσοδος της CRAFT αποτελείται από δύο σύνολα. Το σύνολο εγκαταστάσεων και το σύνολο τοποθεσιών. Αντίστοιχα προσδιορίζεται ο πίνακας με τις τιμές των ροών μεταξύ των εγκαταστάσεων και ο πίνακας με τα κόστη για τη μετακίνηση ενός αντικειμένου ανάμεσα σε δύο τοποθεσίες. Τα κόστη είναι εκφρασμένα σε μονάδες απόστασης. Η διαδικασία επαναληπτικά βελτιώνει μια αρχική διάταξη τοποθεσιών, που έχει δώσει ο χρήστης, με αμοιβαίες ανταλλαγές θέσεων. Σε κάθε βήμα ερευνά όλες τις πιθανές 2-ανταλλαγές ή 3-ανταλλαγές ή 2-ανταλλαγές και 3-ανταλλαγές μαζί. Τελικά επιλέγει την αλλαγή, η οποία παρέχει τη μεγαλύτερη βελτιστοποίηση, μειώνοντας περισσότερο το ολικό κόστος. Η διαδικασία επαναλαμβάνεται μέχρις ότου να μην είναι δυνατή περαιτέρω βελτίωση.

6.2 Μέθοδοι περιορισμένης απαρίθμησης (limited enumeration methods – LEM)

Η λειτουργία των μεθόδων αυτών βασίζεται στην παρατήρηση ότι οι μέθοδοι απαρίθμησης, όπως η διακλάδωση και περιορισμός (branch and bound) ή η τομή επιφανειών (cutting plane), βρίσκουν καλές λύσεις σε αρχικά στάδια της αναζήτησης και στη συνέχεια ξοδεύουν πολύ χρόνο στο να βελτιώσουν τις λύσεις αυτές ή να αποδείξουν ότι είναι βέλτιστες. Ένα χαρακτηριστικό παράδειγμα της περίπτωσης αυτής είναι η παρατήρηση που έγινε στη διαδικασία εύρεσης της βέλτιστης λύσης του στιγμιότυπου του QAP του Nugent με μέγεθος 15 ($n = 15$). Σε μια συγκεκριμένη εκτέλεση αλγορίθμου για την επίλυσή του, η βέλτιστη λύση βρέθηκε σε 23.48 δευτερόλεπτα ενώ χρειάστηκε περισσότερο από μία ώρα για να αποδείξει ο αλγόριθμος ότι η λύση αυτή είναι βέλτιστη.

Σύμφωνα με την παραπάνω παρατήρηση, αν η επιδίωξή μας είναι η γρήγορη εύρεση μιας καλής λύσης του προβλήματος, η οποία να μην είναι απαραίτητα βέλτιστη, το μόνο που έχουμε να κάνουμε είναι να επιβάλουμε κάποιον περιορισμό στη μέθοδο απαρίθμησης. Ο περιορισμός αυτός μπορεί να είναι χρονικός ή μπορεί να είναι ένα φράγμα στο πλήθος των επαναλήψεων που θα κάνει ο αλγόριθμος αναζήτησης. Ο χρονικός περιορισμός μπορεί να αφορά το συνολικό χρόνο που θα εκτελείται ο αλγόριθμος ή το χρονικό περιθώριο στο οποίο αν καμιά βελτιστοποίηση δεν

πραγματοποιηθεί, να σταματάει η εκτέλεση. Μια διαφορετική στρατηγική για την επιβολή περιορισμού είναι η παραποίηση του κατώτερου ορίου ανάλογα με τις ανάγκες μας. Αν για ένα μεγάλο πλήθος επαναλήψεων του αλγορίθμου αναζήτησης δεν έχει επιτευχθεί καμιά βελτιστοποίηση της καλύτερης λύσης, τότε η μείωση της τιμής του κατώτερου ορίου κατά ένα συγκεκριμένο ποσοστό μπορεί να οδηγήσει σε μεγαλύτερους περιορισμούς (βαθύτερα κοψίματα στην περίπτωση της διακλάδωσης και περιορισμού) στο δέντρο αναζήτησης, ώστε η διαδικασία να επιταχυνθεί. Η ενέργεια αυτή απαιτεί μεγάλη προσοχή στην παραποίηση της τιμής του κατώτερου ορίου, επειδή είναι πολύ πιθανή η αποκοπή της βέλτιστης λύσης. Ακόμη και σ' αυτήν την περίπτωση όμως με την τεχνική αυτή μπορούμε να είμαστε σίγουροι ότι η βέλτιστη λύση διαφέρει από την καλύτερη που βρέθηκε όχι περισσότερο από ένα συγκεκριμένο ποσοστό.

6.3 Μέθοδοι βελτίωσης (improvement methods – IM)

Οι μέθοδοι βελτίωσης (improvement methods – IM) ανήκουν σε μια μεγαλύτερη κλάση αλγορίθμων τοπικής αναζήτησης (local search algorithms). Οι αλγόριθμοι τοπικής αναζήτησης είναι επαναληπτικές διαδικασίες, οι οποίες σε κάθε βήμα αναζητούν στη γειτονιά της υπάρχουσας λύσης μια διαφορετική (όχι απαραίτητα καλύτερη) λύση, η οποία θα αποτελέσει τη βάση για την αναζήτηση στο επόμενο βήμα της επανάληψης. Με τον τρόπο αυτό προσπαθούν να βελτιώσουν μια αρχική λύση μέχρις ότου περαιτέρω βελτίωση να μην είναι δυνατή.

Οι μέθοδοι βελτίωσης είναι αλγόριθμοι τοπικής αναζήτησης, οι οποίοι σε κάθε βήμα της επαναληπτικής διαδικασίας επιτρέπουν μόνο βελτίωση της υπάρχουσας λύσης. Τρία βασικά χαρακτηριστικά των μεθόδων αυτών είναι ο ορισμός της γειτονιάς κάθε λύσης, ο τρόπος σάρωσης αυτής της γειτονιάς και ο κανόνας ενημέρωσης της υπάρχουσας λύσης.

Στην περίπτωση του QAP την κάθε λύση αποτελεί μια μετάθεση (permutation), έστω p. Συχνά χρησιμοποιούμενες γειτονιές είναι αυτές που προκύπτουν από την αμοιβαία ανταλλαγή όλων των ζευγών τιμών του p (pair-exchange neighborhood) ή την κυκλική αμοιβαία ανταλλαγή τριάδων τιμών του p (cyclic triple-exchange neighborhood). Η πρώτη από αυτές τις γειτονιές έχει μέγεθος $\binom{n}{2}$ ενώ η δεύτερη $\binom{n}{3}$. Πειραματικά αποτελέσματα δείχνουν ότι η κυκλική ανταλλαγή τριάδων δεν παρουσιάζει κάποιο πλεονέκτημα έναντι της ανταλλαγής ζευγών.

Εκτός από τον ορισμό της γειτονιάς, πολύ σημαντικός στην εκτέλεση του αλγορίθμου είναι και ο τρόπος με τον οποίο σαρώνεται αυτή η γειτονιά. Ο τρόπος αυτός υποδεικνύει στον αλγόριθμο με ποια σειρά θα προσπελαστούν τα στοιχεία της γειτονιάς της υπάρχουσας λύσης. Η σειρά αυτή μπορεί να είναι είτε καθορισμένη από πριν με κάποιον κανόνα είτε να επιλέγεται τυχαία.

Το τρίτο χαρακτηριστικό των μεθόδων βελτίωσης, που συμπληρώνει την επιλογή μιας δομής γειτονίας και ενός τρόπου σάρωσης αυτής, είναι ο κανόνας ενημέρωσης της υπάρχουσας λύσης. Συχνά χρησιμοποιούμενοι κανόνες ενημέρωσης μπορούν να χωριστούς σε τρεις κατηγορίες.

- Πρώτη βελτίωση: στην περίπτωση αυτή η υπάρχουσα λύση ενημερώνεται μόλις βρεθεί ο πρώτος καλύτερος γείτονάς της (αν υπάρχει κάποιος τέτοιος), ακόμη και αν δεν έχει σαρωθεί ολόκληρη η γειτονιά της.
- Καλύτερη βελτίωση: στην περίπτωση αυτή ολόκληρη η γειτονιά της υπάρχουσας λύσης σαρώνεται και επιλέγεται για την ενημέρωση η καλύτερη λύση της γειτονιάς (αν βέβαια υπάρχει κάποια τέτοια).
- Κανόνας του Heider [52]: σύμφωνα με τον κανόνα αυτό, η γειτονιά της αρχικής λύσης αρχίζει να σαρώνεται με έναν προκαθορισμένο, κυκλικό τρόπο. Μόλις βρεθεί κάποια καλύτερη λύση, αυτή αντικαθιστά την υπάρχουσα (γίνεται η ενημέρωση). Η σάρωση της γειτονιάς της νέας λύσης ξεκινά από εκεί που διακόπηκε η σάρωση της προηγούμενης λύσης (από το συγκεκριμένο σημείο της κυκλικής διάταξης).

Οι μέθοδοι βελτίωσης σε κάθε τους εκτέλεση βρίσκουν ένα τοπικό ελάχιστο της αντικειμενικής συνάρτησης του προβλήματος. Για να υπάρχει η δυνατότητα εύρεσης του ολικού ελαχίστου, αλλά και γενικά για την επίτευξη καλύτερων λύσεων, τόσο οι αλγόριθμοι τοπικής αναζήτησης όσο και οι μέθοδοι βελτίωσης εκτελούνται αρκετές φορές ξεκινώντας από διαφορετικές αρχικές λύσεις.

6.4 Ταμπού αναζήτηση (tabu search – TS)

Η ταμπού αναζήτηση (tabu search – TS) εισήχθηκε από το Fred Glover [41, 42]. Πρόκειται για μια μέθοδο τοπικής αναζήτησης, η οποία προσπαθεί να ξεπεράσει τον εγκλωβισμό σε τοπικά ελάχιστα της αντικειμενικής συνάρτησης. Ένας τρόπος για την επίτευξη του στόχου αυτού είναι η αλλαγή της τεχνικής ενημέρωσης της λύσης. Σε αντίθεση με τις μεθόδους βελτίωσης, σε κάθε βήμα της επαναληπτικής διαδικασίας επιτρέπεται η επιλογή μιας λύσης όχι μόνο με μικρότερη, αλλά και με μεγαλύτερη τιμή από αυτή της υπάρχουσας λύσης. Στην ταμπού αναζήτηση η βασική ιδέα είναι ο περιορισμός των κατευθύνσεων αναζήτησης σε κάθε βήμα της επανάληψης, ώστε να επιτευχθεί η εύρεση καλής λύσης με αποδοτικό τρόπο. Η μέθοδος «θυμάται» ποιες λύσεις έχουν ήδη αναζητηθεί, ώστε να υπολογίσει τις πιο υποσχόμενες κατευθύνσεις αναζήτησης. Με τον τρόπο αυτό όχι μόνο η τοπικότητα της γειτονιάς, αλλά και η μνήμη της μεθόδου καθοδηγούν την αναζήτηση.

Μια βασική εισαγωγή στους αλγορίθμους ταμπού αναζήτησης γίνεται στο βιβλίο των Fred Glover, Manuel Laguna, Eric Taillard και Dominique de Werra [43]. Διαφορετικές ιδέες επί του θέματος έχουν παρουσιαστεί, όπως η εύρωστη (robust) ταμπού αναζήτηση από τον Eric Taillard [98], όπου το μέγεθος της ταμπού λίστας επιλέγεται από ένα διάστημα τιμών, η ταμπού αναζήτηση με αμετάβλητη λίστα από τον Jadranka Skorin-Kapov [96] και η ταμπού αναζήτηση με ανάδραση από τους Roberto Battiti και Gampietro Tecchiolli [10], όπου το μέγεθος της ταμπού λίστας μεταβάλλεται ανάλογα με το μέγεθος του μεγαλύτερου κύκλου που έχει κάνει η αναζήτηση.

Τα βασικά χαρακτηριστικά γνωρίσματα μιας ταμπού αναζήτησης είναι η δομή της γειτονιάς (neighborhood structure), οι κινήσεις (moves), η λίστα ταμπού (tabu list) και το κριτήριο φιλοδοξίας (aspirations criterion). Μια κίνηση είναι η διαδικασία εύρεσης μιας γειτονικής λύσης δεδομένης μιας υπάρχουσας λύσης. Όπως αναφέρθηκε και παραπάνω, στην περίπτωση του QAP η κινήσεις συνήθως είναι οι αλληλομεταθέσεις, ώστε να προκύψουν οι γειτονιές από την ανταλλαγή ζευγών ή τριάδων της υπάρχουσας μετάθεσης (permutation). Η λίστα ταμπού περιλαμβάνει ένα σύνολο απαγορευμένων

κινήσεων, κινήσεων οι οποίες δεν επιτρέπεται να συμπεριληφθούν στη λύση που αναζητείται. Η λίστα αυτή ενημερώνεται διαρκώς με νέες εισόδους σε αυτήν ή διαγραφές από αυτήν. Το κριτήριο φιλοδοξίας είναι μια προϋπόθεση, την οποία αν ικανοποιεί μια κίνηση τότε η τελευταία διαγράφεται από τη λίστα ταμπού.

Κάθε αλγόριθμος ταμπού αναζήτησης ξεκινάει με μια αρχική λύση. Σε κάθε βήμα της επαναληπτικής διαδικασίας επιλέγεται η λύση με την καλύτερη ποιότητα από ένα μέρος της γειτονιάς της υπάρχουσας λύσης. Σε αντίθεση με τις μεθόδους βελτίωσης, εδώ η λύση καλύτερης ποιότητας μπορεί και να αυξάνει την τιμή της αντικειμενικής συνάρτησης. Η υπάρχουσα λύση αντικαθιστάται από την επιλεχθείσα. Αν δεν υπάρχει κανένας έλεγχος, η διαδικασία αναζήτησης μπορεί να κάνει κύκλους, προσπελάζοντας κάποιες λύσεις περισσότερες από μια φορές. Για την αποφυγή του φαινομένου αυτού, υπάρχει ένας μηχανισμός αναγνώρισης των κινήσεων, που μπορεί να οδηγήσουν σε κύκλους. Οι κινήσεις αυτές χαρακτηρίζονται απαγορευμένες. Μπαίνουν στη λίστα ταμπού και η αναζήτηση δεν τις περιλαμβάνει. Έτσι η διαδικασία γίνεται ταχύτερη. Πάντα υπάρχει όμως η περίπτωση να χαρακτηριστεί ως απαγορευμένη μια κίνηση, η οποία οδηγεί σε μια καλή λύση ή ακόμη και στη βέλτιστη. Για το λόγο, αυτό ένα κριτήριο φιλοδοξίας ξεχωρίζει τις πιθανώς ενδιαφέρουσες κινήσεις από τη λίστα ταμπού και τις βγάζει από αυτήν. Η αναζήτηση συνεχίζεται μέχρις ότου ικανοποιηθεί κάποιο κριτήριο τερματισμού.

6.5 Προσομοίωση ισχυροποίησης (simulated annealing – SA)

Η προσομοίωση ισχυροποίησης (simulated annealing – SA) είναι μια μέθοδος τοπικής αναζήτησης, η οποία αξιοποιεί την αναλογία που υπάρχει μεταξύ των προβλημάτων συνδυαστικής βελτιστοποίησης και των προβλημάτων της στατιστικής μηχανικής (statistical mechanics). Οι πρώτοι που παρατήρησαν την αναλογία αυτή ήταν οι S Kirpatrick, C Gelatti και M Vecchi [65], οι οποίοι τη χρησιμοποίησαν ως μια μέθοδο για να ξεπεράσουν τον εγκλωβισμό σε τοπικά ελάχιστα. Έδειξαν πώς ο αλγόριθμος Metropolis (Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller και Edward Teller [76]), εφαρμοζόμενος για την προσομοίωση της συμπεριφοράς φυσικών συστημάτων υψηλής ενέργειας, μπορεί να χρησιμοποιηθεί ως ευρετική μέθοδος για την επίλυση του προβλήματος του πλανόδιου πωλητή (traveling salesman problem).

Η αναλογία μεταξύ των προβλημάτων συνδυαστικής βελτιστοποίησης και των φυσικών συστημάτων υψηλής ενέργειας αναδεικνύεται σε δύο επίπεδα.

- Οι εφικτές λύσεις του προβλήματος συνδυαστικής βελτιστοποίησης αντιστοιχούν στις καταστάσεις του φυσικού συστήματος.
- Οι τιμές της αντικειμενικής συνάρτησης του προβλήματος αντιστοιχούν στην ενέργεια των καταστάσεων του φυσικού συστήματος.

Ο όρος ισχυροποίηση (annealing) χρησιμοποιείται στην επεξεργασία ενός θερμικού συστήματος, το οποίο αρχικά λιώνει σε υψηλή θερμοκρασία. Στη συνέχεια, η θερμοκρασία του ελαττώνεται αργά με ρυθμό που καθορίζεται από ένα χρονοδιάγραμμα ισχυροποίησης. Η διαδικασία συνεχίζεται έως ότου βρεθεί η θερμοκρασία σταθεροποίησης, στην οποία το σύστημα βρίσκεται σε κατάσταση ελαχίστης ενέργειας. Η προσομοίωση ισχυροποίησης είναι μια Monte Carlo μέθοδος, η οποία προσομοιώνει τη συμπεριφορά του συστήματος, ώστε να πετύχει θερμική ισορροπία με δεδομένο

χρονοδιάγραμμα ισχυροποίησης σε συγκεκριμένη θερμοκρασία. Η διαδικασία αυτή υιοθετείται και για τη λύση προβλημάτων συνδυαστικής βελτιστοποίησης.

Συγκεκριμένα, η εφαρμογή της στο QAP μπορεί να γίνει με τον παρακάτω τρόπο. Δεδομένης μιας λύσης του προβλήματος, δηλαδή μιας μετάθεσης (permutation) των εγκαταστάσεων σε σχέση με τις τοποθεσίες, τυχαία επιλέγονται δύο εγκαταστάσεις και αλλάζουν μεταξύ τους θέση. Υπολογίζεται η διαφορά δf στην τιμή της αντικειμενικής συνάρτησης μετά την αλλαγή αυτή. Η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί κάποια διαφορά $\delta f < 0$. Αν δε βρεθεί λύση που να δίνει αρνητική διαφορά, τότε επιλέγεται μια τυχαία τιμή για τη μεταβλητή x από την ομοιόμορφη κατανομή $U(0,1)$. Αν $x < \exp(-\delta f / t_i)$, τότε γίνεται δεκτή η ανταλλαγή του ζεύγους και η διαδικασία επαναλαμβάνεται. Το t_i εδώ αντιπροσωπεύει τη θερμοκρασία στην κατάσταση i, ενώ η ανισότητα $t_1 > t_2 > ... > t_r$ αντιπροσωπεύει το χρονοδιάγραμμα ισχυροποίησης. Το σύστημα παραμένει στην κατάσταση i όσο ένα προκαθορισμένο πλήθος ανταλλαγής ζευγών εφαρμόζεται και υπολογίζεται η διαφορά που επιφέρουν στην αντικειμενική συνάρτηση. Αν όλες οι θερμοκρασίες στο χρονοδιάγραμμα έχουν χρησιμοποιηθεί, άρα $i > r$, τότε η διαδικασία σταματά.

Οι Rainer Burkard και Franz Rendl έδειξαν ότι η παραπάνω διαδικασία μπορεί να χρησιμοποιηθεί ως ευρετική μέθοδος, η οποία μπορεί να εφαρμοστεί σε οποιοδήποτε πρόβλημα συνδυαστικής βελτιστοποίησης, για το οποίο μπορεί να περιγραφεί μια δομή γειτνίασης στο σύνολο των εφικτών του λύσεων. Το QAP πληροί την παραπάνω προϋπόθεση. Τόσο ο αρχικός αλγόριθμος των Burkard και Rendl [19] όσο και άλλοι αλγόριθμοι προσομοίωσης ισχυροποίησης, όπως των Mickey Wilhelm και Thomas Ward [103] και του David Connolly [24], για την επίλυση του QAP χρησιμοποιούν ως δομή γειτονιάς αυτή που προκύπτει από την αμοιβαία ανταλλαγή ζευγών (pair-exchange neighborhood). Οι αλγόριθμοι αυτοί διαφέρουν στη διαδικασία ψύξης και στη διαδικασία θερμικής εξισορρόπησης, δηλαδή στον τρόπο μετάβασης από τη μια λύση στην άλλη και στο κριτήριο τερματισμού. Αριθμητικά αποτελέσματα δείχνουν ότι η απόδοση αυτών των αλγορίθμων εξαρτάται πολύ από τις τιμές των παραμέτρων που ελέγχουν την εκτέλεσή τους.

6.6 Γενετικοί αλγόριθμοι (genetic algorithms – GA)

Οι γενετικοί αλγόριθμοι (genetic algorithm – GA), όπως υποδεικνύει και το όνομά τους, έχουν εμπνευστεί από τη διαδικασία της φυσικής επιλογής και της εξέλιξης των ειδών. Ο πρώτος που είχε την ιδέα της εφαρμογής της διαδικασίας αυτής στην επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης ήταν ο John Holland [53]. Τον ακολούθησαν χρονικά οι David Tate και Alice Smith [99], οι Charles Fleurent και Jacques Ferland [33] και οι Ravindra Ahuja, James Orlin και Ashish Tiwari [2].

Ο γενετικός αλγόριθμος ξεκινάει με ένα πλήθος αρχικών λύσεων, οι οποίες είτε επιλέγονται τυχαία, είτε παράγονται με μια ευρετική διαδικασία. Το σύνολο των λύσεων αυτών αποτελεί τον αρχικό πληθυσμό. Τα στοιχεία του συνόλου συνήθως ονομάζονται άτομα. Ο αλγόριθμος επιλέγει ένα πλήθος ζευγών από τα άτομα για να αποτελέσουν τους γονείς της επόμενης γενιάς. Χρησιμοποιώντας κανόνες διασταύρωσης παράγει απογόνους των ζευγαριών και έτσι ο πληθυσμός ανανεώνεται. Επιπλέον, μια διαδικασία επιλογής αναγνωρίζει τα άτομα που δε θα συνεχίσουν να ανήκουν στον πληθυσμό.

Τέτοια άτομα συνήθως είναι αυτά που αντιστοιχούν σε λύσεις, για τις οποίες η αντικειμενική συνάρτηση του προβλήματος έχει μεγάλη τιμή. Η διαδικασία επαναλαμβάνεται έως ότου εκπληρωθεί ένα κριτήριο τερματισμού. Αυτό μπορεί να είναι κάποιο χρονικό φράγμα, κάποιο φράγμα στο πλήθος των επαναλήψεων ή ακόμη και ένα μέτρο σύγκλισης της τιμής της αντικειμενικής συνάρτησης.

Κατά τη διάρκεια της επαναληπτικής διαδικασίας, περιοδικά επιβάλλονται στον πληθυσμό κάποιες τυχαίες μεταλλάξεις και μεταναστεύσεις. Με τον τρόπο αυτό κάποια από τα άτομα – λύσεις είτε μεταβάλλονται, είτε αντικαθίστανται από άλλα. Οι δύο αυτές διαδικασίες διαφοροποίησης του πληθυσμού ως στόχο έχουν την αποφυγή τοπικών ελαχίστων της αντικειμενικής συνάρτησης και γενικά τη διεύρυνση των ορίων της αναζήτησης. Ως κανόνας διασταύρωσης μπορεί να χρησιμοποιηθεί οποιαδήποτε διαδικασία, η οποία παίρνοντας ως είσοδο δύο άτομα – λύσεις μπορεί να παράγει μια τρίτη, η οποία θα έχει χαρακτηριστικά και από τους δύο γονείς τις. Στην περίπτωση του QAP, που η κάθε λύση είναι μια μετάθεση (permutation), ο κανόνας διασταύρωση μπορεί να περιγραφεί ως εξής: κάθε παιδί παίρνει την τιμή κάθε θέσης στη μετάθεσή του είτε από τον πατέρα του, είτε από τη μητέρα του. Έτσι, τα στοιχεία της κάθε μετάθεσης αντιμετωπίζονται ως χαρακτηριστικά που το παιδί κληρονομεί είτε από τον πατέρα του είτε από τη μητέρα του.

Στους γενετικούς αλγορίθμους η διαδικασία αναζήτησης διαιρείται σε περιοδείες. Κάθε περιοδεία περιλαμβάνει ένα πλήθος εκτελέσεων του γενετικού αλγορίθμου, ξεκινώντας από διαφορετικούς αρχικούς πληθυσμούς και σταματάει πριν αυτοί συγκλίνουν μεταξύ τους. Ένας πληθυσμός επιλέγεται ως καλύτερος από αυτούς που έχουν προκύψει στο τέλος κάθε εκτέλεσης. Τέλος, μια νέα εκτέλεση του γενετικού αλγορίθμου πραγματοποιείται με είσοδο αυτόν τον πληθυσμό.

6.7 Συστήματα μυρμηγκιών (ant systems – AS)

Η φερομόνη είναι μια χημική ουσία που απελευθερώνεται από ένα ζώο για να επικοινωνήσει με άλλα του ίδιου είδους. Μπορεί να προκαλέσει μια συγκεκριμένη αντίδραση ή να μεταδώσει κάποιο μήνυμα. Υπάρχουν φερομόνες κινδύνου, εντοπισμού φαγητού, σεξουαλικές και πολλές άλλες που επηρεάζουν τη συμπεριφορά και τη φυσιολογία των μελών του είδους. Ο όρος φερομόνη προτάθηκε από τους Peter Karlson και Martin Lüscher [63] το 1959. Ετυμολογικά προέρχεται από τις ελληνικές λέξεις «φέρω» και «ορμόνη».

Η φερομόνη χρησιμοποιείται πολύ από τα μυρμήγκια στις αποικίες τους κατά την αναζήτηση τροφής. Σε κάθε νέα αποικία τους, αρχικά, τα μυρμήγκια μη γνωρίζοντας την περιοχή ψάχνουν για τροφή με τυχαίο τρόπο. Μόλις κάποιο μυρμήγκι βρει μια ποσότητα τροφής, κουβαλάει ένα μέρος της πίσω στη φωλιά του. Στη διαδρομή αυτή το μυρμήγκι αφήνει πίσω του ίχνη φερομόνης στο έδαφος από το οποίο περνάει. Το μονοπάτι φερομόνης που δημιουργείται, αποτελεί έναν οδηγό για μελλοντική αναζήτηση, κατευθύνοντας τα μυρμήγκια στην περιοχή που έχει ήδη βρεθεί τροφή. Η αναζήτηση παύει να είναι τυχαία. Η περιεκτικότητα της φερομόνης που αφήνει το κάθε μυρμήγκι πίσω του είναι ανάλογη με την ποσότητα τροφής που βρήκε. Με τον τρόπο αυτό οι διαδρομές που οδηγούν σε μεγαλύτερες ποσότητες τροφής υποδεικνύονται από ίχνη φερομόνης μεγαλύτερης περιεκτικότητας. Τις περιοχές αυτές θα επισκέπτεται μεγαλύτερο πλήθος μυρμηγκιών, καθοδηγούμενων από τη φερομόνη.

Τα συστήματα μυρμηγκιών (ant system – AS) προσπαθούν να μιμηθούν τη συμπεριφορά μιας αποικίας μυρμηγκιών κατά την αναζήτηση τροφής. Πρόκειται για ευρετικές τεχνικές στη βελτιστοποίηση συνδυαστικών προβλημάτων. Για να επιτευχθεί η αναλογία μεταξύ ενός προβλήματος συνδυαστικής βελτιστοποίησης και του προβλήματος εντοπισμού τροφής στην αποικία των μυρμηγκιών, χρειάζεται να γίνουν οι εξής αντιστοιχίες:

- Η περιοχή αναζήτησης γύρω από την αποικία των μυρμηγκιών αντιστοιχεί στο σύνολο των εφικτών λύσεων του προβλήματος.
- Η ποσότητα του φαγητού που, ανακαλύπτεται σε κάποια περιοχή, αντιστοιχεί στην τιμή της αντικειμενικής συνάρτησης του προβλήματος για την αντίστοιχη λύση.
- Τα μονοπάτια φερομόνης αντιστοιχούν στα στοιχεία μιας δομής δεδομένων που παρέχει ένα είδος προσαρμόσιμης μνήμης στη μέθοδο.
- Τα μυρμήγκια αντιστοιχούν σε συνεργαζόμενους πράκτορες (agents), κάθε ένας από τους οποίους ψάχνει για λύσεις του προβλήματος.

Τα συστήματα μυρμηγκιών αρχικά παρουσιάστηκαν από το Marco Dorigo [27]. Βασισμένοι στην ιδέα του οι Alberto Colorni, Vittorio Maniezzo και Marco Dorigo [23] και οι Luca Maria Gambardella, Eric Taillard και Marco Dorigo [37] ανέπτυξαν αλγορίθμους για την επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης.

Συγκεκριμένα για το QAP, τα μονοπάτια φερομόνης υλοποιούνται από έναν πίνακα, τα στοιχεία του οποίου αντιπροσωπεύουν ένα μέτρο για το πόσο επιθυμητή είναι η κάθε ανάθεση. Αν ο πίνακας είναι ο $T = (\tau_{ij})$, τότε το στοιχείο τ_{ij} είναι ένα μέτρο καταλληλότητας της τοποθέτησης της εγκατάστασης i στη θέση j. Ο αλγόριθμος περιγράφεται από μια επαναληπτική διαδικασία, σε κάθε βήμα της οποίας παράγεται ένα πλήθος λύσεων του προβλήματος. Το πλήθος αυτό ανήκει στις παραμέτρους που προκαθορίζονται για τον αλγόριθμο και αντιστοιχεί στο πλήθος των μυρμηγκιών της αποικίας. Αρχικά, τα στοιχεία του πίνακα T έχουν όλα την ίδια τιμή, με αποτέλεσμα στην πρώτη επανάληψη να μην υπάρχει γνώση για την καταλληλότητα της κάθε κατεύθυνσης αναζήτησης. Οι λύσεις της πρώτης επανάληψης δημιουργούνται τυχαία. Ανάλογα, τα μυρμήγκια αρχικά ψάχνουν χωρίς γνώση στην περιοχή. Η τιμή των στοιχείων τ_{ij} του πίνακα T είναι αντιστρόφως ανάλογη από την καλύτερη τιμή της αντικειμενικής συνάρτησης που έχει βρεθεί. Σε κάθε βήμα της επανάληψης, η τιμή κάποιων στοιχείων του πίνακα T αυξάνεται, αντιπροσωπεύοντας έτσι τη δημιουργία του μονοπατιού φερομόνης. Έστω ότι η καλύτερη λύση (μετάθεση) που έχει βρεθεί είναι η $p*$ και η αντίστοιχη τιμή της αντικειμενικής συνάρτησης $f(p*)$. Τα στοιχεία $\tau_{ip*(i)}$ του πίνακα T είναι αυτά, των οποίων η τιμή θα αυξηθεί. Η αύξηση είναι ίδια για όλα τα στοιχεία και ανάλογη με την τιμή της $f(p*)$. Αντίστοιχα, η περιεκτικότητα της φερομόνης στο κάθε μονοπάτι είναι ανάλογη με την ποσότητα τροφής στην περιοχή που οδηγεί το μονοπάτι. Η διαδικασία συνεχίζεται με τις λύσεις να μην παράγονται τυχαία αλλά να προτιμούνται λύσεις, οι οποίες περιέχουν στοιχεία του πίνακα T με όσο το δυνατόν μεγαλύτερο άθροισμα. Τα μυρμήγκια πλέον δεν ψάχνουν στα τυφλά, αλλά οδηγούνται από τη φερομόνη. Έτσι, μόλις βρεθεί κάποια καλή λύση ο αλγόριθμος προτρέπει τα μυρμήγκια – λύσεις σε περαιτέρω αναζήτηση στην περιοχή της λύσης αυτής. Αν μετά από ένα μεγάλο αριθμό επαναλήψεων δεν υπάρχει βελτίωση της καλύτερης λύσης που έχει βρεθεί, μια νέα εκτέλεση της διαδικασίας με ένα νέο πλήθος αρχικών τυχαίων λύσεων μπορεί ξεκινήσει.

Αριθμητικά αποτελέσματα έχουν δείξει ότι τα συστήματα μυρμηγκιών δίνουν πολύ καλά αποτελέσματα σε στιγμιότυπα του QAP που αντιπροσωπεύουν δεδομένα του πραγματικού κόσμου. Ειδικά για περιπτώσεις που τα στιγμιότυπα έχουν κάποιες καλές λύσεις τους ομαδοποιημένες, τα συστήματα μυρμηγκιών αποδεικνύονται από τους καλύτερους ευρετικούς τρόπους επίλυσης. Η ομαδοποίηση των καλών λύσεων ορίζεται πάντα στα πλαίσια της δομής γειτνίασης που ισχύει για κάθε μέθοδο. Από την άλλη πλευρά, σε στιγμιότυπα του QAP που έχουν παραχθεί τυχαία και έχουν καλές λύσεις ομοιόμορφα κατανεμημένες σε όλη την περιοχή αναζήτησης, τα συστήματα μυρμηγκιών δεν αποδεικνύονται και πολύ ανταγωνιστικά σε σχέση με άλλες μεθόδους.

6.8 Νευρωνικά δίκτυα (neural networks – NN)

Μια ιδιαίτερη αντιμετώπιση του QAP αποτελεί η προσπάθεια επίλυσής του με τη βοήθεια νευρωνικών δικτύων (neural network – NN). Η μέθοδος αυτή, παρόλο που δεν έχει δώσει μέχρι στιγμής ικανοποιητικά αριθμητικά αποτελέσματα ώστε να μπορέσει να ανταγωνιστεί κάποια από τις υπόλοιπες μεθόδους, παρουσιάζει μεγάλο θεωρητικό ενδιαφέρον. Όπως παρατηρεί ο Gamin Azim [4], που ανέπτυξε αυτή τη μέθοδο, η μελέτη της μπορεί να οδηγήσει σε διαφορετικούς χαρακτηρισμούς και διατυπώσεις του προβλήματος και των συνθηκών που το πλαισιώνουν.

Στις αρχές του 1980 ο John Hopfield [56, 57, 58] ανέπτυξε μια νέα ιδέα για τη χρησιμοποίηση φυσικών συστημάτων όπως τα τεχνητά νευρωνικά δίκτυα στην επίλυση υπολογιστικών προβλημάτων. Τα νευρωνικά δίκτυα που αναπτύχθηκαν από την ιδέα αυτή πήραν το όνομά του (Hopfield neural network). Το κύριο χαρακτηριστικό τους είναι η παρουσία ανάδρασης (feedback) μεταξύ κάθε ζεύγους νευρώνων. Σε κάθε ανάδραση αντιστοιχεί ένα βάρος, που εκφράζει την επιρροή που έχει ο ένας νευρώνας στον άλλο και συγκεκριμένα, ο νευρώνας από τον οποίο ξεκινάει η ανάδραση στο νευρώνα στον οποίο καταλήγει η ανάδραση. Κατά τη διαδικασία εκπαίδευσης του νευρωνικού δικτύου, όχι μόνο τα βάρη αλλά και μια εξωτερική επίδραση εφαρμόζεται σε κάθε νευρώνα. Η ανάδραση και η εξωτερική αυτή επίδραση καθορίζουν την είσοδο y_i του νευρώνα i ως

$$y_i^{new} = y_i^{old} + \Delta t\left(\sum_{j=1}^{n} T_{ij}x_j + I_i - \frac{y_i^{old}}{\eta}\right) \tag{21}$$

όπου

η είναι μια παράμετρος κανονικοποίησης

Δt είναι το μέγεθος του βήματος χρόνου

x_j είναι η έξοδος του νευρώνα j

I_i είναι η εξωτερική επίδραση στο νευρώνα i

T_{ij} είναι το βάρος της σύνδεσης μεταξύ του νευρώνα i και του νευρώνα j

Η έξοδος x_j του νευρώνα j καθορίζεται από μια μη γραμμική, σιγμοειδή συνάρτηση και δίνεται από τον τύπο

$$x_j = \frac{1}{2}\left(1 + \tanh \lambda y_j\right) \tag{22}$$

όπου

λ είναι μια παράμετρος του αλγορίθμου που καθορίζει την κλίση της σιγμοειδούς συνάρτησης.

Σε ένα νευρωνικό δίκτυο αυτού του τύπου με n νευρώνες, η ενέργεια της κάθε κατάστασής του εξαρτάται από τα βάρη και τις εξωτερικές επιδράσεις. Η τιμή της δίνεται από τη συνάρτηση Lyapunov

$$E(x) = -\frac{1}{2}\sum_{i,j=1}^{n} T_{ij} x_j x_i - \sum_{i=1}^{n} I_i x_i + \sum_{i=1}^{n} \int_{0.5}^{x} g_\lambda^{-1}(x) dx \tag{23}$$

όπου

$g_\lambda(x)$ είναι η σχέση εισόδου – εξόδου που εξαρτάται από την παράμετρο λ και δηλώνει τη συνεισφορά του κάθε νευρώνα στην συνάρτηση ενέργειας. Αναλυτικά ο ορισμός της συνάρτησης αυτής και η σημασία της ερευνήθηκαν από τους John Hopfield και David Tank [54, 55]. Η ενέργεια του νευρωνικού δικτύου συγκλίνει σε κάποιο τοπικό ελάχιστο, που αντιστοιχεί σε κάποια ακμή του υπερκύβου n-διαστάσεων, που ορίζει η διάταξη του νευρωνικού δικτύου.

Για την επίλυση ενός προβλήματος συνδυαστικής βελτιστοποίησης με ένα νευρωνικό δίκτυο Hopfield χρειάζεται να γίνουν τα παρακάτω:

- Επιλογή ενός τρόπου αναπαράστασης των καταστάσεων των νευρώνων ώστε να αντιστοιχούν σε στοιχεία της λύσης του προβλήματος.
- Επιλογή της συνάρτησης ενέργειας έτσι ώστε η μικρότερη τιμή της να αντιστοιχεί στο ολικό ελάχιστο της αντικειμενικής συνάρτησης του προβλήματος. Ένας γενικός κανόνας, που ακολουθείται για την επίτευξη αυτής της αντιστοιχίας, είναι η εισαγωγή επιπρόσθετων όρων στη συνάρτηση ενέργειας, οι οποίοι θα «τιμωρούν» την παράβαση των περιορισμών του προβλήματος.
- Εξαγωγή των παραμέτρων του νευρωνικού δικτύου (βάρη και εξωτερικές επιδράσεις) από τη συνάρτηση ενέργειας που ορίστηκε. Οι παράμετροι αυτοί αντιπροσωπεύουν τις εισόδους κάθε στιγμιότυπου του προβλήματος.

Για την περίπτωση του QAP οι νευρώνες αναπαριστούνται από μία δομή πίνακα (έστω T) στην οποία κάθε νευρώνας αναγνωρίζεται από ένα σύνολο δύο δεικτών i και j, που δηλώνουν τη γραμμή και τη στήλη στις οποίες ανήκει, αντίστοιχα. Η έξοδος του νευρώνα (i, j) δηλώνει αν έχει γίνει ή όχι η ανάθεση της εγκατάστασης i στην τοποθεσία j. Για την επίλυση ενός QAP μεγέθους n χρειάζεται ένα νευρωνικό δίκτυο $n \times n$ νευρώνων. Η σχέση εισόδου – εξόδου του κάθε νευρώνα δίνεται από την εξίσωση

$$x_{ij} = g_\lambda(y_{ij})$$

ενώ η δυναμική συμπεριφορά του από τη διαφορική εξίσωση

$$\frac{dy_{ij}}{dt} = -\frac{y_{ij}}{\gamma} + \sum_{k=1}^{n}\sum_{l=1}^{n} T_{ij,lk} x_{lk} + I_{ij}$$

όπου

γ είναι η σταθερά χρόνου του νευρώνα όπως αυτή ορίστηκε από τον Hopfield.

Σύμφωνα με τα παραπάνω η τετραγωνική συνάρτηση ενέργειας ορίζεται ως

$$E(x) = -\frac{1}{2}\sum_{i,j=1}^{n}\sum_{k.l=1}^{n} T_{ij,lk}x_{ij}x_{lk} - \sum_{i,j=1}^{n} I_{ij}x_{ij} + \sum_{i,j=1}^{n}\int_{0.5}^{x} g_{\lambda}^{-1}(x)dx \tag{24}$$

Για τα νευρωνικά δίκτυα αυτού του τύπου η τιμή της συνάρτησης ενέργειας βρίσκεται στο διάστημα $[0,1]$. Συγκεκριμένα κάθε κατάσταση του νευρωνικού δικτύου βρίσκεται σε έναν υπερκύβο n-διαστάσεων που ορίζεται από το σύνολο των ανισοτήτων $0 \le x_{ij} \le 1$ για $i, j = 1,2,...,n$. Έχει αποδειχθεί ότι τα τοπικά ελάχιστα της συνάρτησης ενέργειας βρίσκονται στις 2^n γωνίες του υπερκύβου. Η συνάρτηση, η οποία θα καθορίζει ποιες λύσεις ανήκουν στην περιοχή αναζήτησης του QAP, «τιμωρώντας» όλες τις υπόλοιπες, προσδίδοντας τους μεγαλύτερες τιμές, ονομάζεται συνάρτηση ποινής και είναι η

$$\Pi(x) = \alpha\sum_{i=1}^{n}\left\{\sum_{j=1}^{n} x_{ij} - 1\right\}^2 + \beta\sum_{\xi=1}^{n}\left\{\sum_{\iota=1}^{n} x_{ij} - 1\right\}^2 + \gamma\left\{\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} - n\right\}^2 + \sum_{i=1}^{n}\sum_{j=1}^{n}\xi_{ij}x_{ij}(1 - x_{ij})$$

Οι τρεις πρώτοι όροι της συνάρτησης αυτής παίρνουν την τιμή 0 στην περίπτωση αποδεκτών πινάκων μετάθεσης, σύμφωνα με τις εξισώσεις (4), (5) και (6). Επειδή όμως είναι δυνατό να μηδενιστούν και σε άλλες περιπτώσεις, προστέθηκε ο τέταρτος όρος ώστε να εξασφαλίζει ότι η τελική λύση θα αποτελείται μόνο από δυαδικές τιμές. Έτσι αποφεύγεται η περίπτωση, η συνάρτηση ποινής να μηδενιστεί χωρίς να έχουμε πίνακες μετάθεσης. Άρα

$$\Pi(x) = \begin{cases} 0 & \text{αν ο } x \text{ είναι πίνακας μετάθεσης} \\ 1 & \text{διαφορετικά} \end{cases} \tag{25}$$

Η συνάρτηση ποινής προστίθεται στη συνάρτηση ενέργειας του νευρωνικού δικτύου ώστε το άθροισμα που προκύπτει να δίνει τις μικρότερες τιμές του, όταν και μόνο όταν παίρνει ως είσοδο πίνακες μετάθεσης. Μετά την απλοποίηση των περιορισμών με τη χρησιμοποίηση της συνάρτησης ποινής το QAP μπορεί να εκφραστεί ως

$$\min_{x}\left(-\frac{1}{2}\sum_{i,j=1}^{n}\sum_{k.l=1}^{n} T_{ij,lk}x_{ij}x_{lk} - \sum_{i,j=1}^{n} I_{ij}x_{ij}\right), \quad x_{ij} \in \{0,1\} \tag{26}$$

Συγκρίνοντας τις εξισώσεις (24) και (26) υπολογίζονται οι τιμές του πίνακα συνδεσμολογίας T και του πίνακα εξωτερικών επιδράσεων I του νευρωνικού δικτύου που θα αποτελέσουν τις παραμέτρους της μεθόδου. Το νευρωνικό δίκτυο θα σταθεροποιηθεί σε μια κατάσταση, η οποία θα αντιστοιχεί σε ένα σημείο της περιοχής αναζήτησης του στιγμιότυπου του QAP που ορίζεται από τις παραμέτρους που υπολογίστηκαν. Στη λύση αυτή μπορεί να εφαρμοστεί μια μέθοδος βελτιστοποίησης για

την εύρεση του τοπικού ελαχίστου της περιοχής στην οποία ανήκει. Για να γίνει αυτό χρειάζεται να οριστεί μια δομή γειτνίασης και να εκτελεστεί τοπική αναζήτηση.

Το πλεονέκτημα των νευρωνικών δικτύων Hopfield είναι ότι ο χρόνος σύγκλισης στην τελική τους κατάσταση δεν αυξάνεται γρήγορα σε σχέση με το μέγεθος του προβλήματος. Το μεγάλο τους όμως μειονέκτημα είναι ότι παρόλο που βρίσκουν λύσεις κοντά σε τοπικά ελάχιστα της περιοχής αναζήτησης, σπάνια η συνάρτηση ενέργειας συγκλίνει στο ολικό ελάχιστο της αντικειμενικής συνάρτησης του προβλήματος συνδυαστικής βελτιστοποίησης και κανένας τρόπος βελτίωσης της απόδοσής τους δεν έχει προταθεί μέχρι τώρα. Επίσης, μεγάλη δυσκολία παρουσιάζεται στην επίλυση στιγμιότυπων του QAP μεγάλου μεγέθους, λόγω της μεγάλης αύξησης των νευρώνων και των συνδέσεών τους. Για ένα πρόβλημα μεγέθους n χρειάζονται n^2 νευρώνες και n^4 συνδέσεις μεταξύ τους.

