**Αλεξάνδρειο Τ.Ε.Ι. Θεσσαλονίκης**
**Σχολή Τεχνολογικών Εφαρμογών**
**Τμήμα Ηλεκτρονικής**
**Π.Μ.Σ Εφαρμοσμένα Ηλεκτρονικά Συστήματα**


**Μεταπτυχιακή Διπλωματική Εργασία**

# "Αλγόριθμοι βελτιστοποίησης''

**Διδασκάλου Ηλίας**

**Εισηγητής**: Παναγιώτης Τζέκης, Αναπληρωτής Καθηγητής

Θεσσαλονίκη, Απρίλιος 2019

Μεταπτυχιακή Διπλωματική Εργασία με τίτλο:
«Αλγόριθμοι Βελτιστοποίησης»
Κωδικός διπλωματικής εργασίας: 1814
Ημερομηνία ανάληψης: Σεπτέμβριος 2018
Ημερομηνία περάτωσης: Απρίλιος 2019

Postgraduate Diploma Thesis entitled:
"Optimization Algorithms"
Diploma thesis code: 1814
Date of withdrawal: September 2018
End date: April 2019

## Ευχαριστίες

Η παρούσα εργασία αποτελεί διπλωματική εργασία στα πλαίσια του μεταπτυχιακού προγράμματος (ΠΜΣ) «Εφαρμοσμένα Ηλεκτρονικά» του τμήματος Ηλεκτρονικών Μηχανικών του Α.Τ.Ε.Ι Θεσσαλονίκης.

Πρώτα από όλα θα ήθελα να ευχαριστήσω τον επιβλέποντα της εργασίας Δρ. Παναγιώτη Τζέκη, για τη βοήθεια και καθοδήγησή του στη συγκρότηση και ολοκλήρωσή της.

Προσωπικές ευχαριστίες θα ήθελα να αποδώσω στο σύνολο των καθηγητών του Μεταπτυχιακού που συνέβαλαν στην απόκτηση των απαραίτητων γνώσεων για την επιτυχή φοίτησή μου.

Και τέλος να ευχαριστήσω, το παρεάκι του «πάρκινγ του Καρφούρ» ……..

# Περίληψη

Στην σύγχρονη εποχή, η βελτιστοποίηση εμφανίζεται σχεδόν σε όλες τις πτυχές της ζωής μας. Όλα τα συστήματα με τα οποία ερχόμαστε σε επαφή καθημερινά (κοινωνικά, επιστημονικά, οικονομικά και άλλα) έχουν παραμέτρους, οι οποίες απαιτούν ρυθμίσεις, ώστε να λαμβάνουμε την καλύτερη δυνατή απόδοση των αντίστοιχων συστημάτων. Κάποια από τα προβλήματα αυτά επιλύονται σχετικά εύκολα, κάποια άλλα όμως απαιτούν ιδιαίτερη προσπάθεια και εξειδικευμένους αλγόριθμους.

Σκοπός αυτής της εργασίας είναι να βρει τις τεχνικές και τις μεθόδους που χρησιμοποιούνται στους αλγόριθμους βελτιστοποίησης, να τις παρουσιάσει σε απλή μορφή και να δει την εφαρμογή τους σε πραγματικά προβλήματα.

Αρχικά κατηγοριοποιούνται οι τεχνικές και οι μέθοδοι σε τρεις βασικές κατηγορίες ανάλογα με την λογική που χρησιμοποιούν, και κατόπιν παρουσιάζονται οι πιο γνωστές και διαδεδομένες τεχνικές.

Κατόπιν περνάμε στην παρουσίαση μεθόδων που έχουν εφαρμοσθεί σε πολλά γνωστά προβλήματα.



# Abstract

In modern times, optimization occurs in almost all aspects of our lives. All systems with which we come into contact daily (socially, scientifically, economically, etc.) have parameters that require regulation to get the best possible system performance. Some of these problems are relatively easy to solve, others require special effort and algorithms.

The purpose of this work is to find the techniques and methods used in optimization algorithms, to present them in simple form and to see their application to real problems.

Initially, the techniques and methods are categorized into three basic categories according to the logic they use, and then the most well-known and widespread techniques are presented.

Then we pass on the presentation of methods that have been applied to many known problems.

# Πίνακας περιεχομένων

# 1. Εισαγωγή

*Ξυπνάς με τον ήχο του ρολογιού σου. Ένα ρολόι που κατασκευάστηκε από μια εταιρεία, η οποία προσπάθησε να μεγιστοποιήσει το κέρδος της, καθορίζοντας τις πηγές της και την λειτουργία της με τον καλύτερο δυνατό τρόπο. Ανάβεις ένα μάτι της κουζίνας για να παρασκευάσεις καφέ, χωρίς να έχεις κατά νου, τις τιτάνιες προσπάθειες που κατέβαλαν κάποιοι άνθρωποι για να βελτιστοποιήσουν την παροχή της ηλεκτρικής σου ενέργειας. Χιλιάδες παράμετροι του δικτύου ενέργειας καθορίστηκαν με τέτοιο τρόπο ώστε να ελαχιστοποιούνται οι απώλειες του δικτύου και να μεγιστοποιηθεί το κέρδος της εταιρίας ηλεκτρισμού. Μπαίνεις στο αυτοκίνητό σου και βάζεις μπροστά την μηχανή, χωρίς να συλλογιστείς την πολυπλοκότητα αυτού του μικρού θαύματος μηχανικής. Χιλιάδες παράμετροι βελτιστοποιήθηκαν από τον κατασκευαστή ώστε να σου παραδοθεί ένα όχημα που ανταποκρίνεται στις προσδοκίες σου, από την αισθητική του οχήματος μέχρι τις στρογγυλεμένες άκρες των καθρεπτών που ελαχιστοποιούν την αντίσταση του αέρα. Καθώς βγαίνεις με το αυτοκίνητό σου στον κεντρικό δρόμο και συναντάς αυξημένη κίνηση σκέφτεσαι: "Μα δεν γινόταν να βελτιστοποιηθεί η σχεδίαση των δρόμων ώστε να χρειάζομαι λιγότερη ώρα για να πάω στην δουλειά μου;"*

*Frans Van den Bergh, Πραιτόρια 2001.*[1]

Η έννοια του αλγορίθμου (algorithm) είναι γνωστή στο φοιτητή της Πληροφορικής ήδη από αρκετά μαθήματα των πρώτων εξαμήνων σπουδών του. Η έννοια είναι κεντρική για την Πληροφορική και η μελέτη της είναι πολύ ενδιαφέρουσα, γιατί αποτελεί την πρώτη ύλη για την εμβάθυνση στα επιμέρους αντικείμενα της Θεωρητικής Πληροφορικής (όπως για παράδειγμα γλώσσες προγραμματισμού, αυτόματα συστήματα διαδικασιών, υπολογιστική πολυπλοκότητα, κρυπτογραφία), καθώς και σε άλλες γνωστικές περιοχές της Πληροφορικής, όπως στις Βάσεις Δεδομένων, τα Δίκτυα, την Επεξεργασία Εικόνας, την Τεχνητή Νοημοσύνη, στον Παγκόσμιο Ιστό και λοιπές.

Γενικότερα, ο όρος αυτός χρησιμοποιείται, για να δηλώσει ένα συγκεκριμένο σύνολο βημάτων – ενεργειών για την επίλυση προβλημάτων [2].

Η λέξη "αλγόριθμος" προέρχεται από μια μελέτη του Πέρση μαθηματικού Abu Ja'far Mohammed ibn al Khowarizmi με τίτλο "Κανόνες σύνθεσης και αναγωγές", ο οποίος έζησε περί το 825 μ.Χ. στην πόλη Χίβα του σημερινού Ουζμπεκιστάν.

Πέντε αιώνες αργότερα η μελέτη αυτή μεταφράστηκε στα λατινικά και άρχιζε με τη φράση "Algoritmi dixit..." (ο αλγόριθμος λέει...).

Η μελέτη του al Khowarizmi υπήρξε η πρώτη πραγματεία άλγεβρας (όρος που και αυτός προέρχεται από το αραβικό al-jabr = αποκατάσταση), γιατί ένας από τους σκοπούς της άλγεβρας είναι η αποκατάσταση της ισότητας μέσα σε μια εξίσωση.

Η λέξη επέζησε επί χίλια χρόνια ως σπάνιος όρος, που σήμαινε κάτι σαν "συστηματική διαδικασία αριθμητικών χειρισμών". Τη σημερινή του χρήση απόκτησε από την αρχή του

20ου αιώνα με την ανάπτυξη της ομώνυμης θεωρίας και φυσικά με την επικαιρότητα των ηλεκτρονικών υπολογιστών. Η έννοια του αλγόριθμου είναι βασική στον προγραμματισμό των Ηλεκτρονικών Υπολογιστών [3].

## 1.1 Ιστορική αναδρομή

Μολονότι η λέξη αλγόριθμος δεν ανάγεται στο απώτατο παρελθόν, εντούτοις η έννοια αυτή είχε συλληφθεί στην Αρχαία Ελλάδα. Αρκεί να θυμηθούμε το κόσκινο του Ερατοσθένη για την εύρεση πρώτων αριθμών ή τη μέθοδο του Ευκλείδη για την εύρεση του μέγιστου κοινού διαιρέτη δύο αριθμών. Αλλά μήπως κάθε ενέργεια στην καθημερινή μας ζωή δεν μπορεί να χαρακτηρισθεί ως αλγόριθμος; Για παράδειγμα, τι διαφορετικό είναι μία συνταγή μαγειρικής από έναν αλγόριθμο; [2]

### 1.1.1 Το κόσκινο του Ερατοσθένη (Ερατοσθένης ο Κυρηναίος 276-198 π.Χ.)

'Κάθε ακέραιος α ($\alpha \neq 0, \pm 1$), λέγεται **πρώτος αριθμός** ή απλώς **πρώτος**, αν οι μόνοι θετικοί διαιρέτες του είναι οι 1 και $|\alpha|$.'

Ο Ερατοσθένης επινόησε μια μέθοδο για την εύρεση της ακολουθίας των πρώτων αριθμών που είναι μικρότεροι ενός επιλεγμένου αριθμού (n), η οποία περιγράφεται ως εξής:

- n = 300 (ποιοι είναι οι πρώτοι αριθμοί που είναι μικρότεροι του 300)
- Κρατάμε τον πρώτο αριθμό (2) και διαγράφουμε όλα τα πολλαπλάσια του,
- στη συνέχεια κρατάμε τον επόμενο πρώτο αριθμό (3) και διαγράφουμε όλα τα πολλαπλάσια του,
- ο επόμενος πρώτος είναι το (5) και συνεχίζουμε με την ίδια λογική

Στην πρώτη εκατοντάδα υπάρχουν 25 πρώτοι αριθμοί, στην δεύτερη εκατοντάδα υπάρχουν 21 πρώτοι αριθμοί, στην τρίτη εκατοντάδα υπάρχουν 16 πρώτοι αριθμοί ενώ στην εκατοντάδα από 901 μέχρι και το 1000 υπάρχουν μόλις 14.

Οι πρώτοι αριθμοί μέχρι το 300 δίνονται στον πίνακα 1.

*Πίνακας 1*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1η 100άδα | 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 |
| | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 89 | 86 | 97 | |
| 2η 100άδα | 101 | 103 | 107 | 109 | 113 | 127 | 131 | 137 | 139 | 149 | 157 | 157 | 163 |
| | 167 | 173 | 179 | 181 | 191 | 193 | 197 | 199 | | | | | |
| 3η 100άδα | 211 | 223 | 227 | 229 | 233 | 239 | 241 | 251 | 257 | 263 | 269 | 271 | 277 |
| | 281 | 283 | 293 | | | | | | | | | | |

Στην πρώτη χιλιάδα υπάρχουν 168 πρώτοι αριθμοί, στην χιλιάδα από 999,901 μέχρι το 1,000,000 υπάρχουν μόνο 8 πρώτοι αριθμοί.

Ο κατάλογος των πρώτων αριθμών μας βοηθά να διαπιστώσουμε ότι καθώς οι αριθμοί μεγαλώνουν, οι πρώτοι αραιώνουν. Ποτέ όμως δεν τελειώνουν. Ποτέ η αραίωση δεν φτάνει στο μηδέν και άρα **δεν** ορίζεται ο μέγιστος πρώτος αριθμός. [4]

Στα "Στοιχεία" του Ευκλείδη αναφέρεται, ότι οι πρώτοι αριθμοί είναι άπειροι (βιβλίο IX, πρόταση 20):

*"Οι πρώτοι αριθμοί πλείους εισί παντός του προτεθέντος πλήθους πρώτων αριθμών".*[5]

### 1.1.4 Αντικειμενική συνάρτηση

Ως αντικειμενική συνάρτηση ονομάζουμε μια συνεχή και διαφορίσιμη συνάρτηση της μορφής $f(X) = \{f1(X), f2(X),..., fm(X)\}$, με $X = \{x1, x2,..., xn\}$, ένα σύνολο μεταβλητών.

Υπάρχουν περιπτώσεις όπου οι επιτρεπτές τιμές των παραμέτρων της συνάρτησης είναι διακριτές ή ακέραιες, και άλλες όπου οι παράμετροι είναι συνεχείς μεταβλητές, αλλά και μικτά προβλήματα όπου μερικές παράμετροι παίρνουν διακριτές και οι υπόλοιπες συνεχείς τιμές [6].

Μία αντικειμενική συνάρτηση *f* μπορεί να είναι:

- Βαθμωτή ($m = 1$) ή διανυσματική ($m > 1$)
- Μονοδιάστατη ($n = 1$) ή πολυδιάστατη ($n > 1$)
- Ντετερμινιστική ή στοχαστική
- Με συνεχείς, διακριτές ή μικτές μεταβλητές
- Με περιορισμούς (ρητούς ή/και ασαφείς) ή χωρίς περιορισμούς
- Γραμμική ή μη γραμμική
- Με μοναδικό ακρότατο ή με πολλαπλά ακρότατα

### 1.1.2 Ο αλγόριθμος του Ευκλείδη (~ 323 π.Χ. ~275 π.Χ.)

Ο αλγόριθμος του Ευκλείδη είναι μια αποτελεσματική μέθοδος για τον υπολογισμό του μέγιστου κοινού διαιρέτη (ΜΚΔ) δύο ακέραιων αριθμών **x** και **y**.

- Βήμα 1: Όσο το **y** είναι διάφορο του 0, διαίρεσε το **x** με το **y**.
- Βήμα 2: Σε κάθε βήμα ανάθεσε το β στο **x** και το υπόλοιπο της διαίρεσης του **x** / **y** στο **y**
- Βήμα 3: Όταν το **y** γίνει 0, τότε επέστρεψε τον **x** σαν Μέγιστο Κοινό Διαιρέτη [7]

Για να επιτύχουμε την «ακριβή περιγραφή» ενός αλγόριθμου, χρησιμοποιούμε γλώσσα που μπορεί να περιγράφει τη σειρά των ενεργειών που θα εκτελέσουμε με τρόπο αυστηρό, χωρίς ασάφειες και διφορούμενα. Τέτοιες γλώσσες είναι οι γλώσσες προγραμματισμού, τα μαθηματικά μοντέλα, κάποιες συμβολικές γλώσσες που χρησιμοποιούν αυστηρά καθορισμένους κανόνες περιγραφής, τα διαγράμματα ροής καθώς και κατάλληλα διαμορφωμένα υποσύνολα των φυσικών (ομιλούμενων) γλωσσών.[3]

Αν η διατύπωση του αλγορίθμου είναι σαφής, μπορούμε να εμπιστευθούμε την εκτέλεσή του σε κάποιον ή σε ένα υπολογιστή, ο οποίος μπορεί να εκτελεί τα βήματα του αλγορίθμου χωρίς να χρειάζεται να γνωρίζει τι ακριβώς επιδιώκει ο αλγόριθμος. [3]

### 1.1.3 Μηχανή Turing

Καθοριστική για την εξέλιξη της τεχνητής νοημοσύνης, ήταν η πρόταση του Alan Turing *(1912 – 1954)*, ότι ο ανθρώπινος νους είναι ένα υπολογιστικό όργανο. Ο Turing το 1936 ανέπτυξε την ιδέα μιας απλής μηχανής -της καθολικής μηχανής Turing - η οποία μπορούσε να εκτελέσει οποιοδήποτε υπολογισμό εφόσον έχουν εκφραστεί ξεκάθαρα τα βήματα που είναι αναγκαία για να εκτελεσθεί αυτός ο υπολογισμός. Η ανακάλυψη του Turing είχε μεγάλη σημασία γιατί έδειξε πως μια απλή μηχανή που ήταν προγραμματισμένη μόνο από ένα διπλό κώδικα (μηδέν και ένα) ήταν δυνατό να εκτελέσει ένα απεριόριστο αριθμό προγραμμάτων. Με τον τρόπο αυτό κατάφερε να συνδέσει την αφηρημένη έννοια του

υπολογισμού με τις συγκεκριμένες διαδικασίες μιας μηχανής. Οι επιπτώσεις αυτών των ιδεών έγιναν γρήγορα αντικείμενο περαιτέρω επεξεργασίας από αυτούς που ενδιαφέρονταν να μελετήσουν την ανθρώπινη σκέψη και νόηση. Δηλαδή, ότι θα ήταν δυνατόν να κατασκευάσει κάποιος υπολογιστικές μηχανές οι οποίες λειτουργούν με τρόπους όμοιους με τον σκεπτόμενο άνθρωπο (Turing, 1963)[8].

Συνεχίζοντας τις ιδέες του Turing, ο John von Newmann κατασκεύασε ένα νέο είδος υπολογιστή, τον Eniac ο οποίος λειτουργούσε ακολουθώντας τις οδηγίες ενός *προγράμματος*, το οποίο ήταν αποθηκευμένο στη μνήμη του και που "έλεγε στον υπολογιστή τι να κάνει". Με τον τρόπο αυτό δεν ήταν ανάγκη για τη μηχανή να επαναπρογραμματιστεί για να αντιμετωπίσει τις ανάγκες ενός νέου έργου. Απλώς επέστρεφε στο βασικό πρόγραμμα και επέλεγε να εκτελέσει μια νέα ρουτίνα. Οι νέοι υπολογιστές δεν ήταν πια απλοί εκτελεστές αριθμητικών πράξεων αλλά πολύπλοκοι επεξεργαστές συμβόλων, οι οποίοι μπορούσαν να προγραμματιστούν για να επεξεργαστούν διάφορες ανθρώπινες συμβολικές λειτουργίες, όπως αυτές της γλώσσας και της λύσης προβλημάτων. Οι εξελίξεις αυτές οδήγησαν στην δημιουργία της Τεχνητής Νοημοσύνης, η οποία συστάθηκε ως κλάδος της Πληροφορικής (Kyburg, 1990).[8]

Η επικοινωνία του ανθρώπου με τον υπολογιστή σε φυσική γλώσσα είναι από πολλά χρόνια ένας από τους βασικούς κλάδους της Τεχνητής Νοημοσύνης. Οι χρήστες βάσεων δεδομένων, συστημάτων λογισμικού και έμπειρων συστημάτων έχουν ανάγκη από ευέλικτες διεπαφές (interfaces) που στην τελειότερη μορφή τους θα δέχονται και θα παράγουν μηνύματα σε φυσική γλώσσα χωρίς περιορισμούς. Ιστορικά ο κλάδος της Τεχνητής Νοημοσύνης που περιλαμβάνει τέτοιου είδους θέματα αποκαλείται Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing) [8].

## 1.2 Τυπικός ορισμός της έννοιας του αλγορίθμου

Αλγόριθμος είναι ένα πεπερασμένο σύνολο εντολών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, οι οποίες όταν ακολουθηθούν, επιτυγχάνεται ένα επιθυμητό αποτέλεσμα ή επιλύεται ένα συγκεκριμένο πρόβλημα. Επιπροσθέτως, μία ακολουθία εντολών, για να θεωρηθεί αλγόριθμος, πρέπει να ικανοποιεί τα παρακάτω κριτήρια: [2]

1. **Είσοδος (input).** Καμία, μία ή περισσότερες ποσότητες να δίνονται ως είσοδοι στον αλγόριθμο.
2. **Έξοδος (output).** Ο αλγόριθμος να δημιουργεί τουλάχιστον μία ποσότητα ως αποτέλεσμα.
3. **Καθοριστικότητα (definiteness).** Κάθε εντολή να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής της.
4. **Περατότητα (finiteness).** Ο αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα πεπερασμένο αριθμό βημάτων λέγεται απλώς υπολογιστική διαδικασία (computational procedure).
5. **Αποτελεσματικότητα (effectiveness).** Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι επαρκώς απλή, έτσι ώστε να μπορεί να εκτελεστεί από ένα άτομο με χρήση

χαρτιού και μολυβιού. Δεν αρκεί δηλαδή να είναι ορισμένη (κριτήριο 3), αλλά πρέπει να είναι και εκτελέσιμη

Τα βήματα δημιουργίας αλγόριθμου είναι:

- Διατύπωση του προβλήματος
- Κατανόηση του προβλήματος
- Λύση του προβλήματος
- Διατύπωση του αλγόριθμου
- Έλεγχος της λύσης

Όταν σχεδιάζουμε ένα αλγόριθμο μας ενδιαφέρει άμεσα το είδος των τύπων δεδομένων που υποστηρίζονται, ώστε να μπορούμε να εκτελέσουμε τις κατάλληλες πράξεις. Αξίζει να αναφερθεί ότι για το χειρισμό διαφορετικών τύπων δεδομένων αναπτύσσονται διαρκώς νέοι αλγόριθμοι με σκοπό τον αποτελεσματικότερο και ταχύτερο χειρισμό τους.

### 1.2.1 Παράσταση αλγορίθμων

Για την παράσταση των αλγορίθμων έχουν χρησιμοποιηθεί πλήθος μεθόδων. Οι κυριότερες είναι:

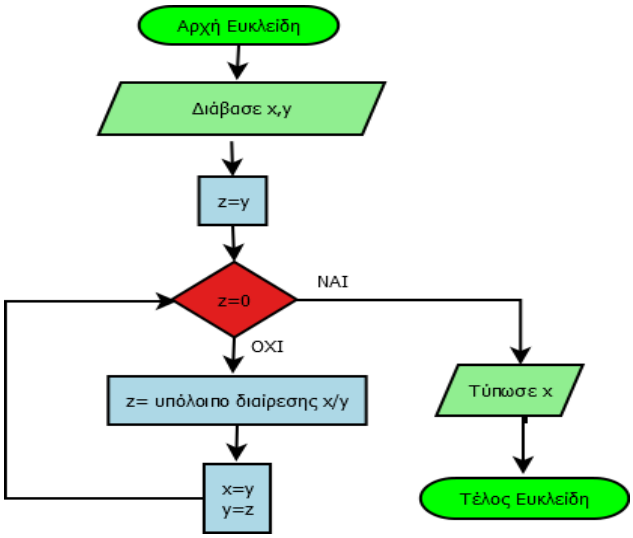
- η ψευδογλώσσα

**Αλγόριθμος 1**:Ευκλείδης (x,y)

**Είσοδος:** *Δύο ακέραιοι αριθμοί x,y >0*
**Έξοδος:** *Μέγιστος Κοινός Διαιρέτης*

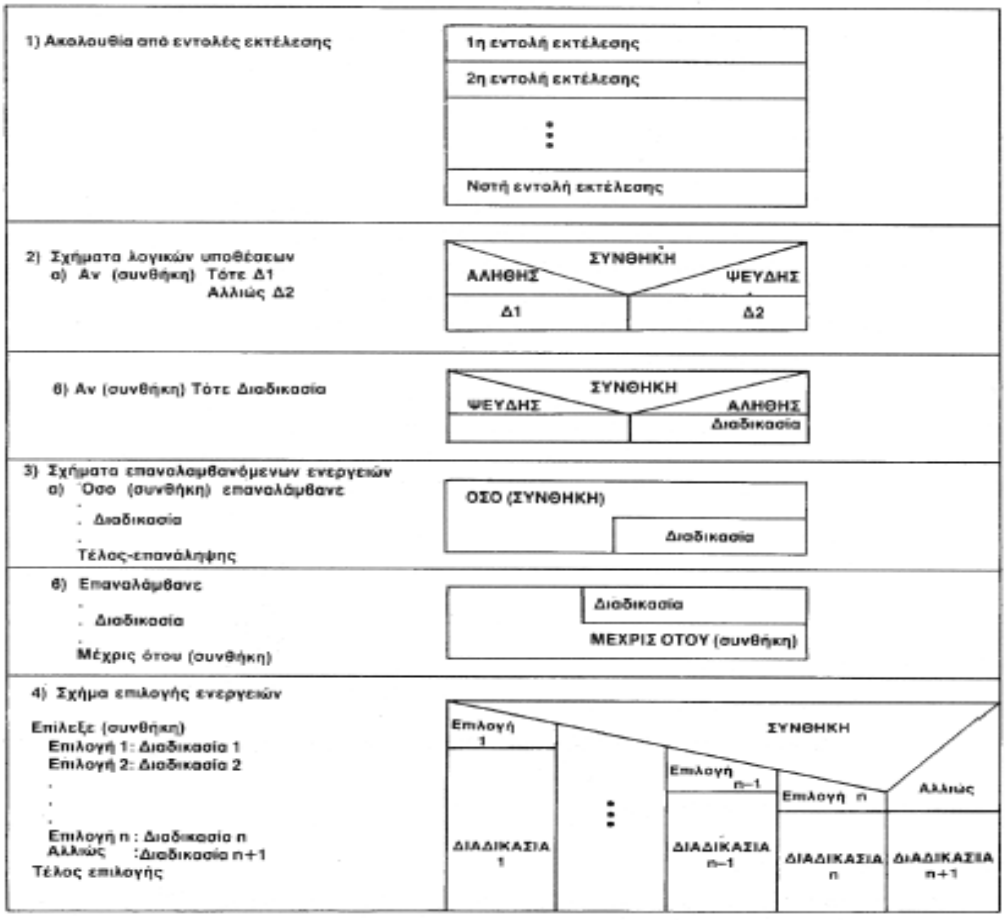
```
1:  z←y
2:  Όσο z≠ 0 επανέλαβε
3:          z←x mod y
4:          x←y
5:          y←z
6:  Τέλος επανάληψης
7:  Εμφάνισε //x//
    Τέλος Ευκλείδης
```

- τα διαγράμματα ροής




*Σχήμα 1. Διάγραμμα ροής του Ευκλείδειου Αλγόριθμου*

- και τα διαγράμματα Nassi-Shneiderman (N-S).




*Σχήμα 2. Διάγραμμα Nassi-Shneiderman*

Οι τρόποι – μέθοδοι αυτοί δεν παρουσιάζουν τα ίδια πλεονεκτήματα. Συνήθως χρησιμοποιούνται εκείνες που είναι σύμφωνες με το πνεύμα του δομημένου προγραμματισμού, ο οποίος χρησιμοποιεί τις τρεις βασικές αλγοριθμικές δομές [3]:

- της ακολουθίας,
- της επιλογής
- και της επανάληψης

Ο ψευδοκώδικας και τα διαγράμματα ροής είναι δομημένοι τρόποι έκφρασης των αλγορίθμων, που αποφεύγουν τις ασάφειες που συμβαίνουν συχνά στις δηλώσεις της φυσικής γλώσσας, ενώ παραμένουν ανεξάρτητοι από συγκεκριμένες εφαρμογές της γλώσσας προγραμματισμού. Οι γλώσσες προγραμματισμού έχουν στόχο την έκφραση των αλγορίθμων σε τέτοια μορφή, που να μπορούν να εκτελούνται από τον υπολογιστή, αλλά συχνά χρησιμοποιούνται και ως τρόποι καθορισμού ή τεκμηρίωσης των αλγορίθμων.

Οι αλγόριθμοι εφαρμόζονται ή προσομοιώνονται από προγράμματα υπολογιστή (λένε στον υπολογιστή πια ακριβώς βήματα να ακολουθήσει στην εκτέλεση), ώστε να φέρουν εις πέρας μια συγκεκριμένη εργασία – ακολουθία διαδικασιών. Επειδή ο αλγόριθμος είναι μια ακριβής λίστα από ακριβή βήματα, η σειρά των υπολογισμών είναι πάντα πολύ κρίσιμη για τη λειτουργικότητα του αλγορίθμου. Θεωρείται πάντα ότι δίδονται ρητές οδηγίες σε λίστα, και περιγράφονται, αρχίζοντας «από τη κορυφή» και πηγαίνοντας «προς τα κάτω», σαν μια ιδέα που τυπικά ονομάζεται *έλεγχος ροής (flow of control)*.

### 1.2.2 Δομές Δεδομένων

Δομή δεδομένων είναι ένα σύνολο αποθηκευμένων δεδομένων, τα οποία είναι έτσι οργανωμένα, ώστε να υπόκεινται σε συγκεκριμένες απαιτούμενες επεξεργασίες. Ο όρος αναφέρεται σε ένα σύνολο δεδομένων μαζί με ένα σύνολο λειτουργιών που επιτρέπονται στα δεδομένα αυτά. Πρέπει να αναφερθεί ότι οι δομές δεδομένων είναι πολύ στενά συνδεδεμένες με την έννοια του αλγορίθμου.

Είναι πολύ χαρακτηριστική η ακόλουθη «σχέση» που διατύπωσε ο N. Wirth, δημιουργός της γλώσσας Pascal:

***Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα***

Κάθε δομή δεδομένων αποτελείται, στην πιο γενική της μορφή, από ένα σύνολο στοιχείων ή κόμβων. Οι βασικές λειτουργίες ή πράξεις επί των δομών δεδομένων είναι οι ακόλουθες:

- **Προσπέλαση** (access), πρόσβαση σε έναν κόμβο με σκοπό να εξεταστεί ή να τροποποιηθεί το περιεχόμενό του
- **Ανάκτηση** (retrieval), η με οποιονδήποτε τρόπο λήψη (ανάγνωση) του περιεχομένου ενός κόμβου
- **Αναζήτηση** (searching) ενός συνόλου στοιχείων δεδομένων προκειμένου να εντοπιστούν ένα ή περισσότερα στοιχεία, που έχουν μια δεδομένη ιδιότητα
- **Εισαγωγή** (insertion), η προσθήκη ή δημιουργία νέων κόμβων σε μια υπάρχουσα δομή
- **Διαγραφή** (deletion) που συνιστά το αντίθετο της εισαγωγής

- **Ταξινόμηση** (sorting) όπου τα στοιχεία μιας δομής διατάσσονται κατά αύξουσα ή φθίνουσα τάξη
- **Συγχώνευση** (merging) κατά την οποία δύο ή περισσότερες δομές συνενώνονται σε μια ενιαία δομή
- **Διαχωρισμός** (separation) που αποτελεί την αντίστροφη πράξη της συγχώνευσης
- **Προσάρτηση** (append) κατά την οποία μία δομή επικολλάται στο τέλος μιας άλλης
- **Αντιγραφή** (copying) όπου κάποιοι οι κόμβοι μιας δομής αντιγράφονται σε μια άλλη

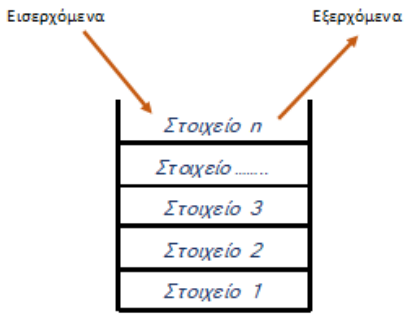
Πρέπει να αναφερθεί ότι οι ανωτέρω λειτουργίες, στην πράξη, σπάνια υπάρχουν ταυτόχρονα όλες σε μια δομή. Συνήθως παρατηρείται το φαινόμενο μια δομή δεδομένων να είναι αποδοτικότερη από μια άλλη για κάποια λειτουργία όπως η αναζήτηση, αλλά λιγότερο αποδοτική για κάποια άλλη λειτουργία, όπως η εισαγωγή δεδομένων.

Αυτό εξηγεί τόσο την ύπαρξη διαφορετικών δομών όσο και τη σπουδαιότητα της επιλογής της κατάλληλης δομής κάθε φορά. Οι πιο ευρέως χρησιμοποιούμενες δομές δεδομένων είναι ο πίνακας (table), η στοίβα (stack), η ουρά (queue), η συνδεδεμένη λίστα (linked list), το δέντρο (tree) και ο γράφος (graph).

| | | | | | |
|---|---|---|---|---|---|
| Στοιχείο 1.1 | Στοιχείο 1.2 | Στοιχείο 1.3 | Στοιχείο 1.4 | Στοιχείο ..... | Στοιχείο 1.n |
| Στοιχείο 2.1 | Στοιχείο 2.2 | Στοιχείο 2.3 | Στοιχείο 2.4 | Στοιχείο ..... | Στοιχείο 2.n |
| Στοιχείο 3.1 | Στοιχείο 3.2 | Στοιχείο 3.3 | Στοιχείο 3.4 | Στοιχείο ..... | Στοιχείο 3.n |
| Στοιχείο ..... | Στοιχείο ..... | Στοιχείο ..... | Στοιχείο ..... | Στοιχείο ..... | Στοιχείο ..... |
| Στοιχείο m.1 | Στοιχείο m.2 | Στοιχείο m.3 | Στοιχείο m.4 | Στοιχείο ..... | Στοιχείο m.n |

*Σχήμα 3. Δισδιάστατος πίνακας [m*n] στοιχείων*

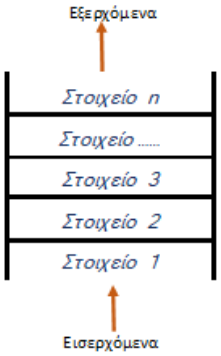
Ο Πίνακας αποτελείται από ένα σύνολο ομοειδών απλών στοιχείων, καθένα από τα οποία καθορίζεται με τη βοήθεια ενός ή περισσοτέρων δεικτών (index). Ένας πίνακας μπορεί να είναι μίας, δύο ή περισσοτέρων διαστάσεων, ανάλογα με το πλήθος δεικτών που χρειάζονται για να καθοριστεί η θέση του. Στο σχήμα 3, βλέπουμε ένα δισδιάστατο πίνακα με (m) γραμμές και (n) στήλες.




*Σχήμα 4. Διάταξη στοίβας*

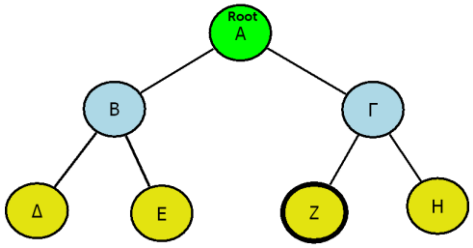
Μία στοίβα (stack, σχήμα 4) είναι μια γραμμική διάταξη στοιχείων, στην οποία εισάγονται και εξάγονται στοιχεία μόνο από το ένα άκρο. Η λειτουργία της εισαγωγής

αποκαλείται ώθηση (push) και της εξαγωγής, απώθηση (pull). Η φιλοσοφία εισαγωγής και εξαγωγής των στοιχείων ονομάζεται LIFO (Last In, First Out), δηλαδή το τελευταίο εισαγόμενο δεδομένο εξέρχεται και πρώτο.



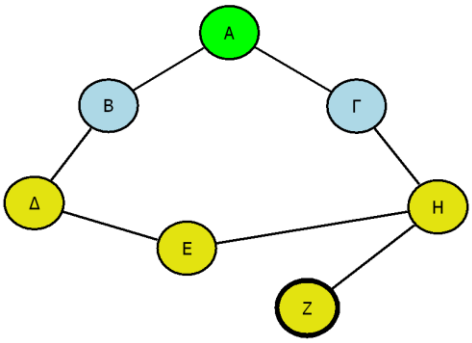

*Σχήμα 5. Διάταξη ουράς*

Μια ουρά (Queue, σχήμα 5) αποτελεί μια γραμμική διάταξη στοιχείων, στην οποία εισάγονται (enqueue) νέα στοιχεία από ένα άκρο και εξάγονται (dequeue) υπάρχοντα στοιχεία από το άλλο άκρο. Η λειτουργίας της ουράς αποκαλείται FIFO (First In, First Out), δηλαδή το στοιχείο το οποίο εισάγεται πρώτο στην ουρά εξέρχεται και πρώτο.




*Σχήμα 6. Δεντρική δομή*

Τα Δένδρα (trees, σχήμα 6) είναι μη γραμμικές δομές που αποτελούνται από ένα σύνολο κόμβων, οι οποίοι συνδέονται με ακμές. Υπάρχει μόνο ένας κόμβος, από τον οποίο μόνο ξεκινούν ακμές, που ονομάζεται ρίζα (root). Σε όλους τους άλλους κόμβους καταλήγει μία ακμή και ξεκινούν καμία, μία ή περισσότερες. Όταν ένας αλγόριθμος διασχίζει ένα δέντρο, ελέγχει ή ενημερώνει κάθε κορυφή της δομής. Στη συνέχεια, σηματοδοτεί κάθε κόμβο που έχει επισκεφθεί για να εξασφαλίσει ότι δεν θα επισκεφθεί τον ίδιο κόμβο περισσότερες από μία φορές.




*Σχήμα 7. Γράφος*

Οι Γράφοι (Graphs, σχήμα 7) αποτελούν τη πιο γενική δομή δεδομένων μια και αποτελούνται από κόμβους και ακμές χωρίς όμως κάποια ιεράρχηση. Σε ένα γράφο, μπορούμε να ξεκινήσουμε από μια κορυφή και να τελειώσουμε σε μια άλλη, ή να αρχίσουμε και να τελειώσουμε στην ίδια κορυφή. Δεδομένου ότι υπάρχουν πολλά μονοπάτια που εμπλέκονται σε ένα γράφο, υπάρχουν φορές που μπορεί να βρούμε μια διαδρομή που δεν θα μας αφήσει να διασχίσουμε τον ίδιο κόμβο ή ακμή δύο φορές.

- **Μονοπάτι Euler** ονομάζεται ένα μονοπάτι που περνάει ακριβώς μία φορά από κάθε ακμή του γραφήματος.
- **Μονοπάτι Hamilton** ονομάζεται ένα μονοπάτι που περνάει ακριβώς μία φορά από κάθε κορυφή του γραφήματος.

# 2. Συμβολισμοί Πολυπλοκότητας

Η πολυπλοκότητα ενός αλγορίθμου είναι μια ιδιαίτερα σημαντική και χαρακτηριστική παράμετρός του, και μας δίνει ένα μέτρο της χρονικής καθυστέρησης του για την επίλυση ενός προβλήματος

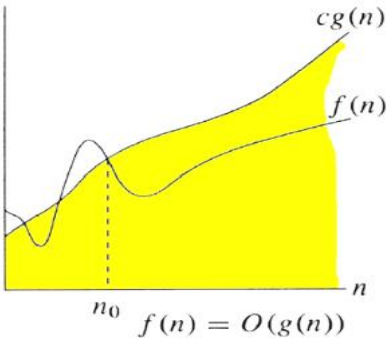
Για την πολυπλοκότητα ενός αλγορίθμου χρησιμοποιούμε το συμβολισμό O (O-notation, όπου το O προέρχεται από τη λέξη order και προφέρεται "big-oh"). Ο συμβολισμός O(*f(n)*) σημαίνει ότι όταν το n μεγαλώνει ο χρόνος εκτέλεσης του αλγορίθμου είναι ανάλογος το πολύ με την *f(n)*. [2]

Κοινός τόπος σε όλους τους συμβολισμούς, είναι η λέξη «ασυμπτωτικά». Η έννοια αυτή είναι δάνειο από τα καθαρά μαθηματικά (θεωρία αριθμών) και εμπεδώθηκε στην Πληροφορική από τον Knuth. Με τον όρο ασυμπτωτική συμπεριφορά μιας αριθμητικής συνάρτησης *f(n)* εννοούμε το πως μεταβάλλεται η τιμή της συνάρτησης για μεγάλο *n*.

## 2.1 Συμβολισμοί O, Ω, Θ.

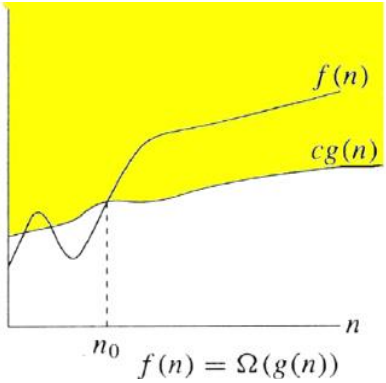
**Συμβολισμός O. Άνω φράγμα στην τάξη μεγέθους**

Μία συνάρτηση *f(n)* λέγεται ότι έχει πολυπλοκότητα της τάξης O(*g(n)*) (όπου g(n) συνάρτηση του n που μπορεί να είναι της μορφής n, logn, $n^2$ ,$n^3$ , αλλά και μονάδα) και συμβολίζεται με *f(n)* = O(*g(n)*) ή με *f(n)* ∈ O(*g(n)*), αν υπάρχει μία θετική σταθερά c και μία τιμή $n_0$, έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) < c \cdot g(n)$.




*Σχήμα 8. Γραφική αναπαράσταση του O*

**Συμβολισμός Ω. Κάτω φράγμα στην τάξη μεγέθους**

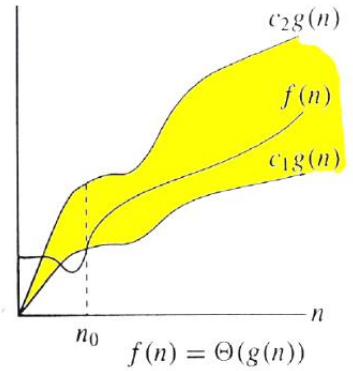



*Σχήμα 9. Γραφική αναπαράσταση του Ω*

Μία συνάρτηση *f(n)* λέγεται ότι έχει πολυπλοκότητα της τάξης Ω(*g(n)*) και συμβολίζεται με *f(n)* = Ω(*g(n)*) ή με f(n) ∈ Ω(*g(n)*), αν υπάρχει μία θετική σταθερά *c* και μία τιμή $n_0$, έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) > c \cdot g(n)$.

**Συμβολισμός Θ. Άνω και κάτω φράγμα στην τάξη μεγέθους**

Μία συνάρτηση *f(n)* λέγεται ότι έχει πολυπλοκότητα της τάξης Θ(*g(n)*) και συμβολίζεται με *f(n) = (g(n))* ή με *f(n) ∈ (g(n))*, αν υπάρχουν δύο θετικές σταθερές $c_1$ και $c_2$ και μία τιμή $n_0$, έτσι ώστε κάθε για $n > n_0$ να ισχύει η σχέση $c_2 \cdot g(n) < f(n) < c_1 \cdot g(n)$.




*Σχήμα 10. Γραφική αναπαράσταση του Θ*

Με απλά λόγια, χρησιμοποιούμε το συμβολισμό Ο και το συμβολισμό Ω για να δηλώσουμε ότι η επίδοση ενός αλγορίθμου είναι ασυμπτωτικά φραγμένη από επάνω και από κάτω αντίστοιχα. Με το συμβολισμό Θ δηλώνουμε ότι η επίδοση ενός αλγορίθμου είναι ασυμπτωτικά φραγμένη από επάνω και από κάτω ταυτόχρονα. Οι συμβολισμοί Ο, Ω και Θ μπορεί να είναι περισσότερο ή λιγότερο περιοριστικοί ή σφικτοί (tight).

**Συμβολισμός ο. Άνω φράγμα στην τάξη μεγέθους που δεν είναι ακριβές.**

Μία συνάρτηση *f(n)* λέγεται ότι έχει πολυπλοκότητα της τάξης ο(*g(n)*) και συμβολίζεται με *f(n) = o(g(n))* ή με *f(n) ∈ o(g(n))*, αν για κάθε θετική σταθερά *c > 0* υπάρχει μία τιμή $n_0$, έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) < c \cdot g(n)$.

**Συμβολισμός ω. Κάτω στην τάξη μεγέθους που δεν είναι ακριβές.**

Μία συνάρτηση *f(n)* λέγεται ότι έχει πολυπλοκότητα της τάξης ω(*g(n)*) και συμβολίζεται με *f(n) = ω(g(n))* ή με *f(n) ∈ ω(g(n))*, αν για κάθε θετική σταθερά *c > 0* υπάρχει μία τιμή $n_0$, έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) > c \cdot g(n)$.

## 2.2 Ιδιότητες πολυπλοκότητας

Για τους συμβολισμούς της πολυπλοκότητας, ισχύουν πολλές από τις ιδιότητες των πραγματικών αριθμών.

- Μεταβατική ιδιότητα. (transitivity)
  *f(n)=O(g(n)) και g(n)=O(h(n)))f(n)=O(h(n))*
  *f(n)=Ω(g(n)) και g(n)=Ω(h(n)) )f(n)=Ω(h(n))*
  *f(n)=Θ(g(n)) και g(n)=Θ(h(n)) )f(n)=Θ(h(n))*
  *f(n)=o(g(n)) και g(n)=o(h(n))) f(n)=o(h(n))*
  *f(n)=ω(g(n)) και g(n)=ω(h(n))) f(n)=ω(h(n))*
- Ανακλαστική ιδιότητα. (reflexivity)
  *f(n) = O(f(n))*
  *f(n) = Ω(f(n))*
  *f(n) = Θ(f(n))*
- Συμμετρική ιδιότητα. (symmetry)
  *f(n) = Θ(g(n)) αν και μόνο αν g(n) = Θ(f(n))*
- Ανάστροφη Συμμετρική ιδιότητα. (transpose symmetry)
  *f(n) = O(g(n)) αν και μόνο αν g(n) = Ω(f(n))*
  *f(n) = o(g(n)) αν και μόνο αν g(n) = ω(f(n))*

## 2.3 Lagrangian μέθοδος χαλάρωσης

Στα μέσα της δεκαετίας του '70 ξεκίνησε και αναπτύχθηκε ως μέθοδος επίλυσης προβλημάτων βελτιστοποίησης, η ‘Χαλάρωση κατά Lagrange’. Η ιδέα αυτή βασίστηκε στην παρατήρηση ότι πολλά δύσκολα προβλήματα ακεραίου προγραμματισμού είναι δυνατόν να ανασυντεθούν, με την μορφή σχετικά εύκολων προβλημάτων και τα οποία συνδέονται από ομάδα περιορισμών.

