ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΤΜΗΜΑ ΛΟΓΙΣΤΙΚΗΣ & ΧΡΗΜΑΤΟΟΙΚΟΝΟΜΙΚΗΣ

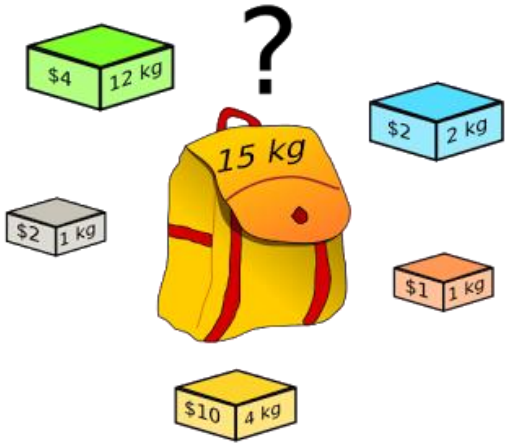
**Πρόγραμμα Μεταπτυχιακών Σπουδών στα Χρηματοοικονομικά**
**Master of Science in Finance**

**Μεταπτυχιακή Διατριβή**

# Βέλτιστη επένδυση με τη χρήση του αλγορίθμου knapsack

**Ονοματεπώνυμο φοιτητή: Χαραλαμπόπουλος Νικόλαος**

**Επιβλέποντας Καθηγητής: Νικολαΐδης Βασίλειος**




Διατριβή υποβληθείσα στο Τμήμα Λογιστικής & Χρηματοοικονομικής του ΤΕΙ Πελοποννήσου.
Η παρούσα διατριβή αποτελεί μέρος των απαιτήσεων για την απόκτηση του Μεταπτυχιακού Διπλώματος στα Χρηματοοικονομικά

Καλαμάτα, Οκτώβριος 2017

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΤΜΗΜΑ ΛΟΓΙΣΤΙΚΗΣ & ΧΡΗΜΑΤΟΟΙΚΟΝΟΜΙΚΗΣ

**Πρόγραμμα Μεταπτυχιακών Σπουδών στα Χρηματοοικονομικά**
**Master of Science in Finance**

## Τριμελής Εξεταστική Επιτροπή

**Νικολαΐδης Βασίλειος**
**Επίκουρος καθηγητής, Λογιστικής & Χρηματοοικονομικής, ΤΕΙ Πελοποννήσου**

**Γιακουμάτος Στέφανος**
**Καθηγητής, Λογιστικής & Χρηματοοικονομικής, ΤΕΙ Πελοποννήσου**

**Σταυρόγιαννης Σταύρος**
**Καθηγητής, Λογιστικής & Χρηματοοικονομικής, ΤΕΙ Πελοποννήσου**

Ο Χαραλαμπόπουλος Νικόλαος,

δηλώνω υπεύθυνα ότι:

**1)** Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

**2)** Αποδέχομαι ότι το Τμήμα Λογιστικής & Χρηματοοικονομικής μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη του Ιδρύματος, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

*Η παρούσα εργασία αφιερώνεται σε αυτούς που ψάχνουν…*

*Από τότε που κουράστηκα να ψάχνω, έμαθα να βρίσκω.*

*Φρήντριχ Νίτσε*

# ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ τους καθηγητές μου και την οικογένειά μου που με στήριξαν και με βοήθησαν στην υλοποίηση της παρούσας εργασίας.

# ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

# Περίληψη στα Ελληνικά

Στις αναπτυσσόμενες αγορές υπάρχει η ανάγκη της γρήγορης και βέλτιστης επένδυσης με σκοπό το κέρδος. Όταν ένας επενδυτής, με συγκεκριμένο αρχικό κεφάλαιο, έχει να επιλέξει μέσα από έναν αριθμό επενδυτικών ευκαιριών με σκοπό το μέγιστο κέρδος, παρουσιάζεται η ανάγκη, αυτό να γίνει εύκολα και γρήγορα, καθώς το χρηματοοικονομικό περιβάλλον αλλάζει συνεχώς. Αυτό μπορεί να επιτευχθεί με τη χρήση της τεχνολογίας και συγκεκριμένα των ηλεκτρονικών υπολογιστών.

Στην παρούσα εργασία γίνεται μια προσπάθεια εύρεσης βέλτιστης επένδυσης επιλέγοντας τις κατάλληλες επενδύσεις μέσα από έναν αριθμό επενδυτικών ευκαιριών. Αυτό επιτυγχάνεται με τη χρήση ενός αλγορίθμου που λύνει το πρόβλημα του σακιδίου (knapsack problem) και εφαρμόζεται στο συγκεκριμένο πρόβλημα. Ο αλγόριθμος γράφτηκε σε γλώσσα προγραμματισμού C στο προγραμματιστικό περιβάλλον Dev-C++. Έχουν αναπτυχθεί πολλές τεχνικές επίλυσης του συγκεκριμένου προβλήματος και παρουσιάζουμε κάποιες σημαντικές από αυτές. Σκοπός μας είναι να παροτρύνουμε τον αναγνώστη να ασχοληθεί με το συγκεκριμένο θέμα βοηθώντας τον να κατανοήσει το πρόβλημα και πώς μπορεί να επιλυθεί.

Αρχικά γίνεται μια εισαγωγή στο πρόβλημα και πως αυτό συνδέεται και μπορεί να λυθεί με τη χρήση του συγκεκριμένου αλγορίθμου. Κάνουμε μια εισαγωγή στη γλώσσα προγραμματισμού C και στα np-hard προβλήματα. Στη συνέχεια παρουσιάζουμε τον αλγόριθμο σε C και κάποιες περιπτώσεις χρήσης του. Παρουσιάζουμε επίσης επιπλέον τεχνικές επίλυσης του προβλήματος όπως γενετικό αλγόριθμο και τις συγκρίνουμε μεταξύ τους με παραδείγματα χρήσης.

**Λέξεις κλειδιά:** [πρόβλημα σακιδίου, np-hard, knapsack, αλγόριθμος, βέλτιστη επένδυση, γενετικός αλγόριθμος, brute force knapsack, greedy knapsack]

# Abstract

In developing markets there is a need for quick and optimal investment for profit. When an investor, with a certain initial capital, has to choose from a number of investment opportunities for maximum profit, this need becomes quick and easy as the financial environment is constantly changing. This can be achieved through the use of technology and in particular of computers.

In the present work we try to find optimal investment by selecting the appropriate investments through a number of investment opportunities. This is accomplished by using an algorithm that solves the problem of the knapsack problem and is applied to the problem. The algorithm was written in programming language C in the Dev-C ++ programming environment. Several techniques have been developed to solve this problem and we present some of these important ones. Our goal is to urge the reader to deal with this issue by helping them to understand the problem and how to resolve it.

First, an introduction to the problem is made and how it is linked and can be solved using this algorithm. We make an introduction to programming language C and np-hard problems. Then we present the algorithm in C and some cases of use. We also present additional techniques to solve the problem such as genetic algorithm and compare them with examples of use.



**Keywords:** [knapsack problem, np-hard, algorithm, algorithm, optimal investment, brute force knapsack, greedy knapsack, genetic algorithm]

# ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

# ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

| | |
|---|---|
| γ.α | γενετικός αλγόριθμος |

# ΚΕΦΑΛΑΙΟ 1: Το πρόβλημα του σακιδίου (knapsack problem)

## 1.1. Ορισμός προβλήματος

Στο πρόβλημα του σακιδίου θέλουμε να γεμίσουμε ένα σακίδιο συγκεκριμένης χωρητικότητας. Μέσα από έναν αριθμό n αντικειμένων τα οποία έχουν συγκεκριμένο βάρος και αξία το κάθε ένα, πρέπει να επιλέξουμε εκείνα που μεγιστοποιούν την συνολική αξία του σάκου. Κάθε αντικείμενο έχει βάρος $w_i$ και αξία ή κέρδος $p_i$. Το σακίδιο έχει χωρητικότητα c και το συνολικό βάρος των αντικειμένων που θα εισάγουμε στον σάκο δεν πρέπει να ξεπερνά την χωρητικότητα του c. Στόχος μας είναι να μεγιστοποιήσουμε την αξία του σάκου χωρίς να ξεπεράσουμε την χωρητικότητά του σε βάρος. Το πρόβλημα αυτό μπορεί να διατυπωθεί μαθηματικά ως εξής:

$$\sum_{i=1}^{n} p_i x_i = \mu\acute{\epsilon}\gamma\iota\sigma\tau o$$

με περιορισμούς:

$$\sum_{i=1}^{n} w_i x_i \leq c \; \kappa\alpha\iota \; x_i \; \in [0,1], 1 \leq i \leq n$$

Το $x_i$ παίρνει τιμή 0 αν το αντικείμενο δεν έχει εισαχθεί στο σακίδιο και τιμή 1 αν τελικά το αντικείμενο εισάχθηκε στο σακίδιο. Στο πρόβλημά μας δεχόμαστε ότι μπορεί να υπάρχουν και ίδια αντικείμενα, δηλαδή αντικείμενα με ίδια αξία και ίδιο βάρος. Έτσι στο σακίδιο μπορούμε να βάλουμε και ίδια αντικείμενα, με άλλα λόγια το ίδιο αντικείμενο πάνω από μια φορά.

Για το παραπάνω πρόβλημα έχουν εφαρμοστεί αρκετές τεχνικές επίλυσής του, οι οποίες βρίσκουν την βέλτιστη λύση ή λύσεις κοντά στη βέλτιστη. Όπως θα δούμε στη συνέχεια το συγκεκριμένο πρόβλημα είναι ένα np-hard πρόβλημα και μέχρι στιγμής δεν υπάρχει αλγόριθμος γνωστός για την επίλυσή του σε πολυωνυμικό χρόνο.

Επίσης, έχουν εφαρμοστεί τεχνικές  όπως αυτή της πυκνότητας κέρδους $p_i/w_i$ όπου επιλέγονται πρώτα τα αντικείμενα με μεγαλύτερη πυκνότητα κέρδους. Αυτές οι τεχνικές επιτυγχάνουν μικρό χρόνο εκτέλεσης του αλγορίθμου όμως δεν εγγυώνται την βέλτιστη λύση.

«Το πρόβλημα του σακιδίου ανήκει στην κατηγορία των np-hard προβλημάτων και πιθανώς δεν μπορούμε να βρούμε έναν πολυωνυμικό αλγόριθμο επίλυσής του.»[2]

## 1.2. NP-hard προβλήματα

Το πρόβλημα σακιδίου είναι ένα np-hard πρόβλημα. Τα np-hard προβλήματα (np = non deterministic polynomial) ή μη αιτιοκρατικά πολυωνυμικά είναι προβλήματα που δεν έχει βρεθεί πολυωνυμικός αλγόριθμος επίλυσής τους.

Στην παρούσα εργασία παρουσιάζουμε έναν αλγόριθμο που βρίσκει την βέλτιστη λύση σε ένα πρόβλημα λήψης επενδυτικής απόφασης χρησιμοποιώντας το πρόβλημα του σακιδίου, όμως δεν εκτελείται σε πολυωνυμικό χρόνο. Όπως θα δούμε, από μια συγκεκριμένη τιμή και μετά, ο αλγόριθμος που παρουσιάζουμε δεν εκτελείται σε εύλογο χρονικό διάστημα όμως πάντα βρίσκει την βέλτιστη λύση γιατί ελέγχει όλες τις πιθανές περιπτώσεις.

Για την αντιμετώπιση αυτών των προβλημάτων έχουν αναπτυχθεί προσεγγιστικοί αλγόριθμοι οι οποίοι δεν δίνουν πάντα την βέλτιστη λύση αλλά εκτελούνται σε πολυωνυμικό χρόνο και η λύση που δίνουν, αν δεν είναι η βέλτιστη, δύναται να είναι κοντά στη βέλτιστη.

Τέλος, αξίζει να σημειωθεί, ότι μια λύση σε πολυωνυμικό χρόνο για το συγκεκριμένο πρόβλημα συνεπάγεται λύση για όλα τα προβλήματα της κλάσης np-hard. «Δηλαδή αν υπάρχει αλγόριθμος που να λύνει το πρόβλημα του σακιδίου βρίσκοντας την βέλτιστη λύση σε πολυωνυμικό χρόνο τότε αυτός ο αλγόριθμος μπορεί να επεκταθεί ώστε να λύνει όλα τα προβλήματα τάξης np.»[2]

## 1.3. Σύνδεση προβλήματος knapsack με το πρόβλημα επενδυτικής απόφασης σε χρηματοοικονομικό περιβάλλον

Το πρόβλημα knapsack μπορεί να συνδεθεί εύκολα με το πρόβλημα που παρουσιάζεται σε έναν επενδυτή όταν έχει να επιλέξει ανάμεσα από πολλές επενδύσεις με σκοπό πάντα να μεγιστοποιήσει το όφελός του. Όπως προαναφέραμε, στο πρόβλημα του σάκου έχουμε ένα σακίδιο με συγκεκριμένη χωρητικότητα c και έναν αριθμό (έστω n) αντικειμένων με συγκεκριμένη αξία και βάρος το κάθε ένα. Θέλουμε λοιπόν να γεμίσουμε τον σάκο, χωρίς να ξεπεράσουμε την χωρητικότητά του, με αντικείμενα έτσι ώστε να μεγιστοποιήσουμε την συνολική αξία του σάκου. Για να γίνει πιο κατανοητό, μια παραλλαγή του συγκεκριμένου προβλήματος είναι να σκεφτούμε έναν ληστή ο οποίος έχει διαρρήξει ένα σπίτι και θέλει να χωρέσει στο σακίδιό του όσο το δυνατόν μεγαλύτερης αξίας κλοπιμαία. Το πρόβλημα που αντιμετωπίζει ο ληστής είναι ότι ο σάκος του έχει συγκεκριμένη χωρητικότητα και έχει να επιλέξει ανάμεσα σε πολλά κλοπιμαία τα οποία έχουν και διαφορετικά βάρη και αξίες. Το ερώτημα λοιπόν είναι, ποια αντικείμενα θα πρέπει ο ληστής να βάλει στο σάκο του ώστε να μεγιστοποιήσει την αξία των κλοπιμαίων του αλλά και να μην ξεπεράσει τη χωρητικότητα του σάκου;

Σε αυτό το σημείο μπορούμε να κάνουμε τη σύνδεση του παραπάνω προβλήματος και του διλλήματος του κλέφτη με το πρόβλημα που αντιμετωπίζει ένας επενδυτής. Ας υποθέσουμε έναν επενδυτή με αρχικό κεφάλαιο c, ο οποίος θέλει να επενδύσει το κεφάλαιό του (ολόκληρο ή τμηματικά) σε διάφορες επενδύσεις. Αν υποθέσουμε ότι το αρχικό κεφάλαιο του επενδυτή είναι η χωρητικότητα του σάκου και οι διάφορες επενδύσεις έχουν η κάθε μία ένα κόστος (βάρος) και μία απόδοση (αξία), τότε έχουμε συνδέσει το πρόβλημα του σάκου με το πρόβλημα ενός επενδυτή. Στην ουσία βρίσκοντας λύση στο πρόβλημα knapsack βρίσκουμε λύση και στο πρόβλημα του επενδυτή.