6.9 Χαοτική βελτιστοποίηση (Chaotic optimization – CO)

Από τη δεκαετία του 60, όταν και άρχισε να αναπτύσσεται η επιστήμη του Χάους, άρχισε και η βαθμιαία συνειδητοποίηση ότι πολύ απλές μαθηματικές εξισώσεις μπορούν να απεικονίζουν βίαια και απρόβλεπτα φαινόμενα που μοιάζουν τυχαία. Ελάχιστες διαφορές στη είσοδο ενός συστήματος μπορούσαν γρήγορα, κατά την εξέλιξη του φαινομένου, να μετατραπούν σε τεράστιες διαφορές στην έξοδό του – ένα φαινόμενο που πήρε το όνομα «ευαίσθητη εξάρτηση από τις αρχικές συνθήκες». Οι πρωτοπόροι της επιστήμης αυτής της απέδωσαν το όνομα: «Χάος και Πολυπλοκότητα». Ο όρος «πολυπλοκότητα» δεν αναφέρεται στο μεγάλο αριθμό σωμάτων ή στους πολλούς βαθμούς ελευθερίας καθενός απ' αυτά ώστε το σύστημα που αποτελούν να παρουσιάζει χαοτική συμπεριφορά. Αναφέρεται στην πολυπλοκότητα που διέπει την ίδια τη φύση του φαινομένου και εξαρτάται από τις μαθηματικές εξισώσεις που περιγράφουν τη δυναμική συμπεριφορά του στο χώρο και στο χρόνο.

Το Χάος περιγράφεται ως στοχαστική συμπεριφορά που εκδηλώνεται σε ένα ντετερμινιστικό σύστημα. Στις περιοχές που το σύστημα παρουσιάζει χαοτική συμπεριφορά μικρές αλλαγές στα αίτια μπορούν να οδηγήσουν σε μεγάλες αλλαγές στα αποτελέσματα. Ένα χαρακτηριστικό παράδειγμα χαοτικής συμπεριφοράς είναι το φαινόμενο της πεταλούδας, που χτυπάει τα φτερά της σε ένα μέρος του κόσμου και αυτή η ενέργειά της είναι υπαίτια για τη πρόκληση τυφώνα σε ένα άλλο μέρος το κόσμου. Όμως ποιά φαινόμενα είναι χαοτικά και ποιά όχι; Μια τέτοια διάκριση μεταξύ των φαινομένων που απαντώνται στη φύση είναι λανθασμένη. Η σωστή διατύπωση αυτής της ερώτησης είναι πότε ένα σύστημα παρουσιάζει κανονική συμπεριφορά και πότε χαοτική. Αυτό σημαίνει πως τα συστήματα άλλοτε παρουσιάζουν προβλέψιμη συμπεριφορά (κανονική, περιοδική, ημιπεριοδική) ενώ άλλοτε γίνονται χαοτικά και δεν μπορεί να προβλεφθεί κάθε μελλοντική τους κατάσταση.

Τη χαοτική αυτή συμπεριφορά του μοντέλου που δημιούργησαν εκμεταλλεύονται οι Tohru Ikeguchi, Keiichi Sato, Mikio Hasegawa και Kazuyuki Aihara [60], προτείνοντας μια νέα μέθοδο επίλυσης του QAP. Η μοντελοποίηση του προβλήματος είναι παρόμοια με αυτή των νευρωνικών δικτύων Hopfield, που παρουσιάστηκαν στην προηγούμενη παράγραφο. Με μια βασική διαφορά όμως. Η είσοδος κάθε νευρώνα (εξίσωση (21)) διαφοροποιείται, ώστε να περιγράφεται από μια χαοτική συνάρτηση. Από μια συνάρτηση, δηλαδή, η οποία είναι μη γραμμική και για κάποια διαστήματα του

πεδίου ορισμού της, δίνει τιμές, οι οποίες εξαρτώνται πολύ από τις εισόδους της. Έτσι η δυναμική συμπεριφορά του κάθε νευρώνα, παρόλο που για ένα σύνολο των τιμών εισόδου είναι η αναμενόμενη, για κάποιες περιοχές τιμών είναι χαοτική. Αυτό έχει ως αποτέλεσμα, στις περιοχές αυτές η έξοδος του νευρώνα να είναι πολύ ευπαθής σε μικρές αλλαγές της εισόδου του. Η χαοτική αυτή συμπεριφορά του κάθε νευρώνα δίνει στο σύνολο του νευρωνικού δικτύου τη δυνατότητα να διαφεύγει από τοπικά ελάχιστα της περιοχής αναζήτησης του QAP. Όπως παρατηρούν οι ερευνητές, που ανέπτυξαν τη μέθοδο αυτή, οι διαταραχές στις παραμέτρους περιγραφής του συστήματος στο χώρο των καταστάσεων (state space representation) πιθανόν οδηγούν στην εύρεση του ολικού ελάχιστου της αντικειμενικής συνάρτησης.

Με τη μέθοδο αυτή κατάφεραν να λύσουν αρκετά στιγμιότυπα του QAP με μέγεθος $n \le 30$. Σε μεγαλύτερα προβλήματα αντιμετώπισαν πρόβλημα στη μοντελοποίηση του προβλήματος, λόγω της μεγάλης αύξησης του αριθμού νευρώνων και συνδέσμων. Η σχεδίαση και υλοποίηση νευρωνικών δικτύων με μερικές εκατοντάδες νευρώνες και μερικές χιλιάδες συνάψεις μεταξύ αυτών δεν είναι εύκολη υπόθεση, είτε γίνεται μέσω ηλεκτρονικών κυκλωμάτων πολύ μεγάλης ολοκλήρωσης (very large scale integration – VLSI), είτε γίνεται μέσω προσομοίωσής τους με λογισμικό.

6.10 Μέθοδος άπληστης τυχαίας προσαρμόσιμης αναζήτησης (greedy randomized adaptive search procedure – GRASP)

Η μέθοδος άπληστης τυχαίας προσαρμόσιμης αναζήτησης είναι μια επαναληπτική, ευρετική μέθοδος για την επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης και ως εκ τούτου και του QAP. Ένα πλήθος ανεξάρτητων επαναλήψεων GRASP εκτελείται και η καλύτερη λύση μεταξύ όλων των επαναλήψεων αποτελεί την έξοδο του αλγορίθμου. Η μέθοδος συνδυάζει τη φιλοσοφία των άπληστων αλγορίθμων και της τυχαίας αναζήτησης με μια διαδικασία προσαρμογής της λειτουργίας της στα μέχρι στιγμής πεπραγμένα και αποφάσεις που έχει υλοποιήσει.

Σε κάθε επανάληψη του GRASP υπάρχουν δύο φάσεις. Η φάση κατασκευής και η φάση τοπικής αναζήτησης. Στην πρώτη φάση, εκτελείται ένας άπληστος αλγόριθμος τυχαίας επιλογής, σκοπεύοντας στη δημιουργία λύσεων που δίνουν μικρές τιμές στην αντικειμενική συνάρτηση του προβλήματος. Στη δεύτερη φάση, η γειτονιά της λύσης που βρέθηκε στη στην πρώτη φάση διερευνάται για πιθανές καλύτερες λύσεις. Σχηματικά μπορούμε να φανταστούμε την πρώτη φάση ως τη δημιουργία μιας κουκίδας σε όσο το δυνατόν ευνοϊκότερη θέση στην επιφάνεια αναζήτησης και τη δεύτερη φάση ως τη διαδικασία αναζήτησης της γειτονιάς αυτής της κουκίδας και εύρεσης του τοπικού ελαχίστου που υπάρχει εκεί. Τόσο η αναζήτηση στη δεύτερη φάση όσο και το τοπικό ελάχιστο εξαρτώνται από τον ορισμό της γειτονιάς μιας λύσης.

Στην περίπτωση του QAP, η φάση κατασκευής περιλαμβάνει δύο στάδια. Στο πρώτο στάδιο πραγματοποιούνται δύο αναθέσεις. Για παράδειγμα η εγκατάσταση i τοποθετείται στη θέση k και η εγκατάσταση j τοποθετείται στη θέση l. Αν αναπαραστήσουμε τις εγκαταστάσεις με τις κορυφές ενός γράφου, με πίνακα γειτνίασης τον F και τις τοποθεσίες με τις κορυφές ενός γράφου, με πίνακα γειτνίασης τον D, τότε το πρώτο στάδιο χαρακτηρίζεται από την αντιστοίχηση δύο κορυφών του δεύτερου γράφου σε δύο κορυφές του πρώτου. Για την πραγματοποίηση των αναθέσεων αυτών,

αρχικά ταξινομούνται οι ροές (flows) μεταξύ των εγκαταστάσεων σε φθίνουσα σειρά και οι αποστάσεις (distances) μεταξύ των τοποθεσιών σε αύξουσα σειρά:

$$f_{i_1,j_1} \geq f_{i_2,j_2} \geq \ldots \geq f_{i_p,j_p}$$
$$d_{k_1,l_1} \leq d_{k_2,l_2} \leq \ldots \leq d_{k_p.l_p}$$

όπου $p = n(n-1)$ και n το μέγεθος του προβλήματος. Τα γινόμενα που προκύπτουν είναι

$$f_{i_1,j_1} \cdot d_{k_1,l_1}, f_{i_2,j_2} \cdot d_{k_2,l_2}, \ldots, f_{i_p,j_p} \cdot d_{k_p,l_p} \tag{27}$$

Το κάθε γινόμενο $f_{i_s,j_s} \cdot d_{k_s,l_s}$ αντιπροσωπεύει το κόστος των δύο αναθέσεων που επιλέγονται και εφαρμόζονται με την τοποθέτηση της εγκατάστασης i_s στη θέση k_s και της εγκατάστασης j_s στη θέση l_s με $1 \leq s \leq p$. Στην περίπτωση των γράφων, το κόστος αυτό ισούται με το γινόμενο των βαρών των ακμών των δύο γράφων που προσπίπτουν στις κορυφές που αντιστοιχήθηκαν. Τα γινόμενα αυτά ταξινομούνται σε αύξουσα σειρά. Το πλήθος τους είναι p. Τα πρώτα ap γινόμενα με το μικρότερο κόστος, όπου $a \in [0,1]$ είναι μια παράμετρος του αλγορίθμου, εισάγονται στην περιορισμένη λίστα υποψηφιοτήτων (restricted candidate list – RLC). Ένα από τα ap στοιχεία της λίστας επιλέγεται τυχαία ως η έξοδος του πρώτου σταδίου.

Στο δεύτερο στάδιο της πρώτης φάσης του αλγορίθμου, πραγματοποιούνται οι υπόλοιπες $n-2$ αναθέσεις. Οι εναπομείνασες από το πρώτο στάδιο εγκαταστάσεις τοποθετούνται διαδοχικά σε ελεύθερες θέσεις. Σε κάθε βήμα της επανάληψης ένα πλήθος αναθέσεων έχει ήδη γίνει, έστω m, και $n-m$ αναθέσεις πρόκειται να γίνουν με την πρώτη να πραγματοποιείται στο βήμα αυτό. Αν δηλώσουμε ως Γ_r το σύνολο των ζευγών των αναθέσεων που έχουν ήδη γίνει πριν από την r-ιοστή ανάθεση, τότε

$$\Gamma_r = \{(i_1,k_1),(i_2,k_2),\ldots,(i_{r-1},k_{r-1})\} \tag{28}$$

Ας σημειωθεί ότι ο αριθμός των στοιχείων (cardinality) του συνόλου Γ_r δείχνει το πλήθος των αναθέσεων που έχουν ήδη γίνει. Στην αρχή του δεύτερου σταδίου της πρώτης φάσης ισχύει $|\Gamma_3| = 2$, αφού εξ΄ ορισμού στο πρώτο στάδιο έχουν γίνει δύο αναθέσεις. Επιπλέον, σε κάθε βήμα της επανάληψης του δεύτερου σταδίου, έστω r, ισχύει $|\Gamma_r| = r-1$.

Σε κάθε ένα απ' αυτά τα βήματα πραγματοποιείται μια νέα ανάθεση, η οποία επιλέγεται, όπως και στο πρώτο στάδιο, από μια περιορισμένη λίστα υποψηφιοτήτων (RCL). Η λίστα αυτή περιέχει κάθε ανάθεση $j \to l$ με $(j,l) \notin \Gamma_r$, που αντιστοιχεί στην τοποθέτηση της εγκατάστασης j στη θέση l. Το κόστος κάθε μιας από αυτές τις υποψήφιες αναθέσεις, σε σχέση με αυτές που έχουν ήδη γίνει, είναι

$$c_{jl} = \sum_{(i,k)\in\Gamma_r} f_{ij} d_{kl} \tag{29}$$

Στο βήμα της επανάληψης, στο οποίο έχουν γίνει ήδη $r-1$ αναθέσεις, το σύνολο των εφικτών αναθέσεων $j \to l$ είναι $N = (n-r+1)^2$. Από αυτές οι aN εισάγονται στην RLC, όπου $a \in [0,1]$ είναι η παράμετρος του αλγορίθμου που χρησιμοποιήθηκε και στο πρώτο στάδιο. Τέλος, μια ανάθεση από την RLC επιλέγεται τυχαία και το σύνολο Γ_r ενημερώνεται

$$\Gamma_r = \Gamma_r \cup \{(j,l)\}$$

Η διαδικασία επαναλαμβάνεται μέχρις ότου γίνουν $n-1$ αναθέσεις. Η πραγματοποίηση της τελευταίας ανάθεσης είναι τετριμμένη διαδικασία. Η εναπομείνασα εγκατάσταση τοποθετείται στην εναπομείνασα θέση.

Η δεύτερη φάση, που ολοκληρώνει την κάθε επανάληψη του GRASP, είναι η φάση της τοπικής αναζήτησης. Ορίζεται μια δομή γειτνίασης, που στην περίπτωση του QAP είναι αυτή που προκύπτει από την αμοιβαία ανταλλαγή όλων των ζευγών τιμών της μετάθεσης (pair-exchange neighborhood). Επίσης, ορίζεται μια μέθοδος ενημέρωσης της υπάρχουσας λύσης, όπως περιγράφηκε στην παράγραφο των μεθόδων βελτιστοποίησης. Έτσι, για κάθε λύση – μετάθεση που παράγεται από τη πρώτη φάση του αλγορίθμου, αναζητείται στη γειτονιά της μια καλύτερη λύση που αποτελεί και ένα τοπικό ελάχιστο της περιοχής αναζήτησης.

Οι παράμετροι του αλγορίθμου που καθορίζουν τη λειτουργία του είναι:

- ο πραγματικός αριθμός $a \in [0,1]$, από τον οποίο υπολογίζεται δυναμικά το μέγεθος της περιορισμένης λίστας υποψηφιοτήτων (RLC)
- ένας αριθμός που δηλώνει το μέγιστο αριθμό επαναλήψεων (max iterations) και αποτελεί ένα κριτήριο τερματισμού
- η αρχική τιμή της συνάρτησης παραγωγής ψευδοτυχαίων αριθμών (seed)

Ας προσπαθήσουμε τώρα να δικαιολογήσουμε την ονομασία της μεθόδου αυτής. Μέθοδος άπληστης, τυχαίας, προσαρμόσιμης αναζήτησης (greedy randomized adaptive search procedure – GRASP). Είναι άπληστη γιατί σε κάθε βήμα της φάσης κατασκευής τα κόστη των υποψήφιων αναθέσεων ταξινομούνται σε αύξουσα σειρά και αυτά με το μικρότερο κόστος έχουν μεγαλύτερη πιθανότητα να επιλεγούν. Είναι τυχαία γιατί δεν επιλέγεται πάντα η ανάθεση με το μικρότερο κόστος αλλά επιλέγεται με τυχαίο τρόπο μία από αυτές με τα μικρότερα κόστη. Είναι προσαρμόσιμη γιατί μετά από την επιλογή κάθε νέας ανάθεσης τα κόστη για τις υπόλοιπες υποψήφιες αναθέσεις ανανεώνονται υπολογίζοντας και την τελευταία ανάθεση που έγινε.

Η παράμετρος $a \in [0,1]$, που χρησιμοποιείται στην πρώτη φάση του αλγορίθμου, καθορίζει το πόσο άπληστη και το πόσο τυχαία θα είναι η μέθοδος. Αν $a = 0$, τότε σε κάθε βήμα της επανάληψης επιλέγεται η ανάθεση με το μικρότερο κόστος. Η μέθοδος είναι αμιγώς άπληστη, αφού σε κάθε βήμα κάνει την βραχυπρόθεσμα καλύτερη επιλογή. Οι αναθέσεις με μεγαλύτερα κόστη δεν έχουν καμιά ελπίδα επιλογής. Τίποτα δεν αφήνεται στην τύχη. Αν η παράμετρος $a = 1$, τότε η μέθοδος είναι αμιγώς τυχαία και καθόλου άπληστη. Κάθε ανάθεση της περιορισμένης λίστας υποψηφιοτήτων μπορεί να επιλεγεί με την ίδια πιθανότητα. Σε όλα τα βήματα της επανάληψης δεν υπάρχουν λύσεις

που να χαρακτηρίζονται ως καλύτερες ή χειρότερες. Όλες είναι ίσες απέναντι στη διαδικασία επιλογής. Τέλος, αν $0 < a < 1$, η μέθοδος είναι και άπληστη και τυχαία. Είναι άπληστη γιατί υποψήφιες για επιλογή δεν είναι όλες οι εναπομείνασες αναθέσεις, αλλά μόνο οι aN με το μικρότερο κόστος, όπου N είναι το πλήθος τους. Είναι τυχαία γιατί ανάμεσα από αυτές τις aN αναθέσεις, δεν επιλέγεται πάντα αυτή με το μικρότερο κόστος, αλλά όλες έχουν την ίδια πιθανότητα επιλογής.

Αφήσαμε τελευταία την περιγραφή του αλγορίθμου GRASP για τον απλούστατο λόγο ότι η νέα μέθοδος που ακολουθεί, βασίζεται σ' αυτόν. Ο αλγόριθμος, που περιγράφεται στη συνέχεια αυτής της εργασίας, είναι μια μέθοδος άπληστης, τυχαίας, προσαρμόσιμης αναζήτησης, που όμως διαφέρει σε κάποια βασικά στοιχεία από τη μέθοδο αυτής της παραγράφου.

7. Μια νέα απόπειρα επίλυσης του Τετραγωνικού Προβλήματος Ανάθεσης

Στις προηγούμενες ενότητες αυτής της εργασίας παρουσιάστηκαν οι προσπάθειες που έχουν γίνει για την επίλυση του Τετραγωνικού Προβλήματος Ανάθεσης. Έχουν αναπτυχθεί από τους ερευνητές τόσο μέθοδοι εξαντλητικής αναζήτησης, που βρίσκουν τη βέλτιστη λύση κάθε στιγμιότυπου του προβλήματος, όσο και προσεγγιστικές τεχνικές εύρεσης λύσεων, που δεν είναι απαραίτητα οι καλύτερες, για χάρη του υπολογιστικού χρόνου εκτέλεσης. Η δυσκολία που παρουσιάζει εκ φύσεως το QAP, είναι οι αιτία που οι πρώτες από αυτές τις μεθόδους δεν μπορούν να λύσουν στιγμιότυπα με μέγεθος μεγαλύτερο από 30 ($n > 30$) σε λογικό υπολογιστικό χρόνο και οι δεύτερες δεν μπορούν να ανιχνεύσουν πάντα το ολικό ελάχιστο της αντικειμενικής συνάρτησης και άρα τη βέλτιστη λύση του προβλήματος. Ως εκ τούτου, η προσπάθεια χρειάζεται να συνεχιστεί.

7.1 Άπληστοι αλγόριθμοι

Στην ενότητα αυτή περιγράφεται μια νέα απόπειρα επίλυσης του QAP. Πρόκειται για μια μέθοδο άπληστης, τυχαίας, προσαρμόσιμης αναζήτησης (greedy randomized adaptive search procedure – GRASP), όπως αυτή που παρουσιάστηκε στην προηγούμενη παράγραφο. Σε σχέση με την τελευταία όμως έχει μια βασική διαφορά. Ο αλγόριθμος της προηγούμενης παραγράφου χρησιμοποιεί **απληστία εισόδου** (**greedy in**). Αυτός που θα περιγραφεί παρακάτω χρησιμοποιεί **απληστία εξόδου** (**greedy out**).

Και στις δύο περιπτώσεις ο αλγόριθμος, σε κάθε βήμα της επαναληπτικής του διαδικασίας, κάνει την βραχυπρόθεσμα καλύτερη επιλογή, ελπίζοντας ότι η τακτική αυτή θα τον οδηγήσει στη βέλτιστη λύση του προβλήματος. Δρώντας άπληστα, επιλέγει με καθαρά τοπικά κριτήρια. Ένα βασικό χαρακτηριστικό των άπληστων αλγορίθμων είναι η ιδιότητα της άπληστης επιλογής: μια τοπικά βέλτιστη επιλογή είναι δυνατό να οδηγήσει σε μια καθολικά βέλτιστη λύση. Δυστυχώς κάτι τέτοιο δε συμβαίνει πάντα. Για να ισχύει, πρέπει το πρόβλημα προς λύση να έχει την ιδιότητα της βέλτιστης υποδομής: κάθε βέλτιστη λύση του προβλήματος εμπεριέχει βέλτιστες λύσεις υποπροβλημάτων.

Εκτός από το γενικό πλαίσιο, που είναι κοινό και στις δύο εκδοχές άπληστου αλγορίθμου, υπάρχει μια βασική διαφορά μεταξύ της απληστίας εισόδου και της απληστίας εξόδου. Οι τεχνικές, που βασίζονται στην πρώτη, ξεκινούν με ένα υποσύνολο της λύσης. Σε κάθε βήμα επιλέγουν το στοιχείο που θεωρούν καλύτερο για να εισαχθεί στο υποσύνολο. Αρχικά, το υποσύνολο είναι κενό. Επαναληπτικά αυξάνεται το πλήθος

των στοιχείων του. Κατά την επαναληπτική διαδικασία, τα υποψήφια προς εισαγωγή στοιχεία είναι αυτά που οδηγούν σε υποσύνολα επιτρεπτών λύσεων. Κάθε στοιχείο, που εισάγεται, επιλέγεται με καθαρά τοπικά κριτήρια και η επιλογή του αμέσως αποκλείει την επιλογή άλλων στοιχείων. Τελικά, στη δομή, που αποτελούσε το υποσύνολο, υπάρχουν ακριβώς τα στοιχεία που αποτελούν μια ολοκληρωμένη λύση του προβλήματος.

Σε αντίθεση με τα παραπάνω, οι τεχνικές που βασίζονται στην απληστία εξόδου ξεκινούν με ένα υπερσύνολο της λύσης. Σε κάθε βήμα επιλέγουν το στοιχείο που θεωρούν καλύτερο για να διαγραφεί από το υπερσύνολο. Το πλήθος των στοιχείων του υπερσυνόλου, αρχικά, είναι μεγαλύτερο από το πλήθος των στοιχείων μιας εφικτής λύσης και επαναληπτικά μειώνεται. Κάθε φορά, τα υποψήφια προς διαγραφή στοιχεία είναι αυτά που οδηγούν σε εφικτά υπερσύνολα, δηλαδή σε υπερσύνολα που περιέχουν μια τουλάχιστον λύση του προβλήματος. Κάθε στοιχείο που διαγράφεται, επιλέγεται καθαρά με τοπικά κριτήρια και η επιλογή του μπορεί να αποκλείει την επιλογή άλλων στοιχείων, καθιστώντας έτσι αδύνατη τη διαγραφή τους. Τα στοιχεία, που δεν μπορούν να διαγραφούν, σίγουρα θα συμμετέχουν στη λύση. Τελικά στη δομή, που αποτελούσε το υπερσύνολο, υπάρχουν ακριβώς τα στοιχεία, που αποτελούν μια ολοκληρωμένη λύση του προβλήματος.

7.2 Θεωρητική βάση

Τα **matroid** επινοήθηκαν από τον Hassler Whitney [101, 102] το 1935. Ο δημιουργός τους είναι αυτός που τους έδωσε και το όνομά τους, η ελληνική απόδοση του οποίου είναι «πινακοειδή». Ένα matroid (**πινακοειδές**) είναι ένα σύνολο στο οποίο ορίζεται μια δομή ανεξαρτησίας. Πρόκειται για τη γενίκευση της έννοιας της γραμμικής ανεξαρτησίας στο διανυσματικό χώρο. Υπάρχουν πολλοί ισοδύναμοι τρόποι ορισμού ενός matroid, οι οποίοι οδηγούν σε ποικίλες μαθηματικές περιγραφές. Ένας, από τους πιο συχνά χρησιμοποιούμενους, είναι αυτός που ορίζει ένα matroid σε σχέση με την έννοια της ανεξαρτησίας.

Ορισμός 1: *Ένα πεπερασμένο matroid* M *είναι το ζευγάρι* (E, I)*, όπου το* E *είναι ένα πεπερασμένο σύνολο και το* I *μια συλλογή από υποσύνολα του* E*, που καλούνται ανεξάρτητα και έχουν τα παρακάτω αξιώματα*:

- *Το κενό σύνολο είναι ανεξάρτητο. Αυτό ισοδυναμεί με το γεγονός ότι τουλάχιστον ένα υποσύνολο του* E *είναι ανεξάρτητο.*
- *Κάθε υποσύνολο ενός ανεξάρτητου υποσυνόλου είναι επίσης ανεξάρτητο. Η έννοια αυτή αναφέρεται και ως κληρονομική ιδιότητα.*
- *Αν* A *και* B *είναι δύο ανεξάρτητα σύνολα και το* A *έχει περισσότερα στοιχεία από το* B*, τότε υπάρχει ένα* $x \in A \setminus B$ *τέτοιο ώστε* $B \bigcup \{x\} \in I$*. Αναφέρεται συνήθως ως αυξητική ιδιότητα ή ιδιότητα ανταλλαγής ανεξάρτητων συνόλων.*

■

Τα δύο πρώτα αξιώματα, που είναι και οι πιο απλές, από μόνες τους ορίζουν μια συνδυαστική δομή, γνωστή ως **ανεξάρτητο σύστημα** (**independent system**). Ένα υποσύνολο του E, που δεν είναι ανεξάρτητο, ονομάζεται εξαρτημένο.

Ένας δεύτερος ορισμός των matroid μπορεί να γίνει μέσω των βάσεων που αυτά περιέχουν. Ένα σύνολο αποτελεί βάση του matroid M, εάν είναι μείζον ανεξάρτητο, δηλαδή αν είναι ανεξάρτητο και μετατρέπεται σε εξαρτημένο με την προσθήκη οποιουδήποτε στοιχείου του συνόλου E.

Ορισμός 2: *Ένα πεπερασμένο matroid M είναι το ζευγάρι (E,B), όπου το E είναι ένα πεπερασμένο σύνολο και το B μια συλλογή από υποσύνολα του E, που καλούνται βάσεις και έχουν τα παρακάτω αξιώματα*:

- *Το E περιέχει τουλάχιστον μια βάση.*
- *Κάθε γνήσιο υποσύνολο μιας βάσης δεν είναι βάση.*
- *Αν B_1 και B_2 είναι βάσεις του συνόλου E και $x \in B_1 \setminus B_2$ τότε υπάρχει τουλάχιστον ένα $y \in B_2 \setminus B_1$, τέτοιο ώστε το σύνολο $(B_1 \setminus \{x\}) \cup \{y\}$ να αποτελεί επίσης βάση του E.*

■

Ένας διαφορετικός ορισμός των matroid στα πλαίσια των Προβλημάτων Συνδυαστικής Βελτιστοποίησης έχει δοθεί από τους Χρήστο Παπαδημητρίου και Kenneth Steiglitz [82].

Ορισμός 3: *Έστω $M = (E,g)$ είναι ένα σύστημα υποσυνόλων. Το M είναι matroid, αν ο άπληστος αλγόριθμος λύνει ορθά το Πρόβλημα Συνδυαστικής Βελτιστοποίησης X για οποιαδήποτε συνάρτηση βάρους σχετίζεται με το M*
όπου

το σύστημα υποσυνόλων (subset system) (E,g) αποτελείται από ένα πεπερασμένο σύνολο E και μια συλλογή υποσυνόλων για τα οποία ισχύει: αν $A \in g$ και $B \subseteq A$ τότε $B \in g$

το Πρόβλημα Συνδυαστικής Βελτιστοποίησης X ορίζεται ως η εύρεση του υποσυνόλου του g με το μεγαλύτερο συνολικό βάρος, αν στο E οριστεί μια πραγματική, μη αρνητική συνάρτηση βάρους $w : E \to \Re_+$

ο άπληστος αλγόριθμος, για την κατασκευή του επιθυμητού υποσυνόλου του g, ξεκινάει με ένα κενό σύνολο I, επαναληπτικά προσθέτει σ' αυτό το στοιχείο e, για το οποίο ισχύει $w(e) \geq w(f)$ για κάθε $f \in E$ και $I \cup \{e\} \in g$, και τελικά τερματίζει, όταν προσπελάσει όλα τα στοιχεία του E

■

Αφού δόθηκαν οι ορισμοί των matroid και των ανεξάρτητων συστημάτων, είναι δυνατή πλέον η περιγραφή των δύο τύπων άπληστων αλγορίθμων, μέσω αυτών. Η απληστία εισόδου χρησιμοποιεί ένα ανεξάρτητο σύστημα (E,I), το οποίο βασίζεται στον πρώτο ορισμό και μια διαδικασία απόφασης, ως μαύρο κουτί, για το αν ένα υποσύνολο του E, $F \subseteq E$, είναι ανεξάρτητο, $F \in I$, ή όχι, $F \notin I$. Η μέθοδος της απληστίας εισόδου μπορεί να περιγραφεί με τον παρακάτω ψευδοκώδικα.

Αλγόριθμος απληστίας εισόδου
Είσοδος: Ένα ανεξάρτητο σύνολο (E, I)
Βάρη $w: E \to \Re_+$
Έξοδος: Ένα σύνολο $F \in I$

1. ταξινόμησε τα στοιχεία του συνόλου $E = \{e_1, e_2, ..., e_n\}$ σε μη αύξουσα σειρά των βαρών τους, έτσι ώστε να ισχύει $c(e_1) \geq c(e_2) \geq ... \geq c(e_n)$.
2. αρχικοποίησε το F με το κενό σύνολο, $F = \emptyset$.
3. for $i = 1 \ldots n$
    a. if $F \cup \{e_i\} \in I$
        i. $F = F \cup \{e_i\}$
    b. end if
4. end for

Για τη μέθοδο της απληστίας εξόδου χρησιμοποιείται ο δεύτερος ορισμός των matroid, που κάνει χρήση των μειζόνων ανεξάρτητων συνόλων. Μια διαδικασία, ως μαύρο κουτί, αποφαίνεται, αν το σύνολο $F \subseteq E$ περιέχει μια βάση ή όχι. Αν με $B(E)$ δηλώσουμε το σύνολο των βάσεων του E, τότε η συλλογή ανεξάρτητων υποσυνόλων I στον πρώτο ορισμό του matroid είναι

$$I = \{x \subseteq E \mid \exists B \in B(E), x \subseteq B\}$$

Ο ψευδοκώδικας, που περιγράφει την απληστία εξόδου, φαίνεται παρακάτω.

Αλγόριθμος απληστίας εξόδου
Είσοδος: Ένα ανεξάρτητο σύνολο (E, I)
Βάρη $w: E \to \Re_+$
Έξοδος: Μια βάση F του (E, I)

1. ταξινόμησε τα στοιχεία του συνόλου $E = \{e_1, e_2, ..., e_n\}$ σε μη φθίνουσα σειρά των βαρών τους, έτσι ώστε να ισχύει $c(e_1) \leq c(e_2) \leq ... \leq c(e_n)$.
2. αρχικοποίησε $F = E$.
3. for $i = 1 \ldots n$
    a. if $F \setminus \{e_i\}$ περιέχει μια βάση
        i. $F = F \setminus \{e_i\}$
    b. end if
4. end for

Όπως είναι φανερό από τους παραπάνω αλγορίθμους, η εφαρμογή της μεθόδου απληστίας εισόδου σε προβλήματα μεγιστοποίησης, ισοδυναμεί με την εφαρμογή της μεθόδου απληστίας εξόδου σε προβλήματα ελαχιστοποίησης. Σε κάθε βήμα της

επαναληπτικής διαδικασίας, η εισαγωγή του καλύτερου στοιχείου στο σύνολο F στην περίπτωση της απληστίας εισόδου, αντιστοιχεί στη διαγραφή του χειρότερου στοιχείου στην περίπτωση της απληστίας εξόδου. Αυτό ισχύει βέβαια ως προς τα βήματα υλοποίησης των μεθόδων και όχι ως προς την ποιότητα των αποτελεσμάτων, που αυτές παράγουν. Δύο θεωρήματα μας δίνουν ένα μέτρο της ποιότητας των λύσεων, που κατασκευάζουν οι δύο τεχνικές απληστίας.

Θεώρημα 1: *Έστω* (E,I) *ένα ανεξάρτητο σύστημα. Η αντικειμενική συνάρτηση* $G(E,I,w)$ *δίνει το κόστος μιας λύσης, που βρέθηκε με τη μέθοδο απληστίας εισόδου για ένα πρόβλημα μεγιστοποίησης, όπου* $w: E \to \Re_+$ *είναι η συνάρτηση βάρους. Τότε ισχύει*

$$q(E,I) = \min_{F \subseteq E} \frac{\rho(F)}{r(F)} \le \frac{G(E,I,w)}{opt(E,I,w)} \le 1$$

για κάθε $w: E \to \Re_+$. *Η* $opt(E,I,w)$ *είναι η βέλτιστη λύση του προβλήματος και* $q(E,I)$ *είναι το κάτω φράγμα που ορίζεται. Οι* $\rho^*, r^*: 2^E \to Z^+$ *είναι συναρτήσεις βαθμίδος, που αντιπροσωπεύουν την κατώτερη και την ανώτερη βαθμίδα του* (E,F^*), *αντίστοιχα. Οι τιμές τους υπολογίζονται από τους τύπους*

$$\rho(F) = \min\left\{\left|H\right| : H \subseteq F,\ H \in I,\ H \cup \{f\} \notin I \quad \forall f \in F \setminus H\right\}$$
$$r(F) = \max\left\{\left|H\right| : H \subseteq F,\ H \in I\right\}$$

■

Θεώρημα 2: *Έστω* (E,I) *ένα ανεξάρτητο σύστημα. Η αντικειμενική συνάρτηση* $G(E,I,w)$ *δίνει το κόστος μιας λύσης, που βρέθηκε με τη μέθοδο απληστίας εξόδου για ένα πρόβλημα ελαχιστοποίησης, όπου* $w: E \to \Re_+$ *είναι η συνάρτηση βάρους. Τότε ισχύει*

$$1 \le \frac{G(E,I,w)}{opt(E,I,w)} \le \max_{F \subseteq E} \frac{\left|F\right| - \rho^*(F)}{\left|F\right| - r^*(F)}$$

για κάθε $w: E \to \Re_+$. *Η* $opt(E,I,w)$ *είναι η βέλτιστη λύση του προβλήματος.*

■

Για την εφαρμογή της απληστίας εξόδου στο QAP, είναι απαραίτητος ο καθορισμός της δομής, που θα αποτελεί το υπερσύνολο εφικτών λύσεων. Η δομή αυτή, μετά από ένα πλήθος διαγραφών, θα μας δίνει μια άπληστη λύση του προβλήματος.

7.3 Αναπαράσταση των δεδομένων του προβλήματος

Κάθε στιγμιότυπο του QAP χαρακτηρίζεται από το μέγεθός του και από τα στοιχεία των πινάκων F και D, δηλαδή τις τιμές των ροών μεταξύ των εγκαταστάσεων και των αποστάσεων μεταξύ των τοποθεσιών, αντίστοιχα. Η μέθοδος, που χρησιμοποιεί

την απληστία εισόδου, σε κάθε βήμα της επαναληπτικής διαδικασίας, δημιουργεί ένα σύνολο υποψήφιων επιλογών, οι οποίες αποτελούν την περιορισμένη λίστα υποψηφιοτήτων. Οι επιλογές αυτές είναι ένα μόνο μέρος των γινομένων μεταξύ των στοιχείων του ενός πίνακα με τα στοιχεία του άλλου. Κάθε ένα από αυτά τα γινόμενα, ανεξάρτητο από τα υπόλοιπα, αντιπροσωπεύει το κόστος των δύο αναθέσεων, στις οποίες αντιστοιχεί.

Για την υλοποίηση της απληστίας εξόδου χρειάζεται η δημιουργία μιας δομής, η οποία να περιλαμβάνει όλα τα δυνατά γινόμενα μεταξύ των στοιχείων των πινάκων F και D. Έτσι, έχει ως υποψήφιες επιλογές όλες τις δυνατές αναθέσεις. Από το σύνολο αυτών των αναθέσεων – γινομένων, σε κάθε βήμα της επαναληπτικής διαδικασίας, διαγράφεται μία. Υποψήφιες για διαγραφή είναι οι αναθέσεις, που έχουν ένα από τα μεγαλύτερα κόστη. Αν υπάρχουν κάποια γινόμενα, που η διαγραφή τους οδηγεί σε ένα μη εφικτό υπερσύνολο, τότε τα γινόμενα αυτά και οι αναθέσεις που αντιπροσωπεύουν ανήκουν σίγουρα στην άπληστη λύση του προβλήματος. Ένα υπερσύνολο είναι εφικτό, αν περιλαμβάνει κάποια λύση του προβλήματος. Με τον τρόπο αυτό και καθώς οι διαγραφές συνεχίζονται, επιλέγονται τα στοιχεία, που ανήκουν στη λύση του προβλήματος, με την ελπίδα πως η απληστία εξόδου θα οδηγήσει σε μια λύση κοντά στη βέλτιστη, αν όχι στην ίδια τη βέλτιστη.

Μια δομή, που ταιριάζει στη παραπάνω περιγραφή, είναι ο πίνακας $C = (c_{ijkl})$, που παρουσιάστηκε από το Lawler για τη διατύπωση του QAP. Περιλαμβάνει τα γινόμενα που είναι απαραίτητα για την παραπάνω διαδικασία και επίσης παρουσιάζει χαρακτηριστικά, που βοηθούν στην υλοποίηση των λειτουργιών επιλογής και διαγραφής στοιχείων. Για ένα στιγμιότυπο μεγέθους n, ο πίνακας C είναι ένας δισδιάστατος πίνακας μεγέθους $n^2 \times n^2$ και περιλαμβάνει n^4 συντελεστές. Ο καθένας από αυτούς είναι ένα γινόμενο ενός στοιχείου του πίνακα $F = (f_{ij})$ και ενός στοιχείου του πίνακα $D = (d_{kl})$. Έτσι, ο πίνακας C περιλαμβάνει όλα τα υποψήφια προς επιλογή κόστη. Συγκεκριμένα ο συντελεστής c_{ijkl} έχει τιμή $f_{ij} \cdot d_{kl}$ και βρίσκεται στη γραμμή $(i-1)n + j$ και στη στήλη $(k-1)n + l$ του πίνακα C. Βασισμένος στα παραπάνω ο Lawler διατύπωσε το QAP ως ένα Γραμμικό Πρόβλημα Ανάθεσης (Linear Assignment Problem – LAP) με έναν επιπλέον περιορισμό.

$$\min \quad \langle C, Y \rangle$$

έτσι ώστε

$$Y = X \otimes X$$
$$X \in X_n$$

Ο πίνακας X είναι ένας $n \times n$ πίνακας μετάθεσης. Η μετάθεση, που αντιστοιχεί μοναδικά σε αυτόν, αποτελεί μια λύση του προβλήματος. Ο πίνακας Y, μεγέθους $n^2 \times n^2$, είναι το γινόμενο Kronecker του πίνακα X με τον εαυτό του. Το κάθε στοιχείο του πίνακα Y, όπως και του X, είναι ή μηδενικό ή άσσος. Οι θέσεις των άσσων του πίνακα Y υποδεικνύουν τα στοιχεία του πίνακα C που επιλέχθηκαν για τη λύση. Μόνο ένας άσσος υπάρχει σε κάθε γραμμή και κάθε στήλη του πίνακα Y.

Οι πίνακες F και D είναι τετραγωνικοί. Άρα, τετραγωνικοί θα είναι και οι πίνακες X και Y. Ο τελευταίος ορίζεται ως

$$Y = X \otimes X = \begin{bmatrix} x_{11}X & x_{12}X & \cdots & x_{1n}X \\ x_{21}X & x_{22}X & \cdots & x_{2n}X \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1}X & x_{n2}X & \cdots & x_{nn}X \end{bmatrix} = (y_{ijkl}) \tag{30}$$

Τόσο ο πίνακας C, όσο και ο Y, μπορούν να αντιμετωπιστούν είτε ως πίνακες μεγέθους $n^2 \times n^2$ με στοιχεία τους συντελεστές του προβλήματος, είτε ως πίνακες μεγέθους $n \times n$ με στοιχεία υποπίνακες μεγέθους επίσης $n \times n$. Στην περίπτωση του πίνακα C ο κάθε ένας από αυτούς τους υποπίνακες περιλαμβάνει τα γινόμενα των στοιχείων μιας γραμμής του πίνακα F με τα στοιχεία μιας γραμμής του πίνακα D. Το στοιχείο $c_{ijkl} = f_{ij} \cdot d_{kl}$ ανήκει στην j γραμμή του υποπίνακα της γραμμής i και στην l στήλη του υποπίνακα της στήλης k. Στην περίπτωση του πίνακα Y ο κάθε ένας υποπίνακας είτε είναι μηδενικός, είτε ισούται με τον πίνακα X.

Στον πίνακα Y ισχύει

$$y_{ijkl} = x_{ij}x_{kl} = x_{kl}x_{ij} = y_{klij} \tag{31}$$

και άρα κάθε στοιχείο του έχει ένα συμπληρωματικό. Αυτό πρακτικά σημαίνει ότι αν το στοιχείο y_{ijkl} έχει την τιμή 1 και άρα συμμετέχει στη λύση, τότε και το στοιχείο y_{klij} έχει την ίδια τιμή και συμμετέχει επίσης στη λύση. Μια ιδιότητα των συμπληρωματικών ζευγαριών, που συμμετέχουν στη λύση του προβλήματος, είναι ότι βρίσκονται πάντα σε δύο διαφορετικούς υποπίνακες, οι οποίοι ποτέ δεν ανήκουν στην ίδια γραμμή ή στην ίδια στήλη. Αυτό οφείλεται στο γεγονός ότι ο πίνακας Y, αν θεωρηθεί ως πίνακας υποπινάκων, παρουσιάζει το ίδιο πρότυπο, μεγεθυσμένο, με τον πίνακα X, που είναι ένας πίνακας μετάθεσης. Αν ο πίνακας X έχει μηδενικό σε μία θέση, ο πίνακας Y στην αντίστοιχη θέση υποπίνακα έχει το μηδενικό υποπίνακα. Αν ο πίνακας X έχει άσσο σε μια θέση, ο πίνακας Y στην αντίστοιχη θέση υποπίνακα έχει τον πίνακα X. Το παρακάτω παράδειγμα δείχνει όσα ειπώθηκαν.

<u>Παράδειγμα</u>
Αν ο πίνακας μετάθεσης X είναι

$$X = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

τότε ο αντίστοιχος πίνακας Y είναι

$$Y = \left[\begin{array}{ccc:ccc:ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hdashline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Η μετάθεση (permutation), που αντιστοιχεί στους παραπάνω πίνακες και αποτελεί μια λύση του προβλήματος, είναι

$$p = (2,3,1)$$

■

Τα στοιχεία του πίνακα Y που έχουν την τιμή 1 είναι αυτά που συμμετέχουν στη λύση. Το άθροισμα των αντίστοιχων στοιχείων του πίνακα C ισούται με την τιμή της αντικειμενικής συνάρτησης για τη λύση αυτή. Η τιμή 1 σε μια θέση του πίνακα Y σημαίνει την επιλογή του αντίστοιχου στοιχείου του πίνακα C. Αν το στοιχείο c_{ijkl} έχει επιλεγεί, τότε έχει γίνει η ανάθεση της εγκατάστασης i στην τοποθεσία k και της εγκατάστασης j στην τοποθεσία l. Παρακάτω φαίνεται ένα απλοϊκό στιγμιότυπο του QAP. Παρουσιάζονται οι πίνακες F και D καθώς και ο πίνακας C που προκύπτει από αυτούς. Στον τελευταίο είναι σημειωμένα τα στοιχεία που έχουν επιλεγεί σύμφωνα με τη μετάθεση και τον πίνακα Y του προηγούμενου παραδείγματος.

