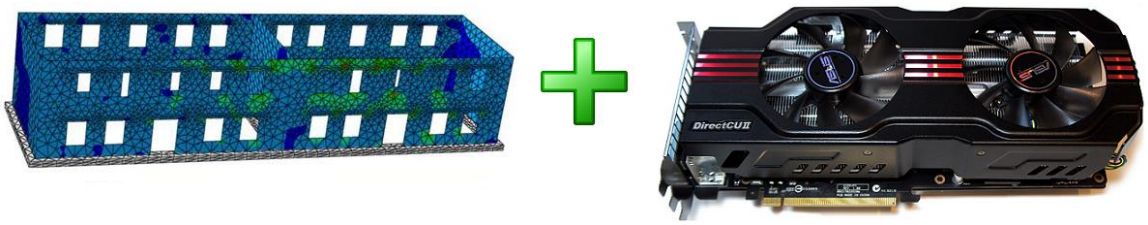
ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΠΟΛΙΤΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΔΟΜΟΣΤΑΤΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΣΤΑΤΙΚΗΣ ΚΑΙ ΑΝΤΙΣΕΙΣΜΙΚΩΝ ΚΑΤΑΣΚΕΥΩΝ

ΒΕΛΤΙΣΤΟΣ ΣΧΕΔΙΑΣΜΟΣ ΚΑΤΑΣΚΕΥΩΝ ΣΕ ΕΤΕΡΟΓΕΝΗ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ CPU & GPU

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΕΡΑΦΕΙΜ ΜΠΑΚΑΛΑΚΟΣ



Επιβλέπων:
Μανόλης Παπαδρακάκης, Καθηγητής ΕΜΠ

Αθήνα, 20 Οκτωβρίου 2015

Σύνοψη

Η διπλωματική αυτή εργασία ασχολείται με το βέλτιστο σχεδιασμό κατασκευών χρησιμοποιώντας τη μέθοδο των πεπερασμένων στοιχείων (FEM). Εξετάζονται κυρίως μεταευριστικοί αλγόριθμοι σε προβλήματα βελτιστοποίησης μεγέθους διατομών. Η ανάλυση των φορέων με τη FEM δεν εκτελειται με τον παραδοσιακό τρόπο στη CPU (Central Processing Unit), αλλά σε ετερογενή υπολογιστικά συστήματα που χρησιμοποιούν τη CPU σε συνεργασία με τη GPU (Graphics Processing Unit). Οι GPU έχουν γενικά πολύ μεγαλύτερη υπολογιστική ισχύ, που μπορεί να μειώσει δραστικά το χρόνο εκτέλεσης των πιο απαιτητικών διεργασιών της FEM, όπως είναι η επίλυση γραμμικών συστημάτων μεγάλης τάξης. Όλοι οι σύγχρονοι υπολογιστές είναι εφοδιασμένοι με GPU που παραδοσιακά χρησιμοποιούνται μόνο για την απόδοση γραφικών στην οθόνη. Με την ανάθεση των πλέον χρονοβόρων υπολογισμών στην GPU και τη χρήση της CPU για τη ροή ελέγχου του προγράμματος και άλλες πιο περίπλοκες αλλά όχι χρονοβόρες διεργασίες, αξιοποιούνται όλοι οι υπολογιστικοί πόροι του υπολογιστή και μειώνεται σημαντικά ο συνολικός χρόνος εκτέλεσης. Κατά τη διαδικασία της βελτιστοποίησης πραγματοποιείται μεγάλο πλήθος αναλύσεων FEM, οπότε είναι αναγκαίο να βρεθούν όσο το δυνατόν γρηγορότεροι τρόποι ανάλυσης.

Κατά τη διάρκεια της εργασίας, για τις αναλύσεις πεπερασμένων στοιχείων χρησιμοποιήθηκε το λογισμικό Solverize, που αναπτύσσεται από το Εργαστήριο Στατικής και Αντισεισμικών Κατασκευών του τμήματος Πολιτικών Μηχανικών, ΕΜΠ. Σε αυτό προστέθηκε η δυνατότητα εκτέλεσης βελτιστοποίησης, χρησιμοποιοώντας τους μεταευριστικούς αλγορίθμους: Βελτιστοποίηση Σμήνους Σωματιδίων, Διαφορική Εξέλιξη και Στρατηγικές Εξέλιξης. Επιπλέον προστέθηκε και κώδικας σε CUDA C που επίλυει συστήματα γραμμικών εξισώσεων στη GPU.

Στο πρώτο κεφάλαιο γίνεται εισαγωγή του αναγνώστη στο κλάδο της βελτιστοποίησης. Ομαδοποιούνται σε μερικές βασικές κατηγορίες τα προβλήματα και οι μέθοδοι βελτιστοποίησης. Περεταίρω ενδιαφέρον δίνεται στο βέλτιστο σχεδιασμό κατασκευών.

Στο δευτερο κεφάλαιο αναλύονται οι αλγόριθμοι Βελτιστοποίηση Σμήνους Σωματιδίων, Διαφορική Εξέλιξη και Στρατηγικές Εξέλιξης. Παρουσιάζονται οι βασικές εκδοχές τους και ο τρόπος εφαρμογής σε προβλήματα βέλτιστου σχεδιασμού κατασκευών.

Στο τρίτο κεφάλαιο ελέγχεται η συμπεριφορά των παραπάνω αλγορίθμων σε μερικά από τα συνηθέστερα προβλήματα αξιολόγησης (benchmarks).

Στο τέταρτο κεφάλαιο δίνεται μια συνοπτική εισαγωγή στον προγραμματισμό για GPU και τον παράλληλο προγραμματισμό γενικότερα. Εξηγούνται οι βασικές διαφορές με τον παραδοσιακό σειριακό προγραμματισμό

Στο πέμπτο κεφάλαιο παρουσιάζονται εποπτικά τα λογισμικά που συνδυάζονται για το βέλτιστο σχεδιασμό κατασκευών και εφαρμόζονται σε ένα αριθμητικό πράδειγμα. Επίσης δοκιμάζονται παράλληλα μοτίβα, για την καλύτερη εκμετάλλευση των CPU και GPU.

Abstract

The current thesis deals with structural optimization using the Finite Element Method.The main focus is on metaheuristic optimization algorithms for sizing optimization problems. The analysis of the structures using FEM is not executed by the CPU (Central Processing Unit) as usual. Instead heterogenous computing systems are employed, utilizing both the CPU and the GPU (Grahics Processing Unit) of the used computer. GPUs boast a much higher processing power, that can significantly reduce the execution time of the more demanding parts of the Finite Element Method, such as the solution of large linear systems. Nowadays all personal computers ship with a GPU, which is used for rendering computer graphics on a monitor. By using the GPU to tackle the most computationally heavy parts of thw program, while the CPU takes care of the main program flow and other trickier processes, one can take advantage of all available resources and greatly improve overall efficiency. The need for faster structural analyses is even greater in optimization, as each design that is tested requires a separate FEM analysis and in most cases the number of tested designs is quite high.

All FEM analyses in this project were performed by Solverize, a software being developed by the Statics and Antiseismic Research Laboratory of NTUA's Civil Engineering department. A basic optimization framework has been developed in Java to allow the application of metaheuristic algorithms for structural optimization. Additionally CUDA C code was developed for solving linear systems on NVidia brand GPUs. Both were integrated with Solverize.

The first chapter introduces the main aspects of optimization and attempts to classify the broad spectrum of optimization problems, especially those pertaining to structural optimization, as well as the methologies for solving them.

The second chapter presents three metaheuristic optimization algorithms, which are then used throughout this thesis: Particle Swarm Optimization, Differential Evolution and Evolutionary Strategies.

The third chapter tests the performance of those algorithms in a set of the most common benchmarks.

The fourth chapter introduces General Purpose GPU computing and some main aspects of parallel computing.

The fifth chapter describes the workflow of the software used in this thesis to perform structural optimization. Additionaly a numerical example is examined: optimizing the weight of a steel frame structure. Finally, parallel software patterns are examined in an attempt to take better advantage of the processing power of the GPU and CPU cores.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά:

- Τον καθηγητή μου κ. Μανόλη Παπαδρακάκη για την πολύτιμη καθοδήγησή του κατά το μεγαλύτερο μέρος των προπτυχιακών μου σπουδών, συμπεριλαμβονομένης και της παρούσας εργασίας, και για μερικά από τα πιο εμπνευσμένα μαθήματα που διδάχθηκα στη σχολή Πολιτικών Μηχανικών.

- Το διδάκτορα και φίλο μου Αλέξανδρο Καραταράκη, που ουσιαστικά είναι ο δάσκαλός μου σε θέματα προγραμματισμού τα τελευταία 3 χρόνια και μεταξύ άλλων βοήθησε τον εγκλιματισμό μου με το Solverize και προσέθεσε δυνατότητες που χρειαζόμουν για να αναπτύξω τον δικό μου κώδικα.

- Το Θεόφιλο Μανιταρά για την πολύτιμη συμβολή του σε θέματα αριθμητικής ανάλυσης και στις πολλές δυσκολίες που προέκυψαν κατά την εφαρμογή σε πρακτικά προβλήματα των διαφόρων υπολογιστικών εργαλείων, που χρησιμοποίησα στην εργασία αυτή.

- Τον κ. Βαγγέλη Πλεύρη, Επίκουρο Καθηγητή στην ΑΣΠΑΙΤΕ, για την καθοδήγηση και τις συμβουλές του στον κλάδο της βελτιστοποίησης.

- Τους γονείς μου για την υλική και ψυχολογική στήριξη που μου έχουν προσφέρει καθόλη τη διάρκεια της ζωής μου.

1 Contents

Λιστα Σχημάτων

Λίστα Πινάκων

Κεφάλαιο 1

1 Βελτιστοποίηση

1.1 Εισαγωγή

Βελτιστοποίηση είναι η εύρεση της καλύτερης (ως προς κάποια κριτήρια) λύσης ενός προβλήματος από ένα σύνολο δυνατών εναλλακτικών λύσεων. Ένα τυπικό πρόβλημα βελτιστοποίησης περιλαμβάνει μια συνάρτηση πραγματικών μεταβλητών που βρίσκονται σε ορισμένο εύρος και υπόκεινται σε συγκεκριμένους περιορισμούς. Ζητούμενο είναι η εύρεση των τιμών των μεταβλητών αυτών για τις οποίες η συνάρτηση ελαχιστοποιείται ή μεγιστοποιείται, χωρίς κάποιος από τους περιορισμούς να παραβιάζεται.

Στον κλάδο των μηχανικών, επιδιώκουμε το σχεδιασμό κατασκευών που να μπορούν να αναλάβουν τα επιβαλλόμενα φορτία με ασφάλεια. Αυτό όμως δεν είναι το μοναδικό κριτήριο. Ένα κτίριο που απλά ικανοποιεί τις απαιτήσεις αντοχής και λειτουργικότητας δεν θεωρείται αποδεκτή λύση. Στην πράξη ο μηχανικός πρέπει να δώσει μία λύση που να είναι όσο το δυνατόν πιο οικονομική, χωρίς να αποκλείονται και άλλα κριτήρια, όπως η κατασκευαστική ευκολία. Για να το πετύχει αυτό εφαρμόζει μια επαναληπτική διαδικασία, κατά την οποία δοκιμάζει διάφορες λύσεις μέχρι να καταλήξει σε κάποια που να είναι ασφαλής και ικανοποιητικά οικονομική. Επομένως ο σχεδιασμός μιας κατασκευής είναι ένα πρόβλημα βελτιστοποίησης, με μεταβλητές τα μέρη της κατασκευής, αντικείμενο την ελαχιστοποίηση του κόστους και περιορισμούς τις απαιτήσεις ασφαλείας και λειτουργικότητας.

Παλαιότερα, πριν αναπτυχθούν και γίνουν δημοφιλείς αλγόριθμοι βελτιστοποίησης που να μπορούν να δώσουν λύση σε προβλήματα σχεδιασμού κτιρίων, ο μηχανικός ήταν αναγκασμένος να κάνει διαδοχικές δοκιμές βασιζόμενος στην εμπειρία του για να ελαχιστοποιήσει το κόστος. Τέτοιες παραμετρικές διερευνήσεις «δοκιμής και διόρθωσης» (trial and error), μπορούσαν να εφαρμοστούν για μικρό μόνο πλήθος πιθανών συνδυασμών. Σε μεγάλους και πολύπλοκους φορείς, όπου οι απαιτούμενοι υπολογισμοί είναι υπερβολικά χρονοβόροι, η παραπάνω εμπειρική μέθοδος δεν είναι ρεαλιστική. Η επανάληψη όλων των υπολογισμών για κάθε λύση που δοκιμάζεται διαδοχικά ήταν αδύνατη ή στην καλύτερη περίπτωση υπερέβαινε τα χρονικά πλαίσια μιας μελέτης.

Σήμερα η ραγδαία εξέλιξη της τεχνολογίας υπολογιστών και η ανάπτυξη αποδοτικών μεθόδων βελτιστοποίησης επιτρέπουν την αυτοματοποίηση της διαδικασίας. Τα σύγχρονα υπολογιστικά συστήματα μπορούν να χρησιμοποιήσουν τη μέθοδο των πεπερασμένων στοιχείων για να δώσουν ακριβή αποτελέσματα σε λογικό χρόνο ακόμα και για φορείς με μεγάλο αριθμό βαθμών ελευθερίας. Η ανάπτυξη των κλάδων των Εφαρμοσμένων Μαθηματικών, των

Οικονομικών και της Επιχειρησιακής Έρευνας έχει γεννήσει ποικίλους αλγορίθμους που εντοπίζουν γρήγορα τη βέλτιστη λύση χωρίς μεγάλο πλήθος δοκιμών. Επίσης, τα τελευταία χρόνια έχουν αναπτυχθεί γενικοί αλγόριθμοι βελτιστοποίησης, που χρειάζονται ελάχιστες πληροφορίες από το σύστημα υπό εξέταση και άρα μπορούν να εφαρμοστούν αποδοτικά σε διάφορα προβλήματα.

Η αυτοματοποίηση της διαδικασίας σχεδιασμού επιτρέπει την εύρεση οικονομικότερων λύσεων, με ακρίβεια, ασφάλεια και εντός του διαθέσιμου χρόνου για τη μελέτη. Έτσι, ο μηχανικός αποδεσμεύεται από τον υπολογιστικό φόρτο, αλλά πρέπει να κατέχει την απαραίτητη γνώση και εμπειρία για τη σωστή χρήση των υπολογιστικών αυτών εργαλείων. Για να επιλύσει ένα τυπικό πρόβλημα βέλτιστου σχεδιασμού κτιρίου, πρέπει να προσομοιώσει με ικανοποιητική πιστότητα το δομικό μοντέλο, να επιλέξει τις μεταβλητές, τα κριτήρια και τους περιορισμούς του προβλήματος βελτιστοποίησης και να εφαρμόσει έναν κατάλληλο αλγόριθμο που όχι μόνο να είναι ικανός να λύσει το συγκεκριμένο πρόβλημα, αλλά και να είναι αποδοτικός. Τέλος πρέπει να αξιολογήσει τη λύση με βάση την εμπειρία του και να την εφαρμόσει στην πράξη.

1.2 Ιστορική Αναδρομή

Κάποιες από τις σημαντικότερες εξελίξεις στο χώρο της βελτιστοποίησης είναι:

- Ο Ευκλείδης (300 π.Χ) απέδειξε ότι η ελάχιστη απόσταση μεταξύ δύο σημείων είναι το ευθύγραμμο τμήμα και ότι από τα τετράπλευρα που έχουν δεδομένη περίμετρο, μέγιστο εμβαδό έχει το τετράγωνο.
- Ο Ήρων (100π.Χ.) στο έργο του *Κατοπτρικά*, αποδεικνύει ότι το φως ταξιδεύει μεταξύ δύο σημείων ακολουθώντας τη διαδρομή με το μικρότερο μήκος, όταν ανακλάται σε έναν καθρέπτη.
- Τον 17^{ο} αιώνα ο P. Fermat βρίσκει τα θεωρήματα πρώτης και δευτέρας παραγώγου, που αφορούν τον εντοπισμό σημείων ακρότατων και το χαρακτηρισμό τους σε ελάχιστα ή μέγιστα.
- Ο I. Newton (17^{ος}-18^{ος} αιώνας) δημιουργεί επαναληπτικές μεθόδους για την προσεγγιστική εύρεση λύσεων συναρτήσεων.
- Τον 18^{ο} αιώνα οι L. Euler και J.L. Lagrange εξελίσσουν τον λογισμό μεταβολών που αποτελεί βάση για την μετέπειτα ανάπτυξη του κλάδου της βελτιστοποίησης. Κάποιες από τις σημαντικότερες συνεισφορές είναι η εξίσωση Euler-Lagrange και η μέθοδος πολλαπλασιαστών Lagrange, οι οποίες δίνουν λύση σε προβλήματα βελτιστοποίησης. Επιπλέον, ο Euler ασχολείται με τον βέλτιστο σχεδιασμό κατασκευών και άλλους τομείς της μηχανικής.
- Ο C.F. Gauss (18^{ος} -19^{ος} αιώνας) αναπτύσσει μεταξύ άλλων επαναληπτικές μεθόδους που βρίσκουν εφαρμογή στη βελτιστοποίηση.
- Το 1806 ο A.M. Legendre δημοσιεύει τη μέθοδο ελαχίστων τετραγώνων, με τη οποία ο Gauss έχει ήδη ασχοληθεί από το 1795.

- Το 1826 ο J.B.J. Fourier ασχολείται με προβλήματα γραμμικού προγραμματισμού που συναντώνται στην μηχανική και τη θεωρία πιθανοτήτων. Βρίσκει μία μέθοδο απαλοιφής γραμμικών συναρτήσεων περιορισμών που μπορεί να χρησιμοποιηθεί για την επίλυση τέτοιων προβλημάτων.
- Το 1847 ο A.L. Cauchy παρουσιάζει τη μέθοδό βελτιστοποίησης του χρησιμοποιώντας κλίσεις (gradient method). Πολλές παραλλαγές έχουν δημιουργηθεί έκτοτε, όπως η Steepest Descent Method.
- Ο Αυστραλός μηχανικός A. Michell θεωρείται πρωτεργάτης στο χώρο του βέλτιστου σχεδιασμού κατασκευών. Το 1904 δημοσιεύει άρθρο στο οποίο ασχολείται με την ελαχιστοποίηση του βάρους δικτυωτών κατασκευών υπό περιορισμούς τάσεων.
- Το 1906 ο J. Jensen ασχολείται με τις ιδιότητες των κυρτών συναρτήσεων στο πλαίσιο της θεωρίας πιθανοτήτων. Η έννοια βέβαια της κυρτής συνάρτησης αποδίδεται στον Αρχιμήδη.
- Το 1917 ο H. Hancock δημοσιεύει το πρώτο εγχειρίδιο πάνω στον τομέα της βελτιστοποίησης, *Theory of Minima and Maxima.*
- Ο L.V. Kantorovich θεωρείται ιδρυτής του κλάδου του γραμμικού προγραμματισμού. Το 1939 παρουσιάζει το LP μοντέλο και μία μέθοδο επίλυσής του.
- Το 1943 ο R. Courant εισάγει την έννοια των συναρτήσεων ποινής (penalty functions).
- Το 1947 ο μαθηματικός G. Dantzig, εργαζόμενος για την στρατιωτική αεροπορία των ΗΠΑ, δημιουργεί τον αλγόριθμο Simplex για την επίλυση προβλημάτων γραμμικού προγραμματισμού. Ο Simplex θεωρείται από τους πλέον σημαντικούς αλγορίθμους του $20^{ου}$ αιώνα.
- Την ίδια χρονιά ο Von Neumann παρουσιάζει τη θεωρία της δυαδικότητας (duality) για προβλήματα βελτιστοποίησης γραμμικού προγραμματισμού
- Το 1951 οι H.W. Kuhn και A.W. Tucker διατυπώνουν τις απαραίτητες συνθήκες που πρέπει να πληροί μια λύση σε προβλήματα μη γραμμικού προγραμματισμού, ώστε να είναι βέλτιστη. Τέτοιες συνθήκες δημοσίευσε πρώτος ο W. Karush το 1939, αλλά δεν αναγνωρίστηκε μέχρι το 1951.
- Οι Robbins & Monro το 1951 και οι Kiefer & Wolfowitz το 1952 εισήγαγαν στοχαστικές μεθόδους βελτιστοποίησης, οι οποίες μπορούν να θεωρηθούν ως πρωτότυπα για τους μεταευριστικούς αλγορίθμους.
- Στα μέσα της δεκαετίας του 1950, ο Νορβηγός οικονομολόγος R. Frisch αναπτύσσει έναν από τους πρώτους αλγόριθμους για την επίλυση μη γραμμικών προβλημάτων.
- Το 1957 ο R. Bellman παρουσιάζει μία εξίσωση (Principle Of Optimality) για προβλήματα δυναμικού προγραμματισμού, που αποτελεί αναγκαία συνθήκη ώστε μια λύση να είναι βέλτιστη.
- Το 1960 ο G. Zoutendijk δημιουργεί τη Μέθοδο Εφικτών Διευθύνσεων (Feasible Directions Method), που γενικεύει τον αλγόριθμο Simplex για μη γραμμικά προβλήματα. Οι Roben, Wolfe και Powell αναπτύσσουν παρόμοιες ιδέες.
- Ο L. Schmit δημοσιεύει ένα ιστορικό άρθρο το 1960, όπου προαναγγέλλει τη σύγχρονη εποχή της βελτιστοποίησης κατασκευών, που βασίζεται στους ηλεκτρονικούς υπολογιστές. Στο άρθρο αυτό γίνεται μια περιεκτική παρουσίαση της χρήσης μαθηματικά

προγραμματιζόμενων τεχνικών για την επίλυση μη γραμμικών προβλημάτων με ανισοτικές συναρτήσεις περιορισμών, για τον σχεδιασμό ελαστικών κατασκευών υπό την επίδραση πολλαπλών φορτίσεων. Ο Schmit κάνει επίσης διάκριση μεταξύ των κλάδων του μαθηματικού προγραμματισμού και του βέλτιστου σχεδιασμού κατασκευών.