Για να βρούμε λύση θα πρέπει να εξετάσουμε όλες τις πιθανές επιλογές αντικειμένων, ή στη περίπτωσή μας επενδύσεων, και να συγκρίνουμε τις τελικές αξίες ή αποδόσεις διαλέγοντας την μέγιστη. Στη περίπτωση που το αρχικό κεφάλαιο είναι μεγάλο ή και οι ευκαιρίες είναι πολλές οι πιθανές επιλογές και ο αριθμός

διεργασιών για να βρεθεί το βέλτιστο αυξάνονται εκθετικά και είναι δύσκολο να υπολογιστούν σε μικρό χρονικό διάστημα και χωρίς τη χρήση αλγορίθμου. Χρειαζόμαστε λοιπόν γρήγορες διεργασίες και υπολογισμούς εξετάζοντας όλα τα πιθανά αποτελέσματα. Σε αυτό θα μας βοηθήσει η γλώσσα προγραμματισμού C η οποία θα εκτελέσει τον αλγόριθμο που θα παρουσιάσουμε παρακάτω.

# ΚΕΦΑΛΑΙΟ 2: Η γλώσσα προγραμματισμού C

## 2.1. Εισαγωγή στη C

Η γλώσσα προγραμματισμού C είναι μια γενικής χρήσης γλώσσα που την διακρίνουν οι δομές δεδομένων της και η γενικότητά της. Επιλέξαμε την γλώσσα C γιατί λόγων της γενικότητάς της και της ευκολίας που έχει να υλοποιεί αλγορίθμους γίνεται πιο άνετη σε όλους τους χρήστες αρχάριους ή μη. Η γλώσσα σχεδιάστηκε αρχικά για το λειτουργικό σύστημα UNIX από τον Dennis Ritchie. Η γλώσσα C δεν έχει περιορισμούς ως προς το υλικό ή το σύστημα που θα την χρησιμοποιήσει και είναι εύκολο να υλοποιηθούν προγράμματα που θα «τρέχουν» σε οποιοδήποτε μηχάνημα την υποστηρίζει.

Τα κυριότερα στοιχεία της γλώσσας που θα μας βοηθήσουν είναι οι τύποι δεδομένων, οι συναρτήσεις, οι δομές ελέγχου, οι δομές δεδομένων και η δυνατότητα αναδρομής και επαναλήψεων.

## 2.2. Τύποι δεδομένων

Η γλώσσα C υποστηρίζει πολλούς τύπους δεδομένων που καλύπτουν τις βασικές ανάγκες ενός προγραμματιστή. Οι κυριότεροι τύποι που θα χρησιμοποιήσουμε είναι οι int και float .

Ο τύπος int σημαίνει ότι οι μεταβλητές αυτού του τύπου είναι ακέραιες σε αντίθεση με τον τύπο float που είναι αριθμοί ή μεταβλητές με κλασματικό μέρος. Αν για παράδειγμα θέλουμε να δηλώσουμε μια μεταβλητή “x” σαν ακέραιο αριθμό αρκεί να γράψουμε “int x;” ενώ για αριθμό κινητής υποδιαστολής γράφουμε “float x;”.

## 2.3. Συναρτήσεις

Οι συναρτήσεις όπως και στα μαθηματικά είναι διαδικασίες που δέχονται ορίσματα (π.χ αριθμούς) και επιστρέφουν κάποια τιμή. Υπάρχουν και συναρτήσεις που δε δέχονται ορίσματα αλλά και συναρτήσεις που δεν επιστρέφουν κάποια τιμή.

Η C έχει σχεδιαστεί για να κάνει τις συναρτήσεις εύχρηστες και αποτελεσματικές. Με αυτό τον τρόπο οι συναρτήσεις χωρίζουν το πρόγραμμα σε μικρότερες εργασίες και επιτρέπουν στους προγραμματιστές να τις αλλάζουν εύκολα σε περίπτωση που χρειάζεται αλλαγή το πρόγραμμα, αντί να αρχίζουν από την αρχή.

Η συνάρτηση δηλώνεται ανάλογα με τον τύπο δεδομένων που επιστρέφει και δέχεται. Δηλαδή αν μια συνάρτηση “function()” επιστρέφει μια ακέραια τιμή θα την δηλώσουμε ως “int function()”. Παρατηρούμε ότι εντός των παρενθέσεων υπάρχει κενό γιατί η συγκεκριμένη συνάρτηση επιλέξαμε να μην δέχεται ορίσματα. Αν η συγκεκριμένη συνάρτηση δεχόταν μία ακέραια μεταβλητή “x”, τύπου “int” δηλαδή, τότε η συνάρτηση θα δηλωνόταν ως “int function(int x)”. Άρα η συνάρτηση “int function(int x)” δέχεται έναν ακέραιο αριθμό “x” και επιστρέφει σαν αποτέλεσμα έναν ακέραιο αριθμό. Αν η συνάρτηση δεν επέστρεφε τίποτα τότε θα την δηλώναμε ως void, δηλαδή “void function(int x)”.

## 2.4. Δομές ελέγχου και δομές δεδομένων

Οι δομές ελέγχου στη C είναι οι έλεγχοι που μπορεί να πραγματοποιήσει η γλώσσα για την επίλυση ενός προβλήματος. Αν έχουμε την περίπτωση όπου ένας χρήστης πληκτρολογεί χαρακτήρες και θέλουμε όταν ο χρήστης πληκτρολογήσει τον χαρακτήρα ‘c’ να του εμφανιστεί ένα μήνυμα, τότε ένα πρόγραμμα που θα έλυνε το παραπάνω θα περιείχε έναν έλεγχο. Ο έλεγχος αυτός θα ήταν, “αν ο χρήστης πληκτρολόγησε τον χαρακτήρα ‘c’ τότε εμφάνισε το μήνυμα” .

Ο έλεγχος στη C γίνεται με την εντολή “if”. Η σύνταξη της εντολής είναι η παρακάτω:

```
if(παράσταση)
εντολή 1
εντολή 2
```

Δομές δεδομένων στη C είναι μεταβλητές ίδιου η διαφορετικού τύπου που έχουν ομαδοποιηθεί για ευκολία στη χρήση τους. Σε μεγάλα προγράμματα

χρησιμοποιούνται οι δομές για να υπάρχει ευκολία και ομαδοποίηση στις μεταβλητές διαφορετικού τύπου.

Ένα απλό παράδειγμα δομής δεδομένων είναι η αναπαράσταση ενός σημείου στο δισδιάστατο χώρο. Στα μαθηματικά ένα σημείο στο καρτεσιανό επίπεδο αναπαρίσταται με δύο μεταβλητές “x” και “y”, την τετμημένη και την τεταγμένη δηλαδή. Για ευκολία θα θεωρήσουμε ότι οι δύο αυτές μεταβλητές είναι ακέραιοι αριθμοί. Στη C οι δύο αυτές μεταβλητές μπορούν να τοποθετηθούν σε μια δομή ως εξής:

```
struct point {
int x;
int y;
};
```

Η δομή “point” περιέχει τις μεταβλητές “x” και “y” και μπορούμε να δημιουργήσουμε ένα αντικείμενο τύπου point το οποίο θα αντιπροσωπεύει ένα σημείο στο χώρο. Για παράδειγμα αν θέλουμε να δημιουργήσουμε το σημείο (x,y)=(1,2) θα δηλώσουμε το εξής:

```
struct point shmeio = {1,2};
```

Για να χειριστούμε και να κάνουμε αναφορά στις μεταβλητές του αντικειμένου (σημείου) δηλώνουμε «όνομα αντικειμένου».«όνομα μεταβλητής», δηλαδή για τη μεταβλητή x του σημείου που αναφέραμε θα δηλώναμε “shmeio.x” και αναφέρομαι στη x μεταβλητή (τετμημένη) του αντικειμένου (σημείου) “shmeio” τύπου “point”.

## 2.5. Αναδρομικότητα και επανάληψη

Τέλος, η C μας παρέχει τη δυνατότητα της επανάληψης και της αναδρομής. Οι συναρτήσεις της C μπορούν να χρησιμοποιηθούν αναδρομικά, δηλαδή μια συνάρτηση μπορεί να καλεί τον εαυτό της.

Στο πρόβλημά μας η αναδρομή μας βοηθάει στο να καλούμε την ίδια κύρια συνάρτηση που έχουμε χρησιμοποιήσει στον αλγόριθμο, ώστε κάθε φορά να

ελέγχεται τι διαθέσιμο χώρο έχουμε στο σάκο μας. Έχοντας κάνει τη σύνδεση με το χρηματοοικονομικό πρόβλημα του επενδυτή μετά από κάθε επένδυση το αρχικό κεφάλαιο μειώνεται αφού μέρος του ξοδεύεται στην αγορά επενδυτικής ευκαιρίας. Έτσι στο επόμενο βήμα επένδυσης το κεφάλαιο είναι μικρότερο από το αρχικό. Κάθε φορά λοιπόν με διαφορετικό κεφάλαιο ο επενδυτής είναι σαν να βρίσκεται στο αρχικό βήμα, όπου έχει ένα αρχικό κεφάλαιο και διάφορες ευκαιρίες. Έτσι επαναλαμβάνουμε το πρώτο βήμα πολλές φορές και αυτό επιτυγχάνεται με την αναδρομή που θα γίνει στην υλοποίηση του κώδικα σε C.

# ΚΕΦΑΛΑΙΟ 3: Υλοποίηση του αλγορίθμου

## 3.1. Το πρόγραμμα που αναπτύχθηκε σε C

Σε αυτό το κεφάλαιο παρουσιάζουμε το πρόγραμμα σε C που λύνει το πρόβλημα knapsack και κατ' επέκταση το πρόβλημα του επενδυτή. Το πρόγραμμα φαίνεται στον σχολιασμένο κώδικα που ακολουθεί.

```
//----------------------Αρχή κώδικα---------------
#include<stdio.h>
#include <float.h>
#include <limits.h>
//-------------------------------------------------------------------
--------
int main(void){
//Δομή (struct) "Item" που περιέχει τις μεταβλητές chosen, size και val
struct Item
      {
      unsigned chosen;//
      float size;
      float val;
      };

//-------------------------------------------------------------------
-------
// problem parameters
//(Οι παράμετροι του προβλήματος που είναι 7 επενδυτικές ευκαιρίες ή
//items
//-------------------------------------------------------------------
-------
```

```
int N=7;
//---------------------
//εδώ δηλώνουμε έναν πίνακα αντικειμένων τύπου "Item" με όνομα item.
//Άρα π.χ το πρώτο στοιχείο  του πίνακα item, δηλαδή το item[0],
//θα έχει κόστος (μέγεθος ή βάρος) 7.5 εκατομμύρια ευρώ και απόδοση val 37.5 //εκ. Ευρώ.//
//----Δήλωση πίνακα item, N θέσεων, που περιέχει αντικείμενα τύπου //"Item", δηλαδή τις επενδύσεις που δηλώνουμε παρακάτω.//
struct Item item[N];
//---------------------
//----Δήλωση των επενδύσεων. Κάθε επένδυση έχει ένα κόστος (size) και//
//μια απόδοση ή κέρδος (val)-------------//
item[0].size=7.5;
item[0].val=37.5;
item[1].size=6.25;
item[1].val=18.75;
item[2].size=0.5;
item[2].val=1;
item[3].size=12.5;
item[3].val=18.75;
item[4].size=10.25;
item[4].val=11.275;
item[5].size=10.775;
item[5].val=5.3875;
item[6].size=12.250;
item[6].val=1.225;
```

```
//---------------------------

float    Space = 25;//υποθέτουμε ότι σαν επενδυτής έχουμε ένα αρχικό
//κεφάλαιο 25 εκ. ευρώ (=space).//

//----Δήλωση της μεταβλητής Rechoose_Limit-----

//Η Rechoose_Limit είναι μια μεταβλητή με την οποία

//δηλώνουμε τις φορές που επιτρέπουμε να επιλεχτεί ένα item.

//Αυθαίρετα επιλέγουμε 8 ώστε να καλύψουμε

//το όριο επιλογής και να έχουμε την βέλτιστη απόδοση.

unsigned   Rechoose_Limit = 8;

//--------------------//

unsigned Max_chosen [N];//βοηθητικός πίνακας 7 θέσεων

// που θα χρειαστεί στη συνέχεια.

float  Max = 0;

float  Space_remaining_when_at_Max = 0;


//-----------------------------------------------------------------------
-------

void knap(float space, float total_val)

      {

      float t=0;

      float s=0;

      for(int i=0;i<N;i++)

            {

            if(item[i].chosen>=Rechoose_Limit)

//Εαν το item επιλεχτεί περισσότερες φορές από το όριο τύπωσε το
//παρακάτω:

                  {
```

```
// printf("* note: a used-defined limit was reached (item %i can be chosen up to %u times).\n",i,Rechoose_Limit);
                }
            else //αλλιώς
              if(space>=item[i].size)//αν το space είναι μεγαλύτερο ή //ίσο από το κόστος της ευκαιρίας,
//δηλαδή αν μπορώ να αγοράζω κάποια επιπλέον ευκαιρία, τότε
//κάνε τα παρακάτω:
                  {
                  item[i].chosen++;//Αύξησε το chosen του εκάστοτε //item κατά 1, δηλαδή επέλεξέ το μια φορά,
                  t = total_val + item[i].val;//κάνε το t ίσο με: (total-val //(συνολική απόδοση ή αξία) + η απόδοση του item[i])
                  s = space - item[i].size;//το s τώρα ισούται με: (space - το //κόστος του item[i]), αφού μειώθηκε το αρχικό κεφάλαιο
                  if(t>Max) //αν το t δηλαδή η μέχρι στιγμής συνολική //απόδοση είναι μεγαλύτερη του Max τότε:
//εδώ το Max αρχικά είναι 0, άρα θα μπούμε στο πρώτο if αναγκαστικά:
                        {
Max=t;//Μετά όμως το Max θα ισούται με t, με την συνολική απόδοση δηλαδή.
//Άρα αν επιστρέψω στην προηγούμενη if, θα ελέγχεται αν η μέχρι στιγμής //απόδοσή μου ξεπερνάει το Max
//Εδώ στην ουσία ελέγχουμε αν η εκάστοτε απόδοση t, ξεπερνάει το Max.
//Με άλλα λόγια βρίσκουμε το μέγιστο t, συγκρίνοντάς το με αυτό που //προκύπτει κάθε φορά.
for(int j=0;j<N;j++) Max_chosen[j]=item[j].chosen;//Αν λοιπόν t>Max //τότε αποθήκευσε το item[j].chosen στον πίνακα Max_chosen[j]
```

```
            Space_remaining_when_at_Max = s;//s είναι τα χρήματα που
//δεν χρησιμοποιήθηκαν από το αρχικό κεφάλαιο.//
printf("* max %f found (with %f space remaining)
\n",Max,Space_remaining_when_at_Max);
                        }
knap(s, t);//ξανατρέξε την knap με το νέο space=s και το νέο
//total_val=t.
//Εδώ βλέπουμε την αναδρομή.
            item[i].chosen--;//μείωση του item[i].chosen κατά 1
                  }
            }
      }
//-------------------------------------------------------------------

printf("Space Limit =  %f\n",Space);
printf("Rechoose Limit = up to %u times per item\n",Rechoose_Limit);
//-------αρχικά μηδενίζουμε τις μεταβλητές chosen και όλες τις θέσεις
του πίνακα //Max_chosen[j]----
for(int j=0;j<N;j++)  { item[j].chosen=0; Max_chosen[j]=0; }
//---------------------
knap(Space,0);//καλούμε την knap με το αρχικό κεφάλαιό μας Space και
αρχική //αξία = 0
//(λογικό, γιατί στο πρώτο βήμα δεν έχουμε επιλέξει ακόμα κάποιο
επενδυτικό //προϊόν)
//-----------------------------------------------------
float t = 0;
for(int j=0;j<N;j++)
 {
```

```
 t = t + Max_chosen[j]*item[j].val;
//εδώ υπολογίζουμε το t που ισούται
// με τις αποθηκευμένες φορές που επιλέχτηκε ένα item στο πίνακα
//Max_chosen[j] επί την απόδοση του item.
printf("item %d (goodness = %10.2f,%d) used %d times (running total
%f)\n",j,(item[j].val/item[j].size),item[j].chosen, Max_chosen[j],t);
 }
printf("Optimal investment benefit = %f\n",Max);//εκτύπωση του
//συνολικού κέρδους.
printf("Unused = %f\n",Space_remaining_when_at_Max);//εκτύπωση του
//ποσού που έμεινε ανεκμετάλλευτο.
}
//----------------------Τέλος κώδικα ----------------//
```