Παράδειγμα

Οι πίνακες F και D ενός στιγμιότυπου του QAP μεγέθους $n = 3$ είναι:

$$F = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{bmatrix} \text{ και } D = \begin{bmatrix} 0 & 6 & 4 \\ 6 & 0 & 5 \\ 4 & 5 & 0 \end{bmatrix}$$

Ο πίνακας C που προκύπτει από αυτούς είναι:

		$x_j=1$			$x_j=2$			$x_j=3$		
		$c_j=1$	$c_j=2$	$c_j=3$	$c_j=1$	$c_j=2$	$c_j=3$	$c_j=1$	$c_j=2$	$c_j=3$
	$c_i=1$	0	{0}	{0}	{0}	$\underline{0}$	{0}	{0}	{0}	0
$x_i=1$	$c_i=2$	{0}	6	4	6	{0}	$\underline{5}$	4	5	{0}
	$c_i=3$	{0}	12	8	$\underline{12}$	{0}	10	8	10	{0}
	$c_i=1$	{0}	6	4	6	{0}	5	4	$\underline{5}$	{0}
$x_i=2$	$c_i=2$	0	{0}	{0}	{0}	0	{0}	{0}	{0}	$\underline{0}$
	$c_i=3$	{0}	18	12	18	{0}	15	$\underline{12}$	15	{0}
	$c_i=1$	{0}	$\underline{12}$	8	12	{0}	10	8	10	{0}
$x_i=3$	$c_i=2$	{0}	18	$\underline{12}$	18	{0}	15	12	15	{0}
	$c_i=3$	$\underline{0}$	{0}	{0}	{0}	0	{0}	{0}	{0}	0

Τα στοιχεία που είναι υπογραμμισμένα είναι αυτά που αντιστοιχούν στους άσσους του πίνακα Y και άρα αυτά που συνεισφέρουν στο άθροισμα του εσωτερικού γινομένου της αντικειμενικής συνάρτησης. Το άθροισμά τους είναι ίσο με 58. Στο ίδιο αποτέλεσμα καταλήγουμε και αν χρησιμοποιήσουμε την αντικειμενική συνάρτηση (1) με τους ίδιους πίνακες και τη μετάθεση p που αντιστοιχεί στον πίνακα Y.

■

Στο τελευταίο σχήμα χρησιμοποιήθηκαν οι δείκτες x_i, x_j, c_i και c_j για να χαρακτηρίσουν το κάθε στοιχείο του πίνακα. Οι δύο πρώτοι χαρακτηρίζουν τον υποπίνακα στον οποίο ανήκει το στοιχείο, ενώ οι δύο τελευταίοι τη γραμμή και τη στήλη του στοιχείου μέσα στον υποπίνακα. Το στοιχείο c_{ijkl} του πίνακα C τώρα μπορεί να δηλωθεί ως $c_{x_i c_i x_j c_j}$. Ισχύει $1 \le x_i, x_j, c_i, c_j \le n$, όπου n είναι το μέγεθος του προβλήματος. Αν οι συντελεστές y_i και y_j δείχνουν τη γραμμή και τη στήλη κάθε στοιχείου, αντίστοιχα, σε ολόκληρο τον πίνακα C, τότε ισχύουν οι παρακάτω σχέσεις:

$$y_i = (x_i - 1)n + c_i$$
$$y_j = (x_j - 1)n + c_j$$
$$x_i = \lfloor (y_i - 1)/n \rfloor + 1$$
$$x_j = \lfloor (y_j - 1)/n \rfloor + 1$$
$$c_i = (y_i - 1) \bmod(n) + 1$$
$$c_j = (y_j - 1) \bmod(n) + 1$$

Η επιλογή του στοιχείου $c_{x_i c_i x_j c_j}$ σημαίνει την ανάθεση της εγκατάστασης x_i στην τοποθεσία x_j και την ανάθεση της εγκατάστασης c_i στην τοποθεσία c_j. Επιλέχθηκε, δηλαδή, για να συμμετέχει στη λύση, ο υποπίνακας με συντεταγμένες (x_i, x_j) και

συγκεκριμένα το στοιχείο (c_i, c_j) αυτού του υποπίνακα. Οι ίδιες αναθέσεις όμως προκύπτουν και από την επιλογή του υποπίνακα με συντεταγμένες (c_i, c_j) και του στοιχείου (x_i, x_j) αυτού του υποπίνακα. Άρα, η επιλογή του στοιχείου $c_{x_i c_i x_j c_j}$ σημαίνει αυτόματα και την επιλογή του στοιχείου $c_{c_i x_i c_j x_j}$. Στο ίδιο συμπέρασμα καταλήγουμε αν σκεφτούμε ότι, για να επιλεγεί το στοιχείο $c_{x_i c_i x_j c_j}$ του πίνακα C, χρειάζεται να είναι άσσοι τα στοιχεία (x_i, x_j) και (c_i, c_j) του πίνακα μετάθεσης X. Άρα, χρειάζεται να έχουν γίνει οι αναθέσεις $x_i \to x_j$ και $c_i \to c_j$. Κάθε στοιχείο του πίνακα C, λοιπόν, έχει το συμπληρωματικό του. Αν οι πίνακες F και D είναι συμμετρικοί, τότε τα συμπληρωματικά στοιχεία του πίνακα C έχουν ίσες τιμές.

Μπορεί εύκολα να παρατηρηθεί ότι, αν ο γραμμικός όρος στη συνάρτηση (1) είναι ίσος με μηδέν ($B = 0$), τότε ισχύει $c_{iikk} = 0$ για κάθε $1 \le i, k \le n$.

Τα στοιχεία c_{iikk} έχουν ως συμπληρωματικό τον εαυτό τους. Οι τιμές τους αντιστοιχούν σε γραμμικά κόστη, που περιλαμβάνονται στον πίνακα B της συνάρτησης (1). Είναι τα στοιχεία για τα οποία ισχύει $x_i = c_i$ και $x_j = c_j$. Όταν επιλέγεται το στοιχείο $c_{x_i c_i x_j c_j}$ του πίνακα C με $x_i \ne c_i$ και $x_j \ne c_j$, αυτόματα επιλέγονται και τα στοιχεία $c_{x_i x_i x_j x_j}$ και $c_{c_i c_i c_j c_j}$, που αντιστοιχούν στις τιμές του γραμμικού όρου της αντικειμενικής συνάρτησης για τη συγκεκριμένη ανάθεση. Πρόκειται για τα κόστη τοποθέτησης της εγκατάστασης x_i στη θέση x_j και της εγκατάστασης c_i στη θέση c_j. Οι συντεταγμένες των στοιχείων αυτών μέσα στον υποπίνακα, στον οποίο ανήκουν, είναι ίδιες με τις συντεταγμένες αυτού του υποπίνακα μέσα στον πίνακα C. Για κάθε υποπίνακα υπάρχει ένα ακριβώς τέτοιο στοιχείο. Άρα, το πλήθος των στοιχείων του πίνακα C, που έχουν ως συμπληρωματικό τον εαυτό τους, είναι n^2.

Τέλος, στον πίνακα C υπάρχει ένα σύνολο στοιχείων, που η επιλογή τους είναι εξ' αρχής απαγορευμένη, όχι λόγω της τιμής τους, αλλά λόγω της θέσης που κατέχουν στον πίνακα. Το γεγονός αυτό είναι απόρροια των περιορισμών που θέτονται στον πίνακα Y, ο οποίος είναι το γινόμενο Kronecker ενός πίνακα μετάθεσης με τον εαυτό του. Αυτό σημαίνει ότι σε κάθε γραμμή του και σε κάθε στήλη του θα υπάρχει ακριβώς ένας άσσος. Επίσης, αν θεωρηθεί ως πίνακας υποπινάκων, σε κάθε γραμμή του και σε κάθε στήλη του πρέπει να υπάρχει ένας υποπίνακας μετάθεσης και όλοι οι υπόλοιποι να είναι μηδενικοί. Μεταφέροντας τους ίδιους περιορισμούς στον πίνακα C, τα στοιχεία που δεν επιτρέπεται να επιλεγούν είναι αυτά, των οποίων οι συντελεστές χαρακτηρίζονται από τις σχέσεις

$$x_i = c_i \text{ και } x_j \ne c_j$$
$$\text{ή}$$
$$x_i \ne c_i \text{ και } x_j = c_j$$

Στον πίνακα C υπάρχουν n^2 υποπίνακες. Σε κάθε έναν από αυτούς υπάρχουν n στοιχεία, για τα οποία ισχύει $x_i = c_i$ και n στοιχεία, για τα οποία ισχύει $x_j = c_j$. Στα

πρώτα συμπίπτει η γραμμή τους μέσα στον υποπίνακα με τη γραμμή του υποπίνακα μέσα στον πίνακα C, ενώ στα δεύτερα η στήλη τους μέσα στον υποπίνακα με τη στήλη του υποπίνακα μέσα στον πίνακα C. Από τα πρώτα n στοιχεία, οι συντελεστές μόνο του ενός ικανοποιούν τη σχέση $x_j = c_j$. Για τα υπόλοιπα $n-1$ ισχύει $x_j \neq c_j$. Ομοίως, για τα στοιχεία της κάθε στήλης με $x_j = c_j$, τα $n-1$ έχουν $x_i \neq c_i$. Τα δύο αυτά σύνολα με πλήθος στοιχείων $n-1$ δεν έχουν κανένα κοινό στοιχείο, λόγω της δομής του πίνακα C. Έτσι, σε κάθε έναν από τους n^2 υποπίνακες υπάρχουν $2(n-1)$ στοιχεία, που δεν επιτρέπεται η επιλογή τους, εξ' αρχής. Συνολικά στον πίνακα C υπάρχουν $2n^2(n-1)$ τέτοια στοιχεία. Στον τελευταίο πίνακα του παραδείγματος τα στοιχεία αυτά βρίσκονται μέσα σε αγκύλες.

Όπως περιγράφηκε παραπάνω η επιλογή ενός στοιχείου οδηγεί στην αυτόματη επιλογή κι άλλων, γεγονός που οφείλεται στη δομή του πίνακα C και τη σημασία της θέσης των στοιχείων μέσα σ' αυτόν. Εκτός αυτού, η επιλογή ενός στοιχείου οδηγεί και στην απαγόρευση επιλογής ενός συνόλου άλλων στοιχείων. Πρόκειται για τα στοιχεία, που αν επιλεγούν συγχρόνως με το πρώτο, παραβιάζουν κάποιον περιορισμό του πίνακα Y. Τα στοιχεία αυτά, εξ' αρχής, έχουν τη δυνατότητα να επιλεγούν. Χάνουν, όμως, αυτή τη δυνατότητα, όταν κατά τη διάρκεια της διαδικασίας απληστίας εξόδου άλλα στοιχεία επιλέγονται. Κάθε επιλογή στοιχείου οδηγεί στην απαγόρευση τόσο μεμονωμένων στοιχείων στον ίδιο ή σε διαφορετικούς υποπίνακες, όσο και απαγόρευση ολόκληρων υποπινάκων. Η επιλογή του στοιχείου $c_{x_i' c_i' x_j' c_j'}$ του πίνακα C με $x_i' \neq c_i'$ και $x_j' \neq c_j'$ οδηγεί στην απαγόρευση επιλογής, άρα και στην επιβολή μόνιμης διαγραφής από τη λύση, των στοιχείων με συντελεστές που χαρακτηρίζονται από τις σχέσεις

$$x_i = x_i', x_j = x_j', c_i = c_i' \text{ και } c_j \neq c_j'$$
ή
$$x_i = x_i', x_j = x_j', c_i \neq c_i' \text{ και } c_j = c_j'$$
ή
$$x_i \neq x_i', x_j \neq x_j', c_i = c_i' \text{ και } c_j \neq c_j'$$
ή
$$x_i \neq x_i', x_j \neq x_j', c_i \neq c_i' \text{ και } c_j = c_j'$$
ή
$$x_i = x_i', x_j \neq x_j' \text{ και } 1 \leq c_i, c_j \leq n$$
ή
$$x_i \neq x_i', x_j = x_j' \text{ και } 1 \leq c_i, c_j \leq n$$

7.4 Το QAP ως n^2 LAP

Στην προηγούμενη ενότητα, περιγράφηκε ο πίνακας C, ο τρόπος κατασκευής του και η σημασία των στοιχείων του. Σε κάθε στιγμιότυπο μεγέθους n, το πλήθος των στοιχείων του πίνακα C είναι n^4, με αποτέλεσμα να μεγαλώνει πολύ με την αύξηση του μεγέθους του προβλήματος. Το γεγονός αυτό έχει αρνητικές συνέπειες τόσο στον αποθηκευτικό χώρο, όσο και στο χρόνο εκτέλεσης που χρειάζονται οι διεργασίες

υλοποίησης του αλγορίθμου απληστίας εξόδου. Ο χειρισμός των στοιχείων του πίνακα C στο σύνολό τους είναι χρονοβόρος και υπολογιστικά πολύ απαιτητικός.

Για το λόγο αυτό, στη μέθοδο που αναπτύχθηκε, ο πίνακας C δεν αντιμετωπίζεται ως ένας πίνακας συντελεστών μεγέθους $n^2 \times n^2$, αλλά ως πίνακας n^2 υποπινάκων. Κάθε ένας από αυτούς τους υποπίνακες έχει μέγεθος $n \times n$. Αν ο πίνακας C περιέχει όλη την «αλήθεια» για ένα στιγμιότυπο του QAP, τότε κάθε ένας υποπίνακάς του περιέχει ένα μέρος αυτής της «αλήθειας». Κάθε προσεγγιστικός αλγόριθμος προσπαθεί να υλοποιήσει μεθόδους, που εκμεταλλεύονται όσο το δυνατόν πιο αποδοτικά την «αλήθεια» του προβλήματος, ώστε να οδηγηθεί σε καλές λύσεις του υπερχώρου που ορίζει η αντικειμενική συνάρτηση. Στόχος της παρούσας μεθόδου είναι η ανίχνευση των υποπινάκων, που αντιπροσωπεύουν όσο το δυνατόν καλύτερα τον πίνακα C στο σύνολό του, ώστε ο αλγόριθμος απληστίας εξόδου να εφαρμοστεί σε καθέναν από αυτούς και όχι σε ολόκληρο τον πίνακα C.

Όπως αναφέρθηκε και παραπάνω, σε κάθε έναν από αυτούς τους υποπίνακες υπάρχει η πιθανότητα να επιλεγούν τα στοιχεία, που αντιστοιχούν στους άσσους ενός πίνακα μετάθεσης. Για το λόγο αυτό, σε κάθε γραμμή και κάθε στήλη τους υπάρχει ακριβώς ένα επιλεχθέν στοιχείο. Έτσι, κάθε ένας από αυτούς τους υποπίνακες μπορεί να αντιμετωπιστεί ως ένα Γραμμικό Πρόβλημα Ανάθεσης (Linear Assignment Problem – LAP). Ένα LAP, που η άπληστη λύση του είναι πολύ κοντά στην άπληστη λύση του ολικού QAP. Κάθε πίνακας X, που αποτελεί μια λύση για ένα από αυτά τα LAP, ορίζει και έναν πίνακα $Y = X \otimes X$, που αποτελεί μια λύση για το QAP. Με τον τρόπο αυτό το QAP αντιμετωπίζεται ως ένα σύνολο n^2 στο πλήθος LAP.

Με την παραπάνω θεώρηση επιτρέψαμε μια χαλάρωση στον τρόπο υπολογισμού της άπληστης λύσης του προβλήματος, για χάρη του υπολογιστικού χρόνου και χώρου που απαιτεί η μέθοδος. Αυτό που απομένει τώρα είναι ο υπολογισμός μιας λύσης του κάθε LAP με χρήση της απληστίας εξόδου. Κάθε ένα LAP αποτελεί έναν πίνακα μεγέθους $n \times n$. Τα n^2 στοιχεία του ταξινομούνται σε μη αύξουσα σειρά. Επαναληπτικά, διαγράφονται στοιχεία του πίνακα, ξεκινώντας από αυτά με τη μεγαλύτερη τιμή, έτσι ώστε αυτά που θα απομείνουν τελικά, να αποτελούν την άπληστη λύση. Σε κάθε βήμα της επανάληψης, το στοιχείο με το μεγαλύτερο κόστος διαγράφεται από τον πίνακα. Η διαδικασία διαγραφής συνεχίζεται, μέχρι το στοιχείο με το μεγαλύτερο κόστος να είναι το τελευταίο στοιχείο που έχει απομείνει σε μια γραμμή ή μια στήλη. Το στοιχείο αυτό δεν μπορεί να διαγραφεί. Άρα, επιλέγεται και συμμετέχει στην άπληστη λύση. Γνωρίζουμε πλέον τον ένα άσσο του πίνακα μετάθεσης. Κανένα άλλο στοιχείο της γραμμής και της στήλης του στοιχείου που επιλέχθηκε δεν μπορεί να επιλεγεί. Έτσι, αυτή η γραμμή και αυτή η στήλη δεν χρειάζεται να συμμετέχουν πλέον στο πρόβλημα. Όλα τα υπόλοιπα στοιχεία, που έχουν διαγραφεί, ξαναμπαίνουν στην εν δυνάμει λύση και η διαδικασία απληστίας εξόδου επαναλαμβάνεται. Από ένα πρόβλημα μεγέθους n, μεταβήκαμε σε ένα πρόβλημα μεγέθους $n-1$. Με επαναληπτικές αναγωγές σε όλο και μικρότερα προβλήματα, γίνονται γνωστά όλα τα στοιχεία του πίνακα μετάθεσης και μια λύση απληστίας εξόδου έχει βρεθεί για το LAP.

Τέλος, για να γίνει η μέθοδος εκτός από άπληστη και τυχαία, σε κάθε βήμα της επανάληψης δε διαγράφεται το στοιχείο με το μεγαλύτερο κόστος, αλλά ένα από αυτά με το μεγαλύτερο κόστος. Τα υποψήφια στοιχεία εισάγονται σε μια λίστα και επιλέγεται ψευδοτυχαία ένα για διαγραφή.

7.5 Η απληστία εξόδου ως μια maxmin μέθοδος

Χωρίς βλάβη της γενικότητας, ας υποθέσουμε αρχικά ότι η μέθοδός που περιγράφηκε στην προηγούμενη ενότητα δε χρησιμοποιεί την τυχαιότητα στη διαγραφή των στοιχείων. Για την εύρεση της λύσης απληστίας εξόδου του LAP, σε κάθε βήμα διαγράφεται ακριβώς το στοιχείο με το μεγαλύτερο κόστος από τον πίνακα μεγέθους $n \times n$ που το περιγράφει.

Η επαναληπτική διαγραφή των μεγαλύτερων στοιχείων του πίνακα του LAP μέχρι να φτάσει και η σειρά του τελευταίου μιας γραμμής ή μιας στήλης, ώστε αυτό να επιλεγεί, ισοδυναμεί με την απευθείας επιλογή του μεγαλύτερου στοιχείου μεταξύ των μικρότερων κάθε γραμμής και στήλης.

Έτσι, σε κάθε ένα LAP υπάρχει η δυνατότητα εύρεσης της λύσης απληστίας εξόδου όχι μόνο μέσω διαγραφών στοιχείων, αλλά και μέσω απευθείας επιλογών, ακολουθώντας μια στρατηγική maxmin. Την υπερίσχυση μιας από τις δύο περιπτώσεις θα την αναδείξει η σύγκριση των υπολογιστικών τους αποδόσεων. Οι λειτουργίες που είναι απαραίτητες για την υλοποίηση της τεχνικής διαγραφών είναι:

- ταξινόμηση των n^2 στοιχείων του πίνακα του LAP
- διαγραφή αυτών με τη μεγαλύτερη τιμή. Στη χειρότερη περίπτωση θα διαγραφούν $(m-1)^2$ στοιχεία μέχρι να βρεθεί αυτό που θα επιλεγεί, όπου m είναι το μέγεθος κάθε υποπροβλήματος που προκύπτει μετά τη δημιουργία κάποιων αναθέσεων. Αρχικά $m = n$ και σε κάθε υποπρόβλημα η τιμή του m μειώνεται κατά ένα. Στην καλύτερη περίπτωση το πλήθος αυτό είναι $m-1$, ενώ στη μέση περίπτωση ισούται με το μέσο όρο όλων των επιτρεπτών περιπτώσεων

$$\frac{\sum_{i=1}^{(m-1)^2} i}{(m-1)^2} = \frac{(m-1)^2\left[(m-1)^2+1\right]}{2(m-1)^2} = \frac{1+(m-1)^2}{2}$$

- επιλογή του τελευταίου στοιχείου μιας γραμμής ή μιας στήλης.
- μόνιμη διαγραφή από το ταξινομημένο σύνολο των στοιχείων που βρίσκονται στην ίδια γραμμή ή στην ίδια στήλη με το στοιχείο που επιλέχθηκε. Το σύνολο των στοιχείων αυτών είναι $2(m-1)$. Προτιμάται αυτή η διαδικασία για να μην είναι απαραίτητη η επαναταξινόμηση των εναπομεινάντων στοιχείων.
- επανάληψη της διαδικασίας για τα εναπομείναντα στοιχεία και για μέγεθος προβλήματος $m \leftarrow m-1$. Σε κάθε υποπρόβλημα, το m ισούται με το αρχικό μέγεθος του προβλήματος n μείον το πλήθος των αναθέσεων που έχουν γίνει πριν από το υποπρόβλημα αυτό.

Για την αποδοτική υλοποίηση των παραπάνω, το σύνολο των υποψηφίων προς διαγραφή στοιχείων μπορεί να αναπαρασταθεί με τη δομή δεδομένων της συνδεδεμένης λίστας και η ταξινόμηση να γίνει με μια από τις ταχύτερες μεθόδους, όπως η ταξινόμηση σωρού. Η ταξινόμηση των n^2 στοιχείων απαιτεί χρόνο στην τάξη του $O(n^2 \log n^2)$. Για την άμεση εύρεση της θέσης ενός από αυτών μέσα στον πίνακα χρειάζεται η ταξινόμηση όχι μόνο των τιμών τους, αλλά και των συντεταγμένων τους. Τόσο οι ταξινομημένες τιμές, όσο και οι δείκτες των στοιχείων, αποθηκεύονται σε συνδεδεμένες λίστες. Η

ταξινόμηση γίνεται μια φορά στην αρχή της διαδικασίας και δεν επαναλαμβάνεται. Σε κάθε υποπρόβλημα απαιτείται στη μέση περίπτωση η διαγραφή $\frac{1+(m-1)^2}{2}$ στοιχείων, η οποία γίνεται σε χρόνο στην τάξη του $O((m-1)^2)$. Ο καθορισμός της θέσης τους μέσα στον πίνακα γίνεται άμεσα από τις ταξινομημένες λίστες. Για τον εντοπισμό και τη διαγραφή από τις λίστες αυτές των στοιχείων, που δεν μπορούν πλέον να επιλεγούν, λόγω της επιλογής που έγινε, απαιτείται χρόνος στην τάξη του $O(2(m-1))$. Η διαδικασία της διαγραφής επαναλαμβάνεται για $m = n, n-1,...,2,1$. Άρα, η μέθοδος της εύρεσης λύσης απληστίας εξόδου του LAP με διαγραφή στοιχείων απαιτεί χρόνο στην τάξη του

$$O\left(n^2 \log n^2 + \sum_{m=1}^{n}\left[(m-1)^2 + 2(m-1)\right]\right) =$$
$$O\left(n^2 \log n^2 + \sum_{m=1}^{n}\left[m^2 - 1\right]\right) =$$
$$O\left(n^2 \log n^2 + \sum_{m=1}^{n} m^2 - \sum_{m=1}^{n} 1\right) =$$
$$O\left(n^2 \log n^2 + \frac{2n^3 + 3n^2 + n}{6} - n\right) =$$
$$O(n^2 \log n^2 + \frac{1}{3}n^3) =$$
$$O(n^3)$$

Η παραπάνω άπληστη διαδικασία εκτελείται ξεχωριστά σε κάθε ένα από τα n^2 LAP του πίνακα C, χωρίς να υπάρχουν λειτουργίες, που να είναι κοινές για περισσότερα από ένα LAP. Ως εκ τούτου, ο συνολικός χρόνος που απαιτείται, ανήκει στην τάξη του $O(2n^4 \log n^2 + n^5) = O(n^5)$. Ένα παράδειγμα εκτέλεσης της λειτουργίας αυτής δείχνει πλήρως τις παραπάνω λειτουργίες και τον τρόπο υλοποίησής τους. Έστω ότι το LAP δημιουργείται από τη γραμμή [4 1 9] του πίνακα F και τη γραμμή [5 7 2] του πίνακα D και άρα, περιγράφεται από τον πίνακα

$$LAP = \begin{bmatrix} 20 & 28 & 8 \\ 5 & 7 & 2 \\ 45 & 63 & 18 \end{bmatrix}$$

Τρεις ταξινομημένες, συνδεδεμένες λίστες δημιουργούνται και αντιπροσωπεύουν τα στοιχεία του πίνακα, τον δείκτη του καθενός μέσα στον πίνακα και τη θέση του μέσα στη λίστα, αντίστοιχα.

i	1	2	3	4	5	6	7	8	9
y	62	45	28	20	18	8	7	5	2
$index$	8	7	2	1	9	3	5	4	6
r_index	4	3	6	8	7	9	2	1	5

Η λίστα y περιέχει τα στοιχεία του πίνακα ταξινομημένα σε μη αύξουσα σειρά. Η λίστα $index$ περιέχει τους αντίστοιχους δείκτες. Η τεταγμένη του i-οστού στοιχείου της λίστας μέσα στο δισδιάστατο πίνακα είναι $c_i = (index(i)-1)/n+1$ και η τετμημένη του $c_j = index(i)-(c_i-1)*n$. Με την έναρξη της επαναληπτικής διαδικασίας, διατρέχεται η λίστα y από την αρχή μέχρι το τέλος της και διαγράφονται τα στοιχεία με τιμές 63, 45, 28 και 20. Το επόμενο μεγαλύτερο στοιχείο είναι αυτό με τιμή 18. Αποτελεί το τελευταίο εναπομείναν στην τρίτη γραμμή και άρα, επιλέγεται να συμμετέχει στη λύση. Τώρα πρέπει να γίνει ο αποκλεισμός των στοιχείων με τιμές 45, 63, 8 και 2 που ανήκουν στην ίδια γραμμή ή την ίδια στήλη με το 8. Οι συντεταγμένες του καθενός είναι (c_i, c_j). Για τη λίστα r_index (reversed index) ισχύει $index(r\_index(i)) = i$ και $r\_index(index(i)) = i$. Η θέση του στοιχείου προς διαγραφή μέσα στη λίστα υπολογίζεται από τον τύπο $r\_index((c_i-1)*n+c_j)$. Έτσι, αποκλείονται και διαγράφονται μόνιμα οι θέσεις 1,2,6 και 9 της λίστας από τη λύση. Τα στοιχεία 28 και 20, που είχαν διαγραφεί πριν επιλεγεί το 18, ξαναμπαίνουν στη λύση. Η διαδικασία επαναλαμβάνεται και τελικά επιλέγονται για τη λύση τα στοιχεία με τιμές 18, 20 και 7, με τη συγκεκριμένη σειρά. Η μετάθεση που αντιστοιχεί στη λύση αυτή είναι η $p = (1,2,3)$.

Στη δεύτερη περίπτωση με τη διαδικασίας maxmin, σε κάθε βήμα της επαναληπτικής διαδικασίας, επιλέγεται το μεγαλύτερο στοιχείο από το σύνολο με στοιχεία τα μικρότερα κάθε γραμμής και στήλης. Η ταξινόμηση όλων των n^2 στοιχείων δεν είναι πλέον απαραίτητη. Αρκεί η ταξινόμηση των στοιχείων κάθε γραμμής και κάθε στήλης, ξεχωριστά. Όμως, ο κάθε ένας υποπίνακας που αντιπροσωπεύει ένα LAP του πίνακα C, δημιουργείται από το γινόμενο των στοιχείων μιας γραμμής του πίνακα F και μιας γραμμής του πίνακα D. Άρα τελικά, αρκεί η ταξινόμηση μόνο των στοιχείων των αντίστοιχων γραμμών των πινάκων F και D σε μη φθίνουσα σειρά. Αν χρησιμοποιηθεί η ταξινόμηση σωρού, απαιτείται χρόνος στην τάξη του $O(2n\log n)$. Βέβαια, εκτός από τις τιμές ταξινομούνται και οι δείκτες των δύο γραμμών.

Εφόσον έχει γίνει η ταξινόμηση, η εύρεση και επιλογή του μεγαλύτερου στοιχείου είναι απλή υπόθεση και πραγματοποιείται σε σταθερό χρόνο. Μετά την επιλογή κάποιου στοιχείου χρειάζεται να γίνει η εξαίρεση των στοιχείων της ίδιας γραμμής και στήλης από τη λύση. Αφού τώρα τα στοιχεία συνεχίζουν να βρίσκονται τοποθετημένα στον δισδιάστατο πίνακα μεγέθους $n \times n$, δεν είναι απαραίτητη η διαγραφή του κάθε στοιχείου ξεχωριστά. Αρκεί ένας πίνακας ετικετών για τις γραμμές και ένας για τις στήλες. Οι πίνακες αυτοί δηλώνουν ποιες από τις γραμμές και τις στήλες ανήκουν ακόμη στο τρέχον υποπρόβλημα. Οι συντεταγμένες του στοιχείου, που επιλέχθηκε, είναι γνωστές. Άρα, η απαλοιφή της αντίστοιχης γραμμής και στήλης από το πρόβλημα γίνεται σε σταθερό χρόνο. Η διαδικασία επαναλαμβάνεται n φορές, μέχρι να

πραγματοποιηθούν όλες οι αναθέσεις και ολοκληρώνεται σε γραμμικό χρόνο. Συνολικά, η εύρεση της λύσης απληστίας εξόδου του LAP με τον τρόπο αυτό απαιτεί χρόνο στην τάξη του $O(2n\log n+n)$. Με τις γραμμές των πινάκων F και D του προηγούμενου παραδείγματος, η διαδικασία χρησιμοποιεί τις παρακάτω δομές:

tf			*t*	*t*	*t*
	index		3	1	2
		y	2	5	7
t	2	1	2	5	7
t	1	4	8	20	28
t	3	9	18	45	63

Οι λίστες του παραδείγματος μπορούν να υλοποιηθούν μέσω πινάκων. Η λίστα – πίνακας *tf* δείχνει ποιες γραμμές και ποιες στήλες συμμετέχουν σε κάθε υποπρόβλημα με την τιμή *t*(*rue*) να δηλώνει τη συμμετοχή και την τιμή *f*(*alse*) τη μη συμμετοχή. Η λίστα – πίνακας *index* περιέχει τους δείκτες των γραμμών των πινάκων F και D, όπως προέκυψαν μετά την ταξινόμηση. Η λίστα – πίνακας *y* δείχνει τα αντίστοιχα στοιχεία ταξινομημένα. Τα μικρότερα στοιχεία κάθε γραμμής βρίσκονται στα αριστερά. Τα μικρότερα στοιχεία κάθε στήλης βρίσκονται προς τα πάνω. Το μεγαλύτερο από όλα αυτά τα στοιχεία είναι αυτό με την τιμή 18 και το οποίο επιλέγεται για να συμμετέχει στην άπληστη λύση. Οι συντεταγμένες του στοιχείου αυτού είναι (c_i,c_j). Η ανάθεση στην οποία αντιστοιχεί η επιλογή του είναι η $index(c_i)\rightarrow index(c_j)$. Οι αλλαγές, που επιφέρει στην παραπάνω δομή η επιλογή αυτή, φαίνονται στο παρακάτω σχήμα.

tf			*f*	*t*	*t*
	index		3	1	2
		y	2	5	7
t	2	1	2	5	7
t	1	4	8	20	28
f	3	9	18	45	63

Η τρίτη γραμμή και η τρίτη στήλη δε συμμετέχουν πλέον στο πρόβλημα. Έγινε η μετάβαση σε ένα υποπρόβλημα μεγέθους $n=2$, που περιλαμβάνει τις γραμμές 1 και 2 του πίνακα F και τις γραμμές 1 και 2 του πίνακα D. Η διαδικασία επαναλαμβάνεται και τελικά, επιλέγονται τα στοιχεία με τιμές 18, 20 και 7, με τη συγκεκριμένη σειρά. Το αποτέλεσμα είναι ίδιο με αυτό της προηγούμενης τεχνικής. Η διαφορά είναι ότι με τον τρόπο αυτό, βρέθηκε σε λιγότερο υπολογιστικό χρόνο.

Αν ο πίνακας C αντιμετωπιστεί ως πίνακας υποπινάκων ισχύει ότι:

- οι υποπίνακες, που ανήκουν στην ίδια γραμμή, δημιουργούνται από την ίδια γραμμή του πίνακα F.

- οι υποπίνακες, που ανήκουν στην ίδια στήλη, δημιουργούνται από την ίδια γραμμή του πίνακα D.

Έτσι, η διαδικασία της ταξινόμησης δε χρειάζεται να επαναλαμβάνεται σε κάθε ένα LAP ξεχωριστά. Αρχικά και μόνο μια φορά ταξινομούνται οι γραμμές των πινάκων F και D. Για την ταξινόμηση των στοιχείων όλων των γραμμών του πίνακα F, χρειάζεται χρόνος στην τάξη του $O(n^2 \log n)$. Άλλος τόσος υπολογιστικός χρόνος χρειάζεται για τον πίνακα D. Ο συνολικός χρόνος που απαιτείται για την εύρεση της λύσης απληστίας εξόδου για όλα τα n^2 LAP, είναι στην τάξη του $O(n^2 \log n + n^3) = O(n^3)$.

Από τα αποτελέσματα της σύγκρισης των δύο τεχνικών προκύπτει ότι είναι πολύ πιο αποδοτικό υπολογιστικά, η απληστία εξόδου να αντιμετωπιστεί ως μια maxmin μέθοδος και αφού βέβαια η δομή του LAP δίνει αυτή τη δυνατότητα.

7.6 Σπειροειδής διάσχιση

Στην παραπάνω ανάλυση υποθέσαμε ότι ο αλγόριθμος δεν είναι τυχαίος και ότι σε κάθε βήμα κάνει πάντα την τοπικά καλύτερη επιλογή. Στην τεχνική των διαγραφών, σε κάθε βήμα διαγράφει το στοιχείο με το μεγαλύτερο κόστος. Στην τεχνική maxmin, σε κάθε βήμα επιλέγει το στοιχείο με το μεγαλύτερο κόστος από το σύνολο P, όπου στο σύνολο P συμμετέχει το μικρότερο στοιχείο κάθε γραμμής και στήλης. Η εισαγωγή της τυχαιότητας στον αλγόριθμο, σε συνδυασμό με επαναλαμβανόμενες εκτελέσεις του, διευρύνει τα όρια της αναζήτησης που πραγματοποιείται και αυξάνει την πιθανότητα εύρεσης της βέλτιστης λύσης. Με την εισαγωγή της τυχαιότητας, σε κάθε βήμα ο αλγόριθμος δεν κάνει την πιο άπληστη επιλογή, αλλά μία από το σύνολο των πιο άπληστων.

Με την εισαγωγή της τυχαιότητας στην τεχνική maxmin, η πιθανότητα των παρακάτω ενδεχομένων δεν είναι πλέον μηδενική.

- Επιλογή ενός στοιχείου του συνόλου P, που δεν έχει το μέγιστο κόστος ανάμεσα στα στοιχεία του συνόλου.
- Επιλογή ενός στοιχείου κάποιας γραμμής ή στήλης, που δεν έχει το μικρότερο κόστος ανάμεσα στα στοιχεία της γραμμής ή της στήλης αυτής.

Χρειάζεται λοιπόν να δημιουργηθεί μια λίστα υποψηφιοτήτων, από την οποία θα επιλέγεται κάθε φορά ένα στοιχείο, στο πλαίσιο πάντα της maxmin τεχνικής. Κάθε στοιχείο στη λίστα αυτή έχει την ίδια πιθανότητα επιλογής. Το μέγεθός της εξαρτάται από το μέγεθος του προβλήματος και από μια παράμετρο εισόδου του αλγορίθμου. Η παράμετρος αυτή, έστω a, καθορίζει το βαθμό απληστίας και το βαθμό τυχαιότητας του αλγορίθμου, καθώς παίρνει τιμές από το διάστημα $[0,1]$. Τα στοιχεία του πίνακα χρειάζεται να μπουν σε μια συγκεκριμένη διάταξη, σύμφωνα με την οποία θα συμμετέχουν στη λίστα υποψηφιοτήτων. Ο τρόπος υλοποίησης της λίστας αυτής στον κώδικα παίζει σημαντικό ρόλο στην απόδοση του αλγορίθμου που θα προκύψει. Αν υλοποιηθεί με κάποια γραμμική δομή δεδομένων, αμέσως χάνονται τα πλεονεκτήματα της maxmin διαδικασίας που προκύπτουν από τη δομή του δισδιάστατου πίνακα και περιγράφηκαν στην προηγούμενη ενότητα. Το ζητούμενο είναι λοιπόν η διατήρηση αποθήκευσης των στοιχείων του LAP σε έναν δισδιάστατο πίνακα, αλλά και η ταυτόχρονη, γραμμική διάταξη αυτών σε μια ακολουθία με φθίνουσα δυνατότητα συμμετοχής στη λίστα υποψηφιοτήτων σε σχέση με το μέγεθος της τελευταίας.

Σύμφωνα με το πρότυπο που ακολουθήθηκε, για τη συμμετοχή στη λίστα υποψηφιοτήτων προηγούνται τα στοιχεία της πρώτης στήλης και της πρώτης γραμμής του αρχικού πίνακα του προβλήματος, έστω C_{ik}. Ακολουθούν τα στοιχεία της πρώτης στήλης και της πρώτης γραμμής του υποπίνακα $C_{ik}(2-n,2-n)$, όπου $A(x-y,z-w)$ είναι ο υποπίνακας, που περιέχει τις γραμμές x έως y και τις στήλες z έως w του πίνακα A. Ακολουθούν η πρώτη στήλη και η πρώτη γραμμή του υποπίνακα $C_{ik}(3-n,3-n)$ και ούτω καθεξής. Επίσης, μέσα σε κάθε γραμμή και κάθε στήλη τα στοιχεία δεν έχουν την ίδια πιθανότητα να συμμετάσχουν στη λίστα. Αν ένα μόνο μέρος τους πρέπει να συμμετάσχει σ' αυτήν, η σειρά διάταξης καθορίζει αυτά που προηγούνται. Σε κάθε γραμμή προηγούνται τα στοιχεία που βρίσκονται πιο δεξιά, ενώ σε κάθε στήλη αυτά που βρίσκονται πιο κάτω.

Παράδειγμα

Όλα τα παραπάνω υποδεικνύουν έναν τρόπο διάσχισης των στοιχείων του πίνακα του LAP, ο οποίος να προσαρμόζεται στη γραμμική διάταξη, που μόλις περιγράφηκε. Τα παρακάτω σχήματα δείχνουν αυτόν τον τρόπο διάσχισης σε στιγμιότυπα προβλημάτων μεγέθους 3, 4, 5 και 6. Στη θέση κάθε στοιχείου δε φαίνεται η τιμή του, αλλά μια ετικέτα. Από την ετικέτα κάθε στοιχείου εξαρτάται αν αυτό συμμετέχει ή όχι στη λίστα υποψηφιοτήτων. Σε μια λίστα μεγέθους m θα συμμετέχουν τα στοιχεία με ετικέτες $e \le m$.

$$\begin{bmatrix} 5 & 4 & 3 \\ 2 & 8 & 7 \\ 1 & 6 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 6 & 5 & 4 \\ 3 & 12 & 11 & 10 \\ 2 & 9 & 15 & 14 \\ 1 & 8 & 13 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 9 & 8 & 7 & 6 & 5 \\ 4 & 16 & 15 & 14 & 13 \\ 3 & 12 & 21 & 20 & 19 \\ 2 & 11 & 18 & 24 & 23 \\ 1 & 10 & 17 & 22 & 25 \end{bmatrix}$$

11	10	9	8	7	6
5	20	19	18	17	16
4	15	27	26	25	24
3	14	23	32	31	30
2	13	22	29	35	34
1	12	21	28	33	36

Αν για παράδειγμα η τυχαία διαδικασία καταλήξει στο $4^{ο}$ στοιχείο της λίστας υποψηφιοτήτων σε κάθε πίνακα, θα επιλεγεί στοιχείο με διαφορετικές συντεταγμένες ως εξής:

στην περίπτωση του πίνακα μεγέθους 3 θα επιλεγεί το στοιχείο με συντεταγμένες (1,2)

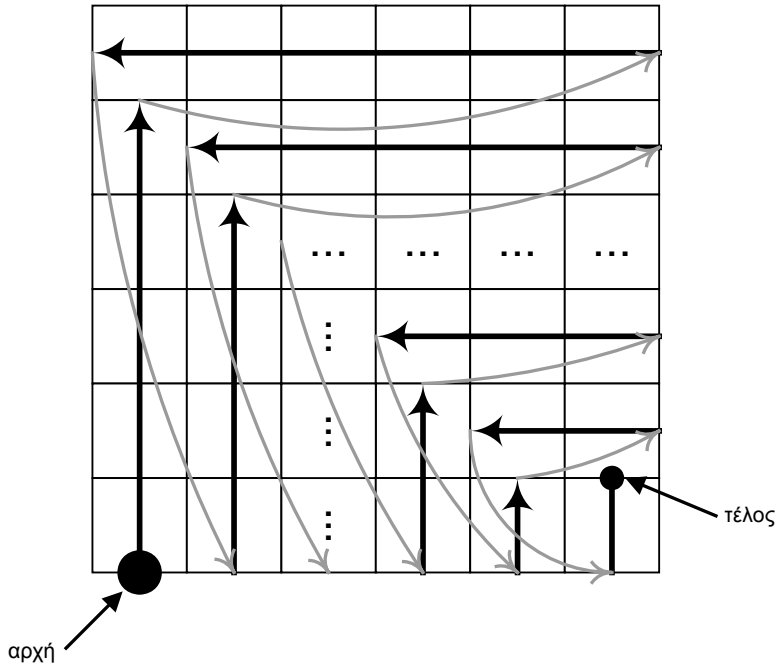
στην περίπτωση του πίνακα μεγέθους 4 θα επιλεγεί το στοιχείο με συντεταγμένες (1,4)

στην περίπτωση του πίνακα μεγέθους 3 θα επιλεγεί το στοιχείο με συντεταγμένες (2,1)

στην περίπτωση του πίνακα μεγέθους 3 θα επιλεγεί το στοιχείο με συντεταγμένες (3,1)

■

Παρατηρούμε ότι, για να προσπελάσει ο αλγόριθμος με αυτήν τη σειρά τα στοιχεία, χρειάζεται να πραγματοποιήσει μια διάσχιση, που μοιάζει με σπείρα. Το γενικό πρότυπο της σπειροειδούς διάσχισης, που υλοποιεί την διαδικασία maxmin με τυχαία επιλογή, είναι



7.7 Εξισώσεις συντεταγμένων

Η διάσχιση των στοιχείων του πίνακα, με τον τρόπο που περιγράφηκε, είναι χρονοβόρα διαδικασία, αν γίνει σειριακά. Ενώ πριν την εισαγωγή της τυχαιότητας για

την επιλογή ενός στοιχείου χρειαζόταν σταθερός χρόνος, τώρα χρειάζεται μια επαναληπτική διαδικασία m βημάτων, όπου m είναι η θέση του στοιχείου της λίστας υποψηφιοτήτων που επιλέχθηκε τυχαία. Ισχύει $1 \le m \le an^2$, όπου a είναι η παράμετρος του αλγορίθμου που καθορίζει το μέγεθος της λίστας και n το μέγεθος του προβλήματος. Παρατηρούμε λοιπόν ότι για την επιλογή ενός στοιχείου στη χειρότερη περίπτωση χρειάζεται τετραγωνικός χρόνος, έστω και αν η παράμετρος a έχει μικρή τιμή. Το πρόβλημα αυτό μπορεί να ξεπεραστεί με την εφεύρεση κατάλληλων συναρτήσεων, οι οποίες σε σταθερό χρόνο να δίνουν την τιμή της ετικέτας ενός στοιχείου γνωστών συντεταγμένων και αντίστροφα, τις συντεταγμένες ενός στοιχείου με γνωστή ετικέτα. Με τον τρόπο αυτό η γραμμική διάταξη, που εφαρμόστηκε στο κάθε υποπίνακα γίνεται τυχαίας προσπέλασης.