Η χαλάρωση ενός προβλήματος ακεραίου προγραμματισμού κατά Lagrange, εκμεταλλεύεται την δομή του δημιουργώντας μια πιο 'εύκολη' σύνθεση. Αυτό επιτυγχάνεται με την εισαγωγή των 'περίπλοκων' περιορισμών, στην αντικειμενική συνάρτηση που περιγράφει το πρόβλημα, αφού έχουν πολλαπλασιασθεί με κάποιους συντελεστές 'ποινής' λ.

Έτσι κατασκευάζεται ένα 'Lagrangean' πρόβλημα το οποίο είναι πολύ πιο εύκολο να λυθεί, σε σχέση πάντα με το αρχικό, και του οποίου η βέλτιστη τιμή αποτελεί ένα κατώτερο όριο (για προβλήματα ελαχιστοποίησης) της βέλτιστης τιμής του αρχικού προβλήματος.

Η 'γέννηση' της μεθόδου αυτής με την σημερινή μορφή της αποδίδεται στους Held and Karp (1970), οι οποίοι χρησιμοποίησαν ένα Lagrangian' πρόβλημα για την κατασκευή ενός τρομακτικά επιτυχημένου αλγόριθμου για το 'Πρόβλημα του Περιοδεύοντος Πωλητή'.

Τρία κυρίως ερωτήματα γεννώνται κατά την εφαρμογή της μεθόδου.

- Ποιοι περιορισμοί θα χαλαρωθούν,
- Πως να υπολογισθούν οι πολλαπλασιαστές λ,
- Πως να παραχθεί μια καλή εφικτή λύση στο αρχικό πρόβλημα, από μια λύση του χαλαρωμένου προβλήματος.

Η απάντηση στο πρώτο ερώτημα είναι ότι η χαλάρωση πρέπει να κάνει το ‘χαλαρωμένο’ πρόβλημα πολύ πιο εύκολο στη λύση του.

Για τον προσδιορισμό των εκάστοτε τιμών λ χρησιμοποιείται η μέθοδος 'υποβαθμωτής βελτιστοποίησης' (sub-gradient optimization method). Κατά την εφαρμογή της μεθόδου αυτής υπεισέρχονται διάφοροι παράμετροι οι οποίοι μελετώνται και εξετάζονται τόσο ως προς τις τιμές που μπορεί να λάβουν, όσο και ως προς τους διάφορους τρόπους υπολογισμού των. Επίσης δοκιμάζονται εναλλακτικοί τύποι προσέγγισης της μεθόδου αυτής και ο καλύτερος εφαρμόζεται σε δύο διαφορετικές χαλαρώσεις του προβλήματος. Οι δύο αυτοί τύποι χαλαρώσεως δοκιμάζονται σε ένα αριθμό προβλημάτων και επιλέγεται αυτός που δίνει τα καλλίτερα αποτελέσματα.

Στη συνέχεια η χαλάρωση που έχει επιλεχθεί ενσωματώνεται στον αλγόριθμο που θα κατασκευαστεί. Δεν είναι σίγουρο ότι είναι δυνατόν να βρεθούν τέτοια λ, ώστε να ικανοποιούνται οι συνθήκες βελτιστοποίησης, και έτσι να βρίσκεται η βέλτιστη λύση του προβλήματος ,επιλύοντας το χαλαρωμένο πρόβλημα.

Τέλος για το τρίτο ερώτημα είναι προφανές ότι για την εύρεση μιας λύσης εξαρτάται αποκλειστικά από την δομή του προβλήματος.[9]

# 3. Κατηγοριοποίηση Αλγορίθμων

Ένας αλγόριθμος είναι μια βήμα προς βήμα διαδικασία για την εκτέλεση μιας εργασίας σε πεπερασμένο χρόνο, ενώ δομή δεδομένων είναι ένας συστηματικός τρόπος οργάνωσης και προσπέλασης δεδομένων. Η ανάλυση των αλγορίθμων γίνεται με βάση τον χρόνο εκτέλεσής τους, ενώ η ανάλυση των δομών δεδομένων γίνεται με βάση τον αποθηκευτικό χώρο που καταλαμβάνουν.

Κατά την ανάλυση των αλγορίθμων επιθυμούμε να υπάρχει ένα πλαίσιο που να μην εξαρτά τα αποτελέσματα από το υλικό ή το λογισμικό ενός υπολογιστή και να παράγει εκτιμήσεις χρόνου εκτέλεσης για οποιοδήποτε μέγεθος εισόδου. Συνηθισμένες μορφές ανάλυσης αλγορίθμων είναι η ανάλυση χειρότερης και η ανάλυση μέσης περίπτωσης, με την πρώτη να θεωρείται ευκολότερη διότι, δεν απαιτεί προεργασία σχετικά με την πιθανότητα εμφάνισης κάθε εισόδου.

## 3.1 Με βάση την πολυπλοκότητα χρόνου

Αφού λοιπόν ορίσθηκαν και τυπικά οι συμβολισμοί της πολυπλοκότητας, είναι δυνατόν να κατηγοριοποιήσουμε τους διαφόρους αλγορίθμους σε μία από τις επόμενες κατηγορίες:

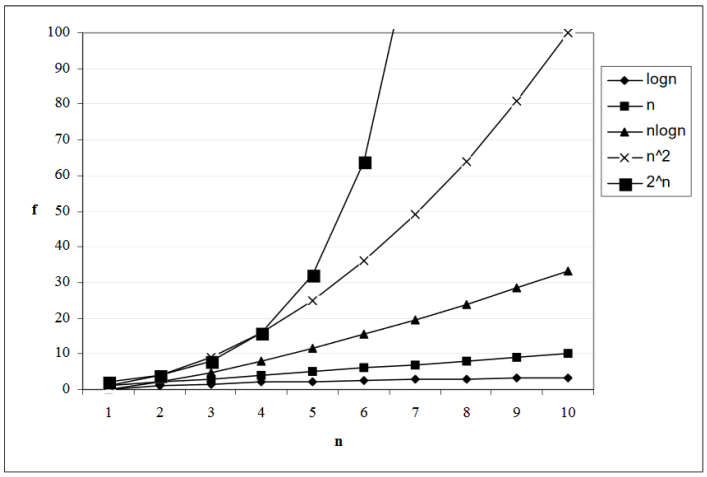
- **O(1).** Κάθε εντολή εκτελείται σε πεπερασμένο πλήθος φορών, οπότε λέγεται ότι ο αλγόριθμος είναι «σταθερής πολυπλοκότητας». Συνήθως αναφέρεται σε πράξεις εύρεσης ενός στοιχείου ή ενημέρωσης ενός συνόλου στοιχείων.
- **O(log n).** Ο αλγόριθμος είναι «λογαριθμικής πολυπλοκότητας». Με «log» και με «ln» θα συμβολίζεται ο δυαδικός και ο φυσικός λογάριθμος, αντιστοίχως. Πρακτικά, οι λογάριθμοι που θα χρησιμοποιηθούν είναι κυρίως δυαδικοί. Συνήθως αναφέρεται σε πράξεις εύρεσης ενός στοιχείου ή ενημέρωσης ενός συνόλου στοιχείων.
- **O(n).** Η πολυπλοκότητα λέγεται «γραμμική». Γενικώς, αυτή είναι η επίδοση ενός αλγορίθμου που πρέπει να εξετάσει ή να δώσει στην έξοδο n στοιχεία.
- **O(nlogn).** Διαβάζεται όπως ακριβώς γράφεται (δηλαδή «n log n»). Στην κατηγορία αυτή ανήκουν πολλοί αλγόριθμοι ταξινόμησης.
- **O($n^2$).** Αναφέρεται ως «τετραγωνική πολυπλοκότητα». Στην κατηγορία αυτή ανήκει ο δυναμικός προγραμματισμός για απλά προβλήματα, όπως αντιστοίχιση ακολουθιών.
- **O($n^3$).** Αναφέρεται ως «κυβική πολυπλοκότητα». Οι αλγόριθμοι αυτοί πρέπει να χρησιμοποιούνται μόνο για προβλήματα μικρού μεγέθους. Χαρακτηριστικό παράδειγμα είναι ο γραμμικός προγραμματισμός.
- **O($2^n$).** Σπάνια στην πράξη χρησιμοποιούνται αλγόριθμοι «εκθετικής πολυπλοκότητας», ενώ πάρα πολλά γνωστά προβλήματα έχουν αυτή την επίδοση. Ένα από τα πιο γνωστά είναι το πρόβλημα του περιοδεύοντος πωλητή.

Στον πίνακα 2 υπολογίζεται ο χρόνος που απαιτείται από αλγορίθμους διαφόρων πολυωνυμικών και εκθετικών πολυπλοκοτήτων ως συνάρτηση του μεγέθους του προβλήματος. Για να γίνουν οι χρονικές εκτιμήσεις υποτίθεται ότι κάθε στοιχειώδης πράξη απαιτεί ένα msec στη CPU. Έτσι, αν για έναν αλγόριθμο τάξης $O(n^3)$, διπλασιάζεται το μέγεθος του προβλήματος, τότε απαιτείται οκταπλάσιος ($2^3$) χρόνος για να περατωθεί ο αλγόριθμος.

*Πίνακας 2*[2]

| *Πολυπλοκότητα* | *n=20* | *n=40* | *n=60* |
|---|---|---|---|
| *O(n)* | *0,00002 sec* | *0,00004 sec* | *0,00006 sec* |
| *$O(n^2)$* | *0,0004 sec* | *0,0016 sec* | *0,0036 sec* |
| *$O(n^3)$* | *0,008 sec* | *0,064 sec* | *0,216 sec* |
| *$O(2^n)$* | *1 sec* | *12,7 ημέρες* | *366 αιώνες* |
| *O(n!)* | *771 αιώνες* | *$3*10^{32}$ αιώνες* | *$3*10^{66}$ αιώνες* |

Στο επόμενο σχήμα (σχ. 11) φαίνεται ο ρυθμός αύξησης της πολυπλοκότητας των συναρτήσεων *f(n)=logn, f(n)=n, f(n)=nlogn, f(n)=$n^2$ και f(n)=$2^n$.*




*Σχήμα 11. Ρυθμός αύξησης πολυπλοκότητας μια συνάρτησης* [10]

Πολυωνυμικοί (polynomial) λέγονται οι αλγόριθμοι με πολυπλοκότητα που φράσσεται άνω από μία πολυωνυμική έκφραση. Για παράδειγμα, πολυωνυμικοί είναι οι αλγόριθμοι τάξης O(n), $O(n^{3/2})$, $O(n^2)$. Συνήθως δεν απαιτούν μεγάλη υπολογιστική προσπάθεια σε αντίθεση με τους αλγορίθμους πολυπλοκότητας τάξης O(n!), $O(2^n)$ ή $O(n^n)$, που ονομάζονται μη πολυωνυμικοί (non-polynomial).

Διαπιστώνεται αμέσως ότι οι αλγόριθμοι εκθετικής πολυπλοκότητας δεν είναι πρακτικής χρησιμότητας ακόμη και για μικρό αριθμό δεδομένων. Επειδή η βελτίωση των αλγορίθμων είναι ζωτικής σημασίας, μία ιδιαίτερη περιοχή της Πληροφορικής, η Υπολογιστική Πολυπλοκότητα (Computational Complexity), διερευνά από θεωρητική και αναλυτική άποψη τους αλγορίθμους και τους ταξινομεί αναλόγως με την επίδοσή τους.

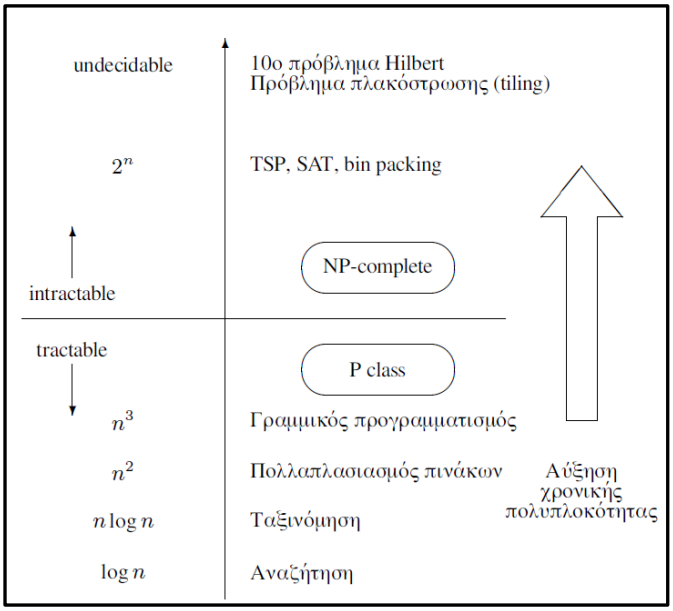
Συνεχώς αναπτύσσονται νέοι καλύτεροι αλγόριθμοι, ενώ άλλοι περιθωριοποιούνται ως μη αποτελεσματικοί. Ένας αλγόριθμος λέγεται βέλτιστος (optimal), αν αποδειχθεί ότι δεν μπορεί να κατασκευασθεί καλύτερος.

Είναι αξιοσημείωτο ότι για το πρόβλημα της ταξινόμησης ενός πίνακα δεν έχει βρεθεί μέχρι σήμερα ταχύτερος αλγόριθμος από τον Quicksort του R. Hoare [11], ενώ για το πρόβλημα του υπολογισμού του ΜΚΔ δεν έχει βρεθεί ταχύτερος αλγόριθμος από εκείνον του Ευκλείδη [3].

## 3.2 P or NP

Πέρα από το ερώτημα, του αν για ένα πρόβλημα υπάρχει Μηχανή Turing, που το επιλύει, μας απασχολεί επίσης και το ερώτημα του αν ένα πρόβλημα είναι «πρακτικά» επιλύσιμο. Το τελευταίο αυτό ερώτημα σχετίζεται με το γεγονός ότι τα προγράμματα που σχεδιάζουμε, έχουν συγκεκριμένες απαιτήσεις σε πόρους χρόνου και χώρου. [2]

Συχνά λοιπόν βρισκόμαστε αντιμέτωποι με προβλήματα, που είναι πρακτικά μη επιλύσιμα, καθώς για μεγάλες εισόδους οι απαιτήσεις τους σε πόρους αυξάνονται τόσο γρήγορα, που γίνονται μη ρεαλιστικές. Γενικά θεωρούμε ότι, αν οι απαιτήσεις σε πόρους αυξάνονται με ρυθμό πολυωνυμικής συνάρτησης, ως προς το μέγεθος του προβλήματος, τότε το πρόβλημα είναι «πρακτικά» επιλύσιμο. Αν οι απαιτήσεις αυξάνονται με μεγαλύτερο ρυθμό, τότε μπορεί να επιλυθούν μόνο περιπτώσεις του προβλήματος με πολύ περιορισμένο μέγεθος εισόδου, ενώ στη γενική περίπτωση το πρόβλημα είναι πρακτικά μη επιλύσιμο.




*Σχήμα 12. Φάσμα υπολογιστικής πολυπλοκότητας*

Η θεωρία πολυπλοκότητας ταξινομεί τα προβλήματα σε κλάσεις (σύνολα) ισοδυναμίας που ορίζουν ότι τα προβλήματα στην ίδια κλάση έχουν την ίδια δυσκολία. Τα προβλήματα διαχωρίζονται ανάλογα με την κλάση πολυπλοκότητας σε αποδοτικά (tractable), αν υπάρχει κάποιος αλγόριθμος που μπορεί να λύσει το πρόβλημα σε πολυωνυμικό χρόνο, ή μη-αποδοτικά (intractable) αν δεν υπάρχει κανένας αλγόριθμος που να τα επιλύει σε πολυωνυμικό χρόνο. Στην δεύτερη περίπτωση, το πρόβλημα είτε είναι άλυτο (δεν μπορεί να επιλυθεί με κανέναν αλγόριθμο) ή η επίλυσή του απαιτεί εκθετικό υπολογιστικό χρόνο.

Ιδιαιτέρου ενδιαφέροντος στο πλαίσιο αυτό είναι οι κλάσεις που θα αναφερθούν στη συνέχεια.

### 3.2.1 Η Κλάση P (Polynomial)

Περιλαμβάνει τα προβλήματα που επιλύονται (αποφασίζονται) από κάποιον αλγόριθμο εντός πολυωνυμικού χρόνου (για παράδειγμα το να βάλεις σε αλφαβητική σειρά μία λίστα ονομάτων ή να πολλαπλασιάσεις αριθμούς).

Περιλαμβάνει όλα τα (ισοδύναμα) προβλήματα απόφασης (δηλαδή προβλήματα στα οποία καλούμαστε να απαντήσουμε μια συγκεκριμένη ερώτηση με ναι ή όχι).

### 3.2.2 Η κλάση NP (Non Deterministic Polynomial)

Εδώ, το NP σημαίνει «μη ντετερμινιστικό πολυωνυμικό χρόνο». Περιλαμβάνει τα προβλήματα που αν μας δοθεί μία σωστή λύση, μπορούν τουλάχιστον να επαληθευτούν εντός πολυωνυμικού χρόνου (για παράδειγμα πολλά σημαντικά προβλήματα όπως η δρομολόγηση οχημάτων TSP (Travelling Salesman Problems), ο προγραμματισμός εργασιών (job scheduling),ο σχεδιασμός κυκλωμάτων και βάσεων δεδομένων, Sudoku, και άλλα).

Ένας μη ντετερμινιστικός αλγόριθμος αποτελείται από δύο στάδια. Το πρώτο στάδιο του αλγόριθμου απλά μαντεύει μία δομή S του προβλήματος I, η οποία εισάγεται στο δεύτερο στάδιο για να ελεγχθεί αν η δομή S αυτή αποτελεί λύση του προβλήματος I ή όχι. Σημειώνεται ότι στο δεύτερο στάδιο του μη ντετερμινιστικού αλγόριθμου χρησιμοποιείται ένας ντετερμινιστικός αλγόριθμος που λειτουργεί σε πολυωνυμικό χρόνο.

Από αυτήν την άποψη, η NP τάξη περιλαμβάνει την κατηγορία των προβλημάτων, για τα οποία μία λύση μπορεί να πιστοποιηθεί αποτελεσματικά σε πολυωνυμικό χρόνο, αλλά δεν είναι γνωστός ο τρόπος με τον οποίον αποκτάται αυτή η λύση. Είναι προφανές λοιπόν ότι, κάθε πρόβλημα που ανήκει στην κλάση P, ανήκει σίγουρα και στην κλάση NP και επομένως $P \subseteq NP$

### 3.2.3 Η κλάση NP-hard

Στη θεωρία της υπολογιστικής πολυπλοκότητας, η NP-δυσκολία (μη ντετερμινιστική πολυωνυμικού χρόνου δυσκολία), είναι η καθοριστική ιδιότητα μιας τάξης προβλημάτων που είναι τουλάχιστον τόσο δύσκολα, όσο τα δυσκολότερα προβλήματα στο NP.

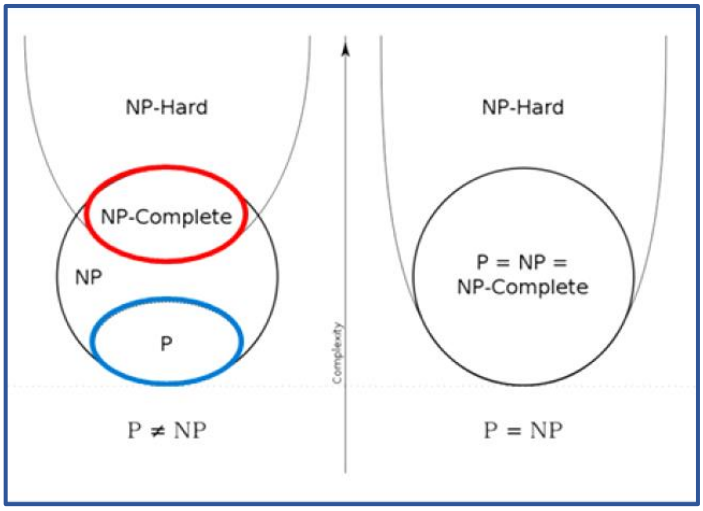
Πιο συγκεκριμένα, ένα πρόβλημα (H) είναι NP-δύσκολο, όταν κάθε πρόβλημα (L) σε NP μπορεί να μειωθεί σε πολυωνυμικό χρόνο σε πρόβλημα (H). Δηλαδή, υποθέτοντας ότι μια λύση για το (H) παίρνει 1 μονάδα χρόνου, μπορούμε να χρησιμοποιήσουμε τη λύση του (H) για να λύσουμε το (L) σε πολυωνυμικό χρόνο. Κατά συνέπεια, η εξεύρεση πολυωνυμικού αλγορίθμου για την επίλυση οποιουδήποτε προβλήματος NP-hard θα δώσει πολυωνυμικούς αλγορίθμους για όλα τα προβλήματα στο NP, κάτι που είναι απίθανο, καθώς πολλές από αυτές θεωρούνται δύσκολες.

### 3.2.4 Η κλάση NP-complete

Περιλαμβάνει τα προβλήματα που ανήκουν τόσο στην NP όσο και στην NP-hard κλάση πολυπλοκότητας. Αν και οποιαδήποτε δεδομένη λύση σε ένα NP-πλήρες πρόβλημα μπορεί να επαληθευτεί γρήγορα (σε πολυωνυμικό χρόνο), δεν υπάρχει γνωστός

αποτελεσματικός τρόπος για να εντοπιστεί μια. Το πιο αξιοσημείωτο χαρακτηριστικό των NP-πλήρων προβλημάτων είναι ότι δεν υπάρχει γρήγορη λύση σε αυτά. Δηλαδή, ο χρόνος που απαιτείται για την επίλυση του προβλήματος χρησιμοποιώντας οποιονδήποτε γνωστό αλγόριθμο αυξάνεται πολύ γρήγορα καθώς μεγαλώνει το μέγεθος του προβλήματος.

Ως εκ τούτου, ο προσδιορισμός του, εάν είναι δυνατή η ταχεία επίλυση αυτών των προβλημάτων, που ονομάζεται πρόβλημα P έναντι NP, είναι ένα από τα θεμελιώδη άλυτα προβλήματα στην επιστήμη των υπολογιστών σήμερα. Ενώ μια μέθοδος για τον υπολογισμό των λύσεων σε NP-πλήρη προβλήματα, χρησιμοποιώντας ένα λογικό χρονικό διάστημα παραμένει άγνωστη, οι επιστήμονες υπολογιστών και οι προγραμματιστές εξακολουθούν να αντιμετωπίζουν συχνά NP-πλήρη προβλήματα. Τα NP-πλήρη προβλήματα αντιμετωπίζονται συχνά χρησιμοποιώντας ευρετικές μεθόδους και αλγόριθμους προσέγγισης.




*Σχήμα 13.Γραφική απεικόνιση κλάσεων προβλημάτων P, NP, NP-πλήρη και NP-δύσκολα στις περιπτώσεις P≠NP και P=NP*

## 3.3 Διαχωρισμός των αλγορίθμων βελτιστοποίησης με βάση τις ιδιότητές τους

Συνήθως, πέρα από τη θεωρία και την ταξινόμηση των προβλημάτων με βάση τις θεωρητικές τους ιδιότητες, αυτό που μας ενδιαφέρει στην πράξη, κατά την ανάπτυξη ενός λογισμικού, έχει να κάνει με την ακρίβεια και την ταχύτητα του αλγορίθμου που θα επιλέξουμε για να επιλύσουμε το πρόβλημά μας.

Δυστυχώς οι δύο αυτές ιδιότητες είναι αντικρουόμενες, πράγμα που σημαίνει πως εάν προσπαθήσουμε να αυξήσουμε το ένα από τα δύο θα μειωθεί το άλλο. Σύμφωνα με την ιδιότητα της ταχύτητας που πρέπει να επιλυθεί το πρόβλημα τα διακρίνουμε σε δύο κατηγορίες: τα On Line Optimization και τα Off Line Optimization προβλήματα.

Τα πρώτα είναι αυτά που πρέπει να επιλυθούν σε χρονικό διάστημα λίγων μόνο λεπτών. Σε αυτή την περίπτωση, προσπαθούμε να επιτύχουμε όση μεγαλύτερη ταχύτητα γίνεται, θεωρώντας την ιδιότητα αυτή ως πρωταρχική, που το πρόγραμμα πρέπει να διαθέτει ακόμα και αν περιορίσουμε κάποιές άλλες, όπως την ακρίβεια προκειμένου να διασφαλίσουμε την ικανοποιητική ταχύτητα στην εκτέλεση του προγράμματος. Ένα κλασσικό παράδειγμα αλγόριθμου, είναι αυτός που προσπαθεί να επιλύσει το πρόβλημα της ανάθεσης κάποιων έργων σε μηχανές, με τρόπο αποδοτικό ώστε να ελαχιστοποιήσουμε το χρόνο αναμονής για την ολοκλήρωση των έργων στις μηχανές.

Στην περίπτωση Off Line Optimization προβλημάτων αυτό που ενδιαφέρει σε μέγιστο βαθμό είναι η ακρίβεια των αποτελεσμάτων και η εύρεση μιας λύσης όσο το δυνατόν πιο κοντά στην βέλτιστη. Εδώ υπάρχει το περιθώριο να περιμένουμε για τα αποτελέσματα ακόμη και κάποιες ημέρες.

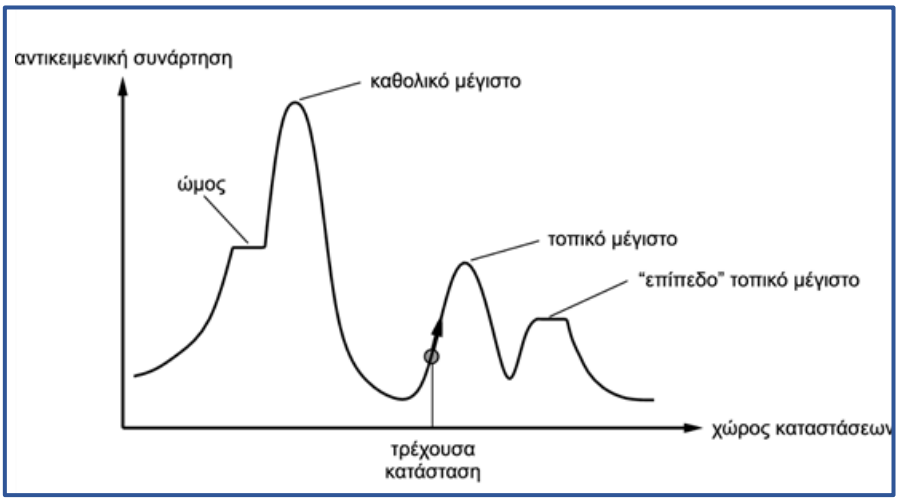
Υπάρχουν επίσης και οι λεγόμενοι semi-online αλγόριθμοι που βρίσκονται μεταξύ των γνωστών online και offline αλγορίθμων. Είναι πιο 'χαλαροί' από τους online αλγόριθμους, με τέτοιο τρόπο ώστε να επιτρέπουν ορισμένες επιπλέον λειτουργίες ή ώστε ο αλγόριθμος να γνωρίζει περισσότερα εκ των προτέρων (όπως τη βέλτιστη τιμή OPT-value). Για παράδειγμα, μπορεί να επιτρέπονται οι ακόλουθες λειτουργίες: ξαναπακετάρισμα ορισμένων βαρών, εμφάνιση των επόμενων βαρών ή κάποιο είδος αναδιάταξης.

# 4. Επισκόπηση γνωστών τεχνικών αλγόριθμων βελτιστοποίησης

*"Εάν υπάρχουν ... αρκετές λύσεις σε ένα πρόβλημα, ποια από αυτές είναι η καλύτερη;"*

Η παραπάνω πρόταση θα μπορούσε να είναι ο ορισμός των Προβλημάτων Βελτιστοποίησης και την απάντηση προσπαθούν να δώσουν οι αλγόριθμοι βελτιστοποίησης. Τα προβλήματα βελτιστοποίησης επιλύονται με τη χρήση αλγορίθμων ολικής αναζήτησης (global search algorithms) ή αλλιώς αλγόριθμοι βελτιστοποίησης (optimization algorithms). Οι αλγόριθμοι αυτοί είναι ικανοί να βρίσκουν λύσεις μέσα σε μεγάλους και συνεχείς χώρους καταστάσεων με την χρήση ποικίλων μεθοδολογιών που θα εξεταστούν στη συνέχεια.

Ως χώρος αναζήτησης (search space) ορίζεται το σύνολο των πιθανών τιμών της αντικειμενικής συνάρτησης (ή συνάρτηση κόστους όπως αλλιώς καλείται). Ένα παράδειγμα χώρου αναζήτησης φαίνεται στο σχήμα 14.[12]




*Σχήμα 14. Χώρος αναζήτησης λύσεων* [13]

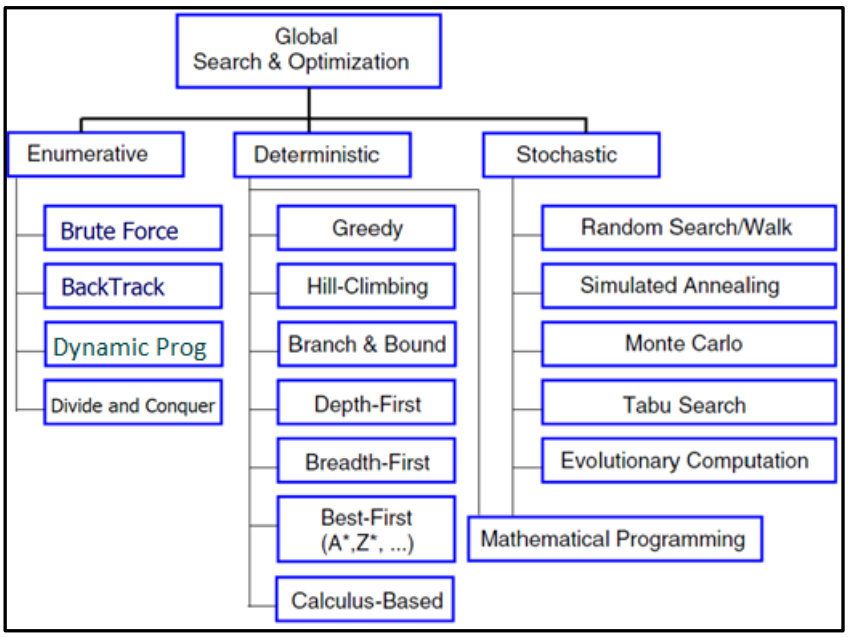
Υπάρχουν αρκετοί τρόποι ταξινόμησης των αλγορίθμων. Κατά κανόνα ακολουθούνται οι δυο παρακάτω:

Ο πρώτος τρόπος κατατάσσει τους αλγόριθμους σύμφωνα με το είδος του προβλήματος (αλγόριθμοι ταξινόμησης , ευρετικοί, γραφημάτων κλπ.). Το πλεονέκτημα αυτής της προσέγγισης είναι ότι επιτρέπει την άμεση σύγκριση της αποδοτικότητας διαφορετικών αλγορίθμων για το ίδιο πρόβλημα. Το μειονέκτημα αυτής της προσέγγισης είναι ότι επικεντρώνεται στο είδος του προβλήματος αφήνοντας σε δεύτερη μοίρα τις τεχνικές σχεδίασης.

Ο δεύτερος τρόπος οργανώνει την παρουσίαση με επίκεντρο τις τεχνικές σχεδίασης. Σε αυτήν την οργάνωση ομαδοποιούνται αλγόριθμοι από όλο το χώρο της πληροφορικής με μόνο κριτήριο το αν έχουν την ίδια σχεδιαστική προσέγγιση. [14]

Ακολουθώντας τον δεύτερο τρόπο οργάνωσης , οι αλγόριθμοι ολικής αναζήτησης ή βελτιστοποίησης μπορούν να χωριστούν σε τρεις κατηγορίες:[15]

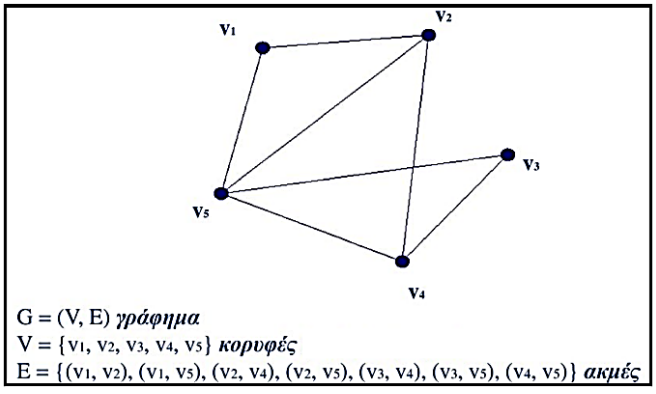
- απαριθμητικούς (enumerative)
- ντετερμινιστικούς (deterministic), ή ευρεστικούς (heuristic) και
- στοχαστικούς ή μεθευρετικούς




*Σχήμα 15.Ταξινόμηση αλγορίθμων βελτιστοποίησης* [15]

## 4.1 Απαριθμητικοί αλγόριθμοι

Οι απαριθμητικοί αλγόριθμοι παρόλο που είναι ντετερμινιστικοί (αιτιοκρατικοί) διαχωρίζονται από αυτούς, καθώς δεν χρησιμοποιούν ευρετικές τεχνικές (heuristics, κεφάλαιο 4.2). Αποτελούν την πιο απλή στρατηγική εξεύρεσης λύσης αφού υπολογίζουν και αξιολογούν όλες τις πιθανές λύσεις του χώρου αναζήτησης. Μπορεί να γίνει εύκολα αντιληπτό ότι τέτοιοι αλγόριθμοι δεν μπορούν να εφαρμοστούν σε πολύπλοκα προβλήματα με μεγάλους χώρους αναζήτησης καθώς αδυνατούν να δώσουν αποτέλεσμα μέσα σε αποδεκτό χρόνο. Ο σκοπός της απαρίθμησης είναι να απαριθμεί όλες τις εφικτές λύσεις ενός συγκεκριμένου προβλήματος.




*Σχήμα 16. Γράφημα G = (V, E)*

Για παράδειγμα, δίνοντας ένα γράφημα G = (V, E), απαριθμούμαι όλες τις διαδρομές μεταξύ των κορυφών για να βρούμε το συντομότερο μονοπάτι από μια κορυφή $V_1$ σε μια κορυφή $V_2$.

Ενώ ένας αλγόριθμος βελτιστοποίησης έχει ως στόχο να βρει μόνο την καλύτερη λύση σε ένα συγκεκριμένο πρόβλημα έστω και αν αυτή είναι μια ακραία περίπτωση, ένα πρόβλημα απαρίθμησης, στοχεύει να βρει όλες τις λύσεις που ικανοποιούν τους περιορισμούς. Η απαρίθμηση είναι ιδιαίτερα χρήσιμη για την βελτιστοποίηση όταν το πρόβλημα μας είναι περίπλοκο. Σε αυτές τις περιπτώσεις, η βέλτιστη λύση, επιλέγεται μεταξύ των αποτελεσμάτων της απαρίθμησης.[16]

Ο σχεδιασμός των αλγορίθμων απαρίθμησης περιλαμβάνει διάφορα θέματα που πρέπει να λάβουμε υπόψιν, προκειμένου να επιτευχθεί ορθότητα και αποτελεσματικότητα. Πράγματι, σε οποιαδήποτε απαρίθμηση, ο αλγόριθμος πρέπει να εγγυάται ότι κάθε λύση εξάγεται ακριβώς μία φορά, δηλαδή πρέπει αποφεύγεται η επανάληψη. Ένας απλός τρόπος, για να επιτευχθεί αυτό, είναι να αποθηκεύονται στη μνήμη όλες οι λύσεις που έχουν ήδη βρεθεί και κάθε φορά που εμφανίζεται μια νέα λύση, να ελέγχεται εάν έχει ήδη εξαχθεί ή όχι. Είναι σαφές ότι αυτή η προσέγγιση μπορεί να είναι αναποτελεσματική από την πλευρά της χρήσης μνήμης όταν οι λύσεις είναι πολλές. Η ενασχόληση με αυτό θα απαιτούσε μηχανισμό δυναμικής κατανομής μνήμης και αποτελεσματικής αναζήτησης [16].

Αν ο αριθμός των λύσεων είναι μικρός, ένας αποτελεσματικός αλγόριθμος πρέπει να τερματίζει μετά από σύντομη χρονικά (πολυωνυμική) αναζήτηση, και να μην πέφτει σε αέναους κύκλους. [16]

### 4.1.1 Εξαντλητική Αναζήτηση (Brute Force)

Οι αλγόριθμοι που βασίζονται στην μέθοδο της εξαντλητικής αναζήτησης (Brute Force) συνίσταται στον έλεγχο, σε όλες τις θέσεις ενός κειμένου ή πίνακα, διαστάσεων από 0 έως n, αν υπάρχει ή όχι ο αναζητούμενος αριθμός ή το αναζητούμενο σχέδιο αλφαριθμητικών, διάστασης m. Στη συνέχεια, μετά από κάθε προσπάθεια, συνεχίζει την αναζήτηση μεταφέροντας την θέση ελέγχου κατά μία ακριβώς θέση προς τα δεξιά.

Η καλύτερη περίπτωση μιας μεθόδου Brute Force είναι όταν το στοιχείο που αναζητούμε βρίσκεται στην αρχή του πεδίου αναζήτησης, αφού η λύση θα βρεθεί αμέσως. Η μέση περίπτωση θα ήταν, αν το στοιχείο είναι κάπου στη μέση και η χειρότερη περίπτωση θα ήταν να βρίσκεται στο τέλος. Συνεπώς όσο αυξάνει το πεδίο αναζήτησης τόσο αυξάνει και ο χρονική πολυπλοκότητα. Η μέθοδος λειτουργεί ικανοποιητικά για μικρές τιμές πεδίων αναζήτησης αλλά κρίνεται ανεπαρκής όσο το πεδίο αυξάνεται.

Στα θετικά του αλγορίθμου συμπεριλαμβάνονται ότι δεν απαιτεί φάση προ-επεξεργασίας και ούτε επιπλέον χώρο μνήμης για την αποθήκευση του σχεδίου ή του κειμένου αναζήτησης. [17]

**Αλγόριθμος 2: BruteForce** *(i,X)* [16]

**Input:** *An integer $i \geq 1$, a sequence of values $X= (x_0, x_1...x_{i-1})$, eventually empty*
**Output:** *All the feasible sequences of length n whose prefix is X*

```
1:  If no solution includes X, then return;
2:  If i>n, then
3:      If X is a solution, then output X;
4:  else
5:      Foreach feasible value e of x_i do
6:          BruteForce(i+1, (X,e))
7:      end
8:  end
```

Η χρονική πολυπλοκότητα του αλγόριθμου είναι **O (m*n)**. Αν αναζητούμε 'n' χαρακτήρες μέσα σε μια συμβολοσειρά, αποτελούμενη από 'm' χαρακτήρες, τότε χρειαζόμαστε n * m προσπάθειες για την εύρεση της λύσης.

### 4.1.2 Οπισθοδρόμηση (Backtrack)

Η οπισθοδρόμηση είναι μία γενική μέθοδος σχεδιασμού αλγορίθμων που επί της ουσίας αποτελεί μία βελτίωση της εξαντλητικής μεθόδου. Γι' αυτό, πολλές φορές τα αποτελέσματα των δύο μεθόδων συμπίπτουν αλλά σε πολλές περιπτώσεις οδηγεί σε πιο αποδοτικές λύσεις.

Μπορούμε να δούμε την οπισθοδρόμηση ως έναν τρόπο να εξερευνούμε το χώρο των λύσεων μέχρι να βρούμε μία λύση που να έχει την επιθυμητή ιδιότητα. Η εξερεύνηση αυτή γίνεται με έναν δομημένο τρόπο, ως δένδρο. Αυτό το δένδρο το ονομάζουμε δένδρο χώρου καταστάσεων (state space tree). Η ρίζα του δένδρου αποτελεί την αρχική κατάσταση, ενώ οι κόμβοι του αντιστοιχούν σε επιλογές που πιθανόν να μας οδηγούν σε μια λύση. Κάθε φύλλο του δένδρου αντιστοιχεί σε μία λύση. Κάποιοι κόμβοι μπορεί να οδηγούν σε επιθυμητές λύσεις, ενώ κάποιοι άλλοι να μην οδηγούν σε επιθυμητές λύσεις, με την έννοια ότι δεν υπάρχει φύλλο στο υποδένδρο τους που να έχει την επιθυμητή ιδιότητα.

Για να βρούμε την έξοδο σε ένα λαβύρινθο, ξεκινάμε από την αρχή του λαβύρινθου και αν στα βήματα που κάνουμε βρούμε την έξοδο, τότε σταματάμε (τερματική συνθήκη). Αν βρεθούμε σε αδιέξοδο επιστρέφουμε πίσω, στο προηγούμενο βήμα και αναδρομικά συνεχίζουμε. Αν όλα τα βήματα αποτύχουν, τότε επιστρέφουμε την απάντηση ότι ο λαβύρινθος δεν έχει έξοδο. Στην ουσία εφαρμόζουμε εξάντληση μόνο που δείχνουμε ξεκάθαρα το βήμα οπισθοδρόμησης (πήγαινε πίσω στην αναδρομή στην προηγούμενη επιλογή που έκανες και κάνε μία καινούργια). [2]

Κλασικό παράδειγμα εφαρμογής αυτής της τεχνικής είναι το πρόβλημα τοποθέτησης k-βασιλισσών σε μία σκακιέρα 8×8, σύμφωνα με την οποία αφού τοποθετηθεί μία βασίλισσα, στη συνέχεια ελέγχεται αν υπάρχει επιτρεπτή θέση, για να τοποθετηθεί η επόμενη. Αν βρεθεί τέτοια θέση, τότε ο αλγόριθμος διαχειρίζεται την τοποθέτηση της επόμενης βασίλισσας, διαφορετικά επιστρέφει (δηλαδή, οπισθοδρομεί) στην πιο πρόσφατα τοποθετημένη και την τοποθετεί στη επόμενη δυνατή θέση σε σχέση με αυτή όπου είχε τοποθετηθεί προηγουμένως. Αν τέτοια θέση δεν υπάρχει, τότε γίνεται οπισθοδρόμηση στην αμέσως προηγούμενη κ.ο.κ.

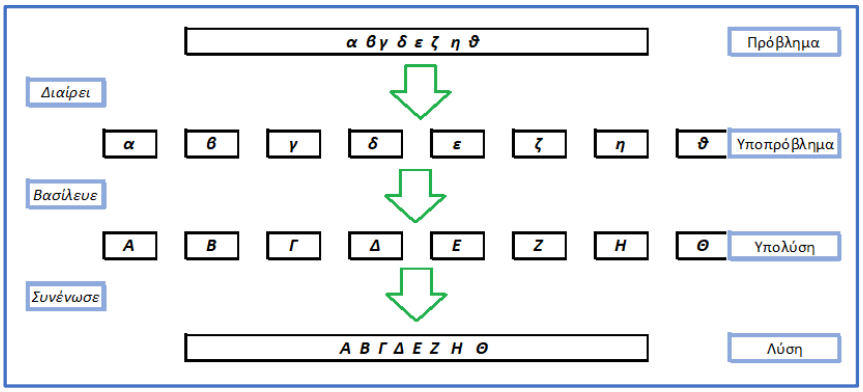
**Αλγόριθμος 3: Queens** *(n)*[2]

**Input:** *An integer n ≥ 1(number of Queens),*
**Output:** *All the feasible positions*

```
1:    X [1] ← 0; k ←1;;
2:    While k>0, do
3:        X[k] ← X[k]+1;
4:        While (X[k]<=n) and (not place(k)) do
5:            X[k] ← X[k]+1;
6:        if X[k]<=n then
7:            if k=n then
8:                for i ← 1 to n do write(X[i],' ')
9:            else
10:               k ← k+1; X[k] ← 0;
11:       else
12:           k ← k-1{* backtracking *}
13: end
```

### 4.1.3 Διαίρει και βασίλευε (Divide and Conquer)

Στην μέθοδο διαίρει και βασίλευε, το πρόβλημα χωρίζεται σε μικρότερα υποπροβλήματα (όχι απαραίτητα ίσα μεταξύ τους) με βάση ένα στοιχείο διαχωρισμού (pivot). Στη συνέχεια κάθε πρόβλημα επιλύεται ανεξάρτητα. Όταν συνεχίζουμε να διαιρούμε τα υποπροβλήματα σε ακόμη μικρότερα, μπορούμε τελικά να φθάσουμε σε ένα στάδιο όπου δεν υπάρχει πλέον διαίρεση. Αυτά τα "ατομικά" μικρότερα υποπροβλήματα επιλύονται. Οι λύσεις όλων των υποπροβλημάτων τελικά συγχωνεύονται προκειμένου να επιτευχθεί η λύση ενός αρχικού προβλήματος.




*Σχήμα 17. Μέθοδος «Διαίρει και Βασίλευε»*

Σε γενικές γραμμές, μπορούμε να κατανοήσουμε την προσέγγιση διαίρεσης και κατάκτησης σε μια διαδικασία τριών βημάτων.