Όπως φαίνεται ο κώδικας είναι σχολιασμένος για να μπορεί ο αναγνώστης να καταλάβει πως λειτουργεί. Αυτό που πρέπει να επισημανθεί είναι ότι η παραπάνω μέθοδος βρίσκει όλα τα πιθανά αποτελέσματα του προβλήματος και τα συγκρίνει ώστε να επιλεγεί το καλύτερο. Προφανώς, επαναλαμβάνουμε ότι αυτή η μέθοδος, όταν οι παράμετροι είναι πολλές, εκτελείται σε μη μεγάλο χρόνο που σημαίνει ότι αργεί πολύ να δώσει αποτελέσματα και αυτό είναι απαγορευτικό σε κάποιες περιπτώσεις. Για μικρό αριθμό δεδομένων όμως, όπως της παραπάνω περίπτωσης, δίνει γρήγορα αποτελέσματα και πάντα την βέλτιστη λύση.

## 3.2. Αποτελέσματα χρήσης κώδικα knapsack

Τα αποτελέσματα του κώδικα έδωσαν την βέλτιστη λύση επιλέγοντας κάποιες επενδυτικές ευκαιρίες. Στον παρακάτω πίνακα φαίνονται οι επενδυτικές ευκαιρίες που είχαμε στην αρχή του προβλήματος.

Πίνακας 1. Επενδυτικές ευκαιρίες μοντέλου knapsack

| Αύξων αριθμός επένδυσης (Α/Α) | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) |
|---|---|---|
| 0 | 7.5 | 37.5 |
| 1 | 6.25 | 18.75 |
| 2 | 0.5 | 1 |
| 3 | 12.5 | 18.75 |
| 4 | 10.25 | 11.275 |
| 5 | 10.775 | 5.3875 |
| 6 | 12.25 | 1.225 |

Τα αποτελέσματα παρουσιάζονται παρακάτω:

Συνολική απόδοση (βέλτιστη): 117.5 εκατομμύρια ευρώ

Η επένδυση 0 επιλέχτηκε 3 φορές και η επένδυση 2 επιλέχτηκε 5 φορές. Δηλαδή ο επενδυτής για να έχει την βέλτιστη απόδοση των 117.5 εκ. ευρώ, θα πρέπει να επενδύσει το αρχικό του κεφάλαιο (25 εκ. ευρώ) αγοράζοντας το προϊόν 0, 3 φορές και το προϊόν 2 , 5 φορές.

Πίνακας 2. Αποτελέσματα μοντέλου knapsack

| Α/Α επένδυσης που επιλέχτηκε | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Πλήθος επιλογών επένδυσης | Συνολικό κόστος επιλογής(εκ. ευρώ) | Συνολική απόδοση επιλογής(εκ. ευρώ) |
|---|---|---|---|---|---|
| 0 | 7.5 | 37.5 | 3 | 22.5 | 112.5 |
| 2 | 0.5 | 1 | 5 | 2.5 | 5 |
| Σύνολο | | | | | 117.5 |

Έχουμε υποθέσει ότι οι επενδυτικές ευκαιρίες μπορούν να επιλεχτούν πάνω από μια φορά. Από χρηματοοικονομικής πλευράς αυτές θα μπορούσαν να είναι μετοχές, όπου ένας επενδυτής μπορεί να αγοράσει πολλά μερίδια από κάθε μετοχή. Θα μπορούσε επίσης να είναι απλές επενδύσεις με μια απόδοση, για παράδειγμα αγορά γης ή ακινήτων με μια μελλοντική απόδοση. Οι αποδόσεις μπορεί να είναι σχετικά σίγουρες (όπως ένα δεκαετές ομόλογο μιας ισχυρής οικονομίας που έχει μικρή πιθανότητα απώλειας χρημάτων) μπορεί και όχι. Τέλος, βλέπουμε ότι χρησιμοποιήθηκε όλο το αρχικό κεφάλαιο των 25 εκατομμυρίων ευρώ.

## 3.3. Η brute-force προσέγγιση του κώδικα

Ο κώδικας για το πρόβλημα που παρουσιάσαμε χρησιμοποιεί τη τεχνική brute force, η οποία βρίσκει όλες τις πιθανές επενδύσεις που μπορούν να επιλεγούν και διαλέγει εκείνες που μεγιστοποιούν την απόδοση. Η τεχνική αυτή είναι ο πιο προφανής τρόπος επίλυσης ενός προβλήματος και έχει μειονεκτήματα και πλεονεκτήματα.

Το βασικό μειονέκτημά της είναι ότι όσο προσθέτουμε παραμέτρους στο πρόβλημα τόσο πιο αργή γίνεται σαν μέθοδος. Έχει δηλαδή μεγάλη πολυπλοκότητα γιατί ελέγχονται όλες οι δυνατές επιλογές και όσο αυξάνονται οι επενδύσεις αυξάνονται εκθετικά και οι δυνατές επιλογές. Το πρόγραμμα όπως θα δούμε στη συνέχεια, για κάποια τιμή αρχικού κεφαλαίου και μετά, δεν δίνει αποτελέσματα σε εύλογο χρονικό διάστημα. Το ίδιο συμβαίνει και για μεγάλο αριθμό επενδύσεων.

Το πλεονέκτημα της τεχνικής αυτής είναι ότι είναι εύκολα υλοποιήσιμη, κατανοητή και δίνει βέλτιστη λύση. Επειδή αποτελεί τον πιο απλό τρόπο επίλυσης ενός προβλήματος τέτοιου είδους, είναι και εύκολο να γίνει εφαρμόσιμη. Στο πρόγραμμά μας, βρήκαμε όλους τους δυνατούς συνδυασμούς που μπορούν να επιλεγούν, τους συγκρίναμε μεταξύ τους ως προς την απόδοσή τους και διαλέξαμε αυτή την επένδυση με την μεγαλύτερη απόδοση.

Τέλος, έχουν αναπτυχθεί πολλές τεχνικές βελτιστοποίησης του κώδικα επίλυσης και έχουν γίνει βελτιώσεις, όμως αρκετές από αυτές τις τεχνικές αν και μειώνουν την ταχύτητα δεν δίνουν πάντα την βέλτιστη λύση.

## 3.4. Περίπτωση χρήσης 1

Στην προηγούμενη εφαρμογή του κώδικα, που παρουσιάζεται στο 3.1. υποκεφάλαιο, υποθέσαμε ότι οι επενδύσεις μπορούν να επιλεχτούν πάνω από μια φορά. Τώρα θα υποθέσουμε ότι μια επένδυση μπορεί να επιλεχτεί το μέγιστο 1 φορά.

Οι επενδύσεις δηλαδή μπορεί να είναι μοναδικές και αυτό μπορεί να συμβεί στη περίπτωση που έχουμε επενδύσεις όπως αγοράς ακινήτων ή αγοράς γης, γιατί δεν γίνεται να αγοραστεί ένα κομμάτι γης ή ένα σπίτι πάνω από μια φορά. Έτσι θα τροποποιήσουμε τον κώδικα ώστε κάθε επένδυση αν επιλέγεται, να επιλέγεται το πολύ μια φορά. Επιπλέον, αλλάζουμε και τις αξίες και αποδόσεις των επενδύσεων.

Για να γίνει επιλογή κάθε επένδυσης μόνο 1 φορά απλά θέτουμε την μεταβλητή “Rechoose_Limit” να ισούται με 1. Με αυτό τον τρόπο το όριο επανεπιλογής μιας επένδυσης (item) γίνεται 1 και δεν μπορεί να επιλεχτεί δεύτερη φορά.

Σε αυτή τη περίπτωση υποθέτουμε ξανά ότι ο επενδυτής έχει αρχικό κεφάλαιο 25 εκατομμύρια ευρώ και θέλει να τα επενδύσει σε χρηματοοικονομικά προϊόντα. Τα προϊόντα είναι πάλι 7 και κάθε ένα έχει μια αξία και μία απόδοση. Στον επόμενο πίνακα φαίνονται οι νέες επενδυτικές ευκαιρίες (προϊόντα) με τα στοιχεία τους.

**Πίνακας 3. Επενδυτικές ευκαιρίες περίπτωσης 1**

| Αύξων αριθμός επένδυσης (Α/Α) | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) |
|---|---|---|
| 0 | 8.25 | 7.25 |
| 1 | 10.25 | 11.275 |
| 2 | 6.775 | 3.375 |
| 3 | 10.5 | 18.75 |
| 4 | 1.5 | 1 |
| 5 | 4.25 | 18 |
| 6 | 13.5 | 3.5 |

Αφού εκτελέσουμε τον κώδικα με τις αλλαγές των προϊόντων και της μεταβλητής “Rechoose_Limit” παρουσιάζουμε τα αποτελέσματα στον επόμενο πίνακα.

Πίνακας 4. Αποτελέσματα περίπτωσης 1

| Α/Α επένδυσης που επιλέχτηκε | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Πλήθος επιλογών επένδυσης | Συνολικό κόστος επιλογής(εκ. ευρώ) | Συνολική απόδοση επιλογής(εκ. ευρώ) |
|---|---|---|---|---|---|
| 1 | 10.25 | 11.275 | 1 | 10.25 | 11.275 |
| 3 | 10.5 | 18.75 | 1 | 10.5 | 18.75 |
| 5 | 4.25 | 18 | 1 | 4.25 | 18 |
| Σύνολο | | | | | 48.025 |

Συνολική απόδοση (βέλτιστη): 48.025 εκατομμύρια ευρώ

Όπως φαίνεται στον προηγούμενο πίνακα κάθε επένδυση επιλέχτηκε μια φορά και χρησιμοποιήθηκε όλο το αρχικό κεφάλαιο. Οι επενδύσεις που επιλέχτηκαν είναι οι 1, 3 και 5 με συνολική απόδοση 48.025 εκ. ευρώ. Και σε αυτή την περίπτωση χρησιμοποιήθηκε όλο το αρχικό κεφάλαιο.

## 3.5. Περίπτωση χρήσης 2

Στη περίπτωση χρήσης 2 θα προσθέσουμε επιπλέον επενδύσεις και θα αυξήσουμε το αρχικό κεφάλαιο από 25 εκατομμύρια σε 40 εκατομμύρια ευρώ. Επίσης θα υποθέσουμε ότι κάθε επένδυση μπορεί να επιλεχτεί πάνω από μια φορά. Είναι λογικό πως αυξάνοντας το κεφάλαιο και προσθέτοντας επενδύσεις αυξάνουμε και την πολυπλοκότητα. Ακολουθώντας αυτή τη τεχνική θα έχουμε μια αρχική εικόνα για τα όρια αυτής της μεθόδου και μέχρι ποιο σημείο μπορεί να αποδώσει επαρκώς. Στον παρακάτω πίνακα βλέπουμε τα νέα δεδομένα που προκύπτουν για την περίπτωση χρήσης 2 ύστερα από την προσθήκη νέων επενδύσεων. Οι επενδύσεις που προστέθηκαν είναι οι 7,8 και 9.

Πίνακας 5. Επενδυτικές ευκαιρίες περίπτωσης 2

| Αύξων αριθμός επένδυσης (Α/Α) | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) |
|---|---|---|
| 0 | 8.25 | 7.25 |
| 1 | 10.25 | 11.275 |
| 2 | 6.775 | 3.375 |
| 3 | 10.5 | 18.75 |
| 4 | 1.5 | 1 |
| 5 | 4.25 | 18 |
| 6 | 13.5 | 3.5 |
| 7 | 8 | 12 |
| 8 | 3 | 7 |
| 9 | 14 | 20 |

Πίνακας 6. Αποτελέσματα περίπτωσης 2

| Α/Α επένδυσης που επιλέχτηκε | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Πλήθος επιλογών επένδυσης | Συνολικό κόστος επιλογής(εκ. ευρώ) | Συνολική απόδοση επιλογής(εκ. ευρώ) |
|---|---|---|---|---|---|
| 4 | 1.5 | 1 | 1 | 1.5 | 1 |
| 5 | 4.25 | 18 | 9 | 162 | 162 |
| Σύνολο | | | | | 163 |

Στον παραπάνω πίνακα των αποτελεσμάτων βλέπουμε ότι επιλέχτηκε 9 φορές η επένδυση 5 και 1 φορά η επένδυση 4, με συνολική απόδοση 163 εκατομμύρια ενώ έμειναν ανεκμετάλλευτα 0.25 εκατομμύρια ευρώ από το αρχικό κεφάλαιο αφού δεν υπήρχε επιλογή που το κόστος της να ταιριάζει σε αυτό το ποσό.