Στην παρακάτω ανάλυση και στους τύπους των συναρτήσεων που ακολουθούν, χρησιμοποιούνται οι εξής συμβολισμοί:

n είναι το μέγεθος του προβλήματος
e είναι η ετικέτα του στοιχείου
i είναι η τετμημένη του στοιχείου
j είναι η τεταγμένη του στοιχείου

Η θέση κάθε στοιχείου μέσα στη λίστα υποψηφιοτήτων αντιστοιχεί στην ετικέτα του στοιχείου μέσα στον πίνακα και ισούται με το πλήθος των στοιχείων, που χρειάζεται να προσπελάσει σειριακά μια μέθοδος για να φτάσει στο στοιχείο αυτό. Τα στοιχεία της κύριας διαγωνίου του πίνακα σπειροειδούς διάσχισης και οι αντίστοιχες ετικέτες τους φαίνονται στις παρακάτω αντιστοιχίες:

$$\begin{array}{lll}
(1,1) \rightarrow & (n-1)+n & = 2n-1 \\
(2,2) \rightarrow & (n-1)+n+(n-2)+(n-1) & = 4n-4 \\
(3,3) \rightarrow & (n-1)+n+(n-2)+(n-1)+(n-3)+(n-2) & = 6n-9 \\
\vdots & \vdots & \vdots \\
(n,n) \rightarrow & 2n^2-n^2 & = n^2
\end{array}$$

Γενικά ισχύει ότι το στοιχείο (i,i) έχει ετικέτα

$$e = 2ni - i^2 \tag{32}$$

Η ετικέτα του κάθε στοιχείου του πίνακα, που δεν ανήκει στην κύρια διαγώνιό του, καθορίζεται με βάση την ετικέτα του διαγώνιου στοιχείου στο οποίο αντιστοιχεί. Τα στοιχεία, για τα οποία ισχύει $i < j$, αντιστοιχούν στο στοιχείο (i,i) της διαγωνίου, ενώ τα στοιχεία, για τα οποία ισχύει $i > j$, αντιστοιχούν στο στοιχείο (j,j) της διαγωνίου. Τα στοιχεία που ανήκουν στην ίδια γραμμή με το στοιχείο (i,i) και βρίσκονται δεξιά από αυτό, έχουν ετικέτες μικρότερες από τη δική του κατά $j-i$ μονάδες και συγκεκριμένα

$$e = 2ni - i^2 + i - j \tag{33}$$

Τα στοιχεία που ανήκουν στην ίδια στήλη με το στοιχείο (j, j) και βρίσκονται κάτω από αυτό έχουν ετικέτες μικρότερες από τη δική του κατά $+(n-j)+(i-j)$ μονάδες και συγκεκριμένα

$$e = 2nj - j^2 + 2j - n - i \tag{34}$$

Άρα τελικά, δεδομένων των συντεταγμένων του στοιχείου, i και j, η ετικέτα του είναι:

$$e = \begin{cases} 2ni - i^2 + i - j & \text{αν } i \le j \\ 2nj - j^2 + 2j - n - i & \text{αν } i > j \end{cases} \tag{35}$$

Αν $i \le j$, το στοιχείο βρίσκεται στη κύρια διαγώνιο του πίνακα ή πάνω από αυτήν και άρα, η διάσχιση γίνεται επί της γραμμής από τα δεξιά προς τα αριστερά. Αν $i > j$, το στοιχείο βρίσκεται κάτω από την κύρια διαγώνιο του πίνακα και άρα, η διάσχιση γίνεται επί της στήλης από κάτω προς τα πάνω.

Αν στην εξίσωση (32) θεωρηθεί ως άγνωστη μεταβλητή το i, τότε προκύπτει μια δευτεροβάθμια εξίσωση η οποία έχει ρίζες

$$i_{1,2} = n \pm \sqrt{n^2 - e} \tag{36}$$

Όμως $1 \le e \le n^2$ και άρα, η υπόριζη ποσότητα είναι πάντα μη αρνητική. Συγκεκριμένα, $0 \le \sqrt{n^2 - e} \le \sqrt{n^2 - 1} < n$. Αφού όμως $1 \le i \le n$, αποκλείεται από την εξίσωση (36) η ρίζα που αντιστοιχεί στην πρόσθεση. Άρα έχουμε

$$i = n - d$$

όπου

$$d = \sqrt{n^2 - e}$$

Αυτό όμως ισχύει μόνο για τα στοιχεία της κύριας διαγωνίου, δηλαδή για $j = i$. Στην περίπτωση που $j > i$ και επειδή $0 < j - i < n - 1$, από την εξίσωση (33) έχουμε

$$\begin{gathered} i = \lceil n - d \rceil \\ \text{και} \\ j = 2ni - i^2 + i - e \end{gathered} \tag{37}$$

Αν $j < i$, από την εξίσωση (34) έχουμε

$$\begin{gathered} j = \lceil n - d \rceil \\ \text{και} \\ i = 2nj - j^2 + 2j - n - e \end{gathered} \tag{38}$$

Τις εξισώσεις (37) τις χρησιμοποιούμε όταν ισχύει

$$e \geq 2nc - c^2 + c - n$$

όπου

$$c = \lceil n - d \rceil$$

ενώ τις εξισώσεις (38) όταν ισχύει

$$e < 2nc - c^2 + c - n$$

Έχοντας στη διάθεσή μας αυτές τις εξισώσεις, η εύρεση των συντεταγμένων οποιουδήποτε στοιχείου του πίνακα διάσχισης μπορεί να γίνει σε σταθερό υπολογιστικό χρόνο.

7.8 Ψευδο-μετάθεση

Στους πίνακες σπειροειδούς διάσχισης μπορεί να γίνει η παρατήρηση ότι τα μικρότερα στοιχεία μαζεύονται στην πάνω αριστερή γωνία, ενώ τα μεγαλύτερα στην κάτω δεξιά. Το γεγονός αυτό συμβαίνει σε κάθε πίνακα που προκύπτει από το γινόμενο δύο γραμμών, που έχουν προηγουμένως ταξινομηθεί. Αυτό που αλλάζει είναι η διάταξη των δεικτών στον πίνακα $index$, δηλαδή η ανάθεση στην οποία αντιστοιχεί το κάθε στοιχείο του πίνακα. Στη διαδικασία maxmin, κάθε ένας από τους πίνακες των LAP που προκύπτουν, ακολουθούν αυτό το πρότυπο. Έτσι, η διαδικασία εύρεσης μιας άπληστης και τυχαίας λύσης είναι όμοια σε κάθε ένα από τα n^2 LAP, ανεξάρτητα από τις τιμές των στοιχείων του. Ο τρόπος διάσχισης είναι ανεξάρτητος από τη θέση του LAP μέσα στον πίνακα C. Αυτό που αλλάζει, είναι το αποτέλεσμα της διαδικασίας, δηλαδή η μετάθεση στην οποία καταλήγει η άπληστη, τυχαία μέθοδος.

Λόγω των παραπάνω, η σπειροειδής διάσχιση δε χρειάζεται να γίνει σε κάθε έναν υποπίνακα ξεχωριστά. Μάλιστα, μπορεί να πραγματοποιηθεί χωρίς καν να υπολογιστούν τα στοιχεία κάποιου απ' αυτούς. Μπορεί να γίνει μια φορά και στη συνέχεια να αντιστοιχηθεί στις τιμές και τους δείκτες του κάθε υποπίνακα. Έτσι, αρχικά, παράγεται μια ψευδο-μετάθεση που εξαρτάται μόνο από το μέγεθος του προβλήματος και δεν έχει σχέση με κανέναν συγκεκριμένο υποπίνακα. Στη συνέχεια, από αυτήν υπολογίζονται οι πραγματικές μεταθέσεις για κάθε έναν υποπίνακα. Η πραγματική μετάθεση κάθε υποπίνακα εξαρτάται από την ανάθεση που αντιστοιχεί σε κάθε στοιχείο του και άρα από τη σειρά που προέκυψε μετά την ταξινόμηση των δύο γραμμών που τον παράγουν.

8. Ο αλγόριθμος

Ο αλγόριθμος, που υλοποιεί την άπληστη (εξόδου), τυχαία, προσαρμόσιμη μέθοδο αναζήτησης (greedy out randomized adaptive search procedure – GoRASP), αποτελείται από δύο φάσεις. Η πρώτη είναι η φάση διάσχισης. Σε αυτήν αρχικά ταξινομείται ξεχωριστά η κάθε γραμμή των πινάκων F και D. Πλέον τα στοιχεία και η δομή όλων των υποπινάκων του πίνακα C, όπως αυτά περιγράφηκαν στην ενότητα της σπειροειδούς διάσχισης, μπορούν να υπολογιστούν από αυτές τις ταξινομημένες

γραμμές. Έτσι, η μέθοδος διατρέχει όλους τους υποπίνακες για να βρει ποιοι από αυτούς αντιπροσωπεύουν καλύτερα ολόκληρο τον πίνακα C και άρα και το συγκεκριμένο στιγμιότυπο του QAP. Για να το κάνει αυτό, πρώτα υπολογίζει τα στοιχεία μιας ψευδο-μετάθεσης. Εφαρμόζοντας αυτήν σε κάθε έναν υποπίνακα βρίσκει μια λύση απληστίας εξόδου και υπολογίζει την τιμή της αντικειμενικής συνάρτησης για τη λύση αυτή. Οι συντεταγμένες των υποπινάκων που δίνουν τις καλύτερες λύσεις αποθηκεύονται, ώστε να χρησιμοποιηθούν ως είσοδος στην επόμενη φάση.

Η δεύτερη φάση είναι η φάση της επαναληπτικής διαδικασίας. Κάθε βήμα της χωρίζεται σε δύο στάδια. Στο πρώτο από αυτά, υπολογίζεται μια άπληστη και τυχαία ψευδο-μετάθεση, ανεξάρτητη από τα στοιχεία όλων των υποπινάκων. Στη συνέχεια, αυτή η ψευδο-μετάθεση εφαρμόζεται σε κάθε ένα από τους υποπίνακες που αποτελούν την έξοδο της πρώτης φάσης. Έτσι, υπολογίζεται μια τυχαία λύση απληστίας εξόδου για τον καθένα, με την ελπίδα να βρίσκεται όσο το δυνατόν πιο κοντά στη βέλτιστη λύση του προβλήματος. Στο δεύτερο στάδιο, εκτελείται αναζήτηση για πιθανές βελτιώσεις στη γειτονιά της κάθε λύσης, που βρέθηκε στο πρώτο στάδιο. Η δομή της γειτονιάς που χρησιμοποιείται, είναι αυτή που προκύπτει από την αμοιβαία ανταλλαγή ζευγών της μετάθεσης (pair-exchange neighborhood) και η ενημέρωση γίνεται με βάση τον κανόνα του Heider. Η διαδικασία επαναλαμβάνεται, ώστε να εκμεταλλευτούμε το πλεονέκτημα που μας δίνει η τυχαιότητα και έτσι, να διευρυνθούν τα όρια της αναζήτησης.

8.1 Οι παράμετροι του αλγορίθμου

Το γενικό πλαίσιο εκτέλεσης του αλγορίθμου περιγράφηκε παραπάνω. Κάθε εκτέλεσή του όμως είναι διαφορετική και εξαρτάται από ένα σύνολο παραμέτρων, που καθορίζουν τις λεπτομέρειες των διαδικασιών του. Οι παράμετροι αυτές, καθώς και η σημασία τους παρουσιάζονται παρακάτω:

- Παράμετρος a. Παίρνει τιμές στο διάστημα $[0,1]$. Η τιμή της καθορίζει το μέγεθος της λίστας υποψηφιοτήτων. Με τον τρόπο αυτό, χαρακτηρίζει το βαθμό στον οποίο ο αλγόριθμος χρησιμοποιεί την απληστία και το βαθμό στον οποίο ο αλγόριθμος χρησιμοποιεί την τυχαιότητα. Έτσι, ο αλγόριθμος μπορεί να είναι από πλήρως άπληστος και καθόλου τυχαίος, για $a = 0$, έως εντελώς τυχαίος και καθόλου άπληστος, για $a = 1$.
- Παράμετρος *seed*. Είναι η αρχική τιμή που τροφοδοτεί τη γεννήτρια ψευδοτυχαίων αριθμών.
- Παράμετρος *numberofbests*. Αντιστοιχεί στο πλήθος των υποπινάκων που αντιπροσωπεύουν τον πίνακα C και αποτελούν την έξοδο της πρώτης φάσης, καθώς και μια από τις εισόδους της δεύτερης φάσης.
- Παράμετρος max*iterations*. Η τιμή της προσδιορίζει το μέγιστο πλήθος επαναλήψεων που θα κάνει η αναζήτηση στη δεύτερη φάση του αλγορίθμου. Ως εκ τούτου αποτελεί μια συνθήκη τερματισμού του αλγορίθμου.
- Παράμετρος *lookfor*. Αποτελεί την τιμή στην οποία στοχεύει ο αλγόριθμος για το συγκεκριμένο στιγμιότυπο. Αν βρεθεί μια λύση, για την οποία η αντικειμενική συνάρτηση υπολογίζει τιμή ίση ή μικρότερη από την τιμή στόχο, τότε η εκτέλεση του αλγορίθμου σταματά, ανεξάρτητα από το αν έχει ολοκληρώσει όλες τις επαναλήψεις του. Στην περίπτωση που $lookfor = -1$ και εφόσον τα δεδομένα όλων των στιγμιότυπων του QAP είναι θετικά, ο αλγόριθμος απλά αγνοεί αυτή τη συνθήκη τερματισμού.

8.2 Ψευδοκώδικας

Η περιγραφή του αλγορίθμου GoRASP σε μορφή ψευδοκώδικα φαίνεται στο παρακάτω σχήμα.

Αλγόριθμος GoRASP
Είσοδος: οι παράμετροι a, seed, numberofbests, maxiterations, και lookfor
Έξοδος: μια άπληστη (εξόδου), τυχαία, προσαρμόσιμη λύση του στιγμιότυπου QAP

1. διάβασε τα δεδομένα του προβλήματος
2. ταξινόμησε τις γραμμές των πινάκων F και D
3. εκτέλεσε τη φάση διάσχισης
4. εκτέλεσε τη φάση επανάληψης
5. παρουσίασε τα αποτελέσματα της εκτέλεσης

Τα δεδομένα του κάθε στιγμιότυπου, δηλαδή το μέγεθός του και τα στοιχεία των πινάκων F και D, διαβάζονται από αρχεία με συγκεκριμένη μορφοποίηση. Η ταξινόμηση των γραμμών των δύο πινάκων γίνεται με τον αλγόριθμο ταξινόμησης σωρού. Οι συναρτήσεις, που εκτελούν τη βασική λειτουργία του αλγορίθμου, είναι αυτές που περιγράφουν τις δύο φάσεις του.

Η πρώτη φάση του είναι η φάση της διάσχισης (traverse phase). Στόχος της είναι η εύρεση των *numberofbests* καλύτερων υποπινάκων. Συγκεκριμένα, επιστρέφει τις συντεταγμένες των υποπινάκων αυτών. Για το σκοπό αυτό, χρησιμοποιούνται τρεις πίνακες μεγέθους *numberofbests* . Ένας για τις τιμές των λύσεων, ένας για τις τετμημένες και ένας για τις τεταγμένες. Στην περιγραφή της παρακάτω διαδικασίας, οι τιμές των λύσεων, που αντιστοιχούν σε κάθε υποπίνακα, βρίσκονται στον πίνακα bests, οι τεταγμένες στον πίνακα bestsi και οι τετμημένες στον πίνακα bestsk.

διαδικασία traversePhase
Είσοδος: numberofbests
Έξοδος: οι συντεταγμένες των πινάκων – αντιπροσώπων του πίνακα C

1. υπολόγισε μια άπληστη, τυχαία ψευδο-μετάθεση
2. for i = 1…n
    a. for k = 1…n
        i. εφάρμοσε την ψευδο-μετάθεση στον υποπίνακα (i,k) του C
        ii. υπολόγισε το κόστος της μετάθεσης που προκύπτει, έστω costik
        iii. βρες την θέση της μεγαλύτερης τιμής του πίνακα bests, έστω x
        iv. if costik < bests(x)
            a) bests(x) = costik
            b) bestsi(x) = i
            c) bestsk(x) = k
        v. end if
    b. end for
3. end for
4. επέστρεψε τους πίνακες bestsi και bestsk

Η δεύτερη φάση του αλγορίθμου είναι η φάση της επανάληψης (iteration phase). Στην περιγραφή της παρακάτω διαδικασίας, χρησιμοποιείται η μεταβλητή bestvalue ως η τιμή της καλύτερης λύσης που έχει βρεθεί και η μεταβλητή bp ως η μετάθεση της καλύτερης λύσης που έχει βρεθεί.

διαδικασία iterationPhase
Είσοδος: οι παράμετροι numberofbests, maxiterations, και lookfor και οι μεταβλητές bestsi και bestsk
Έξοδος: η καλύτερη λύση που βρέθηκε (μετάθεση και τιμή)

1. for iterations = 1…maxiterations
    a. υπολόγισε μια άπληστη, τυχαία ψευδο-μετάθεση
    b. for x = 1…numberofbests
        i. εφάρμοσε την ψευδο-μετάθεση στον υποπίνακα με συντεταγμένες (bestsi(x), bestsk(x)) και βρες τη μετάθεση που προκύπτει, έστω p
        ii. υπολόγισε το κόστος της μετάθεσης p
        iii. εκτέλεσε τοπική αναζήτηση και επέστρεψε τη λύση της γειτονιάς με την καλύτερη τιμή, έστω value
        iv. if value < bestvalue
            a) bestvalue = value
            b) bp = p
        v. end if
        vi. if bestvalue <= lookfor
            a) τέλος εκτέλεσης
        vii. end if
    c. end for
2. end for
3. επέστρεψε τη μετάθεση και την τιμή της καλύτερης λύσης

Η διαδικασία, που επαναλαμβάνεται πιο συχνά και αποτελεί την καρδιά του αλγορίθμου, είναι αυτή που βρίσκει μια ψευδο-μετάθεση χρησιμοποιώντας απληστία εξόδου και τυχαιότητα. Η διαδικασία είναι γενική και δε διαφοροποιείται ανάλογα με τα δεδομένα κάθε υποπίνακα. Έτσι, δεν είναι απαραίτητος ο υπολογισμός των τιμών των στοιχείων του πίνακα, ούτε καν και του ίδιου του πίνακα. Για την αναπαράσταση του κάθε υποπίνακα χρησιμοποιούνται δύο συνδεδεμένες λίστες. Μια που περιέχει τους δείκτες των γραμμών, έστω listv, και μια των στηλών, έστω listh, που συμμετέχουν στο κάθε υποπρόβλημα. Μετά από κάθε ανάθεση που γίνεται, το τρέχον υποπρόβλημα μετασχηματίζεται σε ένα μικρότερο κατά μια μονάδα με την εξαγωγή από το αρχικό της γραμμής και της στήλης του στοιχείου που επιλέχθηκε. Στην περιγραφή της διαδικασίας χρησιμοποιούνται οι συμβολισμοί από την ενότητα εξισώσεις συντεταγμένων.

διαδικασία pseudoPermutation
Είσοδος: οι παράμετροι a και seed
Έξοδος: μια τυχαία και άπληστη (εξόδου) ψευδο-μετάθεση

1. while assignments < n
    a. e = τυχαία επιλογή ετικέτας
    b. $c = \lceil n - d \rceil$
    c. if $e \geq 2nc - c^2 + c - n$
        i. $i = \lceil n - d \rceil$
        ii. $j = 2ni - i^2 + i - e$
    d. else
        i. $j = \lceil n - d \rceil$
        ii. $i = 2nj - j^2 + 2j - n - e$
    e. end if
    f. selectionv = το i-οστό στοιχείο της λίστας listv
    g. selectionh = το j-οστό στοιχείο της λίστας listh
    h. κάνε την ανάθεση selectionh → selectionv
    i. assignments = assignments + 1
2. end while

8.3 Οι συναρτήσεις του κώδικα

Ο αλγόριθμος, που μόλις περιγράφηκε, υλοποιήθηκε σε γλώσσα προγραμματισμού Fortran 95/95. Στο κεφάλαιο αυτό παρουσιάζονται οι συναρτήσεις που δημιουργήθηκαν και η λειτουργία της καθεμιάς. Ο τρόπος παρουσίασης είναι "από κάτω προς τα πάνω", με τις πιο απλές συναρτήσεις να περιγράφονται πρώτες και τις πιο σύνθετες στη συνέχεια.

randomGenerator: είναι η γεννήτρια ψευδοτυχαίων αριθμών του Linus Schrage [94].

evaluateij: υπολογίζει τη μεταβολή της τιμής της αντικειμενικής συνάρτησης, που προκύπτει από την αμοιβαία ανταλλαγή των τοποθεσιών στις εγκαταστάσεις i και j.

local: εκτελεί την τοπική αναζήτηση στη γειτονιά κάθε λύσης. Χρησιμοποιεί την *evaluateij* για να υπολογίσει, αν μια γειτονική λύση βελτιώνει ή όχι την υπάρχουσα. Η ενημέρωση της κάθε λύσης γίνεται με τον κανόνα του Heider, που περιγράφηκε στην ενότητα των μεθόδων βελτίωσης.

saveSolution: αποθηκεύει σε κατάλληλες μεταβλητές τόσο την τιμή της καλύτερης λύσης, όσο και τη μετάθεση, που αντιστοιχεί σε αυτή την τιμή. Καλείται κάθε φορά που μια καλύτερη λύση βρίσκεται είτε στην πρώτη φάση, είτε στη δεύτερη.

findCost: υπολογίζει το κόστος που αντιστοιχεί σε μια συγκεκριμένη λύση – μετάθεση.

insertHeap: εισάγει ένα στοιχείο στη δομή ενός σωρού, με τρόπο που να διατηρείται η ιδιότητά του. Χρησιμοποιείται για την υλοποίηση της ταξινόμησης σωρού.

removeHeap: εξάγει ένα στοιχείο από τη δομή ενός σωρού, , με τρόπο που να διατηρείται η ιδιότητά του. Χρησιμοποιείται για την υλοποίηση της ταξινόμησης σωρού.

makeSort: ταξινομεί τις γραμμές των πινάκων F και D καθεμιά ξεχωριστά. Χρησιμοποιεί για το σκοπό αυτό τις δύο προηγούμενες συναρτήσεις.

spiralEquationMaxMin: υπολογίζει μέσα στον τρέχοντα υποπίνακα τη θέση του στοιχείου που θα επιλεγεί, χρησιμοποιώντας τις εξισώσεις συντεταγμένων που παρουσιάστηκαν στην ομώνυμη ενότητα

pseudoPermutation: υπολογίζει τις τιμές των αναθέσεων για να παράγει μια ψευδο-μετάθεση. Επαναληπτικά, κάνει επιλογές στοιχείων με τρόπο άπληστο και τυχαίο, βασιζόμενη στην παράμετρο a. Χρησιμοποιεί τη συνάρτηση *spiralEquationMaxMin*, ώστε να βρει τη θέση του υποπίνακα στην οποία αντιστοιχεί κάθε επιλογή.

traversePhase: υλοποιεί τη φάση διάσχισης του αλγορίθμου. Χρησιμοποιώντας τη συνάρτηση *pseudoPermutation* παράγει μια ψευδο-μετάθεση. Στη συνέχεια, διατρέχει κάθε έναν υποπίνακα και εφαρμόζει την ψευδο-μετάθεση σ' αυτόν για να παράγει μια πραγματική μετάθεση. Οι *numberofbests* καλύτερες μεταθέσεις – λύσεις, καθώς και οι συντεταγμένες των αντίστοιχων υποπινάκων, αποτελούν την έξοδο της συνάρτησης.

iterationPhase: υλοποιεί τη φάση επανάληψης του αλγορίθμου. Παράγει μια ψευδο-μετάθεση χρησιμοποιώντας τη συνάρτηση *pseudoPermutation* και την εφαρμόζει στους υποπίνακες που προέκυψαν από την πρώτη φάση. Για κάθε μια από τις λύσεις που βρίσκει, αναζητεί στη γειτονιά της καλώντας τη συνάρτηση *local*. Η διαδικασία επαναλαμβάνεται, έως ότου ένα κριτήριο λήξης ικανοποιηθεί. Κάθε φορά που βρίσκει μια καλύτερη λύση, την αποθηκεύει με τη βοήθεια της συνάρτησης *saveSolution*.

readProblem: διαβάζει από ένα αρχείο κειμένου τα δεδομένα που χαρακτηρίζουν κάθε στιγμιότυπου και συγκεκριμένα το μέγεθός του, n, και τα στοιχεία των πινάκων F και D.

output: τυπώνει τα αποτελέσματα του αλγορίθμου, που αποτελούνται από την καλύτερη λύση – μετάθεση που βρέθηκε και την τιμή που αντιστοιχεί σ' αυτήν.

qap: λύνει ένα στιγμιότυπο του QAP βασισμένη σε συγκεκριμένες τιμές των παραμέτρων του αλγορίθμου.

Το σύνολο αυτών των συναρτήσεων βρίσκεται στο αρχείο με όνομα *subroutines*. Ένα άλλο αρχείο με όνομα *driver*, είναι αυτό που τις χρησιμοποιεί και στην ουσία αποτελεί τη διασύνδεση του προγράμματος με το χρήστη, που θέλει να χρησιμοποιήσει τον κώδικα και κατ' επέκταση τον αλγόριθμο.

8.4 Οι βασικές μεταβλητές του κώδικα

Έκτός από τις συναρτήσεις, η γνώση των βασικών μεταβλητών, που χρησιμοποιεί ο αλγόριθμος, κάνει την κατανόησή του πιο εύκολη. Παρακάτω παρουσιάζονται οι βασικότερες από αυτές, οι τύποι τους, καθώς και ο σκοπός τους μέσα στον κώδικα.

n: ακέραια μεταβλητή, που δείχνει το μέγεθος του στιγμιότυπου του QAP.

f: πίνακας ακεραίων μεγέθους $n \times n$, που περιέχει τις τιμές του πίνακα ροών F.

d: πίνακας ακεραίων μεγέθους $n \times n$, που περιέχει τις τιμές του πίνακα αποστάσεων D.

fsi: πίνακας ακεραίων μεγέθους $n \times n$, που περιέχει τους δείκτες κάθε γραμμής του πίνακα F μετά την ταξινόμηση των στοιχείων της καθεμιάς. Τα στοιχεία του υπολογίζονται από τη συνάρτηση *makeSort*.

dsi: πίνακας ακεραίων μεγέθους $n \times n$, που περιέχει τους δείκτες κάθε γραμμής του πίνακα D μετά την ταξινόμηση των στοιχείων της καθεμιάς. Τα στοιχεία του υπολογίζονται από τη συνάρτηση *makeSort*.

p: πίνακας ακεραίων μεγέθους n, ο οποίος περιέχει τις τιμές των αναθέσεων της μετάθεσης που υπολογίζεται για κάθε υποπίνακα.

pseudop: πίνακας ακεραίων μεγέθους n, ο οποίος περιέχει τις τιμές των αναθέσεων της ψευδο-μετάθεσης. Υπολογίζεται από τη συνάρτηση *pseudoPermutation*.

bp: πίνακας ακεραίων μεγέθους n, ο οποίος περιέχει την καλύτερη μετάθεση που έχει βρεθεί κάθε στιγμή από τον αλγόριθμο. Ενημερώνεται από τη συνάρτηση *saveSolution*.

bestvalue: ακέραια μεταβλητή, που αποθηκεύει την τιμή της καλύτερης λύσης – μετάθεσης που έχει βρεθεί από τον αλγόριθμο. Ενημερώνεται από τη συνάρτηση *saveSolution*.

iterations: ακέραια μεταβλητή, που μετρά το πλήθος των επαναλήψεων που έχει κάνει η δεύτερη φάση του αλγορίθμου.

assignments: ακέραια μεταβλητή, που μετρά το πλήθος των αναθέσεων, που έχουν πραγματοποιηθεί σε μια μερική μετάθεση πριν ολοκληρωθεί η διαδικασία εύρεσης της κάθε ψευδο-μετάθεσης. Στο τέλος της διαδικασίας αυτής ισχύει *assignments* = *n*.

xrand: πραγματική μεταβλητή, που περιέχει το αποτέλεσμα της εκτέλεσης της γεννήτριας ψευδοτυχαίων αριθμών και αποτελεί την έξοδο της συνάρτησης *randomGenerator*.

besti: πίνακας ακεραίων μεγέθους *numberofbests*, που κρατά τις τεταγμένες των υποπινάκων που αποτελούν την έξοδο της πρώτης φάσης του αλγορίθμου και συγκεκριμένα της συνάρτησης *traversePhase*.

bestk: πίνακας ακεραίων μεγέθους *numberofbests*, που κρατά τις τετμημένες των υποπινάκων που αποτελούν την έξοδο της πρώτης φάσης του αλγορίθμου και συγκεκριμένα της συνάρτησης *traversePhase*.

nselect: ακέραια μεταβλητή, που δείχνει τη θέση μέσα στη λίστα υποψηφιοτήτων του στοιχείου που επιλέγεται με απληστία εξόδου και τυχαιότητα σε κάθε βήμα της δημιουργίας της ψευδο-μετάθεσης. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

selectionv: ακέραια μεταβλητή, που δείχνει την τεταγμένη μέσα στο δισδιάστατο πίνακα του στοιχείου που επιλέχθηκε. Η τιμή της υπολογίζεται από τη συνάρτηση *spiralEquationMaxMin.*

selectionh: ακέραια μεταβλητή, που δείχνει την τετμημένη μέσα στο δισδιάστατο πίνακα του στοιχείου που επιλέχθηκε. Η τιμή της υπολογίζεται από τη συνάρτηση *spiralEquationMaxMin.*

cvnext: πίνακας ακεραίων μεγέθους n, που συμβάλει στη δημιουργία συνδεδεμένης λίστας, η οποία δείχνει τις γραμμές του υποπίνακα που συμμετέχουν στο κάθε υποπρόβλημα. Συγκεκριμένα, περιέχει το δείκτη της επόμενης γραμμής κάθε γραμμής του πίνακα. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

cvprevious: πίνακας ακεραίων μεγέθους n, που συμβάλει στη δημιουργία συνδεδεμένης λίστας, η οποία δείχνει τις γραμμές του υποπίνακα που συμμετέχουν στο κάθε υποπρόβλημα. Συγκεκριμένα, περιέχει το δείκτη της προηγούμενης γραμμής κάθε γραμμής του πίνακα. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

chnext: πίνακας ακεραίων μεγέθους n, που συμβάλει στη δημιουργία συνδεδεμένης λίστας, η οποία δείχνει τις στήλες του υποπίνακα που συμμετέχουν στο κάθε υποπρόβλημα. Συγκεκριμένα, περιέχει το δείκτη της επόμενης στήλης κάθε στήλης του πίνακα. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

chprevious: πίνακας ακεραίων μεγέθους n, που συμβάλει στη δημιουργία συνδεδεμένης λίστας, η οποία δείχνει τις στήλες του υποπίνακα που συμμετέχουν στο κάθε υποπρόβλημα. Συγκεκριμένα περιέχει το δείκτη της προηγούμενης στήλης κάθε στήλης του πίνακα. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

initialv: ακέραια μεταβλητή, που δείχνει το πρώτο στοιχείο της λίστας των γραμμών. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

initialh: ακέραια μεταβλητή, που δείχνει το πρώτο στοιχείο της λίστας των στηλών. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

endv: ακέραια μεταβλητή, που δείχνει το τελευταίο στοιχείο της λίστας των γραμμών. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

endh: ακέραια μεταβλητή, που δείχνει το τελευταίο στοιχείο της λίστας των στηλών. Χρησιμοποιείται από τη συνάρτηση *pseudoPermutation*.

q: πίνακας ακεραίων μεγέθους n, που υλοποιεί τη δομή ενός σωρού ελαχίστων και χρησιμοποιείται στη ταξινόμηση των γραμμών των πινάκων F και D. Περιέχει τις τιμές των στοιχείων της κάθε γραμμής. Χρησιμοποιείται στις συναρτήσεις *insertHeap* και *removeHeap*.

iq: πίνακας ακεραίων μεγέθους n, που υλοποιεί τη δομή ενός σωρού ελαχίστων και χρησιμοποιείται στη ταξινόμηση των γραμμών των πινάκων F και D. Περιέχει τους δείκτες των στοιχείων της κάθε γραμμής. Χρησιμοποιείται στις συναρτήσεις *insertHeap* και *removeHeap*.

sizeq: ακέραια μεταβλητή, που δείχνει το πλήθος των στοιχείων που περιέχει κάθε στιγμή ο σωρός. Χρησιμοποιείται στις συναρτήσεις *insertHeap* και *removeHeap*.

v: ακέραια μεταβλητή, που περιέχει την τιμή του στοιχείου που μπαίνει στο σωρό στη συνάρτηση *insertHeap* ή βγαίνει από το σωρό στη συνάρτηση *removeHeap*.

iv: ακέραια μεταβλητή, που περιέχει το δείκτη του στοιχείου που μπαίνει στο σωρό στη συνάρτηση *insertHeap* ή βγαίνει από το σωρό στη συνάρτηση *removeHeap*.

9. Αριθμητικά αποτελέσματα

Από τη στιγμή που δεν υπάρχει, τουλάχιστον ακόμα, ευρετικός αλγόριθμος που να βρίσκει πάντα τη βέλτιστη λύση του QAP και άρα να υπερισχύει έναντι των υπολοίπων, κάθε νέα προσπάθεια δημιουργίας προσεγγιστικών αλγορίθμων μπορεί να κριθεί μόνο από τα αποτελέσματά της.

9.1 Η βιβλιοθήκη στιγμιότυπων του QAP

Η βιβλιοθήκη στιγμιότυπων του QAP, η οποία είναι γνωστή ως QAPLIB, αναπτύχθηκε από τους Rainer Burkard, Stefan Karisch και Franz Rendl [17]. Πρόκειται για μια συλλογή από όλα τα γνωστά στιγμιότυπα του προβλήματος, ομαδοποιημένα κατ' όνομα του κατασκευαστή τους. Για κάθε ένα αναφέρεται η καλύτερη λύση που έχει βρεθεί, καθώς και αν γνωρίζουμε ότι η λύση αυτή είναι η βέλτιστη. Αν όχι, αναφέρεται το μεγαλύτερο κατώτερο φράγμα που έχει βρεθεί για το συγκεκριμένο στιγμιότυπο. Για κάθε λύση αναφέρεται τόσο η μετάθεση, όσο και η τιμή της αντικειμενικής συνάρτησης για την μετάθεση αυτή. Επίσης, αναφέρεται και το όνομα του αλγορίθμου με τον οποίο βρέθηκε αυτή η λύση. Το όνομα κάθε στιγμιότυπου στην QAPLIB δηλώνει τον κατασκευαστή του, με τα πρώτα γράμματά του και το μέγεθός του, με τον αριθμό που ακολουθεί.

Κάθε ερευνητής μπορεί να βρει τα δεδομένα των στιγμιότυπων αυτών στην ηλεκτρονική διεύθυνση http://www.opt.math.tu-graz.ac.at/qaplib/. Με τον τρόπο αυτό, έχει δημιουργηθεί μια κοινή βάση για τη μέτρηση της αποτελεσματικότητας κάθε αλγορίθμου επίλυσης του QAP. Από την κατασκευή της QAPLIB και μετά, κάθε απόπειρα επίλυσης του Τετραγωνικού Προβλήματος Ανάθεσης καταλήγει με τη δοκιμή της νέας μεθόδου στα δεδομένα της βιβλιοθήκης αυτής. Αν η μέθοδος βελτιώνει κάποια από τα αποτελέσματα της βιβλιοθήκης γίνεται μέρος της, με την αναφορά του ονόματος του αλγορίθμου που χρησιμοποιεί ή ακόμη και με τη δημοσίευση του κώδικα υλοποίησής της.

9.2 Παράδειγμα εκτέλεσης

Ένα από τα μεγαλύτερα σε μέγεθος στιγμιότυπα της QAPLIB είναι το esc128. Όπως δείχνει και το όνομά του, έχει μέγεθος 128. Ένα παράδειγμα εξόδου του προγράμματος φαίνεται στο παρακάτω σχήμα, που δείχνει την εκτέλεση της συνάρτησης *driver* για το στιγμιότυπο αυτό.

```
---------greedy out randomized adaptive search procedure---------

input-------------------------------------------------------------

 qaplib\Eschermann\esc128.txt
 instance dimencion                    :                       128
 iteration phase parameter a           :                      0.25
 maximum number of iterations          :                       128
 look for                              :                        64
 number of representative submatices   :                         5
 initial seed                          :                    270001

output------------------------------------------------------------

 iterations                            :                         1
 best value                            :                        64
 best permutation found                :        48  51  94  81  47
                                       :        53  88  87  43  42
                                       :        40  16  61  99  20
                                       :         2  85 127  10 113
                                       :         9  28 112   5  36
                                       :       115  44  14  31  82
                                       :        63  34 116  65  83
                                       :        38 104  25 110  96
                                       :       126 121  18  89 128
                                       :        29  86  98 124  68
                                       :       101 111  37  45  19
                                       :       118  35 107  46  24
                                       :        33  64  39  22  66
                                       :        75  52 102  56  41
                                       :        70   4  80  69 109
                                       :        55  91  78  15 100
                                       :        79  73  60  72  50
                                       :        76  67 106  77  71
                                       :         3  17 119  49  12
                                       :       105 123  95   8   1
                                       :       108  26  13 117  32
                                       :       103  93  90 122  92
                                       :       114   7  97 120 125
                                       :        21  27  54  57  59
                                       :        84   6  11  58  30
                                       :        62  74  23
-----------------------------------------------------------------
```

9.3 Τα αποτελέσματα του GoRASP

Η υλοποίηση του GoRASP στη γλώσσα προγραμματισμού Fortran 90/95 μεταγλωττίστηκε με το G95 (Fortran 95 compiler) στο λειτουργικό σύστημα Windows XP professional edition. Κατά τη διαδικασία των πειραμάτων, εκτελέστηκε σε έναν υπολογιστή ACER TravelMate 291LCi με τα εξής χαρακτηριστικά:

- Intel Centrino Mobile 1.4 GHz
- DDR SDRAM 256 MB at 266 MHz
- Cache L1 1024 KB

Ο αλγόριθμος δοκιμάστηκε σε ένα μεγάλο πλήθος στιγμιότυπων από την QAPLIB και συγκεκριμένα σε 116 από αυτά. Η αποτελεσματικότητα της μεθόδου σε κάθε στιγμιότυπο κρίνεται από δύο παράγοντες. Πρώτον, από το πόσο καλή λύση βρίσκει και αν αυτή είναι η βέλτιστη και δεύτερον, από το χρόνο στον οποίο βρίσκει τη λύση αυτή. Για το λόγο αυτό, σε όλους τους παρακάτω πίνακες αποτελεσμάτων, για κάθε στιγμιότυπο, αναφέρεται τόσο η τιμή της λύσης που βρέθηκε όσο και το πλήθος των επαναλήψεων που χρειάστηκε ο αλγόριθμος για να βρει τη λύση αυτή. Επίσης αναφέρεται και η απόκλιση της κάθε λύσης από την καλύτερη που έχει καταχωρηθεί στην QAPLIB, έτσι ώστε στις περιπτώσεις που ο αλγόριθμος δε βρει την καλύτερη γνωστή λύση να φαίνεται η απόκλιση του αποτελέσματός του από αυτήν. Σε κάθε σχήμα, αναφέρονται πρώτα οι τιμές των παραμέτρων και στη συνέχεια, τα αποτελέσματα για τα 116 στιγμιότυπα της QAPLIB.