- Διαίρεση. Αυτό το βήμα περιλαμβάνει τη διάσπαση του προβλήματος σε μικρότερα υποπροβλήματα. Τα δευτερεύοντα προβλήματα πρέπει να αποτελούν μέρος του αρχικού προβλήματος. Αυτό το βήμα απαιτεί γενικά μια αναδρομική προσέγγιση για να διαιρέσει το πρόβλημα έως ότου δεν υποδιαιρείται περαιτέρω

το υποπρόβλημα. Σε αυτό το στάδιο, τα υποπροβλήματα έχουν ατομική φύση, αλλά εξακολουθούν να αποτελούν μέρος του πραγματικού προβλήματος

- Κατάκτηση / Επίλυση. Αυτό το βήμα λαμβάνει πολλά μικρότερα υποπροβλήματα που πρέπει να επιλυθούν. Σε γενικές γραμμές, σε αυτό το επίπεδο, τα προβλήματα θεωρούνται «επιλυμένα» από μόνα τους
- Συγχώνευση / Συνδυασμός. Όταν επιλύονται τα μικρότερα υποπροβλήματα, το στάδιο αυτό συνδυάζεται αναδρομικά μέχρι να διαμορφώσουν μια λύση του αρχικού προβλήματος. Αυτή η αλγοριθμική προσέγγιση λειτουργεί αναδρομικά και κατακτά και συγχωνεύει βήματα που λειτουργούν τόσο κοντά ώστε να φαίνονται σαν ένα

Οι ακόλουθοι αλγόριθμοι υπολογιστών βασίζονται στην προσέγγιση προγραμματισμού Divide and Conquer [18]:

- Συγχώνευση ταξινόμησης
- Γρήγορη Ταξινόμηση
- Δυαδική αναζήτηση
- Ο πολλαπλασιασμός της μήτρας του Strassen
- Πλησιέστερο ζεύγος (σημεία)

**Αλγόριθμος 4: Divide_and_Conquer** *(S,n)* [19]

**Input:** *S is a large problem with input size of n*

```
1:  If (S is small to handle it), solve it X, then return;
2:  else
3:      split  S  into  two (equally-sized) subproblems  S1 and S2
4:      divide_and_conquer(S1)
5:      divide_and_conquer(S2)
6:      combine solutions to S1 and S2
7:  endif
8:  End
```

### 4.1.4 Δυναμικός Προγραμματισμός (Dynamic method)

Αυτή η εξαιρετικά χρήσιμη τεχνική στην ουσία είναι η τεχνική διαίρει και βασίλευε συνδυασμένη με την απομνημόνευση των ενδιάμεσων αποτελεσμάτων. Επί της ουσίας, η επίλυση των υποπροβλημάτων δεν γίνεται ανεξάρτητα αλλά υπάρχει κάποια αλληλεπίδραση.

Η βασική του ιδέα είναι να υπολογίσουμε μια φορά τις λύσεις μικρότερων προβλημάτων του αρχικού μας προβλήματος (sub-problems), να τις αποθηκεύσουμε σε έναν πίνακα έτσι ώστε να μπορούμε να τις επαναχρησιμοποιήσουμε αργότερα, όταν χρειαστεί. Προσπαθούμε ουσιαστικά να πάρουμε περισσότερο χώρο για να γλυτώσουμε χρόνο. [20]

Υπάρχουν δύο τύποι δυναμικού προγραμματισμού ως προς τον τρόπο που υπολογίζουν τις τιμές ενός προβλήματος. Ο "σύνθετος" και ο "αναλυτικός" δυναμικός προγραμματισμός.

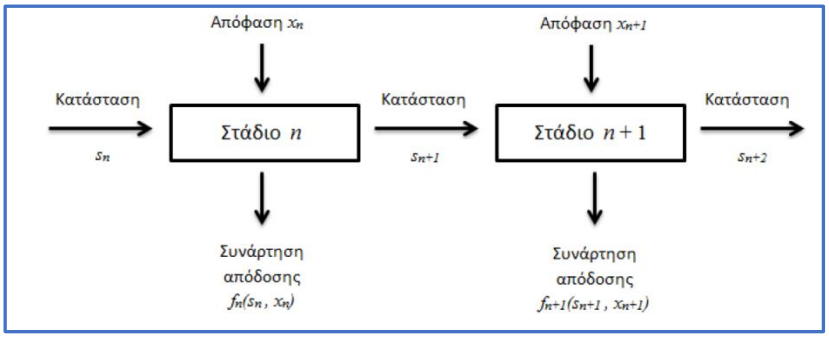
Ο "σύνθετος δυναμικός προγραμματισμός" (bottom-up), υπολογίζει και αποθηκεύει όλες τις τιμές μίας συνάρτησης με τη σειρά, υπολογίζοντας από την μικρότερη τιμή του ορίσματος.

Ο "αναλυτικός δυναμικός προγραμματισμός" (top-down), αποθηκεύει αρχικά τις τιμές που υπολογίζει αποφεύγοντας έτσι τον υπολογισμό των ήδη αποθηκευμένων. Είναι προτιμότερος διότι για την λύση ενός προβλήματος παρέχει μηχανικό τρόπο σχηματισμού και ρυθμίζει αυτόματα τη σειρά υπολογισμού των υπο-προβλημάτων του προβλήματος. Το πιο σημαντικό, είναι ότι αποφεύγει την επίλυση υπο-προβλημάτων που δεν είναι απαραίτητα και για αυτό αποτελεί αποδοτικότερη μέθοδο.

Η ανάπτυξη ενός αλγορίθμου δυναμικού προγραμματισμού μπορεί να αναλυθεί σε μία σειρά από τέσσερα βήματα:

- Χαρακτηρίζουμε τη δομή μιας βέλτιστης λύσης
- Ορίζουμε αναδρομικά την τιμή μιας βέλτιστης λύσης
- Υπολογίζουμε την τιμή μιας βέλτιστης λύσης δουλεύοντας "αναβιβαστικά" (δηλαδή από " bottom-up " ή από " top-down ")
- Κατασκευάζουμε μία βέλτιστη λύση από τα δεδομένα που υπολογίστηκαν

Τα τρία πρώτα βήματα είναι απαραίτητα για την επίλυση ενός προβλήματος χρησιμοποιώντας τη μέθοδο του δυναμικού προγραμματισμού. Εάν επιθυμούμε μπορούμε να συμπεριλάβουμε το 4ο βήμα και συγκεκριμένα την περίπτωση που αναζητούμε μία βέλτιστη λύση. Για να πραγματοποιηθεί αυτό, στο 3ο βήμα, οφείλουμε να κρατήσουμε παραπάνω πληροφορίες ώστε να περάσουμε στο τελευταίο βήμα, που θα επιφέρει τη βέλτιστη λύση.




*Σχήμα 18. Υλοποίηση Δυναμικού προγραμματισμού*[21]

Κλασικά προβλήματα που επιλύονται αποδοτικά με δυναμικό προγραμματισμό είναι τα ακόλουθα [22]:

- Αλυσιδωτός πολλαπλασιασμός πινάκων
- Διακριτό πρόβλημα σακιδίου
- Πρόβλημα εύρεσης συντομότερης διαδρομής σε κατευθυνόμενο γράφο
- Προβλήματα Αντικατάστασης-Συντήρησης Μηχανημάτων
- Προβλήματα Ελέγχου Αποθεμάτων
- Προβλήματα Βέλτιστης Διαδρομής
- Προβλήματα Κατανομής Πόρων και Φόρτωσης Φορτίων
- Προβλήματα Σχεδιασμού Σύνθετων Έργων

## 4.2 Ντετερμινιστικοί ή ευρετικοί αλγόριθμοι

Η αιτιοκρατία (ντετερμινισμός) είναι η φιλοσοφική τάση που επηρέασε ιδιαιτέρως την επιστημονική σκέψη από την αρχαιότητα μέχρι και σήμερα. Αποδέχεται την ύπαρξη της αιτιότητας, την καθολική αιτιώδη και νομοτελειακή συνάφεια όλων των φαινομένων.

Οι ντετερμινιστικοί ή ευρετικοί αλγόριθμοι λειτουργούν αποκτώντας γνώση του πεδίου λύσεων και την αξιοποιούν για να καθορίσουν την βέλτιστη λύση. Εφαρμόζονται κυρίως σε προβλήματα που περιέχουν γράφους και δενδρικές δομές.

Ευρετική αναζήτηση (ή και πληροφορημένη αναζήτηση) είναι η τεχνική επίλυσης προβλημάτων που χρησιμοποιούν και αξιοποιούν κάποια γνώση για το προς επίλυση πρόβλημα έτσι ώστε να οδηγηθούν πιο γρήγορα στην λύση του. Πλεονεκτήματά τους είναι ότι δίνουν καλά αποτελέσματα και ότι απαιτούν λιγότερο χρόνο σε σχέση με άλλες προσεγγίσεις. Ωστόσο πολλές φορές οι ευρετικές τεχνικές δεν μπορούν να εγγυηθούν ότι η συμπεριφορά αυτή θα ισχύει για όλες τις διαφορετικές περιπτώσεις προβλημάτων.

Οι ευρετικοί αλγόριθμοι αναζήτησης χρησιμοποιούνται στην εύρεση σχεδόν βέλτιστων λύσεων σε πολυωνυμικό χρόνο σε δύσκολα υπολογιστικά προβλήματα [23]. Το γεγονός αυτό στηρίζεται στο ότι οι καλές λύσεις βρίσκονται κοντά στο χώρο αναζήτησης. Και το γεγονός αυτό έχει ισχύ για τα περισσότερα προβλήματα πραγματικού κόσμου.

Εφόσον στόχος ενός ευρετικού αλγορίθμου είναι η εύρεση μιας σχεδόν-βέλτιστης λύσης, οι ευρετικοί αλγόριθμοι μπορεί να αποτύχουν να βρουν την ολικά βέλτιστη λύση. Ενώ μια ακριβής μέθοδος, πρώτα ολοκληρώνει τον έλεγχο του πεδίου λύσεων του αλγορίθμου και έπειτα αποφαίνεται για το ποια είναι η ολικά βέλτιστη λύση, ένας ευρετικός αλγόριθμος τερματίζεται σε οποιαδήποτε χρονική στιγμή κατά τη διάρκεια του τρεξίματος και αποδίδει την τοπικά βέλτιστη λύση που έχει εντοπιστεί ως τότε.

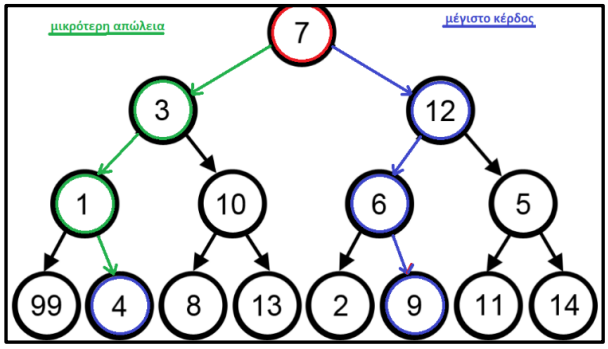
Οι ευρετικοί αλγόριθμοι αναζήτησης έχουν μερικά πλεονεκτήματα συγκρινόμενοι με άλλους αλγορίθμους [1]:

- Είναι εύκολοι γενικά στην υλοποίηση
- Χρησιμοποιούνται αποτελεσματικά σε περιβάλλον πολυεπεξεργαστή
- Δεν απαιτείται συνέχεια της συνάρτησης ορισμού του προβλήματος
- Γενικά παράγουν βέλτιστες ή σχεδόν βέλτιστες λύσεις
- Μπορούν να χρησιμοποιηθούν τόσο σε διακριτά όσο και σε συνδυαστικά προβλήματα

Οι κυριότερες μέθοδοι της κατηγορίας, αναλύονται στη συνέχεια.

### 4.2.1 Άπληστος αλγόριθμος (Greedy)

Οι Άπληστοι αλγόριθμοι, λειτουργούν βρίσκοντας τοπικά βέλτιστες λύσεις σε κάθε βήμα, θεωρώντας ότι οι λύσεις αυτές είναι πάντα κομμάτι της συνολικής βέλτιστης λύσης. Σε περιπτώσεις λοιπόν που δεν ισχύει αυτό οι αλγόριθμοι αυτοί, αποτυγχάνουν.




*Σχήμα 19. Λειτουργία Άπληστου αλγόριθμου*

Γενικά, οι άπληστοι αλγόριθμοι χαρακτηρίζονται από πέντε στοιχεία:

1. Ένα σύνολο πραγμάτων ή αλφαριθμητικών, στο οποίο αναζητείται η λύση
2. Μια λειτουργία σκοπιμότητας, η οποία χρησιμοποιείται για να προσδιορίσει εάν ένας υποψήφιος μπορεί να χρησιμοποιηθεί για να συμβάλει σε μια λύση
3. Μια λειτουργία επιλογής, η οποία επιλέγει τον καλύτερο υποψήφιο που θα προστεθεί στη λύση
4. Μια αντικειμενική συνάρτηση, η οποία αποδίδει μια αξία σε μια λύση, ή μια μερική λύση, και
5. Την λειτουργία λύσης, η οποία θα δείξει πότε θα βρεθεί μια ολοκληρωμένη λύση

Το πλεονέκτημα των άπληστων αλγορίθμων εστιάζεται στο ότι η εύρεση λύσης είναι αρκετά εύκολη. Το ίδιο ισχύει και για την ανάλυση του χρόνου εκτέλεσης. Το μειονέκτημα τους είναι ότι πρέπει να εργαστούμε περισσότερο για να αποδείξουμε ότι ο αλγόριθμος είναι ο βέλτιστος. Ακόμη και αν τον κατασκευάσουμε, είναι δύσκολο να αποδειχθεί γιατί είναι σωστός. Η απόδειξη ότι ένας άπληστος αλγόριθμος είναι σωστός είναι περισσότερο μια τέχνη παρά μια επιστήμη.

**Αλγόριθμος 5 : Greedy Make Change**[24]

**Input:** *set of coins of different denominations, amount-owned*
**Output:** *change*

```
1:  while (more coin-sizes && valueof(change)<amount-owned;
2:        Choose the largest remaining coin-size // Selection feasibility check//
3:        while (adding the coin does not make the valueof(change) exceed the amount-owed)
4:              then add coin to change
5:              if ( valueof(change) equals (amount-owned) //check if solved//
6:                  return (change)
7:              else delete coin-size
8:  return "failed to compute change"
```

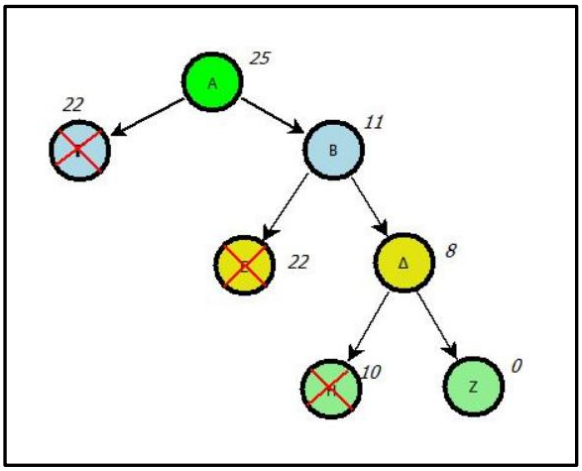
Η πολυπλοκότητα του άπληστου αλγορίθμου είναι O(nlogn) [25]

Κλασικά προβλήματα που επιλύονται αποδοτικά με την άπληστη μέθοδο είναι τα ακόλουθα [22][26]:

- Ο περιοδεύων πωλητής
- Πρόβλημα εύρεσης δένδρου επικάλυψης ελάχιστου κόστους (Prim-Kruskal)
- Πρόβλημα εύρεσης συντομότερης διαδρομής (Dijsktra)
- Χρωματισμός Γράφων
- Διακριτό πρόβλημα σακιδίου
- Προγραμματισμός εργασιών

### 4.2.2 Αναρρίχηση λόφων (Hill-Climbing)

Η Αναρρίχηση λόφων (HC) είναι μια τεχνική βελτιστοποίησης που ανήκει στην οικογένεια της τοπικής αναζήτησης. Ψάχνει για μια καλύτερη λύση στη γειτονιά, αξιολογώντας την τρέχουσα κατάσταση.




*Σχήμα 20. Λειτουργία μεθόδου αναρρίχησης λόφων*

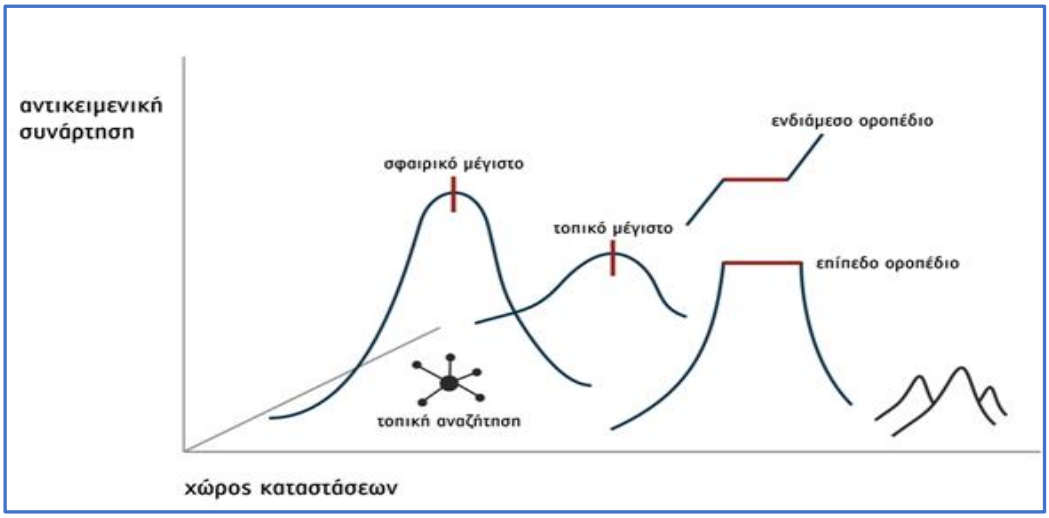
Η αναζήτηση γίνεται κατά μήκος ενός μόνο μονοπατιού στο χώρο αναζήτησης και προχωρά μόνο ανάμεσα σε διαδοχικά καλύτερους κόμβους, φράζοντας σε κάθε κύκλο αναζήτησης όλα τα υπόλοιπα μονοπάτια της γειτονιάς, εκτός από αυτό που οδηγεί στο βέλτιστη διαδρομή. Αν βρει μια καλύτερη κατάσταση μεταβαίνει σε αυτή, διαφορετικά τερματίζει, έχοντας βρει μία τοπικά βέλτιστη λύση. Αν υπάρχουν περισσότεροι του ενός καλύτεροι διάδοχοι κόμβοι, τότε η αναζήτηση επιλέγει τυχαία έναν από αυτούς.

Με βάση τα παραπάνω, στην HC, η βασική ιδέα είναι πάντα να κατευθυνθείτε προς μια κατάσταση που είναι καλύτερη από την τρέχουσα. Έτσι, βελτιώνει πάντα την ποιότητα μιας λύσης.

Προφανώς ο αλγόριθμος αναρρίχησης λόφων, δεν είναι πλήρης, αλλά είναι πολύ γρήγορος και απαιτεί ελάχιστη μνήμη. [27],[28]

Η μέθοδος αποδίδει αρκετά καλά, κάνοντας συχνά πολύ γρήγορη πρόοδο προς μια λύση, επειδή συνήθως είναι πολύ εύκολο να βελτιώσει μια κακή κατάσταση. Δυστυχώς, συχνά η αναρρίχηση λόφων παγιδεύεται όπως φαίνεται στο σχήμα 21 για τους εξής λόγους:

- τα τοπικά μέγιστα (local maxima)
- τα οροπέδια (plateau)
- τις κορυφογραμμές (ridges)




*Σχήμα 21. Παγίδες εγκλωβισμού της μεθόδου αναρρίχησης λόφων*

Η παγίδευση παρουσιάζεται όταν υπάρχει μια θέση που δεν είναι στόχος, αλλά από εκεί καμία κίνηση δεν βελτιώνει την τρέχουσα κατάσταση ή η θέση οδηγεί σε πρώιμη σύγκλιση, δηλαδή εύρεση λύσης που δεν αποτελεί τη βέλτιστη λύση. Αυτό θα συμβεί, αν έχουμε φτάσει σε ένα τοπικό μέγιστο, ένα οροπέδιο ή μια κορυφογραμμή .

Τοπικό μέγιστο: Είναι μια κατάσταση που είναι καλύτερη από όλες τις γειτονικές της, αλλά δεν είναι καλύτερη από όσο κάποιες άλλες πιο μακριά που δεν μπορούν να προσεγγιστούν από την πορεία που έχει ακολουθηθεί. Όλες οι πιθανές επόμενες κινήσεις φαίνεται να οδηγούν σε χειρότερες καταστάσεις. Λύση σε αυτό είναι η οπισθοδρόμηση σε κάποια προηγούμενη κατάσταση και η συνέχιση της αναζήτησης σε διαφορετική κατεύθυνση· όμως, ο κλασικός αλγόριθμος αναρρίχησης λόφων δεν το επιτρέπει.

Οροπέδιο: Είναι μια επίπεδη έκταση του χώρου αναζήτησης στην οποία ένα σύνολο γειτονικών καταστάσεων έχουν την ίδια ευρετική τιμή. Ως εκ τούτου, δεν είναι δυνατόν να καθοριστεί η καλύτερη κατεύθυνση προς την οποία πρέπει να συνεχιστεί η αναζήτηση. Λύση είναι να διαμορφωθεί έτσι ο αλγόριθμος, ώστε να μπορεί να κάνει ένα μεγάλο άλμα προς κάποια κατεύθυνση και να προσπαθήσει να προσεγγίσει ένα νέο τμήμα του χώρου αναζήτησης.

Κορυφογραμμές: Είναι μια περιοχή του χώρου αναζήτησης υψηλότερη από τις γύρω περιοχές (τοπική κορυφή), αλλά, αν η αναζήτηση βρίσκεται σε μια γειτονική κορυφογραμμή, δεν μπορεί να την προσεγγίσει με μια μοναδική κίνηση (ειδική περίπτωση τοπικών μεγίστων). Εδώ η εφαρμογή δύο ή περισσότερων βημάτων μετάβασης προς πολλές κατευθύνσεις ταυτόχρονα πρέπει να προηγηθεί του ευρετικού μηχανισμού για τον έλεγχο της καλύτερης πορείας.

Γενικές ενέργειες για την αντιμετώπιση παγιδεύσεων, όπως αυτές που αναφέραμε παραπάνω, είναι οι ακόλουθες:

- πλάγιες κινήσεις
- στοχαστική αναζήτηση
- τυχαία επιλογή καλών κινήσεων
- τυχαίες επανεκκινήσεις
- πλήρης αλγόριθμος

Διάφορες παραλλαγές του αλγόριθμου αναρρίχησης λόφων χρησιμοποιούν τους παραπάνω τρόπους αποφυγής παγιδεύσεων, θυσιάζοντας λίγη από την ταχύτητα της αναζήτησης, για να αυξήσουν την πιθανότητα να βρεθεί λύση:

- η Εξαναγκασμένη Αναρρίχηση Λόφων (Enforced Hill Climbing – EHC)
- ο αλγόριθμος προσομοιωμένης ανόπτησης (Simulated Annealing Algorithm)
- ο αλγόριθμος αναζήτησης με απαγορεύσεις (Taboo Search Algorithm),

**Αλγόριθμος 6 : Hill Climbing**[29]

1: **Define** *the current state* **as** *an initial state*;
2: **Loop until** *the goal state is achieved or no more operators can be applied on the current state*
3: **Apply** *an operation to current state and get a new state*
4: **Compare** *the new state with the goal*
5: **if** *the goal state is achieved* **Quit**
6: **Evaluate** *new state with heuristic function and compare it with the current state*
7: **If** *the newer state is closer to the goal compared to current state, update the current state*

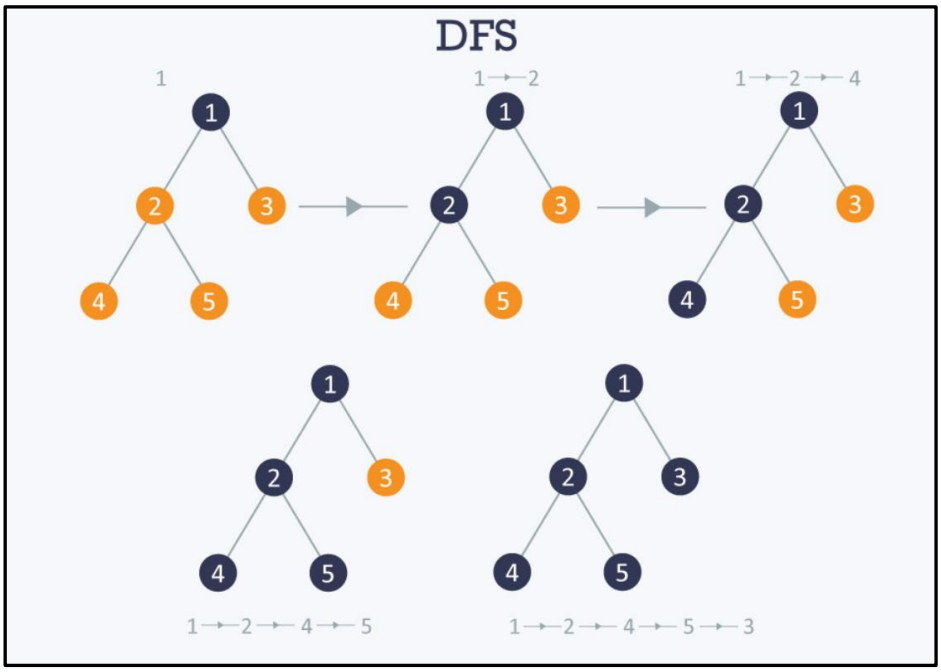
### 4.2.3 Αναζήτηση βάθους (depth-first)

Η αναζήτηση βάθους (DFS) είναι ένας αλγόριθμος διάσχισης γραφήματος σε δενδρική δομή. Διασχίζει ένα δέντρο ή γράφημα από τη γονική κορυφή προς τα κάτω, στις κορυφές των παιδιών και των εγγονών του σε μία μόνο διαδρομή μέχρι να φτάσει σε αδιέξοδο.

Όταν δεν υπάρχουν περισσότερες κορυφές προς επίσκεψη σε μια διαδρομή, ο αλγόριθμος DFS θα επιστρέψει σε ένα σημείο όπου μπορεί να επιλέξει μια άλλη διαδρομή που πρέπει να ακολουθήσει. Θα επαναλάβει τη διαδικασία ξανά και ξανά έως ότου έχουν επισκεφθεί όλες τις κορυφές.

Κατά τον έλεγχο κάθε κορυφής, υπάρχουν τρία πράγματα που πρέπει να κάνει ένας αλγόριθμος DFS:

- Να διαβάσει τα δεδομένα που είναι αποθηκευμένα στον κόμβο που ελέγχεται (Preorder)
- Να ελέγξει την κορυφή στα αριστερά του κόμβου που ελέγχεται.(Inorder)
- Να ελέγξει την κορυφή στα δεξιά του κόμβου που ελέγχεται (Post Order)




*Σχήμα 22. Μέθοδος αναζήτησης βάθους*[30]

Ανάλογα με το ποια σειρά θα εκτελεστούν τα παραπάνω βήματα έχουμε τρεις διαφορετικές στρατηγικές για τον αλγόριθμο:

- (Preorder) DLR. Root → Left Subtree → Right Subtree
- (Inorder) LDR. Left Subtree → Root → Right Subtree
- (Post Order) LRD. Left Subtree → Right Subtree → Root

Κατόπιν με αναδρομή βρίσκει την βέλτιστη λύση.

**Αλγόριθμος 7 :Depth First**[31]

**Input:** *Ένα γράφημα G και μια κορυφή v του G*
**Output:** *Ο χαρακτηρισμός των ακμών στο συνεκτικό στοιχείο της v είτε ως discovery edges (εξερευνημένοι κόμβοι) είτε ως back edges (ανεξερεύνητοι κόμβοι)*

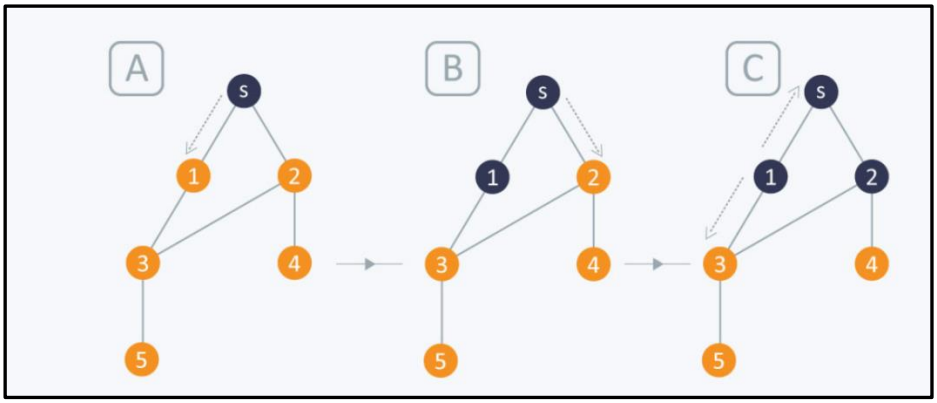
```
1:    procedure DFS(G,v):
2:        label v as explored
3:        for all edges e in G.adjacentEdges(v) do
4:            if edge e is unexplored then
5:                w ← G.adjacentVertex(v,e)
6:                if vertex w is unexplored then
7:                    label e as a discovery edge
8:                    recursively call DFS(G,w)
9:                else
10:                   label e as a back edge
```

Το συνολικό κόστος εκτέλεσης της αναζήτησης κατά βάθος είναι O(n+m).

### 4.2.4 Αναζήτηση κατά πλάτος (Breadth-first)

Ο αλγόριθμός αναζήτησης Breadth-first εφαρμόζεται και αυτός σε δενδρικές δομές και δουλεύει επιλέγοντας έναν κόμβο και ψάχνοντας πρώτα λύσεις σε όλους τους κόμβους που είναι γειτονικοί με αυτόν.

Χρησιμοποιεί την αντίθετη στρατηγική από την μέθοδο depth-first. Ελέγχει πρώτα όλους τους γειτονικούς κατά πλάτος κόμβους και μετά συνεχίζει κατά βάθος.




*Σχήμα 23. Μέθοδος αναζήτησης πλάτους*[32]

**Αλγόριθμος 8 : Breadth First**[32]

**Input:** *Ένα γράφημα G και μια κορυφή s του G*

```
1:   procedure BFS(G,s):        //Where G is the graph and s is the source node//
2:         let Q be queue
3:         Q.enqueue( s )   //Inserting s in queue until all its neighbour vertices are marked//
4:         mark s as visited
5:         while ( Q is not empty)  //Removing that vertex from queue, whose neighbour will
     be visited now//
6:                 v  =  Q.dequeue( )
7:        for all neighbours w of v in Graph G  //processing all the neighbours of v  //
8:                 if w is not visited
9:                         Q.enqueue( w ) //Stores w in Q to further visit its neighbour
10:                      mark w as visited.
```

### 4.2.5 Ο καλύτερος Πρώτος (Best-first)

Οι αλγόριθμοι τύπου Best-first χρησιμοποιούν ευριστικές πληροφορίες για να αποδώσουν αριθμητικές τιμές στους κόμβους ενός γράφου. Έπειτα εξετάζουν πρώτα τον πιο ελπιδοφόρο κόμβο .

Χαρακτηριστικό της μεθόδου είναι ότι σε κάθε βήμα αναζήτησης η μέθοδος επιλέγει προς επέκταση το πιο πολλά υποσχόμενο μονοπάτι μεταξύ των μονοπατιών που βρίσκονται στην ουρά.

Οι αλγόριθμοι A*, Z* είναι παραλλαγές της μεθόδου και πέρα από τον πιο ελπιδοφόρο κόμβο λαμβάνουν υπ' όψιν τους και το συνολικό κόστος για να φτάσουν σε αυτόν τον κόμβο[15].

**Αλγόριθμος 8 : Best First search**[33]

```
Input: closed list [ ], openlist [start node]
1:   do {
2:        if open list is empty then {
3:                              return no solution}
4:        n= heuristic best node
5:        if n== final node then {
6:                              return path from start to goal node}
7:         foreach direct available node do {
8:                 if node not in open and not in closed list do{
9:                          add node to open list
10:                         set n as his parent node}
11:        delete n from open list
12:        add n to closed list}
13:  while (open list is not empty)
```

### 4.2.6 Διακλάδωση και οριοθέτηση (Branch and bound)

Η μέθοδος Διακλάδωσης και Οριοθέτησης θεωρείται από τις αποτελεσματικότερες για την επίλυση προβλημάτων Επιχειρησιακής Έρευνας και ακεραίου προγραμματισμού (IP). Παρουσιάστηκε το 1960 από τους Αλίσα Λαντ και Άλισον Δόιγκ, εχοντας τις ρίζες του στην έρευνα για το πρόβλημα του πλανόδιου πωλητή.

Η μέθοδος βασίζεται στη τεχνική «διαίρει και βασίλευε» και αποτελεί ένα οργανωμένο τρόπο για να κάνουμε εξαντλητική αναζήτηση της άριστης λύσης ενός προβλήματος. Πρώτα χωρίζουμε το σύνολο των επιτρεπτών λύσεων του προβλήματος σε μικρότερα υποσύνολα (διακλάδωση) χτίζοντας ουσιαστικά ένα δέντρο αναζήτησης [34].

Στη συνέχεια χρησιμοποιούμε κανόνες οριοθέτησης ώστε:

- να προσδιορισθούν τα υποσύνολα με την μεγαλύτερη πιθανότητα να περιέχουν μια άριστη λύση,
- να προσδιορισθούν τα υποσύνολα που δεν χρειάζεται να διερευνηθούν περαιτέρω, καθώς υπάρχει μικρή πιθανότητα να περιέχουν βέλτιστη λύση.

Ένας αλγόριθμος Διακλάδωσης και Οριοθέτησης επηρεάζεται από τρεις στρατηγικές επιλογές:

- Την στρατηγική αναζήτησης (Search strategy). Συνηθέστερες επιλογές είναι η αναζήτηση κατά πλάτος (Breadth First) η οποία έχει μεγάλο κόστος σε χώρο μνήμης, η αναζήτηση του πρώτου καλύτερου (Best First) και η αναζήτηση κατά βάθος (Depth First)
- Την στρατηγική επιλογής φραγμάτων (Bound)
- Την στρατηγική επίλυσης ακμών (node selection)

**Αλγόριθμος 8 : Brunch and Bound**[35]

**Input:** *activeset:={problem},U:=∞; currentbest:=anything*

1: **while** *activeset is not empty* **do**
2: *choose a branching node k ∈ activeset*
3: *remove node k from activeset*
4: *generate the children of node k: child i, i= 1, . . . , nk,*
5: *and the corresponding lower bounds* $z_i$
6: **for** $i= 1, \ldots, n_k$ **do**
7: **if** $z_i \geq U$ **then** *kill child i*
8: **else if** *child i is a complete solution and* $z_i < U$ **then**
9: $U:=z_i$; *currentbest :=i*
10: **else** *add child i to activeset*
11: **end**
12: **end**

Το σύνολο activeset περιέχει τους ενεργούς κόμβους (live nodes) και αντιστοιχεί στο σύνολο των λύσεων που παράγονται από τους αλγόριθμους αναζήτησης.

Η μεταβλητή U αποθηκεύει το κόστος της καλύτερης πλήρους λύσης σε κάθε δοθείσα χρονική στιγμή. Η τιμή της U είναι ένα άνω όριο στο βέλτιστο κόστος. [35]

Η ανάλυση του αλγόριθμου μπορεί να παρασταθεί με γράφο για την καλύτερη κατανόηση του. Σε αυτόν ο κάθε κόμβος είναι και ένα υπο-πρόβλημα ενώ ο αρχικός κόμβος είναι το ίδιο το πρόβλημα.

Γενικά οι αλγόριθμοι αυτοί στην χειρότερη περίπτωση δεν είναι καλύτεροι από τη εξαντλητική αναζήτηση. Η απόδοση τους εξαρτάται από την στρατηγική που ακολουθούν. Η επιλογή ενός αρχικού καλού φράγματος επηρεάζει την απόδοση τους [34].

### 4.2.7 Mathematical Programming

Με τον όρο μαθηματικός προγραμματισμός αναφερόμαστε σε τεχνικές βελτιστοποίησης οι οποίες στηρίζονται σε ισχυρό μαθηματικό υπόβαθρο και οι οποίες με αφετηρία τον γραμμικό προγραμματισμό διαμόρφωσαν μια ολόκληρη κατηγορία υπολογιστικών μεθόδων ικανή να επιλύσει πληθώρα δύσκολων προβλημάτων βελτιστοποίησης [22].

Στη μαθηματική γλώσσα, ο μαθηματικός προγραμματισμός (επίσης γνωστός ως μαθηματική βελτιστοποίηση, Mathematical Optimization) είναι ένα μαθηματικό μοντέλο στο οποίο επιχειρείται η βελτιστοποίηση (μεγιστοποίηση ή ελαχιστοποίηση) μιας ή περισσοτέρων γραμμικών ή μη γραμμικών συναρτήσεων (κριτήρια βελτιστοποίησης) αγνώστων πραγματικών μεταβλητών, των οποίων το πεδίο τιμών οριοθετείται έμμεσα από γραμμικούς ή μη-γραμμικούς περιορισμούς (ανισοεξισώσεις) συναρτήσεις των μεταβλητών αυτών. Οι άγνωστες μεταβλητές προσδιορίζουν (μοντελοποιούν) το αντικείμενο απόφασης του προβλήματος και ονομάζονται για το σκοπό αυτό μεταβλητές απόφασης [36].

Ο όρος προγραμματισμός, δεν πρέπει να συγχέεται με τον προγραμματισμό των ηλεκτρονικών υπολογιστών. Ο όρος αυτός υποδηλώνει τον προγραμματισμό της λειτουργίας

ενός συστήματος, υπό την έννοια της λήψης των κατάλληλων αποφάσεων, έτσι ώστε η απόδοσή του να βελτιστοποιείται. Πρέπει, ωστόσο, να παρατηρήσουμε ότι τα συστήματα του πραγματικού κόσμου είναι τόσο πολύπλοκα και τόσο μεγάλης διάστασης, ώστε η βελτιστοποίησή τους με τις τεχνικές του γραμμικού προγραμματισμού να μην είναι δυνατή χωρίς τη χρήση ηλεκτρονικού υπολογιστή και τη βοήθεια ειδικού λογισμικού.

Ο μαθηματικός προγραμματισμός χρησιμοποιείται από τους επιχειρησιακούς ερευνητές ή τους αναλυτές προβλημάτων απόφασης για την προσέγγιση προβλημάτων κατανομής περιορισμένων πόρων ή μέσων σε εναλλακτικές και ανταγωνιστικές μεταξύ τους δραστηριότητες κατά τον καλύτερο δυνατό τρόπο [36].

Υπάρχουν πολλά υπο-πεδία στην μαθηματική βελτιστοποίηση [36], όπως:

- ***linear programming.*** Ο γραμμικός προγραμματισμός περιλαμβάνει όλα τα προβλήματα για τα οποία τόσο η αντικειμενική συνάρτηση όσο και όλοι οι περιορισμοί είναι γραμμικές συναρτήσεις (οι μεταβλητές εμφανίζονται μόνο στην πρώτη δύναμη και δεν υπάρχουν υψηλότερες δυνάμεις, ρίζες, γινόμενα μεταβλητών). Ο πιο γνωστός αλγόριθμος του γραμμικού προγραμματισμού είναι ο Simplex
- ***nonlinear programming.*** Όλα τα προβλήματα για τα οποία δεν ισχύει στην παραπάνω κατηγορία αυτό, ανήκουν στην κατηγορία προβλήματα μη γραμμικού προγραμματισμού
- ***integer programming.*** Ο ακέραιος προγραμματισμός περιλαμβάνει όλα τα προβλήματα στα οποία οι μεταβλητές απόφασης μπορούν να πάρουν μόνο ακέραιες τιμές. Ένα πρόβλημα ακέραιου προγραμματισμού μπορεί κατ' επέκταση να είναι γραμμικό ή μη γραμμικό
- ***mixed integer programming.*** Σε περίπτωση που κάποιες από τις μεταβλητές ενός προβλήματος περιορίζονται σε ακέραιες τιμές και κάποιες όχι, έχουμε ένα πρόβλημα μεικτού ακέραιου προγραμματισμού
- ***pure integer programming.*** Όταν όλες περιορίζονται σε ακέραιες τιμές, έχουμε ένα πρόβλημα αμιγώς ακέραιου προγραμματισμού
- ***binary integer programming.*** Ο δυαδικός ακέραιος προγραμματισμός είναι μία ειδική κατηγορία προβλημάτων ακέραιου προγραμματισμού, όπου οι μεταβλητές απόφασης μπορούν να πάρουν μόνο τιμές 0 ή 1
- ***quadratic programming.*** Ο τετραγωνικός προγραμματισμός είναι μία ειδική κατηγορία προβλημάτων μη γραμμικού προγραμματισμού, όπου η αντικειμενική συνάρτηση είναι δευτέρου βαθμού, ενώ οι μεταβλητές απόφασης υπόκεινται σε γραμμικούς περιορισμούς
- ***stochastic programming.*** Ο στοχαστικός προγραμματισμός είναι μια μέθοδος μοντελοποίησης για προβλήματα βελτιστοποίησης που εμπεριέχουν μεταβλητές και περιορισμούς με έντονο το στοιχείο της αβεβαιότητας. Η αβεβαιότητα αυτή προκύπτει από την έλλειψη αξιόπιστων δεδομένων, από λάθη μετρήσεων ή από παραμέτρους που περιέχουν μελλοντικές πληροφορίες. Είναι φανερό πως ο στοχαστικός προγραμματισμός είναι μια κατηγορία προβλημάτων και όχι μια μέθοδος επίλυσης αυτών.

## 4.3 Στοχαστικοί ή μεταευρετικοί αλγόριθμοι

Οι αλγόριθμοι με απαριθμητική ή ντετερμινιστική αναζήτηση, αδυνατούν να δώσουν λύσεις σε αρκετά πολύπλοκα επιστημονικά προβλήματα με ανώμαλους χώρους αναζήτησης. Για αυτό το λόγο έχουν αναπτυχθεί αλγόριθμοι στοχαστικής αναζήτησης και βελτιστοποίησης οι οποίοι μπορούν να διαχειριστούν τέτοια πολύπλοκα συστήματα.

Τέτοιοι αλγόριθμοι είναι ο Simulated Annealing, ο Monte Carlo, ο Tabu search, και Evolutionary Computation. Οι στοχαστικές μέθοδοι δουλεύουν αναθέτοντας τιμές σε μία συνάρτηση κόστους και αξιολογώντας τις πιθανές ή μερικές λύσεις που προκύπτουν. Αν και οι περισσότερες μέθοδοι βρίσκουν τελικά μία βέλτιστη λύση, δεν μπορούν να εγγυηθούν ότι αυτή είναι η καθολικά βέλτιστη λύση του πεδίου καταστάσεων [15].

Ορισμένες ευρετικές τεχνικές που χρησιμοποιούνται για την εύρεση καλών λύσεων σε συγκεκριμένα προβλήματα είναι ανεξάρτητες από τα ίδια τα προβλήματα που αντιμετωπίζουν. Οι τεχνικές αυτές ονομάζονται μεταευρετικές και πολλές από αυτές έχουν εμπνευστεί από μηχανισμούς που παρατηρούνται στην φύση. Οι μεταευρετικές τεχνικές μπορούν να ομαδοποιηθούν με βάση τον τρόπο με τον οποίο εξετάζουν τις νέες πιθανές λύσεις. Μέθοδοι όπως η tabu αναζήτηση και η προσομοιωμένη ανόπτηση εξετάζουν μόνο μια νέα λύση σε κάθε βήμα, ενώ μέθοδοι όπως οι γενετικοί αλγόριθμοι και η διασκορπισμένη αναζήτηση χειρίζονται παράλληλα, ένα μεγάλο πληθυσμό από λύσεις. Μια άλλη κατηγοριοποίηση των μεταευρετικών τεχνικών τις χωρίζει σε τεχνικές που χρησιμοποιούν μνήμη και σε τεχνικές που δεν χρησιμοποιούν μνήμη. Στις μεν πρώτες καταγράφονται χαρακτηριστικά των ενδιάμεσων λύσεων που παράγονται κατά την εφαρμογή της μεθόδου με στόχο να υποβοηθηθεί η λήψη μελλοντικών αποφάσεων ενώ στις τεχνικές που ανήκουν στην δεύτερη κατηγορία οι ενέργειες που εκτελούνται εξαρτώνται μόνο από την τρέχουσα κατάσταση.[22]

Μεταευρετικές τεχνικές: Ο όρος «μεταευρετικός» αποτελεί σύνθεση των λέξεων «μετά» και «ευρίσκω» και χρησιμοποιείται έχοντας την έννοια της αναζήτησης σε υψηλότερο επίπεδο. Οι μεταευρετικές τεχνικές στηρίζονται στην παραδοχή ότι ενδεχομένως δεν θα εντοπίσουν την βέλτιστη λύση αλλά στην πράξη έχει αποδειχθεί ότι συχνά αποτελούν τον πλέον ενδεδειγμένο τρόπο για την αντιμετώπιση δύσκολων προβλημάτων συνδυαστικής βελτιστοποίησης. Οι Osman και Laporte περιγράφουν τις μεταευρετικές τεχνικές ως εξής: "Μια μεταευρετική τεχνική ορίζεται ως μια επαναληπτική διαδικασία δημιουργίας λύσεων η οποία καθοδηγεί μια δευτερεύουσα ευρετική τεχνική συνδυάζοντας με ευφυή τρόπο διαφορετικούς τρόπους εξερεύνησης και ανάλυσης του χώρου αναζήτησης, ενώ στρατηγικές μάθησης χρησιμοποιούνται προκειμένου να εντοπίσουν με αποδοτικό τρόπο λύσεις οι οποίες βρίσκονται κοντά στις βέλτιστες."

Οι μεταευρετικές τεχνικές φαίνεται να έχουν ισχυρή δυναμική περαιτέρω εξάπλωσης και χρήσης. Σε αυτό το συμπέρασμα συνηγορεί το γεγονός ότι εκτός από τις παραδοσιακές μεταευρετικές τεχνικές (Simulated Annealing, Tabu Search, κ.α.) τα τελευταία χρόνια έχουν αναπτυχθεί νέες τεχνικές (Variable Neighborhood Search, Guided Local Search, …). Οι μεταευρετικές τεχνικές μπορούν να λειτουργήσουν είτε ανεξάρτητα είτε σε υβριδικούς συνδυασμούς είτε οργανωμένες σε διάφορα επίπεδα αφαίρεσης (Hyperheuristics). Θα πρέπει να τονιστεί ότι για ορισμένες τεχνικές τα όρια ανάμεσα στην τεχνητή νοημοσύνη, την υπολογιστική νοημοσύνη και τις μεταευρετικές τεχνικές δεν είναι σαφή. Για παράδειγμα οι

γενετικοί αλγόριθμοι αναφέρονται στην βιβλιογραφία ως παράδειγμα τεχνικής και στις τρεις προαναφερθείσες κατηγορίες.[22]

### 4.3.1 Τυχαία αναζήτηση (Random Search)

Η τυχαία αναζήτηση (random search) είναι η απλούστερη στοχαστική στρατηγική αναζήτησης. Αξιολογεί ένα σύνολο τυχαίων λύσεων στο πεδίο καταστάσεων και επιλέγει την καλύτερη από αυτές .