Με τη πρόσθεση επενδύσεων και την αύξηση του κεφαλαίου αυξήσαμε τον χρόνο εκτέλεσης περίπου στα 3 δευτερόλεπτα στον υπολογιστή που εκτελέσαμε τον κώδικα. Βέβαια αυτός ο χρόνος μπορεί να είναι επιτρεπτός σε μια προσομοίωση επενδύσεων κεφαλαίου αλλά και σε πραγματικά δεδομένα.

Αυτό που παρατηρούμε είναι ότι επιλέγονται οι επενδύσεις με το μεγαλύτερο λόγο απόδοσης επένδυσης προς κόστος επένδυσης ή με όρους προβλήματος σάκου, αξίας αντικειμένου προς βάρος αντικειμένου. Όπως αναφέραμε στη παράγραφο 1.1 αυτή η τεχνική της μεγαλύτερης πυκνότητας κέρδους $p_i/w_i$ μπορεί να εφαρμοστεί αλλά δεν δίνει σε όλες τις περιπτώσεις τη βέλτιστη επένδυση.

Για παράδειγμα αν εφαρμόζαμε την τεχνική πυκνότητας κέρδους το κριτήριο επιλογής επένδυσης θα ήταν: από τις διαθέσιμες επενδύσεις διάλεξε πρώτα εκείνη με τον μεγαλύτερο λόγο πυκνότητας κέρδους $p_i/w_i$ . Αυτή η τεχνική όμως δεν δίνει πάντα τη βέλτιστη λύση γιατί αν είχαμε 3 επενδύσεις με αποδόσεις 40,25,25 και κόστη 20,15,15 αντίστοιχα, με ένα αρχικό κεφάλαιο 30, θα επιλεγόταν η πρώτη επένδυση με πυκνότητα κέρδους $p_i/w_i=\frac{40}{20} = 0.5$ μία φορά και δεν θα μπορούσαμε να επιλέξουμε κάποια άλλη γιατί θα υπερβαίναμε το όριο του κεφαλαίου μας. Στη περίπτωση αυτή η συνολική απόδοση είναι 40 ενώ η βέλτιστη είναι 50, να επιλεγούν δηλαδή η δεύτερη και τρίτη επένδυση με κόστη 15 και 15 αντίστοιχα. [2]

## 3.6. Η άπληστη μέθοδος προσέγγισης (greedy method)

Η τεχνική της πυκνότητας κέρδους είναι μια άπληστη (greedy) μέθοδος προσέγγισης του προβλήματος. Σε κάθε βήμα της μεθόδου εφαρμόζεται το κριτήριο επιλογής του μεγαλύτερου λόγου κέρδους προς κόστος . Το κριτήριο αυτό ονομάζεται και άπληστο κριτήριο. Η μέθοδος αυτή είναι πολύ πιο γρήγορη από τη προσέγγιση brute-force που έχουμε παρουσιάσει και είναι μια καλή προσέγγιση του προβλήματος. Τέτοιου είδους προβλήματα μπορούν να αντιμετωπιστούν με αυτή τη μέθοδο γιατί δεν υπάρχουν προβλήματα μεγάλου χρόνου εκτέλεσης. Όμως αυτή η μέθοδος δεν βρίσκει πάντα τη βέλτιστη λύση αλλά την προσεγγίζει ικανοποιητικά πολλές φορές. Αυτό θα φανεί στη πράξη με τον παρακάτω αλγόριθμο.

```
    # include<stdio.h>
void knapsack(int n, float value[], float benefit[], float space) {
        float x[20], tp = 0;
        int i, j, u;
```

```
 u = space;

   for (i = 0; i < n; i++)
      x[i] = 0.0;
  for (i = 0; i < n; i++) {
      if (value[i] > u)
         break; //'continue' για να μη σταματάει.
      else {
         x[i] = 1.0;
         tp = tp + benefit[i];
         u = u - value[i];
      }
   }
    if (i < n)
    tp = tp + (x[i] * benefit[i]);
   printf("\n Investments options:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);
    printf("\nOptimal benefit is:- %f", tp);
}
 int main() {
   float value[20], benefit[20], space;
   int num, i, j;
   float ratio[20], temp;
    printf("\nEnter the number of investments:- ");
   scanf("%d", &num);
 printf("\nEnter the values and benefits of each investment:- ");
   for (i = 0; i < num; i++) {
      printf("\nEnter the value of %d investment:- ",i);
      scanf("%f", &value[i]);
      printf("\nEnter the benefit of %d investment:- ",i);
      scanf("%f", &benefit[i]);
```

```
    }
     printf("\nEnter the initial capital:- ");
    scanf("%f", &space);

    for (i = 0; i < num; i++) {
       ratio[i] = benefit[i] / value[i];
    }

    for (i = 0; i < num; i++) {
       for (j = i + 1; j < num; j++) {
          if (ratio[i] < ratio[j]) {
             temp = ratio[j];
             ratio[j] = ratio[i];
             ratio[i] = temp;
            temp = value[j];
             value[j] = value[i];
             value[i] = temp;
              temp = benefit[j];
             benefit[j] = benefit[i];
             benefit[i] = temp;         }
       }
    }
     knapsack(num, value, benefit, space);
    return(0);
}
```

Στον παραπάνω αλγόριθμο εφαρμόζεται η άπληστη μέθοδος και δεν υπάρχει αναδρομή στη κλήση της συνάρτησης knapsack. Στην αρχή ζητείται από το χρήστη να εισάγει τον αριθμό των επενδύσεων με τα χαρακτηριστικά τους (κόστος, απόδοση) και το αρχικό κεφάλαιο. Στη συνέχεια τυπώνεται ένας πίνακας όπου σε κάθε θέση του έχει την τιμή 0 ή 1. Αν έχει την τιμή 1 σημαίνει ότι επιλέχτηκε η επένδυση ενώ αν αντίθετα την τιμή 0 αν δεν έχει επιλεχτεί η επένδυση. Δηλαδή, αν

στη θέση 0 έχει τη τιμή 1 τότε έχει επιλεγεί η πρώτη επένδυση που ταξινομήθηκε στη λίστα φθίνουσας πυκνότητας κέρδους. Αυτός ο τρόπος επιστροφής αποτελεσμάτων για το ποιά επένδυση επιλέχτηκε, δίνει στο πρόβλημα και την εναλλακτική ονομασία 0-1 knapsack διότι επιστρέφεται ένας πίνακα με μηδενικά και άσσους που δηλώνει τις επιλεγμένες επενδύσεις. Στη περίπτωσή μας, εισάγαμε τις 3 επενδύσεις που φαίνονται στο παρακάτω πίνακα.

**Πίνακας 7. Επενδυτικές ευκαιρίες άπληστης μεθόδου**

| Αύξων αριθμός επένδυσης (A/A) | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Λόγος απόδοσης προς κόστος ($p_i/w_i$) (Benefit/Cost) |
|---|---|---|---|
| 0 | 20 | 40 | 0.5 |
| 1 | 15 | 25 | 1.67 |
| 2 | 15 | 25 | 1.67 |

Εισάγαμε τις επενδύσεις που αναφέρθηκαν στο τέλος της παραγράφου 3.6 για να δείξουμε ότι η μέθοδος αυτή δεν δίνει πάντα τη βέλτιστη λύση. Στον επόμενο πίνακα φαίνονται τα αποτελέσματα ύστερα από την εκτέλεση του κώδικα (greedy) με αρχικό κεφάλαιο 30 .

**Πίνακας 8. Αποτελέσματα άπληστης μεθόδου**

| A/A επένδυσης που επιλέχτηκε | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Πλήθος επιλογών επένδυσης | Συνολικό κόστος επιλογής(εκ. ευρώ) | Συνολική απόδοση επιλογής(εκ. ευρώ) |
|---|---|---|---|---|---|
| 0 | 20 | 40 | 1 | 20 | 40 |
| Σύνολο | | | | | 40 |

Ο αλγόριθμος αφού βάλει σε φθίνουσα σειρά τις επενδύσεις ως προς το λόγο απόδοσης προς κόστος (ratio), επιλέγει πρώτα αυτή με το μεγαλύτερο λόγο η οποία όπως φαίνεται στο πίνακα 7 τυχαίνει να είναι η πρώτη επένδυση με αύξων αριθμό 0. Αυτή η επένδυση όμως κοστίζει 20 και το αρχικό κεφάλαιο είναι 30, άρα όταν επιλεγεί δεν υπάρχει άλλη διαθέσιμη που να μπορεί να αγοραστεί (οι 2 που μένουν

κοστίζουν 15 η κάθε μία και το διαθέσιμο κεφάλαιο είναι 10). Έτσι παίρνουμε ένα αποτέλεσμα συνολικού κέρδους (ή απόδοσης) 40 το οποίο δεν είναι το βέλτιστο. Το βέλτιστο είναι να επιλεγούν οι επενδύσεις με αύξοντα αριθμό 1 και 2 καλύπτοντας όλο το αρχικό κεφάλαιο και έχοντας βέλτιστη λύση συνολικής απόδοσης 50 (25+25). Η brute-force προσέγγιση του προβλήματος οδηγεί σε αυτή τη βέλτιστη λύση.

Μια διαφορετική προσέγγιση που έχει δοκιμαστεί είναι να αλλάξει το άπληστο κριτήριο επιλέγοντας από τις διαθέσιμες επενδύσεις εκείνη που έχει το μικρότερο κόστος και που μπορεί να αγοραστεί προφανώς . Αν εφαρμοστεί αυτό το κριτήριο στο παραπάνω παράδειγμα με τις 3 επενδύσεις, τότε θα επιλεχτούν πρώτα οι επενδύσεις 1 και 2 με ίσα κόστη 15 και οι 2 και έτσι θα έχουμε βέλτιστη λύση. Για να γίνει αυτό αλλάζουμε το τύπο ratio στο κώδικα να ισούται με το κόστος κάθε επένδυσης και η ταξινόμηση πλέον θα γίνεται με βάση το κόστος και όχι τη πυκνότητα κέρδους. Εναλλακτικά, για να έχουμε τη βέλτιστη απόδοση θα χρησιμοποιούσαμε την brute-force μέθοδο η οποία πάντα δίνει τη βέλτιστη λύση.

**Πίνακας 9. Αποτελέσματα μεθόδου «επιλογής πρώτα μικρότερου κόστους»**

| Α/Α επένδυσης που επιλέχτηκε | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Πλήθος επιλογών επένδυσης | Συνολικό κόστος επιλογής(εκ. ευρώ) | Συνολική απόδοση επιλογής(εκ. ευρώ) |
|---|---|---|---|---|---|
| 1 | 15 | 25 | 1 | 15 | 25 |
| 2 | 15 | 25 | 1 | 15 | 25 |
| Σύνολο | | | | | 50 |

# ΚΕΦΑΛΑΙΟ 4: Τα όρια του αλγορίθμου και συγκρίσεις

## 4.1. Τα όρια της μεθόδου brute force

Όπως έχουμε δει η brute force προσέγγιση έχει μεγάλο χρόνο εκτέλεσης για να δώσει αποτέλεσμα όταν οι περιπτώσεις (επενδύσεις) και το κεφάλαιο ξεπερνούν κάποιο όριο.

Στις περιπτώσεις που έχουμε δει μέχρι στιγμής, οι επενδύσεις είναι μέχρι 10 και πήραμε αποτελέσματα γρήγορα ακόμα και με την brute force προσέγγιση. Τώρα θα δοκιμάσουμε τα όρια της μεθόδου αυτής και στη συνέχεια θα την συγκρίνουμε με την άπληστη μέθοδο. Προφανώς η άπληστη μέθοδος θα δώσει άμεσα αποτελέσματα αλλά ενδεχομένως να μη δώσει τη βέλτιστη απόδοση.

Για να δούμε σε ποιο σημείο η μέθοδος αρχίζει και αυξάνει απαγορευτικά την πολυπλοκότητά της μπορούμε εύκολα να σκεφτούμε ότι όσο αυξάνονται οι επενδύσεις τόσο αυξάνονται και οι περιπτώσεις προς εξέταση από τη μέθοδο. Επιπλέον, αν αυξηθεί το αρχικό κεφάλαιο αυξάνονται και οι έλεγχοι για το ποιες επενδύσεις επιλέγονται αφού το διαθέσιμο υπόλοιπο κεφαλαίου θα είναι μεγαλύτερο και θα μπορούν να επιλεχτούν επιπλέον επενδύσεις. Έτσι υποθέτουμε ότι έχουμε 15 επενδύσεις με αρχικό κεφάλαιο 78 και κάθε επένδυση μπορεί να επιλέγεται μια φορά.

Πίνακας 10. 15 Επενδυτικές ευκαιρίες (όρια brute-force)

| Αύξων αριθμός επένδυσης (Α/Α) | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (σε εκ. ευρώ) | Αύξων αριθμός επένδυσης (Α/Α) | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (σε εκ. ευρώ) |
|---|---|---|---|---|---|
| 0 | 7.5 | 37.5 | 8 | 7.25 | 4.8 |
| 1 | 6.25 | 18.75 | 9 | 13 | 16.725 |
| 2 | 0.5 | 1 | 10 | 1.5 | 3 |
| 3 | 12.5 | 18.75 | 11 | 5 | 7.25 |
| 4 | 10.25 | 11.275 | 12 | 9.5 | 17 |
| 5 | 10.775 | 5.3875 | 13 | 8.325 | 11.5 |
| 6 | 12.25 | 1.225 | 14 | 7 | 21.225 |
| 7 | 10.25 | 4.225 | | | |

Τα αποτελέσματα, τα οποία τα πήραμε σε χρόνο περίπου 9 λεπτών, φαίνονται στο παρακάτω πίνακα (Πίνακας 11.) .