- Το 1963 ο Wilson εφευρίσκει την Sequential Quadratic Programming μία από τις πλέον διαδεδομένες μεθόδους για τη επίλυση μη γραμμικών προβλημάτων, ιδιαίτερα στη βελτιστοποίηση κατασκευών. Οι Han (1975) και Powell (1977) εξελίσσουν περαιτέρω την SQP.
- Ο J. Holland αναπτύσσει τη μέθοδο Genetic Algorithms την δεκαετία του 1960, που γρήγορα γίνεται δημοφιλής και εφαρμόζεται συχνά στο βέλτιστο σχεδιασμό κατασκευών.
- Ο I.Rechenberg (1973) και ο H.P Schwefel (1981) αναπτύσσουν την μέθοδο Evolutionary Strategies, έναν από τους πρώτους μετευρετικούς αλγόριθμους που προσομοιάζουν την βιολογική εξέλιξη οργανισμών. Ο κλάδος των αλγορίθμων αυτών βρίσκει ευρύτατες εφαρμογές στην τεχνητή νοημοσύνη και είχε ήδη ξεκινήσει να αναπτύσσεται από τη δεκαετία του 1960. Η ομάδα του Rechenberg πετυχαίνει την αποτελεσματική εφαρμογή της μεθόδου σε πολύπλοκα προβλήματα μηχανικής.
- Τη δεκαετία του 1980 οι υπολογιστές γίνονται πιο αποδοτικοί και προσβάσιμοι. Οι μετευρετικοί αλγόριθμοι, που απαιτούν μεγάλο αριθμό δοκιμών, αρχίζουν να αποκτούν ευρεία αποδοχή για την καθολική βελτιστοποίηση μεγάλων και περίπλοκων προβλημάτων.
- Το 1984 ο N. Karmarkar εισάγει τον πρώτο αλγόριθμο που επιλύει προβλήματα γραμμικού προγραμματισμού σε πολυωνυμικό χρόνο και είναι ικανοποιητικά αποδοτικός σε πρακτικές εφαρμογές.
- Το 1988 οι Bendsoe και Kikuchi προτείνουν τη μέθοδο ομογενοποίησής τους για την αντιμετώπιση προβλημάτων βελτιστοποίησης τοπολογίας. Σε αντίθεση με τις υπάρχουσες μεθόδους, τους επιτρέπει να δημιουργούν μικροδομές εντός του υλικού.

1.3 Μαθηματική Βελτιστοποίηση

Το μαθηματικό μοντέλο ενός προβλήματος μη γραμμικού προγραμματισμού (Non Linear Programming) μπορεί να διατυπωθεί ως εξής:

$$\begin{aligned} & F(\mathbf{x}) \rightarrow \min, \\ & \mathbf{x} = [x_1, x_2, \ldots, x_n]^T \\ & x_i \in X_i, \qquad i = 1, 2, \ldots, n \\ & g_j(\mathbf{x}) \le 0, \qquad j = 1, 2, \ldots, p \\ & h_k(\mathbf{x}) = 0, \qquad k = 1, 2, \ldots, q \end{aligned} \tag{1.1}$$

όπου:

- $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ είναι διάνυσμα μήκους *n* που περιέχει τις μεταβλητές σχεδιασμού (design variables)
- $F(\mathbf{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}$ είναι η αντικειμενική συνάρτηση (objective function) που επιδιώκουμε να ελαχιστοποιήσουμε
- X_i είναι το σύνολο των δυνατών τιμών που μπορεί να πάρει η μεταβλητή x_i.
- $g_j(\mathbf{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}$ είναι οι συναρτήσεις περιορισμών ανισοτήτων (inequality constraints) πλήθους p
- $h_j(\mathbf{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}$ είναι οι συναρτήσεις περιορισμών ισοτήτων (equality constraints) πλήθους q

Η αντικειμενική συνάρτηση και οι συναρτήσεις των περιορισμών είναι μη γραμμικές. Στην περίπτωση που όλες οι συναρτήσεις είναι γραμμικές, το πρόβλημα αναφέρεται ως πρόβλημα γραμμικού προγραμματισμού (Linear Programming) και είναι πιο εύκολο να επιλυθεί.

Αυτή είναι η διατύπωση ενός προβλήματος ελαχιστοποίησης. Προβλήματα μεγιστοποίησης, όπου ζητούμενο είναι η μεγιστοποίηση της αντικειμενικής συνάρτησης, μπορούν να γραφτούν ως προβλήματα ελαχιστοποίησης με αντικειμενική συνάρτηση την $-F(\mathbf{x})$.

1.3.1 Μεταβλητές Σχεδιασμού

Μεταβλητές σχεδιασμού είναι οι παράμετροι που παίρνοντας διάφορες τιμές προσδιορίζουν πλήρως έναν σχεδιασμό. Αν ο σχεδιασμός αυτός ικανοποιεί τους περιορισμούς του προβλήματος, τότε λέγεται εφικτός (feasible design), αλλιώς λέγεται ανέφικτος (infeasible design). Σε αντίθεση με τους ανέφικτους, κάθε εφικτός σχεδιασμός αποτελεί μία δυνατή λύση του προβλήματος προς βελτιστοποίηση. Ωστόσο, από όλους τους εφικτούς σχεδιασμούς μας ενδιαφέρει αυτός που ελαχιστοποιεί την αντικειμενική συνάρτηση.

Οι μεταβλητές σχεδιασμού μπορεί να είναι συνεχείς, ακέραιες ή διακριτές.

- Οι συνεχείς μεταβλητές είναι πραγματικές. Το σύνολο των δυνατών τιμών X_i που μπορεί να λάβει μια συνεχής μεταβλητή x_i είναι ένα διάστημα πραγματικών αριθμών και μπορεί να περιγραφεί ως $l_i \leq x_i \leq u_i$, όπου l_i και u_i είναι το κάτω (lower) και άνω (upper) όριο αντίστοιχα.
- Οι ακέραιες μεταβλητές χρησιμοποιούνται κυρίως σε προβλήματα ακέραιου προγραμματισμού. Ειδική κατηγορία είναι η περίπτωση που οι μεταβλητές είναι δυαδικές, δηλαδή μπορούν να λάβουν μόνο τις τιμές 0 και 1.
- Οι διακριτές μεταβλητές λαμβάνουν τιμές από πεπερασμένα σύνολα X_i (πραγματικά ή ακέραια). Στο σχεδιασμό κατασκευών πολύ συχνά οι μεταβλητές είναι διακριτές, καθώς τα

μέλη του φορέα είναι ράβδοι με τυποποιημένες διατομές, λόγω κατασκευαστικών περιορισμών. Παρ' ότι οι ιδιότητες των υλικών είναι από φύση τους συνεχείς πραγματικοί αριθμοί, οι μηχανικοί χρησιμοποιούνι ονομαστικές τιμές τους που τελικά είναι διακριτές.

- Στο ίδιο πρόβλημα είναι δυνατό να συνυπάρχουν διαφορετικά ήδη μεταβλητών. Τα προβλήματα αυτά είναι μεικτά και συχνά μετατρέπονται σε συνεχή πριν την επίλυσή τους. Για παράδειγμα μια κατασκευή από οπλισμένο σκυρόδεμα έχει συνεχείς διατομές δοκών και υποστυλωμάτων και πάχη πλακών. Ταυτόχρονα όμως, η αντοχή του σκυροδέματος και το πλήθος των ράβδων οπλισμού ανά μέλος παίρνουν διακριτές τιμές.

Η επιλογή των μεταβλητών είναι πολύ σημαντική, αφού επηρεάζει την ορθότητα και την αποδοτικότητα της βελτιστοποίησης. Η ανεξαρτησία μεταξύ των μεταβλητών είναι απαραίτητη ώστε να αποφευχθούν πολύπλοκα συστήματα. Μεταβλητές των οποίων η διακύμανση δεν επηρεάζει σημαντικά την αντικειμενική συνάρτηση, αυξάνουν μάταια το υπολογιστικό κόστος. Τέτοιες μεταβλητές συνήθως θεωρούνται δευτερεύουσες (secondary) και συνδέονται με τις κύριες (primary) μέσω γραμμικών σχέσεων (design variable linking). Συχνά είναι λοιπόν ωφέλιμο να πραγματοποιηθεί ανάλυση ευαισθησίας, πριν την επιλογή των μεταβλητών σχεδιασμού.

Μία τεχνική, που εφαρμόζεται ευρέως, είναι η ομαδοποίηση των μεταβλητών σχεδιασμού. Επιτρέπει την σημαντική μείωση του υπολογιστικού κόστους και του απαιτούμενου χρόνου επίλυσης. Επιπλέον διευκολύνει την αντιμετώπιση μεταβλητών που μεμονωμένα δεν θα επηρέαζαν σημαντικά την αντικειμενική συνάρτηση. Στο σχεδιασμό κατασκευών, συχνά ομαδοποιούνται μέρη του φορέα που παρουσιάζουν συμμετρία ή επανάληψη (π.χ. ίδιοι όροφοι), ώστε να διευκολυνθεί και η διαδικασία κατασκευής.

1.3.2 Αντικειμενική Συνάρτηση

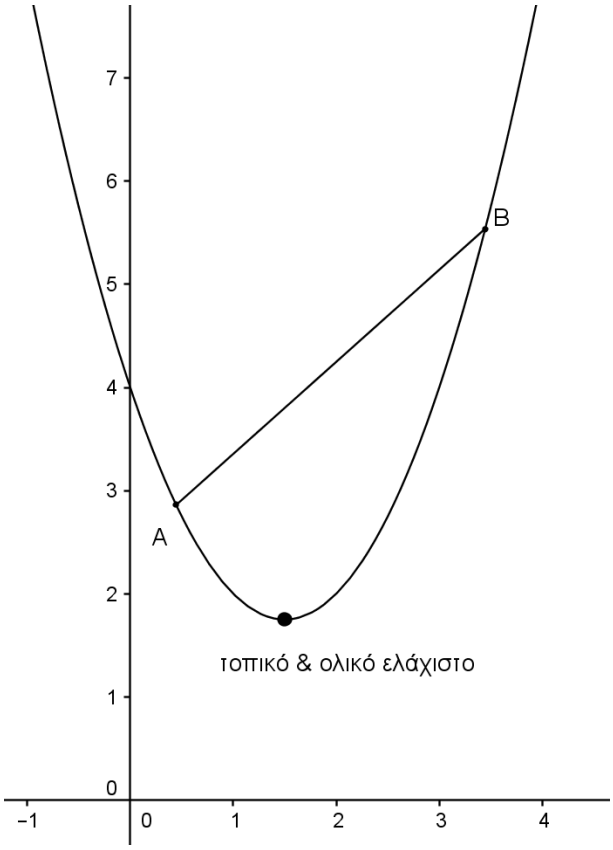
Αντικειμενική συνάρτηση είναι το κριτήριο με το οποίο αξιολογούμε τους διάφορους εφικτούς σχεδιασμούς ώστε να καταλήξουμε στον βέλτιστο. Χωρίς περιορισμό της γενικότητας, σκοπός είναι η ελαχιστοποίηση της αντικειμενικής συνάρτησης. Τέτοια προβλήματα βελτιστοποίησης ονομάζονται προβλήματα ελαχιστοποίησης και η αντικειμενική συνάρτηση συχνά αποκαλείται συνάρτηση κόστους. Τα προβλήματα μεγιστοποίησης μπορούν να αναχθούν σε προβλήματα ελαχιστοποίησης με αντικειμενική συνάρτηση την $-F(\mathbf{x})$.

Προβλήματα, όπου υπάρχει μόνο μία αντικειμενική συνάρτηση ή υπάρχουν περισσότερες αντικειμενικές συναρτήσεις, που όμως βελτιστοποιούνται ταυτόχρονα, ονομάζονται προβλήματα βελτιστοποίησης μίας αντικειμενικής συνάρτησης (Single Objective Optimization). Πολλές φορές όμως επιδιώκουμε να βελτιστοποιήσουμε δύο ή περισσότερες συναρτήσεις, οι οποίες αντικρούονται, δηλαδή η μείωση της τιμής της μιας προκαλεί αύξηση των τιμών των υπολοίπων. Χαρακτηριστικό παράδειγμα είναι η ελαχιστοποίηση του βάρους ενός φορέα που όμως αυξάνει τις τιμές των τάσεων στα στοιχεία του, τις οποίες όμως θέλουμε επίσης να ελαχιστοποιήσουμε. Τέτοια προβλήματα ονομάζονται προβλήματα βελτιστοποίησης πολλών μεταβλητών (Multi Objective Optimization) και δεν έχουν μονοσήμαντη λύση. Ζητούμενο είναι η εύρεση ενός συνόλου βέλτιστων λύσεων (Pareto optimal solutions), οι οποίες θεωρούνται εξίσου καλές και καμία άλλη λύση δεν παρουσιάζει καλύτερη συμπεριφορά ως προς όλες τις αντικειμενικές συναρτήσεις.

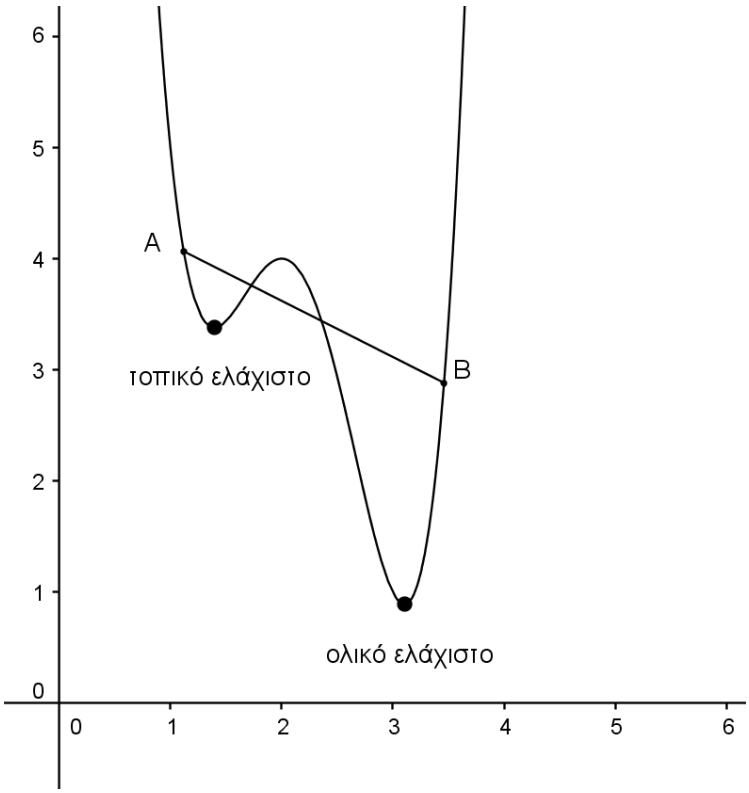
Στον βέλτιστο σχεδιασμό κατασκευών ζητούμενο είναι η ελαχιστοποίηση του κόστους. Χωρίς όμως να παραβιάζονται οι απαιτήσεις αντοχής και λειτουργικότητας. Οι απαιτήσεις αυτές εκφράζονται γενικά ως περιορισμοί τάσεων και μετατοπίσεων. Δεν επιδιώκουμε να τις ελαχιστοποιήσουμε, αλλά να μην ξεπερνούν τα όρια ασφαλείας. Ως αντικειμενική συνάρτηση συχνά χρησιμοποιείται το βάρος της κατασκευής αντί για το συνολικό κόστος, αφού υπολογίζεται ευκολότερα. Το κόστος της κατασκευής θεωρούμε ότι ελαχιστοποιείται ταυτόχρονα με το βάρος της. Αυτό όμως εξαρτάται και από άλλους παράγοντες που δεν σχετίζονται με την μηχανική συμπεριφορά της κατασκευής (πλήθος και κόστος εργασιών, σχετικές τιμές υλικών, κτλ).

Έστω μια μη γραμμική συνάρτηση F(**x**). Αν για κάθε δυο σημεία $\mathbf{x_1}$ και $\mathbf{x_2}$ που βρίσκονται πάνω στη γραφική της παράσταση, το ευθύγραμμο τμήμα που τα ενώνει βρίσκεται εξ' ολοκλήρου είτε πάνω είτε κάτω από αυτήν, τότε η συνάρτηση F λέγεται κυρτή. Στην περίπτωση που το ευθύγραμμο τμήμα τέμνει την γραφική παράσταση, η F λέγεται μη κυρτή.

Στα παρακάτω σχήματα φαίνονται οι γραφικές παραστάσεις μιας κυρτής και μιας μη κυρτής συνάρτησης με μία μόνο μεταβλητή x.



Σχήμα 1.1 Κυρτή συνάρτηση μίας μεταβλητής, με ένα τοπικό ελάχιστο που είναι και ολικό.



Σχήμα 1.2 Μη κυρτή συνάρτηση μίας μεταβλητής, με δύο τοπικά ελάχιστα, από τα οποία μόνο ένα είναι ολικό.

1.3.3 Ελάχιστο

Τα ακρότατα μιας αντικειμενικής συνάρτησης διακρίνονται σε τοπικά και ολικά. Παρακάτω δίνονται ορισμοί για το τοπικό και ολικό ελάχιστο, αλλά αντίστοιχα ισχύουν και για το τοπικό και ολικό μέγιστο.

- **Ολικό ελάχιστο** (global minimum). Έστω ένα διάνυσμα $\mathbf{x}^*$ που παίρνει τιμές στο χώρο σχεδιασμού $\mathbf{X}$. Αν για κάθε άλλο διάνυσμα $\mathbf{x}$ στο χώρο σχεδιασμού ισχύει $F(\mathbf{x}^*) \leq F(\mathbf{x})$, τότε λέμε ότι η F παρουσιάζει ολικό μέγιστο. Το διάνυσμα $\mathbf{x}^*$ λέγεται θέση του ολικού ελαχίστου, ενώ η τιμή $F(\mathbf{x}^*)$ λέγεται ολικό ελάχιστο. Αν ισχύει μόνο η ανισότητα $F(\mathbf{x}^*) < F(\mathbf{x})$, τότε το ολικό ελάχιστο καλείται αυστηρό.
- **Τοπικό ελάχιστο** (local minimum). Έστω ένα διάνυσμα $\mathbf{x}^*$ που παίρνει τιμές στο χώρο σχεδιασμού $\mathbf{X}$. Αν για κάθε άλλο διάνυσμα $\mathbf{x}$, που βρίσκεται εντός μιας περιοχής γύρω από το $\mathbf{x}^*$ (οσοδήποτε μικρής), ισχύει $F(\mathbf{x}^*) \leq F(\mathbf{x})$, τότε λέμε ότι η F παρουσιάζει τοπικό μέγιστο. Το διάνυσμα $\mathbf{x}^*$ λέγεται θέση του τοπικού ελαχίστου, ενώ η τιμή $F(\mathbf{x}^*)$ λέγεται τοπικό ελάχιστο. Αν ισχύει μόνο η ανισότητα $F(\mathbf{x}^*) < F(\mathbf{x})$, τότε το ολικό ελάχιστο καλείται αυστηρό.