Το μειονέκτημα της μεθόδου είναι ότι κατά την παραγωγή νέων λύσεων δεν αξιοποιούνται οι λύσεις από προηγούμενες επαναλήψεις και έτσι ενώ μπορεί να βρεθούμε πολύ κοντά σε μια βέλτιστη λύση, υπάρχει ο κίνδυνος να μην την βρούμε ποτέ.

Η απόδοση της μεθόδου αποτελεί το κάτω όριο στην απόδοση άλλων αλγορίθμων προκειμένου να γίνουν αποδεκτοί. Δηλαδή θα πρέπει να πετυχαίνει καλυτέρα αποτελέσματα από την μέθοδο RS.

Ο παρακάτω αλγόριθμος 9 περιγράφει τον ψευδοκώδικα για την ελαχιστοποίηση μιας συνάρτησης κόστους.

**Αλγόριθμος 9: Random Search** [37]

**Input:** *NumIterations, ProblemSize, SearchSpace*
**Output:** *Best*

*Best* ← *0;*
***For (****iter*$_i$ ∈ *NumIterations)*
    *candidate*$_i$ ←*RandomSolution(ProbleSize,SearchSpace)*
    ***If (Cost(****candidate*$_i$***)< Cost(Best)***
        *Best* ← *candidate*$_i$
    **end**
**end**
**Return(***Best***)**

Υπάρχουν διάφορες παραλλαγές του βασικού αλγορίθμου [38] όπως:

- ο αρκετά συναφής random walk ο οποίος αντί να ψάχνει σε κάθε βήμα μια τυχαία λύση χρησιμοποιεί την λύση κάθε βήματος ως αφετηρία για να επιλέξει την επόμενη τυχαία λύση.
- Ο αλγόριθμος σταθερού βήματος (Fixed Step Size Random Search -FSSRS)
- Ο αλγόριθμος με βέλτιστο μέγεθος βήματος (Optimum Step Size Random Search -OSSRS).
- Ο αλγόριθμος με προσαρμόσιμο μέγεθος βήματος (Adaptive Step Size Random Search -ASSRS)

### 4.3.2 Προσομοιωμένη ανόπτηση (simulated annealing)

Ο αλγόριθμος προσομοιωμένης ανόπτηση (simulated annealing) δουλεύει προσομοιώνοντας την φυσική διαδικασία της ανόπτησης, κατά την οποία, αρχικά ένα υλικό θερμαίνεται αρκετά και έπειτα αφήνεται σταδιακά να ψυχθεί αργά, έως ότου στερεοποιηθεί σε μια τέλεια κρυσταλλική δομή και αλλάξουν οι ιδιότητες του υλικού (για παράδειγμα κατασκευή γυαλιού από πυρίτη).

Οι φυσικές καταστάσεις των υλικών μετά την θέρμανση αντιστοιχούν σε προβληματικές λύσεις, η ενέργεια που καταναλώθηκε αντιστοιχεί στο κόστος μιας λύσης και η θερμοκρασία σε μια παράμετρο ελέγχου της διαδικασίας λύσης.

Ο αλγόριθμος σε κάθε βήμα αντικαθιστά την τρέχουσα λύση με μία τυχαία κοντινή λύση η οποία επιλέγεται βάση πιθανότητας και εξαρτάται από μία παράμετρο T, η οποία αρχικά έχει μία μεγάλη τιμή επιτρέποντας την αναζήτηση σε μακρινές λύσεις (σε σχέση με την αρχική), ενώ σταδιακά μειώνεται (“ψύχεται”), περιορίζοντας έτσι τις αναζητήσεις σε πιο κοντινά διαστήματα ως προς την τρέχουσα λύση. Ο αλγόριθμος σταματάει τις αναζητήσεις όταν η τιμής της μεταβλητής T μηδενιστεί και επιστρέφει την “βέλτιστη” λύση που έχει εντοπίσει .

**Αλγόριθμος 10: Simulated Annealing** [39]

*Initialize the system configuration*
*Randomize x(0)*
**Repeat:**
  **a. Repeat:**
    *i. Apply random pertubations to the state x=x+Δx*
    *ii. Evaluate ΔE(x) = E(x+Δx) – E(x):*
      **If** *ΔE(x)<0, keep the new state;*
      **Otherwise,** *accept the new state with probability* $P = e^{-\frac{\Delta E}{T}}$
    **Until** *the number of accepted transitions is below a threshold level*
  **b. Set T=T-ΔT**
**Until** *T is small enough*

### 4.3.3 Monte Carlo

Η προσομοίωση Monte Carlo πήρε το όνομά της από την πόλη του Μονακό και το ομώνυμο το καζίνο. Στα τυχερά παιχνίδια του καζίνο, όπως η ρουλέτα, συμβαίνουν επαναλαμβανόμενα γεγονότα που βασίζονται στους κανόνες των πιθανοτήτων.

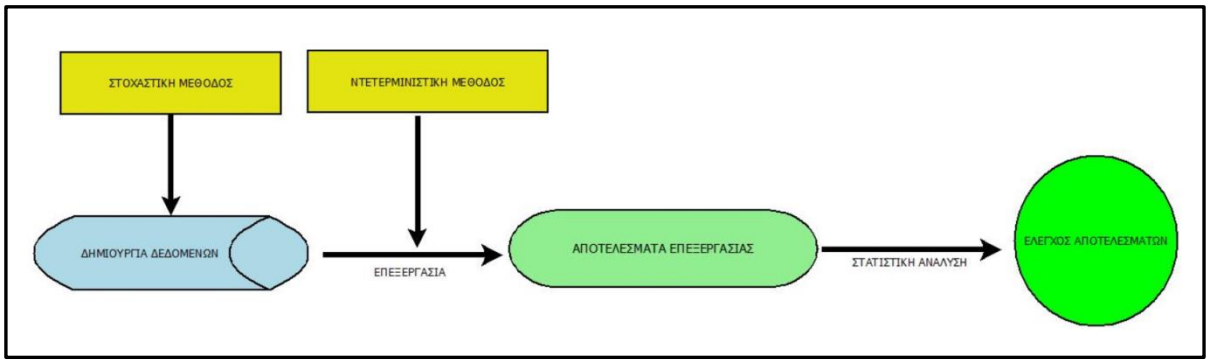
Αν και από αρκετά χρόνια πριν, υπήρχαν πολλές απομονωμένες και ανεπτυγμένες εφαρμογές των αρχών προσομοίωσης Monte Carlo, η σύγχρονη εφαρμογή των μεθόδων Monte Carlo χρονολογείται από τη δεκαετία του 1940 κατά τη διάρκεια της κατασκευής της ατομικής βόμβας. Ο μαθηματικός Stanislaw Ulam, κατάφερε να μετατρέψει την ιδέα της μεθόδου σε πρόγραμμα για υπολογιστές.

Οι αλγόριθμοι τύπου Monte Carlo δουλεύουν στηριζόμενοι σε στατιστικές μεθόδους όπου κάθε τυχαία λύση είναι εντελώς ανεξάρτητη από την προηγούμενη λύση και το

αποτέλεσμά της. Η καλύτερη λύση σε κάθε βήμα αποφασίζεται βάση μεταβλητών απόφασης που στηρίζονται σε στατιστικά σφάλματα. [40]

Οι μέθοδοι Monte-Carlo ακολουθούν γενικά τα παρακάτω βήματα[41]:

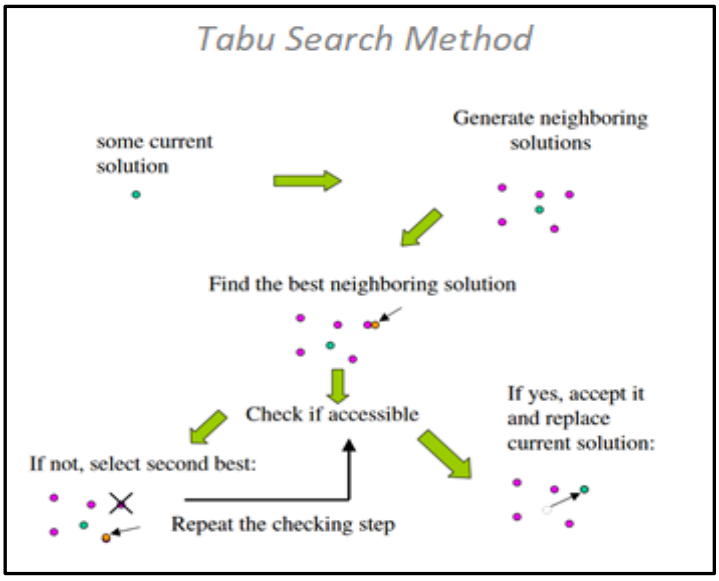
1. Προσδιορίζουν τις στατιστικές ιδιότητες των πιθανών εισόδων
2. Δημιουργούν πολλά σύνολα πιθανών εισόδων που ακολουθούν αυτές τις στατιστικές ιδιότητες
3. Εκτελούν έναν προκαθορισμένο υπολογισμό με αυτά τα σύνολα
4. Αναλύουν στατιστικά τα αποτελέσματα




*Σχήμα 24. Μέθοδος αναζήτησης Μόντε Κάρλο*

### 4.3.4 Tabu search

Ο αλγόριθμος Tabu search (αναζήτηση ταμπού) είναι ένας αλγόριθμος που λειτουργεί αξιολογώντας γειτονικές λύσεις (ως προς την τρέχουσα) στο χώρο αναζήτησης και χρησιμοποιώντας μνήμη όλων των προηγούμενων λύσεων καθώς και το πώς αυτές επιτεύχθηκαν.




*Σχήμα 25. Μέθοδος Tabu* [42]

Χρησιμοποιεί ως κριτήρια αξιολόγησης την ποιότητα της λύσης, την συχνότητα που εμφανίζεται στον χώρο αναζήτησης, το πόσο πρόσφατη είναι καθώς και τι επιρροή έχει η εκάστοτε λύση στην βελτίωση της ποιότητας (δηλαδή κατά πόσο μεταβάλλεται η βέλτιστη λύση). Κάθε φορά που υπάρχει βελτίωση της λύσης αυτή μαρκάρεται ως "ταμπού" ώστε ο αλγόριθμος να μην την ξαναεπισκεφθεί κατά την διαδικασία αναζήτησης .

### 4.3.5 Εξελικτικός προγραμματισμός (Evolutionary Computation)

Ο όρος εξελικτικός προγραμματισμός (evolutionary computation) περιγράφει μία οικογένεια μεθόδων που προσομοιώνουν υπολογιστικά την εξελικτική διαδικασία. Βασίζονται σε πληθυσμούς μεμονωμένων λύσεων, οι οποίοι εξελίσσονται στον χρόνο βάση της δαρβινικής θεωρίας περί επιβίωσης του δυνατότερου [15].

Οι εξελικτικοί αλγόριθμοι είναι η κυριότερη υποκατηγορία μεθόδων εξελικτικού προγραμματισμού και χρησιμοποιούν τις διαδικασίες της αναπαραγωγής, της επιλογής, της τυχαίας διακύμανσης και του ανταγωνισμού για την διαχείριση ενός πληθυσμού λύσεων οι οποίες αξιολογούνται από μία συνάρτηση καταλληλότητας [15].

Οι διαδικασίες που συντελούνται στην λειτουργία των εξελικτικών αλγορίθμων είναι οι ίδιες που συμβαίνουν στους έμβιους οργανισμούς [1]:

- Η αναπαραγωγή. Είναι υπεύθυνη για την μεταφορά του γενετικού υλικού από ένα άτομο στους απογόνους του
- Η μετάλλαξη. Η πληροφορία που μεταφέρεται από γενιά σε γενιά αλλοιώνονται από διάφορους παράγοντες. Οι αλλοιώσεις μπορεί να είναι ευεργετικές η και καταστροφικές.
- Ο συναγωνισμός. Λαμβάνει μέρος όταν οι πόροι του περιβάλλοντος είναι περιορισμένοι και μέσα από αυτό αναδεικνύονται οι πιο προσαρμόσιμοι οργανισμοί οι οποίοι και θα επιβιώσουν
- Και η επιλογή όπου ο γονέας επιλέγει τους καλύτερους απογόνους

Ο διαχωρισμός τους σε υποκατηγορίες (γενετικοί, αποικία μερμηγκιών, νοημοσύνης σμήνους, νευρωνικά δίκτυα, και άλλες) βασίζεται σε λεπτομέρειες που χαρακτηρίζουν την λειτουργεία του κάθε αλγόριθμου.

Οι εξελικτικοί αλγόριθμοι είναι σχετικά οι πιο πρόσφατοι στο χώρο των αλγόριθμων βελτιστοποίησης και επίσης είναι οι πιο "δυνατοί" στο να βρίσκουν ολικά βέλτιστες λύσεις σε μία πληθώρα από διαφορετικά πολύπλοκα επιστημονικά προβλήματα, χωρίς να παγιδεύονται σε τοπικά βέλτιστες λύσεις [12].

Έστω $f$ μία αντικειμενική συνάρτηση και $P$ ένας πληθυσμός αποτελούμενος από μ άτομα, $P(t) = \{x_1(t), x_2(t), x_3(t).....x_\mu(t)\}$, καθένα από τα οποία αποτελεί μια πιθανή λύση του προβλήματος ελαχιστοποίησης. Σε μια δεδομένη χρονική στιγμή t, ο πληθυσμός $P(t) = \{x_1(t), x_2(t), x_3(t).....x_\mu(t)\}$, χαρακτηρίζεται από τις συναρτησιακές τιμές $f(x_1(t)), f(x_2(t)), f(x_3(t)).....f(x_\mu(t))$. Έτσι, ο πληθυσμός μπορεί να χαρακτηριστεί σε κάθε χρονική στιγμή από το διάνυσμα, $F(t) = \{f(x_1(t)), f(x_2(t)), f(x_3(t)).....f(x_\mu(t))\}$, που καλείται και ικανότητα, απόδοση ή προσαρμογή του πληθυσμού.

Αν μας δοθούν, κάποιοι επιπλέον παράμετροι, όπως το μέγεθος του πλήθους (λ), των παραγόμενων απογόνων ανά γενιά και των πιθανοτήτων ανασυνδυασμού ($p_r$), μετάλλαξης ($p_m$) και επιλογής ($p_s$), τότε η λειτουργία των εξελικτικών αλγόριθμων μπορεί να περιγραφεί από τον παρακάτω ψευδοκώδικα[1]:

**Αλγόριθμος 11: Evolutionary Computation**[1]

**Input**: $\mu, \lambda, p_r, p_m, p_s$
1: **Θέσε** $t=0$
2: **Αρχικοποίησε** *τον πληθυσμό P(t)*
3: **Υπολόγισε** *την απόδοση F(t) του P(t)*
4: **Επανέλαβε**
5: $P'(t)=$ *Ανασυνδυασμός* $(P(t), p_r)$
6: $P''(t)=$ *Μετάλλαξη* $(P'(t), p_m)$
7: $F(t)=$ *Αξιολόγηση* $(P''(t), F(t), \mu, p_s)$
8: $t=t+1$
9: **Μέχρι** *να ισχύσει το κριτήριο τερματισμού*

Οι παράμετροι μ, λ, $p_r$, $p_m$, $p_s$ ονομάζονται και παράμετροι στρατηγικής. [1]

### *4.3.5.1 Αλγόριθμος της Αποικίας Μυρμηγκιών (ACO)*

Ο Αλγόριθμος της Αποικίας Μυρμηγκιών (ant colony optimization), όπως εύκολα συμπεραίνεται από το όνομα του, είναι εμπνευσμένος από την συμπεριφορά των μυρμηγκιών στη φύση και συγκεκριμένα από τον τρόπο με τον οποίο αυτά, αναζητούν το βέλτιστο μονοπάτι μεταξύ της αποικίας και της πηγής της τροφής. Προτάθηκε το 1991 από τον Μάρκο Ντορίγκο [43].

Σύγκριση φυσικών και ψηφιακών μυρμηγκιών:

1. Το μονοπάτι που ενώνει τη φωλιά με τη τροφή στα φυσικά μυρμήγκια, στα ψηφιακά μυρμήγκια πρόκειται για ένα γράφημα Κόμβων και Συνδέσεων
2. Η φερομόνη που αφήνουν τα φυσικά μυρμήγκια όταν πάνε να βρουν τροφή και που, όπως είδαμε, εξατμίζεται στα ψηφιακά μυρμήγκια, αντιστοιχεί στις διάφορες θέσεις με αριθμητικές πληροφορίες, δίνοντας μας όλα τα πιθανά μονοπάτια του συστήματος. Στα μονοπάτια που δεν ικανοποιούν το σύστημα υπάρχει αρνητική αντίδραση (εξάτμιση)

Ο αλγόριθμος ACO βρίσκει τις καλύτερες λύσεις που κατασκευάστηκαν από προηγούμενες επαναλήψεις. Συγκεκριμένα σε κάθε σύνθεση του γράφου αντιστοιχεί μια μεταβλητή (τεχνητή φερομόνη) και ανάλογα πόσο καλύτερη είναι η λύση, τόσο περισσότερη είναι η τεχνητή φερομόνη. Στο τέλος που θα έχουμε όλες τις πιθανές λύσεις, θα δούμε την ποσότητα της τεχνητής φερομόνης ώστε να δούμε ποια είναι η βέλτιστη.

Όμως υπάρχουν κάποια στοιχεία που δεν συναντούμε στα φυσικά μυρμήγκια, αλλά υπάρχουν στα συστήματα πολλαπλών πρακτόρων στα ψηφιακά μυρμήγκια. Αυτά είναι η μνήμη, οι περισσότεροι αισθητήρες, ο διακριτός χώρος και χρόνος, η εναπόθεση φερομόνης

σε διαφορετικές χρονικές στιγμές, (αφού τα ψηφιακά μυρμήγκια αφήνουν τη φερομόνη όταν ολοκληρώσουν τις κινήσεις τους, σε αντίθεση με τα φυσικά που την αφήνουν συνέχεια) και τέλος χρησιμοποιούνται αλγόριθμοι για τη βελτίωση της απόδοσης του συστήματος.

Κύριες μορφές αλγορίθμου:

- Ant System (AS)
- MAX-MIN Ant System (MMAS)
- Ant Colony System (ACS)
- Elitist Ant System
- Rank-based Ant System (Asrank)
- Continuous Orthogonal Ant Colony (COAC)
- Recursive Ant Colony Optimization

### *4.3.5.2 Αλγόριθμος Σμήνους Σωματιδίων*

Ο όρος νοημοσύνη σμηνών (SI=Swarm Intelligence) αναφέρεται στις τεχνικές επίλυσης προβλημάτων που χρησιμοποιούν την συλλογική συμπεριφορά ενός πληθυσμού προκειμένου να επιδείξουν κάποιας μορφής νοημοσύνη [22]. Η έμπνευση για αυτές τις μεθόδους έχει προκύψει από την παρατήρηση στην φύση ομάδων εντόμων και ζώων στις οποίες το κάθε άτομο της ομάδας υπακούει σε ένα μικρό σύνολο απλών κανόνων και το αποτέλεσμα είναι ότι παρατηρείται μια συλλογική συμπεριφορά που δρα, προς όφελος της ομάδας.

Η αλληλεπίδραση ανάμεσα στα μέλη δημιουργεί ένα πρότυπο για το σύστημα στο σύνολό του και το φαινόμενο αυτό ονομάζεται αυτό-οργάνωση (selforganization). Χαρακτηριστικά συστημάτων που δημιουργούνται με αυτό-οργάνωση είναι η κατανεμημένη τους φύση, η οργάνωσή τους σε πυκνές ετεροκρατίες και η αλληλεπίδραση με το περιβάλλον για την επικοινωνία μεταξύ των μελών της αποικίας.

Ένα κατανεμημένο σύστημα δεν έχει κάποια κεντρική αρχή που να συντονίζει τα μέλη, ενώ στα πλεονεκτήματά του συγκαταλέγονται η σταθερότητα και η ευελιξία. Η ετεροκρατία είναι σε αντίθεση με την ιεραρχία ένας οριζόντιος τρόπος οργάνωσης στον οποίο κάθε μέλος μπορεί να ανταλλάσσει πληροφορίες με κάθε άλλο μέλος.[1]

**Αλγόριθμος 12: Μέθοδος σμήνους σωματιδίων**[1]

**Input**: *Αρχικό σμήνος, $S_0$ και ταχύτητες, $t \leftarrow -1$*

1: **Θέσε** *$t \leftarrow t+1$ και $j \leftarrow 0$*

2: **Πάρε** *ένα τυχαίο πληθυσμό $P_j$,3-διάστατων διανυσμάτων, $p_l(j)$, $l=1,.....\mu$*

3: **Για καθένα** *$p_l(j)$,*

4: **Θέσε** *$w \leftarrow p_{l1}(j)$, $c_1 \leftarrow p_{l2}(j)$ και $c_2 \leftarrow p_{l3}(j)$*

5: **Υπολόγισε** *τα $u(t+1)$ και $S_{t+1}$*

6: **Αξιολόγησε** *το $S_{t+1}$. Έστω $x_g$ το καλύτερο σωματίδιο του. Πάρε ως συναρτησιακή τιμή του $p_l(j)$, την $f(x_g)$*

7: **Τέλος**

8: *Εφάρμοσε μετάλλαξη, ανασυνδυασμό και επιλογή στον $P_j$,και πάρε τον νέο πληθυσμό $P_{j+1}$. Εστω $p^*$ το καλύτερο άτομο του $P_{j+1}$.*

9: **Ελεγξε** *το κριτήριο τερματισμού του Αλγόριθμου. Αν δεν ικανοποιείται τότε*

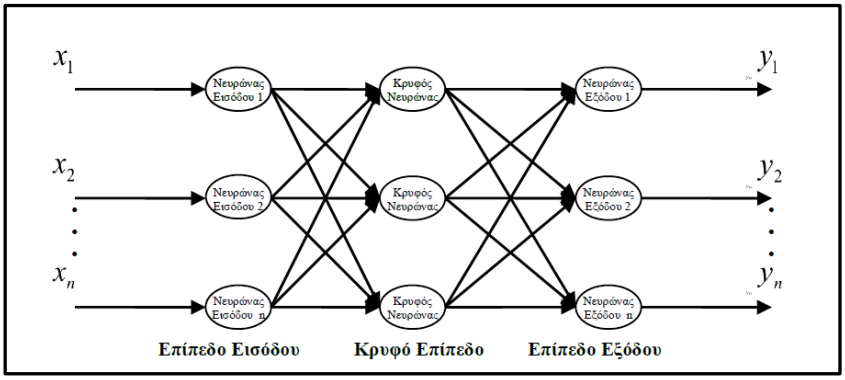
**Θεσε** $j \leftarrow j+1$,*και πήγαινε στο βήμα 3, διαφορετικά στο βήμα 10*

10: **Πάρε** $w \leftarrow p^*_1$, $c_1 \leftarrow p^*_2$ *και* $c_2 \leftarrow p^*_3$ *και* **καθόρισε** *τα* $u(t+1)$ *και* $S_{t+1}$

11: **Ελεγξε** *το κριτήριο τερματισμού του Αλγόριθμου. Αν δεν ικανοποιείται πήγαινε στο βήμα 1, διαφορετικά τερμάτισε τον αλγόριθμο***.**

### *4.3.5.3 Νευρωνικά Δίκτυα*

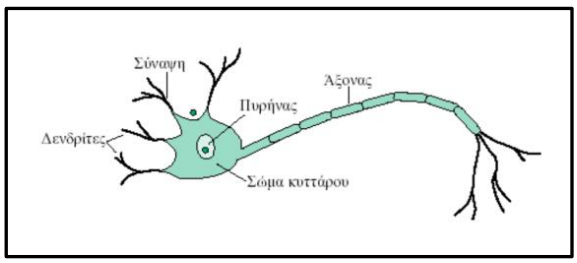
Τα τεχνητά νευρωνικά δίκτυα (Artificial Neural Networks), ή απλώς νευρωνικά δίκτυα (Neural Networks) είναι ένα μαθηματικό μοντέλο για την επεξεργασία πληροφορίας που προσεγγίζει την υπολογιστική και αναπαραστατική δυνατότητα μέσω συνάψεων. Το μοντέλο είναι εμπνευσμένο από τα βιοηλεκτρικά δίκτυα που δημιουργούνται στον εγκέφαλο ανάμεσα στους νευρώνες (νευρικά κύτταρα) και στις συνάψεις (σημεία επαφής των νευρικών απολήξεων).




*Σχήμα 26. Δίκτυο νευρώνων* [44]

Οι νευρώνες είναι τα δομικά στοιχεία του δικτύου. Υπάρχουν δύο είδη νευρώνων, οι νευρώνες εισόδου και οι υπολογιστικοί νευρώνες [44]:

- Οι νευρώνες εισόδου δεν υπολογίζουν τίποτα, μεσολαβούν ανάμεσα στις εισόδους του δικτύου και τους υπολογιστικούς νευρώνες
- Οι υπολογιστικοί νευρώνες πολλαπλασιάζουν τις εισόδους τους με τα συνοπτικά βάρη και υπολογίζουν το άθροισμα του γινομένου. Το άθροισμα που προκύπτει είναι το όρισμα της συνάρτησης μεταφοράς.




*Σχήμα 27. Δομή νευρώνα*[44]

Εάν το συνολικό άθροισμα από τις υπόλοιπες εισόδους του νευρώνα είναι μεγαλύτερο από μία ορισμένη τιμή, τότε ο νευρώνας ενεργοποιείται. Εάν είναι μικρότερο, τότε ο νευρώνας παραμένει ανενεργός**.** Λαμβάνοντας υπόψιν τον τρόπο λειτουργίας του ανθρώπινου

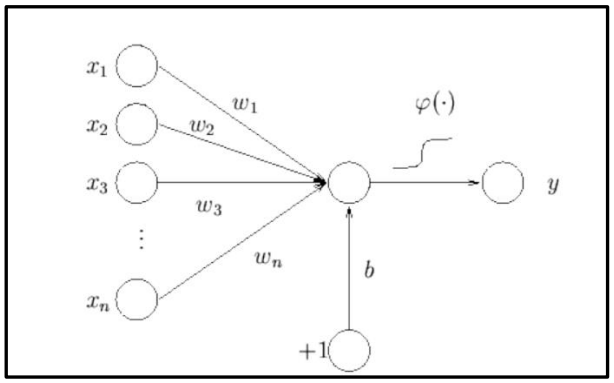
εγκεφάλου και με δεδομένο ότι τα τεχνητά νευρωσικά δίκτυα προσπαθούν να μοιάσουν σε αυτό, τα ΝΔ διαθέτουν παρόμοιες ιδιότητες όπως την ικανότητα να μαθαίνει από εμπειρίες, να γενικεύει την υπάρχουσα γνώση και να εκτελεί λογικές αφαιρέσεις.

Ένα Τ.Ν.Δ. μοιάζει με το βιολογικό στα εξής:

- Η γνώση αποκτάται από το δίκτυο μέσα από διαδικασία μάθησης
- Οι δυνάμεις σύνδεσης των νευρώνων, γνωστές σαν συνοπτικά (synaptic) βάρη, χρησιμοποιούνται για την αποθήκευση γνώσης

Γνωστοί αλγόριθμοι απόκτησης γνώσης είναι [44]:

- Ο Αλγόριθμος Μάθησης του Perceptron (Αισθητήρα)
- Ο Αλγόριθμος Ελάχιστου Μέσου Τετραγωνικού (ΕΜΤ) Σφάλματος
- Ο Αλγόριθμος Πίσω Διάδοσης (Π.Δ.) του Λάθους




*Σχήμα 28. Προσομοίωση νευρωνικών δικτύων*[44]

Ο Perceptron είναι ένας δυαδικός ταξινομητής, δηλαδή μία συνάρτηση η οποία απεικονίζει τις εισόδους $x_1 \ldots x_n$, σε μία και μοναδική δυαδική τιμή εξόδου *f(x)*

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

όπου w είναι το βάρος της συγκεκριμένης εισόδου και w*x είναι το εσωτερικό γινόμενο μεταξύ τιμής εισόδου και βάρους. Το (b) είναι η πόλωση ή κατώφλι για την τελική απόφαση για την τιμή της εξόδου. Το (b), ένας σταθερός όρος ο οποίος δεν εξαρτάται από καμία τιμή εισόδου.

Η τιμή εξόδου της *f(x)* (0 ή 1) χρησιμοποιείται για να ταξινομήσει τις εισόδους $x_n$ είτε ως θετικό ή αρνητικό στιγμιότυπο, στην περίπτωση ενός δυαδικού προβλήματος ταξινόμησης. Το (b) χρησιμοποιείται για την μετατόπιση της συνάρτησης ενεργοποίησης ή για να δώσει στον νευρώνα εξόδου ένα βασικό επίπεδο τιμής.

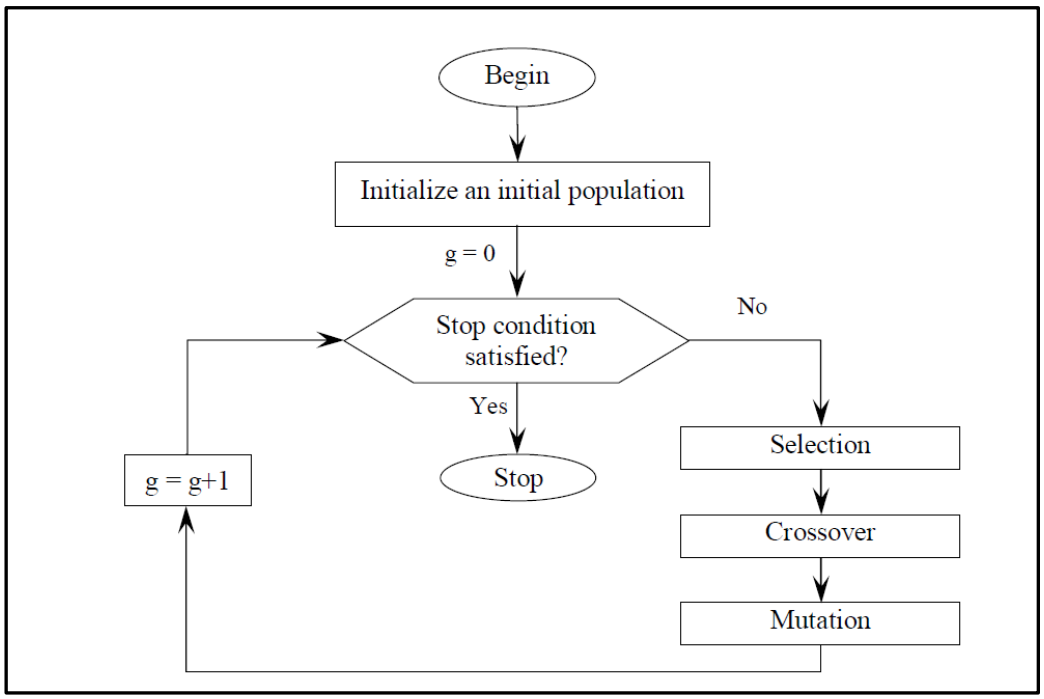
### *4.3.5.4 Γενετικοί αλγόριθμοι*

Όπως σε όλους τους εξελικτικούς αλγορίθμους, ο Γενετικός Αλγόριθμος (GA – Genetic Algorithm) αρχίζει τη λειτουργεία του με έναν αρχικό πληθυσμό. Κάθε άτομο σε αυτό τον πληθυσμό ονομάζεται χρωμόσωμα. Κάθε χρωμόσωμα, αξιολογείται χρησιμοποιώντας μια

λειτουργία φυσικής κατάστασης και ανάθεσης μιας αξίας. Αυτή η τιμή αξιολόγησης, σχετίζεται με την αντικειμενική τιμή λειτουργίας του προβλήματος.

Ο αλγόριθμος επιλέγει τα καλύτερα χρωμοσώματα, μεταξύ του πληθυσμού και απορρίπτει τα χειρότερα με τον κατάλληλο κανόνα επιλογής. Η έξοδος της επιλογής είναι ένας ενδιάμεσος πληθυσμός. Μετά την ολοκλήρωση της επιλογής, ο ενδιάμεσος πληθυσμός ανανεώνεται διασταυρώνοντας ή μεταλλάσσοντας δυο τυχαία χρωμοσώματα και αφού τα αναπαράξει δημιουργεί ένα νέο πληθυσμό. Η μετάλλαξη προκαλεί τη διαφυγή GA από τοπικά βέλτιστα.

Σε κάθε κύκλο διασταύρωσης και μετάλλαξης ο αλγόριθμος δημιουργεί μια νέα γενιά. Αυτό συμβαίνει έως ότου εξασφαλισθούν οι όροι τερματισμού. Ένα διάγραμμα ροής του GA παρουσιάζεται στο σχήμα 29.




*Σχήμα 29. Διάγραμμα ροής γενετικού αλγόριθμου*

### *4.3.5.5 Μιμητικοί αλγόριθμοι*

Ως μιμήδιο (meme) μπορεί να οριστεί οποιαδήποτε ιδέα ή πρότυπο η οποία περνάει από άτομο σε άτομο χωρίς να χρησιμοποιούνται γενετικοί μηχανισμοί Η αρχική αναφορά στον όρο meme έγινε από τον Dawkins το 1976 προκειμένου να περιγραφεί ο τρόπος με τον οποίο τρόποι συμπεριφοράς μεταδίδονται στον πληθυσμό μιας κοινότητας κατ' αναλογία με τον τρόπο διάδοσης των ιών.

Οι μιμητικοί αλγόριθμοι (MA=Memetic Algorithms) ως εξελικτικοί αλγόριθμοι διατηρούν ένα πληθυσμό από λύσεις αλλά επιπλέον χρησιμοποιούν μια φάση τοπικής αναζήτησης η οποία εφαρμόζεται είτε μετά από την μετάλλαξη και τον συνδυασμό των λύσεων είτε είναι ενσωματωμένη στην ίδια την διαδικασία ανασυνδυασμού λύσεων. Κατά τον σχεδιασμό του τρόπου ανασυνδυασμού λύσεων θα πρέπει να αντιμετωπιστεί το πρόβλημα της δημιουργίας λύσεων που δεν είναι εφικτές.

Δύο τεχνικές χειρισμού του εν λόγω προβλήματος χρησιμοποιούνται συνήθως: είτε ένας μηχανισμός επισκευής της λύσης, είτε σχεδίαση των τελεστών συνδυασμού λύσεων έτσι ώστε να επιστρέφουν μόνο έγκυρες λύσεις. Θα πρέπει ωστόσο να σημειωθεί ότι ειδικά στους μιμητικούς αλγορίθμους η έννοια του ανασυνδυασμού λύσεων είναι υποβαθμισμένη σε σχέση με ότι ισχύει στους γενετικούς αλγορίθμους ενώ αντίθετα ο ρόλος της μετάλλαξης εμφανίζεται ισχυροποιημένος. Ως μέθοδος για προβλήματα βελτιστοποίησης έχει δώσει καλά αποτελέσματα σε πλειάδα προβλημάτων ενώ αναφέρεται στην βιβλιογραφία και ως Genetic Local Search. [22].

# 5. Γνωστά προβλήματα

Από την σύντομη περιγραφή ορισμένων μεθόδων για την δημιουργία των αλγορίθμων, που είδαμε στο προηγούμενο κεφάλαιο, φαίνεται ότι δεν υπάρχει κατηγορία μεθόδων που να μπορεί να αντιμετωπίσει επιτυχώς όλα τα προβλήματα βελτιστοποίησης. Οι περισσότεροι αλγόριθμοι είναι αποδοτικοί μόνο σε κάποιες κατηγορίες προβλημάτων, συνήθως με συγκεκριμένα χαρακτηριστικά και προϋποθέσεις, ενώ όταν αυτές δεν ισχύουν τότε η απόδοσή τους μετριάζεται έντονα. Έτσι ενώ οι υπάρχουσες μέθοδοι έχουν κερδίσει την εμπιστοσύνη της επιστημονικής κοινότητας, υπάρχουν πάντα περιθώρια για την ανάπτυξη νέων μεθόδων, οι οποίες βασίζονται σε καινοτόμες ιδέες και μπορούν να αντιμετωπίσουν αποδοτικά δύσκολα προβλήματα.

Ένα από τα γνωστότερα θεωρήματα που αντικατοπτρίζει τα παραπάνω είναι το θεώρημα *No Free Lunch* των Wolpert και Macready σύμφωνα με το οποίο όλοι οι αλγόριθμοι είναι ισοδύναμοι αν δοκιμαστούν όλοι στον ίδιο χώρο αναζήτησης.

"*αν δοθεί ένας αλγόριθμος Α, ο οποίος υπερτερεί έναντι κάποιου αλγόριθμου Β σε ένα σύνολο προβλημάτων, τότε υπάρχουν άλλα τόσα προβλήματα όπου ο αλγόριθμος Β υπερτερεί του αλγόριθμου Α*" [1]

Οι Αλγόριθμοι Βελτιστοποίησης, συνήθως ξεκινούν με το δεδομένο ότι υπάρχει μία λύση για το πρόβλημα και προσπαθούν να την βελτιώσουν με τη χρήση τεχνικών βελτιστοποίησης και για αυτό το λόγο η πολυπλοκότητα τους εξαρτάται από τον κατασκευαστικό αλγόριθμο της αρχικής λύσης.

Στην συνέχεια θα γνωρίσουμε εννέα προβλήματα που έχουν πολλές εφαρμογές στην καθημερινότητα μας και θα παρουσιάσουμε τρόπους που έχουν εφευρεθεί για να βελτιστοποιήσουμε τα αποτελέσματα και την χρησιμότητά τους .

## 5.1 Πρόβλημα περιοδεύοντας πωλητή (TSP - Traveling Salesman Problem)

Το πρόβλημα του Περιοδεύοντος Πωλητή (TSP) αναφέρεται σε ένα πωλητή ο οποίος ξεκινάει μια περιοδεία σε διάφορες πόλεις με σκοπό να πουλήσει τα προϊόντα του. Θα πραγματοποιήσει το ταξίδι του με στόχο να επισκεφτεί την κάθε πόλη ακριβώς μια φορά, πριν την επιστροφή στο σπίτι του. Δεδομένων των αποστάσεων μεταξύ των πόλεων μας ζητείται η εύρεση της σειράς με την οποία θα επισκεφτεί τις εν λόγω πόλεις με γνώμονα την ελάχιστη διανυθείσα διαδρομή.

Αν με Cij συμβολίσουμε το κόστος του ταξιδιού από την πόλη i στην πόλη j, το οποίο θεωρούμε ότι ισούται με τη γεωγραφική απόσταση των πόλεων, τότε το συνολικό μήκος της διαδρομής, το οποίο πρέπει να ελαχιστοποιηθεί είναι :

$$l(\pi) = \sum_{i=1}^{n-1} c_{\pi(i),\pi(i+1)} + c_{\pi(n),\pi(1)} \qquad [45]$$

όπου π είναι το σύνολο *{1,2,3……. n}* των πόλεων που πρέπει να επισκεφθεί.

Υπάρχουν διάφορες παραλλαγές του συγκεκριμένου προβλήματος. Σε ένα συμμετρικό πρόβλημα TSP για κάθε *i,j* ισχύει ότι *Cij=Cji*. Σε διαφορετική περίπτωση, το πρόβλημα

λέγεται ασύμμετρο. Αν οι πόλεις βρίσκονται εντός ενός μετρικού χώρου ικανοποιώντας την τριγωνική ανισότητα (σε ένα τρίγωνο, το μήκος κάθε πλευράς είναι μικρότερο από το άθροισμα των μηκών των άλλων δύο πλευρών, καθώς και μεγαλύτερο από τη διαφορά τους), τότε το πρόβλημα λέγεται μετρικό.

Αν με $(x_i, y_i)$ συμβολίσουμε τις συντεταγμένες της κάθε πόλης και για κάθε κόστος ταξιδιού ισχύει:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

τότε το πρόβλημα είναι και συμμετρικό και μετρικό.[45]

Είναι προφανές ότι όσο αυξάνονται οι πόλεις που πρέπει να επισκεφθεί ο πωλητής τόσο δυσκολότερη γίνεται η εύρεση της συντομότερης διαδρομής. Για να είμαστε ακριβείς, το πρόβλημα του περιοδεύοντος πωλητή αποτελεί ένα από τα πιο περιβόητα υπολογιστικά προβλήματα και αποτελεί δύσκολη διαδικασία ακόμα και για τους Η/Υ. Υπήρξαν αρκετές προσπάθειες επίλυσης του, πολλές αποτυχημένες και κάποιες μερικώς επιτυχημένες, κατά την πορεία της αλματώδους προόδου των αλγορίθμων και της θεωρίας της πολυπλοκότητας. Η πιο δυσάρεστη είδηση για το πρόβλημα του περιοδεύοντος πωλητή είναι ότι μάλλον είναι αδύνατο να μπορέσει να λυθεί σε πολυωνυμικό χρόνο.

Χρησιμοποιώντας την μέθοδο brute force για να αξιολογήσουμε κάθε πιθανή διαδρομή ώστε να βρούμε την βέλτιστη καταλήγουμε στο συμπέρασμα ότι από την στιγμή που υπάρχουν $(n - 1)!$ ενδεχόμενα, με αυτήν την μέθοδο απαιτείται $O(n!)$ χρόνος επίλυσης.[46]

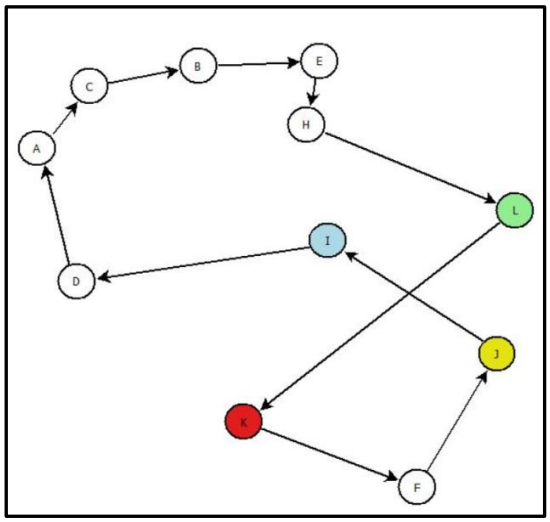
### 5.1.1 2-opt

Ο απλούστερος βελτιωτικός αλγόριθμος παρουσιάστηκε το 1958 από τον Croes και έχει σαν βασική ιδέα την παρατήρηση ότι συνήθως δύο ακμές που διασταυρώνονται σε μια περιοδεία μπορούν να αντικατασταθούν από δύο άλλες με μικρότερο κόστος.

Αρχικά, επιλέγεται ένα μονοπάτι που ικανοποιεί τις προϋποθέσεις του προβλήματος, δηλαδή ξεκινάει από μια αρχική πόλη, περνάει από όλες τις πόλεις ακριβώς μια φορά και αφού τις διασχίσει όλες καταλήγει πάλι στην αρχική πόλη. Το μονοπάτι αυτό μπορεί να είναι ένα τυχαίο, σε περίπτωση που ο αλγόριθμος 2-opt εφαρμόζεται αποκλειστικά, ή ένα ήδη υπολογισμένο καλό μονοπάτι σε περίπτωση που συνδυάζεται με μια ήδη υπολογισμένη καλή λύση από άλλον αλγόριθμο (π.χ. έναν άπληστο αλγόριθμο). Γενικά στην περίπτωση που έχουμε ήδη ένα καλό μονοπάτι ο αλγόριθμος 2-opt, βρίσκει καλές λύσεις πολύ πιο γρήγορα από ότι αν ξεκινήσει με μια τυχαία επιλογή διαδρομής.

Στη συνέχεια, ο αλγόριθμος 2-opt εξετάζει όλες τις ακμές μιας περιοδείας, ανά δύο χωρίς κοινά σημεία, και ελέγχει αν η αντικατάστασή κάθε ζεύγους ακμών, από δύο άλλες, που συνδέουν τις αντίστοιχές κορυφές εναλλάξ, έχει μικρότερο κόστος. Με την ολοκλήρωση του αλγορίθμου ονομάζουμε τη νέα περιοδεία 2-optimal.

Η λειτουργία αυτή, απεικονίζεται στο παρακάτω σχήμα, όπου εξασφαλίζουμε μια καλύτερη λύση αν οι ακμές I-J και K-L αντικαθίστανται από τις ακμές I-K και J-L. [34]




*Σχήμα 30. Αρχική περιοδεία πωλητή*

Δυστυχώς δεν υπάρχει εύκολος τρόπος να υπολογιστεί το πλήθος των πράξεων που απαιτούνται για να βρεθεί μια καλή λύση με την προσέγγιση αυτή. Θεωρητικά αν ο αλγόριθμος αφεθεί να τρέξει θα καταλήξει να δοκιμάσει όλες τις δυνατές εναλλαγές ανά ζεύγη αλλά αυτό θα απαιτήσει πολυωνυμικό χρόνο κάνοντας την μέθοδο αυτή άχρηστη. Ωστόσο, με μικρότερο χρόνο εκτέλεσης προκύπτουν επαρκώς καλές λύσεις. Ο αλγόριθμος έχει τάξη πολυπλοκότητας $O(n^2)$.

Το 1990 ο Bentley παρουσίασε την «fast 2-opt» μέθοδο χρησιμοποιώντας δενδρικές δομές για να βρει πιθανά ζευγάρια για βελτίωση. Η μέθοδος του δεν συγκρίνει ακμές, αλλά ερευνά τις κορυφές, αναζητώντας τους πλησιέστερους γείτονες που μπορεί να χρησιμοποιήσει για βελτίωση. [34]

### 5.1.2 3-opt

Ο αλγόριθμος 3-opt προτάθηκε από τους Bock και Lin και σε αντιστοιχία με τον 2-opt εξετάζει όλες τις ακμές μιας περιοδείας, ανά τρείς χωρίς κοινά σημεία, και ελέγχει αν η αντικατάστασή τους από τρείς άλλες, που συνδέουν τις αντίστοιχές κορυφές εναλλάξ, έχει μικρότερο κόστος.