**Πίνακας 11. Αποτελέσματα μεθόδου brute-force (για 15 επενδύσεις)**

| Α/Α επένδυσης που επιλέχτηκε | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Πλήθος επιλογών επένδυσης | Συνολικό κόστος επιλογής(εκ. ευρώ) | Συνολική απόδοση επιλογής(εκ. ευρώ) |
|---|---|---|---|---|---|
| 0 | 7.5 | 37.5 | 1 | 7.5 | 37.5 |
| 1 | 6.25 | 18.75 | 1 | 6.25 | 18.75 |
| 2 | 0.5 | 1 | 1 | 0.5 | 1 |
| 3 | 12.5 | 18.75 | 1 | 12.5 | 18.75 |
| 4 | 10.25 | 11.275 | 1 | 10.25 | 11.275 |
| 9 | 13 | 16.725 | 1 | 13 | 16.725 |
| 10 | 1.5 | 3 | 1 | 1.5 | 3 |
| 12 | 9.5 | 17 | 1 | 9.5 | 17 |
| 13 | 8.325 | 11.5 | 1 | 8.325 | 11.5 |
| 14 | 7 | 21.225 | 1 | 7 | 21.225 |
| Σύνολο | | | | | 156.725 |

Από τα αποτελέσματα βλέπουμε ότι επιλέχτηκαν 10 επενδύσεις με συνολική απόδοση 156.725. Να σημειώσουμε ότι δεν χρησιμοποιήθηκαν 1.675 από το αρχικό κεφάλαιο (οι τιμές έχουμε υποθέσει ότι είναι σε εκατομμύρια ευρώ). Ο χρόνος εκτέλεσης μπορεί να χαρακτηριστεί μεγάλος και απαγορευτικός για έναν επενδυτή που θέλει αποτελέσματα σε άμεσο χρόνο και όσο προσθέτουμε επενδύσεις αυξάνεται σημαντικά.

## 4.2. Αποτελέσματα της άπληστης μεθόδου και σύγκριση με τη μέθοδο brute-force

Αφού είδαμε ότι η brute-force μέθοδος χρειάζεται μεγάλο χρονικό διάστημα για να δώσει αποτελέσματα όταν τα αντικείμενα και το αρχικό κεφάλαιο είναι μεγαλύτερα από έναν αριθμό, θα δοκιμάσουμε την άπληστη μέθοδο. Στην άπληστη μέθοδο περιμένουμε άμεσα αποτελέσματα αλλά πιθανόν όχι τη βέλτιστη απόδοση.

Αν δεν έχουμε τη βέλτιστη απόδοση θα θέλαμε να έχουμε μία απόδοση κοντά στη βέλτιστη. Η βέλτιστη απόδοση από τη προηγούμενη μέθοδο είναι 156.725.

**Πίνακας 12. Αποτελέσματα άπληστης μεθόδου (για 15 επενδύσεις)**

| Α/Α επένδυσης που επιλέχτηκε | Κόστος επένδυσης (σε εκ. ευρώ) | Απόδοση επένδυσης (κέρδος σε εκ. ευρώ) | Πλήθος επιλογών επένδυσης | Πυκνότητα κέρδους | Συνολική απόδοση επιλογής(εκ. ευρώ) |
|---|---|---|---|---|---|
| 0 | 7.5 | 37.5 | 1 | 5 | 37.5 |
| 14 | 7 | 21.225 | 1 | 3,03 | 21.225 |
| 1 | 6.25 | 18.75 | 1 | 3 | 18.75 |
| 10 | 1.5 | 3 | 1 | 2 | 3 |
| 2 | 0.5 | 1 | 1 | 2 | 1 |
| 12 | 9.5 | 17 | 1 | 1,79 | 17 |
| 3 | 12.5 | 18.75 | 1 | 1,5 | 18.75 |
| 11 | 5 | 7.25 | 1 | 1,45 | 7.25 |
| 13 | 8.325 | 11.5 | 1 | 1,38 | 11.5 |
| 9 | 13 | 16.725 | 1 | 1,28 | 16.725 |
| Σύνολο | | | | | 152.7 |

Στο πίνακα 12 βλέπουμε ότι επιλέχτηκαν 10 επενδύσεις με συνολική απόδοση 152.7. Αυτή η απόδοση δεν είναι βέλτιστη όμως είναι πολύ κοντά στη βέλτιστη απέχοντας μόνο 4.025 μονάδες. Αυτό είναι ικανοποιητικό για μία άπληστη μέθοδο που δίνει άμεσα αποτελέσματα γι' αυτά τα δεδομένα. Επίσης, έμειναν ανεκμετάλλευτα 6.925 εκ. ευρώ από το αρχικό κεφάλαιο.

Στις επιλεγμένες επενδύσεις της άπληστης μεθόδου παρατηρούμε ότι η σειρά επιλογής επενδύσεων είναι ταξινομημένη κατά φθίνουσα τιμή πυκνότητας κέρδους. Αυτό είναι λογικό αφού η άπληστη μέθοδος ταξινομεί τις επιλογές κατά φθίνουσα σειρά πυκνότητας κέρδους ($p_i/w_i$) και ξεκινά με τη σειρά να επιλέγει αυτές που μπορούν να αγοραστούν. Έτσι ξεκινώντας από αυτή με τον μεγαλύτερο λόγο απόδοσης προς κόστος επιλέγει με τη σειρά 10 επενδύσεις και φτάνοντας σε αυτή με α.α 9 σταματάει αφού οι επόμενες και δεν μπορούν να αγοραστούν αλλά η μέθοδος έχει προγραμματιστεί να σταματάει όταν κάποια επένδυση δεν μπορεί να

αποκτηθεί. Στο πίνακα που ακολουθεί παρουσιάζονται συγκεντρωτικά τα αποτελέσματα των 2 μεθόδων.

Πίνακας 13. Συγκεντρωτικά αποτελέσματα μεθόδων brute-force, greedy

| | Μέθοδος brute-force (seconds) | Άπληστη μέθοδος |
|---|---|---|
| Χρόνος εκτέλεσης | 533.8 | 0 |
| Αριθμός επιλογών | 10 | 10 |
| Ανεκμετάλλευτο κεφάλαιο | 1.675 | 6.925 |
| Συνολική απόδοση | 156.725 | 152.7 |

Πλήθος επενδύσεων: 15

Αρχικό κεφάλαιο: 78

## 4.3. Συμπεράσματα σύγκρισης

Όπως φαίνεται στο πίνακα 13 η διαφορά στο χρόνο εκτέλεσης είναι μεγάλη. Και οι 2 μέθοδοι όμως επέλεξαν 10 επενδύσεις. Η άπληστη μέθοδος ελέγχει το διαθέσιμο κεφάλαιο και σταματάει όταν μία επένδυση δεν είναι δυνατόν να αποκτηθεί με το υπόλοιπο των χρημάτων. Αυτό μπορεί να αλλάξει αλλάζοντας την εντολή “break” στον κώδικα της άπληστης μεθόδου σε “continue”. Έτσι θα συνεχίζει να ψάχνει στη λίστα με τις επενδύσεις μέχρι να βρει αυτή που μπορεί να επιλεχτεί. Για αυτό το λόγο σταμάτησε στις 10 επενδύσεις αφού δε γινόταν να επιλέξει άλλη απ’ τις διαθέσιμες. Ενώ η άπληστη μέθοδος δίνει άμεσα αποτελέσματα, η brute-force βλέπουμε ότι χρησιμοποιεί σχεδόν όλο το διαθέσιμο κεφάλαιο αφήνοντας μόνο 1.675 εκατομμύρια αχρησιμοποίητα.

Σχετικά με τους χρόνους εκτέλεσης προφανώς η άπληστη μέθοδος υπερνικά την brute-force και θα μπορούσε να προτιμηθεί σαν μία καλή μέθοδος προσέγγισης της βέλτιστης απόδοσης. Είναι αισιόδοξο το ότι μπορούμε να έχουμε άμεσα αποτελέσματα κοντά στη βέλτιστη λύση με μια τέτοια μέθοδο για ένα πρόβλημα

που είναι NP-hard. Αυτό μπορεί να φανεί στη διαφορά της συνολικής απόδοσης των 2 μεθόδων η οποία είναι πολύ μικρή. «Σ’ ένα πείραμα με 600 τυχαία παραδείγματα προβλήματος knapsack, η άπληστη μέθοδος πυκνότητας κέρδους έδωσε βέλτιστες λύσεις στα 239, ενώ και οι 600 λύσεις είχαν απόκλιση 25% από τη μέγιστη» [2].

Τέλος, μια μέθοδος brute-force θα ήταν χρήσιμη σε περιπτώσεις με λίγες επενδύσεις και προτιμότερη όταν οι επενδύσεις μπορούν να επιλεχτούν μόνο μια φορά η κάθε μία. Επίσης το αρχικό κεφάλαιο δεν θα πρέπει να είναι πολύ μεγάλο. Αυτό το είδαμε στο παράδειγμα με τις 15 επενδύσεις και κεφάλαιο 78. Σε αντίθετη περίπτωση με πολλές επενδύσεις και μεγάλο κεφάλαιο, θα χρησιμοποιούσαμε την άπληστη μέθοδο αφού δίνει γρήγορα  αποτελέσματα το οποίο είναι αναγκαίο στο σύγχρονο χρηματοοικονομικό περιβάλλον όπου τα πράγματα είναι ρευστά. Σε κάθε περίπτωση ο ενδιαφερόμενος επενδυτής που θα χρησιμοποιήσει αυτή τη μέθοδο καλείται να κρίνει ο ίδιος σύμφωνα με τα παραπάνω συμπεράσματα για το πώς θα την χρησιμοποιήσει.

# ΚΕΦΑΛΑΙΟ 5: Αντιμετώπιση του προβλήματος στο excel

## 5.1. Πρόβλημα μεγιστοποίησης

Μια διαφορετική προσέγγιση του προβλήματος knapsack είναι μέσω του προγράμματος excel. Στο excel ο χρήστης δεν γράφει κάποιο κώδικα αλλά υπάρχει έτοιμος κώδικας ενσωματωμένος ο οποίος λύνει το πρόβλημα.

Το πρόβλημα σάκου στην ουσία είναι ένα πρόβλημα μεγιστοποίησης και αυτό μπορεί να επιλυθεί με απλό γραμμικό προγραμματισμό. Αυτό που υποθέτουμε είναι ότι πρέπει να μεγιστοποιήσουμε την αξία μιας εξίσωσης με κάποιους περιορισμούς. Η εξίσωση που πρέπει να μεγιστοποιηθεί είναι αυτή του αθροίσματος των αποδόσεων των αντικειμένων που θα επιλεχτούν με περιορισμό το συνολικό τους κόστος να μη ξεπερνά κάποιο όριο, δηλαδή το αρχικό κεφάλαιο.

Πριν συνεχίσουμε πρέπει να λάβουμε υπόψη ότι στη περίπτωση αυτή θεωρούμε ότι στο πρόβλημα δεν μπορούμε να πάρουμε τμήματα των επενδύσεων (αντικειμένων) δηλαδή δεν μπορούμε να πάρουμε το 50% κάποιας επένδυσης αλλά μόνο ολόκληρη. Και αυτό είναι και ένα βασικό μειονέκτημα της μεθόδου αυτής που θα αναφερθεί και στη συνέχεια. Αυτό συμβαίνει γιατί σε ένα πρόβλημα γραμμικού προγραμματισμού με επενδύσεις όπου οι μεταβλητές παίρνουν τιμή 0 ή 1 πρέπει οι επιλογές των μεταβλητών (επενδύσεις) να είναι ακέραιοι. Με την άπληστη μέθοδο (greedy) που έχουμε αναφέρει θα μπορούσαμε να την τροποποιήσουμε ώστε να επιλέγονται τμήματα των επενδύσεων όταν αυτό είναι επιτρεπτό.

## 5.2. Παραδείγματα χρήσης στο excel

Για να δοκιμάσουμε ένα παράδειγμα στο excel θα χρησιμοποιήσουμε το excel 2007. Δημιουργούμε ένα νέο φύλλο εργασίας και εισάγουμε τις αξίες και αποδόσεις των επενδύσεων όπως φαίνεται παρακάτω. Θα δοκιμάσουμε να τρέξουμε το πρώτο παράδειγμα του πίνακα 1 για να δείξουμε ότι πήραμε ίδια αποτελέσματα. Το παράδειγμα φαίνεται στην παρακάτω εικόνα 1.

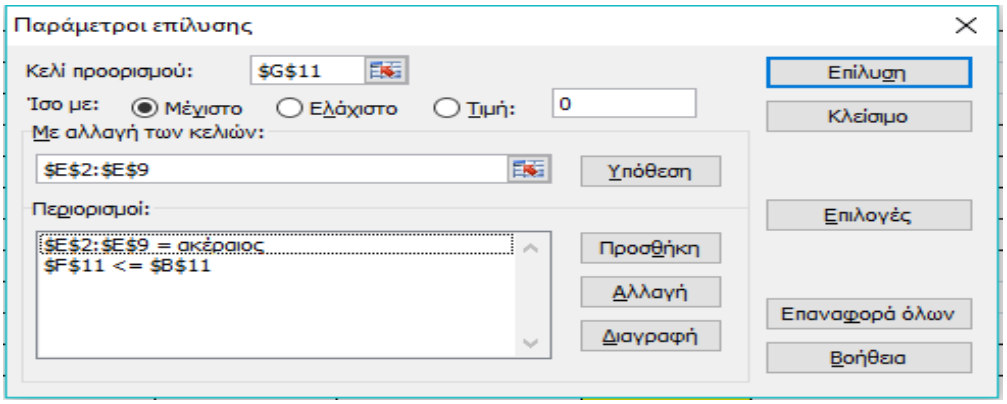
Εικόνα 1

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Επιλογές | Κόστος | Απόδοση | Ratio | Ποσότητα | Κόστος | Τελική Απόδση |
| 2 | 0 | 7,5 | 37,5 | 500,00% | 3 | 22,5 | 112,5 |
| 3 | 1 | 6,25 | 18,75 | 300,00% | 0 | 0 | 0 |
| 4 | 2 | 0,5 | 1 | 200,00% | 5 | 2,5 | 5 |
| 5 | 3 | 12,5 | 18,75 | 150,00% | 0 | 0 | 0 |
| 6 | 4 | 10,25 | 11,275 | 110,00% | 0 | 0 | 0 |
| 7 | 5 | 10,775 | 5,3875 | 50,00% | 0 | 0 | 0 |
| 8 | 6 | 12,25 | 1,225 | 10,00% | 0 | 0 | 0 |
| 9 | | | | | 0 | | |
| 10 | | | | | | | |
| 11 | Όριο | 25 | | | Total | 25 | 117,5 |

Βλέπουμε ότι η συνολική απόδοση (βέλτιστη) είναι 117.5 εκατομμύρια ευρώ και η επένδυση 0 επιλέχτηκε 3 φορές και η επένδυση 2 επιλέχτηκε 5 φορές όπως ακριβώς έχουμε δει στο πίνακα 2. Θα μπορούσαμε να επιλέξουμε κάθε επένδυση να επιλέγεται μία φορά.

Για να εκτελεστεί ένα τέτοιο πρόβλημα στο excel πάμε δεδομένα από τη γραμμή εργαλείων, μετά επίλυση και εκεί εισάγουμε τους περιορισμούς και τα κελιά που θέλουμε να αλλάξουν. Αυτό φαίνεται στην εικόνα 2.