Αν υπάρχει το ολικό ελάχιστο τότε είναι και τοπικό. Το αντίστροφο δεν ισχύει. Οι ορισμοί αυτοί ισχύουν για την περίπτωση που το πρόβλημα βελτιστοποίησης δεν υπόκειται σε περιορισμούς. Αν υπάρχουν περιορισμοί, τότε οι παραπάνω ορισμοί δεν ισχύουν για όλο το χώρο σχεδιασμού $\mathbf{X}$, αλλά μόνο στην περιοχή των εφικτών σχεδιασμών, δηλαδή των λύσεων που δεν παραβιάζουν τους περιορισμούς.

Γενικά δεν γνωρίζουμε εκ των προτέρων αν υπάρχει ολικό ή τοπικό ελάχιστο σε ένα πρόβλημα βελτιστοποίησης. Στην ειδική περίπτωση που η αντικειμενική συνάρτηση είναι συνεχής και η περιοχή των εφικτών σχεδιασμών είναι μη κενή (non empty), κλειστή (closed) και καθορισμένη (bounded), τότε η ύπαρξη ολικού ελαχίστου μπορεί να αποδειχθεί μαθηματικά. Αν δεν ισχύουν οι συνθήκες αυτές, τότε είναι δυνατόν να υπάρχει ολικό ελάχιστο, αλλά δεν μπορούμε να το προβλέψουμε πριν εφαρμόσουμε τη μέθοδο βελτιστοποίησης. Η περιοχή των εφικτών σχεδιασμών είναι κενή όταν υπάρχουν υπεράριθμες και αλληλοσυγκρουόμενες συναρτήσεις περιορισμών. Στην περίπτωση αυτή δεν υπάρχουν δυνατές λύσεις του προβλήματος και πρέπει να αφαιρεθούν κάποιοι περιορισμοί. Κλειστή και καθορισμένη θεωρείται όταν οι συναρτήσεις των περιορισμών είναι συνεχής και οι ανισότητες δεν είναι αυστηρές (π.χ. $g_j(\mathbf{x}) < 0$).

Σκοπός της διαδικασίας βελτιστοποίησης είναι η εύρεση του ολικού μεγίστου. Για αντικειμενικές συναρτήσεις που είναι είτε γραμμικές είτε μη γραμμικές αλλά κυρτές, η εύρεσή του είναι γρήγορη και σίγουρη, αν βέβαια αυτό υπάρχει. Αν όμως η αντικειμενική συνάρτηση είναι μη κυρτή, τότε παρουσιάζει έναν αριθμό τοπικών ελαχίστων, ένα από τα οποία είναι το ολικό. Οι μέθοδοι βελτιστοποίησης κινδυνεύουν να παγιδευτούν στις περιοχές των τοπικών ελαχίστων, ή στην καλύτερη περίπτωση να αναλώσουν υπολογιστικό χρόνο σε μάταιες επαναλήψεις πριν απεγκλωβιστούν. Επομένως η κυρτότητα της συνάρτησης επηρεάζει την

απόδοση της μεθόδου βελτιστοποίησης. Πάντως σε προβλήματα βέλτιστου σχεδιασμού κατασκευών ως αντικειμενική συνάρτηση λαμβάνεται συνήθως το βάρος της κατασκευής, το οποίο δεν παρουσιάζει κυρτότητα, αλλά είναι γενικά αύξουσα πολυωνυμική συνάρτηση.

Το πρόβλημα του εγκλωβισμού σε τοπικά ελάχιστα είναι εντονότερο στις μαθηματικές μεθόδους βελτιστοποίησης λόγω του ντετερμινιστικού χαρακτήρα των τελεστών τους. Αντίθετα οι μέθοδοι βελτιστοποίησης που βασίζονται σε πιθανοτικές θεωρήσεις μπορούν να ξεφύγουν πιο εύκολα από τις περιοχές τοπικών ελαχίστων λόγω της τυχηματικότητάς που τις διέπει.

1.3.4 Συναρτήσεις Περιορισμών

Κάθε μεμονωμένη μεταβλητή λαμβάνει τιμές από ένα σύνολο που περιέχει τις δυνατές τιμές της για το δεδομένο πρόβλημα. Ωστόσο ορισμένοι συνδυασμοί μεταβλητών μπορεί να οδηγούν σε μη λογικές ή μη αποδεκτές λύσεις για το φυσικό σύστημα προς βελτιστοποίηση. Για παράδειγμα, μπορεί οι διαστάσεις κάποιων διατομών να ανήκουν στις έγκυρες τιμές που μπορούν να λάβουν οι αντίστοιχες μεταβλητές, όμως να οδηγούν σε σχεδιασμό που παραβιάζει την ασφάλεια του φορέα. Για να αποκλείσουμε τέτοιους συνδυασμούς τιμών εφαρμόζουμε περιορισμούς στο πρόβλημα βελτιστοποίησης. Οι περιορισμοί αυτοί μπορεί να είναι ανισοτικοί ($g_j(\mathbf{x}) \leq 0$) ή ισοτικοί ($h_j(\mathbf{x}) = 0$). Ένας σχεδιασμός που ικανοποιεί όλους τους περιορισμούς λέγεται εφικτός (feasible design) και αποτελεί μια δυνατή, υλοποιήσιμη λύση του προβλήματος, που όμως δεν είναι απαραίτητα η βέλτιστη. Ένας σχεδιασμός που παραβιάζει τουλάχιστον έναν από τους περιορισμούς λέγεται ανέφικτός (infeasible design) και πρέπει να απορριφθεί. Έτσι, ο βέλτιστος σχεδιασμός αναζητείται ανάμεσα μόνο στους εφικτούς.

Στον σχεδιασμό κατασκευών οι περιορισμοί μπορούν να αντιπροσωπεύουν τις απαιτήσεις αντοχής και λειτουργικότητας που περιγράφονται στους κανονισμούς δόμησης, κατασκευαστικές απαιτήσεις, κ.τ.λ. Συνήθως ως περιορισμοί λαμβάνονται τα επιτρεπόμενα όρια τάσεων των υλικών και μετατοπίσεων των κόμβων των μελών του φορέα. Ο υπολογισμός των συναρτήσεων περιορισμών δεν μπορεί να γίνει άμεσα αν γνωρίζουμε τις τιμές των μεταβλητών σχεδιασμού. Πρέπει πρώτα να προηγηθεί η ανάλυση του φορέα που περιγράφεται από αυτόν τον σχεδιασμό. Η ανάλυση αυτή πλέον γίνεται με τη μέθοδο των πεπερασμένων στοιχείων και αποτελεί το μεγαλύτερο μέρος του υπολογιστικού χρόνου. Μετά την ανάλυση ακολουθεί ο υπολογισμός των μετατοπίσεων των κόμβων του δικτύου και των τάσεων των πεπερασμένων στοιχείων. Οι υπολογισμοί αυτοί όμως είναι χρονοβόροι σε πολύπλοκους φορείς με μεγάλο πλήθος κόμβων και στοιχείων, ακόμα και με τα σύγχρονα υπολογιστικά συστήματα. Επομένως είναι χρήσιμο να ελαττώσουμε τον αριθμό των υπολογισμών. Σε φορείς που παρουσιάζουν συμμετρία, αυτό μπορεί εύκολα να επιτευχθεί με ομαδοποίηση των περιορισμών που αναφέρονται σε συμμετρικά μέλη.

Οι ανισοτικές συναρτήσεις περιορισμών μπορεί να είναι ενεργές (active), ανενεργές (inactive) ή να παραβιάζονται. Μια ανισοτική συνάρτηση περιορισμού $g_j(\mathbf{x}) \leq 0$ είναι ανενεργή όταν ισχύει αυστηρά η ανισότητα $g_j(\mathbf{x}) < 0$. Αν ισχύει η ισότητα $g_j(\mathbf{x}) = 0$ τότε η συνάρτηση είναι ενεργή,

ενώ παραβιάζεται αν η αρχική ανισότητα δεν ισχύει καν, δηλαδή αν ισχύει $g_j(\mathbf{x}) > 0$. Οι ισοτικές συναρτήσεις περιορισμών μπορούν είτε να είναι ενεργές είτε να παραβιάζονται. Μια ισοτική συνάρτηση περιορισμών είναι ενεργή αν ισχύει η ισότητα $h_j(\mathbf{x}) = 0$, ενώ παραβιάζεται αν δεν ισχύει η ισότητα, δηλαδή αν $h_j(\mathbf{x}) \neq 0$.

Είναι δυνατόν ο μηχανικός να επιλέξει επιπλέον συναρτήσεις περιορισμών που δεν χρειάζονται, επειδή εξαρτώνται από άλλους περιορισμούς ή επειδή είναι ανενεργές σε όλα τα βήματα της βελτιστοποίησης και δεν επηρεάζουν την διαδοχική βελτίωση του σχεδιασμού. Αυτό μπορεί να οφείλεται είτε σε σφάλμα του μηχανικού κατά το σχεδιασμό του μοντέλου είτε στην ύπαρξη αβεβαιοτήτων σχετικά με τους περιορισμούς αυτούς. Αποτέλεσμα πάντως είναι η αύξηση του υπολογιστικού κόστους χωρίς κάποιο όφελος ως προς την σύγκλιση της μεθόδου. Ιδιαίτερα σε μεθόδους βελτιστοποίησης μαθηματικού προγραμματισμού, όπου πραγματοποιείται συνήθως χρονοβόρα ανάλυση ευαισθησίας σε κάθε βήμα, η αύξηση του υπολογιστικού κόστους λόγω περιττών περιορισμών είναι εντονότερη.

Η επιβολή μεγάλου αριθμού περιορισμών μπορεί επίσης να καταστήσει το πρόβλημα βελτιστοποίησης άλυτο. Αν οι περιορισμοί είναι αυστηροί, αφήνουν δηλαδή πολύ στενά περιθώρια ικανοποίησης τους, το πλήθος τους είναι μεγάλο ή υπάρχουν περιορισμοί που αλληλοσυγκρούονται, δηλαδή η ικανοποίηση του ενός οδηγεί στην παραβίαση του άλλου, τότε δεν είναι δυνατόν να βρεθεί λύση στο πρόβλημα. Στην περίπτωση αυτή λέμε ότι η περιοχή εφικτών σχεδιασμών είναι κενή. Για να αντιμετωπιστεί μια τέτοια κατάσταση πρέπει να μειώσουμε το πλήθος των περιορισμών, να επιτρέψουμε μεγαλύτερη ελαστικότητα στις απαιτήσεις που εκφράζουν ή να εντοπίσουμε και να αναθεωρήσουμε τους αλληλοσυγκρουόμενους περιορισμούς, πριν εφαρμόσουμε κάποια μέθοδο βελτιστοποίησης.

Μία συνάρτηση περιορισμών ανισοτήτων δεν είναι συνεχώς ενεργή. Σε πολλά βήματα του αλγορίθμου βελτιστοποίησης μπορεί να είναι εντός της ασφαλούς περιοχής, δηλαδή ανενεργή και άρα να μην επηρεάζει την βελτίωση του σχεδιασμού. Σε επόμενο βήμα μπορεί βέβαια να ξαναγίνει ενεργή, ενώ να απενεργοποιηθούν κάποιες άλλες ανισοτικές συναρτήσεις περιορισμών. Μπορούμε να γλιτώσουμε το υπολογιστικό κόστος που προκύπτει από τον υπολογισμό των τιμών και κυρίως των παραγώγων των συναρτήσεων περιορισμών, που είναι απαραίτητες στη φάση ανάλυσης ευαισθησίας πολλών μεθόδων μαθηματικού προγραμματισμού, αν αγνοήσουμε τους ανενεργούς περιορισμούς σε κάθε βήμα. Όμως δεν είναι γνωστό εκ των προτέρων ποιοι περιορισμοί είναι ανενεργοί σε κάθε βήμα. Επομένως ο αλγόριθμος βελτιστοποίησης θα πρέπει σε κάθε βήμα, να εντοπίζει τους ανενεργούς περιορισμούς προκειμένου να εκμεταλλευτεί την ιδιότητα αυτή. Πράγματι πολλοί μαθηματικοί αλγόριθμοι βελτιστοποίησης χρησιμοποιούν την τεχνική αυτή για να αυξήσουν την αποδοτικότητά τους. Αντίθετα, οι ισοτικές συναρτήσεις περιορισμών είναι είτε ενεργές είτε παραβιάζονται, οπότε δεν μπορούν να αγνοηθούν.

Όπως προαναφέρθηκε, μια συνάρτηση περιορισμού ανισότητας είναι ενεργή όταν ικανοποιείται η ισότητα $g_j(\mathbf{x}) = 0$. Πρακτικά όμως αυτό συμβαίνει σπάνια. Λόγω της στοχαστικής φύσης της προσομοίωσης και ανάλυσης των προβλημάτων που αφορούν

κατασκευές, μπορούμε να θεωρήσουμε ότι η μια συνάρτηση περιορισμού (συνήθως τάση ή παραμόρφωση), είναι ενεργή όταν η ισότητα ισχύει στο περίπου, δηλαδή $g_j(\mathbf{x}) \approx 0$, οπότε επιτρέπουμε ένα περιθώριο ανοχής. Παρόμοια περιθώρια ανοχής μπορεί να επιτρέπουμε και σε ισοτικές συναρτήσεις περιορισμών $h_j(\mathbf{x}) \approx 0$, προκειμένου να δεχθούμε ότι ικανοποιούνται. Το περιθώριο ανοχής αυτό εξαρτάται από την τάξη μεγέθους κάθε περιορισμού. Για παράδειγμα οι τάσεις είναι συνήθως τάξης μεγέθους 10 MPa ως 50 MPa ανάλογα με το υλικό, ενώ οι μετατοπίσεις κόμβων είναι της τάξης των 0.1 cm ως μερικά cm. Προκειμένου να θέσουμε ένα ενιαίο όριο για κρίνουμε πότε μια συνάρτηση περιορισμού είναι ανενεργή, ενεργή ή παραβιάζεται, πρέπει πρώτα να κανονικοποιηθεί η τιμή της.

Η κανονικοποίηση γίνεται με βάση την επιτρεπόμενη τιμή του περιορισμού ως εξής:

$$g_j^N(\boldsymbol{x}) = \frac{g_j^{min} - g_j(\boldsymbol{x})}{\left|g_j^{min}\right|} \leq 0, \qquad \textbf{(1.2)}$$

για έναν περιορισμό κάτω ορίου: $g_j(x) \geq g_j^{min}$ και

$$g_j^N(\boldsymbol{x}) = \frac{g_j(\boldsymbol{x}) - g_j^{max}}{\left|g_j^{max}\right|} \leq 0, \qquad \textbf{(1.3)}$$

για έναν περιορισμό κάτω ορίου: $g_j(x) \leq g_j^{max}$, όπου $g_j^N(\boldsymbol{x})$ είναι η κανονικοποιημένη τιμή.

Έτσι μια κανονικοποιημένη τιμή $g_j^N(\boldsymbol{x}) = 0.1$, θεωρείται ότι παραβιάζει τον περιορισμό κατά 10%, ενώ αν $g_j^N(\boldsymbol{x}) = -0.1$, τότε θεωρείται ότι βρίσκεται εντός της ασφαλούς περιοχής κατά 10%. Μια συνάρτηση περιορισμού μπορεί να θεωρηθεί ενεργή αν έχει κανονικοποιημένη τιμή από -0.1 ως 0. Επίσης μπορούμε να δεχθούμε και κανονικοποιημένες τιμές που παραβιάζουν ελάχιστα τον περιορισμό (π.χ. +0.001 ως +0.005), για να λάβουμε υπόψη τις αβεβαιότητες στην προσομοίωση και ανάλυση του φορέα. Οι τιμές των συναρτήσεων περιορισμών εξαρτώνται από το μοντέλο προσομοίωσης, που όμως δεν ταυτίζεται ακριβώς με τον φορέα που τελικά θα κατασκευαστεί, από την εντατική κατάσταση που προσεγγίζεται πιθανοτικά και από την ακρίβεια της ανάλυσης που δεν είναι απόλυτη καθώς χρησιμοποιούνται αριθμητικές μέθοδοι. Τα όρια που εκφράζουν οι περιορισμοί εξαρτώνται συνήθως από τις ιδιότητες των υλικών, οι οποίες παρουσιάζουν διακύμανση. Επομένως δεν έχει νόημα να απαιτείται απόλυτη ακρίβεια. Αντίθετα μια ελαστικότητα στους περιορισμούς μειώνει το υπολογιστικό κόστος και οδηγεί σε οικονομικότερους φορείς.

1.4 Κατηγορίες Μαθηματικών Προβλημάτων

Δεν υπάρχει αλγόριθμος βελτιστοποίησης που να μπορεί να επιλύσει όλα τα πιθανά προβλήματα. Οι περισσότεροι αλγόριθμοι μπορούν να εφαρμοστούν μόνο σε συγκεκριμένα είδη προβλημάτων βελτιστοποίησης, ενώ παρουσιάζουν καλή συμπεριφορά (συγκριτικά με άλλους αλγορίθμους) μόνο σε ένα υποσύνολο αυτών. Συχνά ένας αλγόριθμος που έχει αναπτυχθεί για ένα συγκεκριμένο είδος προβλήματος μπορεί να εφαρμοστεί για ένα άλλο, αλλά πρέπει να μετατραπεί πρώτα, ώστε να αντιμετωπιστούν οι διαφορές. Για να επιλέξουμε λοιπόν τον κατάλληλο αλγόριθμο, θα πρέπει να αναγνωρίσουμε τα χαρακτηριστικά του προβλήματος βελτιστοποίησης και να το κατατάξουμε σε κάποια κατηγορία. Η ταξινόμηση των προβλημάτων βελτιστοποίησης βασίζεται στη φύση των μεταβλητών, των αντικειμενικών συναρτήσεων και των περιορισμών τους. Μία βασική ταξινόμηση είναι η εξής:

1.4.1 Συνεχής ή διακριτή βελτιστοποίηση

Προβλήματα στα οποία οι μεταβλητές σχεδιασμού είναι συνεχείς πραγματικοί αριθμοί ονομάζονται συνεχή (continuous optimization problems). Συχνά όμως οι δυνατές τιμές των μεταβλητών ανήκουν σε ένα πεπερασμένο σύνολο πραγματικών ή ακέραιων αριθμών, οπότε μιλάμε για διακριτά προβλήματα βελτιστοποίησης (discrete optimization problems). Γενικά τα συνεχή προβλήματα είναι ευκολότερα στην επίλυσή τους, αφού οι αντικειμενικές συναρτήσεις και οι συναρτήσεις περιορισμών είναι συνεχείς. Αντίθετα η ασυνέχεια των διακριτών προβλημάτων εμποδίζει την λήψη πληροφοριών για την παράγωγο των συναρτήσεων στα σημεία που αυτές ορίζονται.

Ωστόσο τα τελευταία χρόνια έχουν αναπτυχθεί αλγόριθμοι βελτιστοποίησης σχεδιασμένοι ειδικά για τον χειρισμό διακριτών μεταβλητών. Σε συνδυασμό με την αύξησης της ισχύος των σύγχρονων υπολογιστικών συστημάτων, η πολυπλοκότητα των διακριτών προβλημάτων δεν εμποδίζει πλέον την αποδοτική επίλυσή τους. Μία τεχνική ευρείας εφαρμογής είναι η μετατροπή ενός διακριτού προβλήματος σε συνεχές, π.χ. αντιστοιχίζοντας διαστήματα αριθμών στις δυνατές διακριτές τιμές με στρογγυλοποίηση. Έτσι μπορεί να χρησιμοποιηθεί ένας αλγόριθμος βελτιστοποίησης που προορίζεται για συνεχή προβλήματα.

Συχνά εμφανίζονται προβλήματα που κάποιες από τις μεταβλητές είναι συνεχείς και άλλες διακριτές. Τα προβλήματα αυτά εντάσσονται στην διακριτή βελτιστοποίηση. Σημαντικές υποκατηγορίες της διακριτής βελτιστοποίησης είναι οι ακόλουθες.