Κάθε μία αντικατάσταση 3-opt μπορούμε να την δούμε και σαν δύο διαδοχικές 2-opt αντικαταστάσεις και για αυτό μια 3-optimal περιοδεία είναι και 2-optimal. Ο αλγόριθμος έχει τάξη πολυπλοκότητας $O(n^3)$

### 5.1.3 k-opt

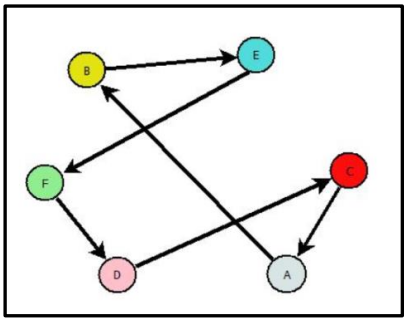
Η γενίκευση των δύο προηγούμενων αλγορίθμων είναι η μέθοδος k-opt όπου επιλέγουμε να αντικαταστήσουμε k>3 ακμές του προβλήματος ερευνώντας για μικρότερου κόστους περιοδείες. Στην γενική περίπτωση ο αλγόριθμος έχει τάξη πολυπλοκότητας $O(n^k)$.

Αν και θεωρητικά η αύξηση του k θα έπρεπε να αποδώσει καλύτερες περιοδείες, η μεγάλη πολυπλοκότητα που επιφέρει, συνδυασμένη με την μη ύπαρξη κάποιου ποιοτικού φράγματος όπως είδαμε, οδηγούν συνήθως στην χρήση των μεθόδων 2-opt και 3-opt.

### 5.1.4 V-opt Lin-Kernighan

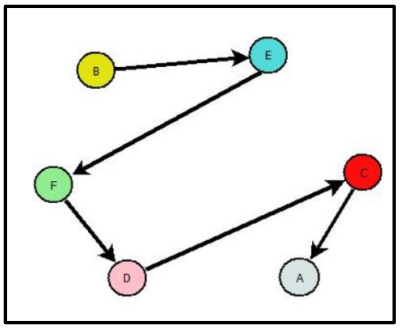
Οι Lin και Kernigham παρουσίασαν έναν αλγόριθμο βελτιστοποίησης το 1972 [10] βασισμένο στην τεχνική k-opt αλλά με κεντρική ιδέα τη δυναμική επιλογή αριθμού ακμών για αντικατάσταση. Ο αλγόριθμος δεν έχει σταθερό k, όπως ο k-opt, αλλά μεταβλητό (variable, v-opt), το οποίο και αποφασίζει σε κάθε βήμα. Στην γενική περίπτωση ο αλγόριθμος έχει τάξη πολυπλοκότητας $O(n^2)$ και αποδίδει περιοδείες κατά μέσο όρο 1-2% μεγαλύτερες της άριστης. Τα βήματα του αλγορίθμου είναι τα εξής:

1. Επιλεγούμε την αρχική περιοδεία. (A→B→E→F→D→Γ→A),



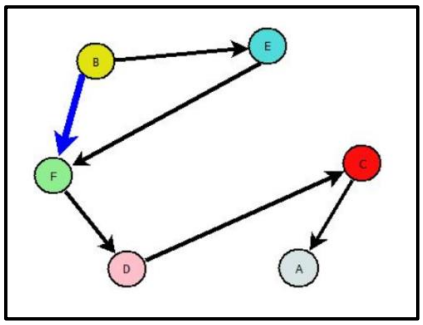

*Σχήμα 31. Αρχική περιοδεία*

2. Επιλέγουμε την αρχική πόλη A της περιοδείας μας και μία ακμή για αντικατάσταση, από τις δύο της περιοδείας που προσπίπτουν (AB, AC) σε αυτή, έστω την AB,



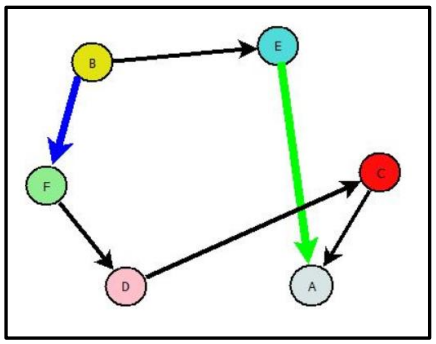

*Σχήμα 32. Αφαίρεση της διαδρομής AB*

3. Από την δεύτερη πόλη B επιλέγουμε μια ακμή που δεν ανήκει στην αρχική περιοδεία, έστω την BF. Η ακμή επιλέγεται με κριτήριο τη μεγιστοποίηση του κέρδους, δηλαδή η ακμή BF να έχει το μικρότερο κόστος από όλες τις προσπίπτουσες ακμές στην πόλη B που δεν συμμετέχουν στην αρχική περιοδεία,




*Σχήμα 33. Πρόσθεση της διαδρομής BF*

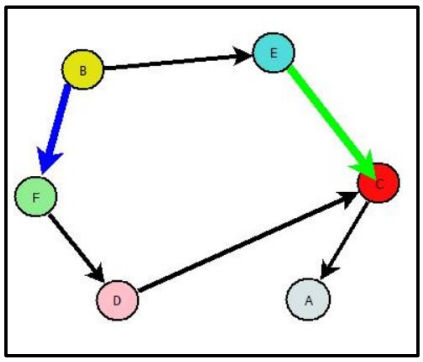
4. Στο προηγούμενα βήματα, επιλέξαμε να αφαιρέσουμε την AB από την περιοδεία και να προσθέσουμε την BF, επομένως η επόμενη ακμή που θα αφαιρεθεί πρέπει να είναι η μία από τις δύο ακμές που προσπίπτουν στην F και συμμετέχουν στην αρχική περιοδεία, της οποίας η αφαίρεση αφήνει τις πόλεις συνδεδεμένες,




*Σχήμα 34. Πρόσθεση της διαδρομής EA*

- Στο παράδειγμα μας η επιλέγεται να αφαιρεθεί η FE καθώς η αφαίρεση της FD αφήνει τις πόλεις ασύνδετες
- Για να έχουμε ξανά μια πλήρη περιοδεία προσθέτουμε την EA

5. Η ακμή EA που προσθέσαμε για να έχουμε ξανά περιοδεία δεν είναι κατ' ανάγκην αυτή που θα επιλέξουμε τελικά. Πρέπει να βρούμε την πόλη που μεγιστοποιεί το κέρδος. Έστω ότι η πόλη αυτή είναι η C και προσθέτουμε την ακμή EC και επαναϋπολογίζουμε το κέρδος,




*Σχήμα 35. Αναζήτηση καλύτερης διαδρομής*

6. Επαναλαμβάνουμε το βήμα 5 έως ότου:
   - Δεν υπάρχει άλλη εφικτή αντικατάσταση
   - Έχουμε μια νέα περιοδεία.
   - Έχουμε το μικρότερο κόστος

7. Ολόκληρη η διαδικασία επαναλαμβάνεται ώστε κάθε πόλη να χρησιμοποιηθεί σαν αρχική πόλη και να μην υπάρχει άλλη εφικτή αντικατάσταση.

Για να πετύχουν έναν αποδοτικό αλγόριθμο οι Lin και Kernighan έθεσαν τα εξής κριτήρια για την επιλογή των ακμών προς αντικατάσταση:

- Το κριτήριο της διαδοχικής αντικατάστασης: Μία ακμή που αντικαθιστά μια άλλη πρέπει να έχει μία κοινή κορυφή με αυτή που θα αντικαταστήσει. Η συνθήκη αυτή είναι αναγκαία, αλλά όχι ικανή, ώστε μετά τις αντικαταστάσεις να έχουμε πάλι μια περιοδεία. Γενικά η βελτίωση μιας περιοδείας μπορεί να γίνει με μια σειρά διαδοχικών αντικαταστάσεων 2-opt και 3-opt.
- Το κριτήριο της εφικτότητας: Απαιτούμε, μετά την αντικατάσταση των ακμών να σχηματίζεται περιοδεία. Το κριτήριο αυτό εξασφαλίζει την δημιουργία περιοδείας και ουσιαστικά επιτρέπει να γίνουν μόνο k-opt αντικαταστάσεις που είναι σειρά διαδοχικών 2-opt και 3-opt.
- Το κριτήριο του κέρδους: Η αντικατάσταση των ακμών επιλέγεται με κριτήριο το συνολικό κέρδος και αυτό να παραμένει θετικό. Το κριτήριο αυτό θεωρείται το σημαντικότερο για την απόδοση του αλγορίθμου.
- Το κριτήριο του διαχωρισμού των ακμών: Το σύνολο των ακμών που φεύγουν από την αρχική περιοδεία με το σύνολο των ακμών που μπαίνουν στην περιοδεία πρέπει να είναι ξένα μεταξύ τους.
- Το κριτήριο του υποψήφιου συνόλου: Οι Lin και Kernighan πρότειναν να περιοριστεί η αναζήτηση για υποψήφια προς ένταξη ακμή σε αυτές των πέντε κοντινότερων γειτόνων (nearest neighborhood) της κάθε πόλης.

## 5.2 Πρόβλημα ικανοποιησιμότητας (SAT - Satisfiability Problem)

Το πρόβλημα ικανοποιησιμότητας (SAT), είναι το πρόβλημα του προσδιορισμού της εξόδου μιας Boolean συνάρτησης με *n* μεταβλητές και η εύρεση της τιμής της κάθε μεταβλητής (αληθής ή ψευδής), που καθιστά την συνάρτηση αληθή (TRUE).

Το πρόβλημα της ικανοποιησιμότητας ανήκει σε μια σημαντική τάξη διακριτών προβλημάτων εκπλήρωσης περιορισμών (CSP- Constraint Satisfaction Problems).

Το SAT είναι το πρώτο πρόβλημα που αποδείχθηκε ότι ήταν NP-πλήρης (θεώρημα Cook - Levin). Αυτό σημαίνει ότι όλα τα προβλήματα στην τάξη πολυπλοκότητας NP, που περιλαμβάνει ένα ευρύ φάσμα φυσικών προβλημάτων λήψης αποφάσεων και βελτιστοποίησης, είναι εξίσου δύσκολα να λυθούν όπως το SAT. Δεν υπάρχει γνωστός αλγόριθμος που να επιλύει αποτελεσματικά κάθε πρόβλημα SAT και ενώ γενικά πιστεύεται ότι δεν υπάρχει τέτοιος αλγόριθμος, αν και αυτό δεν έχει αποδειχθεί μαθηματικά.

Εντούτοις, από το 2007, οι ευρετικοί αλγόριθμοι SAT μπορούν να επιλύσουν προβλήματα που αφορούν δεκάδες χιλιάδες μεταβλητές και τύπους που αποτελούνται από εκατομμύρια σύμβολα, που επαρκούν για πολλά πρακτικά προβλήματα SAT, όπως για παράδειγμα την τεχνητή νοημοσύνη, τον σχεδιασμό κυκλώματος, και άλλα.

Παρακάτω θα παρουσιάσουμε δύο βασικούς αλγόριθμους SAT. Αυτοί είναι ο Walk SAT και ο DPLL .

Οι BOOLEAN εκφράσεις, συνήθως παρουσιάζονται στη μορφή CNF **(**conjunctive normal form - συνηθισμένη φυσιολογική μορφή) δηλαδή η αναπαράσταση όλων των όρων της έκφρασης με μικρότερους όρους, οι οποίοι συνδέονται μεταξύ τους με “BOOLEAN OR” και ενώνονται με “BOOLEAN AND”. Έχει αποδειχθεί μάλιστα ότι οποιαδήποτε φόρμουλα

BOOLEAN μπορεί να εκφραστεί σε μορφή CNF. Επίσης έχει αποδειχθεί ότι οι μικρότεροι όροι που βρίσκονται στην έκφραση δεν χρειάζονται πάνω από 3 μεταβλητές για να εκφράσουν οποιαδήποτε έκφραση BOOLEAN.

Η επίλυση μιας BOOLEAN έκφρασης είναι στην ουσία ένα πρόβλημα αναζήτησης. Ως τέτοιο μπορούμε να εφαρμόσουμε διάφορες τεχνικές και αλγορίθμους που χρησιμοποιούνται γενικότερα στα προβλήματα αναζήτησης. Δύο τέτοιες κλάσεις αλγορίθμων αναζήτησης που χρησιμοποιούνται στο SAT είναι ο αλγόριθμος αναζήτησης με οπισθοδρόμηση (backtracking) και ο αλγόριθμος τοπικής αναζήτησης (local search).[47]

### 5.2.1 Αλγόριθμος WalkSAT

Ο αλγόριθμος WalkSAT θεωρείται αλγόριθμος τοπικής αναζήτησης δηλαδή δεν κρατάει ιστορικό αναζήτησης όταν ψάχνει για μια ικανοποιητική ανάθεση τιμών στις μεταβλητές. Σαν αποτέλεσμα των ανωτέρω ο αλγόριθμος αυτός δεν είναι σε θέση να γνωρίζει πότε έχει ολοκληρωθεί η αναζήτηση στο διάστημα έρευνας. Ακολουθεί ο ψευδοκώδικας

**Αλγόριθμος 13: WALKSAT***(clauses; p; maxFlips)*[47]

**Input:** *clauses, //*a set of clauses in propositional logic//
*p,* //the probability of choosing a random walk step//
*maxFlips,* //number of flips allowed before giving up//
*model,* // a random assignment of true or false to the symbols in clauses//

*1:* **for** i=1 **to** *maxFlips* **do**
*2:* ***if*** *model* ***satisfies*** *clauses* ***then return*** *model)*
*3:* *clause* = a randomly selected clause from *clauses* that is false
*4:* **with probability** p flip a random symbol from the clause
*5:* **else** flip the symbol in *clause* to maximize the num. of satisfied clauses
6: **return** failure

Όπως φαίνεται υπάρχουν 2 παράμετροι οι οποίες μπορούν να ρυθμιστούν επηρεάζοντας τις επιδόσεις του αλγορίθμου. Η πρώτη είναι η ***maxFlips***, δηλαδή για πόσα βήματα θα τρέχει ο αλγόριθμος. Εάν στις προβλεπόμενες επαναλήψεις δεν βρεθεί λύση τότε ο αλγόριθμος σταματάει. Η δεύτερη παράμετρος είναι η ***p*** δηλαδή η πιθανότητα με την οποία επιλέγουμε να αντιστρέψουμε ένα σύμβολο ενός μη-ικανοποιημένου υπό όρου της έκφρασης μας (δηλαδή ενός υπό-όρου που δεν είναι TRUE) σε αντίθεση με την ευρετική λογική μεγιστοποίησης των ικανοποιημένων υπο-όρων που ακολουθεί μετά το else.

Ο παραπάνω αλγόριθμος παρουσιάζει κάποια ενδιαφέροντα χαρακτηριστικά. Πρώτα από όλα, λόγω της τυχαιότητας μετριάζεται η πιθανότητα να κολλήσει σε κάποιο τοπικό ακρότατο. Επίσης, επειδή τρέχει για προκαθορισμένο αριθμό επαναλήψεων είμαστε σίγουροι ότι ο αλγόριθμος μας πάντα θα τερματίζει. Αυτό με τη σειρά του σημαίνει, ότι μπορούμε να οριοθετήσουμε το χρόνο εκτέλεσης. Ακόμη, επειδή η ευρετική μέθοδος είναι αποτελεσματική ο αλγόριθμος συγκλίνει γρήγορα προς την λύση.

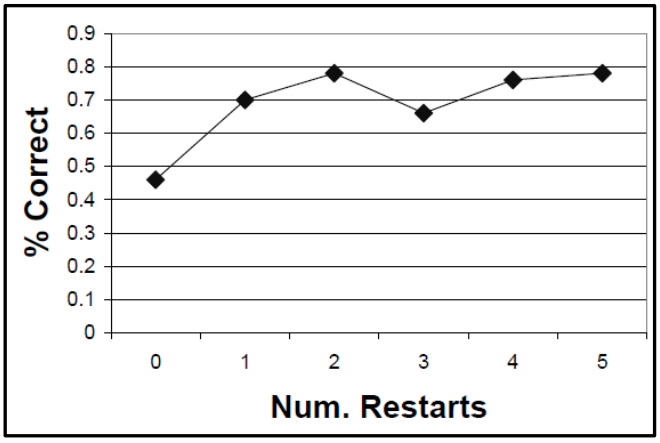
Ενώ βέβαια ο WalkSAT έχει μερικά ωραία χαρακτηριστικά έχει και κάποια μειονεκτήματα. Πιο σημαντικό είναι το γεγονός ότι δεν είναι ολοκληρωμένος, δηλαδή όταν ο αλγόριθμος μας επιστρέφει μια ανάθεση τιμών για την οποία η έκφραση μας είναι TRUE,

όταν αποτυγχάνει δεν είμαστε σίγουροι εάν απέτυχε διότι η έκφραση μας είναι μη ικανοποιήσιμη ή διότι δεν έτρεξε αρκετό χρόνο (maxFlips).

Από τα παραπάνω γίνεται αντιληπτό, ότι η αρχική τυχαία απόδοση τιμών μπορεί να έχει μεγάλο αντίκτυπο στην προσπάθεια του αλγορίθμου να βρει λύση. Λαμβάνοντας αυτό υπόψιν, οι ερευνητές κατέληξαν, πως η απόδοση του αλγορίθμου μπορεί να βελτιωθεί επανεκκινώντας τον WalkSAT σε διάφορους χρόνους και δημιούργησαν 3 διαδικασίες βελτιστοποίησης επανεκκίνησης:

- **Retry**: Εάν ο αλγόριθμος αποτύχει κάνε επανεκκίνηση ***n*** φορές
- **Fixed Time Restart**: Εάν ο αλγόριθμος τρέξει για κάποιο διάστημα χωρίς να βρει λύση κάνε restart
- **Randomized Restart**: Μετά από κάθε επανάληψη υπάρχει πιθανότητα να κάνεις restart. Όσο πιο πολύ χρόνο τρέχει ο αλγόριθμος τόσο μεγαλύτερη η πιθανότητα[47]

Εφαρμόζοντας τον βασικό μηχανισμό, Retry, η εργασία των Ankur, Madhyastha, & Prince[47], μέτρησε το πως επηρεάζει αυτός, την απόδοση του WalkSAT. Στο σχήμα 35 φαίνεται η ακρίβεια του αλγορίθμου σαν συνάρτηση των επανεκκίνησεων που έγιναν. Το γράφημα έγινε ώστε ο αριθμός των flips να είναι σταθερός όποτε και αν εφαρμόζεται ο WalkSAT. Έτσι για την περίπτωση με 0 επανεκκινήσεις έχουμε maxFlips 800, για την περίπτωση με 1 επανεκκίνηση, 400 maxFlips (ώστε συνολικά να είναι 800). Τα αποτελέσματα με 1 επανεκκίνηση είναι εντυπωσιακά. Αλλά παρατηρούμε ότι δεν υπάρχει βελτίωση όταν αυξάνονται τα restart.




*Σχήμα 35. Αποτελέσματα βελτιστοποίησης WalkSAT*

Συμπερασματικά μπορούμε να πούμε ότι ο WalkSAT είναι πολύ καλός όταν μπορεί να βρει τη λύση. Επειδή όμως δεν είναι πλήρης έχει περιορισμένη χρήση [47].

### 5.2.2 Αλγόριθμος DPLL

Ο δεύτερος αλγόριθμος που θα ασχοληθούμε βασίζεται στον αλγόριθμο Davis-Putnam-Logemann-Loveland (DPLL). Πραγματοποιεί αρχικά, μια σε βάθος αναζήτηση του δέντρου απόδοσης τιμών. Σε κάθε βήμα αντί να επιλέγει ένα τυχαίο σύμβολο του υπό-όρου (clause) και να του δίνει τυχαία μια τιμή, χρησιμοποιούνται ευρετικές μέθοδοι, για να επιλέξει το λεκτικό στο οποίο θα το αντιστοιχήσει. Με τον όρο λεκτικό (literal) εννοούμε έναν

συμβολισμό που αντιπροσωπεύει μια σταθερή τιμή. Υπάρχουν 2 βασικές ευρετικές μέθοδοι που χρησιμοποιούνται.

- Η πρώτη ονομάζεται Αμιγή Σύμβολα (pure symbol).Η βασική ιδέα πίσω από την μέθοδο βασίζεται στο γεγονός ότι εάν ένα σύμβολο (μεταβλητή) ενός υπό-όρου (clause) εμφανίζεται με τον ίδιο τρόπο σε όλους τους υπό-όρους της έκφρασης τότε μπορεί σε αυτό το σύμβολο να αποδοθεί η τιμή TRUE. Αυτή η ανάθεση βασίζεται στο γεγονός ότι σε οποιονδήποτε υπό-όρο εμφανίζεται το ίδιο σύμβολο δεν δημιουργεί πρόβλημα δεδομένου ότι τα σύμβολα εντός του clause συνδέονται με σχέση OR.
- Η δεύτερη ονομάζεται Μοναδιαίες Διαζευκτικές Προτάσεις (Unit Clause). Σε αυτή τη μέθοδο ο DPLL ελέγχει εαν υπάρχει κάποιος υπό-όρος με μια μόνο μεταβλητή στην οποία δεν έχει αποδοθεί καμία τιμή. Εάν υπάρχει τέτοιος υπό-όρος τότε ικανοποιώντας τη μοναδική μεταβλητή μπορούμε να ικανοποιήσουμε ταυτόχρονα και τον υπό-όρο.

Εκτός από τις παραπάνω ευρετικές μεθόδους για την ανάθεση τιμής σε μια μεταβλητή ο DPLL χρησιμοποιεί κάποια τεστ για να "κλαδέψει" κάποια κλαδιά εάν καταλάβει ότι πηγαίνοντας προς τα κάτω στο δέντρο δεν υπάρχει πιθανότητα να βρεθεί λύση.

Ο πρώτος έλεγχος γίνεται για να δει εάν όλοι οι όροι έχουν ικανοποιηθεί. Εάν ναι, τότε θεωρεί ότι έχει γίνει μια ικανοποιητική ανάθεση και δεν χρειάζεται να συνεχίσει.

Επιπλέον, ελέγχει, εάν υπάρχει υπό-όρος στον οποίο έχει γίνει ανάθεση τιμής σε όλες τις μεταβλητές, αλλά είναι FALSE. Είναι λοιπόν ανώφελο να συνεχίσει να ψάχνουμε σε αυτό το μονοπάτι και ξεκινάει η διαδικασία backtrack. [47]

**Αλγόριθμος 14: DPLL** [47]

```
   Input: clauses; symbols; model
1:    if (all clauses are satisfied)
2:          return TRUE;
3:    if (some clause is false)
4:          return FALSE;
5:    (P, v) = FIND PURE SYMBOL(clauses, symbols, model)
6:    if (P not equal to NIL)
8:          return DPLL(clauses, symbol, EXTEND(model, P, v))
9:    (P; v) = FIND UNIT CLAUSE(clauses, symbols, model)
10:   if (P not equal to NIL)
11:         return dpll(clauses, symbol, EXTEND(model, P, v));
12:   P = PICK UNASSIGNED SYMBOL(symbols, model)
13:   Heuristically pick some value v for P
14:   if (DPLL(clauses, symbols, EXTEND(model, P, v)) == TRUE)
15:         return TRUE
16:    return DPLL(clauses, symbols, EXTEND(model, P, ~v));
```

Στον παραπάνω κώδικα μπορεί να γίνει παρέμβαση μόνο στα παρακάτω:

- Στην αρχική επιλογή του αμιγή σύμβολου, όταν υπάρχει πάνω από ένα

- Στην αρχική επιλογή του μοναδιαίου υπό-όρου όταν υπάρχει πάνω από ένας
- Στην επιλογή σύμβολου, στο οποίο δεν έχει γίνει ανάθεση και να του ανατεθεί μια τιμή, όταν δεν υπάρχει αμιγές σύμβολο και μοναδιαίος υπό-όρος

Δεν υπάρχει κανένα πλεονέκτημα στην επιλογή «σωστού» αμιγούς σύμβολου και μοναδιαίου υπό-όρου όταν υπάρχει πάνω από ένα. Αυτό συμβαίνει, επειδή σε κάθε βήμα της αναδρομής ο DPLL, πρώτα προσπαθεί να αναθέσει τιμές σε αμιγή σύμβολα και μοναδιαίους υπό-όρους προτού επεκταθεί σε άλλες μεταβλητές.

Από τα παραπάνω γίνεται κατανοητό ότι υπάρχει «χώρος» για βελτιστοποίηση επιλέγοντας την κατάλληλη μεταβλητή προς ανάθεση τιμής στην περίπτωση που δεν έχουμε ούτε αμιγές σύμβολο ούτε μοναδιαίο υπό-όρο. Όταν χρησιμοποιείται μια ευρετική μέθοδος για την επιλογή αρχικής μεταβλητής και ανάθεση τιμής σε αυτήν, που να μας εξασφαλίζει ότι ο μέγιστος αριθμός υπό-όρων θα ικανοποιηθεί, μπορούμε να θεωρήσουμε την μέθοδο ως βελτιστοποιημένη εκδοχή του DPLL (optimized DPLL). Αντιθέτως στην εκδοχή του DPLL στην οποία γίνεται τυχαία επιλογή συμβόλου και απόδοσης τιμής, τη χαρακτηρίζουμε μη-βελτιστοποιημένη (unoptimized DPLL).

Η εργασία των Ankur, Madhyastha, & Prince[47], διαπίστωσε ότι η ακρίβεια του WalkSAT βελτιώνεται σημαντικά ενσωματώνοντας τυχαίες επανεκκινήσεις και ότι η ευριστική ανάθεση αρχικών τιμών στον στον DPLL μπορεί να μειώσει το χρόνο εκτέλεσης. Στηριζόμενοι στις παρατηρήσεις αυτές εισήγαγαν την έννοια του HybridSAT που συνδυάζει τα πλεονεκτήματα των WalkSAT και DPLL.

## 5.3 Διακριτό πρόβλημα σακιδίου (0-1 Knapsack problem)

Το Knapsack problem μελετάται για περισσότερο από έναν αιώνα, με αφετηρία περίπου το 1897. Από τα τέλη της δεκαετίας του ‘50, ο Tobias Dantzing (1884 - 1956) ξεκίνησε τη μελέτη των Knapsack problems και έκτοτε η έρευνα τους εντάθηκε για δύο σημαντικούς λόγους. Πρώτον, τα προβλήματα αυτά έχουν άμεση εφαρμογή στη βιομηχανία, στη μηχανική, στο οικονομικό management και γενικότερα σε οποιονδήποτε τομέα υπάρχει ένα μοναδικό και σπάνιο αγαθό – πόρος που τον διεκδικούν πολλοί και δεύτερον για θεωρητικούς λόγους αφού τα Knapsack problems εμφανίζονται σε προβλήματα ακέραιου προγραμματισμού.

Το όνομα του εμπνεύστηκε από τον προβληματισμό που δημιουργείται σε όλους όταν φτιάχνουμε μια βαλίτσα. «Ποια πολύτιμα / χρήσιμα αντικείμενα πρέπει να επιλέξω όταν ο χώρος που έχω στη διάθεση μου είναι περιορισμένος»;

Στην οικογένεια των Knapsack Problems δίδεται ένα σύνολο από αντικείμενα και ζητείται να επιλεγούν αυτά που θα μεγιστοποιήσουν το αντίστοιχο κέρδος, χωρίς όμως να παραβιάζεται η χωρητικότητα του ή των σακιδίων. Υπάρχουν διαφορετικά είδη Knapsack Problems, ανάλογο με την κατανομή των αντικειμένων και των σακιδίων. Σε άλλα είδη προβλημάτων, κάποια αντικείμενα μπορούν να επιλεγούν το πολύ μία φορά, ενώ σε κάποια άλλα υπάρχει μια περιορισμένη ποσότητα από κάθε αντικείμενο.

Μία κατηγοριοποίηση την προβλημάτων σακιδίου είναι η παρακάτω [20]:

- **Ενός σακιδίου** (Single knapsack problems). Εδώ υπάρχει μόνο ένα δοχείο ή σακίδιο, που πρέπει να γεμίσει με το βέλτιστο σύνολο αντικειμένων. Στο είδος αυτό ανήκουν:
    - 0-1Knapsack (ένα αντικείμενο από κάθε είδος)
    - Bounded knapsack (πολλά αντικείμενα από κάθε είδος)
    - Subset-sum (επιλογή υποσυνόλων αντικειμένων)
    - Change-making (δυνατότητα αντικατάστασης)

- **Πολλών σακιδίων** (Multiple knapsack problems). Αντίθετα, εδώ είναι διαθέσιμα περισσότερα από ένα σακίδια. Στην κατηγορία αυτή ανήκουν
    - 0-1 Multiple knapsack
    - Generalized assignment
    - Bin-packing

Παρακάτω θα δούμε κάποιες από τις μέθοδούς επίλυσης προβλημάτων 0-1 Knapsack

### 5.3.1 Μέθοδος Brute Force

Η μέθοδος Brute Force, είναι μια απλή προσέγγιση στην επίλυση ενός προβλήματος, Αν υπάρχουν n-αντικείμενα από τα οποία πρέπει να επιλέξουμε για το γέμισμα του σακιδίου, τότε υπάρχουν $2^n$ δυνατοί συνδυασμοί αντικειμένων. Ένα αντικείμενο επιλέγεται ή δεν επιλέγεται. Δημιουργεί μια λίστα, n δυαδικών ψηφίων, δηλαδή 0 και 1. Εάν για το i-οστό σύμβολο της λίστας το ψηφίο είναι 1, τότε το στοιχείο i έχει επιλεγεί και αν είναι 0, όχι.

**Αλγόριθμος 15: BruteForce** *(Weights [1 … N], Values [1 … N], A[1…N]*)[48]

**Input:** *Array Weights contains the weights of all items, Array Values contains the values of all items, Array A initialized with 0s is used to generate the bit strings*
**Output:** *Best possible combination of items in the knapsack bestChoice [1 .. N]*

```
1:  for i = 1 to 2^n do
2:      j ← n
3:      tempWeight ← 0
4:      tempValue ← 0
5:      while ( A[j] != 0 and j > 0)
6:          A[j] ← 0
7:          j ←j – 1
8:      A[j]←1
9:      for k ← 1 to n do
10:         if(A[k] = 1) then
11:             tempWeight ← tempWeight + Weights[k]
12:             tempValue ←tempValue + Values[k]
13:     if ((tempValue > bestValue) AND (tempWeight ← Capacity)) then
14:         bestValue ← tempValue
```

```
15:                    bestWeight ←tempWeight
16:            bestChoice ← A return bestChoice
17:    return bestChoice
```

Η πολυπλοκότητα του αλγόριθμου Brute Force είναι O ($n2^n$) και είναι εκθετικής μορφής. Για αυτό μπορεί να χρησιμοποιηθεί μόνο για μικρά προβλήματα.

### 5.3.2 Δυναμικός Προγραμματισμός[20]

Περιγραφή προβλήματος: Έχουμε επεξεργαστεί $n$ αριθμό αρχείων και θέλουμε να τα αποθηκεύσουμε σε ένα χώρο χωρητικότητας $W$. Το κάθε αρχείο $i$, έχει μέγεθος $w_i$ bytes και χρειάζεται $u_i$ χρόνο, που είναι ανάλογος του μεγέθους του, για να επαναεπεξεργαστεί. Αν θέλουμε να αποφύγουμε χρόνο επεξεργασίας, πριν την αποθήκευση, τότε πρέπει να βρούμε ένα υποσύνολο αρχείων τα οποία θα πρέπει να έχει τα παρακάτω χαρακτηριστικά:

- Το συνολικό μέγεθος του υποσυνόλου των αρχείων πρέπει να μην ξεπερνά την αποθηκευτική μας χωρητικότητα $W$,
- Ο χρόνος επεξεργασίας (ανάλογος του μεγέθους του κάθε αρχείου) να είναι όσο γίνεται μεγαλύτερος
- Δεν μπορούμε να αποθηκεύσουμε τα αρχεία τμηματικά, παρά μόνο ολόκληρα.

**1ο Βήμα:** Χωρίζουμε το αρχικό πρόβλημα σε μικρότερα προβλήματα. Κατασκευάζουμε έναν πίνακα V της μορφής $V[0…n,0…W]$. Για κάθε $i \in [1,n]$ και $w \in [1,W]$, στη θέση V[$i$, $w$], αποθηκεύουμε το μέγιστο χρόνο επεξεργασίας οποιουδήποτε υποσυνόλου αρχείων {1, 2, …, i}, που το καθένα, είναι μεγέθους, το πολύ w.

Εάν μπορούμε να υπολογίσουμε όλες τις καταχωρίσεις του πίνακα, τότε η καταχώριση V[$n$,$W$] θα περιέχει τον μεγαλύτερο χρόνο εκτέλεσης των αρχείων {1, 2, …, i} που θα ταιριάζει στο χώρο αποθήκευσης, δηλαδή τη λύση του προβλήματος.

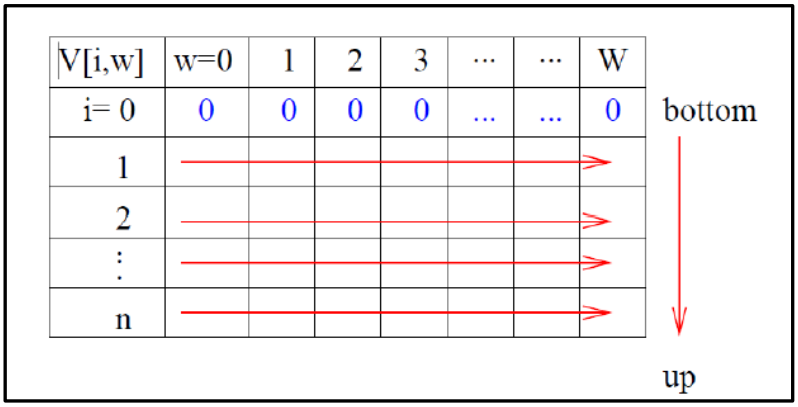
**2ο Βήμα**: Ορισμός αναδρομικής σχέσης.

Για να υπολογίσουμε το $V[i,w]$ έχουμε 2 μόνο επιλογές για το αρχείο i. Είτε να μην επιλέξουμε το αρχείο i, είτε να το επιλέξουμε. Στην 1η περίπτωση, επιλέγουμε το καλύτερο από τα υπόλοιπα αρχεία $\{1,2,\ldots,i-1\}$ και αποθηκεύουμε την τιμή $w=V[i-1,w]$. Στη 2η περίπτωση, η οποία είναι εφικτή μόνο όταν το μέγεθος του αρχείου είναι μικρότερο από το σύνολο του αποθηκευτικού χώρου, $w[i] \le W$. Μετά την επιλογή του αρχείου, ο διαθέσιμος χώρος για τα υπόλοιπα αρχεία $\{1,2,\ldots,i-1\}$ είναι $w-wi$, άρα η επόμενη καλύτερη περίπτωση επιλογής θα είναι: $V[i-1,w-wi]$.

Η αναδρομική μας σχέση θα είναι:

$$V(i,w)\begin{cases} 0 & \text{if } i=0 \\ V(i-1,w) & \text{if } wi>W \\ \max(V[i-1,w],\ vi+V[i-1,w-wi]) & \text{otherwise} \end{cases}$$

**3ο Βήμα:** Υπολογίζουμε από κάτω προς τα πάνω την τιμή $V[i,w]$. Έτσι ξεκινάμε, θέτοντας, για $0 \leq w \leq W$, $V[0,w]=0$. Και συμπληρώνουμε τον υπόλοιπα πίνακα ανά γραμμή, με την τιμή $V[i,w]=\max(V[i-1,w], vi+V[i-1,w-wi])$ που βρίσκουμε κάθε φορά.




*Σχήμα 36. Πίνακας επαλήθευσης δυναμικού προγραμματισμού*[20]

Με τα τρία βήματα που περιγράψαμε**,** καταφέραμε να συμπληρώσουμε τον πίνακα με τις λύσεις των υπο-προβλημάτων, δεν ξέρουμε όμως ποιο είναι εκείνο το σύνολο Τ που μας δίνει την βέλτιστη λύση. Για να το επιτύχουμε αυτό θα εισάγουμε ένα βοηθητικό Boolean διάνυσμα, το keep[i, w] που θα παίρνει την τιμή 1, εάν επιλέγουμε το i-οστό αντικείμενο, διαφορετικά θα παίρνει την τιμή 0.

**Αλγόριθμος 16: Knapsack** (*n, w, W*)[20]

**Input:** *n number of files, w size of files, W amount of space*

```
1:    for (w=0 to W)
2:        V[0,w]=0;
3:    for (i=0 to n)
4:        V[i,0]=0;
5:    for (i=1 to n)
6:        for (w=0 to W)
7:            if w[i] >w then
8:                V[i,w]=V[i−1,w];
9:                keep [i, w] = 0;
10:           else if w[i]≤w then
11:               V[i,w]=max(V[i−1,w],v[i]+V[i−1,w−w[i]]);
12:               keep [i, w] = 1;
13:   K=W;
14:   for (i=n down to 1)
15:       if (keep[i,K]== 1)
16:       output i;
18        K=K−w[i];
18:   return V[n,w];
```

## 5.3.3 A new genetic algorithm for KP [49]

Στον αλγόριθμο n-αντικείμενα αντιπροσωπεύονται από n-γονίδια σε ένα δυαδικό πίνακα όπου οι δυνατές τιμές είναι 0 ή 1. Αν το i-οστό αντικείμενο έχει την τιμή 1 τότε

σημαίνει ότι αυτό το αντικείμενο συμπεριλαμβάνεται στο σακίδιο. Για να υπολογισθεί ο χώρος που καταλαμβάνει ένας συνδυασμός αντικείμενων, προστίθενται όλα τα αντικείμενα που έχουν την τιμή 1 στον πίνακα.

Για την επιλογή γονέων χρησιμοποιείται η μέθοδος της ρουλέτας. Η μέθοδος αυτή εγγυάται ότι δυνατά άτομα παράγουν περισσότερους απογόνους. Η διασταύρωση γίνεται με τέτοιον τρόπο ώστε η ρουλέτα να καταλήγει σε γονίδια με την ίδια καταχώρηση με τους γονείς και ότι ο γονέας μπορεί να μεταφέρει τα γονίδιά του στα παιδιά. Η μετάλλαξη εφαρμόζεται τυχαία για το 25% των γονιδίων και εφαρμόζεται για όλα τα άτομα.

Τα βήματα του αλγόριθμου με δεδομένα:

- $p\ i$ : η αξία του αντικειμένου $i$,
- $w\ i$ : βάρος του αντικειμένου $i$,
- $c$ : Συνολική χωρητικότητα,

**Βήμα 1.** Υπολογίζεται το πηλίκο $p\ i\ /\ w\ i$, (αξία/βάρος) και ταξινομούνται με φθίνουσα τιμή. Έτσι δημιουργείται η λίστα κατάταξης $I$

**Βήμα 2.** Δημιουργείται ο αρχικός μας πληθυσμός από n αντικείμενα ξεκινώντας από το i-στο αντικείμενο της λίστας κατάταξης $I$ με τον περιορισμό ότι το σύνολο των αντικειμένων δεν ξεπερνά την συνολική χωρητικότητα.

**Βήμα 3.** Ο αρχικός πληθυσμός ταξινομείται κατά φθίνουσα σειρά με βάση η αξία του αντικειμένου $i$ ($p\ i$). Στην συνέχεια ο μισός πληθυσμός με την μεγαλύτερη αξία μεταφέρεται στην επόμενη γενεά.

**Βήμα 4.** Για να δημιουργηθεί το υπόλοιπο μισό της νέας γενεάς χρησιμοποιείται η μέθοδος της ρουλέτας, με σκοπό την δημιουργία καλύτερων απογόνων.

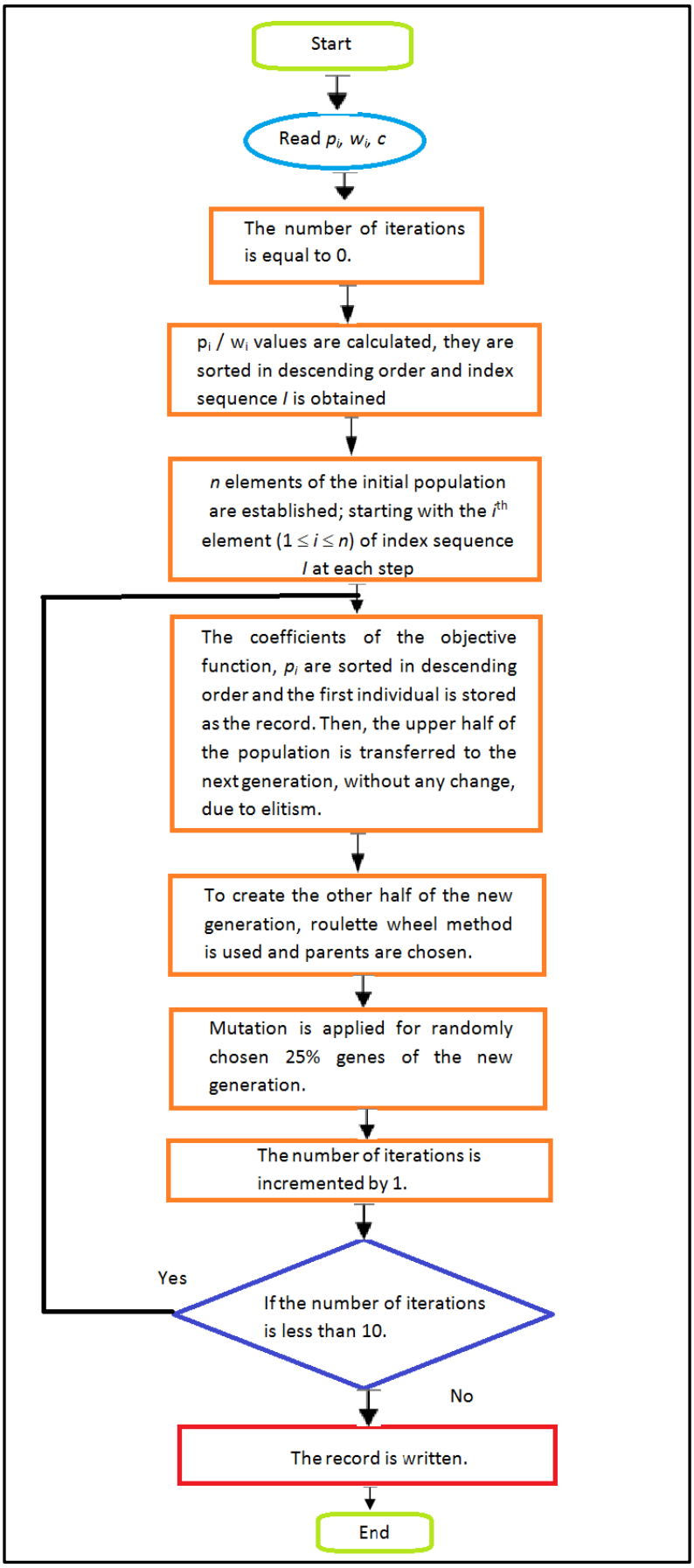
**Βήμα 5.** Η μετάλλαξη εφαρμόζεται τυχαία στο 25% των γονιδίων της νέας γενεάς και δημιουργούνται νέα αντικείμενα με την προϋπόθεση ότι η χωρητικότητα τους δεν θα ξεπερνά την δεδομένη χωρητικότητα.

**Βήμα 6.** Επαναλαμβάνουμε τα βήματα 3,4,5, για τον αριθμό επαναλήψεων που έχουμε ορίσει.

**Βήμα 7.** Καταγράφονται τα αποτελέσματα και ο αλγόριθμος σταματά.

Η διαφορά του παραπάνω αλγόριθμου σε σχέση με τον κλασσικό γενετικό αλγόριθμο είναι ότι δεν γίνεται τυχαία η αρχικοποίηση του πληθυσμού. Έτσι περιορίζεται αρκετά ο χώρος αναζήτησης λύσεων.

Στο επόμενο σχήμα 37 φαίνεται το διάγραμμα ροής του αλγόριθμου.




*Σχήμα 37.Διάγραμμα ροής του γενετικού αλγόριθμου*

## 5.4 Πρόβλημα κοπής υλικών (CSP - Cutting Stock Problem)

Στο πρόβλημα κοπής υλικών (CSP - Cutting Stock Problem) αναζητείται ο τρόπος κοπής ενός υλικού (για παράδειγμα ρολό χαρτιού, ύφασμα, ορθογώνια κομμάτια πλαστικού, και άλλα) έτσι ώστε να καλυφθεί η ζήτηση για συγκεκριμένα σχέδια και να ελαχιστοποιηθεί το υλικό που δεν θα είναι εκμεταλλεύσιμο μετά την κοπή.

Η πρώτη γνωστή διατύπωση του προβλήματος κοπής αποθεμάτων δόθηκε το 1939 από τον Ρώσο οικονομολόγο Kantorovich [50]. Η πρώτη και σημαντική πρόοδος στην επίλυση των προβλημάτων κοπής ήταν το σημαντικό έργο των Gilmore και Gomory (1961, 1963) όπου περιγράφουν την τεχνική δημιουργίας μοτίβων με καθυστέρηση, για την επίλυση του προβλήματος ελαχιστοποίησης απωλειών αποκοπής με χρήση γραμμικού προγραμματισμού.

Από εκείνη τη στιγμή προέκυψε έκρηξη ενδιαφέροντος σε αυτόν τον τομέα εφαρμογής. Οι Sweeney και Paternoster [50] (1991) έχουν εντοπίσει περισσότερα από 500 έγγραφα που ασχολούνται με την κοπή αποθεμάτων και συναφή προβλήματα και εφαρμογές. Οι κύριοι λόγοι αυτής της δραστηριότητας είναι ότι υπάρχουν προβλήματα κοπής αποθεμάτων σε ένα ευρύ φάσμα βιομηχανιών, υπάρχει μεγάλο οικονομικό κίνητρο για την εξεύρεση αποτελεσματικότερων διαδικασιών επίλυσης και είναι εύκολο να συγκριθούν οι εναλλακτικές διαδικασίες επίλυσης και να εντοπιστούν τα πιθανά οφέλη, χρησιμοποιώντας μια προτεινόμενη διαδικασία.