Μια από τις βασικές παραμέτρους που αλγορίθμου είναι η *numberofbests*, που αντιπροσωπεύει το πλήθος των υποπινάκων, που αντιπροσωπεύουν τον πίνακα C. Η τιμή αυτής της παραμέτρου δείχνει την επιθυμία του χρήστη είτε για καλύτερη λύση, είτε για λιγότερο χρόνο εκτέλεσης. Όσο το πλήθος των αντιπροσωπευτικών πινάκων μεγαλώνει, τόσο καλύτερες λύσεις βρίσκονται σε περισσότερο υπολογιστικό χρόνο. Όσο το πλήθος των αντιπροσωπευτικών πινάκων μικραίνει, τόσο ο χρόνο εκτέλεσης μειώνεται με αντιστάθμισμα να βρίσκονται λιγότερο καλές λύσεις. Ένα πάνω φράγμα για τον αλγόριθμο είναι η επιλογή της μεγαλύτερης δυνατής τιμής για την παράμετρο *numberofbests*, που ισούται με n^2, όπου n είναι το μέγεθος του κάθε στιγμιότυπου. Στην περίπτωση αυτή, όλοι οι υποπίνακες χρησιμοποιούνται ως αντιπρόσωποι του πίνακα C. Ο χρόνος εκτέλεσης περιορίζεται μόνο από το μέγιστο πλήθος επαναλήψεων (παράμετρος max*iterations*).

iteration phase parameter a :	0.50
maximum number of iterations :	1024
look for :	best
number of representative submatrices :	n^2
initial seed :	270001

α/α	όνομα στιγμιότυπου	καλύτερη λύση στην QAPLIB	καλύτερη λύση του αλγορίθμου	διαφορά	απόκλιση	πλήθος επαναλήψεων	ένδειξη
1	bur26a	5426670	5426670	0	0,000 %	0	√
2	bur26b	3817852	3817852	0	0,000 %	0	√

3	bur26c	5426795	5426795	0	0,000 %	0	√
4	bur26d	3821225	3821225	0	0,000 %	1	√
5	bur26e	5386879	5386879	0	0,000 %	0	√
6	bur26f	3782044	3782044	0	0,000 %	0	√
7	bur26g	10117172	10117172	0	0,000 %	0	√
8	bur26h	7098658	7098658	0	0,000 %	0	√
9	chr12a	9552	9552	0	0,000 %	1	√
10	chr12b	9742	9742	0	0,000 %	0	√
11	chr12c	11156	11156	0	0,000 %	8	√
12	chr15a	9896	9896	0	0,000 %	11	√
13	chr15b	7990	7990	0	0,000 %	6	√
14	chr15c	9504	9504	0	0,000 %	12	√
15	chr18a	11098	11098	0	0,000 %	15	√
16	chr18b	1534	1534	0	0,000 %	0	√
17	chr20a	2192	2192	0	0,000 %	281	√
18	chr20b	2298	2298	0	0,000 %	1024	√
19	chr20c	14142	14142	0	0,000 %	3	√
20	chr22a	6156	6156	0	0,000 %	638	√
21	chr22b	6194	6194	0	0,000 %	1024	√
22	chr25a	3796	3796	0	0,000 %	505	√
23	els19	17212548	17212548	0	0,000 %	2	√
24	esc16a	68	68	0	0,000 %	0	√
25	esc16b	292	292	0	0,000 %	0	√
26	esc16c	160	160	0	0,000 %	0	√
27	esc16d	16	16	0	0,000 %	0	√
28	esc16e	28	28	0	0,000 %	0	√
29	esc16f	0	0	0	0,000 %	0	√
30	esc16g	26	26	0	0,000 %	0	√
31	esc16h	996	996	0	0,000 %	0	√
32	esc16i	14	14	0	0,000 %	0	√
33	esc16j	8	8	0	0,000 %	0	√
34	esc32a	130	130	0	0,000 %	4	▲
35	esc32b	168	168	0	0,000 %	0	▲
36	esc32c	642	642	0	0,000 %	0	▲
37	esc32d	200	200	0	0,000 %	0	▲
38	esc32e	2	2	0	0,000 %	0	√
39	esc32f	2	2	0	0,000 %	0	√
40	esc32g	6	6	0	0,000 %	0	√
41	esc32h	438	438	0	0,000 %	0	▲
42	esc64a	116	116	0	0,000 %	0	▲
43	esc128	64	64	0	0,000 %	0	▲
44	had12	1652	1652	0	0,000 %	0	√
45	had14	2724	2724	0	0,000 %	0	√
46	had16	3720	3720	0	0,000 %	0	√
47	had18	5358	5358	0	0,000 %	0	√
48	had20	6922	6922	0	0,000 %	0	√
49	kra30a	88900	88900	0	0,000 %	5	√

50	kra30b	91420	91420	0	0,000 %	15	√
51	kra32	88700	88700	0	0,000 %	6	√
52	lipa20a	3683	3683	0	0,000 %	1	√
53	lipa20b	27076	27076	0	0,000 %	0	√
54	lipa30a	13178	13178	0	0,000 %	0	√
55	lipa30b	151426	151426	0	0,000 %	0	√
56	lipa40a	31538	31538	0	0,000 %	46	√
57	lipa40b	476581	476581	0	0,000 %	0	√
58	lipa50a	62093	62567	474	0,758 %	1024	X
59	lipa50b	1210244	1210244	0	0,000 %	0	√
60	lipa60a	107218	108096	878	0,812 %	1024	X
61	lipa60b	2520135	2520135	0	0,000 %	0	√
62	lipa70a	169755	171001	1246	0,729 %	1024	X
63	lipa70b	4603200	4603200	0	0,000 %	0	√
64	lipa80a	253195	254819	1624	0,637 %	1024	X
65	lipa80b	7763962	7763962	0	0,000 %	0	√
66	lipa90a	360630	362891	2261	0,623 %	1024	X
67	lipa90b	12490441	12490441	0	0,000 %	0	√
68	nug12	578	578	0	0,000 %	0	√
69	nug14	1014	1014	0	0,000 %	1	√
70	nug15	1150	1150	0	0,000 %	0	√
71	nug16a	1610	1610	0	0,000 %	0	√
72	nug16b	1240	1240	0	0,000 %	0	√
73	nug17	1732	1732	0	0,000 %	1	√
74	nug18	1930	1930	0	0,000 %	0	√
75	nug20	2570	2570	0	0,000 %	0	√
76	nug21	2438	2438	0	0,000 %	10	√
77	nug22	3596	3596	0	0,000 %	0	√
78	nug24	3488	3488	0	0,000 %	0	√
79	nug25	3744	3744	0	0,000 %	5	√
80	nug27	5234	5234	0	0,000 %	9	√
81	nug28	5166	5166	0	0,000 %	1	√
82	nug30	6124	6124	0	0,000 %	127	√
83	rou12	235528	235528	0	0,000 %	0	√
84	rou15	354210	354210	0	0,000 %	1	√
85	rou20	725522	725522	0	0,000 %	9	√
86	scr12	31410	31410	0	0,000 %	0	√
87	scr15	51140	51140	0	0,000 %	0	√
88	scr20	110030	110030	0	0,000 %	4	√
89	ste36a	9526	9526	0	0,000 %	1024	√
90	ste36b	15852	15852	0	0,000 %	3	√
91	ste36c	8239110	8239110	0	0,000 %	65	√
92	tai12a	224416	224416	0	0,000 %	0	√
93	tai12b	39464925	39464925	0	0,000 %	0	√
94	tai15a	388214	388214	0	0,000 %	1	√
95	tai15b	51765268	51765268	0	0,000 %	0	√
96	tai17a	491812	491812	0	0,000 %	10	√

97	tai20a	703482	703482	0	0,000 %	102	√
98	tai20b	122455319	122455319	0	0,000 %	0	√
99	tai25a	1167256	1167256	0	0,000 %	683	√
100	tai25b	344355646	344355646	0	0,000 %	4	√
101	tai30a	1818146	1818146	0	0,000 %	1024	√
102	tai30b	637117113	637117113	0	0,000 %	128	▲
103	tai35a	2422002	2439454	17452	0,715 %	1024	▼
104	tai35b	283315445	283315445	0	0,000 %	291	▲
105	tai40a	3139370	3174574	35204	1,109 %	1024	▼
106	tai40b	637250948	637250948	0	0,000 %	64	▲
107	tai50a	4938796	5041924	103128	2,045 %	1024	▼
108	tai50b	458821517	459087571	266054	0,058 %	1024	▼
109	tai60a	7205962	7373988	168026	2,279 %	1024	▼
110	tai60b	608215054	608628036	412982	0,068 %	1024	▼
111	tai64c	1855928	1855928	0	0,000 %	0	▲
112	tai80a	13515450	13806398	290948	2,107 %	1024	▼
113	tai80b	818415043	821799259	3384216	0,412 %	1024	▼
114	tai100a	21059006	21540684	481678	2,236 %	1024	▼
115	tai100b	1185996137	1191489353	5493216	0,461 %	1024	▼
116	tai150b	498896643	504217411	5320768	1,055 %	1024	▼

πλήθος επιτυχιών : **100 / 116**
μέσος όρος απόκλισης : **0.139 %**

Επεξήγηση ενδείξεων:

- √ : γνωρίζουμε τη βέλτιστη λύση
  ο αλγόριθμος βρήκε τη βέλτιστη λύση.
- x : γνωρίζουμε τη βέλτιστη λύση
  ο αλγόριθμος δεν βρήκε τη βέλτιστη λύση
- ▲ : δε γνωρίζουμε τη βέλτιστη λύση
  ο αλγόριθμος βρήκε την καλύτερη γνωστή λύση
- ▼ : δε γνωρίζουμε τη βέλτιστη λύση
  ο αλγόριθμος δε βρήκε την καλύτερη γνωστή λύση

Η παραπάνω επιλογή των παραμέτρων δίνει μεγάλη έμφαση στην εύρεση της καλύτερης λύσης, χωρίς να μεριμνεί για το συνολικό χρόνο εκτέλεσης. Αν μειώσουμε το πλήθος των αντιπροσωπευτικών υποπινάκων ο χρόνος εκτέλεσης μειώνεται κατά πολύ. Τα αποτελέσματα του παρακάτω πίνακα ισχύουν για $numberofbests = 10$. Έτσι, το πλήθος των αντιπροσωπευτικών υποπινάκων δεν αυξάνεται τετραγωνικά με το μέγεθος του στιγμιότυπου, αλλά είναι σταθερό. Ο χρόνος εκτέλεσης είναι στην τάξη των μερικών δευτερολέπτων ακόμη και για τα στιγμιότυπα μεγάλου μεγέθους.

iteration phase parameter a : 0.50
maximum number of iterations : 1024
look for : best
number of representative submatrices : 10
initial seed : 270001

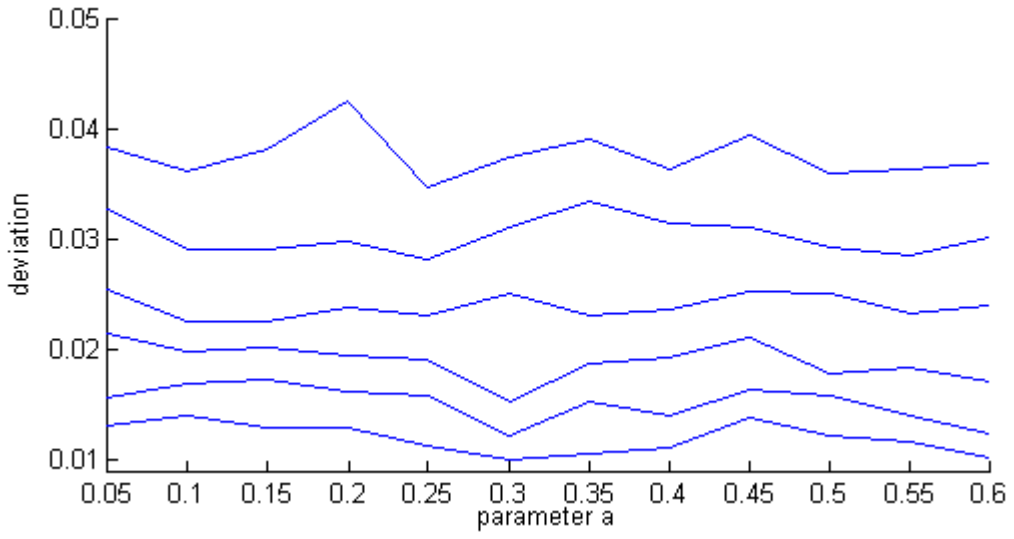
α/α	όνομα στιγμιότυπου	καλύτερη λύση στην QAPLIB	καλύτερη λύση του αλγορίθμου	διαφορά	απόκλιση	πλήθος επαναλήψεων	ένδειξη
1	bur26a	5426670	5426670	0	0.000 %	0	√
2	bur26b	3817852	3817852	0	0.000 %	0	√
3	bur26c	5426795	5426795	0	0.000 %	0	√
4	bur26d	3821225	3821225	0	0.000 %	1	√
5	bur26e	5386879	5386879	0	0.000 %	0	√
6	bur26f	3782044	3782044	0	0.000 %	0	√
7	bur26g	10117172	10117172	0	0.000 %	0	√
8	bur26h	7098658	7098658	0	0.000 %	0	√
9	chr12a	9552	9552	0	0.000 %	1	√
10	chr12b	9742	9742	0	0.000 %	0	√
11	chr12c	11156	11156	0	0.000 %	8	√
12	chr15a	9896	9896	0	0.000 %	11	√
13	chr15b	7990	7990	0	0.000 %	6	√
14	chr15c	9504	9504	0	0.000 %	12	√
15	chr18a	11098	11098	0	0.000 %	15	√
16	chr18b	1534	1534	0	0.000 %	0	√
17	chr20a	2192	2222	30	1.350 %	281	X
18	chr20b	2298	2398	100	4.170 %	1024	X
19	chr20c	14142	14142	0	0.000 %	3	√
20	chr22a	6156	6234	78	1.251 %	638	X
21	chr22b	6194	6338	144	2.272 %	1024	X
22	chr25a	3796	3884	88	2.266 %	505	X
23	els19	17212548	17212548	0	0.000 %	2	√
24	esc16a	68	68	0	0.000 %	0	√
25	esc16b	292	292	0	0.000 %	0	√
26	esc16c	160	160	0	0.000 %	0	√
27	esc16d	16	16	0	0.000 %	0	√
28	esc16e	28	28	0	0.000 %	0	√
29	esc16f	0	0	0	0.000 %	0	√
30	esc16g	26	26	0	0.000 %	0	√
31	esc16h	996	996	0	0.000 %	0	√
32	esc16i	14	14	0	0.000 %	0	√
33	esc16j	8	8	0	0.000 %	0	√
34	esc32a	130	130	0	0.000 %	4	▲
35	esc32b	168	168	0	0.000 %	0	▲
36	esc32c	642	642	0	0.000 %	0	▲
37	esc32d	200	200	0	0.000 %	0	▲
38	esc32e	2	2	0	0.000 %	0	√
39	esc32f	2	2	0	0.000 %	0	√
40	esc32g	6	6	0	0.000 %	0	√
41	esc32h	438	438	0	0.000 %	0	▲
42	esc64a	116	116	0	0.000 %	0	▲
43	esc128	64	64	0	0.000 %	0	▲
44	had12	1652	1652	0	0.000 %	0	√

45	had14	2724	2724	0	0.000 %	0	√
46	had16	3720	3720	0	0.000 %	0	√
47	had18	5358	5358	0	0.000 %	0	√
48	had20	6922	6922	0	0.000 %	0	√
49	kra30a	88900	88900	0	0.000 %	5	√
50	kra30b	91420	91420	0	0.000 %	15	√
51	kra32	88700	88700	0	0.000 %	6	√
52	lipa20a	3683	3683	0	0.000 %	1	√
53	lipa20b	27076	27076	0	0.000 %	0	√
54	lipa30a	13178	13178	0	0.000 %	0	√
55	lipa30b	151426	151426	0	0.000 %	0	√
56	lipa40a	31538	31894	356	1.116 %	46	X
57	lipa40b	476581	476581	0	0.000 %	0	√
58	lipa50a	62093	62689	596	0.951 %	1024	X
59	lipa50b	1210244	1210244	0	0.000 %	0	√
60	lipa60a	107218	108125	907	0.839 %	1024	X
61	lipa60b	2520135	2520135	0	0.000 %	0	√
62	lipa70a	169755	171015	1260	0.737 %	1024	X
63	lipa70b	4603200	4603200	0	0.000 %	0	√
64	lipa80a	253195	254915	1720	0.675 %	1024	X
65	lipa80b	7763962	7763962	0	0.000 %	0	√
66	lipa90a	360630	362910	2280	0.628 %	1024	X
67	lipa90b	12490441	12490441	0	0.000 %	0	√
68	nug12	578	578	0	0.000 %	0	√
69	nug14	1014	1014	0	0.000 %	1	√
70	nug15	1150	1150	0	0.000 %	0	√
71	nug16a	1610	1610	0	0.000 %	0	√
72	nug16b	1240	1240	0	0.000 %	0	√
73	nug17	1732	1732	0	0.000 %	1	√
74	nug18	1930	1930	0	0.000 %	0	√
75	nug20	2570	2570	0	0.000 %	0	√
76	nug21	2438	2438	0	0.000 %	10	√
77	nug22	3596	3596	0	0.000 %	0	√
78	nug24	3488	3488	0	0.000 %	0	√
79	nug25	3744	3744	0	0.000 %	5	√
80	nug27	5234	5234	0	0.000 %	9	√
81	nug28	5166	5166	0	0.000 %	1	√
82	nug30	6124	6124	0	0.000 %	127	√
83	rou12	235528	235528	0	0.000 %	0	√
84	rou15	354210	354210	0	0.000 %	1	√
85	rou20	725522	725582	60	0.008 %	9	X
86	scr12	31410	31410	0	0.000 %	0	√
87	scr15	51140	51140	0	0.000 %	0	√
88	scr20	110030	110030	0	0.000 %	4	√
89	ste36a	9526	9612	86	0.895 %	1024	X
90	ste36b	15852	15852	0	0.000 %	3	√
91	ste36c	8239110	8274054	34944	0.422 %	65	X

92	tai12a	224416	224416	0	0.000 %	0	√
93	tai12b	39464925	39464925	0	0.000 %	0	√
94	tai15a	388214	388214	0	0.000 %	1	√
95	tai15b	51765268	51765268	0	0.000 %	0	√
96	tai17a	491812	491812	0	0.000 %	10	√
97	tai20a	703482	705622	2140	0.303 %	102	X
98	tai20b	122455319	122455319	0	0.000 %	0	√
99	tai25a	1167256	1182486	15230	1.288 %	683	X
100	tai25b	344355646	344355646	0	0.000 %	4	√
101	tai30a	1818146	1845256	27110	1.469 %	1024	X
102	tai30b	637117113	637117113	0	0.000 %	128	▲
103	tai35a	2422002	2467548	45546	1.846 %	1024	▼
104	tai35b	283315445	283844480	529035	0.186 %	291	▼
105	tai40a	3139370	3201848	62478	1.951 %	1024	▼
106	tai40b	637250948	637336250	85302	0.013 %	64	▼
107	tai50a	4938796	5068454	129658	2.558 %	1024	▼
108	tai50b	458821517	459279957	458440	0.100 %	1024	▼
109	tai60a	7205962	7383066	177104	2.399 %	1024	▼
110	tai60b	608215054	608813843	598789	0.098 %	1024	▼
111	tai64c	1855928	1855928	0	0.000 %	0	▲
112	tai80a	13515450	13858646	343196	2.476 %	1024	▼
113	tai80b	818415043	819753494	1338451	0.163 %	1024	▼
114	tai100a	21059006	21553660	494654	2.295 %	1024	▼
115	tai100b	1185996137	1190050491	4054354	0.341 %	1024	▼
116	tai150b	498896643	504217411	5320768	1.055 %	1024	▼

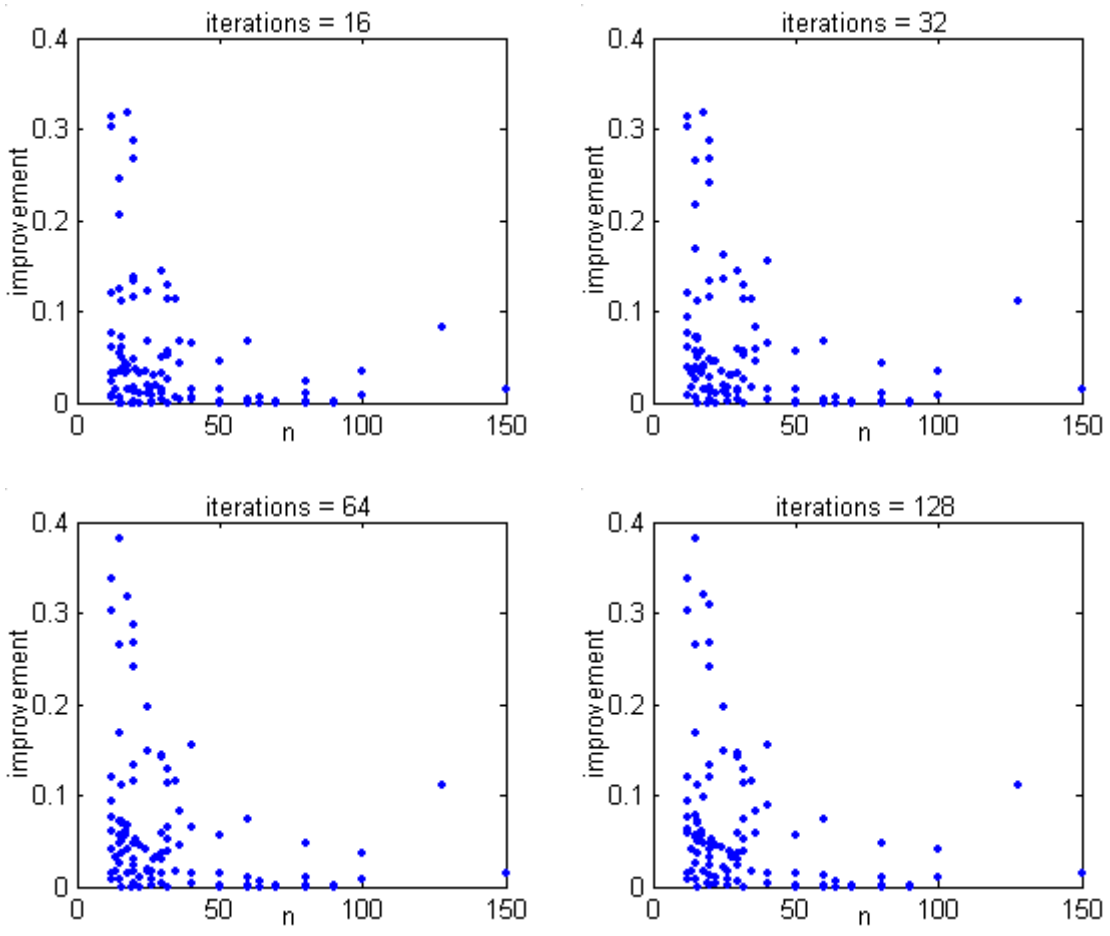
πλήθος επιτυχιών : **86 / 116**
μέσος όρος απόκλισης : **0.311 %**

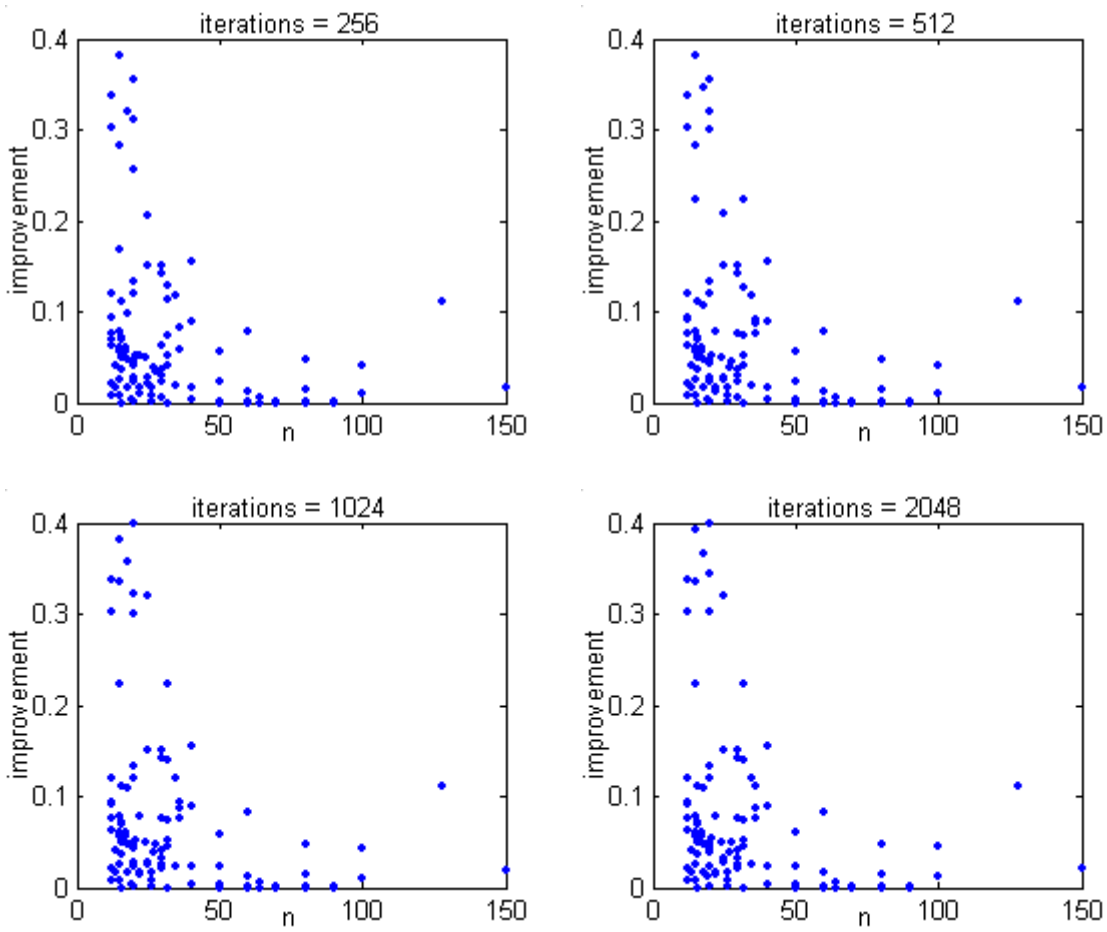
Στους προηγούμενους δύο πίνακες η τιμή της παραμέτρου *a* είναι 0.50. Ο αλγόριθμος δοκιμάστηκε διεξοδικά για πολλές τιμές της παραμέτρου αυτής. Τα αποτελέσματα των εκτελέσεων φαίνονται στο παρακάτω γράφημα.



Από πάνω προς τα κάτω φαίνονται οι γραμμές που αντιπροσωπεύουν τους μέσους όρους απόκλισης για πλήθος επαναλήψεων 16, 32, 64, 128, 256 και 512, αντίστοιχα. Οι τιμές της παραμέτρου a κυμαίνονται από 0.05 έως 0.60. Παρατηρούμε ότι η επιλογή της τιμής της παραμέτρου δεν επιφέρει σημαντικές αλλαγές στο αποτέλεσμα. Καμιά από τις τιμές αυτές δεν υπερισχύει έναντι των άλλων ως προς την ποιότητα των λύσεων που παράγει. Επίσης, λόγω της χρήσης των εξισώσεων συντεταγμένων, για την εύρεση, μέσα στο δισδιάστατο υποπίνακα, του στοιχείου που επιλέγεται κάθε φορά, η αύξηση της τιμής της παραμέτρου a έχει αμελητέα επίδραση στο χρόνο εκτέλεσης του προγράμματος που υλοποιεί τον αλγόριθμο.

Ένα άλλο ενδιαφέρον χαρακτηριστικό του αλγορίθμου είναι ο βαθμός βελτίωσης της καλύτερης λύσης από τη φάση διάσχισης στη φάση επανάληψης. Στα παρακάτω γραφήματα φαίνεται ο λόγος $(bvb - bva)/bvb$ όπου bvb είναι η καλύτερη τιμή, που έχει βρει ο αλγόριθμος πριν τη φάση επανάληψης (best value before) και bva η καλύτερη τιμή, που έχει βρει ο αλγόριθμος μετά τη φάση επανάληψης (best value after). Ο λόγος παρουσιάζεται για κάθε ένα από τα 116 στιγμιότυπα και για πλήθος επαναλήψεων 16, 32, 128, 256, 512, 1024 και 2048.





Παρατηρείται ότι η βελτίωση είναι πολύ μεγαλύτερη για στιγμιότυπα μικρού μεγέθους και μειώνεται καθώς το μέγεθος του στιγμιότυπου μεγαλώνει. Μάλιστα φαίνεται να σχηματίζεται ένα πρότυπο αντίστροφης λογαριθμικής συνάρτησης, η οποία είναι ασύμπτωτη στους δύο άξονες.

10. Επίλογος

Το Τετραγωνικό Πρόβλημα Ανάθεσης (Quadratic Assignment Problem – QAP), ανήκει στα πιο γνωστά Προβλήματα Συνδυαστικής Βελτιστοποίησης. Ένα πλήθος αλγορίθμων εύρεσης της ακριβούς λύσης του έχουν αναπτυχθεί με κοινό μειονέκτημα ότι δεν μπορούν να λύσουν στιγμιότυπα του προβλήματος με μέγεθος μεγαλύτερο του 30 σε λογικό υπολογιστικό χρόνο. Για το λόγο αυτό, έχουν αναπτυχθεί και αρκετοί προσεγγιστικοί αλγόριθμοι. Σκοπός τους είναι η εύρεση σε σχετικά μικρό υπολογιστικό χρόνο μιας καλής λύσης, έστω κι αν αυτή δεν είναι η βέλτιστη. Ένας νέος, τέτοιος, προσεγγιστικός αλγόριθμος αναπτύχθηκε στην εργασία αυτή. Υλοποιεί μια άπληστη (εξόδου), τυχαία, προσαρμόσιμη διαδικασία αναζήτησης (greedy out randomized adaptive search procedure). Η απληστία εξόδου έχει υλοποιηθεί ως μια maxmin διαδικασία. Η τεχνική σπειροειδούς διάσχισης, η δημιουργία των εξισώσεων συντεταγμένων και η χρήση ψευδο-μετάθεσης έχουν μειώσει σημαντικά τον απαιτούμενο υπολογιστικό χρόνο του αλγορίθμου.

Αναφορές

[1] Adams W. P. and Johnson T. A., "Improved linear programming-based lower bounds for the quadratic assignment problem", in Quadratic Assignment and Related Problems, P. M. Pardalos and H. Wolkowicz, eds., DIMACS Series on Discrete Mathematics and Theoretical Computer Science 16, 1994, 43-75, AMS, Providence, RI.

[2] Ahuja R. K., Orlin J. B. and Tivari A., "A greedy genetic algorithm for the quadratic assignment problem", Working paper 3826-95, Sloan School of Management, MIT, 1995.

[3] Assad A. A. and Xu W., "On lower bounds for a class of quadratic 0-1 programs", Operations Research Letters 4, 1985, 175-180.

[4] Azim G. A., "Neural Networks for Solving Quadratic Assignment Problems", Neural Information Processing – Letters and Reviews, Vol. 10, No. 3, 2006.

[5] Balas E. and Mazzola J. B., "Nonlinear programming: I. Linearization techniques", Mathematical Programming 30, 1984, 1-21.

[6] Balas E. and Mazzola J. B., "Nonlinear programming: II. Dominance relations and algorithms", Mathematical Programming 30, 1984, 22-45.

[7] Balas E. and Mazzola J. B., "Quadratic 0-1 programming by a new linearization", presented at the Joint ORSA / TIMS National Meeting, 1980, Washington D. C.

[8] Barazaa M. S. and Sherali H. D., "Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem", Naval Research Logistics Quarterly 27, 1980, 29-41.

[9] Barazaa M. S. and Sherali H. D., "On the use of exact and heuristic cutting plane methods for the quadratic assignment problem", Journal of Operations Research Society 33, 1982, 991-1003.

[10] Battiti R. and Tecchioli G., "The reactive tabu search", ORSA Journal of Computing 6, 1994, 126-140.

[11] Bokhari S. H., "Assignment problems in parallel and distributed computing", Kluwer Academic Publishers, Norwell, MA, 1987.

[12] Bollobás B., "Extremal Graph Theory", Academic Press, London, 1978.

[13] Buffa E. S., Armour G. C. and Vollmann T. E., "Allocating facilities with CRAFT", Harvard Business Review 42, 1962, 136-158.

[14] Burkard R. E., “Die Störungsmethode zur Losung quadratischer Zuordnungs-probleme”, Operations Research Verfahren 16, 1973, 84-108.

[15] Burkard R. E. and Bonniger T., “A heuristic for quadratic boolean programs with applications to quadratic assignment problems”, European Journal of Operational Research 13, 1983, 374-386.

[16] Burkard R. E. and Fincke U., “On random quadratic bottleneck assignment problems”, Mathematical Programming 23, 1982, 227-232.

[17] Burkard R. E., Karisch S. E. and Rendl F., “QAPLIB – a quadratic assignment problem library”, Journal of Global Optimization 10, 1997, 391-403. An on-line version is available via World Wide Web at the following URL: http://www.opt.math.tu-graz.ac.at/qaplib/.

[18] Burkard R. E. and Offermann J., “Entwurf von Schreibmaschinen - Tastaturen mittels quadratischen Zuordnungsprobleme”, Zeitschrift für Operations Research 21, 1977, B121-B132.

[19] Burkard R. E. and Rendl F., “A thermodynamically motivated simulation procedure for combinatorial optimization problems”, European Journal Operational Research 17, 1984, 169-174.

[20] Carraresi P. and Malucelli F., “A new lower bound for the quadratic assignment problem”, Operations Research 40, 1992, Suppl. No. 1, S22-S27.

[21] Çela E., “The Quadratic Assignment Problem: Theory and Algorithms”, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[22] Christofides N. and Benavent E., “An exact algorithm for the quadratic assignment problem”, Operations Research 37, 1989, 760-768.

[23] Colorni A., Dorigo M. and Maniezzo V., “The ant system: optimization by a colony of cooperating agents”, IEEE Transactions on Systems, Man, and Cybernetics – Part B 26, 1996, 29-41.

[24] Connolly D. T., “An improved annealing scheme for the QAP”, European Journal of Operational Research 46, 1990, 93-100.

[25] Davis L., “Genetic Algorithms and Simulated annealing”, Pitman, London, 1987.

[26] Dickey J. W. and Hopkins J. W., “Campus building arrangement using TOPAZ”, Transportation Research 6, 1972, 59-68.

[27] Dorigo M., “Optimization, Learning and Natural algorithms”, Ph. D. Thesis, Dipartimento die Elettronica e Informazione, Politecnico di Milano, Milano, Italy, 1992.

[28] Edwards C. S., “A branch and bound algorithm for the Koopmans-Beckmann quadratic assignment problem”, Mathematical Programming Study 13, 1980, 35-52.

[29] Edwards C. S., “The derivation of a greedy approximator for the Koopmans-Beckmann quadratic assignment problem”, Proceedings of the 77-th Combinatorial Programming Conference (CP77), 1977, 55-86.

[30] Elshafei A. N., “Hospital layout as a quadratic assignment problem”, Operations Research Quarterly 28, 1977, 167-179.

[31] Feo T. A. and Resende M. G. C., “Greedy randomizes adaptive search procedures”, Journal of Global Optimization 6, 1995, 109-133.

[32] Finke G., Burkard R. E. and Rendl F., “Quadratic assignment problems”, Annals of Discrete Mathematics 31, 1987, 61-82.

[33] Fleurent C. and Ferland J., “Genetic hybrids for the quadratic assignment problem”, in Quadratic Assignment and Related Problems, P. Pardalos and H. Wolkowicz, eds., DIMACS Series in Discrete Mathematics and Theoretical Computing Science 16, 1994, 173-187, AMS, Providence, RI.

[34] Francis R. L. and White J. A., “Facility layout and location”, Prentice-Hall, Englewood Cliffs, N. J., 1974.

[35] Frieze A. M. and Yadegar J., “On the quadratic assignment problem”, Discrete Applied Mathematics 5, 1983, 89-98.

[36] Frieze A. M., Yadegar J., El-Horbaty S. and Parkinson D., “Algorithms for assignment problems on an array processor”, Parallel Computing 11, 1989, 151-162.

[37] Gambardella L. M., Taillard E. D. and Dorigo M., “Ant colonies for the QAP”, Technical Report IDSIA-4-97, 1997, Instituto dale Molle Di Studi sull’ Intelligenza Artificiale, Lugano, Switzerland.

[38] Gavett J. W. and Plyter N. V., “The optimal assignment of facilities to locations by branch and bound”, Operations Research 14, 1996, 210-232.

[39] Geoffrion A. M. and Graves G. W., “Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment / LP approach”, Operations Research 24, 1976, 595-610.

[40] Gilmore P. C., "Optimal and suboptimal algorithms for the quadratic assignment problem", SIAM Journal in Applied Mathematics 10, 1962, 305-313.

[41] Glover F., "Tabu search – Part I", ORSA Journal on Computing 1, 1989, 190-206.

[42] Glover F., "Tabu search – Part II", ORSA Journal on Computing 2, 1989, 4-32.

[43] Glover F., M. Laguna, E. Taillard and D. de Werra, eds., "Tabu search", Annals of Operational Research 41, 1993.

[44] Graham A., "Kronecker Products and Matrix Calculus with Applications", Halsted Press, Toronto, 1981.

[45] Hadley S. W., Rendl F. and Wolkowicz H., "Non-symmetric quadratic assignment problems and the Hoffman-Wielandt inequality", Linear Algebra and its Applications 58, 1992, 109-124.

[46] Hahn P. and Grant T., "Lower bounds for the quadratic assignment problem based upon a dual formulation", to appear in European Journal of Operational Research.

[47] Hahn P., Grant T. and Hall N., "Solution of the quadratic assignment problem using the Hungarian method", to appear in European Journal of Operational Research.

[48] Handley S. W., "Continuous Optimization Approaches for the Quadratic Assignment Problem", PhD thesis, University of Waterloo, Ontario, Canada, 1989.

[49] Handley S. W., Rendl F. and Wolkowicz H., "A new lower bound via projection for the quadratic assignment problem", Mathematics of Operations Research 17, 1992, 727-739.

[50] Handley S. W., Rendl F. and Wolkowicz H., "Bounds for the quadratic assignment problem using continuous optimization techniques", Proceedings of the 1-st Integer Programming and Combinatorial Optimization Conference (IPCO), University of Waterloo Press, 1990, 237-248.

[51] Heffley D. R., "Assigning runners to a relay team", in Optimal Strategies in Sports, S. P. Ladany and R. E. Machol. Eds., North-Holland, Amsterdam, 1977, 169-171.

[52] Heider C. H., "A computationally simplified pair exchange algorithm for the quadratic assignment problem", Paper No. 101, Center for Naval Analysis, Arlington, Virginia, 1972.

[53] Holland J. H., “Adaptation in Natural and Artificial Systems”, University of Michigan Press, Ann Arbor, 1975.

[54] Hopfield J. and Tank D., “Computing with neural circuits: A model”, Science 233, 625-633, 1986.

[55] Hopfield J. and Tank D., “Neural computation of decisions in optimization problems”, Biol. Cybern. 52, 141-152, 1985.

[56] Hopfield J., “Learning algorithms and probability distributions in feed-forward and feed-back networks”, PNAS 84, 8429-8433, 1987.

[57] Hopfield J., “Neural networks and physical systems with emergent collective computational abilities”, PNAS 79, 2554, 1982.

[58] Hopfield J., “On the effectiveness of neural networks in computation. Symposium, Parallel models of intelligence: How can slow components think so fast?”, AAAI 135-138, March 22-24, 1988.

[59] Hubert L. G., “Assignment methods in combinatorial data analysis”, Marcel Dekker, Inc., New York, NY 10016, 1987.

[60] Ikeguchi T., Sato K., Hasegawa M. and Aihara K., “Chaotic Optimization for Quadratic Assignment Problems”, Proceedings of international symposium on circuits and systems, 3. 469-472, 2002.

[61] Johnson D. S., Papadimitriou C. H. and Yannakakis M., “How easy is local search”, Journal of Computer and System Sciences 37, 1988, 79-100.

[62] Karisch S. E. and Rendl F., “Lower bounds for the quadratic assignment problem via triangle decompositions”, Mathematical Programming 71, 1995, 137-151.

[63] Karlson P. and Lüscher M., “Pheromones: a new term for a class of biologically active substances”. Nature 183, 1959, 55-56.

[64] Kaufmann L. and Broeckx F., “An algorithm for the quadratic assignment problem using Benders’ decomposition”, European Journal of Operational Research 2, 1978, 204-211.

[65] Kirkpatrick S., Gelatt C. D. and Vecchi M. P., “Optimization by simulated annealing”, Science 220, 1983, 671-680.

[66] Koopmans T. C. and Beckmann M. J., “Assignment problems and the location of economic activities”, Econometrica 25, 1957, 53-76.

[67] Krarup J. and Pruzan P. M., "Computer-aided layout design", Mathematical Programming Study 9, 1978, 75-94.

[68] Kuhn H. W., "The Hungarian method for the assignment problem", Naval Research Logistics Quarterly, 1955, 2:83–87.

[69] Land A. M., "A problem of assignment with interrelated costs", Operations Research Quarterly 14, 1963, 185-198.

[70] Lawler E. L., "The quadratic assignment problem", Management Science 9, 1963, 586-599.

[71] Lawler E. L., Lestra J. K., Rinnooy Kan A. H. G. and Shmoys D. B., eds., "The Traveling Salesman Problem", Wiley, Chichester, 1985.

[72] Li Y. and Pardalos P. M., "Generating quadratic assignment test problems with known optimal permutations", Computational Optimization and Applications 1, 1992, 163-184.

[73] Mautor T. and Roucairol C., "A new exact algorithm for the solution of quadratic assignment problems", Discrete Applied Mathematics 55, 1992, 281-293.

[74] Mavridou T., Pardalos P. M., Pitsoulis L. S. and Resende M. G. C., "A GRASP for the quadratic assignment problem", European Journal of Operations Research 105, 1998, 613-621.

[75] McCormik E. J., "Human factors engineering", McGraw-Hll, New York, 1970.

[76] Metropolis N., Rosenbluth A., Rosenbluth M., Teller A. and Teller E., "Equations of state calculations by fast computing machines", Journal of Chemical Physics 21, 1953, 1087-1092.

[77] Mirchandani P. B. and Obata T., "Locational decisions with interactions between facilities: the quadratic assignment problem a review", Working Paper Ps-79-1, Rensselaer Polytechnic Institute, Troy, New York, May 1979.

[78] Munkres J. R., "Algorithm for the assignment and transportation problems", Journal of the Society for Industrial and Applied Mathematics, 1957, Vol. 5, No. 1, pp. 32-38.

[79] Murthy K. A., Pardalos P. and Li Y., "A local search algorithm for the quadratic assignment problem", Informatica 3, 1992, 524-538.

[80] Nugent C. E., T. E. Vollmann and J Ruml, "An experimental comparison of techniques for the assignment of facilities to locations", Journal of Operations Research 16, 1969, 150-173.

[81] Palubeckis G. S., “Generation of quadratic assignment test problems with known optimal solutions”, U. S. S. R. Comput. Maths. Math. Phys. 28, 1988, 97-98.

[82] Papadimitriou C. H. and Steiglitz K., “Combinatorial optimization: algorithms and complexity”, Prentice-Hall, Inc., Upper Saddle River, NJ, 1982.

[83] Papadimitriou C. H. and Wolfe D., “The complexity of facets resolved”, Proceedings of the 25-th Annual IEEE Symposium on the Foundations of Computer Science (FOCS), 1985, 74-78.

[84] Pardalos P. M., Pitsoulis L. S. and Resende M. G. C., “A parallel GRASP implementation for solving the quadratic assignment problem”, in Parallel Algorithms for Irregular Problems: State of the Art, A. Ferreira and José D. P. Rolim, eds., Kluwer Academic Publishers, 1995, 115-133.

[85] Pardalos P. M., Pitsoulis L. S. and Resende M. G. C., “Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP”, ACM Transactions on Mathematical Software 23, 1997, 196-208.

[86] Pardalos P. M., Ramakrishnan K. G., Resende M. G. C. and Li Y., “Implementation of a variable reduction based lower bound in a branch and bound algorithm for the quadratic assignment problem”, SIAM Journal on Optimization 7, 1997, 280-294.

[87] Quayranne M., “Performance ratio of heuristics for triangle inequality quadratic assignment problems”, Operations Research Letters 4, 1986, 231-234.

[88] Rendl F., “Ranking scalar products to improve bounds for the quadratic assignment problem”, European Journal of Operations Research 20, 1985, 363-372.

[89] Rendl F. and Wolkowicz H., “Applications of parametric programming and eigenvalue maximization to the quadratic assignment problem”, Mathematical Programming 53, 1992, 63-78.

[90] Rhee W. T., “Stochastic analysis of the quadratic assignment problem”, Mathematics of Operations Research 16, 1991, 223-239.

[91] Roucairol C., “A parallel branch and bound algorithm for the quadratic assignment problem”, Discrete Applied Mathematics 18, 1987, 221-225.

[92] Roucairol C., “A reduction method for quadratic assignment problems”, Operations Research Verfahren 32, 1979, 183-187.

[93] Sahni S. and Gonzalez T., "P-complete approximation problems", Journal of the Association of Computing Machinery 23, 1976, 555-565.

[94] Schrange L., "A more portable Fortran random number generator", ACM Transactions on Mathematical Software 5, 1979, 132-138.

[95] Skorin-Kapov J., "Extensions of tabu search adaptation to the quadratic assignment problem", to appear in Computers and Operations Research.

[96] Skorin-Kapov J., "Tabu search applied to the quadratic assignment problem", ORSA Journal on Computing 20, 1991, 56-87.

[97] Steinberg L., "The backboard wiring problem: A placement algorithm", SIAM Review 3, 1961, 37-50.

[98] Taillard E., "Robust tabu search for the quadratic assignment problem", Parallel Computing 17, 1991, 443-455.

[99] Tate D. M. and Smith A. E., "A genetic approach to the quadratic assignment problem", Computers and Operations Research 22, 1995, 73-83.

[100] Ugi I., Bauer J., Friedrich J., Gasteiger J., Jochum C. and Schubert W., "Neue Anwendungsgebiete für Computer in der Chemie", Angewandte Chemie 91, 1979, 99-111.

[101] Whitney H., "Differentiable Manifolds in Euclidean Space", PNAS 1935, 21: 462-464.

[102] Whitney H., "On the abstract properties of linear dependence". American Journal of Mathematics, vol. 57, 1935, pp. 509-533.

[103] Wilhelm M. R. and Ward T. L., "Solving quadratic problems by simulates annealing", IEEE Transactions 19, 1987, 107-119.

Παράρτημα Α – Ταξινόμηση Σωρού

Γράφος (Graph) – Δέντρο (tree) – Δυαδικό δέντρο (binary tree)

Ορισμός: Ένας **κατευθυνόμενος γράφος** G είναι το ζεύγος G = (V, E) όπου το V είναι ένα πεπερασμένο σύνολο, το σύνολο των κορυφών (ή κόμβων) και το $E \subseteq V \times V$ μια διμελής σχέση στο σύνολο V, το σύνολο των ακμών.

Ορισμός: **Βαθμός εξόδου** μιας κορυφής είναι το πλήθος των ακμών που εξέρχονται από αυτήν, ενώ **βαθμός εισόδου** μιας κορυφής είναι το πλήθος των ακμών που εισέρχονται σε αυτήν.

Ορισμός: **Διαδρομή** σε ένα γράφο είναι μια πεπερασμένη ακολουθία κορυφών $\langle u_0, u_1, u_2, \ldots, u_k \rangle$ τέτοια ώστε $(u_{i-1}, u_i) \in E$ για i = 1, 2, …, k. Το **μήκος** του μονοπατιού είναι k. Η u_0 ονομάζεται **αρχική κορυφή** ενώ η u_k **τελική κορυφή** της διαδρομής.

Ορισμός: Ένας γράφος είναι **συνδεδεμένος** αν υπάρχει διαδρομή από οποιαδήποτε κορυφή του σε οποιαδήποτε άλλη.

Ορισμός: Ένα **μονοπάτι** είναι μια διαδρομή στην οποία κάθε κορυφή της εμφανίζεται μία μόνο φορά.

Ορισμός: **Κύκλος** είναι ένα κλειστό μονοπάτι, δηλαδή ένα μονοπάτι στο οποίο η αρχική του κορυφή είναι ίδια με την τελική του.

Ορισμός: **Δένδρο** είναι ένας συνδεδεμένος, άκυκλος (χωρίς κύκλους), κατευθυνόμενος γράφος.

Ορισμός: **Δυαδικό δένδρο** είναι ένα δένδρο στο οποίο κάθε κόμβος (κορυφή) του έχει το πολύ δύο παιδιά, δηλαδή έχει βαθμό εξόδου το πολύ δύο.

Ορισμός: **Ρίζα** ονομάζεται ο μοναδικός κόμβος του δένδρου με βαθμό εισόδου μηδέν.

Ορισμός: **Φύλλο** σε ένα δένδρο είναι ένας κόμβος χωρίς παιδιά, δηλαδή με βαθμό εξόδου μηδέν.

Ορισμός: Το **βάθος** d(v) ενός κόμβου v σε ένα δένδρο είναι το πλήθος των ακμών του μοναδικού μονοπατιού από τη ρίζα στον κόμβο αυτό, δηλαδή είναι το μήκος του μονοπατιού με αρχικό κόμβο τη ρίζα του δέντρου και τελικό κόμβο τον κόμβο v.

Ορισμός: Το **βάθος** d(T) ενός δένδρου T ισούται με το μεγαλύτερο από τα βάθη των κόμβων του.

Ορισμός: **Ύψος** ενός κόμβου σε ένα δέντρο είναι το πλήθος των ακμών στο μακρύτερο μονοπάτι από τον κόμβο αυτό μέχρι κάποιο φύλλο.

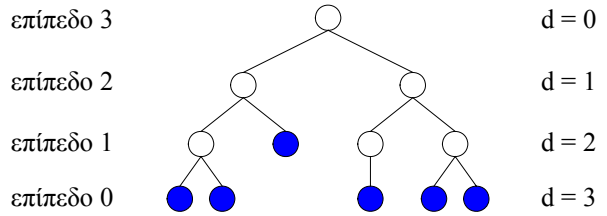
Ορισμός: **Ύψος** ενός δένδρου είναι το ύψος της ρίζας του.

Ορισμός: Το **επίπεδο** ενός κόμβου ισούται με το ύψος του δένδρου μείον το βάθος του κόμβου.

Ορισμός: Ένα δένδρο T με n κόμβους είναι ένα **σχεδόν πλήρες δυαδικό δένδρο** αν

- όλοι οι κόμβοι του με βάθος μικρότερο από d(T) – 1 έχουν βαθμό εξόδου 2.
- το πολύ ένας κόμβος του έχει βαθμό εξόδου 1.