1.4.1.1 Ακέραιος Προγραμματισμός

Στον ακέραιο προγραμματισμό (Integer Programming) κάποιες ή όλες οι μεταβλητές σχεδιασμού μπορούν να λάβουν μόνο ακέραιες τιμές. Περαιτέρω διακρίνουμε τις εξής περιπτώσεις:

- Γραμμικός ακέραιος προγραμματισμός (Integer Linear Programming – ILP) ή μη γραμμικός ακέραιος προγραμματισμός (Integer Non Linear Programming – INLP), όπου η αντικειμενική συνάρτηση και οι συναρτήσεις περιορισμών είναι γραμμικές ή μη γραμμικές αντίστοιχα.
- Αμιγώς ακέραιος προγραμματισμός (Pure Integer Programming) ή μεικτός ακέραιος προγραμματισμός (Mixed Integer Programming), όπου όλες ή μόνο κάποιες από τις μεταβλητές σχεδιασμού είναι ακέραιες αντίστοιχα.
- Δυαδική βελτιστοποίηση (Binary Optimization ή Zero-one programming), όπου οι μεταβλητές μπορούν να λάβουν μόνο τις τιμές 0 και 1. Οι ακέραιες και διακριτές μεταβλητές μπορούν να μετατραπούν σε δυαδικές.

1.4.1.2 Συνδυαστική Βελτιστοποίηση

Στη συνδυαστική βελτιστοποίηση (Combinatorial Optimization) ζητούμενο είναι η εύρεση του βέλτιστου συνδυασμού ή αλληλουχίας αντικειμένων από ένα πεπερασμένο σύνολο. Τα αντικείμενα αυτά δεν είναι απαραίτητα αριθμοί. Στα προβλήματα συνδυαστικής βελτιστοποίησης η εξάντληση των εναλλακτικών λύσεων είναι ασύμφορη. Ως παραδείγματα αναφέρονται το Πρόβλημα του Περιπλανώμενου Πωλητή (Traveling Salesman Problem) και το Πρόβλημα του Ελάχιστου Γεννητικού Δέντρου (Minimum Spanning Tree Problem). Η συνδυαστική βελτιστοποίηση βρίσκει πολλές εφαρμογές στην τεχνητή νοημοσύνη και στο σχεδιασμό μεταφορικών συστημάτων.

1.4.2 Βελτιστοποίηση Με ή Χωρίς Περιορισμούς

Οποιαδήποτε απαίτηση ενός σχεδιασμού δεν αφορά άμεσα μόνο μια μεταβλητή μοντελοποιείται ως περιορισμός. Τα περισσότερα πρακτικά προβλήματα με πολλαπλές απαιτήσεις έχουν περιορισμούς. Προφανώς ο βέλτιστος σχεδιασμός κατασκευών υπάγεται σε αυτήν την κατηγορία. Οι τιμές των συναρτήσεων περιορισμών μπορεί να υπολογίζονται απευθείας από τις μεταβλητές σχεδιασμού ή μετά από σημαντική επεξεργασία (π.χ. ανάλυση πεπερασμένων στοιχείων). Έχουν αναπτυχθεί διάφορες τεχνικές αντιμετώπισης των περιορισμών που συνήθως αποσκοπούν στην αναγωγή σε πρόβλημα βελτιστοποίησης χωρίς περιορισμούς. Περιορισμοί ισότητας μπορούν να μετατραπούν σε επιπλέον μεταβλητές σχεδιασμού μέσω των συντελεστών Lagrange. Μια πολύ συχνή τεχνική για όλες τις συναρτήσεις περιορισμών είναι η επιβολή ποινής στην τιμή της αντικειμενικής συνάρτησης, αν φυσικά παραβιάζεται κάποιος περιορισμός.

Προβλήματα χωρίς περιορισμούς εμφανίζονται σε αρκετές πρακτικές εφαρμογές ή από την αίρεση των περιορισμών με τις προαναφερθείσες τεχνικές. Μία σημαντική κατηγορία προβλημάτων χωρίς περιορισμούς είναι η βελτιστοποίηση των παραμέτρων μιας υπάρχουσας μεθόδου βελτιστοποίησης (meta-optimization). Πολλές μέθοδοι βελτιστοποίησης έχουν παραμέτρους που επηρεάζουν τη συμπεριφορά και την αποδοτικότητά τους και τίθενται από τον χρήστη της μεθόδου. Έτσι είναι δυνατό το καλιμπράρισμα της μεθόδου για συγκεκριμένα είδη προβλημάτων. Χρησιμοποιώντας μια άλλη μέθοδο βελτιστοποίησης ή και την ίδια μπορούμε να αυτοματοποιήσουμε την ρύθμιση των παραμέτρων.

Στα προβλήματα βέλτιστου σχεδιασμού κατασκευών έχουμε πάντα περιορισμούς που συνήθως εκφράζουν τις απαιτήσεις αντοχής και λειτουργικότητας ενός φορέα. Η σημασία των περιορισμών είναι μεγάλη, καθώς το μεγαλύτερο μέρος του υπολογιστικού χρόνου αφιερώνεται στη μόρφωση και ανάλυση του μοντέλου πεπερασμένων στοιχείων, οι οποίες είναι απαραίτητες πριν υπολογιστούν οι συναρτήσεις περιορισμών.

1.4.3 Καμία, Μία ή Περισσότερες Αντικειμενικές Συναρτήσεις

Τα περισσότερα προβλήματα βελτιστοποίησης έχουν ως ζητούμενο την εύρεση της βέλτιστης τιμής μίας ή περισσότερων αντικειμενικών συναρτήσεων. Ωστόσο υπάρχουν ενδιαφέρουσες περιπτώσεις προβλημάτων που δεν έχουν κάποια αντικειμενική συνάρτηση. Στα προβλήματα εφικτότητας (feasiblility problems) σκοπός είναι η εύρεση τιμών για τις μεταβλητές που να ικανοποιούν τους περιορισμούς του προβλήματος, χωρίς να υπάρχει κάποιο κριτήριο αξιολόγησης των εφικτών λύσεων που θα προκύψουν. Ιδιαίτερα συχνά στην επιστήμη των οικονομικών και της μηχανικής είναι τα προβλήματα συμπληρωματικότητας (complementarity problems), όπου ζητούμενο είναι η εύρεση λύσης που να ικανοποιεί τις συμπληρωματικές συνθήκες. Οι συμπληρωματικές συνθήκες σε αυτά τα προβλήματα απαιτούν το γινόμενο δύο μη αρνητικών μεταβλητών (ή διανυσμάτων με μη αρνητικές μεταβλητές) να είναι 0, επομένως αναζητούν ορθογωνικές λύσεις.

Τα βασικά προβλημάτα βελτιστοποίησης είναι μονοκριτηριακά (Single Objective Optimization), ασχολούνται δηλαδή με την ελαχιστοποίηση ή μεγιστοποίηση μίας μόνο αντικειμενικής συνάρτησης ή περισσότερων που να βελτιστοποιούνται ταυτόχρονα. Η δυσκολία αυτών των προβλημάτων εξαρτάται από την γραμμικότητα, την παραγωγισιμότητα και την κυρτότητα της συνάρτησης αυτής. Προβλήματα με περισσότερες αντικειμενικές συναρτήσεις συχνά ανάγονται σε μονοκριτηριακά προβλήματα ώστε να μπορούν να επιλυθούν με αλγορίθμους βελτιστοποίησης που χειρίζονται μία μόνο αντικειμενική συνάρτηση.

Στην πολυκριτηριακή βελτιστοποίηση (Multi Objective Optimization) εντάσσονται τα προβλήματα που έχουν ως ζητούμενο την εύρεση λύσης, που να βελτιστοποιεί ταυτόχρονα δύο ή περισσότερες αντικειμενικές συναρτήσεις, οι οποίες όμως αλληλοσυγκρούονται, δηλαδή η μείωση μίας συνάρτησης προκαλεί αύξηση μιας άλλης. Σε πρακτικές εφαρμογές επικρατεί η ύπαρξη ασυμβίβαστων κριτηρίων. Για παράδειγμα ο μηχανικός καλείται συχνά να

αντιμετωπίσει προβλήματα όπου η ελαχιστοποίηση του βάρους μιας κατασκευής ή ενός εξαρτήματος εμφανίζεται ταυτόχρονα με την μεγιστοποίηση της μηχανικής αντοχής .

Στα μη τετριμμένα πολυκριτηριακά προβλήματα δεν υπάρχει μοναδική λύση. Αντίθετα αναζητείται ένα σύνολο λύσεων (Parento Optimal Solutions) που επιτυγχάνουν συμβιβασμό μεταξύ των διαφόρων κριτηρίων. Οι λύσεις αυτές δεν μπορούν βελτιωθούν άλλο ως προς κάποιο κριτήριο χωρίς να υποβαθμίσουν ένα άλλο. Επιπλέον θεωρούνται ισότιμες και μπορεί να είναι άπειρες. Για την αντιμετώπιση τους έχουν αναπτυχθεί διάφορες τεχνικές. Πολλές μέθοδοι προσπαθούν να μετατρέψουν το πολυκριτηριακό πρόβλημα σε μονοκριτηριακό (scalarized problem). Αυτό μπορεί να γίνει εύκολα με την εφαρμογή μίας αντικειμενικής συνάρτησης που υπολογίζει το σταθμισμένο άθροισμα των ζητούμενων κριτηρίων χρησιμοποιώντας κάποιους συντελεστές βάρους (linear scalarization):

$$F(\mathbf{x}) = \Sigma\{w_i * f_i(\mathbf{x})\} \rightarrow \min \quad \textbf{(1.4)}$$

Μια εναλλακτική τεχνική είναι να χρησιμοποιηθεί μόνο ένα από τα κριτήρια ως αντικειμενική συνάρτηση. Τα υπόλοιπα μετατρέπονται σε συναρτήσεις περιορισμών με άνω όρια παραμέτρους ε_i (ε-constrained method) :

$$\begin{gathered} f_j(\mathbf{x}) \rightarrow \min \\ f_i(\mathbf{x}) - \varepsilon_i \leq 0,\ i \neq j \end{gathered} \quad \textbf{(1.5)}$$

1.4.4 Ντετερμινιστική ή Στοχαστική Βελτιστοποίηση

Οι βασικές μέθοδοι βελτιστοποίησης θεωρούν ότι τα δεδομένα του προβλήματος, δηλαδή ο χώρος τιμών των μεταβλητών και τα όρια των περιορισμών, είναι γνωστά με ακρίβεια και σταθερά και ότι ο υπολογισμός της αντικειμενικής συνάρτησης και των συναρτήσεων περιορισμών είναι ντετερμινιστικές διαδικασίες, δηλαδή για τα ίδια επιχειρήματα θα επιστρέφονται τα ίδια αποτελέσματα κάθε φορά. Οι αλγόριθμοι που βασίζονται σε τέτοιες υποθέσεις ονομάζονται αιτιοκρατικοί (Deterministic Optimization).

Στην πραγματικότητα όμως οι υποθέσεις αυτές ισχύουν μόνο υπό ιδανικές συνθήκες. Στην επιστήμη του μηχανικού συγκεκριμένα υπάρχουν πολλές αβεβαιότητες. Οι τιμές των μεταβλητών σχεδιασμού που βρίσκονται κατά την μελέτη δεν συμπίπτουν απαραίτητα με τα αντίστοιχα κατασκευαστικά μεγέθη, αφού ο ανθρώπινος παράγοντας και οι δυνατότητες των κατασκευαστικών μέσων αναπόφευκτα εισάγουν ατέλειες στα μέλη του φορέα. Ακόμα και όταν οι μεταβλητές αντιπροσωπεύουν τυποποιημένες διατομές, δηλαδή διακριτές τιμές, τα γεωμετρικά τους χαρακτηριστικά μπορεί να παρουσιάζουν κάποια απόκλιση από τα προβλεπόμενα. Οι ιδιότητες των δομικών υλικών, από τις οποίες εξαρτάται το βάρος, αλλά και τα όρια των περιορισμών τάσεων και μετακινήσεων, δεν είναι εξασφαλισμένες. Συνήθως το βάρος και κυρίως η αντοχή παρουσιάζουν σημαντική διακύμανση, γεγονός που φαίνεται και στους κανονισμούς που θεσπίζουν ως χαρακτηριστική τιμή αυτή που έχει υπέρβαση 95%. Αν το μοντέλο προσομοίωσης περιλαμβάνει και το έδαφος τότε οι αβεβαιότητες κλιμακώνονται.

Επιπλέον η φόρτιση δεν είναι ποτέ γνωστή, ιδιαίτερα στα δυναμικά φαινόμενα. Η κατανομή των στατικών φορτίων, τα φορτία κυκλοφορίας μιας γέφυρας και η ένταση, κατεύθυνση και διάρκεια ενός σεισμού βασίζονται αναγκαστικά σε πιθανοτικές θεωρήσεις. Τέλος πρέπει να σημειωθεί ότι οι προαναφερθείσες αβεβαιότητες ισχύουν κατά τη φάση μόρφωσης του μοντέλου προσομοίωσης, αλλά και κατά την διάρκεια της επίλυσης, αφού τα μεγέθη αυτά σπάνια παρουσιάζουν χρονική σταθερότητα.

Πολλές από αυτές τις διακυμάνσεις έχουν σημαντική επίδραση στις τιμές της αντικειμενικής συνάρτησης και των συναρτήσεων περιορισμών, με αποτέλεσμα η λύση που προκύπτει από την βελτιστοποίηση να μην είναι πράγματι η καλύτερη ή ακόμα και να μην είναι εφικτή λόγω υπέρβασης της ασφάλειας. Για την αντιμετώπιση των αβεβαιοτήτων έχουν αναπτυχθεί οι κλάδοι της Βελτιστοποίησης Εύρωστου Σχεδιασμού (Robust Design Optimization) και Βελτιστοποίησης με βάση την Αξιοπιστία (Reliability Based Design Optimization). Οι μέθοδοι βελτιστοποίησης αυτής της κατηγορίας επιχειρούν να άρουν τις αβεβαιότητες του μοντέλου. Τυπικά μελετούν τις χειρότερες εναλλακτικές περιπτώσεις (worst-case scenario) και χρησιμοποιώντας το κριτήριο του Wald (Wald's maximin rule).

Μια εναλλακτική προσέγγιση δίνει ο στοχαστικός προγραμματισμός (Stochastic Programming). Οι μέθοδοι αυτής της κατηγορίας εκμεταλλεύονται το γεγονός ότι οι μη ντετερμινιστικές τιμές (π.χ. μεταβλητών) υπακούν σε κάποια πιθανοτική κατανομή, που είναι γνωστή ακριβώς ή μπορεί να προσεγγιστεί. Σκοπός είναι βελτιστοποίηση της αναμενόμενης τιμής της αντικειμενικής συνάρτησης.

Στον κλάδο του δομοστατικού σχεδιασμού οι αβεβαιότητες σε φορτίσεις και υλικά δεν λαμβάνονται υπόψη άμεσα. Οι περισσότεροι κανονισμοί (π.χ. ΕΑΚ) έχουν τη φιλοσοφία της σχεδίασης του φορέα για τις οριακές καταστάσεις αντοχής και λειτουργικότητας, χρησιμοποιώντας συντελεστές για το χειρισμό των μη γραμμικών και δυναμικών (σεισμός) φαινομένων που παρουσιάζουν αβεβαιότητες. Ωστόσο η εφαρμογή βελτιστοποίησης εύρωστου σχεδιασμού παρέχει πιο οικονομικές λύσεις από τις αντίστοιχες αιτιοκρατικές που ακολουθούν την φιλοσοφία αυτή (Πλεύρης Β., Λαγαρός Ν., Παπαδρακάκης Μ., 2008).

1.5 Κατηγορίες Βελτιστοποίησης Κατασκευών

Τα προβλήματα βέλτιστου σχεδιασμού κατασκευών χαρακτηρίζονται συνήθως από α) μεγάλο πλήθος συνεχών, διακριτών ή και των δύο ειδών μεταβλητών σχεδιασμού, β) μία απλή αντικειμενική συνάρτηση και γ) έμμεσες συναρτήσεις περιορισμών, που δεν μπορούν να υπολογιστούν απευθείας αλλά χρειάζεται ανάλυση του στατικού μοντέλου πρώτα. Ανεξάρτητα από την προηγούμενη διακριτοποίηση, μπορούμε να αναγνωρίσουμε τρεις μεγάλες κατηγορίες προβλημάτων βέλτιστου σχεδιασμού κατασκευών (structural design optimization):

- Βελτιστοποίηση του μεγέθους των διατομών
- Βελτιστοποίηση του σχήματος
- Βελτιστοποίηση της τοπολογίας

1.5.1 Βελτιστοποίηση Μεγέθους Διατομών (Sizing Optimization)

Στα προβλήματα αυτά τα μέλη του φορέα (δοκοί, κολώνες, πλάκες, κελύφη, κτλ.) και οι κόμβοι σύνδεσής τους είναι γνωστά πριν την διαδικασία της βελτιστοποίησης. Ζητούμενο είναι η διαστασιολόγηση τους, δηλαδή η εύρεση των διατομών που να ελαχιστοποιούν το συνολικό βάρος του φορέα χωρίς να παραβιάζεται η ασφάλεια. Είναι η πρώτη χρονικά κατηγορία προβλημάτων βελτιστοποίησης που ασχολήθηκαν οι μηχανικοί, ιδιαίτερα σε δικτυωτούς φορείς.

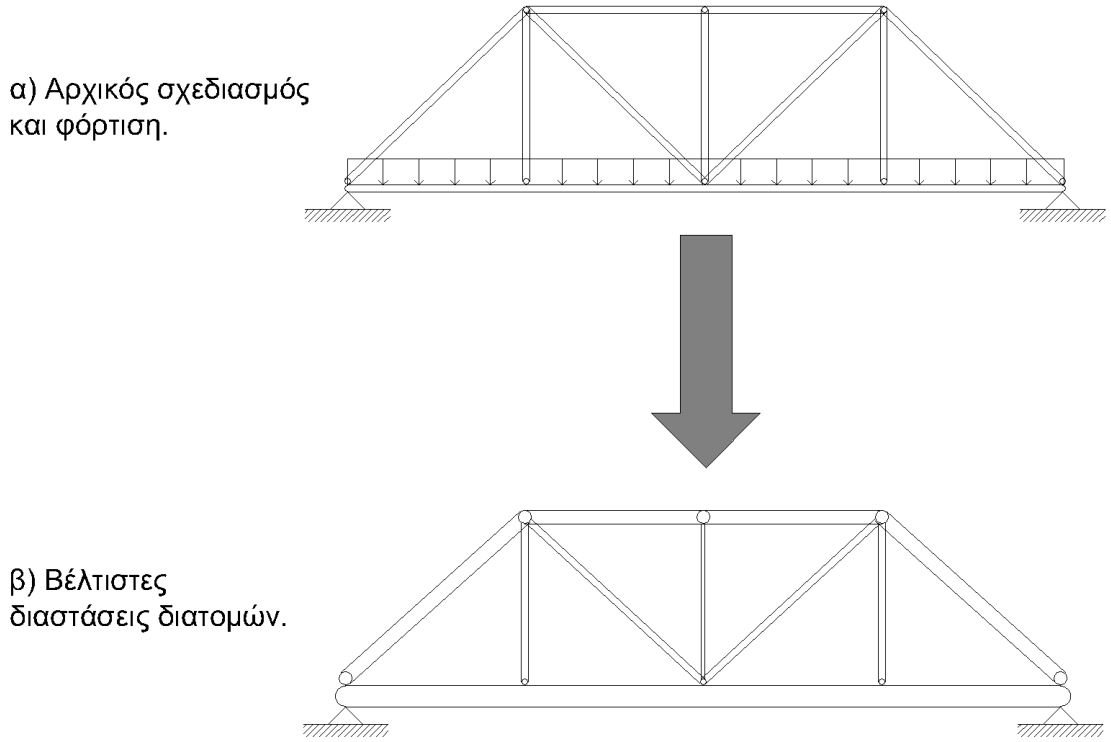
Σε ένα τυπικό πρόβλημα βελτιστοποίησης μεγέθους διατομών, οι μεταβλητές σχεδιασμού αντιπροσωπεύουν τις διαστάσεις των διατομών των μελών του φορέα. Ως αντικειμενική συνάρτηση λαμβάνεται συνήθως το βάρος, εκτός και αν το κόστος της κατασκευαστικής διαδικασίας είναι εύκολο να υπολογισθεί, εφόσον οι διαστάσεις των μελών είναι γνωστές. Οι συναρτήσεις περιορισμών μοντελοποιούν τις απαιτήσεις αντοχής και λειτουργικότητας, οπότε συνήθως διατυπώνονται σε όρους τάσεων στα στοιχεία του φορέα και μετακινήσεων κάποιων ή όλων από τους κόμβους του.