Η μεγάλη ποικιλία εφαρμογών που αναφέρθηκαν στη βιβλιογραφία οδήγησε τον Dyckhoff (1990) να αναπτύξει ένα σχήμα ταξινόμησης για τα προβλήματα κοπής και των προβλημάτων συσκευασίας (τα προβλήματα συσκευασίας σχετίζονται στενά με τα προβλήματα κοπής των αποθεμάτων).

Κατατάσσει τα προβλήματα χρησιμοποιώντας τέσσερα χαρακτηριστικά ως εξής [50]:

- Το πρώτο κριτήριο είναι η διαστασιολόγηση ***(Dimensionality)*** δηλαδή ο ελάχιστος αριθμός διαστάσεων πραγματικών αριθμών που είναι απαραίτητος για την περιγραφή της γεωμετρίας των μοτίβων: μιας διάστασης (1D), δύο διαστάσεων (2D), τριών διαστάσεων (3D) και N-Dimensional packing (N>3)
- Τύπος ανάθεσης. Το δεύτερο κριτήριο παρέχει πληροφορίες σχετικά με τον τύπο της ανάθεσης (***Kind of assignment***): αν θέλουμε να παράγουμε όλα τα αντικείμενα με το μικρότερο δυνατό κόστος ή αν έχουμε ένα συγκεκριμένο στοκ υλικών και θέλουμε να παράγουμε τα περισσότερο δυνατό αντικείμενα
- Ποικιλία μεγάλων αντικειμένων (***Assortment of large objects***): Θέλουμε να παράγουμε ένα μεγάλο αντικείμενο, πολλά πανομοιότυπα μεγάλα αντικείμενα ή διαφορετικά μεγάλα αντικείμενα
- Ποικιλία μικρών αντικειμένων (***Assortment of small items***): Θέλουμε να παράγουμε λίγα αντικείμενα διαφόρων διαστάσεων, πολλά αντικείμενα πολλών διαφορετικών διαστάσεων, πολλά στοιχεία με σχετικά λίγες διαστάσεις ή πολλά πανομοιότυπα στοιχεία.

Σύμφωνα με τον Dyckhoff, οι προαναφερθείσες κατηγορίες οδηγούν στην δημιουργία ενός πίνακα, στον οποίο χαρακτηρίζονται τα περισσότερα προβλήματα, από αυτά, που συναντώνται στην βιβλιογραφία.

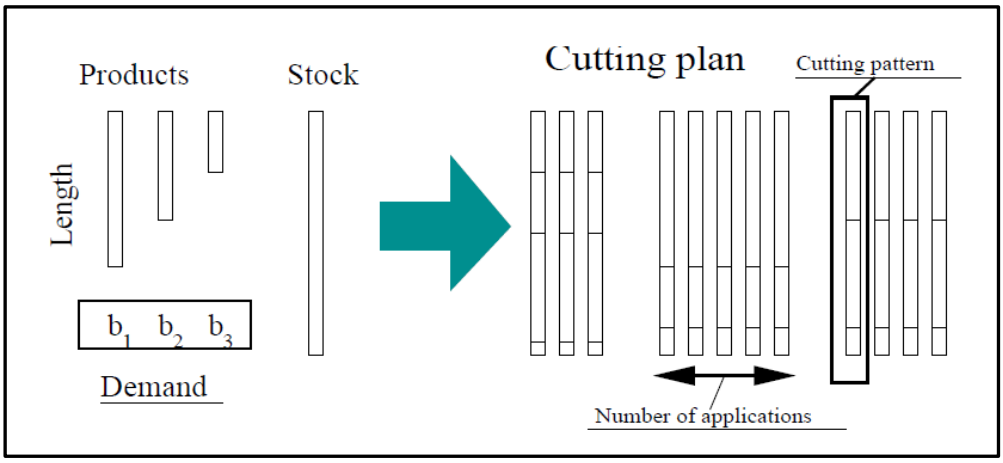
*Πίνακας 3*[20]

| **Πρόβλημα** | **Διαστάσεις** | **Τρόπος επιλογής αντικειμένων** | **Ταξινόμηση μεγάλων αντικειμένων** | **Ταξινόμηση μικρών αντικειμένων** | **Τύπος προβλήματος** |
|---|---|---|---|---|---|
| Knapsack (Classical) | 1 | B | O | | 1/B/O/ |
| Pallet loading | 2 | B | O | C | 2/B/O/C |
| More-dimensional knapsack | | B | O | | /B/O/ |
| Dual bin packing | 1 | B | O | M | 1/B/O/M |
| Vehicle loading | 1 | V | I | F | 1/V/I/F |
| //- | 1 | V | I | M | 1/V/I/M |
| Container loading | 1 | V | I | | 1/V/I |
| //- | 3 | B | O | M | 3/B/O/M |
| Bin packing (Classical) | 1 | V | I | R | 1/V/I/R |
| Cutting stock (Classical) | 1 | V | I | M | 1/V/I/M |
| 2-d bin packing | 2 | V | D | R | 2/V/D/R |
| Usual 2-d cutting stock | 2 | V | I | | 2/V/I |
| Assembly line balancing | 1 | V | I | M | 1/V/I/M |
| Multiprocessor scheduling | 1 | V | I | M | 1/V/I/M |
| Memory allocation | 1 | V | I | M | 1/V/I/M |
| Change making | 1 | B | O | R | 1/B/O/R |
| Multi-period capital budgeting | N | B | O | | N/B/O/ |

Συμβολισμοί κατηγοριοποίησης:

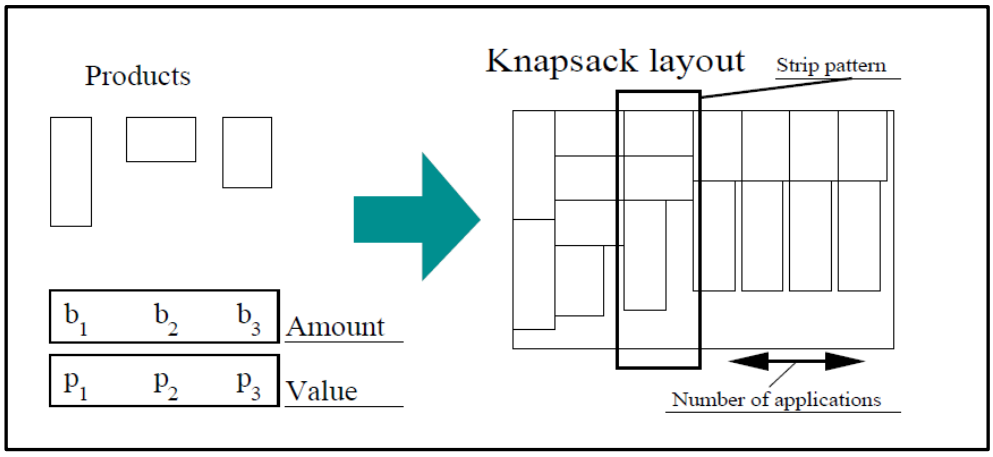
- (1….N) Αριθμός διαστάσεων,
- (B) Επιλογή όλων των χώρων και ένα μέρος από τα αντικείμενα,
- (V) Επιλογή ενός μέρους από τους χώρους και όλα τα αντικείμενα,
- (O) Ένας χώρος,
- (I) Ίδιας μορφής χώροι,
- (D) Διαφορετικοί χώροι,
- (F) Μερικά αντικείμενα διαφορετικής μορφής,
- (M) Πολλά αντικείμενα διαφορετικής μορφές,
- (R) Πολλά αντικείμενα με σχετικά με λίγες διαφορετικές μορφές,
- (C) Πανομοιότυπα αντικείμενα

Το μονοδιάστατο πρόβλημα κοπής αποθεμάτων (1D-CSP) είναι να αποκτηθεί ένα προκαθορισμένο σύνολο από ράβδους διαφόρων μεγεθών. Ο στόχος είναι η ελαχιστοποίηση του αριθμού των αρχικών ράβδων που θα χρησιμοποιηθούν (σχήμα 38).




*Σχήμα 38. Πρόβλημα 1D-CSP* [51]

Το πρόβλημα κοπής δύο διαστάσεων σε δύο στάδια (2D-2CP) επεξεργάζεται τον τρόπο τού πως θα αποκτηθεί ένα υποσύνολο ορθογώνιων στοιχείων από μια ενιαία ορθογώνια πλάκα έτσι ώστε να ελαχιστοποιηθούν οι απώλειες και να μεγιστοποιηθεί η συνολική αξία των επιλεγμένων στοιχείων (σχήμα 39).




*Σχήμα 39. Πρόβλημα 2D-CSP* [51]

Στο πρώτο στάδιο παίρνουμε κατακόρυφες λωρίδες, με κοψίματα από άκρη σε άκρη και στο δεύτερο στάδιο η κατεύθυνση κοπής περιστρέφεται κατά 90 ° και οι τομές κατά μήκος των λωρίδων παράγουν τα αντικείμενα.

### 5.4.1 Γενετικός Αλγόριθμος κοπής δυο διαστάσεων [52]

Στα δύσκολα προβλήματα, χρησιμοποιούνται πολύ συχνά, ευριστικές μέθοδοι για την επίλυσή τους, ενώ μια λύση που θα είναι πολύ κοντά στην βέλτιστη μπορεί να θεωρηθεί αποδεκτή. Η δυσκολία στα προβλήματα κοπής έγκειται ότι όλα τα παραλληλόγραμμα πρέπει να ταξινομούνται ταυτόχρονα και στους δύο άξονες (X και Y).

Όταν χρησιμοποιούμε γενετικούς αλγόριθμους πρέπει να καθοριστούν πριν από οποιαδήποτε διαδικασία ο ορισμός και η αντιστοίχιση των παραμέτρων:

- η κωδικοποίηση των δεδομένων,
- η διασταύρωση
- η μετάλλαξη και
- η επιλογή.

Στην περίπτωση της κοπής σε δυο διαστάσεις προτείνονται οι ακόλουθοι παράμετροι:

1. Η κωδικοποίηση των παραμέτρων γίνεται με τη ταξινόμηση και παρουσίαση όλων των ορθογώνιων με την σειρά πάνω / αριστερά σε όλες τις διαστάσεις πλάτους ύψους και προσανατολισμού. Τα ορθογώνια παρουσιάζονται σε ταξινομημένη λίστα σαν χρωμόσωμα.
2. Στο στάδιο της διασταύρωσης υποδεικνύεται ένα σημείο κοπής.
3. Ο συντελεστής μετάλλαξης αποτελείται από την τυχαία περιστροφή των ορθογωνίων και έτσι έχουμε την δημιουργία νέων χρωμοσωμάτων και την αμοιβαία ανταλλαγή μεταξύ δυο ορθογωνίων για την δημιουργία ενός νέου χρωμοσώματος
4. Ο συντελεστής επιλογής βασίζεται στην ομοιόμορφη τυχαία διασπορά, αλλά στο τέλος ένα καλύτερο χρωμόσωμα αντικαθιστά ένα χειρότερο. Με αυτή την τακτική δημιουργείται έμμεσα ένας ελιτισμός

Στο πρώτο βήμα, κατά την αρχικοποίηση των συνθηκών του αλγόριθμου κοπής, έχουν γίνει ορισμένες τροποποιήσεις από την γενική μορφή των Γ.Α., όπως η τυχαία διάταξη ορισμένων χρωμοσωμάτων και η διάταξη σε φθίνουσα σειρά ως προς το πλάτος και το ύψος

Επίσης, στο δεύτερο βήμα, ένα μέρος των χρωμοσωμάτων είναι γεμάτο με όλα τα ορθογώνια σε προσανατολισμό πορτραίτου, ένα μέρος των χρωμοσωμάτων είναι γεμάτο με όλα τα ορθογώνια σε προσανατολισμό τοπίου και ένα μέρος με τα ορθογώνια σε μικτό προσανατολισμό. Όλες αυτές οι τροποποιήσεις γίνονται για να παρουσιαστεί καλύτερη γενετική ποικιλομορφία, κατά την αρχικοποίηση του αλγορίθμου.

## 5.5 Πρόβλημα bin packing μιας διάστασης (1D Bin Packing Problem)

Το 1D Bin Packing Problem (Πρόβλημα Τοποθέτησης Σε Κάδους Μίας Διάστασης), είναι ένα από τα πιο διάσημα προβλήματα συνδυαστικής βελτιστοποίησης. Η δομή και οι εφαρμογές του έχουν μελετηθεί από το 1930. Το 1961 οι Gilmore και Gomory εισήγαγαν, για αυτή την κατηγορία προβλημάτων, την έννοια της δημιουργίας στήλης, χρησιμοποιώντας προηγούμενες ιδέες των Ford - Fulkerson και Dantzig - Wolfe. Αυτό είναι ένα από τα πρώτα προβλήματα για τα οποία διερευνήθηκε η χειρότερη περίπτωση απόδοσης αλγορίθμων προσέγγισης. Τις επόμενες δεκαετίες μελετήθηκαν τα κατώτατα όρια *(lower bounds)* και προτάθηκαν ακριβείς αλγόριθμοι.

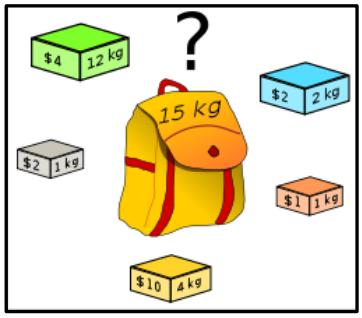
Το bin packing problem, θέτει το εξής ερώτημα:

*Δεδομένης μιας πεπερασμένης συλλογής n-βαρών και μιας συλλογής πανομοιότυπων δοχείων χωρητικότητας C (που υπερβαίνει το μεγαλύτερο από τα βάρη), ποιος είναι ο ελάχιστος αριθμός k δοχείων στα οποία μπορούν να τοποθετηθούν τα βάρη χωρίς να ξεπερνιέται η χωρητικότητα των δοχείων C;*

Το πρόβλημα αυτό μπορούμε να το δούμε σαν ιδιαίτερη περίπτωση του προβλήματος κοπής υλικών, στο οποίο θέλουμε από δεδομένες πλάκες (φύλλα) να κόψουμε πλάκες

ορισμένων διαστάσεων, έτσι ώστε τα κομμάτια που θα περισσέψουν να είναι εύκολο να χρησιμοποιηθούν στο μέλλον.

Όταν ο αριθμός των δοχείων περιορίζεται στον 1 και κάθε στοιχείο χαρακτηρίζεται τόσο από όγκο όσο και από αξία, το πρόβλημα της μεγιστοποίησης της αξίας των αντικειμένων που μπορούν να χωρέσουν στον κάδο είναι γνωστό ως πρόβλημα του σακιδίου (κεφάλαιο 5.3).




*Σχήμα 40. Διακριτό πρόβλημα σακιδίου*

Υπάρχουν πολλές εφαρμογές του bin packing problem, οι οποίες το καθιστούν επίκαιρο και διαρκώς ενδιαφέρον. Οι περισσότερες φυσικά έχουν να κάνουν με τον τομέα της πληροφορικής, της βιομηχανίας, της επιχειρηματικής έρευνας και της οικονομίας, όπως:

- η ταξινόμηση ψηφιακών αρχείων σε σκληρούς δίσκους η στο Cloud
- ο προγραμματισμός Ανεξάρτητων Εργασιών
- η κοπή των αποθεμάτων
- η φόρτωση φορτηγών και εμπορευματοκιβωτίων
- η εκχώρηση άρθρων εφημερίδων σε στήλες
- η ανάθεση διαφημίσεων σε διαλείμματα σε τηλεοπτικό σταθμό

## 5.5.1 OnLine Αλγόριθμοι

### 5.5.1.1 Next Fit

Η απλούστερη προσεγγιστική λύση στο πρόβλημα της συσκευασίας του δοχείου είναι ο αλγόριθμος ***Next Fit (NF)***. Η φιλοσοφία του αλγόριθμου είναι η εξής:

- *Ανοίξτε* ένα δοχείο και τοποθετήστε τα αντικείμενα που χωράνε σε αυτό με τη σειρά που εμφανίζονται στη λίστα
- Όταν ένα αντικείμενο δεν ταιριάζει στον ανοιχτό κάδο, κλείστε μόνιμα αυτόν τον κάδο και ξεκινήστε έναν άλλο
- Συνεχίστε να συσκευάζετε τα υπόλοιπα αντικείμενα της λίστας
- Αν βέβαια κάποια από τα διαδοχικά βάρη στη λίστα γεμίζουν ακριβώς έναν κάδο, ο κάδος κλείνει και ανοίγει ένας νέος κάδος

**Αλγόριθμος 17: Next Fit** [36]

```
1:  for All objects i = 1, 2, 3,..., n do
2:      if Object i fits in current bin then
3:          Pack object i in current bin
4:      else
5:          Create new bin, make it the current bin, and pack object i
6:      end if
7:  end for
```

Από τη στιγμή που η πακετοποίηση του αντικειμένου γίνεται σε σταθερό χρόνο, ο αλγόριθμος κυριαρχείται από το βρόχο οπότε η χρονική πολυπλοκότητα του είναι σαφώς ***γραμμική***, δηλαδή ***O (n)***. Επίσης, η πολυπλοκότητά του ως προς το χώρο είναι O (1).

Σαφώς, ο αλγόριθμος Next Fit δεν παράγει πάντα μια βέλτιστη λύση για ένα δεδομένο σύνολο βαρών. Διαδικασίες όπως ο NF μερικές φορές αναφέρονται ως ευριστικοί αλγόριθμοι επειδή, παρόλο που σχεδιάστηκαν ως τρόποι βέλτιστης επίλυσης ενός προβλήματος, δεν δίνουν πάντα τη βέλτιστη λύση.

### 5.5.1.2 First Fit

Βλέποντας τα αποτελέσματα του Next Fit (NF), είναι σαφές ότι θα μπορούσαμε να γεμίσουμε τους κάδους με καλύτερο τρόπο, και πιθανά να χρησιμοποιούσαμε και μικρότερο αριθμό κάδων. Ο First Fit αλγόριθμος έχει καλύτερη απόδοση από το Next Fit: βρίσκει μια λύση που είναι το πολύ 70% μακριά από τη βέλτιστη λύση. Αυτό επιτυγχάνεται αν αφήσουμε ανοιχτό τον κάδο όταν δε χωράει το επόμενο αντικείμενο, με την ελπίδα ότι στη συνέχεια θα φτάσει ένα μικρότερο βάρος το οποίο και θα χωράει. Αυτή είναι η βάση του ***First Fit Algorithm (FF)***, ο οποίος είναι ένας άπληστος αλγόριθμος. Η φιλοσοφία του είναι η εξής:

- Ανοίξτε ένα δοχείο και τοποθετήστε τα αντικείμενα που χωράνε σε αυτό με τη σειρά που εμφανίζονται στη λίστα.
- Όταν ένα αντικείμενο δεν ταιριάζει στον ανοιχτό κάδο, ξεκινήστε έναν άλλο κάδο.
- Τοποθετείστε το επόμενο στοιχείο στη λίστα στον πρώτο κάδο που δεν έχει γεμίσει πλήρως και στον οποίο θα ταιριάζει. Η αρίθμηση γίνεται από αριστερά προς τα δεξιά.
- Όταν οι κάδοι γεμίσουν τελείως, κλείνουν και αν ένα αντικείμενο δε χωράει σε οποιονδήποτε ανοιχτό κάδο, ανοίγει ένας καινούργιος.

**Αλγόριθμος 18: First Fit** [36]

```
1:  for All objects i = 1, 2, 3,..., n do
2:      for All bins j = 1, 2, 3,... do
3:          if Object i fits in bin j then
4:              Pack object i in bin j
5:              Break the loop and pack the next object
```

```
6:              end if
7:          end for
8:          if Object i did not fit in any available bin then
9:              Create new bin and pack object i
10:         end if
11:     end for
```

Η παραπάνω εφαρμογή του First Fit απαιτεί χρόνο O ($n^2$), αλλά ο First Fit μπορεί να εφαρμοστεί σε χρόνο O($n*\log n$) χρησιμοποιώντας δέντρα δυαδικής αναζήτησης (Self-Balancing Binary Search Trees).

### 5.5.1.3 Best Fit

Για τον BF, κρατάμε και πάλι ανοιχτούς τους κάδους ακόμη και όταν το επόμενο στοιχείο της λίστας δεν ταιριάζει σε κάδους που είχαν ανοίξει προηγουμένως, με την ελπίδα ότι θα χωρέσει σε αυτούς ένα μικρότερο στοιχείο. Το κριτήριο για την τοποθέτηση είναι ότι βάζουμε το επόμενο στοιχείο στον τρέχοντα ανοιχτό κάδο (που δεν είναι ακόμα γεμάτος) που αφήνει το ελάχιστο υπόλοιπο. Στην περίπτωση της ισοβαθμίας, τοποθετούμε το αντικείμενο στον κάδο με τη μικρότερο κατάταξη (indexing) από αριστερά προς τα δεξιά. Η φιλοσοφία του BF είναι η εξής:

- Ανοίξτε ένα δοχείο και τοποθετήστε τα αντικείμενα που χωράνε σε αυτό με τη σειρά που εμφανίζονται στη λίστα
- Όταν ένα αντικείμενο δεν ταιριάζει στον ανοιχτό κάδο, ξεκινήστε έναν άλλο κάδο
- Τοποθετείστε το επόμενο στοιχείο στη λίστα στον πρώτο κάδο που δεν έχει γεμίσει πλήρως και στον οποίο θα ταιριάζει και θα αφήνει το μικρότερο υπόλοιπο. Η αρίθμηση γίνεται από αριστερά προς τα δεξιά
- Όταν οι κάδοι γεμίσουν τελείως, κλείνουν και αν ένα αντικείμενο δε χωράει σε οποιονδήποτε ανοιχτό κάδο, ανοίγει πάντα ένας καινούργιος

**Αλγόριθμος 19: Best Fit** [36]

```
1:  for All objects i = 1, 2, 3,..., n do
2:      for All bins j = 1, 2, 3,... do
3:          if Object i fits in bin j then
4:              Calculate remaining capacity after the object has been added
5:          end if
6:      end for
7:      Pack object i in bin j, where j is the bin with minimum remaining capacity after adding
        the object (i.e. the object 'fits best').
8:      If no such bin exists, open a new one and add the object
9:  end for
```

Η παραπάνω εφαρμογή του Best Fit απαιτεί χρόνο O($n^2$), αλλά ο Best Fit μπορεί να εφαρμοστεί σε χρόνο O($n\log n$) όπως και ο FF. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας

ένα δυαδικό δέντρο του οποίου τα φύλλα αποθηκεύουν τις τρέχουσες υπολειπόμενες χωρητικότητες των αρχικών κάδων. Το δέντρο αυτό είναι ένα δέντρο στο οποίο:

- κάθε κόμβος που δεν έχει φύλλα έχει 2 ή 3 γιους
- κάθε διαδρομή από τη ρίζα σε ένα φύλλο έχει το ίδιο μήκος $l$
- οι ετικέτες στους κόμβους επιτρέπουν την αναζήτηση μιας δεδομένης τιμής φύλλου, την ενημέρωσή της ή την εισαγωγή ενός νέου φύλλου σε O($l$) χρόνο

Με τον τρόπο αυτό, κάθε επανάληψη του FF ή BF απαιτεί O(nlogn) χρόνο, αφού ο αριθμός των φύλλων περιορίζεται από το n. Η διάταξη του δέντρου γίνεται από τον δείκτη του κάδου. Τέλος, η πολυπλοκότητά του ***Best Fit Algorithm*** ως προς το χώρο είναι O(***n***).

### 5.5.2 Offline Bin Packing

Ένα θέμα με τους αλγορίθμους που έχουμε δει μέχρι στιγμής, είναι ότι είναι πολύ ευαίσθητοι στη σειρά των αντικειμένων. Ειδικότερα, αν τα μεγάλα αντικείμενα ακολουθούν τα μικρά, είναι δύσκολο να τοποθετηθούν στους ανοικτούς κάδους οπότε ένας νέος κάδος θα πρέπει να ανοίξει για αυτά.

Αυτή η κατασταση οδηγεί στους επόμενους αλγόριθμους, που λειτουργούν ταξινομώντας πρώτα τα αντικείμενα σε φθίνουσα σειρά μεγέθους, έτσι ώστε τα μεγαλύτερα αντικείμενα να είναι τα πρώτα, με την πρόθεση τα μικρότερα αντικείμενα να προσαρμοστούν στα εναπομείναντα κενά που αφήνουν. Με αυτό τον τρόπο, μπορεί κανείς να συσκευάσει τα βάρη σε λιγότερους κάδους, χωρίς να υπάρχει μεγάλη σπατάλη χώρου. Έτσι λοιπόν, είναι γνωστά εκ των προτέρων τα ***n*** και τα $s_1,\ldots, s_n$. Μια βέλτιστη λύση μπορεί να βρεθεί με εξαντλητική αναζήτηση. Η προσέγγιση σε έναν offline προσεγγιστικό αλγόριθμο είναι η εξής:

*Ταξινομήστε τα στοιχεία σε φθίνουσα σειρά μεγέθους και συνεχίστε με τον αντίστοιχο αλγόριθμο.*

Το αποτέλεσμα είναι να βελτιώνεται η απόδοση τόσο του ***First Fit*** όσο και του ***Best Fit*** αλγόριθμου. Ας υποθέσουμε λοιπόν ότι τα αντικείμενα έχουν ταξινομηθεί με φθίνουσα σειρά έτσι ώστε:

$v_1 \geq v_2 \geq \ldots \geq v_n$

και τότε εφαρμόζουμε τους NF, FF, BF. Οι αλγόριθμοι που προκύπτουν, έχουν χρονική πολυπλοκότητα ***O(nlogn)*** και ονομάζονται Next Fit Decreasing (NFD), First Fit Decreasing (FFD) και Best Fit Decreasing (BFD) αντίστοιχα.

### 5.5.3 Αλγόριθμος Martello and Toth

Ένας καλός αλγόριθμος, για να βρούμε την βέλτιστη λύση του bin-packing problem μας δίνεται από τους Martello και Toth [53][54]. Ο branch-and-bound αλγόριθμος τους είναι πολύπλοκος και θα περιγράψουμε μόνο τα κύρια σημεία του. Ο αλγόριθμος ταξινομεί τα αντικείμενα με φθίνουσα σειρά και τα τοποθετεί μέσα σε ένα ημισυμπληρωμένο bin εάν χωράνε, και σε ένα νέο bin, διακλαδώνοντας σε αυτές τις 2 εναλλακτικές.

Αυτό έχει ως αποτέλεσμα ο χώρος του προβλήματος να περιορίζεται στο n!, όπου το n είναι ο αριθμός τον αντικειμένων, άλλα αυτό είναι ένα «κακό» πάνω όριο, αφού πολλά αντικείμενα δεν θα χωρέσουν στο ίδιο bin. Σε κάθε κόμβο του δέντρου αναζήτησης οι

Martello & Toth υπολογίζουν για τον first-fit, best-fit, και worst-fit decreasing την πληρότητα των δοχείων για την κάθε αντίστοιχη μερική λύση.

Μια μερική λύση στο bin-packing problem, είναι η λύση όπου κάποια, αλλά όχι όλα, από τα αντικείμενα έχουν ανατεθεί σε bins. Η πληρότητα μιας μερικής λύσης, παίρνει τα τρέχοντα μερικώς-συμπληρωμένα bins, και αναθέτει τα υπόλοιπα αντικείμενα σε νέα bins.

Ο worst-fit decreasing αλγόριθμος τοποθετεί κάθε διαδοχικό αντικείμενο σε ημισυμπληρωμένο bin, διαλέγοντας αυτό που εχει μεγαλύτερη απομείνουσα χωρητικότητα εφόσον, μπορεί να χωρέσει την αξία του αντικειμένου. Κάθε μία από αυτές τις προσεγγιστικές λύσεις, συγκρίνεται με το κάτω όριο στην εναπομείνασα λύση και αποκαλείται L3. Το L3 όριο, υπολογίζεται «χαλαρώνοντας» το υπολειπόμενο ψευδοπρόβλημα, αφαιρώντας το μικρότερο αντικείμενο και στην συνέχεια εφαρμόζοντας το L2 όριο σε κάθε μια από τις «χαλαρωμένες» περιπτώσεις, επιστρέφοντας το μεγαλύτερο κάτω όριο.

Το όριο L2 των Martello & Toth's περιορίζει ισάξια το εκτιμώμενο wasted-space όριο που περιγράφτηκε παραπάνω, άλλα χρησιμοποιεί ένα πιο περίπλοκο αλγόριθμο για να το υπολογίσει. Αν ο αριθμός των bins που χρησιμοποιήθηκαν από οποιοδήποτε από τις προσεγγιστικές περατώσεις ισούται με το κάτω όριο για την συμπλήρωση της αντίστοιχης μερικής λύσης, κανένας παραπάνω έλεγχος δεν γίνεται κάτω απο αυτό τον κόμβο. Αν ο αριθμός των bins σε κάθε προσεγγιστική λύση ισούται με το κάτω όριο του αρχικού προβλήματος, ο αλγόριθμος τερματίζει επιστρέφοντας την λύση ως βέλτιστη. Η κύρια πηγή αποτελεσματικότητας του αλγόριθμου των Martello & Toth είναι ότι μειώνει το μέγεθος των εναπομεινάντων υποπροβλημάτων.

### 5.5.4 Αλγόριθμος Bin – Completion Korf

Ο καλύτερος μέχρι στιγμής αλγόριθμος για bin-packing, μας δίνεται από τον Korf [53] που τον ονόμασε bin completion. Ο αλγόριθμος αυτός κάνει πιο αποδοτική τη χρήση των σχέσεων κυριαρχίας. Όπως ο αλγόριθμος Martello και Toth, ο bin completion είναι επίσης ένας αλγόριθμος διακλάδωσης και οριοθέτησης, αλλά ερευνά ένα διαφορετικό κομμάτι του προβλήματος. Αντί να εξετάζει κάθε στοιχείο με τη σειρά του και να αποφασίζει σε ποιο κάδο θα το τοποθετήσει, ερευνά τον κάθε κάδο, και εξετάζει τα μη-κυριαρχούμενα εφικτά στοιχεία που θα μπορούσαν να χρησιμοποιηθούν για την συμπλήρωση αυτού του bin.

Ταξινομεί τα στοιχεία με φθίνουσα σειρά μεγέθους, και εξετάζει τους κάδους που περιέχουν κάθε στοιχείο με τη σειρά του, απαριθμώντας όλες τις μη-κυριαρχούμενες συμπληρώσεις αυτού του bin, και τις διακλαδώσεις, αν υπάρχουν περισσότερες από μία. Με άλλα λόγια, πρώτα συμπληρώνει τον κάδου που περιέχει το μεγαλύτερο στοιχείο, στη συνέχεια ολοκληρώσει τον κάδο που περιέχει το δεύτερο μεγαλύτερο στοιχείο, και ουτο καθεξής.

## 5.6 Πρόβλημα κάλυψης συνόλου (SCP - Set Covering Problem)

Πρόκειται για ένα κλασσικό πρόβλημα της συνδυαστικής, της επιστήμης των υπολογιστών και της θεωρίας της πολυπλοκότητας. Είναι ένα από τα 21 NP-complete προβλήματα του Καρπ (Richard Karp) και αποδείχτηκε ότι είναι NP-complete το 1972. Είναι ένα πρόβλημα που η μελέτη του έχει οδηγήσει στην ανάπτυξη σημαντικών τεχνικών για τον κλάδο των προσεγγιστικών αλγορίθμων.

Ορισμός: Δίνεται ένα σύνολο από στοιχεία *X={1,2,..,m}* (που ονομάζεται σύμπαν) και μια συλλογή συνόλων, *n={S1,S2,..,Sn}* των οποίων η ένωση ισοδυναμεί με το σύνολο των στοιχείων *X*. Το πρόβλημα της κάλυψης του συνόλου είναι να βρεθεί μια υπο-συλλογή συνόλων *n* όπου πάλι η ένωση τους δίνει το *X*.

Αν υποθέσουμε ότι σύμπαν μας είναι το X=*{1,2,3,4,5}* και η παρεχόμενη συλλογή συνόλων το n=*{{1,2,3},{2,4},{3,4},{4,5}}*, η ένωση του n μας δίνει το X. Ωστόσο, μπορούμε να καλύψουμε όλα τα στοιχεία του X με ένα μικρότερο αριθμό συνόλων από το n, για παράδειγμα: {1,2,3},{4,5} [43].

Στα προβλήματα βελτιστοποίησης, η είσοδος είναι μόνο το ζευγάρι (X,n) και ο σκοπός είναι να βρεθεί η κάλυψη με το λιγότερο αριθμό συνόλων. Τα δρομολόγια μιας αεροπορικής εταιρείας όπως θα δούμε και παρακάτω, η γραμμή παραγωγής μιας αυτοκινητοβιομηχανίας και άλλα μπορούν να αποτελέσουν προβλήματα κάλυψης συνόλου. [43]

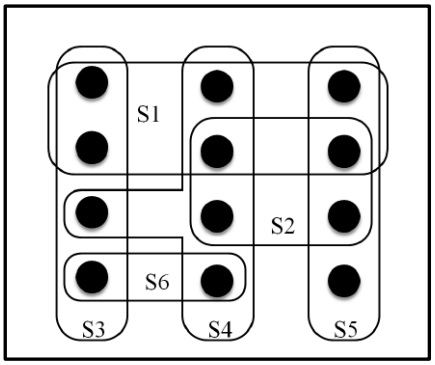
### 5.6.1 Greedy Set Cover

Το 1981 οι R. Bar-Yehuda και S.  παρουσίασαν έναν αλγόριθμο γραμμικής χρονικής προσέγγισης για το πρόβλημα της σταθμισμένης κάλυψης. Ο αλγόριθμος GreedySetCover προέρχεται από τη λύση τους [55].

**Αλγόριθμος 19: GREEDYSETCOVER**(*X,F*)[55]

```
1:  U ← X
2:  C ← 0
3:      while U ≠0  do      //while uncovered elements exist//
4:          select S ∈ F that maximizes |S ∩U|
5:          U ← U - S
6:          C ←C ∪ {S}
7:      end while
8:  return C      // C is the cover//
9:   end procedure
```

Ο αλγόριθμος *GREEDYSETCOVER* εχει ως  προτεραιότητα να επιλέγει τα  υποσύνολα που θα καλύπτουν, τα περισσότερα από τα ακάλυπτα στοιχεία και να τα προσθέτει στο σύνολο αποτελεσμάτων. Αυτός ο αλγόριθμος τελειώνει σε γραμμικό χρόνο.




*Σχήμα 41. Εφαρμογή του GREEDYSETCOVER σε 12 στοιχεία*

Στο σχήμα 41 παρουσιάζεται ένα σύνολο 12 στοιχείων (σύμπαν) και έξι υποσύνολα των οποίων η ένωση αποτελεί το σύμπαν. Σύμφωνα με τον άπληστο αλγόριθμο, όπως φαίνεται στο σχήμα 41, το πρώτο επιλεγμένο σύνολο είναι το S1, καθώς είναι το μεγαλύτερο υποσύνολο και καλύπτει αρκετά από τα στοιχεία του σύμπαντος. Στην συνέχεια επιλέγει το S4 που καλύπτει 3 επιπλέον στοιχεία. Κατόπιν θα επιλέξει το S5, ενώ στο τέλος η επιλογή θα είναι μεταξύ S6 ή S3, καθώς και τα δύο από αυτά καλύπτουν τον ίδιο αριθμό ακάλυπτων στοιχείων. Με αυτήν την προσέγγιση ο αριθμός των χρησιμοποιημένων υποσυνόλων είναι n= 4, ενώ ο βέλτιστη λύση θα ήταν n=3 υποσύνολα, S3, S4 και S5.

Μια σημαντική βελτίωση του προβλήματος δόθηκε με το υπο-πρόβλημα k-set. Σε αυτή την μέθοδο υπάρχει ο περιορισμός ότι ένα στοιχείο m $\in X=\{1,2,..,m\}$ θα εμφανίζεται μόνο σε κ-υποσύνολα.

### 5.6.2 Optimized Set Cover

Αν ένα στοιχείο m $\in X=\{1,2,..,m\}$ είναι μέλος μόνο σε ένα υποσύνολο Si, τότε είναι σαφές ότι το Si πρέπει να είναι μέλος της λύσης.

Με βάση αυτή την παρατήρηση, ο Optimized Set Cover εκτελεί τα παρακάτω βήματα:

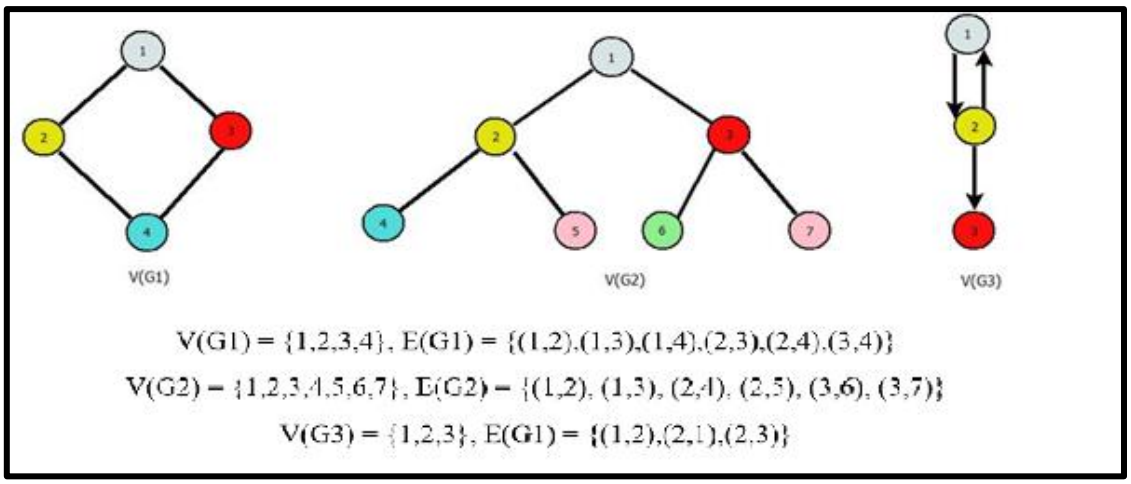
- βρίσκει όλα τα στοιχεία που ανήκουν σε ένα μόνο υποσύνολο
- ταξινομεί τα υποσύνολα με τον αριθμό στοιχείων που περιέχουν
- αφαιρεί όλα τα υποσύνολα που είναι λύσεις από το σύμπαν
- επανεκκινεί τον άπληστο αλγόριθμο

**Αλγόριθμος 20: OPTIMIZEDSETCOVER** (*X,F*)[55]

```
1:   U ← X
2:   C ← 0
3:   for all a ∈ X do   // preprocessing//
4:        if ∃!S ∈ F that a ∈ S then
5:             U ← U - S
6:             C ←C ∪ {S}
7:        end if
8:   end for
9:        while U ≠0  do       //while uncovered elements exist//
10:            select S ∈ F that maximizes |S ∩U|
11:            U ← U - S
12:           C ←C ∪ {S}
13:      end while
14:  return C       // C is the cover//
15:   end procedure
```

## 5.7 Πρόβλημα χρωματισμού κόμβων γράφου (Graph Coloring Problem)

Σε μία πρώτη προσέγγιση θα μπορούσαμε να πούμε ότι ένα γράφημα (graph) είναι ένα μαθηματικό ή συνδυαστικό αντικείμενο (mathematical or combinatorial object) που μπορεί απλά και εύκολα να αναπαρασταθεί με εικόνες ή, ισοδύναμα, που επιδέχεται μία απλή και εύκολη εικονογραφημένη αναπαράσταση (pictorial representation) [56][57].




*Σχήμα 41. Είδη γραφημάτων*

Ένα γράφημα G είναι ένα ζεύγος συνόλων V και E, όπου το σύνολο V είναι ένα πεπερασμένο σύνολο, που περιέχει ως στοιχεία τις κορυφές (vertices) ή κόμβους (nodes) ή σημεία (points) του γράφου. Το σύνολο E περιέχει τα ζεύγη που ενώνει τις κορυφές του γράφου, και ονομάζονται ακμές (edges) ή τόξα (arcs) ή σύνδεσμοι (links) του.

Οι κόμβοι ή οι ακμές ενός γράφου χαρακτηρίζονται από ένα μοναδικό όνομα που ονομάζεται ετικέτα (label). Ο αριθμός των ακμών που προσπίπτουν σε μια κορυφή λέγεται βαθμός της κορυφής$_i$. Ο μέγιστος βαθμός από όλες τις κορυφές ενός γραφήματος ονομάζεται βαθμός του γραφήματος.

Μονοπάτι μεταξύ δύο κορυφών ονομάζουμε μία ακολουθία ακμών, που είναι διαφορετικές μεταξύ τους. Το πλήθος των ακμών που περιέχονται σε ένα μονοπάτι ονομάζεται μήκος μονοπατιού.

Συνεκτικός γράφος, λέγεται ο γράφος του οποίου κάθε ζεύγος κορυφών συνδέεται με ένα τουλάχιστον μονοπάτι.

Χρωματισμός ενός γράφου είναι η ανάθεση ενός χρώματος σε κάθε αντικείμενο του γράφου, η οποία πληροί κάποιους περιορισμούς. Υπάρχουν οι παρακάτω γενικές κατηγορίες προβλημάτων που αφορούν τον χρωματισμό:

- χρωματισμός κόμβων (vertex coloring)
- χρωματισμός ακμών (edge coloring), και
- χρωματισμός περιοχών ενός επίπεδου γραφήματος (map coloring)

Ένας προφανής τρόπος να χρωματίσουμε τις κορυφές (ή τις ακμές και περιοχές) ενός γραφήματος, ώστε γειτονικές κορυφές να έχουν διαφορετικά χρώματα, είναι να

χρησιμοποιήσουμε ένα χρώμα για κάθε κορυφή. Ενδιαφερόμαστε όμως για χρωματισμούς που χρησιμοποιούν λιγότερα χρώματα.

Ονομάζουμε χρωματικό αριθμό ενός γραφήματος τον ελάχιστο αριθμό χρωμάτων με τον οποίο μπορούν να χρωματιστούν οι κορυφές του έτσι ώστε γειτονικές κορυφές να έχουν διαφορετικά χρώματα.

Ονομάζουμε χρωματικό δείκτη ενός γραφήματος τον ελάχιστο αριθμό χρωμάτων με τον οποίο μπορούν να χρωματιστούν οι ακμές του έτσι ώστε οι ακμές που προσπίπτουν στην ίδια κορυφή να έχουν διαφορετικά χρώματα.

Κορεσμός είναι ο αριθμός των χρωμάτων που θα πρέπει να χρησιμοποιηθούν για να χρωματίσουμε τους γειτονικούς κόμβους:

- Harder nodes: κόμβοι με υψηλό βαθμό κορεσμού
- Easiest node: κόμβοι με χαμηλό βαθμό κορεσμού

Το πρόβλημα υπολογισμού του χρωματικού αριθμού ενός γενικού γραφήματος είναι NP-πλήρες και, επομένως, δεν υπάρχει αποτελεσματικός αλγόριθμος για το χρωματισμό του γραφήματος (εκτός εάν P=NP ). Επειδή το πρόβλημα εμφανίζεται σε πολλές πρακτικές εφαρμογές και σε διάφορες εκδοχές, αναπτύχθηκαν αρκετοί ευριστικοί αλγόριθμοι που υπολογίζουν μία τιμή για το χρωματικό αριθμό, η οποία προσεγγίζει ικανοποιητικά την βέλτιστη.

Εφαρμογές:

- Εγκαταστάσεις δικτυών (υπολογιστών, οπτικά, κεραιών κινητής τηλεφωνίας)
- Αεροπορικές πτήσεις μεταξύ κάποιων πόλεων
- Επιλογή χρωμάτων στην δημιουργία χαρτών
- Πολλά παιχνίδια μπορούν να μοντελοποιηθούν με χρήση γράφων
- Traveling Salesman Problem (Πρόβλημα περιπλανώμενου πωλητή)

### 5.7.1 DSATUR[58]

Η άπληστη μέθοδος, είναι η απλούστερη μέθοδος χρωματισμού γράφων. Χρωματίζει τον πρώτο κόμβο που της δίνεται σαν είσοδος με το μικρότερης αξίας χρώμα και στην συνέχεια, ικανοποιώντας τους περιορισμούς του προβλήματος χρωματίζει τους γειτονικούς κόμβους με διαφορετικά χρώματα. Αυτή η μέθοδος όμως δεν αποδίδει πρακτικά σε γραφήματα με μεγάλο αριθμό κόμβων. Ο αλγόριθμος DSATUR χρησιμοποιεί πρώτα μια ευρετική μέθοδο, που αλλάζει την ταξινόμηση των κόμβων και στη συνέχεια χρησιμοποιεί την άπληστη μέθοδο για να χρωματίσει αυτούς τους κόμβους.

Αρχικά κατανέμει τους κόμβους σε υποσύνολα, βάση των κριτηρίων που του έχουν δοθεί. Επιλέγει το πιο «δύσκολο» υποσύνολο από άποψη μεγέθους, βαθμό κορεσμού και αχρωμάτιστους κόμβους και κατόπιν χρωματίζει τον κόμβο του υποσυνόλου με τον μικρότερο βαθμό κορεσμού.

**Αλγόριθμος 21: DSATUR** [58]

```
1:    Give an initial ordering of vertices as V = v1; v2:::vn;
2:    Find a largest clique V' of G, assign each vertex in V'
3:    a distinct color class; V = V - V'
4:    while V ≠ NULL do
5:        Find a vertex v in V, which is adjacent to the largest number of distinctly colored
      vertices,assign v to the lowest indexed color class that contains no vertices adjancent to v
6:        If (no existing color class to assignment to)
7:        create a new color class for v ;
8:        Move v out of V ;
9:    end while
10:   Return color classes
11:   end procedure
```

Το DSATUR είναι μια ακριβής μέθοδος για την εύρεση ενός βέλτιστου χρωματισμού για ένα γράφημα περιγράφοντας, εμμέσως, όλους τους πιθανούς χρωματισμούς. Είναι ένας απαριθμητικός αλγόριθμος στον οποίο οι κόμβοι επιλέγονται με βάση τον βαθμό κορεσμού.