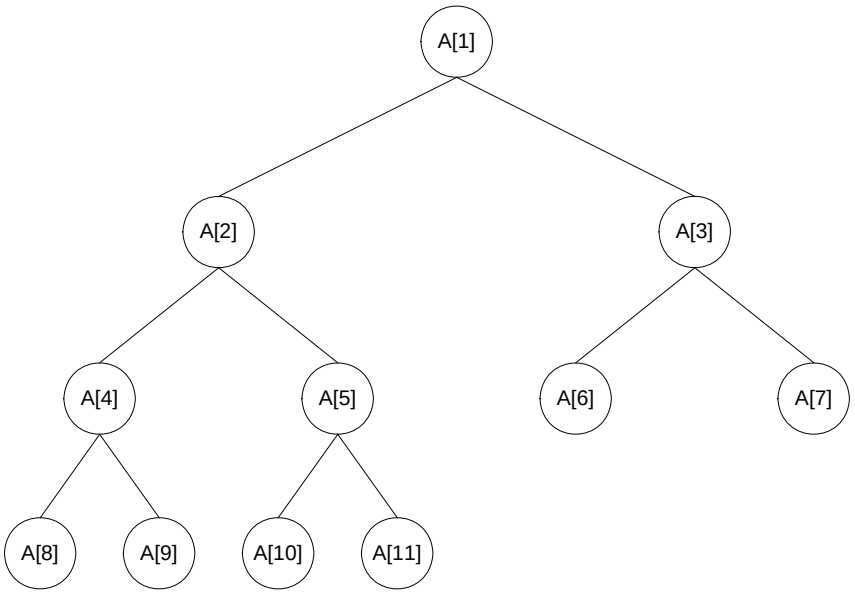
Σε ένα σχεδόν πλήρες δυαδικό δένδρο T με n κόμβους ισχύει η σχέση $d(T) = \lfloor \log_2(n) \rfloor$. Ένα σχεδόν πλήρες δυαδικό δένδρο T φαίνεται στο παρακάτω σχήμα:



όπου n = 12, d(T) = 3.
Παρατηρούμε ότι πραγματικά ισχύει $d(T) = \lfloor \log_2(n) \rfloor = \lfloor \log_2(12) \rfloor = \lfloor 3.585 \rfloor = 3$.

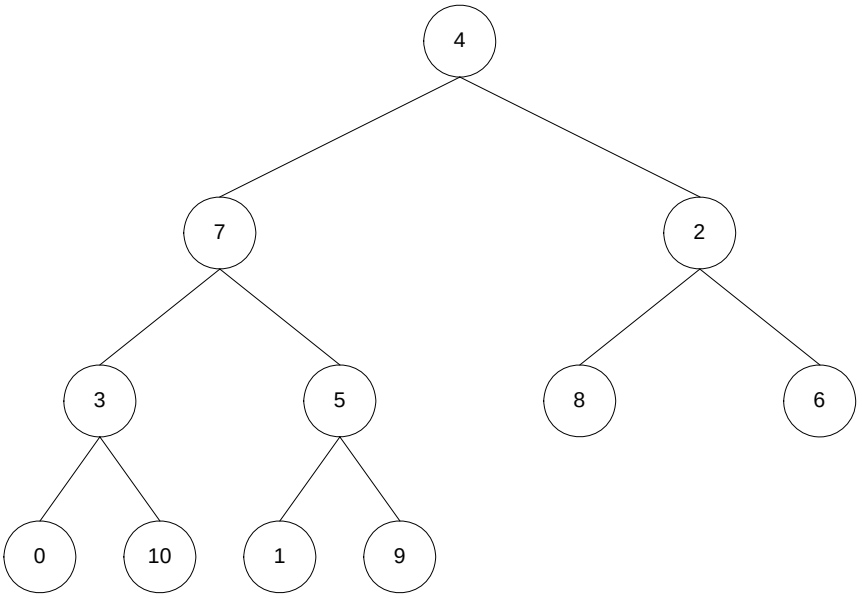
Ένα δυαδικό δένδρο είναι σχεδόν πλήρες (essentially complete) αν κάθε εσωτερικός κόμβος του έχει ακριβώς δύο παιδιά, ένα στα αριστερά του και ένα στα δεξιά του, με πιθανή εξαίρεση ενός μόνο κόμβου, ο οποίος βρίσκεται στο επίπεδο 1, και που έχει ένα μόνο παιδί. Επίσης όλα τα φύλλα του δένδρου βρίσκονται είτε στο επίπεδο 0, είτε στα επίπεδα 0 και 1.

Τα δένδρα αυτού του είδους μπορούν να αναπαρασταθούν στον υπολογιστή από μία δομή πίνακα. Στον πίνακα αυτό, έστω A, οι κόμβοι του δένδρου T βάθους k τοποθετούνται από αριστερά προς τα δεξιά στις θέσεις $A[2^k]$, $A[2^k+1]$, … , $A[2^{k+1}-1]$ (με πιθανή εξαίρεση του επιπέδου 0 που μπορεί να μην είναι πλήρες). Παρακάτω φαίνεται η αναπαράσταση ενός σχεδόν πλήρους δυαδικού δένδρου που περιέχει 11 κόμβους καθώς και ο αντίστοιχος πίνακας αποθήκευσής του στη μνήμη του υπολογιστή.



A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]

Ο γονέας του κόμβου που αναπαριστάται στη θέση A[i] βρίσκεται στη θέση A[i div 2] για i > 1. Τα παιδιά του κόμβου που αναπαριστάται στη θέση A[i] βρίσκονται στις θέσεις A[2i] και A[2i+1], οποτεδήποτε υπάρχουν. Παρακάτω φαίνεται ένα αριθμητικό παράδειγμα αποθήκευσης ενός σχεδόν πλήρους δυαδικού δένδρου στον υπολογιστή.



1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
4	7	2	3	5	8	6	0	10	1	9

Παρατηρούμε πως υπάρχει μια 1 – 1 αντιστοιχία μεταξύ των κορυφών του δένδρου και των στοιχείων του πίνακα.

Η ρίζα του δένδρου είναι το πρώτο στοιχείο του πίνακα.

Ο κόμβος με τιμή 7, που βρίσκεται στη θέση 2 του πίνακα, έχει γονέα τον κόμβο στη θέση 2 div 2 = 1, που είναι η ρίζα του δένδρου με τιμή 4.

Ο κόμβος με τιμή 3, που βρίσκεται στη θέση 4 του πίνακα, έχει παιδιά τους κόμβους που βρίσκονται στις θέσεις 2*4 = 8 και 2*4+1 = 9, δηλαδή τους κόμβους με τιμές 0 και 10 αντίστοιχα.

Όλα αυτά είναι φανερά και από τη σχηματική αναπαράσταση του δέντρου (συνηθίζεται να σχεδιάζεται σε επίπεδα όπως παραπάνω) αλλά και από τον πίνακα αποθήκευσής του στη μνήμη του υπολογιστή, όπου μια σχηματική του αναπαράσταση δεν είναι αναγκαία. Με τη δομή αυτή του πίνακα δεν είναι απαραίτητη η χρήση δεικτών για τη δημιουργία του δέντρου σε οποιαδήποτε γλώσσα προγραμματισμού.

Σωρός (heap)

Έστω T = (V, E) ένα σχεδόν πλήρες δυαδικό δέντρο και έστω μια απεικόνιση a : V → M η οποία αναθέτει σε κάθε κορυφή u του δέντρου μια τιμή (ετικέτα) a(u) από ένα διατεταγμένο σύνολο (M,≤).

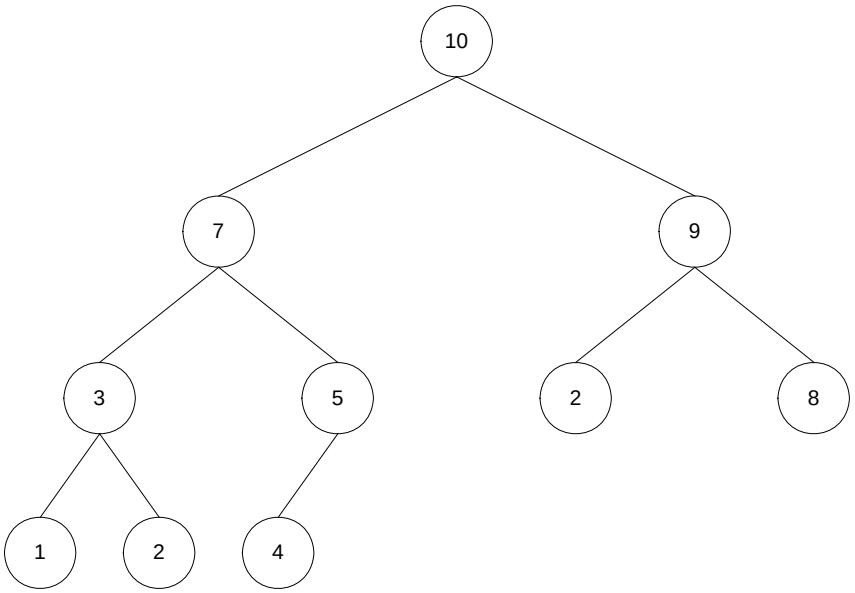
Ορισμός: Ένας κόμβος του δέντρου, έστω u, έχει την **ιδιότητα του σωρού** αν δεν έχει παιδιά με τιμή μεγαλύτερη από τη δική του.

$$\forall v \in V : (u,v) \in E \Rightarrow a(u) \geq a(v)$$

Ορισμός: Ένα σχεδόν πλήρες δυαδικό δέντρο T είναι **σωρός (μεγίστων)** αν όλοι οι κόμβοι του έχουν την ιδιότητα του σωρού.

$$\forall (u,v) \in E : a(u) \geq a(v)$$

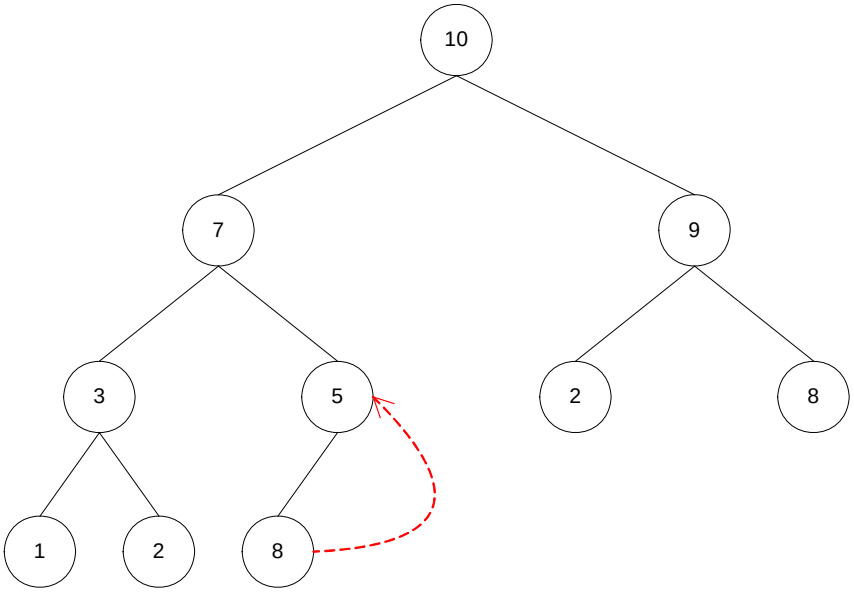
Ένας σωρός μεγίστων είναι ένα σχεδόν πλήρες δυαδικό δένδρο στο οποίο κάθε κόμβος έχει τιμή μεγαλύτερη ή ίση από τις τιμές των παιδιών του. Ένα παράδειγμα σωρού καθώς και η αντίστοιχη αναπαράσταση αυτού στον υπολογιστή φαίνεται στο παρακάτω σχήμα.



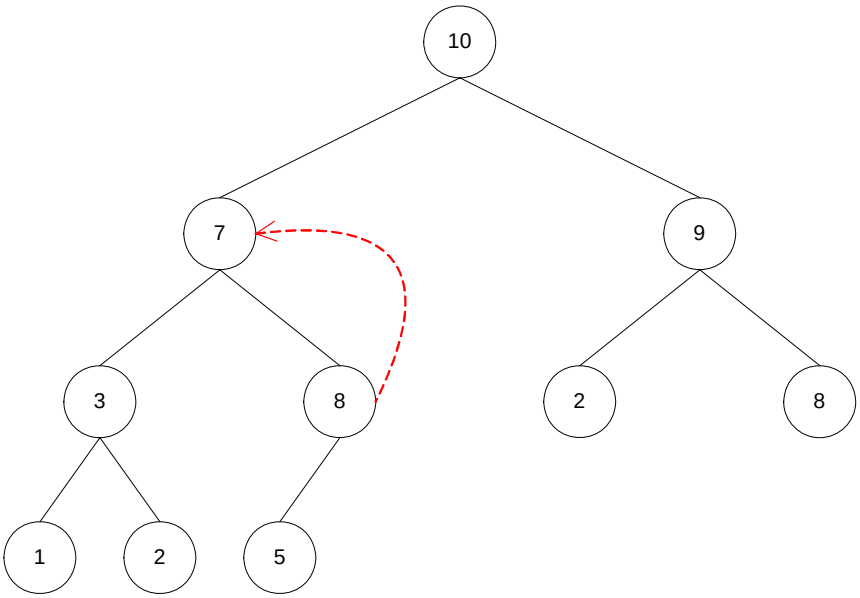
1	2	3	4	5	6	7	8	9	10
10	7	9	3	5	2	8	1	2	4

Η ιδιότητά του σωρού μπορεί να επιτευχθεί ξανά αποδοτικά μετά την αλλαγή της τιμής ενός κόμβου του. Αυτό είναι το θεμελιώδες και πολύ επιθυμητό για τους προγραμματιστές χαρακτηριστικό της δομής του σωρού. Αν αυξηθεί η τιμή ενός κόμβου, έτσι ώστε να γίνει μεγαλύτερη από την τιμή του γονέα του (η ιδιότητα του σωρού παύει προσωρινά να ισχύει), τότε επαρκεί η ανταλλαγή των τιμών αυτών των δύο κόμβων. Η διαδικασία αυτή συνεχίζεται προς τα πάνω στο δένδρο (στην πορεία του μονοπατιού από τη ρίζα μέχρι τον κόμβο, η τιμή του οποίου άλλαξε) μέχρι να αποκατασταθεί η ιδιότητα του σωρού.

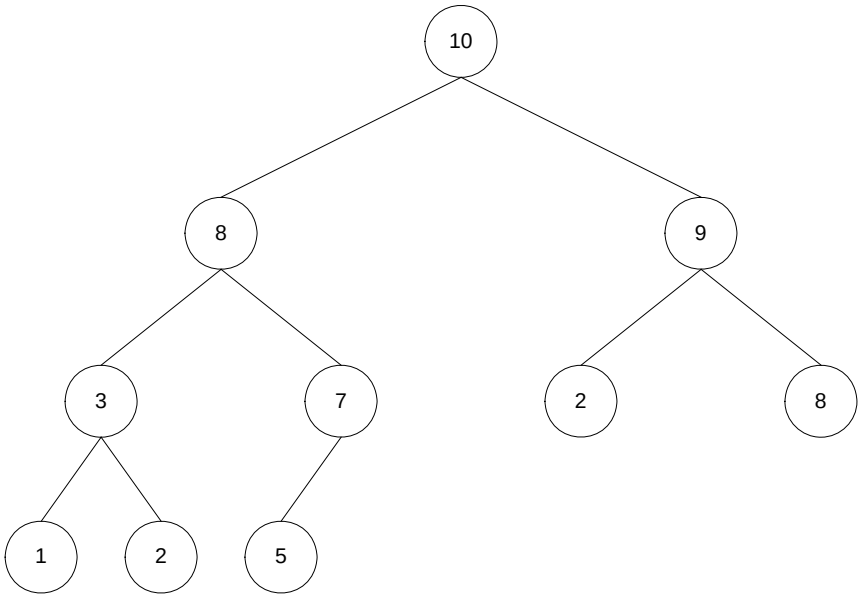
Αν για παράδειγμα στον παραπάνω σωρό η τιμή του τελευταίου κόμβου αλλάξει και από 4 γίνει 8, θα έχουμε τις εξής αλλαγές:



Η τιμή του κόμβου έγινε μεγαλύτερη από αυτή του γονέα του και έτσι αλλάζουν θέσεις.



Μετά την αλλαγή των θέσεων ο κόμβος με την τιμή 8 έχει και πάλι μεγαλύτερη τιμή από αυτή του (νέου) γονέα του. Άρα και πάλι αλλάζουν θέσεις.



Ο κόμβος με την τιμή 8 έχει φτάσει στην τελική του θέση, αφού τώρα ο γονέας του, που τυχαίνει να είναι και η ρίζα του δένδρου, έχει τιμή μεγαλύτερη από τη δική του. Η ιδιότητα του σωρού αποκαταστάθηκε.

Στην περίπτωση αυτή συνηθίζεται να λέγεται ότι η τροποποιημένη τιμή έχει διηθηθεί στη νέα της θέση. Η διαδικασία συχνά αναφέρεται ως μετατόπιση προς τα πάνω (shift – up) και περιγράφεται λεπτομερώς παρακάτω σε μορφή ψευδοκώδικα.

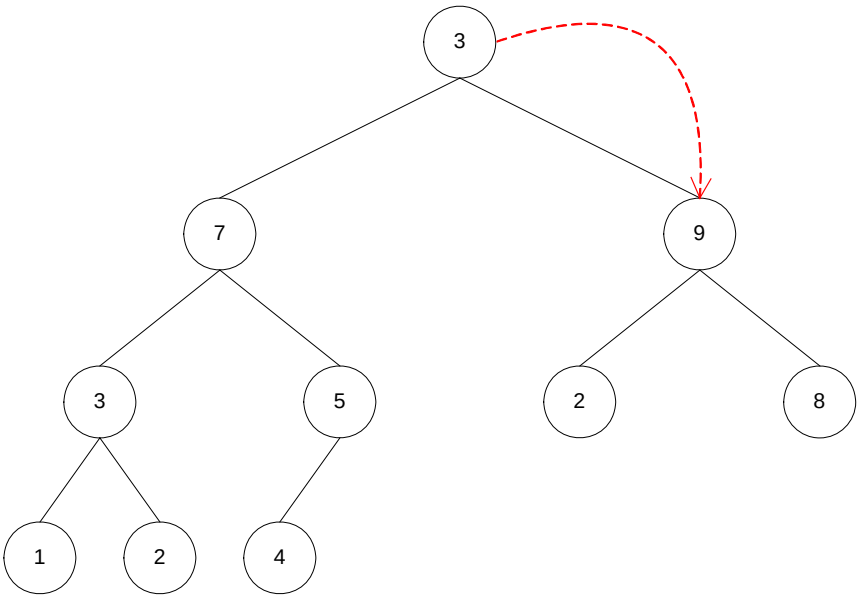
```
διαδικασία shift-up(A[1...n], I)
{
    k = i
    do
    {
        j = k
        if (j > 1 and A[j div 2] < A[k])
            k = j div 2
        exchange(T[j], T[k])
    }
    while j = k
}
```

όπου η ανταλλαγή των τιμών δύο κόμβων γίνεται με την παρακάτω διαδικασία

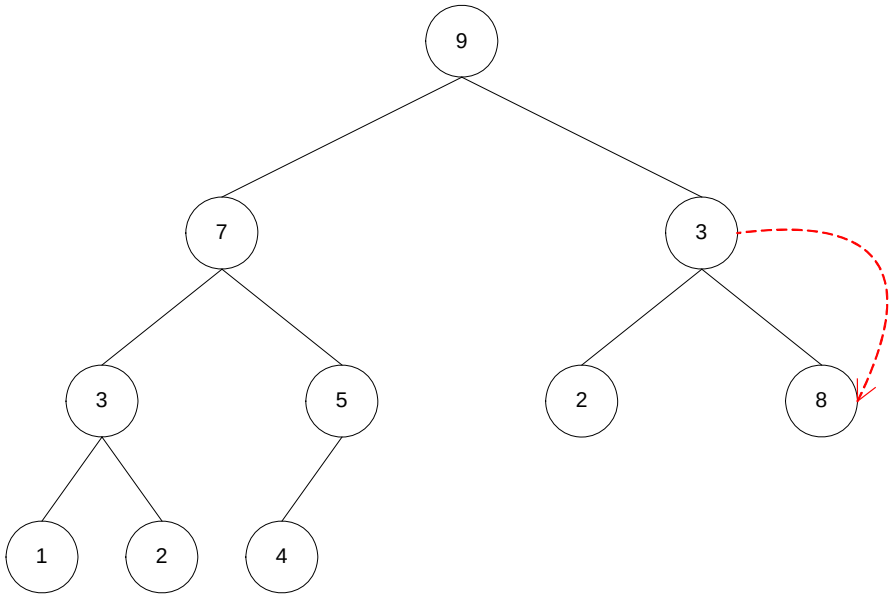
```
διαδικασία exchange(x, y)
{
    temp = x
    x = y
    y = temp
}
```

Αν η τιμή ενός κόμβου ελαττωθεί έτσι ώστε να γίνει μικρότερη από τη τιμή τουλάχιστον ενός από τα παιδιά του, επαρκεί η ανταλλαγή της τροποποιημένης τιμής με τη μεγαλύτερη από τις τιμές των παιδιών. Συνεχίζουμε αυτή τη διαδικασία προς τα κάτω στο δένδρο μέχρι να αποκατασταθεί η ιδιότητα του σωρού.

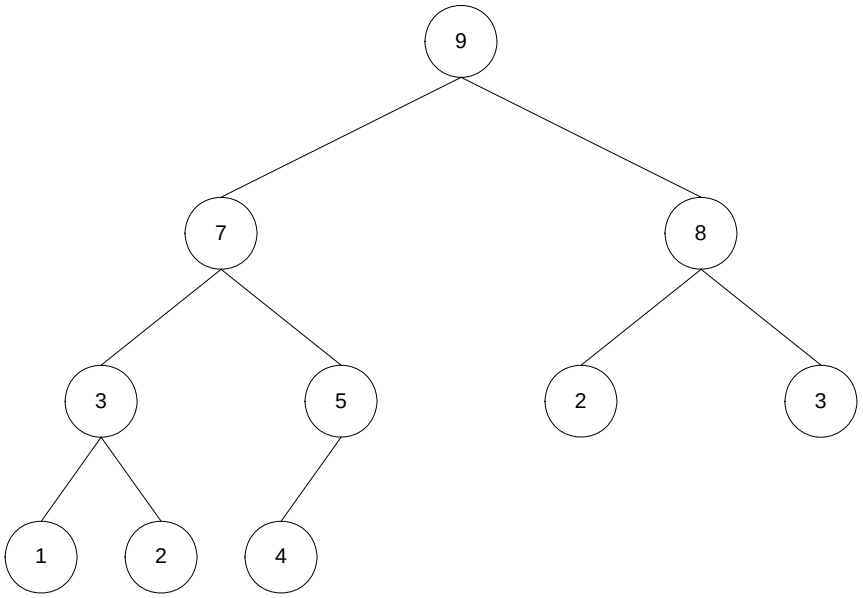
Αν για παράδειγμα στο αρχικό δένδρο η τιμή της ρίζας αλλάξει και γίνει 3 τότε θα έχουμε τις εξής αλλαγές:



Η τιμή της ρίζας τροποποιήθηκε και έγινε μικρότερη από τις τιμές και των δύο παιδιών της. Για το λόγο αυτό αλλάζει θέση με τη μεγαλύτερη από τις τιμές των παιδιών της.



Η διαδικασία συνεχίζεται γιατί ο κόμβος με την τιμή 3 έχει τουλάχιστον ένα παιδί με μεγαλύτερη τιμή. Έτσι αλλάζουν τιμές με τον κόμβο αυτό.



Η διαδικασία τελειώνει αφού ο κόμβος μας έχει γίνει φύλλο και έτσι δεν υπάρχει περίπτωση να κατεβεί πιο κάτω. Η ιδιότητα του σωρού έχει ανακτηθεί. Υπάρχει και η περίπτωση η διαδικασία να τελειώσει πριν ο κόμβος γίνει φύλλο, αν μετά από μια ανταλλαγή η τιμή του είναι μεγαλύτερη από τις τιμές και των δύο παιδιών του.

Στην περίπτωση αυτή λέμε ότι η τροποποιημένη τιμή έχει μετατοπισθεί στη νέα της θέση. Η διαδικασία συχνά αναφέρεται ως μετατόπιση προς τα κάτω (shift – down) και περιγράφεται λεπτομερώς παρακάτω σε μορφή ψευδοκώδικα.

```
διαδικασία shift-down(A[1…n], i)
{
    k = i
    do
    {
        j = k
        if (2j <= n and A[2j] > A[k])
            k = 2j
        if (2j < n and A[2j+1] > A[k])
            k = 2j+1
        exchange(T[j], T[k])
    }
    while j = k;
}
```

Ο σωρός είναι η ιδανική δομή δεδομένων για την εύρεση του μεγαλύτερου στοιχείου ενός συνόλου, αφού αυτό βρίσκεται πάντα στη ρίζα του σωρού. Επίσης η δομή του επιτρέπει αποδοτικούς αλγορίθμους για την αφαίρεση ενός στοιχείου από το σύνολο, για την εισαγωγή ενός νέου στοιχείου ή για την τροποποίηση της τιμής ενός στοιχείου χωρίς να αλλάζει η ιδιότητα του σωρού. Αυτές είναι οι λειτουργίες που χρειάζονται για να υλοποιηθούν αποδοτικά δυναμικές λίστες προτεραιότητας. Η τιμή κάθε κόμβου αντιστοιχεί στην προτεραιότητα ενός γεγονότος. Το γεγονός με την υψηλότερη προτεραιότητα πάντοτε βρίσκεται στη ρίζα του σωρού. Ο δομή του σωρού δίνει τη δυνατότητα αποδοτικής αντιμετώπισης της δυναμικής αλλαγής της προτεραιότητας κάθε γεγονότος όποτε αυτό είναι απαραίτητο.

Ταξινόμηση σωρού (heap sort)

Μία ακόμη διαδικασία η οποία μπορεί να υλοποιηθεί αποδοτικά με τη χρήση του σωρού είναι αυτή της ταξινόμησης των στοιχείων ενός πίνακα. Η διαδικασία αυτή είναι γνωστή ως **ταξινόμηση σωρού** και αποτελεί έναν από τους πιο αποδοτικούς αλγορίθμους ταξινόμησης. Για να γίνει κατανοητή η λειτουργία του αλγορίθμου, το πρώτο βήμα είναι η δημιουργία ενός σωρού, ξεκινώντας από έναν πίνακα P[1…n] με στοιχεία σε τυχαία σειρά (δηλαδή στοιχεία που δεν είναι ταξινομημένα).

Έστω τα στοιχεία του πίνακα P είναι

5	9	2	3	7	1	6	8	0	4

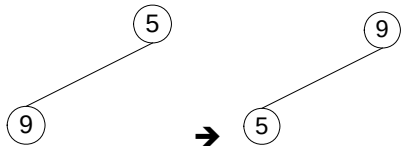
Η διαδικασία εισαγωγής ενός νέου κόμβου σε ένα σωρό ξεκινάει με την τοποθέτηση του νέου αυτού κόμβου στο τέλος του πίνακα που αναπαριστά το σωρό. Με την επιλογή αυτή ο νέος κόμβος καταλαμβάνει την πιο δεξιά θέση του επιπέδου 0 του δένδρου, εκτός κι αν το επίπεδο αυτό είναι γεμάτο, οπότε το ύψος του δένδρου αυξάνει κατά 1, και ο νέος κόμβος γίνεται ο πρώτος κόμβος του νέου επιπέδου. Στη συνέχεια γίνεται διήθηση του νέου αυτού κόμβου ώστε να μπει στη σωστή του θέση και να αποκατασταθεί τη ιδιότητα του σωρού. Η διαδικασία περιγράφεται λεπτομερώς με τον παρακάτω ψευδοκώδικα.

```
διαδικασία insert-node(A[1…n], v)
{
    A[n+1] = v
    shift-up(A[1…n+1], n+1)
}
```

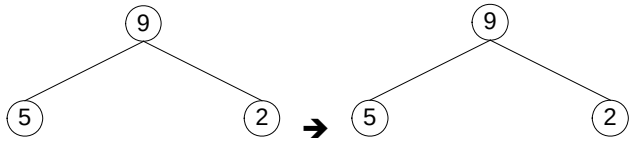
Κατά την εισαγωγή στη δομή του σωρού των στοιχείων του πίνακα P με τη σειρά που αυτά είναι αποθηκευμένα δημιουργούνται τα εξής βήματα κατά την ανάπτυξη της δομής δεδομένων:

εισαγωγή του 5

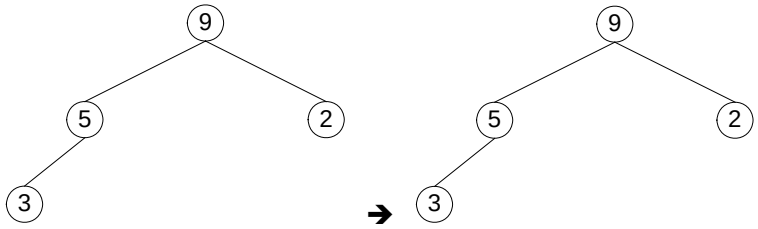
εισαγωγή του 9



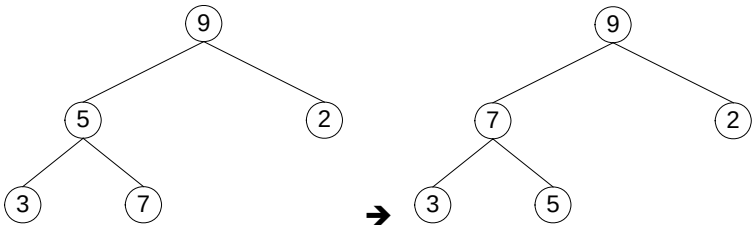
εισαγωγή του 2



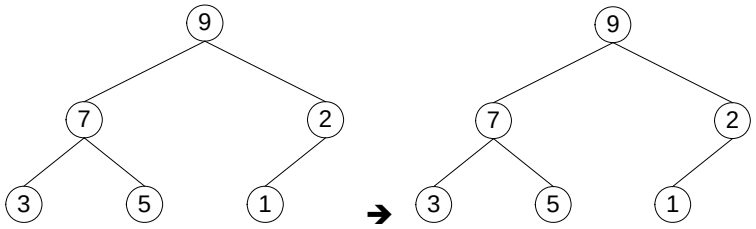
εισαγωγή του 3



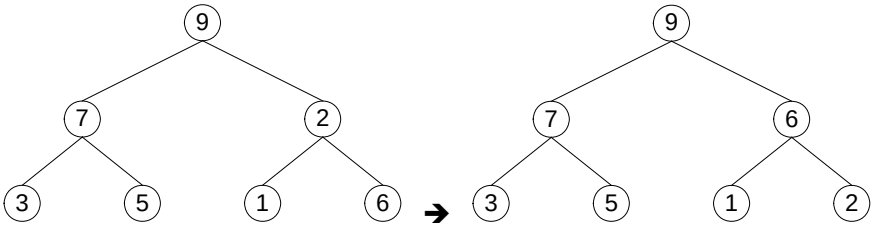
εισαγωγή του 7



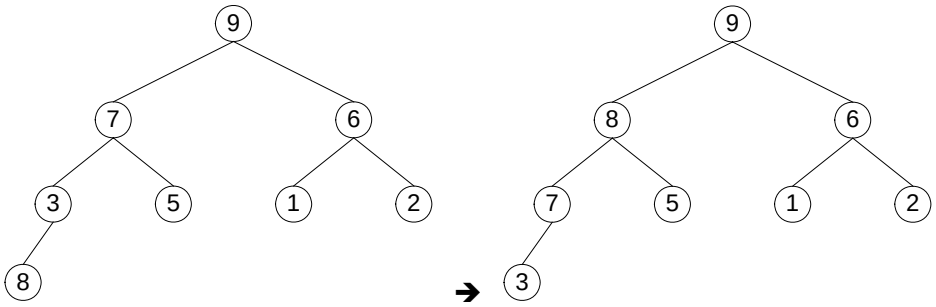
εισαγωγή του 1



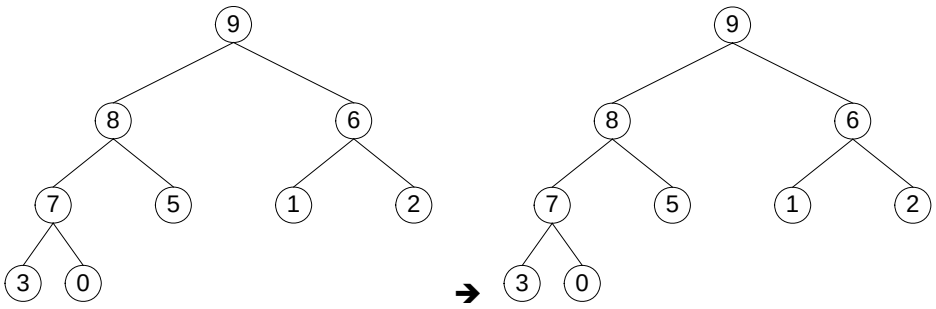
εισαγωγή του 6



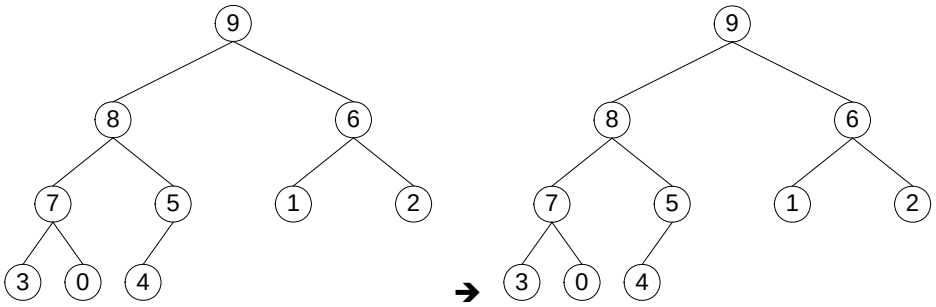
εισαγωγή του 8



εισαγωγή του 0



εισαγωγή του 4

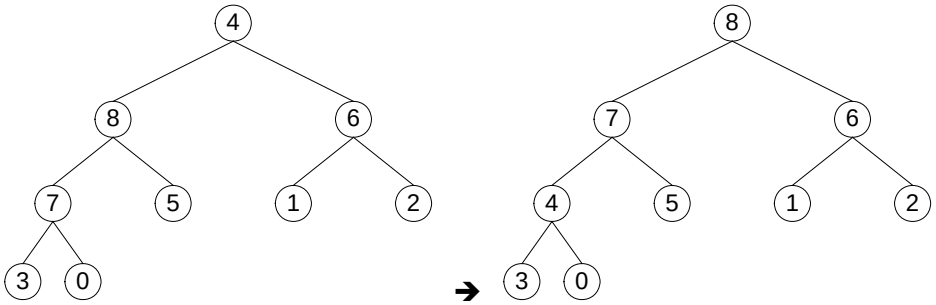


Η διαδικασία εξαγωγής του μεγαλύτερου κόμβου από ένα σωρό (που είναι πάντα η ρίζα του) ξεκινάει με την τοποθέτηση του τελευταίου κόμβου του δένδρου στη θέση της ρίζας του. Στη συνέχεια γίνεται μετατόπιση προς τα κάτω της νέας ρίζας του δένδρου ώστε να μπει στη σωστή της θέση και να αποκατασταθεί η ιδιότητα του σωρού. Η διαδικασία περιγράφεται λεπτομερώς με τον παρακάτω ψευδοκώδικα.

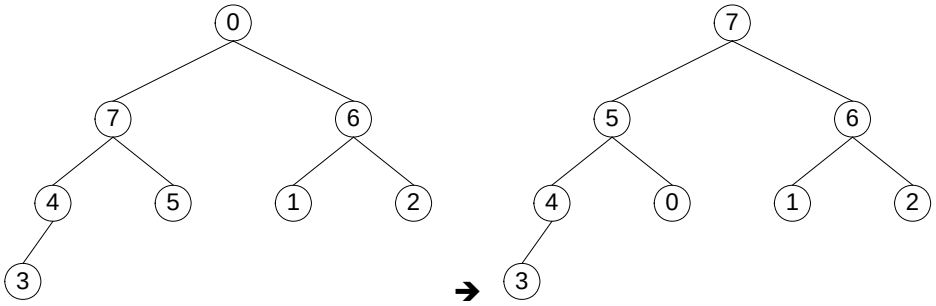
```
διαδικασία remove-max(A[1…n])
{
    max = A[1]
    A[1] = A[n]
    shift-down(A[1…n-1], 1)
    return max
}
```

Κατά την εξαγωγή των στοιχείων του σωρού (κάθε φορά το μεγαλύτερο) δημιουργούνται τα εξής βήματα:

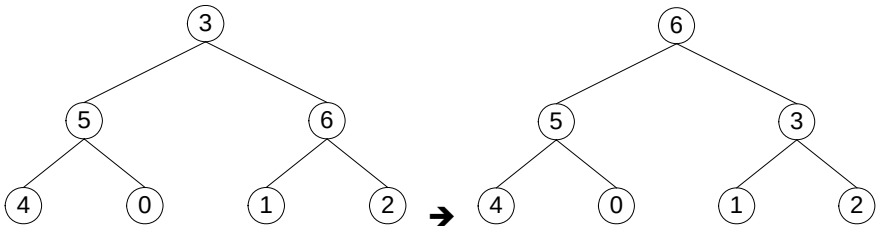
εξαγωγή του 9



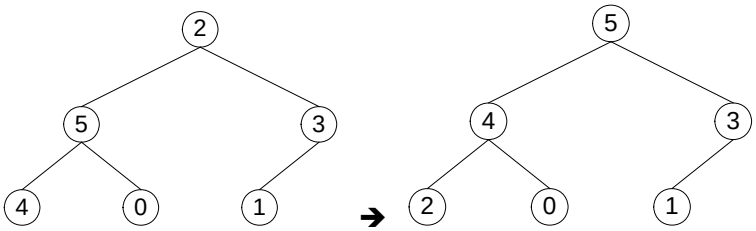
εξαγωγή του 8



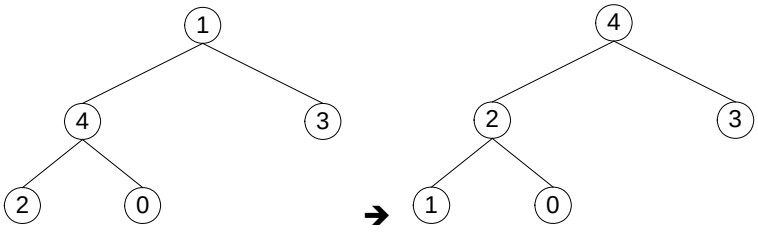
εξαγωγή του 7



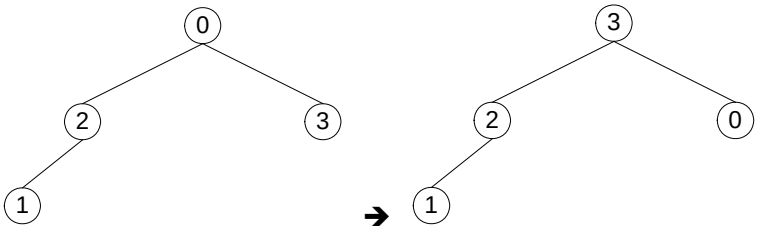
εξαγωγή του 6



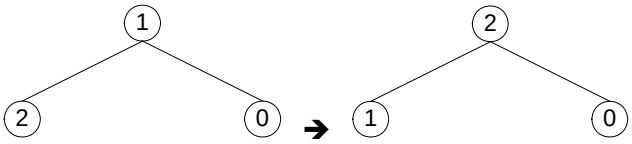
εξαγωγή του 5



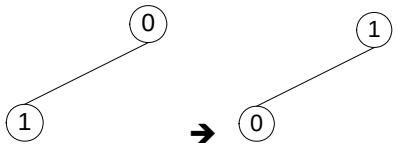
εξαγωγή του 4



εξαγωγή του 3



εξαγωγή του 2



εξαγωγή του 1

εξαγωγή του 0

Παρατηρούμε πως κατά την εξαγωγή των στοιχείων από το σωρό αυτά βγήκαν με σειρά από το μεγαλύτερο προς το μικρότερο, παρόλο που κατά την εισαγωγή τους δεν εισήλθαν με αυτή τη σειρά. Το πλεονέκτημα αυτό που βασίζεται στην ιδιότητα του σωρού, εκμεταλλεύεται ο αλγόριθμος στη λειτουργία της ταξινόμησης. Στο παραπάνω παράδειγμα υλοποιήθηκε η δυνατότητα ταξινόμησης των στοιχείων ενός αρχικά αταξινόμητου πίνακα με τη βοήθεια της δομής του σωρού. Η διαδικασία αυτή μπορεί να εφαρμοστεί σε οποιοδήποτε πίνακα, οποιουδήποτε μεγέθους. Ο παρακάτω ψευδοκώδικας περιγράφει λεπτομερώς τη διαδικασία της ταξινόμησης σωρού.

```
διαδικασία heap-sort(P[1…n])
{
    for i = 1…n
        insert-node(A[i-1], P[n])
    end for
    for i = 1…n
        P[i] = remove-max(A[n-i+1])
    end for
}
```

Απόδοση του αλγορίθμου

Το βασικό στοιχείο κάθε αλγορίθμου εκτός από τη σωστή λειτουργία του είναι η απόδοσή του. Το πόσο γρήγορος είναι. Το χαρακτηριστικό αυτό γίνεται πολύ σημαντικό κυρίως σε αλγορίθμους που έχουν να χειριστούν μεγάλο πλήθος στοιχείων, όπως οι αλγόριθμοι ταξινόμησης. Ένας αλγόριθμος με κακή απόδοση μπορεί να χρησιμοποιηθεί μόνο σε μικρό πλήθος στοιχείων, ενώ η χρήση του σε προβλήματα με μεγάλο μέγεθος μπορεί να είναι απαγορευτική λόγω του χρόνου που απαιτεί για την ολοκλήρωσή του.

Από τον ορισμό του σωρού αλλά και την περιγραφή του αλγορίθμου ταξινόμησης και των διαδικασιών του, συνεπάγεται ότι όλα τα επίπεδα του δέντρου, που αντιστοιχεί στο σωρό, είναι τελείως συμπληρωμένα, εκτός ίσως από το χαμηλότερο επίπεδο, το οποίο είναι γεμάτο από τα αριστερά μέχρι ένα σημείο. Είναι φανερό ότι ένας σωρός με ύψος h έχει τον ελάχιστο αριθμό στοιχείων όταν έχει μόνο έναν κόμβο στο χαμηλότερο επίπεδο. Τα επίπεδα πάνω από το χαμηλότερο επίπεδο (αν αφαιρέσουμε το τελευταίο), αποτελούν ένα πλήρες δυαδικό δέντρο με ύψος h – 1 και πλήθος κόμβων $2^h - 1$. Άρα ο ελάχιστος δυνατός αριθμός κόμβων σε ένα σωρό ύψους h είναι 2^h. Είναι επίσης προφανές ότι ένας σωρός με ύψος h, έχει μέγιστο αριθμό στοιχείων όταν το χαμηλότερο επίπεδό του είναι πλήρως συμπληρωμένο. Στην περίπτωση αυτή ο σωρός είναι ένα πλήρες δυαδικό δέντρο με ύψος h και $2^{h+1} - 1$ κόμβους.

Από τα όρια που μόλις υπολογίστηκαν για το μικρότερο και το μεγαλύτερο πλήθος στοιχείων ενός σωρού ύψους h, έχουμε τη σχέση

$$2^h \le n \le 2^{h+1} - 1$$

όπου n είναι το πλήθος των στοιχείων του σωρού.