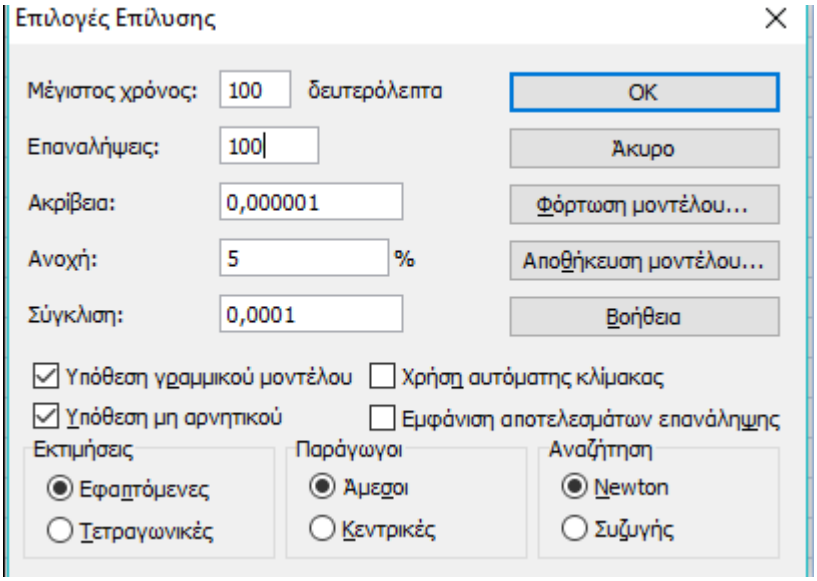
Εικόνα 2




Βλέπουμε ότι υπάρχει ο περιορισμός το κελί F11 να είναι μικρότερο ίσο από το κελί B11 που αυτό σημαίνει να μην ξεπεραστεί το αρχικό κεφάλαιο των 25 εκατομμυρίων ευρώ που έχουμε υποθέσει. Επίσης, τα κελιά που θα αλλάξουν είναι το E2 ως το E9 και πρέπει να παίρνουν ακέραιες τιμές. Για να εισάγουμε έναν νέο περιορισμό επιλέγουμε προσθήκη από την καρτέλα της εικόνας 2 και εισάγουμε έναν νέο. Τέλος, βλέπουμε ότι το κελί προορισμού G11 πρέπει να είναι μέγιστο το οποίο δίνει τη βέλτιστη τελική απόδοση.

Στις επιλογές επίλυσης έχουμε επιλέξει υπόθεση γραμμικού μοντέλου και υπόθεση μη αρνητικού .

Εικόνα 3




## 5.3. Συμπεράσματα και χρόνος εκτέλεσης

Ο χρόνος εκτέλεσης στο excel είναι άμεσος για όλα τα παραδείγματα που έχουμε δει μέχρι στιγμής. Ακόμα και για το παράδειγμα του πίνακα 11, πήραμε αποτελέσματα αμέσως σε αντίθεση με τα περίπου 9 λεπτά που χρειάστηκαν με τη μέθοδο brute-force. Θα αναρωτηθεί κανείς γιατί να μη χρησιμοποιήσει τη μέθοδο αυτή σε αντίθεση με τις άλλες. Η απάντηση είναι ότι μπορεί να τη χρησιμοποιήσει και η μέθοδος αυτή είναι μια καλή μέθοδος προσέγγισης του προβλήματος με τους περιορισμούς που αναφέραμε, όμως αν δοκιμάσουμε να τρέξουμε ένα παράδειγμα με πάνω από 200 επιλογές τότε η μέθοδος δεν δουλεύει και μας εμφανίζεται μήνυμα ότι έχουν επιλεγεί πολλά ρυθμιζόμενα κελιά.

Ένα κώδικας σε μια γλώσσα προγραμματισμού θα έτρεχε το πρόβλημα και για περισσότερες από 200 επιλογές. Ακόμα, αν τροποποιούσαμε έναν κώδικα θα μπορούσαμε να πάρουμε τμήματα των επιλογών μας, πράγμα αδύνατο μέσω της μεθόδου στο excel. Αυτό θα μπορούσε να γίνει στη greedy μέθοδο όπου σε κάθε βήμα αν δε γινόταν να επιλεχτεί μια επένδυση, επειδή το διαθέσιμο κεφάλαιο δεν επαρκούσε, θα επιλεγόταν τμήμα αυτής ώστε να καλύπτεται το κεφάλαιο. Αυτό προφανώς θα γινόταν πάντα μια φορά στο τελευταίο βήμα αφού αν μια επένδυση

δεν μπορεί να επιλεχτεί λόγω κόστους θα επιλεγόταν τμήμα αυτής δαπανώντας όλο το διαθέσιμο κεφάλαιο. Βέβαια και πάλι θα μπορούσε να προγραμματιστεί να επιλέγεται ένα τμήμα της επιλογής του χρήστη. Καταλήγοντας, το βασικό συμπέρασμα είναι ότι η μέθοδος αντιμετώπισης σαν πρόβλημα γραμμικού προγραμματισμού στο excel δεν είναι ευέλικτη σε αλλαγές.

Παρατηρούμε ότι όλες οι παραπάνω μέθοδοι έχουν μειονεκτήματα και πλεονεκτήματα. Εύκολα συμπεραίνει κάποιος ότι μια μέθοδος γρήγορη και χωρίς απαγορευτικούς περιορισμούς θα ήταν ιδανική για ένα τέτοιο πρόβλημα. Εδώ εισάγονται οι γενετικοί αλγόριθμοι οι οποίοι έχουν το χαρακτηριστικό ότι σχετικά πολύ γρήγορα μπορούν να εντοπίσουν μια βέλτιστη ή μια σχεδόν βέλτιστη λύση (πολύ κοντά στη βέλτιστη) ακόμα και για μεγάλο αριθμό επιλογών. Η σύγχρονη έρευνα εστιάζει και σε τέτοιους αλγορίθμους οι οποίοι παρουσιάζουν μεγάλο ενδιαφέρον. Ένα τέτοιο παράδειγμα αλγορίθμου για το πρόβλημα του σάκου παρουσιάζεται στο επόμενο κεφάλαιο.

# ΚΕΦΑΛΑΙΟ 6: Επίλυση με γενετικό αλγόριθμο

## 6.1. Εισαγωγή στους γενετικούς αλγορίθμους

Οι γενετικοί αλγόριθμοι είναι μια τεχνική προγραμματισμού η οποία μιμείται την εξέλιξη της φύσης όπως καταγράφεται στη θεωρία του Δαρβίνου και την εισήγαγε στα τέλη της δεκαετίας του 1960 ο Τζον Χόλαντ, ερευνητής του Ινστιτούτου της Σάντα Φε (ΗΠΑ) [5]. Πιο συγκεκριμένα, σύμφωνα με τη θεωρία της εξέλιξης στη φύση ισχύει η θεωρία της επιβίωσης του ισχυρότερου (survival of the fittest). Αυτό σημαίνει ότι οι καλύτεροι οργανισμοί επιβιώνουν έναντι των πιο αδύναμων. Με άλλα λόγια, οι πιο αδύναμοι οργανισμοί πεθαίνουν λόγω του ότι δε μπορούν να επιβιώσουν στο περιβάλλον και τη θέση τους παίρνουν άλλοι οργανισμοί καλύτεροι από αυτούς.

Η λειτουργία των γενετικών αλγορίθμων (εφεξής γ.α) χρησιμοποιεί την θεωρία της εξέλιξης η οποία περιλαμβάνει 3 χαρακτηριστικά, την διασταύρωση, την μετάλλαξη και τη φυσική επιλογή [5]. Ένας απλός γ.α αρχικά δημιουργεί τυχαία ένα πλήθος οργανισμών έστω Ν. Οι Ν αυτοί οργανισμοί μπορεί να είναι και πιθανές λύσεις του προβλήματος που μας ενδιαφέρει. Πριν όμως δημιουργηθεί αυτό το πλήθος πρέπει να έχει γίνει αναπαράσταση των λύσεων αυτών. Η αναπαράσταση αυτών των λύσεων γίνεται με μια σειρά από δυαδικά ψηφία (bits) 0 ή 1 ή γενικότερα με μια σειρά χαρακτήρων. Κάθε τέτοια λύση αποτελεί και μια μεταβλητή ή πιθανή λύση του προβλήματος και μιμείται το γενετικό κώδικα (γονιδίωμα) των ζωντανών οργανισμών. Μετά την δημιουργία του αρχικού πληθυσμού ακολουθεί η φάση της διασταύρωσης (crossover) όπου τα στοιχεία του αρχικού πληθυσμού διασταυρώνονται μεταξύ τους και παράγουν νέο πληθυσμό (παιδιά/offsprings). Στη συνέχεια γίνονται οι μεταλλάξεις στα ψηφία των παιδιών όπου κάθε ψηφίο (0 ή 1 ή χαρακτήρας) αλλάζει με μια μικρή πιθανότητα. Για παράδειγμα αν ένα παιδί είναι το (0001110) και γίνει μετάλλαξη (mutation) στο πρώτο ψηφίο του, τότε το πρώτο μηδέν θα γίνει 1 και το μεταλλαγμένο πλέον παιδί θα είναι το (1001110). Τέλος, τα παιδιά που είναι καλύτερα από τους γονείς, τους αντικαθιστούν και έχουμε έναν νέο πληθυσμό από Ν άτομα.

Ο έλεγχος για το αν ένα στοιχείο (μια σειρά από bits) είναι καλύτερο από ένα άλλο γίνεται με κάποιο κριτήριο. Αυτό το κριτήριο το καθορίζει ο χρήστης ανάλογα με το πρόβλημα. Για παράδειγμα αυτό το κριτήριο θα μπορούσε να είναι διάλεξε το στοιχείο εκείνο το οποίο έχει μεγαλύτερη απόδοση ή διάλεξε αυτό με το μικρότερο κόστος. Κάθε στοιχείο έχει κάποια χαρακτηριστικά τα οποία μπορεί να είναι κόστος, απόδοση, μήκος τα οποία χρησιμοποιούνται για να συγκριθούν μεταξύ τους τα στοιχεία του πληθυσμού.

## 6.2. Εφαρμογή γενετικού αλγορίθμου στο πρόβλημα σάκου και ένα υποθετικό παράδειγμα

Για να γίνει πιο κατανοητό πως λειτουργεί ένας γ.α θα δείξουμε βήμα-βήμα πως συνδέεται και πως λειτουργεί με το πρόβλημα του σάκου.

Αρχικά, πρέπει να γίνει δυαδική αναπαράσταση των πιθανών λύσεων. Επιλέγουμε να προσεγγίσουμε το δυαδικό πρόβλημα όπου κάθε επιλογή γίνεται έως 1 φορά (0-1 knapsack) γι' αυτό και γίνεται δυαδική αναπαράσταση. Με μια παραλλαγή θα μπορούσε να γίνεται και επιλογή κάθε αντικειμένου πάνω από μια φορά. Ας υποθέσουμε ότι έχουμε τις 6 επενδύσεις investments=(0,1,2,3,4,5) με αντίστοιχα κόστη weights=(7,3,8,3,9,11) και αποδόσεις values=(3,5,4,9,10,6). Το αρχικό κεφάλαιο υποθέτουμε ότι είναι 20. Μια πιθανή λύση (όχι βέλτιστη απαραίτητα) θα μπορούσε να είναι να επιλέξουμε τις επενδύσεις 0,1,4 με συνολικό κόστος 19 και συνολική απόδοση 18. Μια δεύτερη πιθανή λύση θα μπορούσε να ήταν οι επενδύσεις 4, 5 με συνολικό κόστος 20 και συνολική απόδοση 16. Η πρώτη λύση είναι προφανώς καλύτερη από τη δεύτερη αφού έχει μεγαλύτερη απόδοση. Με αυτό τον (τυχαίο) τρόπο μπορούμε να παράγουμε διάφορες πιθανές λύσεις. Για την πρώτη περίπτωση, αν αναπαραστήσουμε δυαδικά τη λύση ως ένα στοιχείο, θα ήταν το στοιχείο (1,1,0,0,1,0) όπου με 1 δηλώνουμε πως η επένδυση έχει επιλεχτεί και με 0 όχι. Έτσι έχουμε το πρώτο στοιχείο το πληθυσμού που είναι το (1,1,0,0,1,0). Με την ίδια λογική το δεύτερο θα είναι το (0,0,0,0,1,1) με επιλεγμένες επενδύσεις τις 4, 5. Η αρχική παραγωγή αυτών των πιθανών λύσεων μπορεί να γίνει τυχαία όπου σε κάθε θέση από τις 6 του στοιχείου θα βρίσκεται ένα μηδέν ή ένας άσσος. Η τυχαία αναπαραγωγή αριθμών σε μία γλώσσα προγραμματισμού όπως η C γίνεται

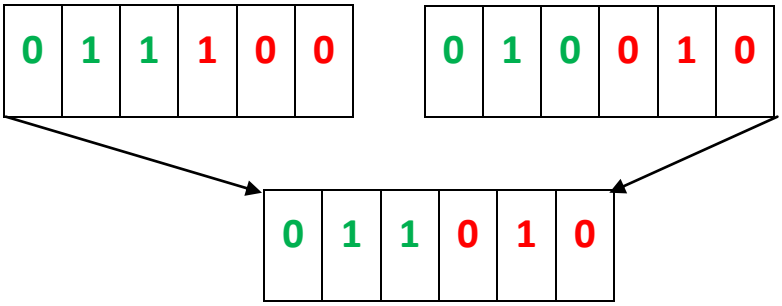
εύκολα με έτοιμες συναρτήσεις. Σε αυτό το σημείο και κατά τη διάρκεια της αρχικοποίησης του πληθυσμού μπορούμε να εισάγουμε ένα κριτήριο που να αποτρέπει τη δημιουργία μη αποδεκτών λύσεων, όπως στοιχείο που να υπερβαίνει το αρχικό κεφάλαιο (π.χ το 1,1,1,1,1,1) ή μηδενικό στοιχείο (0,0,0,0,0,0,). Περισσότερες λεπτομέρειες θα παρουσιαστούν στην υλοποίηση και επεξήγηση του κώδικα σε C στη συνέχεια. Ύστερα από την αρχικοποίηση του πληθυσμού ας υποθέσουμε ότι θα έχουμε τα 4 στοιχεία του παρακάτω πίνακα.