Στα παραδείγματα βελτιστοποίησης μεγέθους διατομών συγκαταλέγεται η εύρεση των βέλτιστων διαστάσεων και οπλισμών των ορθογωνικών διατομών σε φορείς από οπλισμένο σκυρόδεμα, η επιλογή των κατάλληλων τυποποιημένων διατομών σε δικτυωτές ή πλαισιακές μεταλλικές κατασκευές και η ελαχιστοποίηση του πάχους των πλακών ή κελυφών. Στην παρούσα εργασία οι αριθμητικές εφαρμογές, που περιγράφονται στη συνέχεια, ανήκουν σε αυτήν την κατηγορία προβλημάτων.

Φαίνεται λοιπόν ότι οι μεταβλητές σχεδιασμού μπορούν να είναι είτε συνεχής είτε διακριτές, ενώ δεν είναι σπάνιο να συνυπάρχουν και τα δύο είδη στο ίδιο πρόβλημα (π.χ. διακριτές μεταβλητές για τυποποιημένες διατομές δοκών και συνεχείς μεταβλητές για τα πάχη πλακών). Συνήθως τα μέλη του φορέα χωρίζονται σε ομάδες με κοινή διατομή. Η ομαδοποίηση αυτή εξυπηρετεί πρακτικούς σκοπούς. Βέβαια έτσι περιορίζεται η δυνατότητα οικονομίας υλικού. Παρόλα αυτά, το συνολικό κόστος της δόμησης του φορέα μπορεί να μειώνεται, αφού η συμμετρία και ομοιομορφία του διευκολύνει την κατασκευή του ή και την παραγγελία προκατασκευασμένων μελών που θα συναρμολογηθούν στο εργοτάξιο. Ταυτόχρονα ελαττώνει το πλήθος των μεταβλητών σχεδιασμού, με αποτέλεσμα να μειώνεται το υπολογιστικό κόστος της βελτιστοποίησης.

Προαπαιτούμενο για τη διαδικασία βελτιστοποίησης είναι η περιγραφή της γεωμετρίας του φορέα, δηλαδή των μελών του και των κόμβων σύνδεσης, των στηρίξεων και των επιβαλλόμενων φορτίων. Έπειτα ο μηχανικός καθορίζει ποιές διαστάσεις διατομών θα αποτελέσουν μεταβλητές σχεδιασμού και πώς το μοντέλο προσομοίωσης θα παραχθεί από αυτές με αυτόματη διαδικασία. Επίσης απαιτείται προσδιορισμός του βάρους της κατασκευής ως αντικειμενική συνάρτηση και των τάσεων και παραμορφώσεων ως συναρτήσεων περιορισμών. Το βάρος της κατασκευής μπορεί να υπολογισθεί γενικά άμεσα για κάθε συνδυασμό διαστάσεων διατομών. Οι περιορισμοί όμως απαιτούν ανάλυση του μοντέλου

προσομοίωσης, που συνήθως πραγματοποιείται με τη μέθοδο των πεπερασμένων στοιχείων, χωρίς να αποκλείονται μέθοδοι μητρωικής στατικής για απλούς ραβδόμορφους φορείς. Τέλος επιλέγεται και εφαρμόζεται ο κατάλληλος αλγόριθμος βελτιστοποίησης.



Σχήμα 1.3 Βελτιστοποίηση μεγέθους διατομών σε γέφυρα τύπου δικτυώματος Pratt.

1.5.2 Βελτιστοποίηση Σχήματος (Shape Optimization)

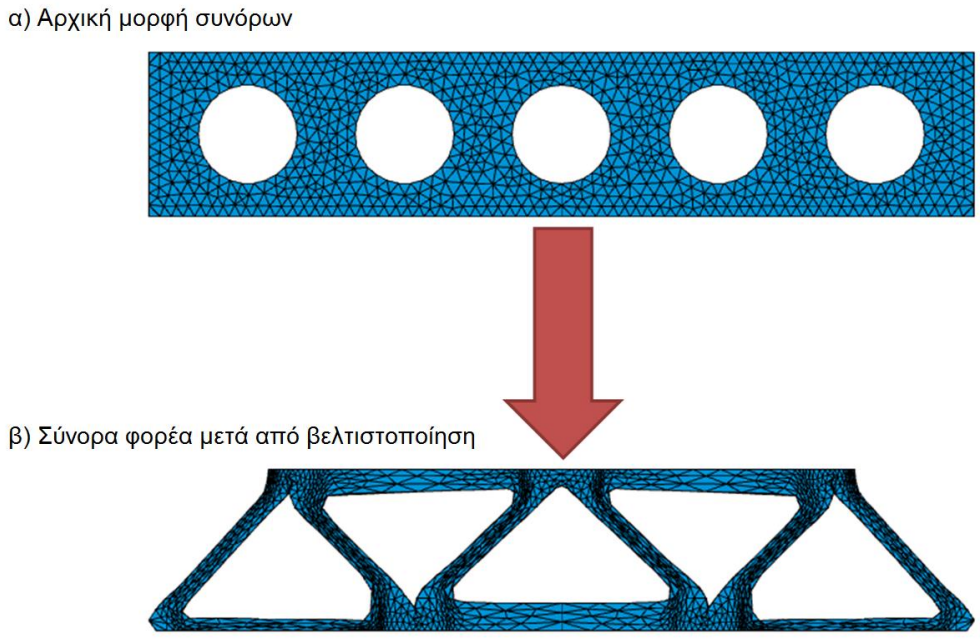
Αυτός ο κλάδος προβλημάτων ασχολείται με την βελτιστοποίηση των εσωτερικών και εξωτερικών περιγραμμάτων του φορέα, προκειμένου να επιτευχθεί οικονομία υλικού. Σχετικές μεθοδολογίες αναπτύχθηκαν μεταγενέστερα της βελτιστοποίησης μεγέθους διατομών. Αυτό οφείλεται τόσο στην μεγαλύτερη πολυπλοκότητα των προβλημάτων όσο και στην απαίτηση σε υπολογιστική ισχύ. Σήμερα έχουν αναπτυχθεί εξελιγμένες τεχνικές μοντελοποίησης πολύπλοκων γεωμετριών, που υποστηρίζονται από τα σύγχρονα υπολογιστικά συστήματα.

Η βελτιστοποίηση σχήματος μπορεί να επιτύχει ιδιαίτερα καλή εκμετάλλευση του υλικού ειδικά αν συνδυασθεί με βελτιστοποίηση μεγέθους ή τοπολογίας. Όμως σε συμβατικές κατασκευές Πολιτικού Μηχανικού η γεωμετρία των φορέων είναι συνήθως προκαθορισμένη και ακολουθεί συνήθως εμπειρικές μεθοδολογίες, την φιλοσοφία των κανονισμών και

πρακτικές που εξυπηρετούν την οικονομική και εύκολη δόμηση του φορέα. Η βελτιστοποίηση σχήματος δεν εφαρμόζεται ευρέως στο σύνολο του φορέα, αλλά πιθανόν να είναι αποτελεσματική για μεμονωμένες λεπτομέρειες όπως οπές.

Σε ένα πρόβλημα βελτιστοποίησης σχήματος μεταβλητές σχεδιασμού είναι οι συντεταγμένες των κόμβων του εξωτερικού συνόρου και πιθανών εσωτερικών. Το εξωτερικό περίγραμμα μεταβάλλεται και στην βελτιστοποίηση μεγέθους διατομών. Εδώ όμως η μεταβολή δεν είναι απαραίτητα ομοιόμορφη, με αποτέλεσμα το σύνορο να παραμορφωθεί ώστε να ακολουθεί καλύτερα τις ροές τάσεων. Ως αντικειμενική συνάρτηση θεωρείται το βάρος και ως συναρτήσεις περιορισμών οι τάσεις και παραμορφώσεις του φορέα, όπως και στη βελτιστοποίηση μεγέθους.

Η βελτιστοποίηση σχήματος πάντως δεν μπορεί να εφαρμοστεί απευθείας. Και εδώ είναι απαραίτητη η περιγραφή της γεωμετρίας του φορέα. Το σχήμα των συνόρων προσδιορίζεται τελικά από τη μέθοδο βελτιστοποίησης, όμως είναι απαραίτητη μια αρχική εκτίμηση. Ο μηχανικός θα πρέπει να περιγράψει τις καμπύλες των περιγραμμάτων και να τις εξαρτήσει από σημεία ελέγχου, οι συντεταγμένες των οποίων θα είναι οι μεταβλητές σχεδιασμού. Για τα εσωτερικά σύνορα ειδικά χρειάζεται ρητός καθορισμός του πλήθους και της αρχικής θέσης τους. Έπειτα πρέπει να προσδιορισθούν οι στηρίξεις και τα φορτία του φορέα. Για να ελεγχθούν οι περιορισμοί του προβλήματος γίνεται ανάλυση με τη μέθοδο των πεπερασμένων στοιχείων, η οποία προαπαιτεί την γένεση δικτύου (mesh), που τώρα έχει μεταβλητά σύνορα. Τέλος επιλέγεται κατάλληλος αλγόριθμος βελτιστοποίησης. Σε αντίθεση με τα προβλήματα βελτιστοποίησης μεγέθους διατομών, σε αυτήν την περίπτωση οι μεταβλητές σχεδιασμού δεν είναι γενικώς διακριτές, οπότε δεν αποκλείονται αλγόριθμοι που είναι αποτελεσματικοί μόνο σε συνεχή προβλήματα.



Σχήμα 1.4 Βελτιστοποίηση σχήματος σε δοκό με οπές.

1.5.3 Βελτιστοποίηση Τοπολογίας (Topology Optimization)

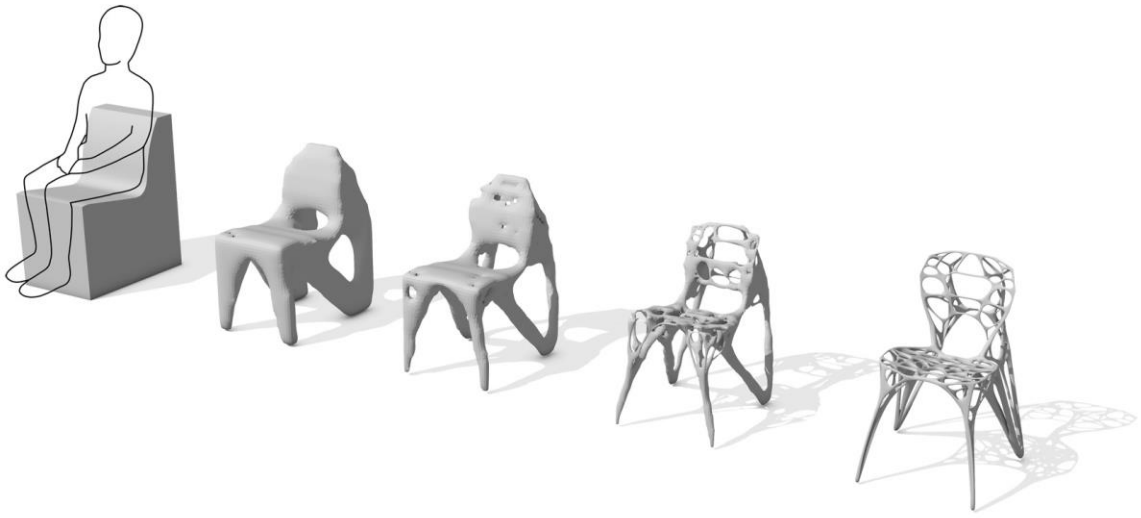
Η βελτιστοποίηση τοπολογίας αποσκοπεί στην σύλληψη της μορφής και διάταξης (layout) του φορέα που να ανταποκρίνεται όσο το δυνατόν αποτελεσματικότερα στις συνθήκες στήριξης και φόρτισης. Μεθοδολογίες για την αντιμετώπιση προβλημάτων βελτιστοποίησης τοπολογίας έχουν αναπτυχθεί συστηματικά τα τελευταία μόλις χρόνια. Οι μέθοδοι βελτιστοποίησης σχήματος έχουν εφαρμογή σε προβλήματα με καθορισμένη γεωμετρία, δηλαδή έχει δοθεί συνήθως η γενική μορφή του περιγράμματος, το πλήθος και οι θέσεις των οπών. Αντίθετα στη βελτιστοποίηση τοπολογίας αυτά δεν είναι δεδομένα.

Οι διάφορες μεθοδολογίες καταλήγουν στην εύρεση μιας γεωμετρίας για τον φορέα ξεκινώντας από ένα πολύ απλό γεωμετρικό σχήμα, όπως ένα πρίσμα. Διαδοχικά αφαιρούνται οι περιοχές υλικού που συμβάλλουν λιγότερο στην ανάληψη της φόρτισης. Η τελική διάταξη που προκύπτει μπορεί να ληφθεί ως σημείο έναρξης για περαιτέρω βελτίωση μέσω αλγορίθμων βελτιστοποίησης σχήματος. Η αρχική αυτή γεωμετρία πάντως, αν και επιτυγχάνει βέλτιστη εκμετάλλευση του διατιθέμενου υλικού, συχνά δεν είναι εφαρμόσιμη ή είναι αντιοικονομική λόγω δυσκολίας κατασκευής της. Για την αποφυγή τέτοιων καταστάσεων μπορούν να επιβληθούν κατασκευαστικοί περιορισμοί που αντικατοπτρίζουν τις δυνατότητες των μέσων παραγωγής.

Κάποιες από τις πλέον διαδεδομένες μεθοδολογίες για την επίλυση προβλημάτων βελτιστοποίησης τοπολογίας είναι οι:

- Homogenization method (Bendsoe and Kikuchi, 1988)
- Solid Isotropic Material with Penalisation (SIMP) (Bendsoe 1989, Buhl et al. 2000, Rozvany 2001)
- Simultaneous Analysis and Design (SAND) (Haftka 1985, Orozco 1997)
- Evolutionary Structural Optimization (ESO) (Xie and Stephen 1993)
- Fully Stressed Design (FSD) (Topping 1983, Hinton and Sienz 1993, Papadrakakis et al. 1996)

Η τελευταία τεχνική δεν είναι μέθοδος βελτιστοποίησης κατά την συμβατική έννοια (μεταβλητές σχεδιασμού, αντικειμενική συνάρτηση, περιορισμοί). Είναι όμως ιδιαίτερα αποτελεσματική αφού συγκλίνει με λίγες επαναλήψεις αφαιρώντας σταδιακά τις περιοχές υλικού που δεν εντείνονται πλήρως (hard kill). Είναι επίσης αρκετά απλή στην εφαρμογή της, καθώς δεν απαιτείται υπολογισμός παραγώγων, ενώ μπορεί να εφαρμοστεί εύκολα σε προβλήματα δύο και τριών διαστάσεων. Η αυτονομία από υπολογισμούς παραγώγων επιτρέπει την εφαρμογή μεταευριστικών αλγορίθμων βελτιστοποίησης, όπως η Evolutionary Strategies (Papadrakakis et al. 1998). Έτσι αντιμετωπίζονται και προβλήματα με διακριτούς χώρους τιμών των μεταβλητών, που εμφανίζονται αρκετά συχνά στο βέλτιστο σχεδιασμό κατασκευών. Ωστόσο είναι πιθανόν η μέθοδος FSD να έχει μειωμένη απόδοση, αν εφαρμοστεί χωρίς τροποποίηση σε προβλήματα με πολλαπλά υλικά, με μετακινήσεις ως περιορισμούς ή με έντονη υπερστατικότητα, η οποία συνεπάγεται πολλές εναλλακτικές τροχιές τάσεων.



Σχήμα 1.5 Βελτιστοποίηση τοπολογίας σε καρέκλα Generico. Ο τελικός φορέας μπορεί να κατασκευαστεί εύκολα κάνοντας χρήση της τεχνολογίας 3D printing.

1.5.4 Συνδυασμοί

Δεν αποκλείεται ο συνδυασμός των παραπάνω προβλημάτων βέλτιστου σχεδιασμού κατασκευών. Συχνά εκτελείται βελτιστοποίηση τοπολογίας ώστε να καθοριστεί η αρχική γεωμετρία του φορέα. Με δεδομένη αυτήν μπορούμε έπειτα να συνεχίσουμε σε βελτιστοποίηση σχήματος για το ραφινάρισμα των συνόρων ή να διακρίνουμε τον φορέα σε μέλη και να εφαρμόσουμε βελτιστοποίηση μεγέθους διατομών για την επιλογή των διαστάσεων τους. Επομένως οι παραπάνω διαφορετικές μέθοδοι εφαρμόζονται διαδοχικά. Η ταυτόχρονη εφαρμογή τους είναι πιο πολύπλοκη αλλά όχι αδύνατη. Για παράδειγμα αρκετές έρευνες (π.χ. Ahrari, Atai & Deb, 2014) έχουν αφιερωθεί στην ταυτόχρονη βελτιστοποίηση τοπολογίας, σχήματος και μεγέθους για δικτυώματα, χρησιμοποιώντας συνήθως στοχαστικούς αλγορίθμους βελτιστοποίησης.

1.6 Αλγόριθμοι Βελτιστοποίησης

Δεν υπάρχει αλγόριθμος βελτιστοποίησης που να μπορεί να επιλύσει αποδοτικά όλες τις κατηγορίες προβλημάτων. Η αποτελεσματικότητα ενός αλγορίθμου ποικίλει ανάλογα με τα χαρακτηριστικά του κάθε προβλήματος. Οι περισσότερες μέθοδοι βελτιστοποίησης αναπτύσσονται και δοκιμάζονται για μια συγκεκριμένη κατηγορία προβλημάτων. Προκείμενου να εφαρμοστούν σε διαφορετικά προβλήματα, πρέπει να αναπροσαρμοστούν, κάτι που δεν είναι πάντα δυνατό, τουλάχιστον χωρίς να αλλάξει η λογική του αλγορίθμου. Έτσι είναι αναμενόμενο να έχει αναπτυχθεί μεγάλο πλήθος διαφορετικών αλγορίθμων βελτιστοποίησης.

Ένας γενικός διαχωρισμός των μεθόδων που χρησιμοποιούνται στο βέλτιστο σχεδιασμό κατασκευών είναι:

- Μαθηματικές μέθοδοι
- Μεταευριστικές μέθοδοι

1.6.1 Μαθηματικές Μέθοδοι

Ο κλάδος των εφαρμοσμένων μαθηματικών που ασχολείται με την αντιμετώπιση προβλημάτων βελτιστοποίησης ονομάζεται Μαθηματικός Προγραμματισμός (Mathematical Programming). Βασίστηκε στον λογισμό των μεταβολών (calculus of variations) και αναπτύχθηκε παράλληλα με την οικονομική επιστήμη και την επιχειρησιακή έρευνα. Τον $20^{ο}$ αιώνα οι μηχανικοί ξεκινούν για πρώτη φορά να εφαρμόζουν μεθόδους του Μαθηματικού Προγραμματισμού στον βέλτιστο σχεδιασμό κατασκευών.

Κάποιες σημαντικές υποκατηγορίες του Μαθηματικού Προγραμματισμού είναι:

- Κυρτός Προγραμματισμός (Convex Programming). Μελετώνται προβλήματα βελτιστοποίησης, στα οποία η αντικειμενική συνάρτηση και οι συναρτήσεις περιορισμών είναι κυρτές. Οι μεταβλητές σχεδιασμού επίσης ανήκουν σε κυρτά υποσύνολα του διανυσματικού χώρου. Τέτοια προβλήματα παρουσιάζουν την ευκολία, ότι ένα τοπικό ελάχιστο είναι και ολικό.
- Γραμμικός Προγραμματισμός (Linear Programming). Αποτελεί ειδική περίπτωση του κυρτού προγραμματισμού, όπου η αντικειμενική συνάρτηση και οι συναρτήσεις περιορισμών είναι γραμμικές, οπότε η περιοχή εφικτών σχεδιασμών είναι κυρτό πολύεδρο. Η βέλτιστη λύση αναζητείται στο όριο του πολυέδρου και συνήθως στις γωνίες του.
- Τετραγωνικός Προγραμματισμός (Quadratic Programming). Στα προβλήματα αυτά η αντικειμενική συνάρτηση είναι δευτέρας τάξης, ενώ οι περιορισμοί είναι γραμμικές συναρτήσεις των μεταβλητών σχεδιασμού. Για συγκεκριμένες μορφές της αντικειμενικής συνάρτησης το πρόβλημα βελτιστοποίησης μπορεί να είναι κυρτό.
- Γεωμετρικός Προγραμματισμός (Geometric Programming). Στα γεωμετρικά προβλήματα βελτιστοποίησης, οι μεταβλητές σχεδιασμού είναι θετικοί αριθμοί, η αντικειμενική συνάρτηση και οι συναρτήσεις περιορισμών ανισοτήτων είναι πολυώνυμα, ενώ οι συναρτήσεις περιορισμών ισοτήτων είναι μονώνυμα. Τα προβλήματα αυτά γενικά δεν είναι κυρτά. Μπορούν όμως να μετατραπούν σε κυρτά με κατάλληλη τροποποίηση των μεταβλητών και συναρτήσεων τους.