Είναι φανερό ότι ισχύει και η ανισότητα

$$2^h \le n \le 2^{h+1}$$

από την οποία αν πάρουμε λογαρίθμους με βάση το 2 σε όλα της τα μέρη έχουμε

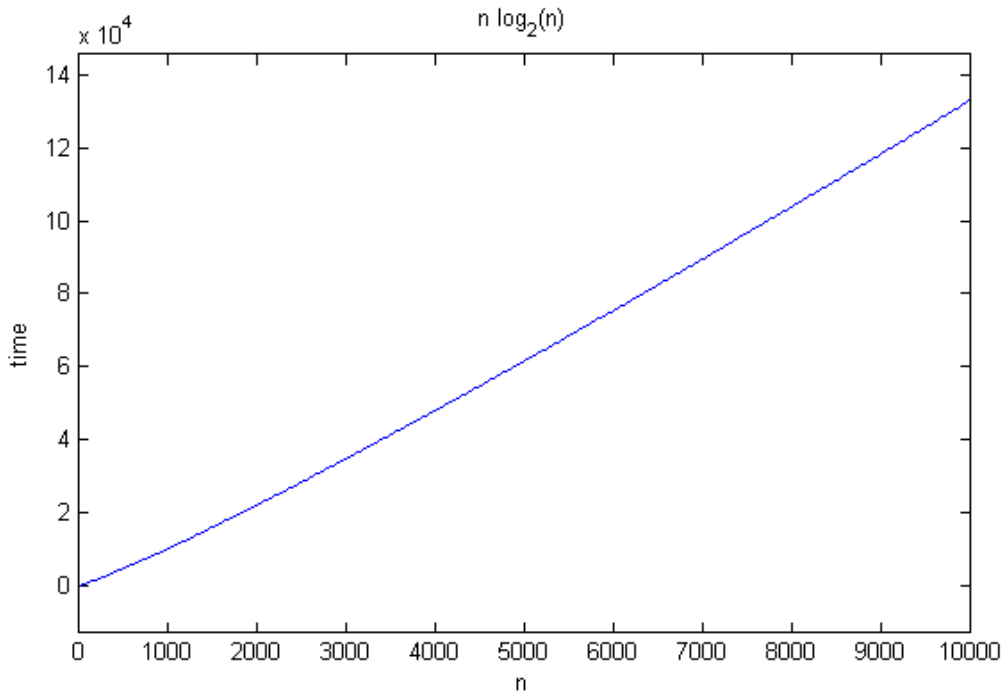
$$h \le \log_2(n) \le h+1$$

από το οποίο συνεπάγεται ότι

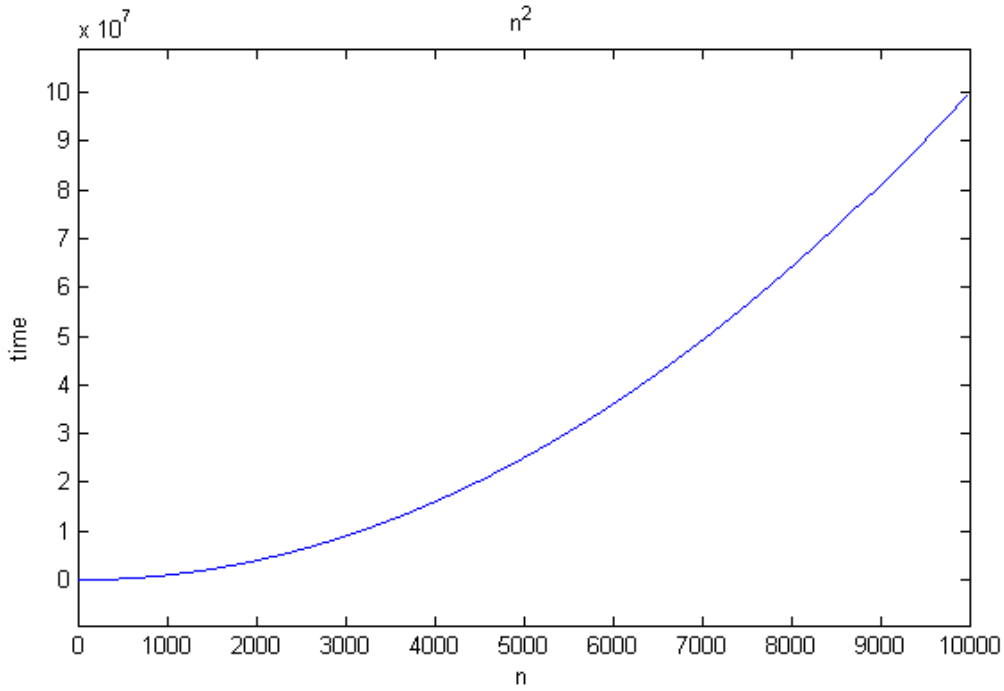
$$h = \lfloor \log_2(n) \rfloor$$

Ένα σχεδόν πλήρες δυαδικό δένδρο με n κόμβους έχει ύψος log(n). Άρα στη χειρότερη περίπτωση οι διαδικασίες shift-up και shift-down κάνουν log(n) εναλλαγές (exchange) στοιχείων. Άρα και οι διαδικασίες insert-node και remove-max κάνουν χρόνο στην τάξη του O(log(n)). Η διαδικασία heap-sort καλεί κάθε μια από αυτές τις διαδικασίες n φορές, μία για κάθε στοιχείο του πίνακα. Άρα η συνολική απόδοση του αλγορίθμου ταξινόμησης σωρού είναι O(2n log(n)) = O(n log(n)).

Η απόδοση αυτή είναι η καλύτερη που μπορεί να επιτευχθεί για έναν αλγόριθμο ταξινόμησης. Ας θυμηθούμε πως η γρήγορη ταξινόμηση (quick sort) έχει απόδοση O(n log(n)), η ταξινόμηση συγχώνευσης (merge sort) επίσης O(n log(n)), η ταξινόμηση εισαγωγής (insert sort) $O(n^2)$, η ταξινόμηση επιλογής (select sort) $O(n^2)$ και τέλος η ταξινόμηση φυσαλίδας (bubble sort) επίσης $O(n^2)$. Παρακάτω φαίνεται η γραφική παράσταση του χρόνου που απαιτείται για την ταξινόμηση σωρού ανάλογα με το μέγεθος του πίνακα προς ταξινόμηση.



Αν συγκριθεί με τον υπολογιστικό χρόνο που απαιτούν οι αλγόριθμοι ταξινόμησης με απόδοση $O(n^2)$ είναι ολοφάνερο πως είναι πολύ πιο κατάλληλος για μεγάλο πλήθος στοιχείων.



Ο αλγόριθμος ταξινόμησης σωρού είναι βέλτιστος αφού επιτυγχάνει το μικρότερο όριο υπολογιστικού χρόνου για το πρόβλημα της ταξινόμησης, τουλάχιστον με όσα γνωρίζουμε μέχρι τώρα στην επιστήμη της πληροφορικής και του προγραμματισμού.

Επίλογος

Η δομή δεδομένων του σωρού είναι η καταλληλότερη για την ταξινόμηση μεγάλου πλήθους στοιχείων και ιδιαίτερα όταν σ' αυτά προσθέτονται ή αφαιρούνται δυναμικά νέα στοιχεία κατά τη διάρκεια της διαδικασίας ταξινόμησης. Η ιδιότητά του δίνει τη δυνατότητα, να εκτελούνται οι απαραίτητες ενέργειες για την ταξινόμηση πολύ αποδοτικά και σε μικρό υπολογιστικό χρόνο.

Η δομή που παρουσιάστηκε παραπάνω είναι η δομή του σωρού μεγίστων. Στη δενδρική αυτή δομή κάθε κόμβος έχει τιμή μεγαλύτερη ή ίση από τις τιμές των παιδιών του. Ανάλογα θα μπορούσε να οριστεί και ο **σωρός ελαχίστων** όπου κάθε κόμβος του δέντρου έχει τιμή μικρότερη ή ίση από τις τιμές των παιδιών του. Στην περίπτωση αυτή οι διαδικασίες διήθησης και μετατόπισης ενός κόμβου, προς τα πάνω και προς τα κάτω αντίστοιχα, χρειάζεται να αλλάξουν και να οριστούν με ανάλογο τρόπο.

Η λειτουργία της ταξινόμησης των στοιχείων ενός αρχικά αταξινόμητου πίνακα μπορεί να επιτευχθεί πολύ εύκολα και αποδοτικά με οποιονδήποτε από τους δύο τύπους σωρών. Διακρίνονται τέσσερις περιπτώσεις:

- Έχει υλοποιηθεί ένας σωρός μεγίστων και είναι επιθυμητή μια φθίνουσα ταξινόμηση των στοιχείων ενός πίνακα. Γίνεται εισαγωγή των στοιχείων του πίνακα ως έχουν στο σωρό και στη συνέχεια εξαγωγή αυτών στην επιθυμητή σειρά (το παράδειγμα που παρουσιάστηκε παραπάνω).
- Έχει υλοποιηθεί ένας σωρός μεγίστων και είναι επιθυμητή μια αύξουσα ταξινόμηση των στοιχείων ενός πίνακα. Γίνεται εισαγωγή των αντίθετων στοιχείων του πίνακα στο σωρό (δηλαδή για κάθε στοιχείο x του πίνακα εισάγεται στο σωρό το στοιχείο –x) και στη συνέχεια εξαγωγή αυτών στην επιθυμητή σειρά.
- Έχει υλοποιηθεί ένας σωρός ελαχίστων και είναι επιθυμητή μια φθίνουσα ταξινόμηση των στοιχείων ενός πίνακα. Γίνεται εισαγωγή των αντίθετων στοιχείων του πίνακα στο σωρό (δηλαδή για κάθε στοιχείο x του πίνακα εισάγεται στο σωρό το στοιχείο –x) και στη συνέχεια εξαγωγή αυτών στην επιθυμητή σειρά.
- Έχει υλοποιηθεί ένας σωρός ελαχίστων και είναι επιθυμητή μια αύξουσα ταξινόμηση των στοιχείων ενός πίνακα. Γίνεται εισαγωγή των στοιχείων του πίνακα ως έχουν στο σωρό και στη συνέχεια εξαγωγή αυτών στην επιθυμητή σειρά.

Παράρτημα Β – Συνδεδεμένες λίστες

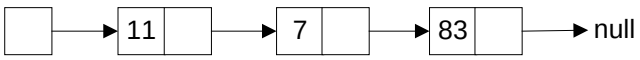
Ένα βασικό στοιχείο στη υλοποίηση κάθε αλγορίθμου, σε κάποια συγκεκριμένη γλώσσα προγραμματισμού, είναι η επιλογή της κατάλληλης δομής δεδομένων. Οι προγραμματιστές έχουν τη δυνατότητα επιλογής ανάμεσα σε διαφορετικές δομές δεδομένων, ανάλογα με τις ανάγκες του κάθε αλγορίθμου και τα δεδομένα που αυτός χρειάζεται κατά την υλοποίηση και εκτέλεσή του. Οι βασικότερες από αυτές τις δομές δεδομένων μπορούν να διαχωριστούν με κριτήριο το αν είναι σειριακές ή όχι. Σε μια σειριακή δομή δεδομένων τα στοιχεία που περιέχει, οργανώνονται και τοποθετούνται το ένα μετά το άλλο. Αυτό βέβαια όσον αφορά τη λογική σχέση μεταξύ τους και όχι την αποθήκευσή τους στον υπολογιστή. Στην πρώτη κατηγορία, των σειριακών δομών δεδομένων, ανήκουν οι **πίνακες**, οι **συνδεδεμένες λίστες**, οι **ουρές** και οι **στοίβες**, ενώ στη δεύτερη κατηγορία, των μη σειριακών δομών δεδομένων, ανήκουν τα **δέντρα**, οι **σωροί** και οι **γράφοι**.

Η μνήμη κάθε υπολογιστή, τουλάχιστον μέχρι σήμερα, είναι σειριακή. Για το λόγο αυτό τα στοιχεία των μη σειριακών δομών δεδομένων αποθηκεύονται σε αυτήν με διαφορετικό σειρά από αυτήν που πραγματικά υπαγορεύει η λογική τους σχέση. Συνήθως υλοποιούνται και αποθηκεύονται στη μνήμη του υπολογιστή με τη βοήθεια πιο απλών, σειριακών δομών δεδομένων, όπως οι πίνακες και οι συνδεδεμένες λίστες. Αυτός είναι ο ένας λόγος για τον οποίο οι δύο αυτές δομές δεδομένων είναι πολύ δημοφιλείς μεταξύ των προγραμματιστών και πολύ συχνά χρησιμοποιήσιμες. Ο άλλος είναι η απλότητά τους σε σχέση με τις υπόλοιπες, τόσο στην κατανόηση όσο και στην υλοποίησή τους.

Παραλλαγές

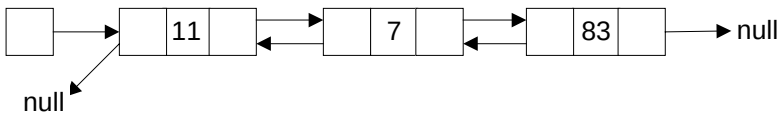
Η συνδεδεμένη λίστα είναι μια από τις θεμελιώδεις δομές δεδομένων στον προγραμματισμό και την υλοποίηση αλγορίθμων. Αποτελείται από μια ακολουθία κόμβων, κάθε ένας από τους οποίους περιέχει δεδομένα και μία ή δύο αναφορές (συνδέσμους, δείκτες) σε γειτονικούς κόμβους. Η μία από τις αναφορές αυτές δείχνει πάντοτε στον επόμενο κόμβο της λίστας. Αν οι κόμβοι περιέχουν και δεύτερη αναφορά τότε αυτή δείχνει στον προηγούμενο κόμβο της λίστας. Υπάρχουν διάφορες παραλλαγές της βασικής δομής της συνδεδεμένης λίστας κάθε μια από τις οποίες έχει τα πλεονεκτήματα και τα μειονεκτήματά της.

Η **απλά συνδεδεμένη λίστα** (**single linked list**) είναι το πιο απλό είδος συνδεδεμένης λίστας, αφού περιέχει μία μόνο αναφορά σε κάθε κόμβο. Η αναφορά αυτή δείχνει πάντα στον επόμενο κόμβο της λίστας ή σε τιμή null αν ο κόμβος είναι ο τελευταίος κόμβος της λίστας. Ένα σχηματικό παράδειγμα μιας απλά συνδεδεμένης λίστας φαίνεται παρακάτω.



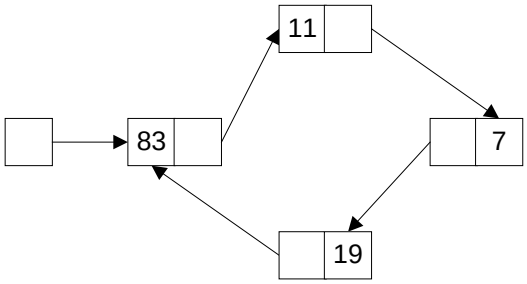
Πρόκειται για μια απλά συνδεδεμένη λίστα η οποία αποτελείται από τρεις κόμβους. Κάθε κόμβος περιλαμβάνει έναν ακέραιο και έναν δείκτη προς τον επόμενο κόμβο της λίστας, σύμφωνα με τη σειριακή της δομή. Ο τελευταίος κόμβος, αφού εξ ορισμού δεν έχει κάποιον επόμενο, δείχνει σε κενή τιμή, την τιμή null. Κάθε συνδεδεμένη λίστα αυτού του τύπου περιέχει πάντα ένα **δείκτη αρχής**, μια αναφορά που δείχνει στον πρώτο κόμβο της λίστας. Όταν η λίστα είναι κενή ο δείκτης αυτός είναι ίσος με την τιμή null. Τα δεδομένα κάθε κόμβου αυθαίρετα επιλέχθηκε να αποτελούνται από έναν ακέραιο. Στην πραγματικότητα κάθε κόμβος μπορεί να περιέχει οτιδήποτε δεδομένα επιβάλει η σχεδίαση, χωρίς αυτό να επηρεάζει τη δομή της λίστας και τις λειτουργίες που γίνονται σ' αυτήν.

Ένα άλλο είδος συνδεδεμένης λίστας που μοιάζει με το προηγούμενο είναι η **διπλά συνδεδεμένη λίστα** (**doubly linked list**) ή όπως ονομάζεται διαφορετικά **συνδεδεμένη λίστα διπλής κατεύθυνσης** (**two way linked list**). Κάθε κόμβος της έχει δύο αναφορές. Η μία δείχνει στον επόμενο κόμβο της λίστας ή έχει την τιμή null αν ο κόμβος που την περιέχει είναι ο τελευταίος της λίστας. Η άλλη δείχνει στον προηγούμενο κόμβο της λίστας ή έχει την τιμή null αν ο κόμβος που την περιέχει είναι ο πρώτος κόμβος της λίστας. Το παρακάτω σχήμα δείχνει μια διπλά συνδεδεμένη λίστα.



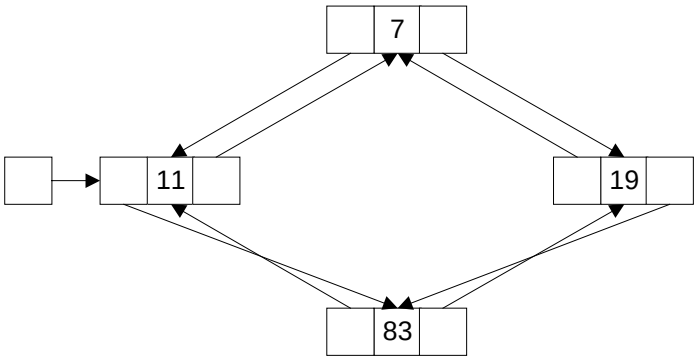
Για την απλότητα του παραδείγματος κάθε κόμβος περιέχει μόνο έναν ακέραιο ως δεδομένο του. Είναι εμφανές ότι κάθε κόμβος περιέχει δύο δείκτες. Έναν προς τον επόμενο και έναν προς τον προηγούμενο κόμβο της λίστας. Ο πρώτος κόμβος της λίστας δεν έχει προηγούμενο κόμβο και έτσι ο αριστερός του δείκτης έχει την τιμή null. Ομοίως ο τελευταίος κόμβος της λίστας δεν έχει κάποιον επόμενο και έτσι ο δεξιά του δείκτης έχει την τιμή null.

Σε μια **απλά συνδεδεμένη κυκλική λίστα** (**singly linked circular list**) κάθε κόμβος έχει ένα μόνο δείκτη, όπως ακριβώς και στην κοινή απλά συνδεδεμένη λίστα που παρουσιάστηκε παραπάνω. Η διαφορά είναι ότι η αναφορά του τελευταίου κόμβου δεν έχει την τιμή null, αλλά δείχνει ξανά στον πρώτο κόμβο της λίστας. Η λίστα έτσι καθίσταται κυκλική και μπορεί να θεωρηθεί ότι δεν έχει ούτε αρχή, ούτε τέλος, αφού είναι δυνατόν να τη διασχίσουμε ολόκληρη ξεκινώντας από οποιονδήποτε κόμβο της. Συνήθως όμως προτιμάται να υπάρχει μια εξωτερική αναφορά η οποία να δείχνει στον τελευταίο κόμβο της λίστας. Με τον τρόπο αυτό ο προγραμματιστής έχει άμεση πρόσβαση τόσο στον τελευταίο όσο και στον πρώτο κόμβο της λίστας, δυνατότητες οι οποίες έχουν πλεονεκτήματα κατά την υλοποίηση, την ανάπτυξη και τη διαχείριση μιας τέτοιας λίστας. Μια απλά συνδεδεμένη κυκλική λίστα παρουσιάζεται στο παρακάτω σχήμα.



Η εξωτερική αναφορά δείχνει στον κόμβο με το δεδομένο 83. Άρα ο κόμβος αυτός είναι ο τελευταίος της λίστας ενώ ο πρώτος κόμβος της λίστας είναι αυτός με την τιμή 11. Μέσω της αναφοράς αυτής είναι δυνατή η άμεση πρόσβαση στον τελευταίο κόμβο της λίστας. Μέσω της αναφοράς αυτής και του δείκτη του τελευταίου κόμβου είναι δυνατή η άμεση πρόσβαση στον πρώτο κόμβο της λίστας.

Σε μια **διπλά συνδεδεμένη κυκλική λίστα** (**doubly linked circular list**) κάθε κόμβος έχει δύο δείκτες, όπως ακριβώς και στην κοινή διπλά συνδεδεμένη λίστα. Η διαφορά είναι ότι η δεξιά αναφορά του τελευταίου κόμβου δεν έχει την τιμή null, αλλά δείχνει ξανά στον πρώτο κόμβο της λίστας και η αριστερή αναφορά του πρώτου κόμβου δεν έχει την τιμή null, αλλά δείχνει στον τελευταίο κόμβο της λίστας. Το παρακάτω σχήμα παρουσιάζει ακριβώς αυτή την ιδιότητα.



Εξωτερική αναφορά δεν είναι απαραίτητη αφού όλοι οι κόμβοι είναι ισάξιοι, αλλά προτιμάται ώστε να διευκολύνει τις διεργασίες πάνω στη λίστα.

Σε κάθε έναν από τους παραπάνω τύπους συνδεδεμένων λιστών μπορεί να χρησιμοποιηθεί ένας **κόμβος φρουρός** (**sentinel node**). Ο κόμβος αυτός μπορεί να μπει είτε στην αρχή, είτε στο τέλος, είτε και στα δύο άκρα της λίστας και δε χρησιμοποιείται για την αποθήκευση δεδομένων. Ο σκοπός του είναι να απλοποιήσει ή και να επιταχύνει κάποιες διεργασίες που υλοποιούνται στις λίστες, όπως η διάσχιση της λίστας, η εισαγωγή ενός κόμβου και η διαγραφή ενός κόμβου. Τα πλεονεκτήματα ενός κόμβου φρουρού προκύπτουν από την εγγύηση που δίνει η χρήση του, ότι κάθε πραγματικός κόμβος (κόμβος με δεδομένα) στη λίστα θα έχει πάντα έναν προηγούμενο και έναν επόμενο κόμβο και ότι κάθε λίστα, ακόμη και μία η οποία δεν περιέχει καθόλου δεδομένα, πάντα θα έχει έναν τουλάχιστον κόμβο. Έτσι κάθε συνδεδεμένη λίστα σίγουρα έχει έναν πρώτο και έναν τελευταίο κόμβο.

Οι κόμβοι που χρησιμοποιούνται ως φρουροί σε μια λίστα υπάρχουν στη δομή της, εξ' αρχής, ακόμη και όταν η λίστα θεωρείται κενή (χωρίς δεδομένα). Από την άλλη πλευρά, όταν η λίστα περιέχει και άλλους κοινούς κόμβους, οι κόμβοι φρουροί δε συμμετέχουν στην αποθήκευση δεδομένων. Η παρουσία τους είναι μόνο βοηθητική ως προς τις διεργασίες που υλοποιούνται σε μια λίστα.

Διεργασίες των συνδεδεμένων λιστών

Οι τρεις βασικές λειτουργίες που υλοποιούνται σε μια συνδεδεμένη λίστα είναι η διάσχισή της, η εισαγωγή ενός κόμβου και η διαγραφή ενός κόμβου. Κατά τον προγραμματισμό των διαδικασιών που εκτελούν αυτές τις λειτουργίες μεγάλη προσοχή πρέπει να δοθεί στη χρήση των δεικτών. Πολύ εύκολα μπορεί να δημιουργηθεί λάθος από την εσφαλμένη απόδοση τιμών σε κάποιο δείκτη. Ένα πολύ απλό παράδειγμα, που παρουσιάζεται στην επόμενη παράγραφο, δείχνει πως ένα πολύ μικρό λάθος μπορεί να καταστρέψει δουλειά αρκετού χρόνου και δεδομένα μεγάλου όγκου.

Σε μια απλά συνδεδεμένη λίστα η μόνη πρόσβαση σε αυτήν και τα δεδομένα της προέρχεται από το δείκτη αρχής. Αν κατά τη διάρκεια μιας διαδικασίας η τιμή αυτού του δείκτη κατά λάθος αλλάξει και πλέον δείχνει σε λέξη μνήμης άσχετη με τη λίστα, όλα τα δεδομένα της λίστας θα έχουν χαθεί, αφού δεν είναι πλέον δυνατή η προσπέλασή τους από το πρόγραμμα. Πολλά άλλα παρόμοια λάθη μπορούν να συμβούν αν η υλοποίηση των διεργασιών μιας λίστας δε γίνει με πολύ προσεκτικό και επιδέξιο τρόπο.

Παρακάτω παρουσιάζονται οι διεργασίες διάσχισης μιας συνδεδεμένης λίστας, εισαγωγής ενός κόμβου και διαγραφής ενός κόμβου, υλοποιημένες σε μορφή ψευδοκώδικα. Ο κώδικας δίνεται ξεχωριστά για κάθε παραλλαγή της συνδεδεμένης λίστας, αφού είναι διαφορετικός σε κάθε περίπτωση. Χρησιμοποιείται η τιμή null για να υποδείξει έναν κενό δείκτη, όπως για παράδειγμα το τέλος της λίστας. Επίσης χρησιμοποιούνται κόμβοι φρουροί για τα πλεονεκτήματά τους, εφαρμόζοντας τη χρήση τους σε διάφορες περιπτώσεις.

Απλά συνδεδεμένες λίστες (singly linked lists)

Η δομή δεδομένων που αντιπροσωπεύει τον κόμβο της λίστας, είναι μια εγγραφή με δύο πεδία.

```
record Node
{
    data        // τα δεδομένα που είναι αποθηκευμένα στον κόμβο
    next        // η αναφορά στον επόμενο κόμβο της λίστας. null για τον τελευταίο κόμβο
}
```

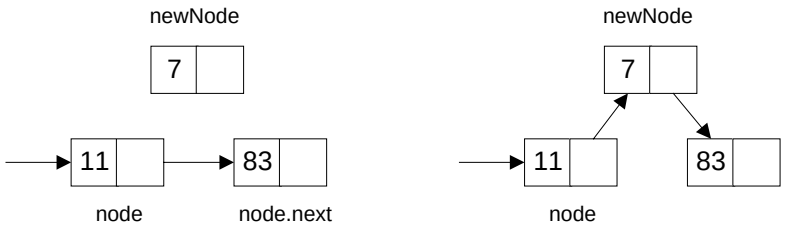
Για την πρόσβαση στη λίστα υπάρχει μια μεταβλητή, ο δείκτης αρχής. Μέσω αυτού μπορεί να προσπελαστεί ολόκληρη η λίστα. Αυτός είναι ο μόνος τρόπος πρόσβασης σ' αυτήν από το υπόλοιπο πρόγραμμα. Για το λόγο αυτό μπορεί να θεωρηθεί ότι ο δείκτης αρχής αντιπροσωπεύει ολόκληρη τη λίστα.

```
record List
{
    Node firstNode        // δείχνει στον πρώτο κόμβο της λίστας. είναι null αν η λίστα είναι κενή
}
```

Η διάσχιση μιας απλά συνδεδεμένης λίστας ξεκινάει από τον πρώτο κόμβο, μέσω του δείκτη αρχής, και συνεχίζει στους επόμενους, μέσω των δεικτών next, του κάθε κόμβου, μέχρι να φτάσει στο τέλος της λίστας.

```
διαδικασία traversal (List list)
{
    Node node                   // μεταβλητή η οποία αντιπροσωπεύει τον τρέχοντα κόμβο
    node = list.firstNode       // η διάσχιση ξεκινά από τον πρώτο κόμβο της λίστας
    while node not null         // όσο δεν έχει φτάσει στο τέλος της λίστας
    {
        node data processing    // επεξεργάζεται τα δεδομένα του τρέχοντος κόμβου
        node = node.next        // προχωράει στον επόμενο κόμβο της λίστας
    }
}
```

Για την εισαγωγή ενός νέου κόμβου στη λίστα η πρώτη πληροφορία που είναι απαραίτητη είναι η θέση στην οποία αυτός θα μπει. Αφού η λίστα είναι απλά συνδεδεμένη, σε κάθε κόμβο υπάρχει δείκτης μόνο προς τον επόμενό του και όχι προς τον προηγούμενό του. Αυτό έχει ως αποτέλεσμα η εισαγωγή ενός νέου κόμβου να μπορεί να γίνει μόνο μετά και όχι πριν από κάποιον ήδη υπάρχοντα. Σχηματικά η διαδικασία αυτή φαίνεται παρακάτω.



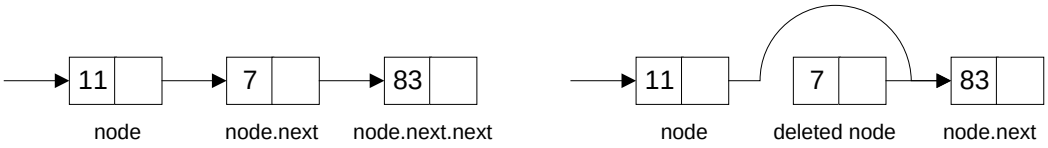
Ο περιορισμός αυτός βέβαια μπορεί να αρθεί τροποποιώντας τον αλγόριθμο, έτσι ώστε να μπορεί να βρίσκει τον προηγούμενο ενός δοθέντος κόμβου της λίστας. Ο παρακάτω ψευδοκώδικας περιγράφει την εισαγωγή ενός νέου κόμβου στη λίστα μετά από έναν ήδη υπάρχοντα σε αυτήν.

```
διαδικασία insertAfter (Node node, Node newNode)
{
    newNode.next = node.next    // ο νέος κόμβος δείχνει στον επόμενό του
    node.next = newNode         // ο προηγούμενος κόμβος δείχνει στο νέο κόμβο
}
```

Η εισαγωγή ενός κόμβου στην αρχή της λίστας απαιτεί ξεχωριστό χειρισμό και άρα διαφορετική συνάρτηση.

```
διαδικασία insertBeginning (List list, Node newNode)
{
    newNode.next = list.firtstNode      // ο νέος κόμβος δείχνει στον πρώτο κόμβο της λίστας
    list.firtstNode = newNode           // ο δείκτης αρχής δείχνει στο νέο κόμβο
}
```

Εκτός από την εισαγωγή κόμβων στη λίστα μια ακόμη βασική διεργασία είναι η διαγραφή κόμβων από αυτήν. Παρακάτω φαίνεται η σχηματική αναπαράσταση της διαγραφής ενός κόμβου που βρίσκεται μετά από έναν δοθέντα κόμβο και ο αντίστοιχος ψευδοκώδικας.




```
διαδικασία removeAfter (Node node)
{
    deletedNode = node.next        // μεταβλητή που δείχνει στον κόμβο που θα διαγραφεί
    node.next = node.next.next     // ο προηγούμενος κόμβος δείχνει στον επόμενο, παρακάμπτοντας
                                   // αυτόν που είναι προς διαγραφή
    destroy deletedNode            // αποδεσμεύεται η μνήμη που κατείχε ο κόμβος
}
```

Στο τέλος της συνάρτησης ο κόμβος που μόλις διαγράφηκε από τη λίστα, καταστρέφεται, ώστε να μην καταλαμβάνει πλέον χώρο στη μνήμη και να μην επιβαρύνει το σύστημα. Και πάλι για τη διαγραφή ενός κόμβου πριν από κάποιον άλλο γνωστό και όχι μετά από κάποιον, είναι απαραίτητη μια συνάρτηση ανίχνευσης του προηγούμενου ενός κόμβου της λίστας.

Η διαγραφή του πρώτου κόμβου της λίστας απαιτεί ξεχωριστό χειρισμό και άρα διαφορετική συνάρτηση.

```
διαδικασία removeBeggining (List list)
{
    deletedNode = list.firstNode          // μεταβλητή που δείχνει στον κόμβο που θα διαγραφεί
    list.firstNode = list.firstNode.next  // ο δείκτης αρχής δείχνει στον επόμενο του,
                                          // παρακάμπτοντας αυτόν που είναι προς διαγραφή
    destroy deletedNode                   // αποδεσμεύεται η μνήμη που κατείχε ο κόμβος
}
```

Σε περίπτωση που η λίστα έχει μόνο έναν κόμβο τότε η συνάρτηση αυτή δίνει στο δείκτη αρχής την τιμή null, μετά τη διαγραφή του κόμβου.

Διπλά συνδεδεμένες λίστες (doubly linked lists)

Ως προς την υλοποίησή τους, οι διπλά συνδεδεμένες λίστες έχουν ένα μειονέκτημα και ένα πλεονέκτημα. Από τη μια μεριά υπάρχουν περισσότεροι δείκτες για διαχείριση και ενημέρωση σε κάθε διεργασία, κάνοντας έτσι την υλοποίηση των λειτουργιών λίγο πιο δύσκολη. Από την άλλη όμως είναι απαραίτητες λιγότερες πληροφορίες για την υλοποίηση μιας διεργασίας, αφού επιπλέον υπάρχουν οι προς τα πίσω δείκτες, μέσω των οποίων είναι δυνατή η άμεση προσπέλαση του προηγούμενου ενός κόμβου της λίστας. Αυτά οδηγούν στη δημιουργία διαφορετικών διαδικασιών για τη δημιουργία, το χειρισμό και γενικότερα τη λειτουργία της λίστας.

Η δομή δεδομένων που χρησιμοποιείται για τη διπλά συνδεδεμένη λίστα έχει ένα επιπλέον στοιχείο από την αντίστοιχη της απλά συνδεδεμένης λίστας. Το τρίτο απαραίτητο πεδίο είναι η αναφορά στον προηγούμενο κόμβο.

```
record Node
{
    data        // τα δεδομένα που είναι αποθηκευμένα στον κόμβο
    next        // η αναφορά στον επόμενο κόμβο της λίστας. null για τον τελευταίο κόμβο
    prev        // η αναφορά στον προηγούμενο κόμβο της λίστας. null για τον πρώτο κόμβο
}
```

Αλλαγή υπάρχει και στη δομή της λίστας. Έχει προστεθεί ένα επιπλέον δεδομένο, μια αναφορά, η οποία δείχνει πάντα στον τελευταίο κόμβο της λίστας. Έτσι η πρόσβαση στη λίστα γίνεται όχι μόνο από το πρώτο στοιχείο της, αλλά και από το τελευταίο της. Αντίστοιχα με την ονομασία δείκτης αρχής, η νέα αυτή αναφορά ονομάζεται **δείκτης τέλους** της λίστας. Η παρουσία του μπορεί να διευκολύνει την υλοποίηση και να επιταχύνει τη λειτουργία κάποιων διεργασιών. Επίσης κάποιες διεργασίες δε θα μπορούσαν να υλοποιηθούν χωρίς το πλεονέκτημα που μας παρέχει, το οποίο δεν είναι άλλο από την άμεση πρόσβαση στον τελευταίο κόμβο της λίστας. Αποτελεί άλλη μια περίπτωση επιπλέον πληροφορίας που περιέχει η διπλά συνδεδεμένη λίστα σε σχέση με την απλά συνδεδεμένη και άλλη μια περίπτωση εξωτερικής αναφοράς προς τη λίστα. Ένα σημείο εισόδου στη λίστα από το υπόλοιπο πρόγραμμα.

```
record List
{
    Node firstNode      // δείχνει στον πρώτο κόμβο της λίστας. είναι null αν η λίστα είναι κενή
    Node lastNode       // δείχνει στον τελευταίο κόμβο της λίστας. είναι null αν η λίστα είναι κενή
}
```

Κάθε κόμβος έχει δύο δείκτες. Έναν προς τον προηγούμενο και έναν προς τον επόμενό του κόμβο. Αυτό έχει ως αποτέλεσμα η λίστα να μπορεί να διασχιστεί και προς τις δύο κατευθύνσεις.

Προς τα εμπρός διάσχιση

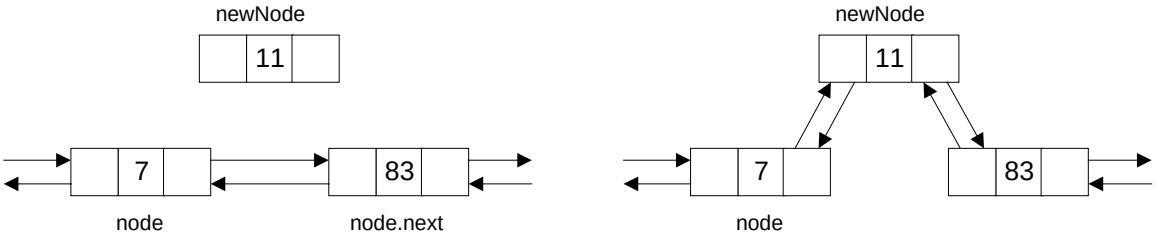
```
διαδικασία traversalForwards (List list)
{
    Node node                   // μεταβλητή η οποία αντιπροσωπεύει τον τρέχοντα κόμβο
    node = list.firstNode       // η διάσχιση ξεκινά από τον πρώτο κόμβο της λίστας
    while node not null         // όσο δεν έχει φτάσει στο τέλος της λίστας
    {
        node data processing    // επεξεργάζεται τα δεδομένα του τρέχοντος κόμβου
        node = node.next        // προχωράει στον επόμενο κόμβο της λίστας
    }
}
```

Προς τα πίσω διάσχιση

```
διαδικασία traversalBackwards (List list)
{
    Node node                   // μεταβλητή η οποία αντιπροσωπεύει τον τρέχοντα κόμβο
    node = list.lastNode        // η διάσχιση ξεκινά από τον τελευταίο κόμβο της λίστας
    while node not null         // όσο δεν έχει φτάσει στην αρχή της λίστας
    {
        node data processing    // επεξεργάζεται τα δεδομένα του τρέχοντος κόμβου
        node = node.prev        // προχωράει στον προηγούμενο κόμβο της λίστας
    }
}
```

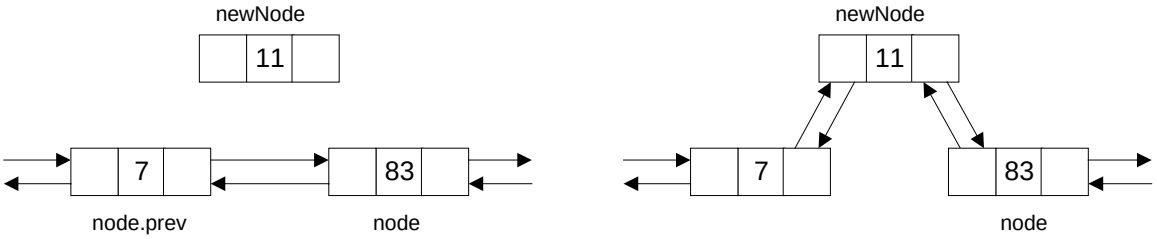
Η εισαγωγή ενός νέου κόμβου στη λίστα μπορεί να γίνει είτε μετά, είτε πριν από κάποιον υπάρχοντα κόμβο. Στην περίπτωση της διπλά συνδεδεμένης λίστας υπάρχουν και οι δύο αυτές δυνατότητες, αντίθετα με την περίπτωση της απλά συνδεδεμένης λίστας. Παρακάτω παρουσιάζονται οι σχηματικές παραστάσεις και οι αντίστοιχες διαδικασίες σε ψευδοκώδικα των δύο περιπτώσεων.

Εισαγωγή μετά από γνωστό κόμβο




```
διαδικασία insertAfter (List list, Node node, Node newNode)
        // list είναι η λίστα
        // Node είναι ο κόμβος μετά τον οποίο θα μπει ο νέος κόμβος. ο τρέχον κόμβος
        // newNode είναι ο νέος κόμβος
{
    newNode.prev = node                 // η αριστερή αναφορά του νέου κόμβου δείχνει στον
                                        // τρέχοντα κόμβο
    newNode.next = node.next            // η δεξιά αναφορά του νέου κόμβου δείχνει στον επόμενο
                                        // του τρέχοντος κόμβου
    if node.next is null                // αν δεν υπάρχει επόμενος του τρέχοντος κόμβου
        list.lastNode = newNode         // ο δείκτης τέλους δείχνει στο νέο κόμβο
    else                                // αν υπάρχει επόμενος του τρέχοντος κόμβου
        node.next.prev = newNode        // η αριστερή αναφορά αυτού δείχνει στο νέο κόμβο
    node.next = newNode                 // η δεξιά αναφορά του τρέχοντος κόμβου δείχνει στο νέο
                                        // κόμβο
}
```

Εισαγωγή πριν από γνωστό κόμβο




```
διαδικασία insertBefore (List list, Node node, Node newNode)
        // list είναι η λίστα
        // Node είναι ο κόμβος πριν τον οποίο θα μπει ο νέος κόμβος. ο τρέχον κόμβος
        // newNode είναι ο νέος κόμβος
{
    newNode.prev = node.prev            // η αριστερή αναφορά του νέου κόμβου δείχνει στον
                                        // προηγούμενο του τρέχοντος κόμβου
    newNode.next = node                 // η δεξιά αναφορά του νέου κόμβου δείχνει στον τρέχοντα
                                        // κόμβο
    if node.prev is null                // αν δεν υπάρχει προηγούμενος του τρέχοντος κόμβου
        list.firstNode = newNode        // ο δείκτης αρχής δείχνει στο νέο κόμβο
    else                                // αν υπάρχει προηγούμενος του τρέχοντος κόμβου
        node.prev.next = newNode        // η δεξιά αναφορά αυτού δείχνει στο νέο κόμβο
    node.next = newNode                 // η δεξιά αναφορά του τρέχοντος κόμβου δείχνει στο νέο
                                        // κόμβο
}
```

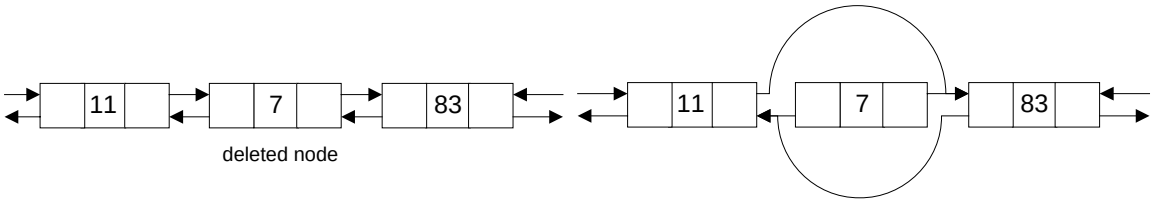
Επίσης είναι απαραίτητη η δημιουργία μιας συνάρτησης για την εισαγωγή ενός κόμβου στην αρχή μιας πιθανόν κενής λίστας.

```
διαδικασία insertBeginning (List list, Node newNode)
{
    if list.firstNode is null                    // αν η λίστα είναι κενή
    {
        list.firstNode = newNode                 // ο δείκτης αρχής δείχνει στον νέο κόμβο
        list.lastNode = newNode                  // ο δείκτης τέλους δείχνει στον νέο κόμβο
        newNode.prev = null                      // ο αριστερός δείκτης του νέου κόμβου δε δείχνει πουθενά
        newNode.next = null                      // ο δεξιός δείκτης του νέου κόμβου δε δείχνει πουθενά
    }
    else                                         // αν η λίστα δεν είναι κενή
        insertBefore(list, list.firstNode, newNode)  // απλά εισάγεται ο νέος κόμβος πριν τον πρώτο
                                                 // της κόμβο
}
```

Μια παρόμοια συνάρτηση εισάγει έναν κόμβο στο τέλος μιας πιθανόν κενής λίστας.

```
διαδικασία insertEnd (List list, Node newNode)
{
    if list.lastNode is null                     // αν η λίστα είναι κενή
        insertBeginning(list, newNode)           // ο νέος κόμβος εισάγεται απλά στην αρχή της
    else                                         // αν η λίστα δεν είναι κενή
        insertAfter(list, list.lastNode, newNode)    // ο νέος κόμβος εισάγεται μετά τον τελευταίο της
                                                 // κόμβο
}
```

Κατά τη διαγραφή κόμβων από τη λίστα, ιδιαίτερη προσοχή χρειάζεται ο χειρισμός των δεικτών αρχής και τέλους της λίστας. Αντίθετα με την περίπτωση της απλά συνδεδεμένης λίστας, δε χρειάζεται να προσδιοριστεί ο προηγούμενος κόμβος του κόμβου προς διαγραφή.




```
διαδικασία remove (List list, Node node)
{
    if node.prev is null                         // αν ο κόμβος προς διαγραφή είναι ο πρώτος κόμβος
        list.firstNode = node.next               // ο δείκτης αρχής δείχνει στον επόμενό του
    else                                         // αν ο κόμβος προς διαγραφή δεν είναι ο πρώτος κόμβος
        node.prev.next = node.next               // ο προηγούμενός του κόμβος δείχνει στον επόμενό του
    if node.next is null                         // αν ο κόμβος προς διαγραφή είναι ο τελευταίος κόμβος
        list.lastNode = node.prev                // ο δείκτης τέλους δείχνει στον προηγούμενό του
    else                                         // αν ο κόμβος προς διαγραφή δεν είναι ο τελευταίος
                                                 // κόμβος
        node.next.prev = node.prev               // ο επόμενός του κόμβος δείχνει στον προηγούμενό του
    destroy node                                 // ο κόμβος που μόλις διαγράφηκε, σβήνεται από τη μνήμη
}
```

Στο τέλος της συνάρτησης ο κόμβος που μόλις διαγράφηκε από τη λίστα, καταστρέφεται, ώστε να μην καταλαμβάνει πλέον χώρο στη μνήμη και να μην επιβαρύνει το σύστημα. Η παραπάνω διαδικασία διαχειρίζεται σωστά τις περιπτώσεις διαγραφής του πρώτου ή του τελευταίου κόμβου της λίστας δίνοντας τις σωστές τιμές στους δείκτες αρχής και τέλους. Ο συνδυασμός των δύο αυτών περιπτώσεων πραγματοποιείται κατά τη διαγραφή του μοναδικού κόμβου της λίστας, μετά την οποία και οι δύο δείκτες έχουν την τιμή null.

Απλά συνδεδεμένες κυκλικές λίστες (singly linked circular lists)

Σε μια κυκλική λίστα, είτε είναι απλά είτε διπλά συνδεδεμένη, οι κόμβοι βρίσκονται συνδεδεμένοι σε ένα συνεχόμενο κύκλο. Κανένας από τους δείκτες των κόμβων δεν έχει την τιμή null. Στην περίπτωση αυτή δεν είναι απαραίτητη η χρήση και του δείκτη αρχής και του δείκτη τέλους. Μόνο ένας δείκτης αρκεί και αυτός είναι ο δείκτης που δείχνει στον τελευταίο κόμβο της λίστας. Μέσω αυτού και του δεξιού δείκτη του τελευταίου κόμβου πραγματοποιείται η άμεση προσπέλαση και του πρώτου κόμβου της λίστας.

Στις απλά συνδεδεμένες κυκλικές λίστες υπάρχει η δυνατότητα διάσχισής τους ξεκινώντας από οποιονδήποτε κόμβο. Όπως εξηγήθηκε παραπάνω, για την πρόσβαση στη λίστα και τη διάσχισή της από το υπόλοιπο πρόγραμμα η μόνη απαραίτητη μεταβλητή είναι ένας δείκτης τέλους. Επίσης απαραίτητη είναι μια αναπαράσταση για την περίπτωση που η λίστα είναι κενή. Έτσι όταν ο δείκτης τέλους είναι null, αυτό σημαίνει ότι η λίστα είναι κενή.