Πίνακας 14. Αρχικός πληθυσμός 4 στοιχείων

| Α.Α Στοιχείου | Δυαδική αναπαράσταση στοιχείου | Συνολικό κόστος στοιχείου | Συνολική αξία στοιχείου |
|---|---|---|---|
| 0 | (0,1,1,1,0,0) | 14 | 18 |
| 1 | (0,1,0,0,1,0) | 12 | 15 |
| 2 | (0,1,1,0,0,0) | 11 | 9 |
| 3 | (1,0,0,1,1,0) | 20 | 16 |
| | Σύνολο πλήθους | 57 | 58 |

Το επόμενο βήμα είναι η διασταύρωση (crossover) των στοιχείων αυτών μεταξύ τους και η παραγωγή παιδιών (offsprings). Θεωρούμε ότι διασταυρώνονται με τη σειρά οι γονείς (ανά ζεύγη) ως εξής, ο γονέας με α.α 0 και ο γονέας με α.α 1 διασταυρώνονται και παράγουν το παιδί (0,1,1,0,1,0) του οποίου τα πρώτα 3 bits είναι αυτά του γονέα 0 και τα υπόλοιπα 3 του γονέα 1. Δηλαδή σε μία διασταύρωση επιλέγουμε το παιδί που παράγεται, να αποτελείται από το πρώτο μισό (πρώτα 3 bits) του πρώτου γονέα της διασταύρωσης και από το δεύτερο μισό του δεύτερου γονέα της ίδιας διασταύρωσης . Στο παρακάτω σχήμα αναπαρίσταται η παραπάνω διασταύρωση.

Σχήμα 1. Διασταύρωση και παραγωγή παιδιού



Το παιδί που παράχθηκε από τη πρώτη διασταύρωση είναι το (0,1,1,0,1,0) και με τον ίδιο τρόπο από τη δεύτερη διασταύρωση μεταξύ των γονιών 2 και 3 παράγεται το παιδί (0,1,1,1,1,0). Παρατηρούμε ότι το πρώτο παιδί (0,1,1,0,1,0) έχει συνολικό κόστος 20 και συνολική απόδοση 19 ενώ το δεύτερο παιδί 23 και 28 αντίστοιχα. Το δεύτερο παιδί είναι μια μη αποδεκτή λύση επειδή ξεπέρασε το αρχικό κεφάλαιο που είναι 20 (23>20). Σε αυτό το στάδιο περνάμε στη φάση της μετάλλαξης παρόλο που κάποιο στοιχείο αποτελεί μια μη αποδεκτή λύση. Στη φάση της μετάλλαξης κάποιο bit από τα bits του δεύτερου παιδιού πιθανόν να αλλάξει και έτσι να αλλάξει το συνολικό του κόστος και να γίνει αποδεκτή λύση.

Στη φάση της μετάλλαξης υποθέτουμε ότι δεν τυχαίνει να γίνεται κάποια μετάλλαξη στα 2 παιδιά που έχουν παραχθεί. Έτσι το δεύτερο παιδί απορρίπτεται και φτάνουμε στη φάση της επιλογής του καλύτερου (survival of the fittest).

Στο στάδιο της επιλογής και αντικατάστασης συγκρίνουμε το πρώτο παιδί, αφού μόνο αυτό έχει απομείνει, με όλους τους γονείς. Παρατηρούμε ότι το πρώτο παιδί μπορεί να αντικαταστήσει τον γονέα 2 αφού είναι αυτός με τη μικρότερη απόδοση. Έτσι ο νέος πληθυσμός είναι:

**Πίνακας 15. Πληθυσμός 4 στοιχείων μετά τη πρώτη διασταύρωση**

| Α.Α Στοιχείου | Δυαδική αναπαράσταση στοιχείου | Συνολικό κόστος στοιχείου | Συνολική αξία στοιχείου |
|---|---|---|---|
| 0 | (0,1,1,1,0,0) | 14 | 18 |
| 1 | (0,1,0,0,1,0) | 12 | 15 |
| 2 | (0,1,1,0,1,0) | 20 | 19 |
| 3 | (1,0,0,1,1,0) | 20 | 16 |
| | Σύνολο | 66 | 68 |

Αν συγκρίνουμε τους 2 πληθυσμούς του πίνακα 14 και πίνακα 15, παρατηρούμε ότι το πλήθος του πίνακα 15 έχει καλύτερη συνολική απόδοση 68 έναντι 58 του πρώτου πλήθους. Αυτό συμβαίνει γιατί σε κάθε επανάληψη (διασταύρωση) επιλέγεται πάντα ο καλύτερος να αντικαταστήσει τον χειρότερο. Με αυτό τον τρόπο δε γίνεται ένας μελλοντικός ή νεότερος πληθυσμός, να είναι χειρότερος από έναν παρελθοντικό ή παλαιότερο. Επιπλέον, τα στοιχεία κάθε πληθυσμού θα είναι καλύτερα ή τουλάχιστον ίδια με τα στοιχεία των παλαιότερων από αυτόν

πληθυσμών. Σε αυτό το σημείο κάποιος μπορεί να αντιπαρατεθεί λέγοντας ότι δεν μπορούμε να συγκρίνουμε ποιο πλήθος είναι καλύτερο ή χειρότερο αφού παίζει ρόλο και το συνολικό κόστος το οποίο στο πρώτο πληθυσμό είναι μικρότερο από του δεύτερου. Η απάντηση είναι ναι παίζει ρόλο και το συνολικό κόστος, αλλά έχουμε υποθέσει ότι θεωρούμε καλύτερο το πλήθος ή το στοιχείο εκείνο με τη μεγαλύτερη απόδοση. Κάποια διαφορετική προσέγγιση θα ήταν να θεωρήσουμε καλύτερο το στοιχείο εκείνο με το μικρότερο κόστος.

Το παραπάνω πρόβλημα της επιλογής του καλύτερου μπορεί να λυθεί συγκρίνοντας τα στοιχεία αλλά και τα πλήθη ως προς το συνολικό τους λόγο απόδοσης προς κόστος. Με αυτό τον τρόπο θα επιλέγαμε εκείνα τα στοιχεία με μικρότερο κόστος αλλά και μεγαλύτερη απόδοση. Αυτό θα μπορούσε να γίνει στη C με μια συνθήκη “if” όπου εκεί θα επιλεγόταν το στοιχείο με το μεγαλύτερο λόγο απόδοσης/κόστους αλλά και τη μεγαλύτερη απόδοση και το μικρότερο κόστος. Σίγουρα μέσα σε όλες τις συνθήκες θα πρέπει να υπάρχουν και οι συνθήκες που αποτρέπουν την επιλογή μη αποδεκτών λύσεων. Σε κάθε περίπτωση ο χρήστης είναι ελεύθερος να προσεγγίσει το πρόβλημα με τον δικό του τρόπο και ανάλογα πάντα με τις απαιτήσεις του.

Μετά τη πρώτη διασταύρωση εκτελούνται μια σειρά από διασταυρώσεις και ο αλγόριθμος τερματίζεται μετά από έναν αριθμό επαναλήψεων ή όταν έχει βρεθεί μια επιθυμητή λύση ή όταν ικανοποιηθεί κάποιο άλλο κριτήριο που ορίζει ο χρήστης. Στο υποθετικό παράδειγμα και μόλις στη πρώτη διασταύρωση η καλύτερη μέχρι στιγμής λύση είναι το στοιχείο 2 με απόδοση 19, δηλαδή το παιδί της πρώτης διασταύρωσης που αντικατέστησε το αρχικό στοιχείο (0,1,1,0,0,0). Η βέλτιστη λύση του παραδείγματος μπορεί να υπολογιστεί με την τεχνική brute-force και είναι το στοιχείο (0,1,0,1,1,0) με απόδοση 24 και κόστος 15. Παρατηρούμε ότι η απόδοση 19 της μέχρι στιγμής καλύτερης λύσης, δηλαδή της (0,1,1,0,1,0), είναι πολύ κοντά στη βέλτιστη 24. Ύστερα από έναν αριθμό επαναλήψεων μπορεί να επιτευχθεί η βέλτιστη απόδοση και αν όχι η βέλτιστη, σίγουρα μια απόδοση πολύ κοντά στη βέλτιστη.

## 6.3. Παρουσίαση γενετικού αλγορίθμου στη γλώσσα C

Σε αυτό το κεφάλαιο θα παρουσιάσουμε έναν απλό γ.α στη γλώσσα C τον οποίο θα εφαρμόσουμε σε υποθετικά παραδείγματα καθώς επίσης και σε παραδείγματα που έχουμε ήδη παρουσιάσει.

Η μορφή ενός απλού γ.α μπορεί να περιέχει τις παρακάτω διαδικασίες:

1. Αρχικοποίηση πληθυσμού. Σε αυτή τη διαδικασία η οποία είναι και η πρώτη διαδικασία που υλοποιείται, δημιουργείται τυχαία ο αρχικός πληθυσμός ο οποίος αποτελείται από άτομα (ή χρωμοσώματα) τα οποία αποτελούνται από δυαδικές σειρές (σειρές από 1 και 0). Ένα άτομο θα μπορούσε να είναι το (100100) το οποίο αποτελείται από 6 δυαδικά ψηφία.
2. Διασταύρωση. Σε αυτή τη διαδικασία γίνεται το ταίριασμα των ατόμων μεταξύ τους (γονέων) και η αναπαραγωγή νέων στοιχείων (παιδιών). Από τη διαδικασία 4 ελέγχεται το καλύτερο παιδί το οποίο θα αντικαταστήσει τον χειρότερο γονέα.
3. Μετάλλαξη. Σε αυτή τη διαδικασία με μια μικρή πιθανότητα (π.χ 1%) κάθε ψηφίο (bit) κάθε παιδιού αλλάζει από 0 σε 1 ή από 1 σε 0.
4. Επιλογή του καλύτερου. Σε αυτή τη διαδικασία επιλέγεται το καλύτερο παιδί και αντικαθιστά τον χειρότερο γονέα σύμφωνα με τη συνάρτηση καταλληλότητας η οποία ελέγχει ποιο στοιχείο θα επιβιώσει έναντι του άλλου.

Οι παραπάνω διαδικασίες μπορούν να υλοποιηθούν σε μια γλώσσα προγραμματισμού όπως η C. Δεν είναι απαραίτητο κάθε μία διαδικασία να υλοποιείται ξεχωριστά αλλά μπορούν να υλοποιηθούν 2 διαδικασίες η μία μέσα στην άλλη. Αυτό εναπόκειται στο προγραμματιστικό στυλ κάθε προγραμματιστή. Για παράδειγμα η διαδικασία της διασταύρωσης μπορεί να εμπεριέχει και τη διαδικασία της μετάλλαξης και επιπλέον τη διαδικασία της επιλογής.

Ένας απλός γ.α που περιέχει τις παραπάνω διαδικασίες λειτουργεί με τα παρακάτω βήματα:

1. Αρχικοποίηση.
2. Διασταύρωση (γονέων).

3. Μετάλλαξη (παιδιών).
4. Επιλογή του καλύτερου.
5. Επιστροφή στο βήμα 2.

Η διαδικασία τερματίζεται όταν ικανοποιηθεί κάποιο κριτήριο ή όταν ολοκληρωθεί ένας αριθμός επαναλήψεων που ορίζει ο χρήστης. Όσες περισσότερες επαναλήψεις και όσο μεγαλύτερος ο αριθμός των ατόμων του αρχικού πληθυσμού τόσο περισσότερες πιθανότητες θα έχουμε να βρούμε μια καλή αν όχι βέλτιστη λύση. Βέβαια, όσο αυξάνεται ο αριθμός των επαναλήψεων και των ατόμων είναι αναμενόμενο ο γ.α να γίνεται πιο αργός γιατί θα πρέπει να γίνονται παραπάνω έλεγχοι και επαναλήψεις. Τέλος, σε κάθε εκτέλεση του γ.α δεν είναι σίγουρο ότι θα έχουμε καλύτερα αποτελέσματα από την προηγούμενη εκτέλεση. Αν για παράδειγμα σε μια εκτέλεση βρεθεί η βέλτιστη λύση δε σημαίνει απαραίτητα ότι στην επόμενη εκτέλεση θα ξαναβρεθεί η βέλτιστη. Αυτό συμβαίνει γιατί όπως έχουμε αναφέρει ο γ.α εμπεριέχει τυχαιότητα (τυχαία αρχικοποίηση, μικρή πιθανότητα μετάλλαξης) και τα αποτελέσματα εξαρτώνται από το κριτήριο καταλληλότητας των πιθανών λύσεων (survival of the fittest). Σίγουρα αυτό θα μπορούσε να αντιμετωπιστεί με διάφορους τρόπους. Ένας τρόπος θα ήταν, επειδή κάθε εκτέλεση είναι ανεξάρτητη από τις προηγούμενες, να αποθηκεύεται το μέχρι στιγμής καλύτερο αποτέλεσμα (π.χ σε μια βάση δεδομένων) και στις επόμενες εκτελέσεις να συγκρινόντουσαν οι πιθανές λύσεις με αυτό το αποτέλεσμα και να εμφανιζόταν, αν όχι καλύτερο αποτέλεσμα, τουλάχιστον ίδιο με το μέχρι στιγμής καλύτερο.