- Μη Γραμμικός Προγραμματισμός (Non Linear Programming). Εδώ ανήκουν οι αλγόριθμοι που αντιμετωπίζουν προβλήματα βελτιστοποίησης στα οποία κάποιες συναρτήσεις περιορισμών ή αντικειμενικές είναι μη γραμμικές. Η κατηγορία αυτή έχει τη μεγαλύτερη σημασία για τον βέλτιστο σχεδιασμό κατασκευών, αφού οι τάσεις και παραμορφώσεις, που συνήθως εκλέγονται ως περιορισμοί, παρουσιάζουν έντονη μη γραμμικότητα.
- Ακέραιος προγραμματισμός (Integer Programming). Μέθοδοι βελτιστοποίησης που ασχολούνται με προβλήματα στα οποία κάποιες (Mixed Integer Programming) ή όλες (Pure Integer Programming) οι μεταβλητές σχεδιασμού παίρνουν μόνο διακριτές ακέραιες τιμές. Η αντικειμενική συνάρτηση και οι συναρτήσεις περιορισμών μπορεί να είναι γραμμικές (Linear Integer Programming) ή μη γραμμικές (Non Linear Integer Programming).
- Δυναμικός Προγραμματισμός (Dynamic Programming). Ο δυναμικός προγραμματισμός περιλαμβάνει μεθοδολογίες οι οποίες διασπούν ένα περίπλοκο πρόβλημα σε πολλά απλούστερα τα οποία επαναλαμβάνονται. Αν και δεν αφορά μόνο τον κλάδο της βελτιστοποίησης, συχνά μπορεί να εφαρμοστεί αποτελεσματικά. Για παράδειγμα το πρόβλημα εύρεσης της ολικά βέλτιστης λύσης ανάγεται στην εύρεση τοπικά βέλτιστων λύσεων. Καθένα από τα τοπικά βέλτιστα βρίσκεται μια μόνο φορά και αποθηκεύεται. Στη συνέχεια τα τοπικά αυτά ελάχιστα συνδυάζονται για την εύρεση του ολικού βέλτιστου. Έτσι πετυχαίνεται σημαντική μείωση του υπολογιστικού κόστους.

Πολλές μέθοδοι μαθηματικού προγραμματισμού εγγυώνται την εύρεση βέλτιστης λύσης μετά από πεπερασμένο αριθμό επαναλήψεων. Από τις πλέον διαδεδομένες τέτοιες μεθόδους είναι ο αλγόριθμος Simplex (Dantzig 1947) για προβλήματα γραμμικού προγραμματισμού, οι επεκτάσεις του Simplex για προβλήματα τετραγωνικού προγραμματισμού και πολλοί αλγόριθμοι για συνδυαστική βελτιστοποίηση. Οι αλγόριθμοι αυτοί επιδιώκουν τον ακριβή προσδιορισμό της βέλτιστης λύσης, αλλά έχουν περιορισμένο πεδίο εφαρμογής.

Στον βέλτιστο σχεδιασμό κατασκευών οι προηγούμενες μέθοδοι δεν εφαρμόζονται γενικά. Αντίθετα προτιμούνται αλγόριθμοι βελτιστοποίησης που δεν υπολογίζουν την ακριβή λύση, αλλά την προσεγγίζουν συγκλίνοντας μετά από διαδοχικές επαναλήψεις (iterative methods). Κάποιες από τις πλέον διαδεδομένες είναι οι:

- Μέθοδοι Sequential Quadratic Programming (SQP) (Belegundu and Arora 1985, Thanedar et al. 1986, Papadrakakis et al. 1996) για μικρά, μεσαία και μεγάλα προβλήματα υπό περιορισμούς. Οι αλγόριθμοι αυτής της κατηγορίας είναι ιδιαίτερα αποτελεσματικοί για την επίλυση προβλημάτων βέλτιστου σχεδιασμού κατασκευών.
- Η μέθοδος Conditional Gradient Method (CRG) (Frank & Wolfe 1956)
- Οι μέθοδοι Feasible Directions Methods (MFD) (Zoutendijk 1960) για προβλήματα με ανισοτικούς περιορισμούς. Σε κάθε επανάληψη προσδιορίζεται μια διεύθυνση, τέτοια ώστε για μια μικρή κίνηση να μην καταλήγει εκτός εφικτής περιοχής και ένα αρκετά μικρό βήμα, τέτοιο ώστε η νέα λύση να είναι καλύτερη από την προηγούμενη.

- Η μέθοδος Generalized Reduced Gradient (GRG) (Lasdon et al. 1978)
- Η μέθοδος Method of Moving Asymptotes (MMA) (Svanberg 1987)
- Διάφορες μέθοδοι Trust-Region Method (TRM) (όπως Conn et al. 2000), που αναπτύσσονται τα τελευταία χρόνια

Οι επαναληπτικοί αυτοί αλγόριθμοι βελτιστοποίησης γενικά κάνουν χρήση της πρώτης (gradient) ή δευτέρας (hessian) παραγώγου των αντικειμενικών συναρτήσεων και περιορισμών, προκειμένου να καθοδηγήσουν την αναζήτηση. Έτσι η διαδικασία είναι ντετερμινιστική και συγκλίνει γρήγορα στην βέλτιστη λύση. Συνήθως σε κάθε επανάληψη πραγματοποιείται ανάλυση ευαισθησίας για τον υπολογισμό των παραγώγων. Ωστόσο το όφελος από τον μικρό αριθμό επαναλήψεων συχνά περιορίζεται από το μεγάλο υπολογιστικό κόστος της ανάλυσης ευαισθησίας. Επιπλέον οι μέθοδοι αυτές παγιδεύονται εύκολα σε τοπικά ακρότατα και δεν παρουσιάζουν τόσο καλή συμπεριφορά σε προβλήματα έντονα μη γραμμικά με πολλούς περιορισμούς, όπως είναι ο βέλτιστος σχεδιασμός κατασκευών. Τέλος ο υπολογισμός των παραγώγων προϋποθέτει την δυνατότητα αναλυτικής έκφρασης των συναρτήσεων του προβλήματος. Κάτι τέτοιο δεν είναι δυνατόν να εφαρμοστεί σε εμπορικά προγράμματα πεπερασμένων στοιχείων, όπου λεπτομέρειες υπολογισμού αποκρύπτονται από τον χρήστη.

1.6.2 Μεταευριστικές Μέθοδοι

Μια άλλη μεγάλη κατηγορία αλγορίθμων βελτιστοποίησης είναι οι μεταευριστικές μέθοδοι (Metaheuristics). Όπως και οι επαναληπτικές μέθοδοι (iterative methods) του μαθηματικού προγραμματισμο, συγκλίνουν διαδοχικά σε μία προσέγγιση της πραγματικά βέλτιστης λύσης. Ωστόσο οι μεταευρετικές μέθοδοι δεν απαιτούν πληροφορίες σχετικά με το πρόβλημα που επιλύουν, πέρα από τον υπολογισμό των τιμών συναρτήσεων περιορισμών και της αντικειμενικής συνάρτησης. Έτσι μπορούν να εφαρμοστούν σε πολύ περισσότερες κατηγορίες προβλημάτων, ακόμα και αν οι υπολογισμοί των συναρτήσεων δεν είναι γνωστοί σε αυτόν που εφαρμόζει τη μέθοδο βελτιστοποίησης ή αν οι συναρτήσεις του προβλήματος δεν είναι διαφορίσιμες ή αν οι ακριβές τιμές των παραμέτρων παρουσιάζουν αβεβαιότητες ή διακυμάνσεις. Επίσης μπορεί να είναι πιο αποδοτικές από τις μεθόδους του μαθηματικού προγραμματισμού, ανάλογα με το πρόβλημα βελτιστοποίησης και τα διαθέσιμα υπολογιστικά μέσα.

Οι μεταευριστικές μέθοδοι είναι αρκετά σύγχρονες. Οι πρώτοι μη μαθηματικοί αλγόριθμοι αναπτύχθηκαν από τους Robbins & Monro (1951) και από τους Kiefer & Wolfowitz (1952), στην προσπάθεια να δώσουν προσεγγιστικές λύσεις σε στοχαστικά προβλήματα βελτιστοποίησης. Το 1962 η ομάδα του βιολόγου Baricelli ανέπτυξε μια μέθοδο αναζήτησης, η οποία χρησιμοποιεί ένα μοντέλο προσομοίωσης της εξέλιξης στην φύση. Το 1963 ο Raistrigin πρότεινε έναν αλγόριθμο τυχαίας αναζήτησης (Random Search).

Στους μεταευριστικούς αλγορίθμους χρησιμοποιείται ένας πληθυσμός πιθανών λύσεων, καθεμία από τις οποίες αντιπροσωπεύει έναν διαφορετικό σχεδιασμό. Κάθε τέτοια λύση

βρίσκεται ανεξάρτητα από τις υπόλοιπες, αλλά σε κάποιο βήμα κάθε επανάληψης επικοινωνούν, ώστε να ανταλλάξουν πληροφορίες για την βέλτιστη λύση που συναντήθηκε μέχρι τότε. Έτσι εξερευνάται ο χώρος σχεδιασμού και η αναζήτηση καθοδηγείται προς την περιοχή της βέλτιστης λύσης από την εμπειρία που έχει αποκτηθεί με τους σχεδιασμούς που έχουν ήδη δοκιμαστεί αντί για την παράγωγο. Σε κάθε επανάληψη για την ανανέωση των διανυσμάτων μεταβλητών σχεδιασμού χρησιμοποιούνται συνήθως στοχαστικοί τελεστές και τυχαίοι αριθμοί αντί για αιτιοκρατικούς μαθηματικούς υπολογισμούς. Η αρχική επιλογή των πιθανών λύσεων μπορεί επίσης να είναι τυχαία.

Η λύση στην οποία καταλήγει η μέθοδος δεν είναι η βέλτιστη αλλά μια προσέγγιση αυτής, με ακρίβεια που εξαρτάται από τις συνθήκες του προβλήματος και τις παραμέτρους της μεθόδου. Μάλιστα οι μεταευριστικοί αλγόριθμοι δεν μπορούν να εγγυηθούν την σύγκλιση στην ολικά βέλτιστη λύση, ακόμα και αν αυτή υπάρχει. Λόγω της μεγάλης και τυχαίας διασποράς των πιθανών λύσεων στο χώρο σχεδιασμού, οι μέθοδοι αυτές δεν παγιδεύονται εύκολα σε τοπικά ελάχιστα, οπότε είναι κατάλληλες για καθολική βελτιστοποίηση (global optimization). Σίγουρα κάποια μέλη του πληθυσμού θα παγιδευτούν σε τοπικά ελάχιστα, ίσως για όλη την υπόλοιπη διάρκεια της διαδικασίας. Ωστόσο κάποια από τα υπόλοιπα θα κινηθούν προς το ολικό ελάχιστο με μεγάλη πιθανότητα. Συνήθως παρουσιάζεται αρχικά ταχεία σύγκλιση προς την περιοχή του ολικού ελαχίστου, αλλά στη συνέχεια η σύγκλιση αυτή επιβραδύνεται και η απόδοση μειώνεται.

Στην εργασία αυτή θα χρησιμοποιηθούν μεταευριστικοί αλγόριθμοι βελτιστοποίησης. Μερικοί από τους πλέον διαδεδομένους είναι:

1. Η μέθοδος των Γενετικών Αλγορίθμων (Genetic Algorithms - GA). Αναπτύχθηκε από τον J.Holland στις αρχές της δεκαετίας του 1970. Εμπνέεται από την επιστήμη της βιολογίας. Σε αυτήν τα μέλη του πληθυσμού κωδικοποιούνται σε δυαδική μορφή ώστε να προσομοιάζουν χρωμοσώματα. Χρησιμοποιεί τους γενετικούς τελεστές της μετάλλαξης (mutation) και του συνδυασμού (recombination), ώστε να δημιουργήσει τα χρωμοσώματα και να επιλέξει αυτό με την καλύτερη συνάρτηση ποιότητας (fitness function). Έχει εφαρμοστεί επιτυχημένα σε πολλά προβλήματα βέλτιστου σχεδιασμού κατασκευών.
2. Η μέθοδος των Στρατηγικών Εξέλιξης (Evolutionary Strategies - ES). Αναπτύχθηκε από τους I. Rechenberg (1973) και H.P. Schwefel (1981). Και αυτή προσομοιάζει την βιολογική εξέλιξη. Παρουσιάζεται αναλυτικά στην παράγραφο **2.5**
3. Η μέθοδος της Διαφορικής Εξέλιξης (Differential Evolution – DE). Προτάθηκε από τους Storn και Price το 1997. Πρόκειται για μια πολύ απλή μέθοδο που επίσης εμπνέεται από τη βιολογική εξέλιξη. Παρουσιάζεται αναλυτικά στην παράγραφο **2.5**
4. Η μέθοδος της Βελτιστοποίησης Σμήνους Σωματιδίων (Particle Swarm Optimization – PSO). Αναπτύχθηκε το 1995 από τους J.Kennedy και R.Eberhart. Προσομοιάζει την κοινωνική συμπεριφορά ενός πληθυσμού οργανισμών καθώς αναζητούν κάποιον πόρο. Παρουσιάζεται αναλυτικά στην παράγραφο **2.4**

5. Η μέθοδος της Βελτιστοποίησης με Αποικίες Μυρμηγκιών (Ant Colony Optimization - ACO). Εισήχθηκε από τον M.Dorigo το 1992. Η μέθοδος εμπνέεται από τη συμπεριφορά των μυρμηγκιών καθώς αναζητούν κάποια διαδρομή μεταξύ της αποικίας τους και μιας πηγής τροφής. Τα μυρμήγκια κινούνται τυχαία και μόλις βρουν τροφή επιστρέφουν στην αποικία αφήνοντας ένα ίχνος από φερορμόνες. Όταν ένα άλλο μυρμήγκι βρει το ίχνος, σταματάει να κινείται τυχαία και το ακολουθεί ενισχύοντάς το. Διαφορετικά οι φερορμόνες εξατμίζονται σταδιακά, ιδιαίτερα σε μεγάλα μονοπάτια. Τα μεγάλα μονοπάτια αντιπροσωπεύουν τοπικά ελάχιστα και με την εξάτμιση ο αλγόριθμος αποφεύγει να παγιδεύεται σε αυτά. Έχουν διατυπωθεί πολλές παραλλαγές του αλγορίθμου. Για κάποιες από αυτές έχει αποδειχθεί η σύγκλιση.
6. Η μέθοδος της Βελτιστοποίησης με Ανόπτηση (Simulated Annealing - SA). Περιγράφηκε από τους S.Kirkpatrick, C. Daniel Gelatt και Mario P. Vecchi (1983) και V.Cerny (1985). Εμπνέεται από τη μεταλλουργία και προσομοιάζει την τυχαία κίνηση των μορίων ενός μεταλλικού σώματος κατά την ψύξη του. Καθώς η θερμοκρασία μειώνεται σταδιακά, τα μόρια αναζητούν την θέση με την ελάχιστη θερμοδυναμική ενέργεια, μέχρι να επέλθει ισορροπία. Χρησιμοποιείται συχνά σε προβλήματα βελτιστοποίησης με διακριτές μεταβλητές.
7. Η μέθοδος της Αναζήτησης Αρμονίας (Harmony Search - HS). Δημιουργήθηκε από τον Z.W.Geem το 2001. Μιμείται τον αυτοσχεδιασμό μιας μπάντας μουσικών. Κάθε μουσικός αντιστοιχεί σε μία μεταβλητή σχεδιασμού, το εύρος τόνου του στο σύνολο τιμών της μεταβλητής, η μουσική αρμονία που πετυχαίνεται στη πιθανή λύση της επανάληψης και η απόκριση του κοινού στην αντικειμενική συνάρτηση. Η αρμονία βελτιώνεται σε κάθε επανάληψη με αυτοσχεδιασμό ή επιστροφή σε παλιότερη νότα για κάθε μουσικό.

Οι μέθοδοι GA, ES και DE ανήκουν στους εξελικτικούς αλγορίθμους (Evolutionary Algorithms). Κοινό χαρακτηριστικό τους είναι η χρήση στοχαστικών τελεστών αναπαραγωγής (reproduction), μετάλλαξης (mutation), συνδυασμού (recombination) και επιλογής (selection) ώστε να επιλεγούν τα καταλληλότερα προς επιβίωση (fitness) μέλη του πληθυσμού. Οι μέθοδοι PSO και ACO ανήκουν στους αλγορίθμους νοημοσύνης σμήνους (Swarm Intelligence) που μιμούνται τη συμπεριφορά ενός αποκεντρωμένου πληθυσμού ανεξάρτητων μελών. Κάποιες μέθοδοι (HS, SA) δεν χρησιμοποιούν πληθυσμούς, αλλά μόνο μια πιθανή λύση κάθε φορά. Είναι επίσης δυνατό κάποια μέθοδος (π.χ. SA) να εφαρμόζει διαδοχικά τοπικές μαθηματικές μεθόδους αναζήτησης αντί να χρησιμοποιεί την εμπειρία ενός πληθυσμού.

Οι μεταευρετικοί αλγόριθμοι παρουσιάζουν πολλά πλεονεκτήματα που ευνοούν την εφαρμογή τους σε προβλήματα βέλτιστου σχεδιασμού κατασκευών:

- Η τυχηματικότητα και η χρήση πληθυσμών εμποδίζει την παγίδευσή τους σε τοπικά ελάχιστα. Αυτό είναι βασικό πρόβλημα των μαθηματικών μεθόδων σε έντονα μη γραμμικά προβλήματα.
- Επίσης χρησιμοποιούν ελάχιστες πληροφορίες για το εκάστοτε πρόβλημα και δεν χρειάζονται παραγωγίσιμες συναρτήσεις και αναλύσεις ευαισθησίας. Αυτό επιτρέπει τον συνδυασμό τους με εμπορικά λογισμικά πεπερασμένων στοιχείων, που δεν εκθέτουν στο χρήστη τα μητρώα (π.χ. δυσκαμψίας, απόσβεσης, κτλ.) και τους υπολογισμούς.
- Η τυχηματική φύση των τελεστών και το πλήθος των εξεταζόμενων λύσεων σε κάθε επανάληψη διευκολύνουν το συνδυασμό τους με τεχνικές δειγματοληψίας (sampling) για την επίλυση στοχαστικών προβλημάτων.

- Οι περισσότερες μεταευριστικές μέθοδοι μπορούν να αντιμετωπίσουν προβλήματα με διακριτές μεταβλητές με μικρές ή χωρίς καθόλου τροποποιήσεις.
- Τέλος η ανεξαρτησία των εναλλακτικών λύσεων που εξετάζονται σε κάθε επανάληψη επιτρέπει την παράλληλη επεξεργασία τους. Αυτό είναι ιδιαίτερα σημαντικό σήμερα, αφού τα σύγχρονα υπολογιστικά συστήματα χρησιμοποιούν επεξεργαστές με πολλούς πυρήνες, κάρτες γραφικών για τις οποίες οι μηχανικοί μπορούν να γράψουν παράλληλο κώδικα και διανεμημένα συστήματα, δηλαδή πολλοί υπολογιστές που συνδέονται μέσω διαδικτύου να συνεργάζονται.