### 5.7.2 Generic Tabu Search[59]

Ο παρακάτω αλγόριθμος στηρίζεται στην μέθοδο Tabu Search και αποτελείται από τρία κυρία τμήματα:

1. Μια τεχνική άπληστου αλγόριθμου για την αρχικό χρωματισμό του γράφου. Ο αλγόριθμος ξεκινά με ένα Dsatur άπληστο αλγόριθμο για να δημιουργήσει τις αρχικές συνθήκες. Αυτό το βήμα είναι πολύ γρήγορο και παρέχει με συνδυασμό με την μέθοδο Tabu Search μια πολύ καλύτερη αρχική συνθήκη από ότι ένας αλγόριθμος τυχαίας αναζήτησης.

2. Καθορισμός της νέας γενιάς. Σε αυτό το βήμα σκοπεύει να χρωματίσει γρήγορα ένα γράφημα με κ-1 συνδυασμούς και τις ελάχιστες συγκρούσεις χρωμάτων. Όταν το πρώτο βήμα μας δίνει ένα γράφημα χρωματισμένο με κ χρώματα, αντικαθιστά τους κόμβους που είναι χρωματισμένοι το κ-χρώμα με ένα κ-1 χρώμα, με τέτοιο τρόπο έτσι ώστε να υπάρχουν οι λιγότερες δυνατές συγκρούσεις χρωμάτων στο γράφημα (randomly break ties).

3. Αναζήτηση της καλύτερης λύσης. Με την μέθοδο Tabu search προσπαθεί να μειώσει τις συγκρούσεις στο μηδέν. Αν αυτό καταστεί εφικτό, ο αλγόριθμος έχει βρει μια σωστή λύση και επιστρέφει στο βήμα 2 για να επαναχρωματίσει το γράφημα με ένα λιγότερο χρώμα.

Ο αλγόριθμος σταματά όταν βρεθεί η βέλτιστη λύση ή όταν συμπληρωθεί ο αριθμός των επαναλήψεων που έχουμε ζητήσει. Σε αυτή την περίπτωση και πάλι μας δίνει το βέλτιστο αποτέλεσμα εκείνης της στιγμής.

**Αλγόριθμος 22: Generic TabuSearch** [59]

**Input:** *G, a praph*
**Output:** *NC, the minimum number of color used (minimum span for T-colorringsand set T-colorings is equal to NC-1)*
**Variables:**
*% f, f* objective function and its best value encountered so far*
*% s, s* current configuration and the best value encountered so far*
*% V*,l, MAX : candidate list , the size of tabu list and the limit of the iterations*
*%NC, the minimum number of color used*
*% color(s): returns the highest color value*

```
1:   M =0        /* initialize tabu matrix*/
2:   NB=0        /* initialize iteration counter*/
3:   s = generate() /*initial configuration generated with a greedy algorithm*/
4:   NC =colors(s)-1
5:   S=re-generate (s,NC) /*re-generation froms with NC colors */
6:   While (NB<MAX) do
7:          S*=s
8:          F*=f(s*)
9:          While (f*>0 and NB<MAX) do
10:                If (f(best_neighbor(s))<f*) then
11:                  s= best_neighbor(s)    /*Aspiration criterion, s(i,m)=new_v */
12:                 else
13:                 s= best_non_tabu_neighbor(s)    /* s(i,m)=new_v */
14:                M[i,m,old_v]=NB+1;  /*<u_i,m,old_v>become tabu*/
15:                l=a* |CD| +random(g);
16:                 If  (f(s)<f* then
18                 s*=s
19:                f*=f(s*)
20:                NB=NB+1
21:         If (f*=0) then
22:           NC=NC-1
23:           s=regenerate (s,NC)
24:          NB=0
25:  return NC+1
26:  end
```

## 5.7.3 Discrete Imperialist Competitive Algorithm (DICA)[60]

Ο DICA είναι μια εξέλιξη του αλγόριθμου ICA (Imperialist Competitive Algorithm) που ανήκει στην κατηγορία των εξελικτικών αλγόριθμών βελτιστοποίησης και αναζήτησης.

Ο κατακτητικός και ανταγωνιστικός αλγόριθμος (ICA), έχει χρησιμοποιηθεί σε πολλές εφαρμογές μηχανικής και βελτιστοποίησης. Είναι εμπνευσμένος από τον ανταγωνισμό των πληθυσμών για εξεύρεση χώρου και πόρων. Κάθε άτομο του πληθυσμού ονομάζεται χώρα και μπορεί να είναι είτε κατακτητής είτε αποικία. Οι αποικίες μαζί με τους κατακτητές αποτελούν τις αυτοκρατορίες. Η κίνηση των αποικιών προς τους κατακτητές και ο ανταγωνισμός, είναι οι δύο βασικές αρχές του ICA.

Ο ICA διατυπώνει το χώρο λύσης του προβλήματος ως χώρο αναζήτησης. Αυτό σημαίνει ότι κάθε σημείο του χώρου αναζήτησης είναι μια πιθανή λύση του προβλήματος. Το ICA στοχεύει να βρει τα καλύτερα σημεία στον χώρο αναζήτησης που ικανοποιούν το πρόβλημα. Επειδή ο ICA είναι σχεδιασμένος για την επίλυση συνεχών προβλημάτων, προκειμένου να χρησιμοποιηθεί στον πρόβλημα του χρωματισμού μεταλλάχθηκε σε DICA. Πειραματικά έχει αποδειχθεί ότι έχει καλύτερα αποτελέσματα από έναν απλό γενετικό αλγόριθμο.

Η είσοδος του αλγόριθμου DICA είναι ένα μη κατευθυνόμενο και ακυκλικό γράφημα G = (V, E) και η έξοδος είναι ο βέλτιστος και πιο αξιόπιστος χρωματισμός για το δεδομένο Graph Coloring Problem (GCP).

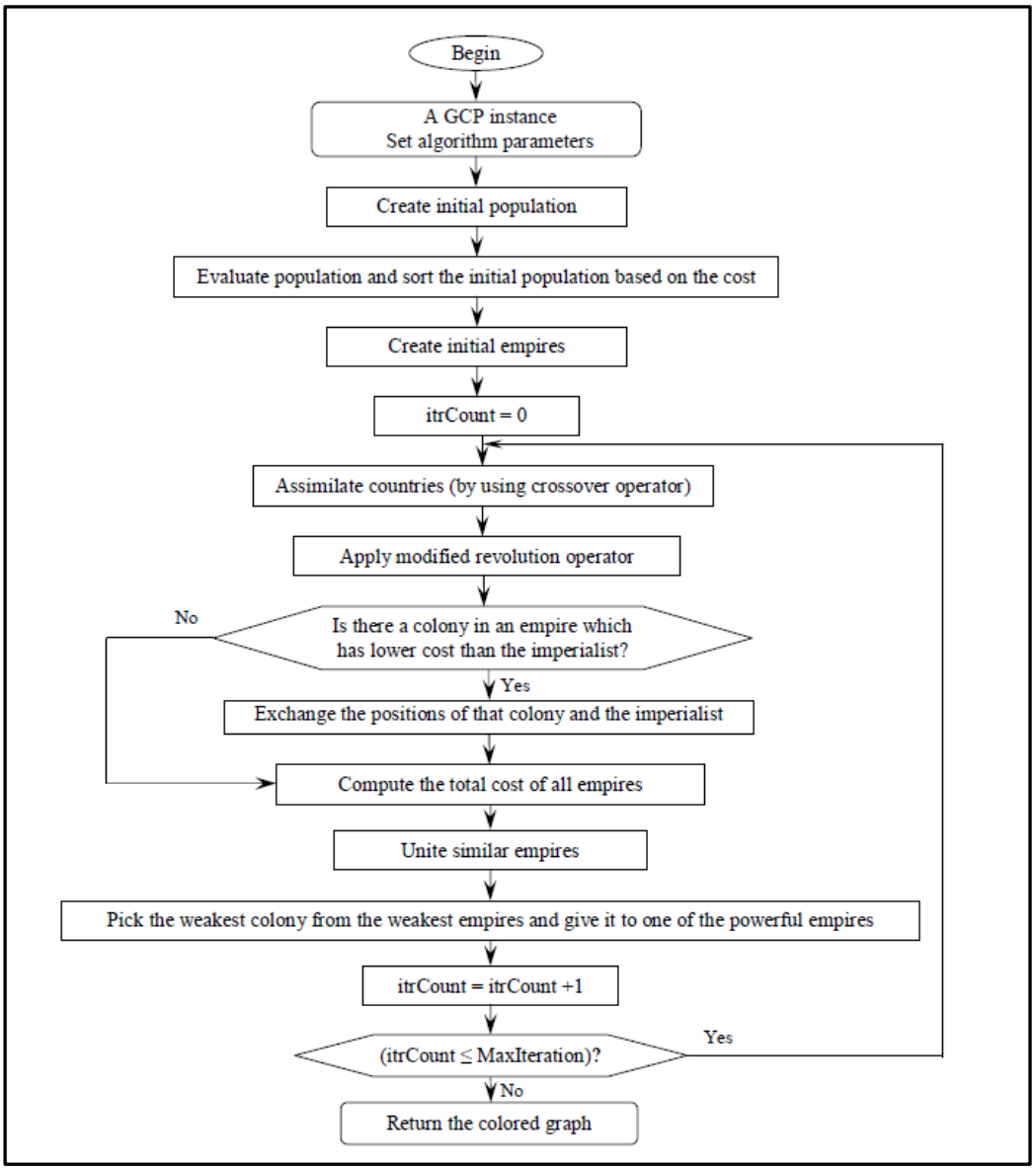
Κατά την έναρξη της διαδικασίας δημιουργείται ένας πληθυσμός χωρών $N_{pop}$. Εάν η παράμετρος GCP έχει *n* κορυφές τότε κάθε χώρα είναι ένας πίνακας n-διαστάσεων. Κάθε στοιχείο των χωρών ισοδυναμεί με ένα δείκτη χρώματος. Μετά τη δημιουργία του αρχικού πληθυσμού, οι χώρες αξιολογούνται, σύμφωνα με τη λειτουργία του κόστους, που εκφράζεται ως εξής:

$$\text{Cost(country)} = \begin{cases} \max\limits_{i=1}^{N} & \text{if conflict} = 0 \\ \text{conflict} \times \text{p} + \max\limits_{i=1}^{N} & \text{if conflict} \neq 0 \end{cases}$$

Όπου p είναι ο συντελεστής ποινικοποίησης και N είναι ο αριθμός κορυφών του γραφήματος. Υπολογίζουμε πόσα μοναδικά χρώματα χρησιμοποιούνται σε μια χώρα και γίνεται μια ταξινόμηση των χωρών, βάση αυτού του αριθμού. Στη συνέχεια, ορισμένες από τις καλύτερες χώρες επιλέγονται για να είναι κατακτητές και ο υπόλοιπος πληθυσμός σχηματίζει τις αποικίες αυτών των κατακτητών. Τα καλύτερα κράτη μαζί με τις αποικίες τους διαμορφώνουν τις αυτοκρατορίες.

Μέσα στην κύρια επανάληψη του αλγορίθμου, οι κατακτητές προσπαθούν να προσελκύσουν τις αποικίες προς τον εαυτό τους και να βελτιώσουν το κόστος τους. Κατά τη διάρκεια αυτής της κίνησης, αν μια αποικία φτάνει σε μια κατάσταση που έχει μικρότερο κόστος από ότι ο αντίστοιχος κατακτητής, τότε ανταλλάσσουν την θέση τους. Μετά την αφομοίωση, ο ανταγωνισμός αρχίζει και όλες οι αυτοκρατορίες προσπαθούν να πάρουν την κατοχή αποικιών άλλων (αδύναμων) αυτοκρατοριών υπό τον έλεγχο τους. Κατά τη διάρκεια αυτού του διαγωνισμού, οι αυτοκρατορίες που είναι πιο αδύναμες από τις άλλες, χάνουν τις αποικίες τους. Το αποτέλεσμα αυτής της διαδικασίας είναι η εξαφάνιση των πιο αδύναμων αυτοκρατοριών.

Ο DICA τρέχει για ένα σταθερό αριθμό επαναλήψεων, όπου μια αναπαραγωγή ορίζεται ως ένας κύκλος αφομοίωσης, επανάστασης, ανταλλαγής, ανταγωνισμού και βήματα απομάκρυνσης. Το σχήμα 42 συνοψίζει τη διαδικασία χρήσης αλγόριθμου διακριτών ιμπεριαλιστικών ανταγωνιστών στο πρόβλημα χρωματισμού γραφήματος.




*Σχήμα 42. Διάγραμμα ροής DICA*[60]

## 5.8 Πρόβλημα δέσμευσης πόρων (Unit Commitment Problem)

Το πρόβλημα δέσμευσης πόρων (Unit Commitment Problem), είναι ένα από τα πιο δύσκολα προβλήματα βελτιστοποίησης, διότι η λύση του επηρεάζεται από ορισμένους ιδιαίτερους περιορισμούς, που επιβάλλονται από τα συστήματα παραγωγής, τις συνθήκες μεταφοράς και από την ζήτηση των καταναλωτών. Η επίλυση του προβλήματος UC είναι σημαντική τόσο για την διαδικασία λήψης αποφάσεων, όσο και από την πλευρά των συστημάτων παραγωγής για ελαχιστοποίηση του κόστους.

Το πρόβλημα αφορά, κυρίως, την παραγωγή ηλεκτρικού ρεύματος μέσω της κατάρτισης ενός ημερήσιου ή εβδομαδιαίου προγράμματος λειτουργίας που προσδιορίζει ανά ώρα το επίπεδο λειτουργίας των διαθέσιμων γεννητριών. Το πρόγραμμα θα πρέπει να ικανοποιεί περιορισμούς όπως η εκτιμώμενη κατανάλωση ανά περίοδο, να μην εισάγει

ανεπιθύμητες αιχμές στο ηλεκτρικό δίκτυο και να λαμβάνονται υπόψη κανόνες λειτουργίας των γεννητριών που προβλέπουν για αυτές χρόνους ενεργοποίησης και απενεργοποίησης.

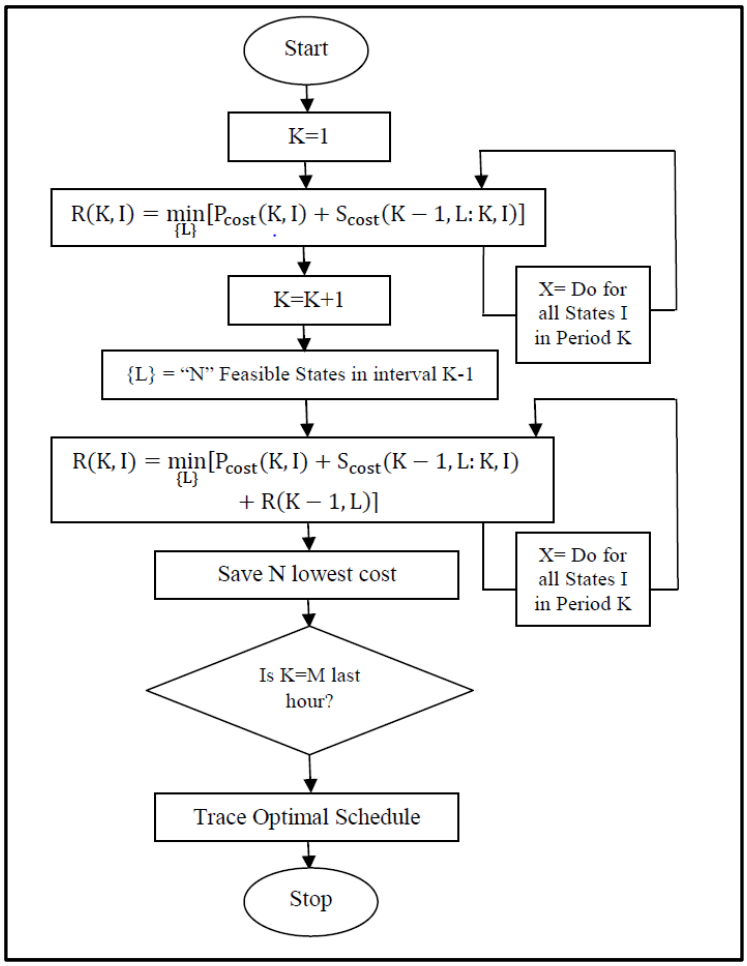
Η μεγάλη διάσταση του χώρου αναζήτησης λύσεων και η συνδυαστική φύση του προβλήματος δέσμευσης πόρων περιορίζουν τις προσπάθειες να αναπτυχθεί οποιαδήποτε αυστηρή μαθηματική βελτιστοποίηση ικανή να λύσει το πρόβλημα ολοκληρωτικά για οποιοδήποτε πραγματικό σύστημα. Παρακάτω ακολουθεί μια σύντομη ματιά στις μεθόδους επίλυσης προβλημάτων του UC στις πρόσφατες βιβλιογραφίες [61], [62]:

- **Priority List (PL)** ταξινομεί με αύξουσα σειρά τις μονάδες παραγωγής με κριτήριο το κόστος λειτουργίας σε κατάσταση πλήρους φόρτου, έτσι ώστε πρώτα να δεσμευτούν οι πιο οικονομικές μονάδες προκειμένου να ικανοποιηθεί η ζήτηση. Η μέθοδος PL είναι πολύ γρήγορη και οικονομική σε μνήμη αλλά εξαιρετικά ευρετική και παρέχει χρονοδιαγράμματα με σχετικά υψηλότερο κόστος λειτουργίας καθώς δεν ελέγχει όλους τους πιθανούς συνδυασμούς παραγωγής.
- Η μέθοδος **branch-and-bound (BB)** δίνει καλά αποτελέσματα για μικρά συστήματα αλλά απαιτεί μεγάλο χώρο αποθήκευσης δεδομένων και μεγάλο χρόνο επεξεργασίας με αποτέλεσμα να είναι ασύμφορη για μεγάλα συστήματα.
- Η μέθοδος **Lagrangian Relaxation (LR)** επικεντρώνεται στην εξεύρεση μιας κατάλληλης τεχνικής συντονισμού για τη δημιουργία μιας αρχικής λύσης. Το κύριο πρόβλημα με τη μέθοδο LR είναι η δυσκολία που συναντάμε στη εύρεση εφικτών λύσεων.
- **Δυναμικός προγραμματισμός**. Η μέθοδος μας δίνει πάντα την βέλτιστη λύση για μικρά συστήματα αλλά έχει το μειονέκτημα ότι κάθε επόμενη λύση βασίζεται στην αξιοπιστία της προηγούμενης.
- **Στοχαστικοί αλγόριθμοι**. Έχουν το πλεονέκτημα ότι αναζητούν το χώρο λύσεων πιο προσεκτικά και αποφεύγουν τον εγκλωβισμό σε τοπικά ελάχιστα και μπορούν να βρουν όχι μόνο τοπικές βέλτιστες λύσεις αλλά και την πιο βέλτιστη λύση. Η κύρια δυσκολία είναι η ευαισθησία τους στην επιλογή των παραμέτρων. Καθώς και ότι σε περίπτωση προβλήματος μεγάλης κλίμακας, καταναλώνουν πολύ χρόνο και χώρο εξαιτίας του επαναληπτικού τους χαρακτήρα.

### 5.8.1 Δυναμικός προγραμματισμός

Σε μια συγκεκριμένη χρονική περίοδο, οι συνδυασμοί των μονάδων παραγωγής ονομάζονται καταστάσεις. Στο Forward Dynamic programming ένα εξαιρετικά οικονομικό χρονοδιάγραμμα επιτυγχάνεται, ελέγχοντας σε κάθε στάδιο το συνολικό κόστος, και στη συνέχεια με αναδρομή βλέπει και συγκρίνει το στάδιο αυτό με το πιο οικονομικό στάδιο που έχει βρει μέχρι εκείνη την στιγμή, έτσι ώστε να προγραμματίσει το επόμενο στάδιο.

Κάθε στάδιο συνήθως αντιστοιχεί σε μία ώρα λειτουργίας. Η μέθοδος αποθηκεύει τους συνδυασμούς των μονάδων με τα κόστη τους και την ζήτηση και προχωρά στις ρυθμίσεις των μονάδων για την επόμενη ώρα.




*Σχήμα 43. Διάγραμμα ροής δυναμικού προγραμματισμού* [63]

Η λειτουργία του δυναμικού προγραμματισμού, για ένα σύστημα παραγωγής με 10 γεννήτριες, διαθέσιμες σε 24ώρη βάση, εκτελεί τα παρακάτω βήματα:

- Ξεκίνα τυχαία εξετάζοντας οποιεσδήποτε δύο μονάδες παραγωγής
- Κατασκεύασε πίνακα με τον πιο οικονομικό συνδυασμό παραγωγής των δύο μονάδων με διακριτά επίπεδα φορτίου
- Προσδιόρισε τον οικονομικότερο συνδυασμό των δύο μονάδων για όλα τα επίπεδα φορτίου, με την παρατήρηση ότι σε κάθε επίπεδο φορτίου, η οικονομική λειτουργία μπορεί να είναι ενεργοποιώντας είτε μια μονάδα είτε και τις δύο μονάδες παραγωγής
- Σχημάτισε την πιο αποδοτική καμπύλη κόστους για τις δύο μονάδες και χαρακτήρισέ το σαν μια ενιαία μονάδα παραγωγής
- Πρόσθεσε και τρίτη μονάδα παραγωγής, βρες την νέα καμπύλη κόστους και επανέλαβε την διαδικασία μέχρι να προστεθούν όλες οι μονάδες παραγωγής

Το πλεονέκτημα αυτής της μεθόδου είναι ότι έχοντας τον καλύτερο τρόπο εκτέλεσης μονάδων N, είναι απλό να βρούμε τον καλύτερο τρόπο λειτουργίας

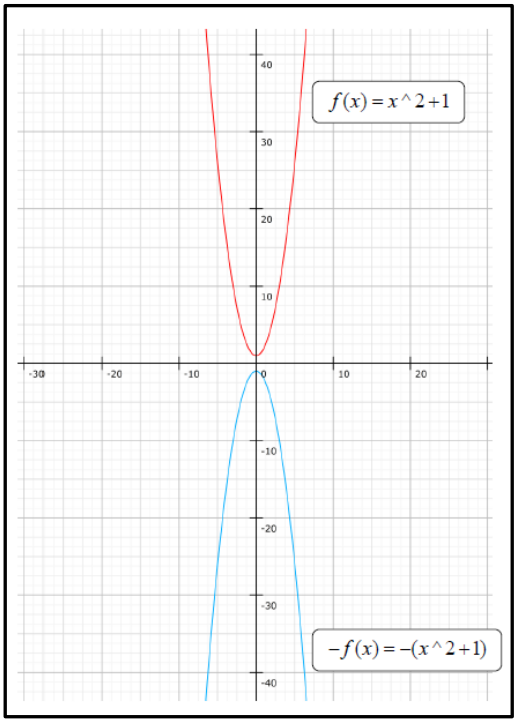
### 5.8.2 Harmony Search Algorithm

Ο αλγόριθμος αναζήτησης αρμονίας βασίζεται σε φυσικές διεργασίες μουσικής απόδοσης, που συμβαίνουν όταν ο μουσικός ψάχνει για μια καλύτερη κατάσταση αρμονίας.

Ο αλγόριθμος HS περιλαμβάνει έναν αριθμό τελεστών βελτιστοποίησης, η οποία αποθηκεύει τις εφικτές λύσεις σε πίνακες, το μέγεθος μνήμης αρμονίας (HMS) που είναι ο αριθμός των λύσεων στην μνήμη, η ταχύτητα προσαρμογής της μνήμης αρμονίας (HMCR) και η ταχύτητα προσαρμογής του βήματος (PAR). Δημιουργείται νέος πίνακας επιλέγοντας τυχαία τα στοιχεία διαφόρων διανυσμάτων στη μνήμη αρμονίας. Ο αλγόριθμος HS κατευθύνεται αποτελεσματικά χρησιμοποιώντας δύο παραμέτρους, τις HMCR και PAR

## 5.9 Συνάρτηση ν μεταβλητών της οποίας ψάχνουμε το μέγιστο

Το πρόβλημα της βελτιστοποίησης μιας συνάρτησης έγκειται στην εύρεση του σημείου *x*, στο οποίο η συνάρτηση *f(x)* παίρνει την ελάχιστη ή την μέγιστη τιμή της. Αν βρούμε το ελάχιστο σημείο *x* για μια συνάρτηση *f*, τότε η συνάρτηση *-f(x)* θα έχει μέγιστο στο ίδιο σημείο. Έτσι έχοντας ένα αλγόριθμο που υπολογίζει το ελάχιστο μπορούμε τα υπολογίσουμε το μέγιστο.




*Σχήμα 44.Γραφική αναπαράσταση συναρτήσεων* [64]

Οι τεχνικές της βελτιστοποίησης αφορούν τη μαθηματική διατύπωση και την επίλυση προβλημάτων μεγιστοποίησης (ή ελαχιστοποίησης) μιας αντικειμενικής συνάρτησης.

Υπάρχουν περιπτώσεις όπου οι επιτρεπτές τιμές των παραμέτρων της συνάρτησης είναι διακριτές ή ακέραιες, και άλλες όπου οι παράμετροι είναι συνεχείς μεταβλητές, αλλά και μικτά προβλήματα όπου μερικές παράμετροι παίρνουν διακριτές και οι υπόλοιπες συνεχείς τιμές [6].

Στα προβλήματα βελτιστοποίησης συνεχών συναρτήσεων, η ύπαρξη πρώτων ή και δεύτερων παραγώγων επιτρέπουν την ανάπτυξη αποτελεσματικών μεθόδων επίλυσης με υψηλές επιδόσεις.

Σε περιπτώσεις όπου οι παράμετροι του προβλήματος είναι ανεξάρτητες μεταξύ τους, η βελτιστοποίηση δεν υπόκειται σε περιορισμούς και η διαδικασία επίλυσης ονομάζεται «βελτιστοποίηση χωρίς περιορισμούς». Αν οι παράμετροι δεν είναι μεταξύ τους ανεξάρτητες αλλά συνδέονται μέσω κάποιων συναρτησιακών σχέσεων (ισοτήτων ή ανισοτήτων) τότε οι σχέσεις αυτές ονομάζονται περιορισμοί και η διαδικασία «βελτιστοποίηση με περιορισμούς».

Στα προβλήματα χωρίς περιορισμούς, κάποιες μέθοδοι ελαχιστοποίησης αναλύονται χωρίς χρήση παραγώγων της αντικειμενικής συνάρτησης, όμως οι περισσότερες βασίζονται στη χρήση πρώτων ή και δεύτερων παραγώγων. Σε προβλήματα όπου έχουμε περιορισμούς, υπάρχουν προσεγγίσεις που χρησιμοποιούν μεθόδους χωρίς περιορισμούς με κατάλληλα τροποποιημένη την αντικειμενική συνάρτηση. Πιο συγκεκριμένα, μελετώνται οι «μέθοδοι ποινής» όταν υπάρχουν περιορισμοί ισότητας και οι «μέθοδοι φραγής» στην περίπτωση ανισοτικών περιορισμών.

### 5.9.1 Βασικές Έννοιες

**Παράγωγος συνάρτησης μιας μεταβλητής**: Μια συνάρτηση $f(x)$: $R \rightarrow R$ είναι παραγωγίσιμη στο σημείο $x_0$, όταν και μόνον όταν υπάρχει το όριο :

$$\lim_{x \to x_0} \frac{f(x)-f(x_0)}{x-x_0}$$

και είναι πεπερασμένο. Το όριο αυτό ονομάζεται παράγωγος $df/dx=f'(x)$ της $f$ στο $x_0$.

Η παράγωγος εκφράζει τη σχετική μεταβολή της τιμής μιας συνάρτησης ως προς τις αλλαγές των τιμών των μεταβλητών της. Γεωμετρικά, η παράγωγος σε ένα σημείο συνάρτησης μιας μεταβλητής είναι η κλίση της εφαπτόμενης γραμμής στη γραφική παράσταση της συνάρτησης στο σημείο αυτό. Έτσι, η παράγωγος είναι η καλύτερη γραμμική προσέγγιση της συνάρτησης στη γειτονιά του εν λόγω σημείου.

Η έννοια της παραγώγου μπορεί να γενικευθεί σε συναρτήσεις πολλών πραγματικών μεταβλητών. Εκεί είναι αναγκαίος ο διαχωρισμός στη μερική παράγωγο ως προς κάθε μεταβλητή, η οποία ορίζεται κατά αντιστοιχία με τα παραπάνω για σταθερές όλες τις υπόλοιπες μεταβλητές, και την ολική παράγωγο της συνάρτησης, η οποία αναφέρεται στη «μέση» μεταβολή της συνάρτησης ως προς όλες τις μεταβλητές.

**Ολική παράγωγος μιας συνάρτησης πολλών μεταβλητών** $f(x)$: $R^n \rightarrow R$ η οποία είναι παραγωγίσιμη σε κάθε κατεύθυνση του χώρου $R^n$, ορίζεται στο σημείο $X_0 = (x_{10}, x_{20}, \ldots, x_{n0})$ ως ίση με $df(X)=\nabla f(X)\cdot dX$, όπου:

$$\nabla f(X) = \left[ \frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_n} \right]$$

είναι η κλίση της συνάρτησης στο συγκεκριμένο σημείο. Όταν έχουμε διανυσματικές συναρτήσεις, η ολική παράγωγος είναι ουσιαστικά ένας πίνακας διαστάσεων [m x 1] με στοιχεία (υποπίνακες) τις κλίσεις κάθε συνιστώσας της συνάρτησης.

Ολική παράγωγος $2^{ης}$ τάξης, συνάρτησης n μεταβλητών μιας συνάρτησης $f(x)$: $R^n \rightarrow R$ ορίζεται ίση με $d^2f(X)=dX^T\cdot\nabla 2f(X)\cdot dX$, όπου:

$$\nabla^2 f(X) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

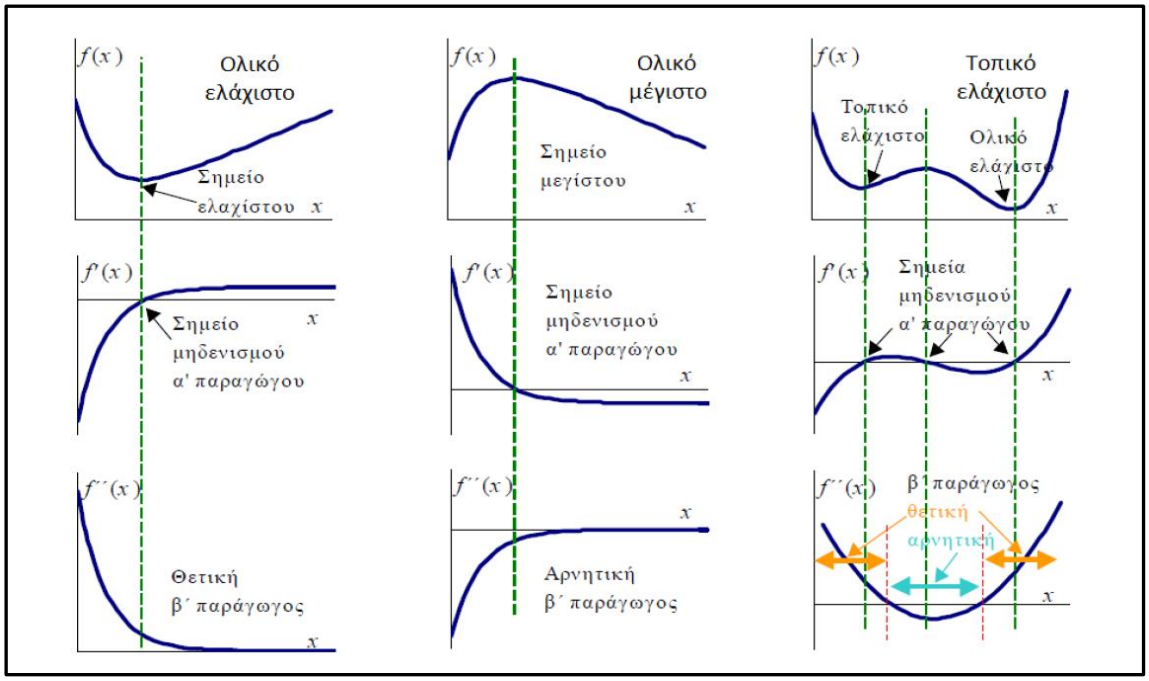
*είναι* ο Εσσιανός πίνακας της *f(X)*.

**Ορισμός ακροτάτων συνάρτησης**: Έστω $f: D \leq R^n \rightarrow R$. Ένα σημείο $X^*$ καλείται **τοπικό** ελάχιστο ή μέγιστο αν υπάρχει $\rho > 0$ ώστε $f(X^*) \leq f(X)$ ή $(f(X^*) \geq f(X))$ για κάθε $X \in D(X^*, \rho)$. ενώ καλείται **ολικό** ελάχιστο ή μέγιστο αν η συνθήκη ισχύει για κάθε $X \in D$.

Ο υπολογισμός ακροτάτων αποτελεί σημαντικό κομμάτι για αρκετές από τις τεχνικές βελτιστοποίησης. Η συνήθης οδός είναι μέσω της εύρεσης των κρίσιμων σημείων στις ρίζες των μερικών παραγώγων 1ης τάξης, εξετάζοντας ακολούθως τα πρόσημα των μερικών παραγώγων 2ης τάξης.

**Θεώρημα διαλογής ακροτάτων συνάρτησης:** Ένα κρίσιμο σημείο $X^*$ της $f(x)$: $R^n \rightarrow R$ είναι ολικό ελάχιστο ή μέγιστο αν ο $\nabla^2 f$ είναι θετικά ή αρνητικά, ημιορισμένος στο $R^n$.

Στο σχήμα 45 βλέπουμε ορισμένες περιπτώσεις ακροτάτων και τις αντίστοιχες σχέσεις που ικανοποιούν οι παράγωγοι 1ης και 2ης τάξης για συναρτήσεις μιας μεταβλητής.




*Σχήμα 45. Συσχετισμός ακροτάτων με τις α' και β' παραγώγους*[6]

## 5.9.2 Επισκόπηση επαναληπτικών μεθόδων αναζήτησης

Οι μέθοδοι που εφαρμόζονται σε προβλήματα βελτιστοποίησης χωρίς περιορισμούς έχουν στόχο τον εντοπισμό τοπικών ελαχίστων μέσω της επαναληπτικής αναζήτησης προς κατευθύνσεις συγκλίνουσες με την πιθανή λύση του προβλήματος. Ανάλογα με την πληροφορία που χρησιμοποιούν, οι τεχνικές αυτές χωρίζονται στις εξής κατηγορίες:

- Τεχνικές που κάνουν χρήση μόνο τιμών της αντικειμενικής συνάρτησης, οι οποίες συνήθως χαρακτηρίζονται ως «άμεσες μέθοδοι»
- Τεχνικές που χρησιμοποιούν τιμές της αντικειμενικής συνάρτησης και των πρώτων και δεύτερων μερικών παραγώγων της, και αναφέρονται ως «έμμεσες μέθοδοι»

Στις περισσότερες περιπτώσεις προτιμώνται οι έμμεσες μέθοδοι. Οι άμεσες μέθοδοι χρησιμοποιούνται αν οι παράγωγοι της συνάρτησης είναι ασυνεχείς ή δεν ορίζονται, και επίσης σε εφαρμογές όπου η συνάρτηση δεν είναι γνωστή αναλυτικά.

#### 5.9.1.1 Άμεσες μέθοδοι

Οι άμεσες μέθοδοι αναζήτησης μιας διάστασης έχουν στηριχθεί στην παραπάνω λογική, με τη διαφορετικότητά τους να έγκειται στον τρόπο επιλογής των εσωτερικών σημείων. Αναφέρουμε ακολούθως τις βασικότερες από αυτές τις μεθόδους:

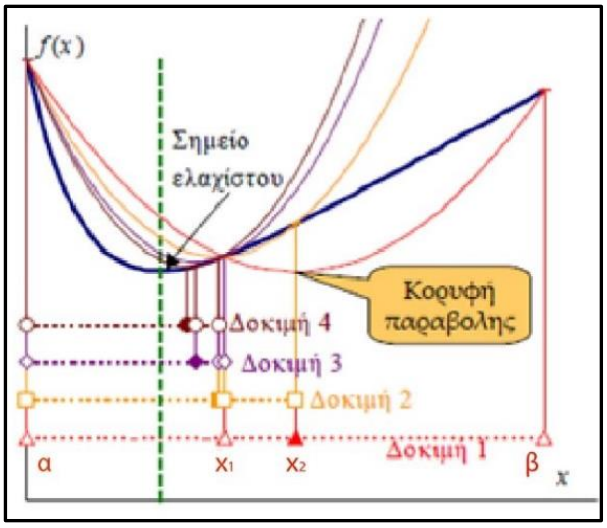
- **Ομοιόμορφη αναζήτηση:** Το διάστημα αναζήτησης [a, β] διαμερίζεται σε n ίσα τμήματα εύρους δ = (β - α)/n, σχηματίζοντας ένα πλέγμα n -1 ενδιάμεσων σημείων όπου υπολογίζονται οι τιμές της *f*. Αν $\hat{x}$ το σημείο που αντιστοιχεί στη μικρότερη τιμή της *f*, τότε το θεώρημα υποδεικνύει ότι το ελάχιστο $x^*$ ανήκει στο διάστημα $(\hat{x}-\delta, \hat{x}+\delta)$. Η ακρίβεια στον προσδιορισμό του $x^*$ είναι ανάλογη του εύρους διαμέρισης δ, άρα και αντιστρόφως ανάλογη του αριθμού τμημάτων n. Η δυσμενής επίπτωση στην απόδοση της μεθόδου μπορεί να αντιμετωπιστεί με σταδιακή μείωση του δ όσο πλησιάζουμε προς το όριο $|\hat{x}-x^*|<\varepsilon$, κάτι βέβαια που μετατρέπει τη μέθοδο σε ανομοιόμορφη.

- **Μέθοδος χρυσής τομής:** Το [a, β] χωρίζεται σε τρία άνισα υποδιαστήματα μέσω δύο εσωτερικών σημείων που επιλέγονται να ισαπέχουν από τα άκρα $(x_1-\alpha = \beta - x_2 = \rho)$. Τα αρχικά διαστήματα αναζήτησης είναι τα $[\alpha, x_2]$ και $[x_1, \beta]$, ενώ όλα τα επόμενα επιλέγονται με δύο παραδοχές: Το εύρος κάθε νέου διαστήματος να συνδέεται με το προηγούμενο με σταθερά αναλογίας $\gamma \in (0, 1)$, και το κάθε προηγούμενο ενδιάμεσο σημείο του οποίου το διάστημα δεν επιλέγεται να αποτελεί ενδιάμεσο σημείο κάθε νέου διαστήματος. Με αυτό τον τρόπο, οι υπολογισμοί τιμών της *f* είναι λιγότεροι, ενώ ως σταθερά αναλογίας προκύπτει (ως λύση της εξίσωσης $\gamma^2 + \gamma - 1 = 0$) ο λόγος χρυσής τομής $\gamma_{GM}=(\sqrt{5}-1)/2$. Μια παραλλαγή της μεθόδου χρυσής τομής είναι ο αλγόριθμος Fibonacci, με τη βασική διαφορά ότι απαιτεί έναν επιπλέον υπολογισμό της *f* μετά τη 2η επανάληψη, αλλά και ότι ο συντελεστής $\gamma$ δεν παραμένει σταθερός.

- **Μέθοδος παραβολικής παρεμβολής:** Στη μέθοδο αυτή, εντοπίζεται καταρχήν ένα ενδιάμεσο σημείο $x_1$ τέτοιο ώστε $f(\alpha) \geq f(x1)$ και $f(x1) \leq f(\beta)$, και μετά παρεμβάλουμε την (μοναδική) παραβολική καμπύλη που περνά από τα 3 σημεία και επιλέγουμε ως νέο ενδιάμεσο σημείο $x_2$ την κορυφή της παραβολής (η οποία και την ελαχιστοποιεί). Η συγκεκριμένη διαδικασία μπορεί να εκφραστεί μαθηματικά μέσω του τύπου:

$$x_2 = \frac{\alpha+\beta}{2} - \frac{x_1-\beta}{2}\,\frac{f(\beta)-f(\alpha)}{\beta-\alpha}\left[\frac{f(x_1)-f(\alpha)}{x_1-\alpha} - \frac{f(\beta)-f(\alpha)}{\beta-\alpha}\right]^{-1}$$

Ανάλογα με τη σχετική διάταξη των $x_1$ *και* $x_2$, το επόμενο ενδιάμεσο σημείο επιλέγεται ανάμεσα στα δυο αυτά σημεία και γίνεται νέα παρεμβολή. Η διαδικασία συνεχίζεται, και τερματίζεται όταν ικανοποιηθεί η ζητούμενη ακρίβεια.




*Σχήμα 46. Αναζήτηση ελάχιστου με τη μέθοδο της παραβολικής παρεμβολής*

Μέθοδος παραβολικής παρεμβολής (σχήμα 46):

1. Ξεκινάμε από τρία σημεία *α, β,* $x_1$.
2. Πυκνώνουμε με ένα ακόμη σημείο $x_2$ , προσαρμόζοντας παραβολή στα τρία γνωστά σημεία *α, β,* $x_1$, και υπολογίζουμε την κορυφή της παραβολής
3. Εξετάζουμε ποιο από τα δυο ενδιάμεσα σημεία έχει την μικρότερη τεταγμένη.
4. Κρατάμε το σημείο αυτό και δύο εναλλάξ.
5. Σταματάμε, όταν το διάστημα που ορίζουν τα δυο ακραία σημεία προς το άθροισμα των τετμημένων των δύο ενδιάμεσων σημείων (κατ' απόλυτη τιμή) γίνει μικρότερο από μια τιμή ανοχής.

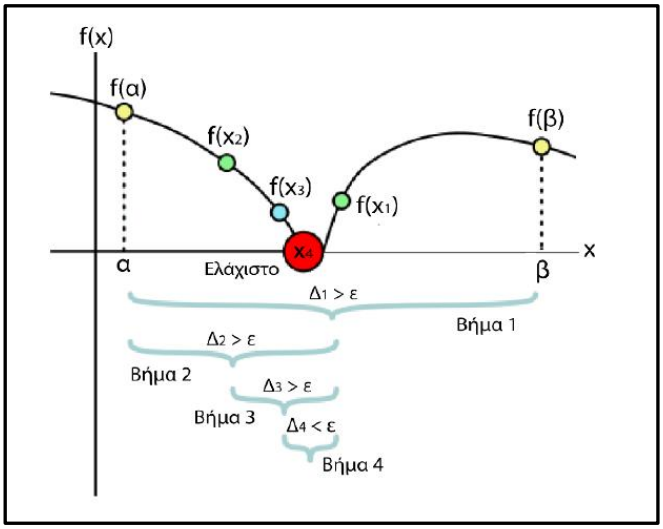
### 5.9.1.2 Έμμεσες μέθοδοι[6]

Οι έμμεσες μέθοδοι έχουν κοινό στοιχείο το ότι συνυπολογίζουν τις παραγώγους της συνάρτησης σε σημεία του διαστήματος αναζήτησης. Αυτό γίνεται είτε στο μήκος κύματος των άμεσων μεθόδων, αποφασίζοντας ποιο θα είναι το επόμενο διάστημα αναζήτησης με βάση τις τιμές και τα πρόσημα των παραγώγων της f, είτε στη λογική της κατευθυντικής αναζήτησης επί των τιμών της f: Ξεκινώντας από το αρχικό σημείο, ορίζεται βάσει ενός σταθερού κανόνα μια κατεύθυνση επάνω στην οποία εντοπίζεται νέο σημείο που επιτυγχάνει μείωση της συνάρτησης. Η διαδικασία συνεχίζεται με τον εντοπισμό νέας κατεύθυνσης από το νέο σημείο, ώστε να προκύπτει μια ακολουθία σημείων που προσεγγίζει σταδιακά τη λύση. Η διαφοροποίηση μεταξύ των μεθόδων έγκειται στον κανόνα βάσει του οποίου επιλέγεται η κατεύθυνση αναζήτησης.

Παρουσιάζουμε δύο από τις μεθόδους της κατηγορίας:

- ***Μέθοδος διχοτόμησης:*** Σε αυτή τη μέθοδο, επιλέγουμε αρχικό διάστημα αναζήτησης στο οποίο η *f* έχει συγκεκριμένο τύπο καμπυλότητας. Ύστερα, χωρίζουμε το διάστημα σε δύο ισομερή υποδιαστήματα με τη βοήθεια του

εσωτερικού σημείου $x_1 = (\alpha + \beta)/2$, και υπολογίζουμε την τιμή $f'(x_1)$ της πρώτης παραγώγου. Εξετάζουμε ακολούθως τις εξής περιπτώσεις: Αν $f'(x_1) = 0$ τότε το $x_1$ *είναι το ζητούμενο ελάχιστο, αν* $f'(x_1) > 0$ *τότε για* $x < x_1$ *θα είναι* $f(x) < f(x_1)$ και ως επόμενο διάστημα αναζήτησης θα πρέπει να ληφθεί το $[\alpha, x_1]$, *ενώ αν* $f'(x_1) < 0$ *τότε για* $x < x_1$ *θα είναι* $f(x) > f(x_1)$ και ως επόμενο διάστημα αναζήτησης θα πρέπει να ληφθεί το $[x_1, \beta]$. Η εφαρμογή της μεθόδου συνεχίζεται μέχρις ότου φτάσουμε σε διάστημα αναζήτησης με μήκος μικρότερο της καθορισμένης ακρίβειας $\varepsilon$. Ο απαιτούμενος αριθμός βημάτων για αυτό υπολογίζεται ως ο μικρότερος ακέραιος που ικανοποιεί τη σχέση $(1/2)^m \leq \varepsilon/(\beta - \alpha)$.