Το σχολιασμένο πρόγραμμα που ακολουθεί είναι η υλοποίηση ενός απλού γ.α για το πρόβλημα της βέλτιστης επένδυσης με τον αλγόριθμο 0-1 knapsack στη γλώσσα C.

```
#include<stdio.h>

#include <stdlib.h>

#include <float.h>

#include <limits.h>

#include <time.h>
```

```
int main(void){
int i,j;
float space=60;
int N=18;//arithmos bits kai arithmos items
int M=30;//arxiko plithos gonidiwn
int paidia=M/2;
 srand ( time(NULL) );
//items//
struct item
      {
      float size;
      float val;
      };
      //Πίνακας τεσσάρων items//
struct item items[N];
//Δήλωση των 18 αντικειμένων//
items[0].size=7.5; items[1].size=4.5; items[2].size=2.7;
items[0].val=13;   items[1].val=9;    items[2].val=6;
items[3].size=1.5; items[4].size=10; items[5].size=8;
items[3].val=4;    items[4].val=12;  items[5].val=12;
items[6].size=4; items[7].size=3.5; items[8].size=6;
items[6].val=8;  items[7].val=7.35; items[8].val=15;
items[9].size=9; items[10].size=9; items[11].size=6;
items[9].val=14; items[10].val=12; items[11].val=9;
items[12].size=2; items[13].size=3; items[14].size=7;
items[12].val=8;  items[13].val=9;  items[14].val=15;
```

```
items[15].size=8; items[16].size=4; items[17].size=2.45;
items[15].val=12; items[16].val=10; items[17].val=10.4;
//-----------------------------------------------------------------
-//
//Η δομή γονίδιο (gonidio)//
struct gonidio
      {
      int matrix[N];//Πίνακας Ν θέσεων, όσων και τα items αφού κάθε
//τιμή 0-1 αντιπροσωπεύει ένα item//
      float SolWeight;
      float SolVal;
      float SolFitness;
      int changed;
      };
      //Αρχικός πληθυσμός //
      struct gonidio arx_plithos[M];
//Τυχαίο γέμισμα πίνακα//
float current_weight=0;
float total_weight=0;
l: for(i=0;i<M;i++){s: for(j=0;j<N;j++){
      arx_plithos[i].matrix[j]=rand()%2;//Δίνουμε τυχαία τιμή 0 ή 1 σε
//κάθε θέση του πίνακα matrix[j]//
current_weight=(current_weight+ (arx_plithos[i].matrix[j])*(items[j].size));
      if (current_weight>space){
//Συνθήκη για να μην ξεπεράσω το space, //εάν το ξεπεράσω μηδενίζω την
//θέση του πίνακα εκείνη τη στιγμή και όλες τις επόμενες, αφού δε
//χωράει άλλο item.//
            for(int k=0;k+j<N;k++){
```

```
            arx_plithos[i].matrix[k+j]=0;}
            current_weight=0;goto l;
      }
}
//Συνθήκη μη μηδενικού γονιδίου//
if(current_weight==0){goto s;
}
current_weight=0;//Μηδενισμός του current_weight αφού στο επόμενο τρέξιμο πρέπει να είναι 0//
}
//Επιλογή//
//Αρχικά υπολογίζουμε τη συνολική απόδοση και συνολικό βάρος κάθε //γονιδίου//
//Αρχικοποίηση (=0) στοιχείων γονιδίου//
for(i=0;i<M;i++){
arx_plithos[i].SolVal=0;
arx_plithos[i].SolWeight=0;
arx_plithos[i].SolFitness=0;
}
float Total_Fitness_Plithismoy=0;
//Υπολογισμός τιμών//
void ypologismos(){

for(i=0;i<M;i++){
for(j=0;j<N;j++){
```

```
arx_plithos[i].SolWeight=          arx_plithos[i].SolWeight          +
(arx_plithos[i].matrix[j])*(items[j].size);
arx_plithos[i].SolVal=             arx_plithos[i].SolVal             +
(arx_plithos[i].matrix[j])*(items[j].val);
}
arx_plithos[i].SolFitness=arx_plithos[i].SolVal/arx_plithos[i].SolWeight;
Total_Fitness_Plithismoy=Total_Fitness_Plithismoy+arx_plithos[i].SolFitness;
}}
//----------------------------------------//
//Προτιμούμε να έχουμε άρτιο αριθμό πληθυσμού, για να παράγονται και
άρτιου πλήθους παιδιά//
//Διασταύρωση (Crossover) ανά 2.//
struct gonidio offsprings[paidia];
void crossover(){
int r=1;
int e=0;
      for(i=0;i<M/2;i++){for(j=0;j<N;j++){

      if(j<M/2){offsprings[i].matrix[j]=arx_plithos[i+e].matrix[j];}
      if(j>=M/2){offsprings[i].matrix[j]=arx_plithos[i+r].matrix[j];}

      }
      e++;
      r++;
      }

//Μετάλλαξη (Mutation)//
```

```
float x ;
for(i=0;i<paidia;i++){for(j=0;j<N;j++){
x= (float)rand()/(float)(RAND_MAX/1);
if(x<0.01){
      if(offsprings[i].matrix[j]==0){offsprings[i].matrix[j]=1;
 }
      else offsprings[i].matrix[j]=0;
      printf("\n");printf("Egine mutation");
      }
}
}
//Ypologismos SolFitness, SolWeight, SolVal paidiwn//
for(i=0;i<paidia;i++){
offsprings[i].SolVal=0;
offsprings[i].SolWeight=0;
offsprings[i].SolFitness=0;
}
for(i=0;i<paidia;i++){for(j=0;j<N;j++){
offsprings[i].SolWeight=offsprings[i].SolWeight+(offsprings[i].matrix[j
])*(items[j].size);
offsprings[i].SolVal=              offsprings[i].SolVal              +
(offsprings[i].matrix[j])*(items[j].val);
}
offsprings[i].SolFitness=offsprings[i].SolVal/offsprings[i].SolWeight;
}
//Έλεγχος αν θα κρατήσουμε τα παιδιά//
float max;
```

```
float temp=0;
int t1;int t2;
for(i=0;i<paidia;i++){
temp=0;t1=-2;t2=-2;
for(j=0;j<M;j++){
max=offsprings[i].SolFitness-arx_plithos[j].SolFitness;
if(offsprings[i].SolVal>arx_plithos[j].SolVal&&max>0&&max>temp&&offspri
ngs[i].SolWeight>0&&offsprings[i].SolWeight<=space&&arx_plithos[j].chan
ged!=1){
temp=max;
t1=i;t2=j;
}
if(t2!=-2){
      arx_plithos[t2].changed=1;
}
}//---------------------------------Αλλάζουν τα παρακάτω--------------
-----//
if(t1!=-2&&t2!=-2){
printf("\n");printf("To paidi ");printf("%d",t1);printf(" Antikathista
apo to arxiko plithos to  ");printf("%d",t2);printf("\n");
arx_plithos[t2].SolWeight=offsprings[t1].SolWeight;
arx_plithos[t2].SolVal=offsprings[t1].SolVal;
arx_plithos[t2].SolFitness=offsprings[t1].SolFitness;
for (int help=0;help<N;help++){
      arx_plithos[t2].matrix[help]=offsprings[t1].matrix[help];
      }
}
```

```
}

for(i=0;i<M;i++){
arx_plithos[i].changed=0;
}
//Εκτύπωση παιδιών//
printf("\n");
for(i=0;i<M/2;i++){for(j=0;j<N;j++){
printf("%d",offsprings[i].matrix[j]);
}
printf("\t");printf("SolFitness=");printf("%f",offsprings[i].SolFitness
);
printf("\t");printf("SolWeight=");printf("%f",offsprings[i].SolWeight);
printf("\n");
}
}
//Εκτύπωση (αρχικού πλήθους)//
void print_plithos(){
for(i=0;i<M;i++){for(j=0;j<N;j++){
printf("%d",arx_plithos[i].matrix[j]);
}
printf("\t");printf("SolFitness=");printf("%f",arx_plithos[i].SolFitnes
s);
printf("\t");printf("SolWeight=");printf("%f",arx_plithos[i].SolWeight)
;
printf("\t");printf("SolVal=");printf("%f",arx_plithos[i].SolVal);
printf("\n");}
```

```
printf("Total_Fitness_Plithismoy=");printf("%f",Total_Fitness_Plithismoy);
}
int helper;
ypologismos();
print_plithos();
float Best_Current_Val=0;
for(int h4=0;h4<1000;h4++){//<-----επαναλήψεις (iterations)//
printf("---------------Iteration %d---------------",h4);
crossover();
for(int g4=0;g4<M;g4++){
      if(arx_plithos[g4].SolVal>140){
      goto tr;     }
}
for(int g5=0;g5<M;g5++){
      if(arx_plithos[g5].SolVal>Best_Current_Val){
      Best_Current_Val=arx_plithos[g5].SolVal;
      helper=g5;
      }
}

}

tr: printf("-------------");printf("\n");
 print_plithos();printf("\n");
 printf("Best_Current_Val=%f",Best_Current_Val);printf("\n");
 printf("Gonidio No: %d",helper);printf("\n");
```

```
 printf("Selected investments:");
 for(i=0;i<N;i++){
      if(arx_plithos[helper].matrix[i]==1){
            printf("%d",i);printf(" , ");
       }
 }
 printf("\n");     printf("Remaining Space: ");
printf("%f",space-arx_plithos[helper].SolWeight);
}
```

## 6.4. Αποτελέσματα του γενετικού αλγορίθμου και μελλοντικές επεκτάσεις

Τα δεδομένα που έχουμε επιλέξει για τον γ.α είναι 18 επενδύσεις (items) τα οποία έχουν μια αξία και μία απόδοση όπως φαίνονται στον παρακάτω πίνακα.

Πίνακας 16. Επενδύσεις γενετικού αλγορίθμου

| Αύξων αριθμός επένδυσης (A/A) | Κόστος επένδυσης | Απόδοση επένδυσης |
|---|---|---|
| 0 | 7.5 | 13 |
| 1 | 4.5 | 9 |
| 2 | 2.7 | 6 |
| 3 | 1.5 | 4 |
| 4 | 10 | 12 |
| 5 | 8 | 12 |
| 6 | 4 | 8 |
| 7 | 3.5 | 7.35 |
| 8 | 6 | 15 |
| 9 | 9 | 14 |
| 10 | 9 | 12 |
| 11 | 6 | 9 |
| 12 | 2 | 8 |
| 13 | 3 | 9 |
| 14 | 7 | 15 |
| 15 | 8 | 12 |
| 16 | 4 | 10 |
| 17 | 2.45 | 10.4 |

Το αρχικό κεφάλαιο έχει οριστεί στις 60 χρηματικές μονάδες το οποίο ορίζεται από τη μεταβλητή “space”. Το αρχικό πλήθος γονιδίων είναι 20 με πιθανότητα μετάλλαξης σε κάθε επανάληψη (γενιά) 1%. Οι επαναλήψεις (γενιές) έχουν οριστεί σε 1000. Ακολουθούν τα αποτελέσματα του γ.α.

Πίνακας 17. Αποτελέσματα γ.α

| | | |
|---|---|---|
| Επιλεγμένες επενδύσεις | 0,2,3,5,6,7,8,12,13,14,15,16,17 | |
| Εναπομείναν κεφάλαιο | 0,349 | |
| Χρόνος εκτέλεσης | 11.67 sec | |
| Συνολική απόδοση | | 129.75 |

Εκτελώντας με τις ίδιες επενδύσεις τον αλγόριθμο της άπληστης μεθόδου, πήραμε αποτέλεσμα συνολικής απόδοσης 128.75. Βλέπουμε ότι ο γ.α μας έδωσε καλύτερο αποτέλεσμα από την άπληστη μέθοδο. Αυτό βέβαια δεν είναι σίγουρο κάθε φορά που εκτελείται ο γ.α. Για παράδειγμα αν εκτελεστεί ξανά ο γ.α δεν σημαίνει απαραίτητα ότι θα δώσει το ίδιο ή και καλύτερο αποτέλεσμα (μπορεί να είναι και χειρότερο) το οποίο οφείλεται στη τυχαιότητα που διακρίνει τον γ.α. Το ενθαρρυντικό σημάδι είναι ότι αυτή τη φορά έδωσε καλύτερο αποτέλεσμα σε σχέση με την άπληστη μέθοδο.

Σίγουρα οι γ.α είναι πολλά υποσχόμενοι σε τέτοιου είδους προβλήματα όμως όπως φάνηκε όλοι οι μέθοδοι έχουνε πλεονεκτήματα και μειονεκτήματα. Η επιλογή της μεθόδου εξαρτάται από το είδος του προβλήματος και τα δεδομένα. Για παράδειγμα αν έχουμε μικρό αριθμό αντικειμένων μπορούμε να χρησιμοποιήσουμε την brute-force προσέγγιση για να πάρουμε σίγουρα την μέγιστη απόδοση. Επειδή το συγκεκριμένο πρόβλημα που εξετάσαμε είναι NP-hard, μια μελλοντική μελέτη και κατασκευή ενός υβριδικού αλγορίθμου ο οποίος ενδεχομένως να χρησιμοποιούσε συνδυασμό των παραπάνω μεθόδων θα ήταν ενδιαφέρουσα. Σε κάθε περίπτωση αφήνεται στον αναγνώστη να αποπειραθεί να δοκιμάσει νέες τεχνικές επίλυσης του συγκεκριμένου προβλήματος καθώς επίσης και να το συνδέσει με πραγματικά προβλήματα.

## ΒΙΒΛΙΟΓΡΑΦΙΑ


[1] Brian W. Kernighan & Dennis M. Ritchie, "*Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C*", ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ, 1990.

[2] Sartaj Sahni, "*Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++*", Μτφρ. Γ. Θεοδωρίδης & Γ. Μανωλόπουλος (επιμ.), ΕΚΔΟΣΕΙΣ ΤΖΙΟΛΑ, 2004.


## ΞΕΝΟΓΛΩΣΣΗ ΒΙΒΛΙΟΓΡΑΦΙΑ


[3] David E. Goldberg, "*Genetic Algorithms in Search, Optimization and Machine Learning*", Addison Wesley Publishing Company, Inc., 1989.

[4] Dennis Komm, "An Introduction to Online Computation", Springer, 2016.

[5] Silvano Martello & Paolo Toth, "*Knapsack Problems, Algorithms and Computer Implementations*", John Wiley & Sons Ltd, 1990.


## ΗΛΕΚΤΡΟΝΙΚΕΣ ΠΗΓΕΣ


[6] *https://en.wikipedia.org/wiki/Knapsack_problem*, (τελευταία πρόσβαση στις 01/10/2017).

[7] *https://en.wikipedia.org/wiki/Genetic_algorithm*, (τελευταία πρόσβαση στις 01/10/2017).

[8] *https://www.youtube.com/watch?v=9kbzMeEBvUY*, (τελευταία πρόσβαση στις 01/10/2017).

[9] *http://www.math.ntua.gr/~coletsos/Documents/paradeigmata.pdf*, (τελευταία πρόσβαση στις 01/10/2017).


## REFERENCES (ΑΝΑΦΟΡΕΣ)


[10] C. Cotta and J. van Hemert (Eds.), pp. 210 – 218, 2007, © Springer-Verlag Berlin Heidelberg, 2007.

[11] Cotta, C., Troya, J., "*A Hybrid Genetic Algorithm for the 0–1 Multiple Knapsack problem. Artificial Neural Nets and Genetic Algorithm 3*", 250–254, 1994.

[12] Chu, P., Beasley, J., "*A Genetic Algorithm for the Multidimensional Knapsack Problem. Heuristics 4'*', Journal of heuristics, Springer, 63–86, 1998.

[13] Djannaty, F., Doostdar, S.,"*A Hybrid Genetic Algorithm for the Multidimensional Knapsack Problem.*" Contemp. Math. Sciences 3(9), 443–456, 2008.

[14] FEDERICO GONZÁLEZ-SANTOYO, BEATRIZ FLORES-ROMERO, JUAN J.FLORES, and ANNA M. GIL-LAFUENTE., "*Optimal Investment in Portfolio Selection Using the Knapsack Model*", Universidad Michoacana de San Nicolás de Hidalgo, Mexico.

[15] Maya Hristakeva and Dipti Shrestha., "*Solving the 0-1 Knapsack Problem with Genetic Algorithms"*, Computer Science Department Simpson College.

[16] Weiss, M. A. (1999). "*Data Structures & Algorithms Analysis in C++"*. USA: Addison Wesley Longman Inc..

[17] Y. Akçay, H Li, SH Xu – "*Greedy algorithm for the general multidimensional knapsack problem*", Annals of Operations Research, Springer, 2007.