Βασικό μειονέκτημα των μεταευριστικών αλγορίθμων είναι η απαίτηση μεγάλου αριθμού επαναλήψεων και πολλών δοκιμών. Οι αλγόριθμοι μαθηματικού προγραμματισμού συγκλίνουν ταχύτερα. Ωστόσο ο υπολογισμός παραγώγων κατά την ανάλυση ευαισθησίας είναι χρονοβόρα διαδικασία, ιδιαίτερα σε προβλήματα μηχανικού, όπου οι συναρτήσεις περιορισμών ή και η αντικειμενική είναι έντονα μη γραμμικές. Το υπολογιστικό κόστος της ανάλυσης ευαισθησίας αυξάνεται στη Στοχαστική Βελτιστοποίηση και τη Βελτιστοποίηση Εύρωστου Σχεδιασμού όπου οι περιορισμοί είναι πιθανοτικοί.

Σε μια σειρά μελετών (Papadrakakis et al. 1998, Papadrakakis et al. 1999, Lagaros et al. 2002) φάνηκε ότι εξελικτικοί αλγόριθμοι βελτιστοποίησης είναι περισσότερο αποδοτικοί από μεθόδους μαθηματικού προγραμματισμού. Παρότι απαιτούνται σημαντικά περισσότερα βήματα για τη σύγκλιση, το υπολογιστικό κόστος κάθε βήματος είναι πολύ λιγότερο αφού αποφεύγεται η ανάλυση ευαισθησίας.

1.6.3 Υβριδικές Μέθοδοι

Τις τελευταίες δεκαετίες έχουν αρχίσει να αναπτύσσονται μεθοδολογίες που συνδυάζουν μεταευριστικούς αλγορίθμους μεταξύ τους ή με μεθόδους μαθηματικού προγραμματισμού. Οι Waggen et al. (1992) εισήγαγε έναν συνδυασμό εξελικτικού προγραμματισμού με τη μέθοδο Directions Set Method για προβλήματα χωρίς περιορισμούς. Οι Papadrakakis et al. (1999) πρότεινε μια υβριδική ES-SQP μέθοδο για βελτιστοποίηση σχήματος. Οι Papadrakakis και Lagaros (2000) εφάρμοσαν έναν υβριδικό αλγόριθμο GA-MP για προβλήματα βελτιστοποίησης μεγέθους διατομών. Ο συνδυασμοί GA-SQP και ES-SQP φάνηκε ότι υπερτερούν της ξεχωριστής εφαρμογής των μεθόδων ως προς την ταχύτητα σύγκλισης και την ποιότητα της λύσης, ιδιαίτερα όταν οι αρχικές λύσεις απέχουν πολύ από τις βέλτιστες (Papadrakakis, Lagaros, Tsompanakis, Plevris 2001). Οι Kaveh και Talahatari (2008) χρησιμοποίησαν συνδυασμό των μεθόδων PSO και ACO για το βέλτιστο σχεδιασμό δικτυωτών κατασκευών.

Οι μεταευρετικοί αλγόριθμοι και οι μέθοδοι μαθηματικού προγραμματισμού παρουσιάζουν ιδιότητες που επιτρέπουν την αλληλοσυμπλήρωσή τους αν συνδυαστούν αποτελεσματικά. Οι μετευρετικές μέθοδοι δεν παγιδεύονται εύκολα σε τοπικά ακρότατα και συνήθως συγκλίνουν γρήγορα στην περιοχή του ολικά ελαχίστου, αλλά έπειτα επιβραδύνονται. Αντίθετα οι αλγόριθμοι μαθηματικού προγραμματισμού είναι πολύ αποδοτικοί στην εύρεση του βέλτιστου σχεδιασμού μόλις βρεθούν στην περιοχή του, αλλά κινδυνεύουν να παγιδευτούν σε τοπικά ελάχιστα αν ξεκινήσουν να ψάχνουν τυφλά. Με το συνδυασμό της τοπικής αναζήτησης των μαθηματικών μεθόδων με την καθολική των μεταευρετικών, μπορούν να προκύψουν αλγόριθμοι που ξεπερνούν σε απόδοση και τις δύο προσεγγίσεις.

Κεφάλαιο 2

2 Αλγόριθμοι Βελτιστοποίησης

2.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφονται αναλυτικά οι τρεις αλγόριθμοι βελτιστοποίησης που χρησιμοποιούνται στις παρακάτω αριθμητικές εφαρμογές: Βελτιστοποίηση Σμήνους Σωματιδίων, Διαφορική Εξέλιξη και Εξελικτικές Στρατηγικές. Παρουσιάζονται επίσης τεχνικές προσαρμογής των αλγορίθμων αυτών σε διαφορετικές κατηγορίες προβλημάτων, από αυτές για τις οποίες είχαν αρχικά αναπτυχθεί.

2.2 Τεχνικές Αντιμετώπισης Περιορισμών

Από τους διάφορους αλγόριθμους βελτιστοποίησης που έχουν προταθεί, ειδικά στην κατηγορία των μεταευριστικών, οι περισσότεροι αναπτύσσονται και δοκιμάζονται αρχικά για προβλήματα χωρίς περιορισμούς. Τα «μαθηματικά» αυτά προβλήματα περιλαμβάνουν μία ή περισσότερες δύσκολες (μη κυρτές δηλαδή) αντικειμενικές συναρτήσεις, που έχουν ερευνηθεί και χρησιμοποιούνται ευρέως για την αξιολόγηση των μεθόδων βελτιστοποίησης. Δυστυχώς δεν υπάρχει κάποια ενιαία μεθοδολογία αντιμετώπισης των συναρτήσεων περιορισμών. Έχουν επινοηθεί ποικίλες τεχνικές χειρισμού των περιορισμών.

2.2.1 Κατηγορίες Τεχνικών

Σύμφωνα με τους J. Michalewicz, S. Koziel και C.A. Coello κάποιες γενικές κατηγορίες είναι:

- Μέθοδοι που βασίζονται σε συναρτήσεις ποινής. Είναι η πλέον διαδεδομένη κατηγορία. Αναλύεται στη συνέχεια.
- Μέθοδοι που διατηρούν την εφικτότητα των λύσεων. Συνήθως ξεκινούν από εφικτές λύσεις και χρησιμοποιούν τελεστές που όταν εφαρμόζονται σε εφικτή λύση παράγουν πάλι εφικτό αποτέλεσμα, είναι δηλαδή κλειστές πράξεις. Ως παράδειγμα δίνεται ο αλγόριθμος GENOCOP (Michalewicz & Janikow, 1991)
- Μέθοδοι που απορρίπτουν μη εφικτές λύσεις. Είναι η πιο απλή αντιμετώπιση ενός μη εφικτού σχεδιασμού. Σε αντίθεση με τις συναρτήσεις ποινής δεν ασχολούνται με τον υπολογισμό της συνάρτησης ποιότητας (fitness) και έτσι δεν εκμεταλλεύονται πληροφορίες που παρέχονται για τις μη εφικτές περιοχές. Ως αποτέλεσμα έχουν μειωμένη απόδοση ιδιαίτερα αν η εφικτή περιοχή του χώρου σχεδιασμού δεν είναι κυρτή. Π.χ. μέθοδος θανατικής καταδίκης (death penalty).

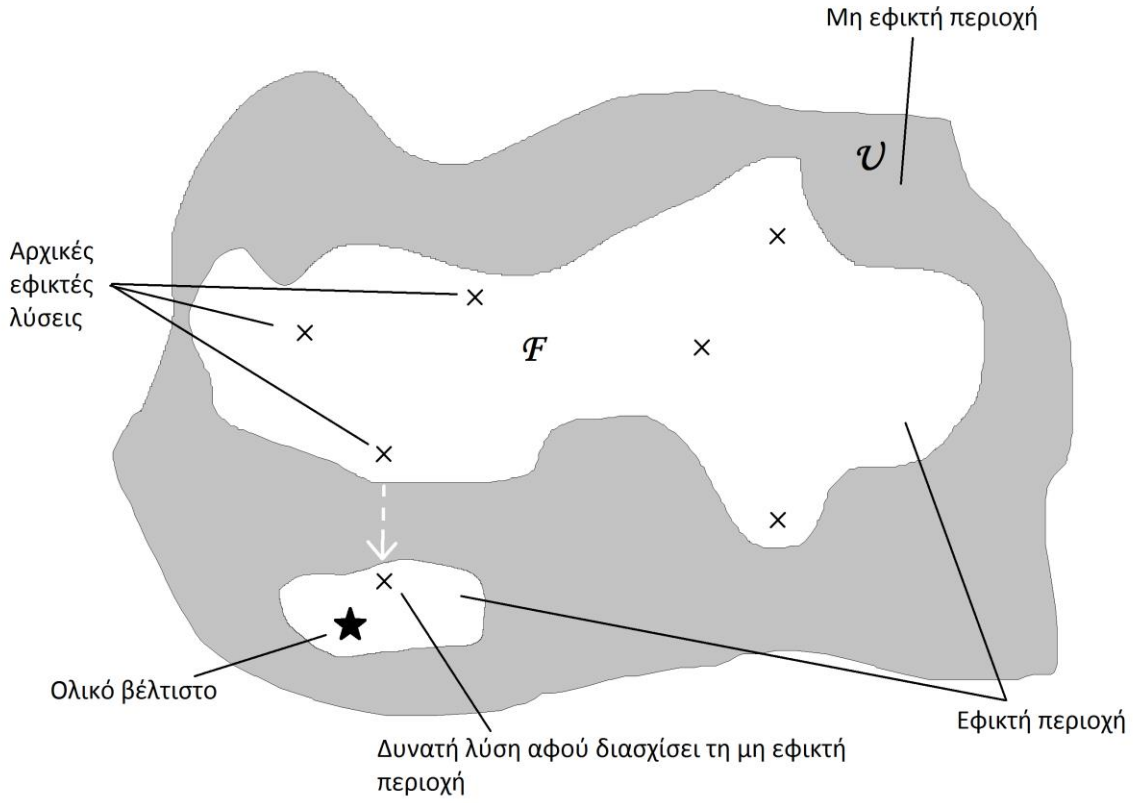
- Μέθοδοι που επιδιορθώνουν μη εφικτές λύσεις. Στις μεθόδους αυτές ένας μη εφικτός σχεδιασμός δεν απορρίπτεται, αλλά είναι πιθανόν να αντικατασταθεί από μια επιδιορθωμένη και εφικτή παραλλαγή του. Η πιθανότητα αυτή συνήθως τίθεται περίπου 5%. Τέτοιες τεχνικές είναι δημοφιλείς στην συνδυαστική βελτιστοποίηση (combinatorial optimization), αλλά δεν έχουν αναπτυχθεί ικανοποιητικά για συνεχή προβλήματα λόγω της δυσκολίας επιδιόρθωσης.
- Μέθοδοι που χρησιμοποιούν απεικονίσεις και αποκωδικοποίηση. Κάποιες τεχνικές εστιάζουν στην απεικόνιση ενός σχεδιασμού από την μη εφικτή περιοχή του προβλήματος στην εφικτή. Μια διαφορετική προσέγγιση είναι ο μετασχηματισμός ολόκληρης της εφικτής περιοχής σε σχήμα που να μπορεί να εξερευνηθεί πιο εύκολα. Η πιο γνωστή τέτοια μέθοδος είναι οι Ομοιόμορφοι Χάρτες (Homomorphous Maps, Koziel & Michalewicz, 1999), που ήταν για καιρό ιδιαίτερα αποδοτική για κυρτές ή μη περιοχές και παραμένει ανταγωνιστική για την πρώτη περίπτωση.
- Μέθοδοι που διαχωρίζουν τους περιορισμούς από τις αντικειμενικές συναρτήσεις. Σε αντίθεση με τις συναρτήσεις ποινής οι τεχνικές αυτές δεν συνδυάζουν την αντικειμενική συνάρτηση με τους περιορισμούς σε μια τιμή ποιότητας (fitness). Για παράδειγμα στη μέθοδο Coevolution (Paredis 1994) αλληλεπιδρούν δύο πληθυσμοί, ένας για την αντικειμενική συνάρτηση και ένας για τους περιορισμούς αντίστοιχα. Οι Schoenauer & Xanthakis (1993) παρουσίασαν τη μέθοδο Behavioural Memory, στην οποία οι λύσεις ικανοποιούν διαδοχικά τους περιορισμούς. Ιδιαίτερο ενδιαφέρον έχει η προσέγγιση της μετατροπής των περιορισμών σε επιπλέον αντικειμενικές συναρτήσεις. Έτσι μπορούν να επιστρατευτούν μεθοδολογίες επίλυσης πολυκριτηριακών προβλημάτων.
- Υβριδικές μέθοδοι. Έχουν αναπτυχθεί αρκετές τεχνικές που συνδυάζουν εξελικτικούς αλγόριθμους και αιτιοκρατικούς (μαθηματικούς). Η αντιμετώπιση των περιορισμών γίνεται με μεθοδολογίες και από τις δύο προσεγγίσεις.

2.2.2 Συναρτήσεις ποινής

Η πιο δημοφιλής τεχνική αντιμετώπισης περιορισμών, ειδικά ανισοτήτων, είναι η επιβολή ποινής. Οι συναρτήσεις ποινής προτάθηκαν αρχικά από τον R. Courant το 1943 και αργότερα αναπτύχθηκαν από τους Carroll και Fiacco & McCormick. Η βασική φιλοσοφία των τεχνικών αυτή της κατηγορίας είναι να αφαιρέσουν τους περιορισμούς ενός προβλήματος βελτιστοποίησης εκφράζοντάς τους ως επιδείνωση της αντικειμενικής συνάρτησης. Η επιδείνωση αυτή εξαρτάται από το βαθμό παραβίασης των περιορισμών και επιτρέπει την αξιολόγηση μη εφικτών λύσεων.

Στο επόμενο σχήμα φαίνεται καθαρά η αναγκαιότητα εξερεύνησης της μη εφικτής περιοχής του χώρου σχεδιασμού. Αν δεν επιτρέπαμε μη εφικτές λύσεις στον πληθυσμό, τότε όλες οι δοκιμές θα περιορίζονταν στο μέρος της εφικτής περιοχής από το οποίο ο πληθυσμός ξεκινάει. Το ολικό βέλτιστο όμως βρίσκεται σε άλλο μέρος της εφικτής περιοχής που περικλείεται από τη

μη εφικτή περιοχή. Τα δύο μέρη αυτά δεν επικοινωνούν. Ο μόνος τρόπος να βρει η μέθοδος το ολικό ελάχιστο είναι να διασχίσει την μη εφικτή περιοχή, να επιτρέψει δηλαδή προσωρινά κάποιες αδύνατες λύσεις.



Σχήμα 2.1 Εφικτή και μη εφικτή περιοχή ενός δυσδιάστατου χώρου σχεδιασμού. Αν επιτραπεί η εξερεύνηση της μη εφικτής περιοχής, η μέθοδος θα καταλήξει στην ολικά βέλτιστη λύση.

Η επιλογή της συνάρτησης ποινής είναι πολύ σημαντική. Υπερβολικά μεγάλες ποινές εμποδίζουν την εξερεύνηση στην μη εφικτή περιοχή του χώρου σχεδιασμού. Αντίθετα υπερβολικά μικρές ποινές σπαταλούν πολλές δοκιμές σε αδύνατες λύσεις προς κάποια κατεύθυνση που η αναζήτηση είναι μάταια.

Ένας περιορισμός ανισότητας παραβιάζεται όταν

$$G_j(x) = g_j(x)^{\nu} > 0, \ \nu = 1 \text{ ή } 2. \qquad \textbf{(2.1)}$$

Ένας περιορισμός ισότητας μπορεί να εκφραστεί με παρόμοιους όρους, αν επιτρέψουμε μια πολύ μικρή ανοχή ε, οπότε

$$H_j(x) = |h_j(x)|^{\mu} - \varepsilon > 0, \qquad \mu = 1 \text{ ή } 2 \qquad \textbf{(2.2)}$$

Έτσι η τιμή ποιότητας για την λύση αυτή υπολογίζεται συνήθως ως

$$fitness(x) = F(x) \pm [\sum(a_j * G_j) + \sum(b_j * H_j)] \quad \textbf{(2.3)}$$

όπου οι παράμετροι α_j και b_j καλούνται συντελεστές ποινής.

Συχνές συναρτήσεις ποινής είναι:

- Η θανατική ποινή (Death Penalty), που αναλύθηκε προηγουμένως. Μια παραλλαγή της είναι να απειριστεί η συνάρτηση ποιότητας που θέλουμε να ελαχιστοποιήσουμε, ώστε να εξασφαλίσουμε ότι μη εφικτοί σχεδιασμοί δεν θα προτιμηθούν.
- Στατικές ποινές, π.χ. τα επίπεδα ποινής των Homaifar, Lai and Qi (1994). Τεχνικές που σε αυτήν την κατηγορία χρησιμοποιούν σταθερούς συντελεστές καθ' όλη τη διάρκεια της διαδικασίας βελτιστοποίησης. Κάτι τέτοιο όμως απαιτεί τον προσδιορισμό συνήθως μεγάλου αριθμού συντελεστών ποινής, που εξαρτώνται από το πρόβλημα και επηρεάζουν σημαντικά την σύγκλιση.
- Δυναμικές Ποινές, π.χ. Joines and Houck (1994). Μεθοδολογίες που ακολουθούν αυτήν την προσέγγιση χρησιμοποιούν συντελεστές ποινής που μεταβάλλονται (συνήθως αυξάνονται) όσο αυξάνονται οι επαναλήψεις του αλγορίθμου βελτιστοποίησης. Γενικά παρουσιάζουν καλύτερη συμπεριφορά από τις στατικές ποινές.
- Προσαρμοστικές ποινές. Αυτές οι τεχνικές προσαρμόζουν την συνάρτηση ποινής ανάλογα με την πιο πρόσφατη συμπεριφορά των μελών του πληθυσμού. Για παράδειγμα η μέθοδος των Bean και Hadj-Alouane (1992,1997) αυξάνει τους συντελεστές ποινής αν η καλύτερη λύση των τελευταίων επαναλήψεων είναι μη εφικτή (οπότε εξερευνάται υπερβολικά η μη εφικτή περιοχή) και τους μειώνει στην αντίθετη περίπτωση.