*Σχήμα 47. Μέθοδος διχοτόμησης*[6]

Μέθοδος διχοτόμησης (σχήμα 47):

1. Αναζητούμε στο διάστημα [α, β], το σημείο $x_1=(\alpha+\beta)/2$, οπου η πρώτη παράγωγος $f(x_1)$ είναι θετική και το διάστημα αναζήτησης μεγαλύτερο από μια προκαθορισμένη τιμή
2. Αναζητούμε στο διάστημα [α, $x_1$], το σημείο $x_2=(\alpha+ x_1)/2$, όπου η πρώτη παράγωγος $f(x_2)$ είναι αρνητική και το διάστημα αναζήτησης μεγαλύτερο από μια προκαθορισμένη τιμή
3. Αναζητούμε στο διάστημα [$x_2$, $x_1$], το σημείο $x_3=( x_2+ x_1)/2$, όπου η πρώτη παράγωγος $f(x_3)$ είναι αρνητική και το διάστημα αναζήτησης μεγαλύτερο από μια προκαθορισμένη τιμή
4. Αναζητούμε στο διάστημα [$x_3$, $x_1$], το σημείο $x_4=( x3+ x_1)/2$, όπου η πρώτη παράγωγος $f(x_4)$ είναι αρνητική και το διάστημα αναζήτησης μικρότερο από μια προκαθορισμένη τιμή

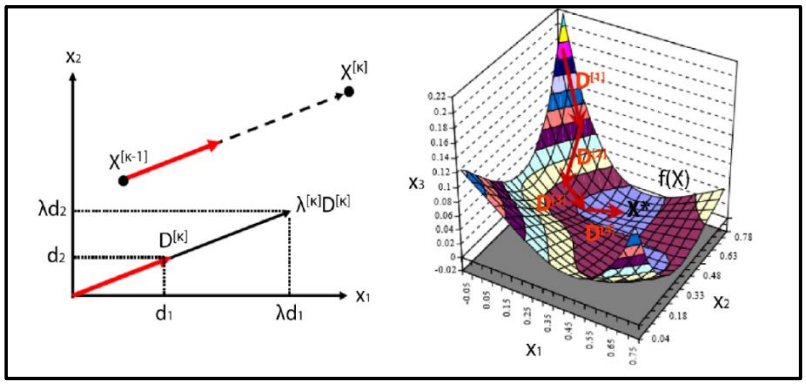
- ***Γραμμική αναζήτηση:*** Στόχος της διαδικασίας είναι να ελαχιστοποιηθεί η συνάρτηση *f* κατά μήκος της ευθείας που περνά από σημείο $x_1$ και ακολουθεί την κατεύθυνση *s*, δηλαδή να βρεθεί $\lambda > 0$ ώστε $min[f(x_1+\lambda s)]$. *Αναπτύσσοντας κατά* Taylor και διατηρώντας όρους μέχρις 1ης τάξης προκύπτει η σχέση $f(x1+\lambda s) = f(x1) + \lambda s f'(x1)$, που υποδεικνύει έναν επαναληπτικό αλγόριθμο: Δοθέντος τρέχοντος

σημείου $x_1$, υπολογίζεται μια κατεύθυνση $s$ προς όπου η $f$ φθίνει ($\lambda sf'(x_1) \leq 0$), ελαχιστοποιείται η $f(x_1+\lambda s)$ ως προς $\lambda$, εντοπίζεται η θέση ελαχίστου $\lambda^*$ και επιλέγεται $x_2 = x_1 + \lambda^* s$ για το επόμενο σημείο. Ο εντοπισμός του $\lambda^*$ με τον αλγόριθμο αυτό είναι επακριβής, αλλά και υπολογιστικά δαπανηρός. Η διαδικασία μπορεί να επιταχυνθεί αν αποδεχθούμε η ακρίβεια να είναι προσεγγιστική, αρκεί σε κάθε επανάληψη η τιμή της $f$ να μειώνεται σημαντικά και, συγχρόνως, το βήμα αναζήτησης να είναι ικανοποιητικά μικρού μεγέθους για ταχεία σύγκλιση. Αυτό αποτυπώνεται στη μαθηματική συνθήκη του να υπάρχουν $\alpha, \beta \in R$ με $0<\alpha<\beta<1$ ώστε να ισχύει $f(x_1+\lambda s) \leq f(x_1) + \alpha\lambda sf'(x_1)$ και $sf'(x_1+\lambda s) > \beta sf'(x_1)$.

### 5.9.2 Τεχνικές με χρήση παραγώγων[6]

Οι αλγόριθμοι αυτοί ξεκινούν από μια αρχική εκτίμηση της λύσης και, εκτελώντας μια διαδοχή βημάτων, διορθώνουν σταδιακά την τιμή της συνάρτησης σε κάθε βήμα. Η εκάστοτε διόρθωση ισούται με το γινόμενο του μήκους βήματος που έχουμε επιλέξει επί το διάνυσμα κατεύθυνσης προς την οποία γίνεται η μετάβαση. Όσο περισσότερο πλησιάζουμε προς το ελάχιστο, οι δύο αυτές παράμετροι μπορούν να μεταβάλλονται μέχρι να μας οδηγήσουν σε μια ικανοποιητική προσέγγιση.

Γενική διατύπωση αλγορίθμου επαναληπτικής καθόδου: Έστω $f(x)$: $R^n \rightarrow R$ συνάρτηση με τοπικό ελάχιστο στο $X^*$. Ένας αλγόριθμος επαναληπτικής καθόδου, ξεκινώντας από ένα σημείο $X^{[0]}$ σε κατάλληλη απόσταση από το $X^*$, ελαχιστοποιεί την $f$ εφαρμόζοντας κανόνες μετάβασης της μορφής $X^{[\kappa+1]} = X^{[\kappa]} + \lambda^{[\kappa]}D^{[\kappa]}$,όπου $\lambda > 0$ παράμετρος κλίμακας και D διεύθυνση στο $R^n$ τέτοιες ώστε $f(X^{[\kappa+1]}) < f(X^{[\kappa]})$ σε κάθε μετατόπιση κ.




*Σχήμα 48. Διαδικασία επιλογής βήματος διόρθωσης (2D & 3D)*[6]

Για να κατανοήσουμε τη λειτουργία του αλγορίθμου αρκεί να αναπτύξουμε την $f$ σε σειρά Taylor γύρω από τη θέση πριν την εκτέλεση ενός βήματος:

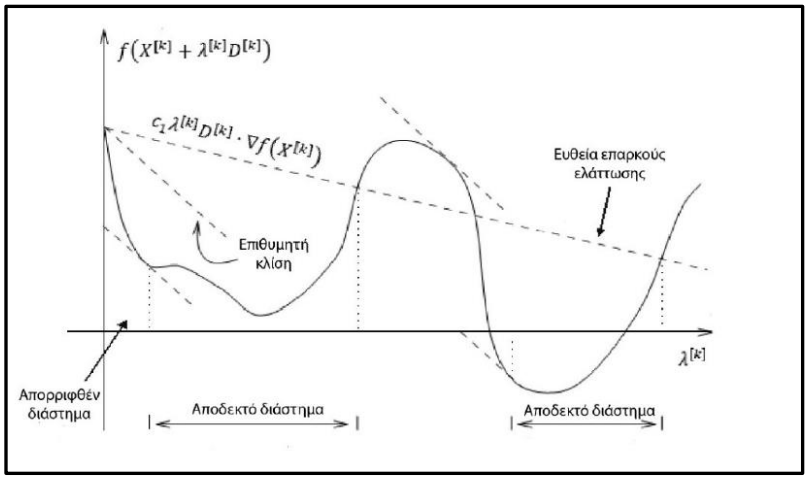
$$f(X[k+1])-f(X[k])=\lambda[k]D[k]\cdot\nabla f(X[k])+O(\lambda[k])$$

Για τιμές του $\lambda[\kappa]$ σχετικά μικρές οι όροι ανώτερης τάξης είναι αμελητέοι, οπότε το πρόσημο της διαφοράς των τιμών της $f$ ανάμεσα στα διαδοχικά βήματα καθορίζεται ουσιαστικά μόνο από τον όρο $D\cdot\nabla f$. Συγκεκριμένα, η διεύθυνση που θα πρέπει να ακολουθήσουμε για να προσεγγίσουμε το $X^*$ είναι τέτοια ώστε $D\cdot\nabla f<0$, δηλαδή η γωνία μεταξύ της επιλεγμένης κατεύθυνσης και της κλίσης της συνάρτησης να είναι αμβλεία. Αυτό

σημαίνει πως αν η f είναι φθίνουσα θα πρέπει να κατευθυνθούμε προς την κλίση της, ενώ αν είναι αύξουσα οφείλουμε να κινηθούμε αντίθετα.

Φαίνεται ακόμη ότι η επιλογή του μήκους βήματος $\lambda$ είναι σημαντική για την επιτυχία της μεθόδου. Όταν το βήμα επιλεγεί πολύ μικρό, η μείωση της f θα είναι ασήμαντη και ο αλγόριθμος θα πλησιάζει την ελάχιστη τιμή πολύ αργά, ενώ σε περίπτωση που το βήμα είναι πολύ μεγάλο υπάρχει το ενδεχόμενο να επιφέρει την ανάγκη αλλαγής της κατεύθυνσης. Γίνεται αντιληπτό ότι η τιμή του βήματος που θα επιλεγεί θα πρέπει να συνδυάζει ικανοποιητικά την ακρίβεια, την ομαλότητα και την ταχύτητα σύγκλισης της αναζήτησης. Σε αυτό το πλαίσιο, για τον εντοπισμό του κατάλληλου βήματος έχει θεσπιστεί ένα σύνολο από συνθήκες, μια τουλάχιστον από τις οποίες θα πρέπει να ικανοποιείται. Τέτοιες είναι:

- Η συνθήκη Armijo που εξασφαλίζει την επαρκή ελάττωση του βήματος
- Η συνθήκη Goldstein που ελέγχει την κλίση της συνάρτησης στο βήμα k αν είναι μικρότερη από την αντίστοιχη κλίση στο αρχικό σημείο




*Σχήμα 49. Επιλογή διαστήματος αναζήτησης βάση της συνθήκη Goldstein*[6]

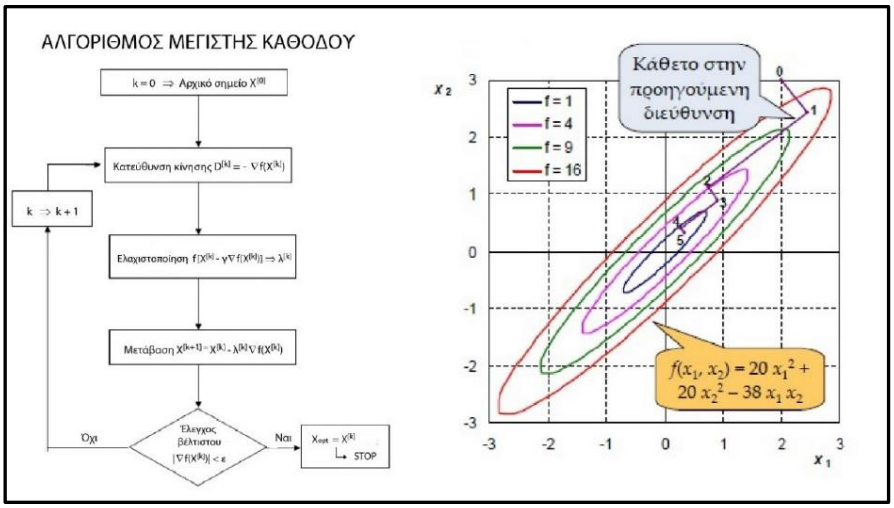
Η συνθήκη Goldstein εκφράζει ότι η κλίση της συνάρτησης στο βήμα k είναι μικρότερη από την αντίστοιχη κλίση στο αρχικό σημείο. Συνεπώς, έχουμε τη δυνατότητα να φέρουμε εφαπτόμενες στα κρίσιμα σημεία της συνάρτησης και, μέσω της συνθήκης, να απορρίψουμε διαστήματα που περιέχονται ανάμεσα σε κάθε δύο κλίσεις. Η λειτουργία της αποτυπώνεται στο σχήμα 49.

Ανάλογα με το συγκεκριμένο τρόπο της επιλογής των $\lambda$ και $D$ οδηγούμαστε σε μια πληθώρα στρατηγικών αναζήτησης του ελαχίστου, κάθε μια από τις οποίες εμφανίζει και διαφορετικές ιδιότητες. Επί της ουσίας, κάθε αλγόριθμος στηρίζεται, εκτός από το βήμα και τη διεύθυνση, σε διαφορετικό τρόπο καθορισμού και των αρχικών τιμών και του κριτηρίου τερματισμού, το οποίο μπορεί να συνδέεται με τη μεταβολή κάτω από ένα όριο $\varepsilon$ (από βήμα σε βήμα) του X, της f, της $\nabla f$ ή/και του $\lambda$.

Ακολουθεί η περιγραφή των κυριότερων μεθόδων επαναληπτικής καθόδου, δίνοντας έμφαση σε χαρακτηριστικά που αναφέραμε παραπάνω. Όπως θα διαπιστώσουμε, οι μέθοδοι της κατηγορίας αυτής διαιρούνται σε δύο ομάδες: σε εκείνες τις μεθόδους όπου η επιλογή της κατεύθυνσης στηρίζεται σε διανύσματα παράλληλα της κλίσης της αντικειμενικής συνάρτησης, και σε εκείνες που η επιλογή ταυτίζεται με ένα γενικότερο υποπρόβλημα εντοπισμού της βέλτιστης κατεύθυνσης.

### 5.9.2.1 Επιλογή κατεύθυνσης με διάνυσμα

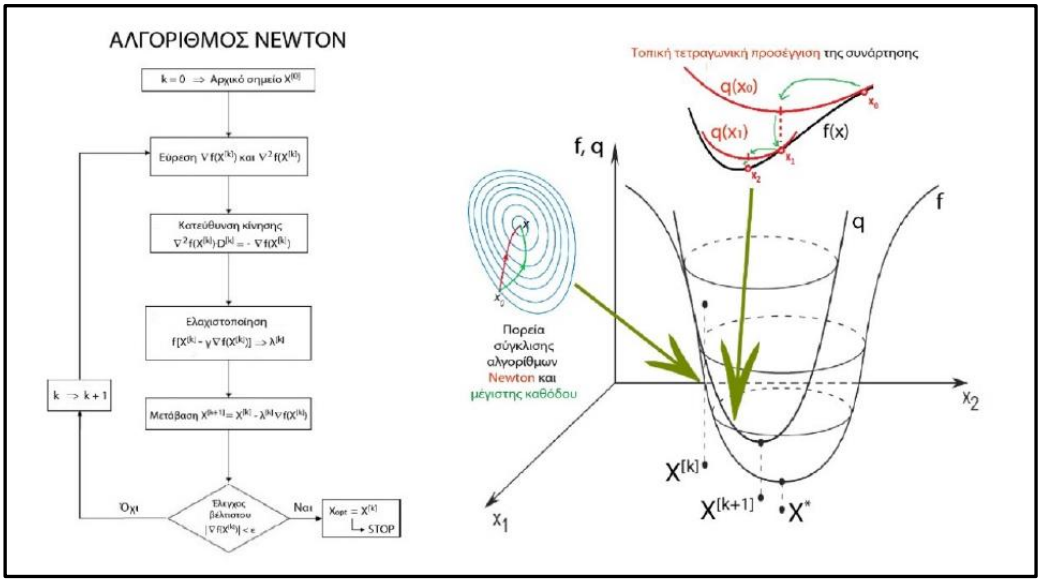
- ***Μέθοδος μέγιστης καθόδου:*** Πρόκειται για την απλούστερη από τις συγκεκριμένες μεθόδους. Εδώ επιλέγουμε $D=-\nabla f$, δηλαδή η διεύθυνση αναζήτησης να είναι αντίθετη στο διάνυσμα κλίσης της $f$. Οι κανόνες μετάβασης έχουν την απλή μορφή $X[k+1]=X[k]-\lambda[k]\nabla f(X[k])$, το οποίο έχει ως αποτέλεσμα η μετακίνηση να είναι πάντα κάθετη στη διεύθυνση της τρέχουσας λύσης και κάθε επόμενο σημείο να είναι η θέση ελαχίστου της συνάρτησης κατά μήκος της διεύθυνσης που ορίζει η κλίση της. Το μέτρο $\lambda[k]$, το οποίο συμβολίζει την ταχύτητα σύγκλισης της μεθόδου, υπολογίζεται στη βάση του να ελαχιστοποιείται η $f$ στο ευθύγραμμο τμήμα $G(\gamma)=X[k]-\gamma\nabla f(X[k])$. Συνεπώς, σε κάθε επαναληπτικό βήμα διαμορφώνεται ένα μονοδιάστατο πρόβλημα ελαχιστοποίησης, το οποίο και επιλύεται αριθμητικά. Ως αντίβαρο στην απλότητά της, το μειονέκτημα της μεθόδου είναι η αργή της σύγκλιση. Αυτό συμβαίνει διότι η αναζήτηση εκτελείται σε μικρά εγκάρσια βήματα, σχηματίζοντας μια τεθλασμένη γραμμή κατά πλάτος των ισοϋψών καμπυλών της $f$. Στο σχήμα 50 δίνεται το διάγραμμα ροής του αλγορίθμου μέγιστης καθόδου και απεικονίζεται σχηματικά ένα παράδειγμα εφαρμογής του.




*Σχήμα 50. Μέθοδος μέγιστης καθόδου*[6]

- ***Μέθοδος Newton:*** Στη μέθοδο αυτή επιλέγουμε $D=-[\nabla 2f]-1\cdot\nabla f$, με τη βασική προϋπόθεση ο Εσσιανός πίνακας της $f$ να είναι θετικά ορισμένος. Μια αναβάθμιση σε σχέση με τη μέγιστη κάθοδο είναι ότι χρησιμοποιούνται και δεύτερες παράγωγοι (εκτός των πρώτων), με σκοπό να συμπεριληφθούν δεδομένα της καμπυλότητας της συνάρτησης στην επιλογή της κατεύθυνσης. Το βήμα υπολογίζεται και πάλι από την ελαχιστοποίηση της $f$ ως προς $\gamma$ κατά μήκος του τμήματος $X[k]-\gamma\nabla f(X[k])$, ενώ οι κανόνες μετάβασης έχουν τη μορφή $X[k+1]=X[k]-\lambda[k][\nabla 2f(X[k])]-1\cdot\nabla f(X[k])$. Αυτός ο αναδρομικός τύπος αντιστοιχεί σε ελαχιστοποίηση της τοπικής προσέγγισης 2ης τάξης *q(X) της f,* όπως αυτή υπολογίζεται με ανάπτυξη σε σειρά Taylor γύρω από το X[*k*]. Αντί χρήσης του τύπου, οι διαδοχικοί όροι *X[k+1]* βρίσκονται (δεδομένου *X[k]*) και λύνοντας το ισοδύναμο γραμμικό σύστημα εξισώσεων που προκύπτει πολλαπλασιάζοντας κατά μέλη με τον Εσσιανό πίνακα:

$A[k]=\nabla 2f(X[k])$, $B[k]=\nabla 2f(X[k])\cdot X[k]-\lambda[k]\cdot\nabla f(X[k]) \Rightarrow A[k]\cdot X[k+1]=B[k]$.
Αν δεν ισχύει η συνθήκη $\nabla 2f>0$, τότε μπορούμε να καταφύγουμε σε τροποποιήσεις της μεθόδου Newton, όπως για παράδειγμα η μέθοδος Levenberg - Marquardt στην οποία ο $\nabla 2f$ αντικαθίσταται από πίνακα της μορφής $\nabla 2f + \mu I$, με το συντελεστή $\mu$ να επιλέγεται έτσι ώστε ο νέος πίνακας να είναι θετικά ορισμένος. Ακολουθεί το διάγραμμα ροής του αλγορίθμου Newton και ένα σχηματικό υπόδειγμα της εφαρμογής του.




*Σχήμα 51. Αλγόριθμος Newton*[6]

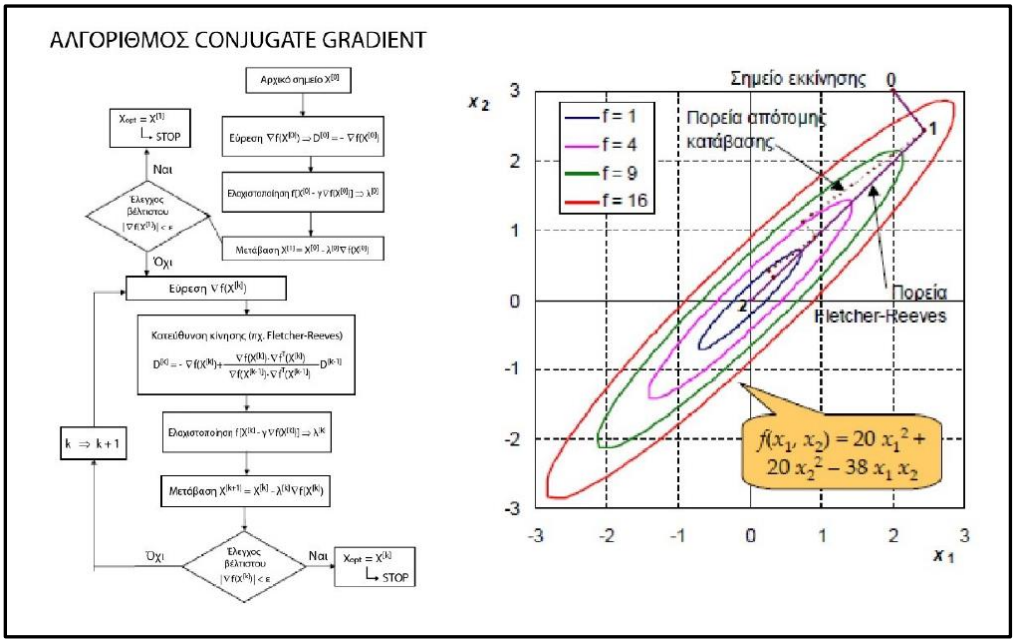
Σε σχέση με τις μεθόδους της πρώτης ομάδας, οι συνθήκες επιλογής κατεύθυνσης και βήματος που είδαμε παραπάνω μπορούν να αξιοποιηθούν και ως κριτήρια καλής λειτουργίας για το σχεδιασμό επαναληπτικών μεθόδων. Για παράδειγμα, υπάρχουν περιπτώσεις όπου ο αλγόριθμος Newton αποκλίνει παρότι οι τιμές των παραμέτρων είναι πολύ κοντά σε εκείνες για τις οποίες λειτουργεί επιτυχώς. Παρατηρούμε επίσης ότι η μέθοδος Levenberg - Marquardt για μεγάλο μ λειτουργεί σαν τη μέγιστη κάθοδο. Κατά συνέπεια, επιλέγοντας τον αλγόριθμο με ενδιάμεση τιμή του μ, συνδυάζονται οι δύο μέθοδοι και μπορούν να αποφευχθούν ορισμένες από τις δυσλειτουργίες.

### 5.9.2.2 Επιλογή κατεύθυνσης χωρίς διάνυσμα[6]

Οι μέθοδοι της δεύτερης ομάδας προσφέρουν ταχύτερη σύγκλιση σε σχέση με αυτές της πρώτης, ειδικά στις περιπτώσεις όπου δεν είναι θετικά ορισμένος ο Εσσιανός πίνακας. Βασίζονται στη βελτιστοποίηση τετραγωνικών συναρτήσεων επειδή, κάθε ομαλή συνάρτηση περιγράφεται με ικανοποιητική ακρίβεια από ανάπτυγμα Taylor 2ης τάξης. Αναφερόμαστε αποκλειστικά σε δύο τεχνικές, οι οποίες συναντώνται σε παραλλαγές αναλόγως της επιλογής συγκεκριμένων παραμέτρων.

- Μέθοδος συζυγών κλίσεων: Η συγκεκριμένη τεχνική υποστηρίζει ότι η αναζήτηση επιταχύνεται σημαντικά όταν η κατεύθυνση επιλέγεται ως γραμμικός συνδυασμός του διανύσματος κλίσης στο τρέχον και το προηγούμενο σημείο. Για αυτό το λόγο, η

αντικειμενική συνάρτηση προσεγγίζεται μέσω του πολυωνυμικού αναπτύγματος 2ης τάξης $f(X)=12⁄XTQX+BTX$, με $X\epsilon Rn$ και Q έναν θετικά ορισμένο πίνακα. Καθώς η συνθήκη $\nabla f=0$ καταλήγει στην QX = B, η ελαχιστοποίηση της f συνδέεται με την επίλυση ενός γραμμικού συστήματος. Η διαδικασία εύρεσης λύσης είναι δυνατό να αναπαρασταθεί στη γνωστή μορφή $X[k+1]=X[k]+\lambda[k]D[k]$, όπου τα διανύσματα D ικανοποιούν την ιδιότητα D[i]QD[j] = 0 για κάθε i ≠ j και ονομάζονται Q-συζυγή. Σε αυτό το πλαίσιο, οι βέλτιστες επιλογές για το σχεδιασμό του αλγορίθμου επανάληψης όσον αφορά την ταχύτητα σύγκλισης είναι η λ[k] = min[X[k] − γ∇ f(X[k])] για το μήκος βήματος και, όσον αφορά την κατεύθυνση, η $D[k+1]=-\nabla f(X[k+1])+\beta[k]D[k]$, με συντελεστές β[k] εκείνους για τους οποίους τα D[k] είναι Q-συζυγή $\beta[k]=-\nabla fT(X[k+1])\cdot Q\cdot D[k][D[k]]T\cdot Q\cdot D[k]$, και αρχική συνθήκη D[0] την κατεύθυνση μέγιστης καθόδου. Ακολουθεί το διάγραμμα ροής του αλγορίθμου και παράδειγμα εφαρμογής του.




*Σχήμα 52. Μέθοδος συζυγών κλίσεων*[6]

- Μέθοδοι σχεδόν Newton: Οι μέθοδοι σχεδόν Newton συνδυάζουν τα πλεονεκτήματα της μεθόδου Newton με την αποφυγή υπολογισμού παραγώγων 2ης τάξης. Οι κανόνες μετάβασης είναι ίδιοι με παραπάνω, αλλά εφαρμόζουν διανύσματα κατεύθυνσης της μορφής $D=-\Delta\cdot\nabla f$, όπου ο Δ είναι θετικά ορισμένος και εν γένει μεταβάλλεται σε κάθε επανάληψη ώστε η κατεύθυνση να προσεγγίζει αυτή της μεθόδου Newton.

### 5.9.3 Τεχνικές χωρίς τη χρήση παραγώγων

Στην πράξη, σε αρκετά προβλήματα ελαχιστοποίησης η εφαρμογή των μεθόδων που περιγράψαμε στην προηγούμενη ενότητα καθίσταται υπολογιστικά ασύμφορη. Μια τέτοια περίπτωση έχουμε όταν δεν είναι γνωστή η αναλυτική έκφραση των μερικών παραγώγων της αναλυτικής συνάρτησης είτε παρέχεται υπό τη μορφή πολύπλοκων εκφράσεων, και άλλη μια έχουμε όταν η αριθμητική επίλυση του μονοδιάστατου προβλήματος βελτιστοποίησης σε κάθε επαναληπτικό βήμα είναι χρονοβόρα διότι η αντικειμενική συνάρτηση δεν έχει

παραβολική μορφή. Στις περιπτώσεις αυτές είναι προτιμητέα η χρήση αλγορίθμων που δεν εμπλέκουν παραγώγους.

Μια προφανής λύση είναι να εφαρμόσουμε και πάλι τις μεθόδους της προηγούμενης ενότητας, αλλά χρησιμοποιώντας αριθμητικές προσεγγίσεις για τον προσδιορισμό των παραγώγων 1ης και 2ης τάξης όπου αυτές απαιτούνται. Οι μερικές παράγωγοι 1ης τάξης μπορούν να προσεγγιστούν είτε μέσω εμπρόσθιας/οπίσθιας διαφόρισης είτε μέσω κεντρικής διαφόρισης που παρέχει καλύτερη ακρίβεια στις περισσότερες περιπτώσεις, αλλά απαιτεί έναν αρκετά μεγαλύτερο αριθμό πράξεων.

# 6. Συμπεράσματα

Το ζήτημα της βελτιστοποίησης συναντάται πολύ συχνά σε προβλήματα που αντιμετωπίζουμε στην καθημερινότητά μας. Στην πλειονότητά τους τα προβλήματα αυτά είναι πολύπλοκα, συνδυαστικά προβλήματα και μάλιστα στην γενική μορφή τους είναι NP-hard. Η πολυπλοκότητα αυτή καθιστά πολύ δύσκολη έως και αδύνατη την αποδοτική και βέλτιστη επίλυσή τους. Το γεγονός αυτό ανάγει την επίλυση τέτοιων προβλημάτων σε μια πολύ δύσκολη, χρονοβόρα, επίπονη και συχνά ακριβή διαδικασία.

Σε κάποιες περιπτώσεις, η επίλυση ενός προβλήματος ουσιαστικά επικεντρώνεται, απλώς και μόνο στην εύρεση ενός αλγορίθμου που να ικανοποιεί όλους τους περιορισμούς. Σε αυτές τις περιπτώσεις το πρόβλημα διατυπώνεται ως *πρόβλημα αναζήτησης* (search problem).

Σε κάποιες άλλες όμως περιπτώσεις το πρόβλημα διατυπώνεται ως *πρόβλημα βελτιστοποίησης* (optimization problem). Η απαίτηση που διατυπώνεται στο πρόβλημα βελτιστοποίησης είναι η εξεύρεση ενός αλγορίθμου που να ικανοποιεί όλους τους «σκληρούς» περιορισμούς (hard constraints) και να ελαχιστοποιεί (ή να μεγιστοποιεί) μια αντικειμενική συνάρτηση που εμπεριέχει τους «μαλακούς» περιορισμούς (soft constraints).

Αναφορικά με τις δύο προαναφερθείσες διατυπώσεις του προβλήματος έχει εκφραστεί και μια εναλλακτική προσέγγιση. Σύμφωνα με τους υποστηρικτές αυτής της προσέγγισης, θεωρείται ότι η διατύπωση ενός προβλήματος βελτιστοποίησης στην πραγματικότητα, δεν είναι τίποτα άλλο, παρά η εφαρμογή τεχνικών βελτιστοποίησης σε ένα πρόβλημα αναζήτησης.

Επίσης, υποστηρίζεται ότι η αντικειμενική συνάρτηση που περιγράφει το πρόβλημα, στοχεύει στην μείωση του αριθμού των μη εφικτών λύσεων στο πρόβλημα, οπότε πρακτικά, αναπαριστά την απόσταση μιας λύσης από την εφικτή λύση του πραγματικού προβλήματος (distance to feasibility).

Και στις δύο περιπτώσεις (πρόβλημα αναζήτησης ή βελτιστοποίησης) ορίζουμε το θεμελιώδες πρόβλημα, που δεν είναι άλλο από το πρόβλημα της επιλογής της λύσης. Στα προβλήματα αναζήτησης, η επιλογή γίνεται με βάση το αν υπάρχει μία λύση, ενώ στα προβλήματα βελτιστοποίησης, το πρόβλημα της επιλογής έγκειται στο αν υπάρχει μία λύση που να αποδίδει δεδομένη τιμή στην αντικειμενική συνάρτηση.

Ήδη έχουμε αναφερθεί στην πολυπλοκότητα των προβλημάτων. Στην ουσία όμως, αναφερόμαστε στην πολυπλοκότητα του θεμελιώδους προβλήματος της επιλογής. Το θεμελιώδες πρόβλημα ανήκει και αυτό στην κατηγορία των NP-hard προβλημάτων σχεδόν για όλες τις παραλλαγές του προβλήματος. Για τον λόγο αυτό, μία ακριβής λύση επιτυγχάνεται μόνο σε μικρού μεγέθους προβλήματα. Έτσι είναι επόμενο ότι μόνο οι ευρετικές μέθοδοι (heuristic methods) (Pearl, 1984), είναι εφικτές και ενδείκνυται να εφαρμοστούν στην διαδικασία επίλυσης του προβλήματος, αν και στην πράξη, ούτε οι ευρετικές μέθοδοι εγγυώνται την εξεύρεση της βέλτιστης λύσης.

Η έννοια της βελτιστοποίησης θέτει ως στόχο του προβλήματος να αναζητηθεί η βέλτιστη λύση ανάμεσα σε όλες τις πιθανές-εφικτές λύσεις. Με άλλα λόγια να βρεθεί μια λύση στην εφικτή περιοχή (feasible region), η οποία να δίνει την ελάχιστη ή την μέγιστη τιμή στην αντικειμενική συνάρτηση.

Μερικά από τα κλασσικά προβλήματα βελτιστοποίησης που ταλανίζουν ακόμα και σήμερα τους ερευνητές είναι: το πρόβλημα του περιοδεύοντος πωλητή (traveling salesman problem), το πρόβλημα πακετοποίησης των κουτιών (bin packing problem), το πρόβλημα του σακιδίου (knapsack problem), και το πρόβλημα της δρομολόγησης των οχημάτων (vehicle routing problem).

Οι μέθοδοι που προτείνονται γενικότερα για τα προβλήματα βελτιστοποίησης ποικίλλουν ανάλογα με τις ιδιαιτερότητες, τα χαρακτηριστικά και τους περιορισμούς του κάθε προβλήματος.

# Βιβλιογραφία


[1] Κ. Ε. Παρσόπουλος, *"Αλγόριθμοι υπολογιστικής νοημοσήνης για αριθμητική βελτιστοποίηση."* Πάτρα: Πανεπιστήμιο Πατρών. Σχολή Θετικών Επιστημών. Τμήμα Μαθηματικών, 2004.

[2] Κ. Τσίχλας, Ι. Μανωλόπουλος, and Αναστάσιος Γούναρης, *"Σχεδίαση και Ανάλυση Αλγορίθμων."* Αθήνα: Εθνικό Μετσόβειο Πολυτεχνείο, 2015.

[3] Μ. Μ. Ι. Βογιατζής, Ν. Ιωαννίδης, Χ. Κοίλιας, Γ. Μελετίου, *"Εισαγωγή στην Αλγοριθμική Υλοποιήσεις αλγορίθμων σε C."* Αθήνα: Εκδόσεις Νέων Τεχνολογιών, 2010.

[4] Μαντζώλας Γιώργος, "Το κόσκινο του Ερατοσθένη." [Online]. Available: http://users.sch.gr/geoman22/mathP/Eratosthenis.htm. [Accessed: 30-Sep-2018].

[5] Α. Λεωνίδας, Γ. Β. Β. Δημήτριος, Π. Γεώργιος, and Σ. Ανδρέας, *Μαθηματικά,Β΄ Τάξη Γενικού Λυκείου*. Αθήνα: ΙΤΥΕ "ΔΙΟΦΑΝΤΟΣ," 1998.

[6] Α. Ντούνης and Χ. Τσιρώνης, *"Εισαγωγή στη Βελτιστοποίηση Συστημάτων."* Πειραιάς: Ανώτατο Εκπαιδευτικό Ίδρυμα Πειραιά Τ.Τ. Τμήμα Μηχανικών Αυτοματισμού Τ.Ε., 2017.

[7] Κατάκης Ιωάννης, "Προγραμματισμός Μεθόδων Επίλυσης Προβλημάτων." [Online]. Available: http://www.cs.ucy.ac.cy/courses/EPL032/epl032-2/lectures/lecture13.pdf. [Accessed: 30-Jan-2019].

[8] Α. Ευθυμιόπουλος, "Et Ego in Arcadia Sum," 2013. [Online]. Available: https://et-in-arcadia-ego8.webnode.gr/κυβερνητικη/καθολικη-μηχανη-turing/. [Accessed: 31-Oct-2018].

[9] Γ. Χ. Μεγαλοκονόμος, "Το ανάστροφο πρόβλημα διάκεντρων με περιορισμούς χωρητικότητας," Πανεπιστήμιο Πατρών, 1997.

[10] Δρ.Σάββας Ηλίας, *"Αλγοριθμοι & πολυπλοκότητα."* Λάρισα: Τεχνολογικό Ιδρυμα Λάρισας,Τμήμα Τεχνολογίας Πληροφορικής & Τηλεπικοινωνιών, 2005.

[11] Wikipedia, "Quicksort." [Online]. Available: https://en.wikipedia.org/wiki/Quicksort. [Accessed: 15-Nov-2018].

[12] Κυριακόπουλος Χρήστος, *"Σύστημα Βέλτιστης Ανάπτυξης Υπηρεσιών σε πολύπλοκα Δυναμικά συστήματα."* Πειραιάς: Πανεπιστήμιο Πειραιά, Τμήμα Ψηφιακών Συστημάτων, 2010.

[13] Γ. Χαλκιαδάκης, *"Τεχνητή Νοημοσύνη."* Χανιά: Πολυτεχνείο Κρήτης, Τμημα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών, 2014.

[14] Anany Levitin, *"The Design & Analysis of Algorithms,"* 2nd ed. Villanova University: Addison-Wesley, 2007.

[15] C. Coello, G. Lamont, and D. Van Veldhuizen, *"Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition,"* 2nd ed. Mexico: Springer Science and Business Media, LLC All, 2007.

[16] A. Marino, "'Analysis and Enumeration,'" vol. 6, Springer Science and Business Media, LLC All, 2015, pp. 13–36.

[17] C. Charras and T. Lecroq, “Brute force algorithm.” [Online]. Available: http://www-igm.univ-mlv.fr/~lecroq/string/node3.html. [Accessed: 30-Nov-2018].

[18] Tutorials and Point, “Data Structures Divide and Conquer,” 2016. [Online]. Available: https://slideplayer.com/slide/4497107/. [Accessed: 02-Feb-2019].

[19] Zhu and D. Yingwu, “Divide-and-Conquer.” [Online]. Available: https://slideplayer.com/slide/4497107/. [Accessed: 15-Dec-2018].

[20] Δ. Αθανασοπούλου, *Knapsack Problem*. Πάτρα: Πανεπιστήμιο Πατρών, 2017.

[21] Νικόλαος Σ. Στυλιανού, *Προσεγγίζοντας Το Πρόβλημα Του Πλανόδιου Πωλητή*. Πάτρα: Πανεπιστημίο Πατρών, 2013.

[22] Χ. Γκόγκος, *Αλγόριθμοι συνδυαστικής βελτιστοποιησης με έμφαση στις μεταευρετικές τεχνικές*. Πάτρα: Πανεπιστήμιο Πατρών, 2009.

[23] Ι. Παπαδημητρούλα, *Το πρόβλημα της Ικανοποιησιμότητας. Ευρετικές και μεθευρετικές μέθοδοι επίλυσης.* Θεσσαλονίκη: Α.Π.Θ., 2013.

[24] “Introduction to Greedy Algorithms.” [Online]. Available: https://developerinsider.co/introduction-to-greedy-algorithms/. [Accessed: 18-Nov-2018].

[25] D. Ponomarev, “Optimization problems,” 2002. [Online]. Available: https://slideplayer.com/slide/4527696/. [Accessed: 18-Nov-2018].

[26] Vineet Choudhary, “Introduction to Greedy Algorithms.” [Online]. Available: https://developerinsider.co/introduction-to-greedy-algorithms/. [Accessed: 23-Jan-2019].

[27] M. Shehab and A. T. Khader, “A hybrid method based on Cuckoo search algorithm for global optimization problems,” *J. Inf. Commun. Technol.*, vol. 17(3), no. July, pp. 469–491, 2018.

[28] Κ. Γεωργουλη, *Τεχνητή Νοημοσύνη, Μία εισαγωγική προσέγγιση.* Αθήνα: Σύνδεσμος Ελληνικών Ακαδημαικών Βιβλιοθηκών, 2015.

[29] Baeldung, “Example of Hill Climbing Algorithm.” [Online]. Available: https://www.baeldung.com/java-hill-climbing-algorithm. [Accessed: 16-Dec-2018].

[30] HackerEarth, “Depth First Search Tutorials & Notes Algorithms.” [Online]. Available: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/. [Accessed: 16-Dec-2018].

[31] Βικιπαίδεια, “Αναζήτηση Κατά Βάθος.” [Online]. Available: https://el.wikipedia.org/wiki/Αναζήτηση_Κατά_Βάθος. [Accessed: 19-Dec-2018].

[32] HackerEarth, “Breadth First Search Tutorials & Notes Algorithms.” [Online]. Available: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/. [Accessed: 16-Dec-2018].

[33] G. Zaharija, “Cognitive Agents and Learning Problems,” *I.J. Intell. Syst. Appl.*, vol. 3, no. March, pp. 1–7, 2017.

[34] Π. Λαμπάτος, “Προσεγγιστικοί Αλγόριθμοι του Προβλήματος του Περιοδεύοντος Πωλητή,” Πανεπιστημιο Πειραώς, 2014.

[35] Κ. Μ., “Διακλάδωση και οριοθέτηση,” 1982. [Online]. Available: http://cgi.di.uoa.gr/~ys02/siteAI2009/lectures/branch-and-bound2spp.pdf. [Accessed:

03-Feb-2019].

[36] Κ. Μικροπούλου, "Επίλυση προβλημάτων βέλτιστης ανάθεσης με το λογισμικό ανοικτού κώδικα Python," ΤΕΙ Σερρών, Σέρρες, 2018.

[37] J. Brownlee, "Random Search - Clever Algorithms Nature-Inspired Programming Recipes," *2015*. [Online]. Available: http://www.cleveralgorithms.com/nature-inspired/index.html.

[38] Wikipedia, "Random search." [Online]. Available: https://en.wikipedia.org/wiki/Random_search. [Accessed: 24-Jan-2018].

[39] K.-L. Du and M. N. S. Swamy, *Search and optimatization By Metaheuristics Technques and Algorithms inspired by Nature*. Springer Science and Business Media, LLC All, 2016.

[40] "Monte Carlo Simulation and Methods Introduction - GoldSim." [Online]. Available: https://www.goldsim.com/web/introduction/montecarlo/. [Accessed: 24-Jan-2019].

[41] S. Paltani, *Monte Carlo Methods Outline*. Geneva: University of Geneva, 2011.

[42] "Overview of Tabu Search," 2012. [Online]. Available: https://researchweb.iiit.ac.in/~kondetimanikanta.purushotham/IE789F_tabu. [Accessed: 16-Jan-2019].

[43] Χ. Φ. Χατζηνικολάου, "Βελτιστοποίηση μέσω αποικιών μυρμηγκιών," Καποδιστριακό Πανεπιστήμιο Αθηνών, 2016.

[44] Ατσαλακης Γεώργιος, "Σύγκριση προβλέψεων παραδοσιακών και νέων τεχνολογιών πηγών ενέργειας," Πολυτεχνειο Κρητης, 2013.

[45] Ισίδωρου Α. Πέτικα, "Υβριδικοί εξελικτικοί αλγόριθμοι βελτιστοποίησης και εφαρμογές σε προβλήματα συνδυαστικής βελτιστοποίησης," Πανεπιστήμιο Πειραιώς, 2012.

[46] Λάζαρος Κήττας, "Ανώτερες τεχνικές ανάλυσης και σχεδίασης αλγορίθμων," Πανεπιστήμιο Πατρών, 2015.

[47] A. Jain, H. V Madhyastha, and C. M. Prince, "An Analysis and Comparison of Satisfiability Solving Techniques," Seattle, 2004.

[48] M. Hristakeva and D. Shrestha, "Different Approaches to Solve the 0 / 1 Knapsack Problem Simpson College The Knapsack Problem ( KP )," Indianola, 2004.

[49] A. Guler, M. E. Berberler, and U. G. Nuriyev, "A New Genetic Algorithm for the 0-1 Knapsack Problem," Izmir, 2016.

[50] R. W. Haessler and P. E. Sweeney, "Cutting stock problems and solution procedures," *Eur. J. Oper. Res. 54*, vol. 54, pp. 141–150, 1991.

[51] A. Fischer, W. Robert, and S. Dempe, *Problems , Models and Algorithms in One- and Two-Dimensional Cutting*. Dresden,: Technischen Universit¨at Dresden, 2004.

[52] A. Avdzhieva, T. Balabanov, G. Evtimov, D. Kirova, H. Kostadinov, and T. Tsachev, "Optimal Cutting Problem," 2004.

[53] Κ. Δαβιλάς, "Ανάλυση του Bín Packing Problem," Καποδιαστριακό Πανεπιστήμιο Αθηνών, ΑΘΗΝΑ, 2016.

[54] R. E. Korf, "A New Algorithm for Optimal Bin Packing," in *AAAI-02 Proceedings*, 2002, pp. 731–736.

[55] S. Spasovski and A. M. Bogdanova, “Optimization of the Polynomial Greedy Solution for the Set Covering Problem,” in *The 10th Conference for Informatics and Information Technology (CIIT 2013) Optimization*, 2013, p. 3.

[56] Σ. Νικολόπουλος, Λ. Γεωργιάδης, and Λ. Παληός, *Αλγοριθμική θεωρία γραφημάτων*. Αθήνα: Σύνδεσμος Ελληνικών Ακαδημαικών Βιβλιοθηκών, 2015.

[57] Τιμόθεος Καραγεργίου, “Πολυχρωματισμός μονοπατιών σε γραφήματα,” TEI Ηπείρου, 2006.

[58] T.-C. Du, “DSATUR,” Indiana, 2013.

[59] B. Telecom and J. Hao, “Tabu Search for Graph Coloring, T-Colorings and Set T-Colorings,” no. December 2013, 1999.

[60] H. Emami, S. Lotfi, and M. Branch, “Graph Colouring Problem based on Discrete Imperialist Competitive Algorithm,” Tabriz, 2013.

[61] S. Najafi, “Energy Procedia A New Heuristic Algorithm for Unit Commitment Problem,” Tabriz, 2012.

[62] N. Thakur and L. S. Titare, “Determination of Unit Commitment Problem Using Dynamic Programming,” vol. 3, no. 1, pp. 24–28, 2016.

[63] P. K. Singhal, “Dynamic Programming Approach for Large Scale Unit Commitment Problem,” *2011 Int. Conf. Commun. Syst. Netw. Technol.*, pp. 714–717, 2011.

[64] Κ. Γαβριήλ, “Προβλημα βελτιστοποίησης χωρίς περιορισμούς,” Πανεπιστήμιο Πατρών, Πάτρα, 2009.