Η εγγραφή που αναπαριστά τον κόμβο της απλά συνδεδεμένης κυκλικής λίστας είναι όμοια με αυτήν της μη κυκλικής.

```
record Node
{
    data        // τα δεδομένα που είναι αποθηκευμένα στον κόμβο
    next        // η αναφορά στον επόμενο κόμβο της λίστας. null για τον τελευταίο κόμβο
}
```

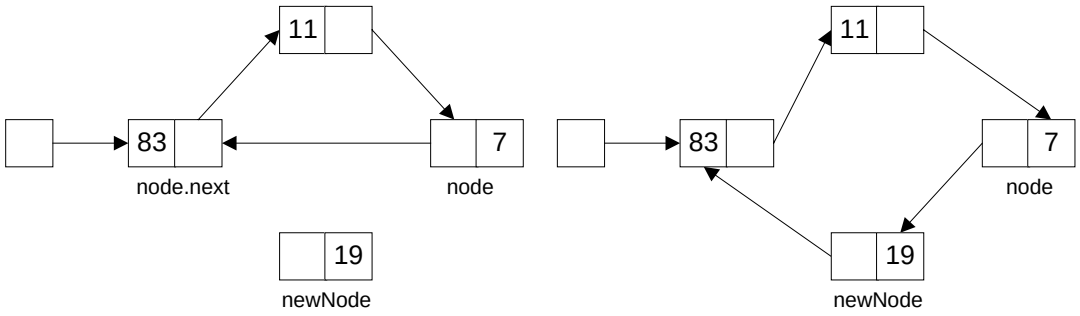
Ο δείκτης τέλους αντιπροσωπεύει ολόκληρη τη λίστα.

```
record List
{
    Node lastNode        // δείχνει στον τελευταίο κόμβο της λίστας. είναι null αν η λίστα είναι κενή
}
```

Η διάσχιση μιας απλά συνδεδεμένης κυκλικής λίστας μπορεί να ξεκινήσεις από οποιονδήποτε κόμβο και μπορεί να πραγματοποιηθεί μόνο προς τη μία κατεύθυνση. Ένας βοηθητικός δείκτης είναι αυτός που διατρέχει τον κάθε κόμβο της λίστας μέχρι να περάσει από όλους τους κόμβους της.

```
διαδικασία traversal (Node node)
        // node είναι ο κόμβος από τον οποίο θα ξεκινήσει η διάσχιση της λίστας
        // η διαδικασία θα συνεχίσει να προσπελαύνει κόμβους προχωρώντας κυκλικά
        // μέχρι να ξαναβρεθεί στον κόμβο node
{
    runner = node                 // βοηθητική μεταβλητή, η οποία θα διασχίσει ολόκληρη τη λίστα
    do                            // επαναλαμβάνει
    {
        node data processing      // επεξεργάζεται τα δεδομένα του τρέχοντος κόμβου
        runner = runner.next      // προχωράει στον επόμενο κόμβο της λίστας
    }
    while runner <> node          // όσο δεν έχει φτάσει και πάλι στον κόμβο από τον οποίο άρχισε
                                  // δηλαδή όσο δεν έχει διασχίσει ολόκληρη τη λίστα
}
```

Η εισαγωγή ενός κόμβου μπορεί να γίνει μόνο μετά από έναν υπάρχοντα κόμβο και όχι πριν από αυτόν. Αφού η λίστα είναι κυκλική και όλοι οι κόμβοι σε αυτήν είναι ισάξιοι, δε χρειάζεται ειδική διαδικασία για την εισαγωγή ενός νέου κόμβου στην αρχή ή στο τέλος της λίστας. Οι λειτουργίες αυτές εκτελούνται με την ίδια διαδικασία. Παρακάτω φαίνεται η σχηματική αναπαράσταση και ο ψευδοκώδικας της εισαγωγής ενός νέου κόμβου σε μια απλά συνδεδεμένη κυκλική λίστα.

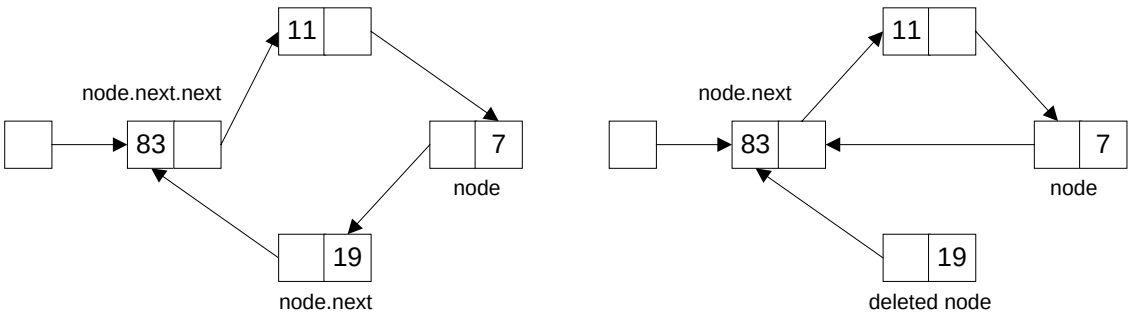



```
διαδικασία insertAfter (List list, Node node, Node newNode)
{
    newNode.next = node.next      // ο νέος κόμβος δείχνει στον επόμενό του
    node.next = newNode           // ο προηγούμενος κόμβος δείχνει στο νέο κόμβο
}
```

Η εισαγωγή ενός κόμβου σε μια πιθανόν κενή λίστα απαιτεί ξεχωριστό χειρισμό.

```
διαδικασία insertInEmpty (List list, Node newNode)
{
    list.lastNode = newNode
}
```

Η διαγραφή ενός κόμβου μπορεί να γίνει μόνο αν είναι γνωστός στη διαδικασία ο προηγούμενός του κόμβος. Παρακάτω φαίνεται η σχηματική αναπαράσταση της διαγραφής ενός κόμβου που βρίσκεται μετά από έναν δοθέντα κόμβο και ο αντίστοιχος ψευδοκώδικας.




```
διαδικασία removeAfter (Node node)
{
    deletedNode = node.next       // μεταβλητή που δείχνει στον κόμβο που θα διαγραφεί
    node.next = node.next.next    // ο προηγούμενος κόμβος δείχνει στον επόμενο, παρακάμπτοντας
                                  // αυτόν που είναι προς διαγραφή
    destroy deletedNode           // αποδεσμεύεται η μνήμη που κατείχε ο κόμβος
}
```

Στο τέλος της διαδικασίας ο κόμβος που μόλις διαγράφηκε από τη λίστα, καταστρέφεται, ώστε να μην καταλαμβάνει πλέον χώρο στη μνήμη του συστήματος και να μην επιβαρύνει το σύστημα. Η διαγραφή του πρώτου ή του τελευταίου κόμβου της λίστας δε χρειάζεται ξεχωριστή αντιμετώπιση. Η παραπάνω διαδικασία, έτσι όπως είναι υλοποιημένη, καλύπτει αυτές τις δύο περιπτώσεις.

Διπλά συνδεδεμένες κυκλικές λίστες (doubly linked circular lists)

Στις διπλά συνδεδεμένες λίστες, είτε κυκλικές είτε μη, κάθε κόμβος περιέχει δύο δείκτες. Ο ένας από αυτούς προς τον επόμενο και ο άλλος προς τον προηγούμενό του κόμβο στη λίστα. Αφού η λίστα είναι κυκλική κάνείς από τους δείκτες αυτούς δεν είναι null σε κανέναν κόμβο. Η δεξιά αναφορά του τελευταίου κόμβου δείχνει στον πρώτο κόμβο και η αριστερή αναφορά του πρώτου κόμβου δείχνει στον τελευταίο κόμβο. Οι δύο αυτοί δείκτες στις μη κυκλικές διπλά συνδεδεμένες λίστες έχουν την τιμή null.

Η εγγραφή που περιγράφει τον κόμβο περιέχει τρία πεδία.

```
record Node
{
    data        // τα δεδομένα που είναι αποθηκευμένα στον κόμβο
    next        // η αναφορά στον επόμενο κόμβο της λίστας. null για τον τελευταίο κόμβο
    prev        // η αναφορά στον προηγούμενο κόμβο της λίστας. null για τον πρώτο κόμβο
}
```

Όπως και στις απλά συνδεδεμένες κυκλικές λίστες, η μόνη πληροφορία που χρειάζεται για την προσπέλαση ολόκληρης της λίστας είναι ένας και μόνο εξωτερικός δείκτης. Αφού η λίστα είναι διπλά συνδεδεμένη αυτός μπορεί να δείχνει είτε στον πρώτο της κόμβο, είτε στον τελευταίο. Για ομοιομορφία με την απλά συνδεδεμένη κυκλική λίστα επιλέχθηκε ο δείκτης τέλους. Μέσω αυτού, αφού δείχνει στον τελευταίο κόμβο της λίστας, είναι δυνατή και η άμεση αναφορά στον πρώτο της κόμβο.

```
record List
{
    Node lastNode        // δείχνει στον τελευταίο κόμβο της λίστας. είναι null αν η λίστα είναι κενή
}
```

Στις διπλά συνδεδεμένες κυκλικές λίστες είναι δυνατή η διάσχιση και προς τις δύο κατευθύνσεις, ξεκινώντας από οποιονδήποτε κόμβο. Ένας βοηθητικός δείκτης είναι αυτός που διατρέχει τον κάθε κόμβο της λίστας μέχρι να περάσει από όλους τους κόμβους της.

Προς τα εμπρός διάσχιση

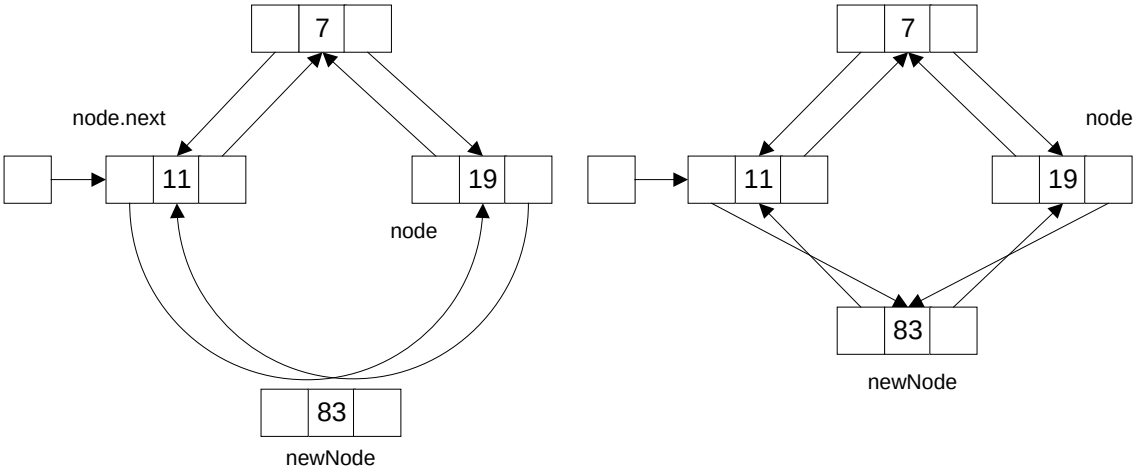
```
διαδικασία traversalForwards (Node node)
        // node είναι ο κόμβος από τον οποίο θα ξεκινήσει η διάσχιση της λίστας
{
    runner = node                // βοηθητική μεταβλητή, η οποία θα διασχίσει ολόκληρη τη λίστα
    do                           // επαναλαμβάνει
    {
        node data processing     // επεξεργάζεται τα δεδομένα του τρέχοντος κόμβου
        runner = runner.next     // προχωράει στον επόμενο κόμβο της λίστας
    }
    while runner <> node         // όσο δεν έχει φτάσει και πάλι στον κόμβο από τον οποίο άρχισε
                                 // δηλαδή όσο δεν έχει διασχίσει ολόκληρη τη λίστα
}
```

Προς τα πίσω διάσχιση

```
διαδικασία traversalBackwards (Node node)
        // node είναι ο κόμβος από τον οποίο θα ξεκινήσει η διάσχιση της λίστας
{
    runner = node                   // βοηθητική μεταβλητή, η οποία θα διασχίσει ολόκληρη τη λίστα
    do                              // επαναλαμβάνει
    {
        node data processing        // επεξεργάζεται τα δεδομένα του τρέχοντος κόμβου
        runner = runner.prev        // προχωράει στον προηγούμενο κόμβο της λίστας
    }
    while runner <> node            // όσο δεν έχει φτάσει και πάλι στον κόμβο από τον οποίο άρχισε
                                    // δηλαδή όσο δεν έχει διασχίσει ολόκληρη τη λίστα
}
```

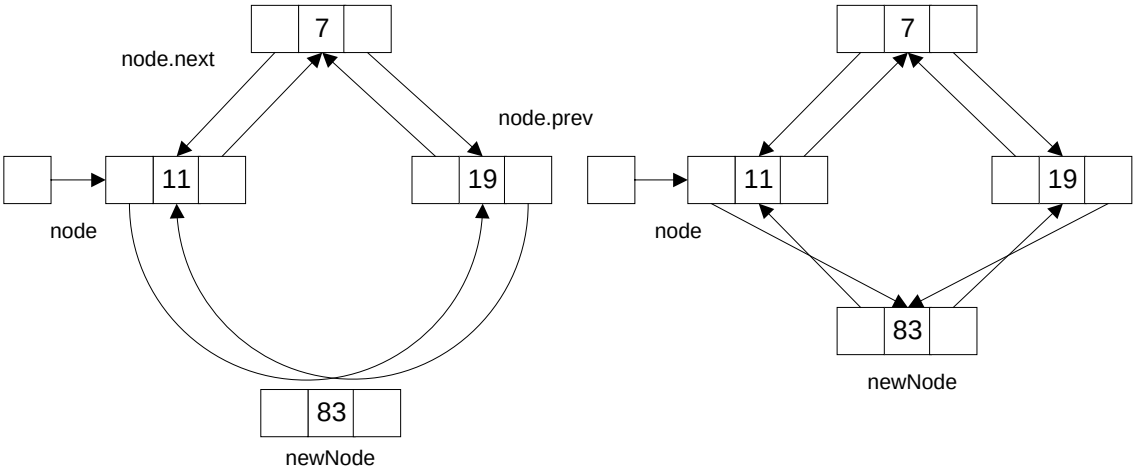
Η εισαγωγή ενός νέου κόμβου στη λίστα μπορεί να γίνει είτε μετά, είτε πριν από κάποιον υπάρχοντα κόμβο. Όλοι οι κόμβοι στη διπλά συνδεδεμένη κυκλική λίστα είναι ισάξιοι και για το λόγο αυτό η εισαγωγή ενός νέου κόμβου πριν τον πρώτο ή μετά τον τελευταίο κόμβο δεν αποτελούν ξεχωριστές περιπτώσεις και εκτελούνται επιτυχώς από την ίδια διαδικασία. Παρακάτω παρουσιάζονται οι σχηματικές παραστάσεις και οι αντίστοιχες διαδικασίες σε ψευδοκώδικα των δύο περιπτώσεων.

Εισαγωγή μετά από γνωστό κόμβο




```
διαδικασία insertAfter (Node node, Node newNode)
        // Node είναι ο κόμβος μετά τον οποίο θα μπει ο νέος κόμβος. ο τρέχον κόμβος
        // newNode είναι ο νέος κόμβος
{
    newNode.next = node.next        // η δεξιά αναφορά του νέου κόμβου δείχνει εκεί που
                                    // έδειχνε η δεξιά αναφορά του τρέχοντος κόμβου
    newNode.prev = node             // η αριστερή αναφορά του νέου κόμβου δείχνει στον
                                    // τρέχοντα κόμβο
    node.next.prev = newNode        // η αριστερή αναφορά του επόμενου από τον τρέχοντα
                                    // κόμβο δείχνει στο νέο κόμβο
    node.next = newNode             // η δεξιά αναφορά του τρέχοντος κόμβου δείχνει στο νέο
                                    // κόμβο
}
```

Εισαγωγή πριν από γνωστό κόμβο




```
διαδικασία insertBefore (Node node, Node newNode)
        // Node είναι ο κόμβος πριν τον οποίο θα μπει ο νέος κόμβος. ο τρέχον κόμβος
        // newNode είναι ο νέος κόμβος
{
    insertAfter (node.prev, newNode)
}
```

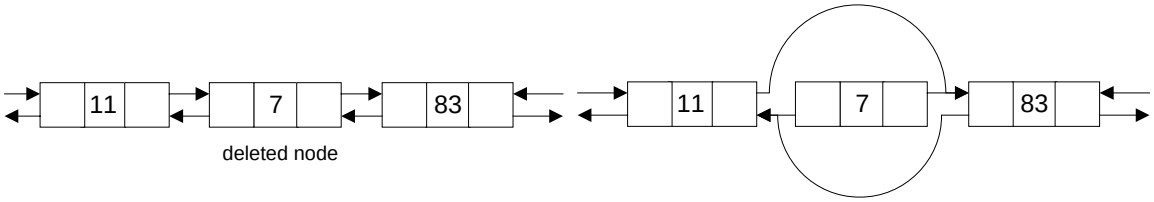
Η εισαγωγή ενός στοιχείου σε μια πιθανόν κενή λίστα έχει κάποια ιδιαιτερότητα και απαιτεί ξεχωριστή διαδικασία.

```
διαδικασία insertEnd (List list, Node newNode)
{
    if list.lastNode is null                    // αν η λίστα είναι κενή
    {
        newNode.prev = newNode                  // η αριστερή αναφορά του νέου κόμβου δείχνει στον ίδιο
                                                // τον κόμβο
        newNode.next = newNode                  // η δεξιά αναφορά του νέου κόμβου δείχνει στον ίδιο τον
        list.lastNode = newNode                 // κόμβο
    }
    else                                                    // αν η λίστα δεν είναι κενή
        insertBefore(list, list.firstNode, newNode)         // απλά εισάγεται ο νέος κόμβος πριν τον πρώτο
                                                            // της κόμβο
}
```

Η εισαγωγή ενός νέου κόμβου στην αρχή της λίστας γίνεται πολύ απλά χρησιμοποιώντας τον τελευταίο κόμβο της λίστας και την εξωτερική αναφορά που δείχνει σε αυτόν.

```
διαδικασία insertBeginning (List list, Node newNode)
{
    insertAfter (list.lastNode, newNode)
}
```

Κατά τη διαγραφή κόμβων από τη λίστα, ιδιαίτερη προσοχή χρειάζεται ο χειρισμός των δεικτών των κόμβων που εμπλέκονται στη διαδικασία καθώς και του δείκτη τέλους της λίστας. Επίσης χρειάζεται να προσεχθεί η περίπτωση που διαγράφεται ο μοναδικός κόμβος της λίστας.




```
διαδικασία remove (List list, Node node)
{
    if node.next =node                  // αν ο κόμβος προς διαγραφή είναι ο μοναδικός κόμβος
                                        // της λίστας
        list.lastNode = null            // ο δείκτης τέλους παίρνει την τιμή null. η λίστα είναι
                                        // πλέον κενή
    else                                // αν ο κόμβος προς διαγραφή δεν είναι ο μοναδικός
    {
        node.next.prev = node.prev      // ο επόμενός του κόμβος δείχνει στον προηγούμενό του
        node.prev.next = node.next      // ο προηγούμενός του κόμβος δείχνει στον επόμενό του
        if node = list.lastNode         // αν ο κόμβος προς διαγραφή είναι ο τελευταίος κόμβος
            list.lastNode = node.prev   // ο δείκτης τέλους δείχνει στον προηγούμενό του
    destroy node                        // ο κόμβος που μόλις διαγράφηκε, σβήνεται από τη μνήμη
}
```

Η διαδικασίες για τη διαγραφή ενός κόμβου μετά ή πριν από κάποιον άλλο στη λίστα μπορούν εύκολα να υλοποιηθούν με τη βοήθεια της παραπάνω διαδικασίας. Η δυνατότητα αυτή είναι εφικτή επειδή οι κόμβοι της διπλά συνδεδεμένης κυκλικής λίστας αφενός έχουν δείκτες τόσο στον προηγούμενο όσο και στον επόμενό τους κόμβο και αφετέρου είναι όλοι ισάξιοι στον κύκλο της λίστας.

```
διαδικασία removeAfter (List list, Node node)
        // list είναι η λίστα
        // node είναι ο κόμβος, ο επόμενος του οποίου είναι προς διαγραφή
{
    remove (list, node.next)
}
```

```
διαδικασία removeBefore (List list, Node node)
        // list είναι η λίστα
        // node είναι ο κόμβος, ο προηγούμενος του οποίου είναι προς διαγραφή
{
    remove (list, node.prev)
}
```

Ιστορία

Οι συνδεδεμένες λίστες αναπτύχθηκαν για πρώτη φορά το 1955 – 66 από τους Allen Newell, Cliff Shaw και Herbert Simon στη RAND Corporation. Τις χρησιμοποίησαν ως την πρωταρχική δομή δεδομένων για τη δημιουργία της Γλώσσα Επεξεργασίας Πληροφορίας (Information Processing Language – IPL). Με τη βοήθεια της γλώσσας αυτής ανέπτυξαν αρκετά προγράμματα τεχνητής νοημοσύνης, όπως η Μηχανή Λογικής Θεωρίας (Logic Theory Machine), ο Γενικός Διαχειριστής Προβλημάτων (General Problem Solver) και ένα πρόγραμμα που παίζει σκάκι [10000000000000000].

Στους ίδιους επιστήμονες και συγκεκριμένα σε μια εργασία τους σχετική με τον προγραμματισμό της Μηχανής Λογικής Θεωρίας το έτος 1957, οφείλεται και η αναπαράσταση των συνδεδεμένων λιστών. Τα κλασσικά πλέον διαγράμματα, που αποτελούνται από ορθογώνια, τα οποία αντιπροσωπεύουν τους κόμβους και βέλη, τα οποία αντιπροσωπεύουν τους δείκτες στους διαδοχικούς κόμβους της λίστας, παρουσιάστηκαν για πρώτη φορά στην εργασία αυτή [10000000000000000].

Το πρόβλημα της μετάφρασης μέσω μηχανής για φυσικές γλώσσες οδήγησε τον Victor Yngve στο Massachusetts Institute of Technology (MIT) να χρησιμοποιήσει συνδεδεμένες λίστες ως δομή δεδομένων στη γλώσσα προγραμματισμού COMIT. Η γλώσσα αυτή σχεδιάστηκε και αναπτύχθηκε το έτος 1958 για τη διευκόλυνση από ηλεκτρονικούς υπολογιστές της έρευνας στο πεδίο της γλωσσολογίας.

Η γλώσσα προγραμματισμού LISP, καθιερωμένη συντομογραφία της φράσης list processor (επεξεργαστής λιστών), δημιουργήθηκε από τον John McCarthy το έτος 1958 όσο αυτός ανήκε στο Massachusetts Institute of Technology (MIT) [10000000000000000]. Στη γλώσσα αυτή η σημαντικότερη δομή δεδομένων είναι οι συνδεδεμένες λίστες. Αποτέλεσε την απαρχή των συναρτησιακών γλωσσών προγραμματισμού. Τα προγράμματα που είναι γραμμένα σε LISP περιέχουν συναρτήσεις διαχείρισης λιστών, όπως αυτές που παρουσιάστηκαν παραπάνω.

Στις αρχές της δεκαετίας του 1960, ήταν φανερό και δεδομένο το πλεονέκτημα των συνδεδεμένων λιστών και των γλωσσών προγραμματισμού που τις χρησιμοποιούν ως βασική δομή δεδομένων. Ο Bert Green, ο οποίος ανήκε στο MIT Lincoln Laboratory, δημοσίευσε το έτος 1961 ένα άρθρο [10000000000000000] στο οποίο παρουσίασε τα πλεονεκτήματα της προσέγγισης με χρήση συνδεδεμένων λιστών. Ένα ακόμη άρθρο ανασκόπησης για τις συνδεδεμένες λίστες γράφτηκε από τους Bobrow και Raphael τον Απρίλιο του 1964.

Διάφορα λειτουργικά συστήματα, που αναπτύχθηκαν από το Technical Systems Consultants, χρησιμοποιούν απλά συνδεδεμένες λίστες ως δομές αρχείων. Ένας κατάλογος εισάγει, καταχωρεί και αποθηκεύει αναφορές (δείκτες) προς τον πρώτο τομέα του κάθε αρχείου. Μπορούν να παρομοιαστούν με τους δείκτες αρχής που χρησιμοποιήθηκαν στα παραδείγματα. Στη συνέχεια οι ακόλουθοι τομείς των αρχείων προσπελάζονται από διαδοχικούς δείκτες, οι οποίοι μπορούν να παρομοιαστούν με τον δείκτη προς τον επόμενο κόμβο στις λίστες των παραδειγμάτων. Λειτουργικά συστήματα που χρησιμοποιούν αυτή την τεχνική είναι τα Flex (για το Motorola 6800 CPU), mini–Flex (για το Motorola 6800 CPU) και Flex9 (για το Motorola 6809 CPU). Μια παραλλαγή που αναπτύχθηκε από το TSC χρησιμοποιεί διπλά συνδεδεμένες λίστες για τη διεκπεραίωση της ίδιας λειτουργίας.

Το λειτουργικό σύστημα TSS, το οποίο αναπτύχθηκε από την IBM, χρησιμοποιεί διπλά συνδεδεμένες λίστες για τη διαχείριση του συστήματος αρχείων και καταλόγων. Η δομή των καταλόγων είναι ίδια με αυτή του Unix, όπου ένας κατάλογος μπορεί να περιέχει τόσο αρχεία όσο και άλλους καταλόγους, οι οποίοι μπορεί να εκτείνονται σε οποιοδήποτε βάθος. Η δομή της συνδεδεμένης λίστας είναι ιδανική για την υλοποίηση αυτού του συστήματος καταλόγων.

Επίσης με τη χρήση των συνδεδεμένων λιστών στα λειτουργικά αυτά συστήματα αναπτύχθηκε η δυνατότητα διόρθωσης του συστήματος αρχείων μετά από πιθανή βλάβη. Όταν μια διακοπή τροφοδοσίας ή κάποιο άλλο παρόμοιο πρόβλημα συμβεί στη μνήμη του υπολογιστή, οι τομείς των αρχείων και του συστήματος καταλόγων διαφοροποιούνται, παίρνοντας λανθασμένες τιμές. Τα προβλήματα που δημιουργούνται μπορούν να ανιχνευθούν από τη σύγκριση των αριστερών και των δεξιών δεικτών των κόμβων και την αναζήτηση συνέπειας μεταξύ αυτών. Αν η τιμή ενός δεξιού δείκτη έχει επηρεαστεί, τότε αρκεί να βρεθεί ο κόμβος του οποίου ο αριστερός δείκτης δείχνει στον κόμβος με τον επηρεασμένο δεξιό δείκτη. Το μόνο που απαιτεί η διόρθωση είναι ο δεξιός επηρεασμένος δείκτης να πάρει την τιμή της διεύθυνσης του κόμβου που μόλις βρέθηκε.

Εκτός από όλα αυτά τα παραδείγματα, καθημερινά πολλές εταιρίες και προγραμματιστές χρησιμοποιούν τις συνδεδεμένες λίστες ως δομή δεδομένων στις εφαρμογές τους, αναγνωρίζοντας τα πολλά πλεονεκτήματα που τους προσφέρουν.

Σύγκριση συνδεδεμένων λιστών και πινάκων

Οι πίνακες και οι συνδεδεμένες λίστες είναι δύο από τις πιο συχνά χρησιμοποιήσιμες δομές δεδομένων. Η γενική υπεροχή της μιας έναντι της άλλης δεν μπορεί να αποδειχτεί ενώ η επιλογή ανάμεσα στις δύο εξαρτάται από τον εκάστοτε αλγόριθμο προς υλοποίηση. Κάθε μια έχει τα πλεονεκτήματα και τα μειονεκτήματά της.

Στη συνδεδεμένη λίστα, στοιχεία μπορούν να εισάγονται και να διαγράφονται χωρίς περιορισμούς. Τα δεδομένα κάθε κόμβου της είναι αποθηκευμένα σε διαφορετικά μέρη της μνήμης και επικοινωνούν μεταξύ τους μέσω των δεικτών. Δεν υπάρχει περιορισμός στο πλήθος των στοιχείων που μπορούν να εισαχθούν σ' αυτήν, εκτός από το φυσικό μέγεθος της μνήμης. Αντίθετα σε ένα πίνακα το πλήθος των θέσεών του πρέπει να καθοριστεί από την αρχή. Για το λόγο αυτό ένας πίνακας μπορεί μετά από εισαγωγές στοιχείων να γεμίσει και να χρειαστεί να αυξήσει τις θέσεις του, μια διαδικασία χρονοβόρα και δύσκολη που καταπονεί πολύ τη μνήμη του υπολογιστή. Από την άλλη μεριά ένας πίνακας, από τον οποίο έχουν διαγραφεί αρκετά στοιχεία ή περιέχει πολλές κενές θέσεις, γίνεται αρκετά σπάταλος. Καταλαμβάνει μνήμη χωρίς αυτή να χρειάζεται.

Περισσότερη μνήμη μπορεί να εξοικονομηθεί αν σε περιπτώσεις που είναι δυνατόν, διαφορετικές λίστες μοιράζονται τα ίδια δεδομένα. Στην περίπτωση των πινάκων κάθε δεδομένο θα χρειαζόταν να επαναλαμβάνεται στη μνήμη τόσες φορές όσες είναι αποθηκευμένο στους πίνακες. Με τις συνδεδεμένες λίστες όμως το κάθε δεδομένο μπορεί να είναι αποθηκευμένο μία μόνο φορά στη μνήμη του υπολογιστή και να υπάρχουν σ' αυτό περισσότερες της μιας αναφορές.

Από την άλλη πλευρά, οι πίνακες επιτρέπουν τυχαία προσπέλαση των δεδομένων τους, ενώ οι συνδεδεμένες λίστες επιτρέπουν μόνο σειριακή προσπέλαση. Ειδικά οι απλά συνδεδεμένες λίστες επιτρέπουν διάσχιση μόνο προς τη μία κατεύθυνση. Έτσι αν είναι επιθυμητή η εύρεση ενός στοιχείου, που βρίσκεται σε συγκεκριμένη θέση, στον πίνακα αυτό μπορεί να προσπελαστεί άμεσα, ενώ στη συνδεδεμένη λίστα η προσπέλασή του απαιτεί χρόνο στην τάξη του O(n), όπου n είναι το πλήθος των στοιχείων της λίστας. Το γεγονός αυτό κάνει τις συνδεδεμένες λίστες μη κατάλληλες για εφαρμογές, στις οποίες γίνεται πολύ συχνά αναζήτηση ενός στοιχείου μέσω της ακριβής του θέσης. Ένα παράδειγμα τέτοιας διεργασίας είναι η ταξινόμηση σωρού.

Επιπλέον ακόμη και η σειριακή προσπέλαση των στοιχείων, παρόλο που είναι εφικτή και στις δύο δομές δεδομένων, στους πίνακες είναι ταχύτερη από ότι στις συνδεδεμένες λίστες. Αυτό συμβαίνει επειδή οι πίνακες κάνουν χρήση της τοπικότητας της μνήμης και εκμεταλλεύονται τα πλεονεκτήματα της κρυφής μνήμης (cache memory) του υπολογιστή. Οι συνδεδεμένες λίστες δεν έχουν κανένα όφελος από την κρυφή μνήμη, όσο μέγεθος και να έχει αυτή.

Ένα ακόμη μειονέκτημα των συνδεδεμένων λιστών έναντι των πινάκων είναι ο επιπλέον αποθηκευτικός χώρος που χρειάζεται για τους δείκτες των κόμβων. Το γεγονός αυτό κάνει τις λίστες μη πρακτικές και σπάταλες για μικρά σύνολα δεδομένων ή δεδομένα που καταλαμβάνουν μικρό χώρο στη μνήμη, όπως χαρακτήρες (characters) και λογικές τιμές (boolean).

Ένα παράδειγμα το οποίο τονίζει τα πλεονεκτήματα και τα μειονεκτήματα που προκύπτουν από τη σύγκριση των συνδεδεμένων λιστών και των πινάκων είναι το πρόβλημα του Josephus. Το πρόβλημα αυτό περιγράφει μια μέθοδο εκλογής ενός ατόμου από ένα σύνολο ατόμων που σχηματίζουν έναν κύκλο. Ξεκινώντας από ένα προκαθορισμένο άτομο, γίνεται μέτρηση προς τη μια κατεύθυνση n ατόμων. Όταν βρεθεί το n-οστό άτομο κατά τη μέτρηση, αυτό βγαίνει από τον κύκλο και οι υπόλοιποι που έμειναν κλείνουν τον κύκλο, ώστε να καλυφθεί το κενό που δημιουργήθηκε. Στη συνέχεια και πάλι το n-οστό άτομο επιλέγεται και βγαίνει από τον κύκλο. Η διαδικασία επαναλαμβάνεται έως να παραμείνει ένα μόνο άτομο στον κύκλο. Το άτομο αυτό είναι και ο νικητής.

Για την αναπαράσταση αυτού του προβλήματος μπορεί να χρησιμοποιηθεί είτε ένας πίνακας, είτε μια συνδεδεμένη λίστα. Κάθε στοιχείο της δομής δεδομένων, όποια κι αν χρησιμοποιηθεί, αντιπροσωπεύει ένα άτομο στον κύκλο. Αν τα άτομα αναπαρασταθούν με κόμβους μιας διπλά συνδεδεμένης κυκλικής λίστας, είναι πολύ εύκολη και γρήγορη η εξαγωγή ενός ατόμου, αφού η διαγραφή ενός κόμβου στη λίστα απαιτεί σταθερό χρόνο, αλλάζοντας μόνο τους δείκτες των διπλανών του κόμβων. Ωστόσο η συνδεδεμένη λίστα είναι ανεπαρκής στη διαδικασία εύρεσης του επόμενου ατόμου προς διαγραφή. Είναι απαραίτητη η διάσχιση όλων των κόμβων από αυτόν που ξεκινάει η μέτρηση μέχρι αυτόν στον οποίο θα σταματήσει. Από την άλλη μεριά, η χρήση πίνακα οδηγεί σε χρονοβόρα διαδικασία για τη διαγραφή ενός ατόμου – στοιχείου του πίνακα, αφού κάθε φορά θα πρέπει να μετακινούνται όλα τα επόμενα στοιχεία κατά μία θέση, ώστε να μην υπάρχουν κενά στον πίνακα. Ωστόσο στην περίπτωση αυτή είναι εξαιρετικά εύκολη η εύρεση του n-οστού ατόμου στον κύκλο απλά υπολογίζοντας αριθμητικά τη θέση του μέσα στον πίνακα.

Υλοποίηση συνδεδεμένων λιστών με πίνακες

Υπάρχουν γλώσσες προγραμματισμού, οι οποίες δε υποστηρίζουν κανέναν τύπο αναφοράς (δείκτη) ή άμεση προσπέλαση της μνήμης μέσω της διεύθυνσης μιας λέξης μνήμης. Ακόμη και σ' αυτές τις γλώσσες όμως είναι δυνατή η δημιουργία συνδεδεμένων λιστών, αντικαθιστώντας σ' αυτές τους δείκτες των κόμβων με αριθμούς θέσεων σε έναν πίνακα.

Συγκεκριμένα για την υλοποίηση της λίστας χρησιμοποιείται ένας πίνακας εγγραφών. Κάθε εγγραφή περιέχει όλα τα στοιχεία που είναι αποθηκευμένα σε έναν κόμβο της λίστας και επιπλέον δύο πεδία. Έτσι κάθε εγγραφή στον πίνακα είναι ακριβώς ότι ένας κόμβος στη λίστα. Τα δύο επιπλέον πεδία περιέχουν ακέραιους αριθμούς, οι οποίοι είναι οι θέσεις μέσα στον πίνακα της επόμενης και της προηγούμενης εγγραφής στον πίνακα. Με τον τρόπο αυτό μπορεί να θεωρηθεί ότι κάθε εγγραφή "δείχνει" στην επόμενη και την προηγούμενή της, όπως αυτές ορίζονται από τη σειρά των κόμβων στη λίστα.

Δεν είναι απαραίτητο να χρησιμοποιούνται όλες οι θέσεις του πίνακα. Αντιθέτως, επιθυμητό είναι πάντα να υπάρχουν κενές θέσεις στον πίνακα, ώστε να είναι εύκολη η εισαγωγή μιας νέας εγγραφής – κόμβου. Αν κάτι τέτοιο δε συμβαίνει και ο πίνακας έχει γεμίσει είναι απαραίτητο είτε να αυξήσει τις θέσεις του, διαδικασία χρονοβόρα, όπως αναφέρθηκε και προηγουμένως, είτε κάποια στοιχεία να διαγραφούν πριν εισαχθούν καινούρια.

Η εγγραφή που αντιπροσωπεύει το κάθε στοιχείο του πίνακα μπορεί να περιέχει οτιδήποτε δεδομένα είναι απαραίτητα, αλλά σίγουρα θα περιέχει και τους δύο ακέραιους – δείκτες.

```
record Node
{
    integer next        // η θέση στον πίνακα της επόμενης εγγραφής
    integer prev        // η θέση στον πίνακα της προηγούμενης εγγραφής
    data                // τα δεδομένα της εγγραφής
}
```

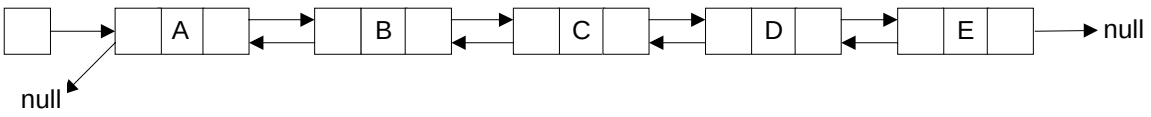
Η συνδεδεμένη λίστα, μπορεί να αναπαρασταθεί από έναν πίνακα εγγραφών αυτού του είδους και μια ακέραια μεταβλητή η οποία περιέχει τη θέση του πίνακα στην οποία βρίσκεται το πρώτο στοιχείο της λίστας. Ο ακέραιος αυτός είναι το αντίστοιχο του δείκτη αρχής.

```
record List
{
    record Node array[1000]     // ο πίνακας που υλοποιεί τη λίστα
    integer first               // ο δείκτης αρχής
}
```

Ένα παράδειγμα ενός τέτοιου πίνακα, καθώς και η σχηματική αναπαράσταση της αντίστοιχης συνδεδεμένης λίστας φαίνονται παρακάτω.

index
0
1
2
3
4
5
6
7

data	next	prev
D	5	4
B	4	3
A	1	-1
C	0	1
E	-1	0



Η μεταβλητή που αναπαριστά τον δείκτη αρχής της λίστας στο παράδειγμά έχει την τιμή 3. Η εγγραφή στη θέση 3 του πίνακα είναι αυτή που αντιστοιχεί στον πρώτο κόμβο της λίστας. Οι θέσεις 2, 6 και 7 του πίνακα είναι κενές και δε συμμετέχουν στην αναπαράσταση της λίστας, αλλά είναι έτοιμες για την εισαγωγή ενός νέου στοιχείου. Μπορεί να χρησιμοποιηθεί ένας επιπλέον πίνακας, ο οποίος να διατηρεί τους αριθμούς των κενών θέσεων του πίνακα – λίστα.

Η παρακάτω διαδικασία διατρέχει ολόκληρη τη λίστα και επεξεργάζεται τα δεδομένα της.

```
διαδικασία traversal (List list)
{
    i =  list.first                 // βοηθητική μεταβλητή για τη διάσχιση της
    while i >= 0                    // όσο δεν έχει φτάσει στο τέλος της λίστας
    {
        node data processing        // επεξεργάζεται τα δεδομένα του τρέχοντος κόμβου
        i = array[i].next           // προχωράει στον επόμενο κόμβο
    }
}
```

Τα πλεονεκτήματα αυτής της προσέγγισης περιλαμβάνουν τα εξής:

- Όλα τα στοιχεία της λίστας βρίσκονται στην ίδια περιοχή μνήμης και επιπλέον σε διαδοχικές της θέσεις, αντίθετα με την περίπτωση της συνδεδεμένης λίστας, όπου κάθε κόμβος της μπορεί να βρίσκεται οπουδήποτε στη μνήμη. Η τοπικότητα αυτή οδηγεί στη εύκολη μετακίνηση ολόκληρης της λίστας μέσα στη μνήμη, σε άμεση υλοποίηση της σειριακής μορφής που είναι απαραίτητη για να αποθηκευτεί στη δευτερεύουσα μνήμη και στην ταχεία μεταφορά της μέσω δικτύου.
- Ειδικά για μικρού μεγέθους λίστες, η υλοποίηση μέσω πίνακα, είναι καλύτερη λύση επειδή σε πολλές αρχιτεκτονικές υπολογιστών οι ακέραιοι που δείχνουν τις θέσεις των εγγραφών μέσα στον πίνακα καταλαμβάνουν σημαντικά μικρότερο χώρο από τους αντίστοιχους δείκτες της λίστας.
- Η τοπικότητα των στοιχείων του πίνακα στη μνήμη οδηγεί στο μεγάλο πλεονέκτημα της εκμετάλλευσης της κρυφής μνήμης (cache memory) του υπολογιστή. Το γεγονός αυτό μπορεί να οδηγήσει σε πολύ ταχύτερες διαδικασίες και υλοποιήσεις. Η τοπικότητα των εγγραφών μπορεί να βελτιώνεται συνέχεια κατά τη διάρκεια επεξεργασίας της λίστας, κρατώντας τους διαδοχικούς κόμβους σε κοντινές θέσεις στον πίνακα και κάνοντας περιοδική αναδιάταξη των εγγραφών όποτε αυτή κρίνεται απαραίτητη.
- Οι δρομολογητές μνήμης μπορούν να οδηγηθούν σε μεγάλη σπατάλη χώρου για την αποθήκευση κάθε κόμβου μιας συνδεδεμένης λίστας, ειδικά αν το μέγεθος αυτού δεν είναι κοντά στο μέγεθος της λέξης της μνήμης ή κάποιου πολλαπλάσιού της. Αντίθετα η προσέγγιση με πίνακα δεν επιφέρει καθόλου σπατάλη μνήμης για κάθε κόμβο.
- Η δέσμευση χώρου μνήμης a-priory για την αποθήκευση του πίνακα οδηγεί σε γρηγορότερη λειτουργία των δυναμικών δρομολογητών μνήμης. Στην περίπτωση μιας συνδεδεμένης λίστας με αναφορές είναι απαραίτητη η εύρεση κενού τμήματος μνήμης κατάλληλου μεγέθους για κάθε εισαγωγή νέου κόμβου.

Η προσέγγιση της υλοποίησης της συνδεδεμένης λίστας με πίνακα έχει ένα βασικό μειονέκτημα.

- Το μέγεθος του πίνακα πρέπει να οριστεί στην αρχή, κατά τη δημιουργία του, πριν ακόμη γίνει γνωστό το πλήθος των στοιχείων της λίστας.

Το μειονέκτημα αυτό οδηγεί και σε άλλα εξίσου σημαντικά.

- Η πολυπλοκότητα της υλοποίησης της λίστας με τη βοήθεια του πίνακα είναι μεγαλύτερη.
- Η αύξηση των θέσεων ενός ήδη μεγάλου πίνακα, όταν αυτός είναι γεμάτος, μπορεί να είναι δύσκολη ή ακόμη και ακατόρθωτη. Αντίθετα η εύρεση χώρου σε ολόκληρη τη μνήμη για την αποθήκευση ενός νέου κόμβου σε μια λίστα με αναφορές είναι πολύ πιο εύκολη υπόθεση για έναν δρομολογητή.
- Η εισαγωγή νέων στοιχείων σε έναν πίνακα, μπορεί περιστασιακά να χρειαστεί γραμμικό χρόνο, δηλαδή χρόνο στην τάξη του O(n). Κάτι τέτοιο μπορεί να συμβεί όταν ο πίνακας είναι σχεδόν γεμάτος και η διαδικασία εύρεσης κενής θέσης σε αυτήν χρειάζεται να ψάξει σε όλες σχεδόν τις θέσεις του.
- Σε πολλές περιπτώσεις ένας πίνακας που αναπαριστά μια συνδεδεμένη λίστα μπορεί να έχει πολλές κενές θέσεις. Κάτι τέτοιο μπορεί να συμβεί είτε επειδή εξ αρχής κατακρατήθηκαν πολύ περισσότερες θέσεις από αυτές που χρειάστηκαν τελικά, είτε επειδή κατά τη διάρκεια της επεξεργασίας της λίστας πολλοί κόμβοι της διαγράφηκαν. Και στις δύο περιπτώσεις οι θέσεις του πίνακα που μένουν κενές δεν απελευθερώνονται, με αποτέλεσμα να χρησιμοποιείται πολύ μεγαλύτερο μέρος της μνήμης από αυτό που είναι απαραίτητο. Το μέρος αυτό θα μπορούσε να διανεμηθεί σε άλλες εφαρμογές ή δεδομένα για τα οποία είναι απαραίτητο.