2.2.3 Αντιμετώπιση Περιορισμών Στην Παρούσα Εργασία

Στην εργασία αυτή θα χρησιμοποιηθεί η ακόλουθη γραμμική σε διαστήματα συνάρτηση ποινής, που προτάθηκε από τον Β. Πλεύρη στο πλαίσιο εκπόνησης της Διδακτορικής του διατριβής:

Έστω περιορισμός της μορφής

$$g_j(\boldsymbol{x}) = |q_j(\boldsymbol{x})| - q_{j,allow} \leq 0 \quad \textbf{(2.4)}$$

Όπου $q_j(\boldsymbol{x})$ **ε**ίναι μια τιμή απόκρισης, που στον βέλτιστο σχεδιασμό κατασκευών εκφράζει κάποια τάση ή παραμόρφωση, για τη λύση $\boldsymbol{x}$, και $q_{j,allow}$ η μέγιστη επιτρεπόμενη απόλυτη τιμή της. Η συνάρτηση ποινής υπολογίζεται ως εξής:

$$pen_j(\boldsymbol{x}) = \begin{cases} 1 & ,\alpha\nu \quad |q_j(\boldsymbol{x})| \le q_{j,allow} \\ \dfrac{|q_j(\boldsymbol{x})|}{q_{j,allow}} & ,\alpha\nu \quad |q_j(\boldsymbol{x})| > q_{j,allow} \end{cases} \qquad \textbf{(2.5)}$$

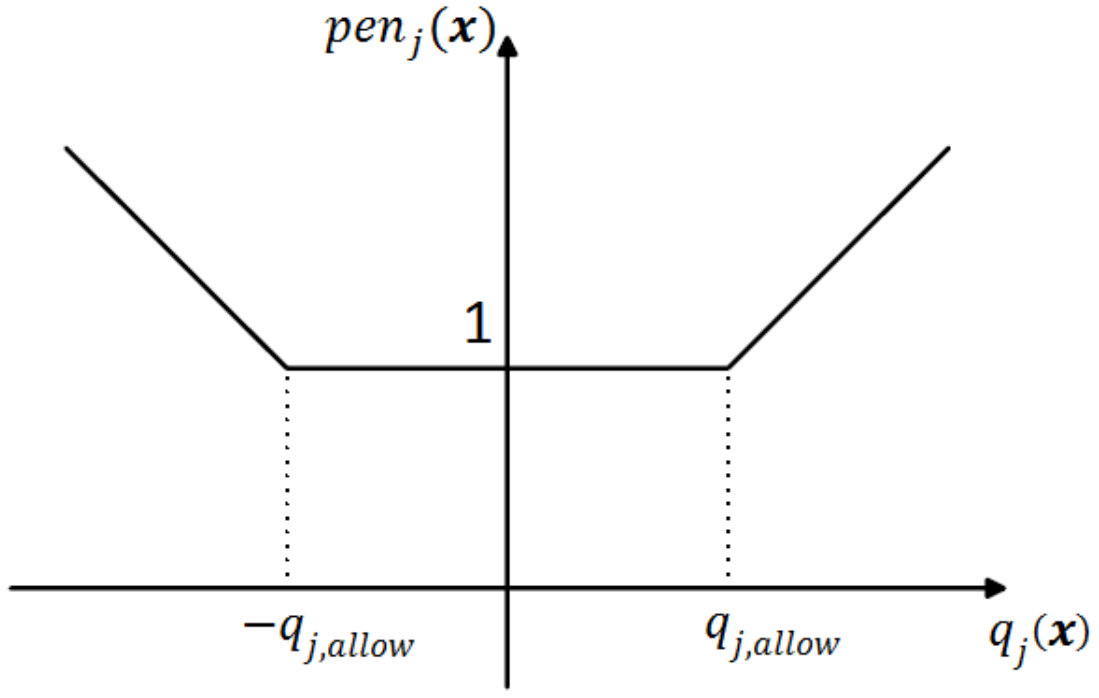
Για οικονομία πράξεων και μνήμης όλοι οι περιορισμοί που έχουν την ίδια $q_{j,allow}$ ομαδοποιούνται και οι συναρτήσεις (2.4) και (2.5) υπολογίζονται μόνο για την χειρότερη περίπτωση, δηλαδή μόνο για την μέγιστη απόκριση $q_j(\boldsymbol{x})$. Για παράδειγμα αν έχουμε N_e πεπερασμένα στοιχεία από υλικό για το οποίο πρέπει $|\sigma| \le \sigma_{allow}$, τότε σαν μέτρο απόκρισης παίρνουμε το:

$$q_j(\boldsymbol{x}) = \max_{i=1\ldots N_e} \{|\sigma_i|\} \qquad \textbf{(2.6)}$$

Η τιμή ποιότητας (fitness) $f_p(\boldsymbol{x})$ της λύσης $\boldsymbol{x}$ υπολογίζεται με πολλαπλασιασμό της αντικειμενικής συνάρτησης $f(\boldsymbol{x})$ με τη μέγιστη συνάρτηση ποινής, που προκύπτει από το σύνολο των m ομάδων περιορισμών σύμφωνα με την εξίσωση (2.7). Με την τιμή αυτή αξιολογείται η καταλληλότητα κάθε σχεδιασμού.

$$f_p(\boldsymbol{x}) = f(\boldsymbol{x}) * \max\left\{pen_j(x)\right\}, \qquad j = 1, \ldots, m \qquad \textbf{(2.7)}$$

Στο **Σχήμα 2.2** φαίνεται η γραφική παράσταση της εν λόγω συνάρτησης ποινής.



Σχήμα 2.2 Συνάρτηση ποινής με γραμμική συμπεριφορά στους κλάδους που ο περιορισμός παραβιάζεται.

Είναι δυνατόν ένας μη εφικτός σχεδιασμός να έχει τόσο μικρή αντικειμενική τιμή, που παρά την επιβολή ποινής να είναι εικονικά καλύτερος από τη βέλτιστη λύση που έχουν συναντήσει τα μέλη του πληθυσμού μέχρι την επανάληψη εκείνη. Οι αλγόριθμοι βελτιστοποίησης αποθηκεύουν την καλύτερη λύση που έχει βρεθεί σε κάθε βήμα. Προφανώς ο μη εφικτός σχεδιασμός αυτός δε μπορεί να αποθηκευτεί σαν βέλτιστη λύση κάποιου βήματος ή συνολικά.

2.3 Τεχνικές Αντιμετώπισης Διακριτών Μεταβλητών

Στον κλάδο του βέλτιστου σχεδιασμού κατασκευών προκύπτουν συχνά προβλήματα που κάποιες τουλάχιστον από τις μεταβλητές σχεδιασμού αντιπροσωπεύουν χαρακτηριστικά τυποποιημένων διατομών, οπότε μπορούν να πάρουν μόνο διακριτές τιμές. Όμως πολλοί δημοφιλείς αλγόριθμοι βελτιστοποίησης αναπτύσσονται για το χειρισμό είτε συνεχών μεταβλητών είτε διακριτών. Φυσικά κάποιοι αλγόριθμοι (π.χ. Στρατηγικές Εξέλιξης) έχουν σχεδιαστεί εξαρχής για την επίλυση μεικτών προβλημάτων, αλλά συχνά θέλουμε να χρησιμοποιήσουμε ένα αλγόριθμο συνεχούς βελτιστοποίησης (όπως η Βελτιστοποίηση Σμήνους Σωματιδίων και η Διαφορική Εξέλιξη) σε ένα μεικτό ή αμιγώς διακριτό πρόβλημα.

Υπάρχουν αρκετές τεχνικές για την προσαρμογή συνεχών αλγορίθμων βελτιστοποίησης σε διακριτά προβλήματα, που διαφέρουν από άποψη ευκολίας εφαρμογής και αποδοτικότητας. Παρακάτω αναφέρονται τρεις συνήθεις τρόποι προσαρμογής.

2.3.1 Παραλλαγή του αλγορίθμου και των τελεστών του

Αποδίδεται διαφορετική λειτουργία για τους τελεστές που χρησιμοποιεί ο αλγόριθμος (π.χ. πρόσθεση, μετάλλαξη, κτλ.) ώστε να έχουν νόημα για διακριτές, ακέραιες ή δυαδικές τιμές. Συνήθως καταβάλλεται προσπάθεια ώστε να μην αλλοιώνεται η αρχική λογική που εμπνεύστηκε ο ιδρυτής της μεθόδου. Ωστόσο το αποτέλεσμα είναι ένας διαφορετικός αλγόριθμος συνήθως που απλά μοιάζει στον πρωτότυπο.

2.3.2 Η μέθοδος Κλάδου - Ορίου (Branch & Bound).

Εισήχθηκε από τους A. H. Land και A. G. Doig το 1960 για την επίλυση προβλημάτων ακέραιας και συνδυαστικής βελτιστοποίησης, αλλά μπορεί να τροποποιηθεί άμεσα για να εφαρμοστεί και σε μεικτά προβλήματα. Η τεχνική αυτή χρησιμοποιεί λογικά δέντρα, γι’ αυτό και ο όρος Branch.

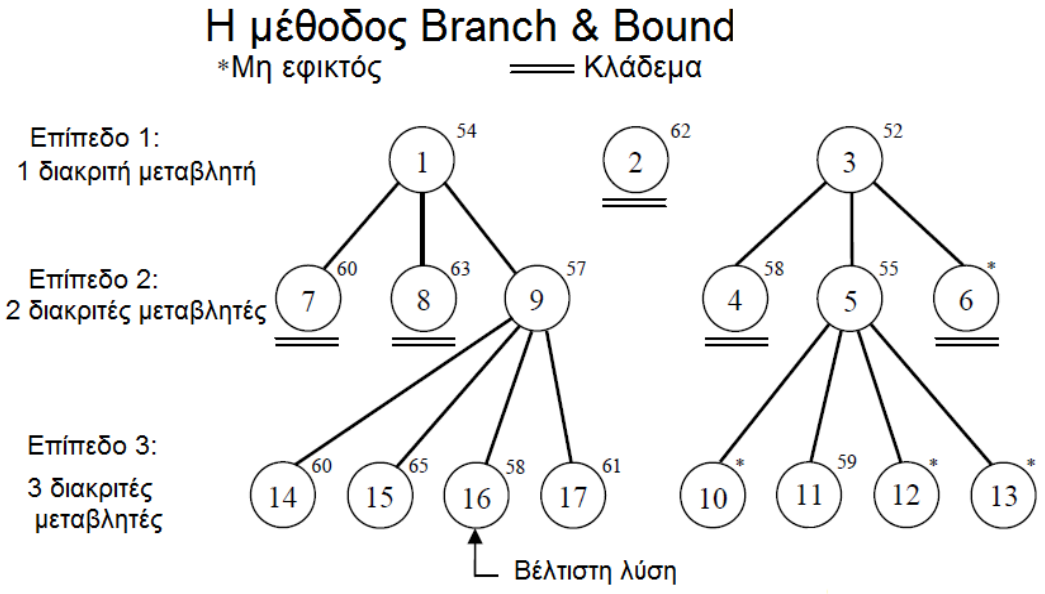
Αρχικά μία μόνο μεταβλητή σχεδιασμού αντιμετωπίζεται ως διακριτή ενώ οι υπόλοιπες θεωρούνται συνεχείς. Δημιουργούνται τόσες ρίζες δέντρων όσες και οι τιμές της μεταβλητής αυτής. Στη συνέχεια προστίθενται κόμβοι σε κάθε ρίζα. Στο δεύτερο αυτό επίπεδο άλλη μια μεταβλητή θεωρείται διακριτή. Συνεχίζοντας έτσι στο τελευταίο επίπεδο όλες οι μεταβλητές είναι πλέον διακριτές. Ωστόσο δεν εξετάζονται όλοι οι κόμβοι. Αυτό άλλωστε θα οδηγούσε σε

εξαντλητική αναζήτηση που δεν είναι δυνατόν να εφαρμοστεί όταν το πλήθος των διακριτών μεταβλητών ή των εναλλακτικών τιμών τους είναι μεγάλο.

Αντίθετα οι περισσότεροι κόμβοι και τα «υπό-δέντρα» που ξεκινούν από αυτούς «κλαδεύονται» ώστε να μην εξετασθούν περαιτέρω. Σε κάθε κόμβο πριν το τελευταίο επίπεδο γίνεται μια βελτιστοποίηση στην οποία όλες οι άλλες μεταβλητές είναι συνεχείς. Έτσι η τιμή της αντικειμενικής συνάρτησης, που θέλουμε να ελαχιστοποιήσουμε, είναι χαμηλότερη από την πραγματική, αποτελεί δηλαδή ένα κάτω όριο (lower bound). Εξετάζοντας τους κόμβους του νεότερου επιπέδου και του προηγούμενου κάθε φορά, επιλέγεται να διακλαδιστεί πάντα ο κόμβος με την χαμηλότερη αντικειμενική συνάρτηση. Μη εφικτοί σχεδιασμοί φυσικά απορρίπτονται.

Το υπολογιστικό κόστος της παραπάνω μεθοδολογίας είναι μεγάλο αφού για κάθε κόμβο απαιτείται μια βελτιστοποίηση, που με τη σειρά της εκτελεί πολλαπλές αναλύσεις πεπερασμένων στοιχείων. Ωστόσο οι περισσότεροι κόμβοι κλαδεύονται πολύ γρήγορα. Υπάρχουν επίσης τεχνικές γραμμικής προσέγγισης των ενδιάμεσων προβλημάτων βελτιστοποίησης που μειώνουν το υπολογιστικό κόστος αλλά δυστυχώς και την ακρίβεια.

Στο παρακάτω σχήμα φαίνεται μια σχηματική απεικόνιση της μεθόδου. Στο πάνω δεξιά μέρος κάθε κόμβου-κύκλου αναγράφεται η τιμή της αντικειμενικής συνάρτησης για τον σχεδιασμό εκείνο. Εντός των κόμβων αναγράφεται η σειρά με την οποία ο κόμβος αυτός εξετάστηκε, πραγματοποιήθηκε δηλαδή βελτιστοποίηση. Στο παράδειγμα αυτό εξετάζονται πάρα πολύ συνδυασμοί, πράγμα που δεν συμβαίνει σε πρακτικές εφαρμογές.

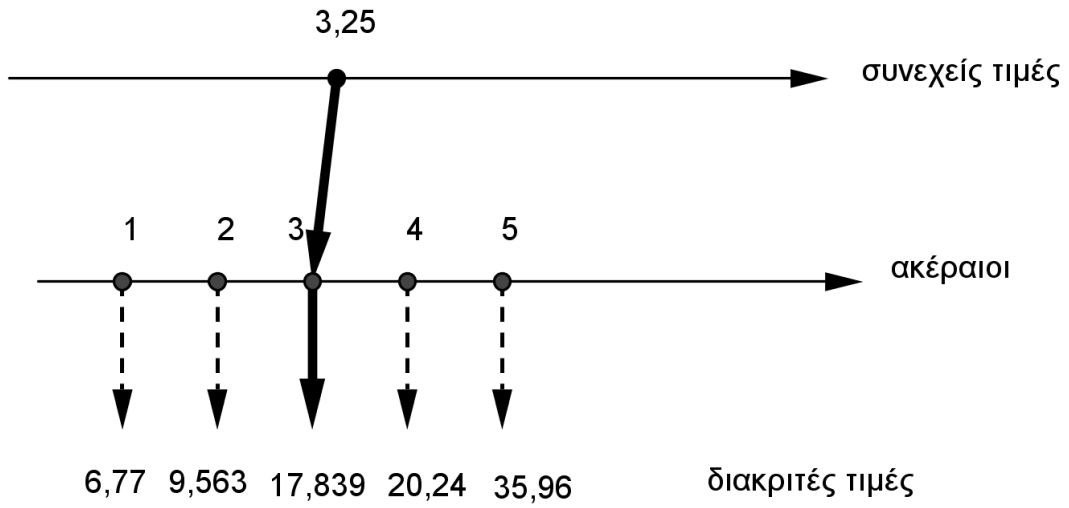


Σχήμα 2.3 Η μέθοδος Branch & Bound για τρεις διακριτές μεταβλητές 3-3-4 δυνατών τιμών, αντίστοιχα.

2.3.3 Απεικονίσεις του συνεχή χώρου σχεδιασμού στον διακριτό.

Τέτοιες μεθοδολογίες είναι πολύ απλές και γι' αυτό είναι αρκετά δημοφιλείς. Ο αλγόριθμος βελτιστοποίησης εφαρμόζεται σε συνεχείς μεταβλητές κατά τα γνωστά, χωρίς κάποια τροποποίηση των βημάτων του ή των τελεστών του. Πριν τον υπολογισμό της αντικειμενικής συνάρτησης και των περιορισμών οι τιμές των συνεχών μεταβλητών αντιστοιχίζονται στο διακριτό χώρο σχεδιασμού. Έτσι οι λύσεις που εξετάζονται έχουν σίγουρα νόημα και το υπολογιστικό κόστος του αρχικού αλγορίθμου δεν αυξάνεται. Όμως δεν εξασφαλίζονται η ταχύτητα της σύγκλισης και η ποιότητα της τελικής λύσης, καθώς εξαρτώνται από τον τρόπο που γίνεται η απεικόνιση.

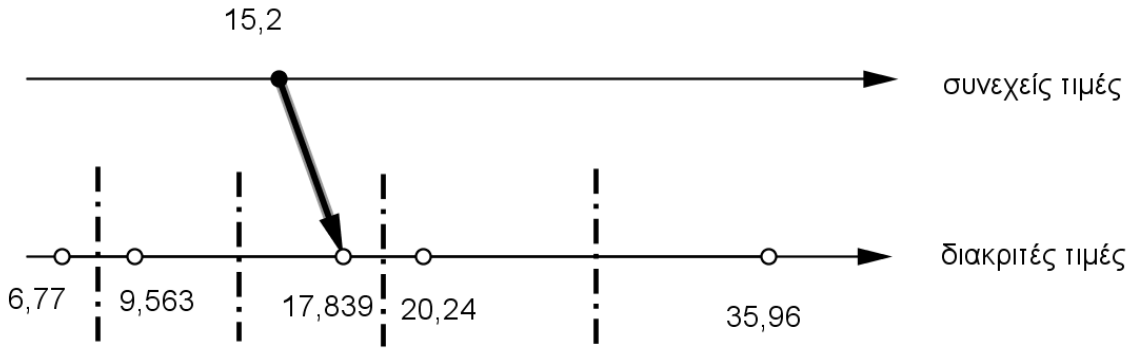
Η αντιστοίχιση αυτή είναι συχνά πολύ απλή. Για πολλές εφαρμογές αρκεί η στρογγυλοποίηση της πραγματικής τιμής μιας μεταβλητής στον πλησιέστερο ακέραιο. Ο ακέραιος αυτός έπειτα αντιστοιχίζεται σε μία από τις διακριτές τιμές που μπορεί να πάρει η μεταβλητή. Η απεικόνιση αυτή φαίνεται στο **Σχήμα 2.4**. Ένα βασικό πρόβλημά της είναι ότι απαλείφει τις αποστάσεις μεταξύ των διακριτών τιμών. Αν για παράδειγμα οι επιτρεπόμενες τιμές μιας μεταβλητής είναι 6.77, 9.563, 17.839, 20.24 και 35.96, η απόσταση μεταξύ της πρώτης και δεύτερης τιμής είναι πολύ μικρότερη από την απόσταση μεταξύ δεύτερης και τρίτης. Και οι πέντε όμως θα αντιστοιχιστούν σε συνεχόμενους ακέραιους.



Σχήμα 2.4 Στρογγυλοποίηση της συνεχούς τιμής στην πλησιέστερη ακέραια. Καθεμία ακέραια τιμή αντιστοιχίζεται έπειτα σε μία από τις διακριτές τιμές που μπορεί να λάβει η μεταβλητή.

Μια εξίσου εύκολη στην εφαρμογή της παραλλαγή της παραπάνω απεικόνισης είναι η εξής: Για κάθε συνεχή πραγματική τιμή που παίρνει μια μεταβλητή θα βρίσκουμε και θα χρησιμοποιούμε την πλησιέστερη επιτρεπόμενη διακριτή. Δηλαδή οι διακριτές τιμές τοποθετούνται στην ευθεία των πραγματικών αριθμών και την χωρίζουν σε περιοχές, καθεμία από τις οποίες αντιστοιχίζεται σε μία διακριτή τιμή. Οι περιοχές αυτές οριοθετούνται από τα

μέσα των αποστάσεων των διαδοχικών τιμών. Η απεικόνιση αυτή φαίνεται στο **Σχήμα 2.5** Αντιστοίχιση της συνεχούς τιμής στην πλησιέστερη διακριτή τιμή που μπορεί να λάβει η μεταβλητή. Με διακεκομμένες σημαίνονται τα μέσα των αποστάσεων μεταξύ δύο διαδοχικών διακριτών τιμών της μεταβλητής.. Καθώς δεν παρεμβάλλονται ακέραιοι αριθμοί, η προσαρμογή στον διακριτό χώρο είναι πιστότερη. Εξακολουθεί όμως να παραμένει μια σημαντική δυσκολία: αν για μια μεταβλητή οι μεταβολές της συνεχούς τιμής είναι μικρές, λόγω των τελεστών του αλγορίθμου, τότε η μεταβλητή δεν μπορεί να υπερπηδήσει το χάσμα μέχρι την επόμενη διακριτή τιμή.



Σχήμα 2.5 Αντιστοίχιση της συνεχούς τιμής στην πλησιέστερη διακριτή τιμή που μπορεί να λάβει η μεταβλητή. Με διακεκομμένες σημαίνονται τα μέσα των αποστάσεων μεταξύ δύο διαδοχικών διακριτών τιμών της μεταβλητής.

2.4 Μέθοδος Βελτιστοποίησης Σμήνους Σωματιδίων (ΒΣΣ)

2.4.1 Εισαγωγή

Πολλές στοχαστικές μέθοδοι βελτιστοποίησης εμπνέονται από την φύση και συγκεκριμένα από την κοινωνική συμπεριφορά πληθυσμών κάποιου είδους (Swarm Intelligence). Μια από αυτές είναι η Βελτιστοποίηση Σμήνους Σωματιδίων (Particle Swarm Optimization – PSO), που ως πληθυσμό χρησιμοποιεί ένα σμήνος πτηνών ή ψαριών καθώς αναζητούν τροφή ή κάποιον άλλο πόρο. Τα μέλη του σμήνους προσαρμόζουν την κίνηση τους για να αποφύγουν κινδύνους, επικοινωνούν μεταξύ τους και εκμεταλλεύονται την εμπειρία που έχουν αποκτήσει τα ίδια ή άλλα μέλη του πληθυσμού, σχετικά με την καλύτερη περιοχή αναζήτησης.

Εισήχθηκε από τους J. Kennedy και R. Eberhart το 1995 για τη βελτιστοποίηση συνεχών προβλημάτων, κυρτών ή μη, και στη συνέχεια αναπτύχθηκε εκτενέστερα και από τον Y. Shi το 1998. Οι Kennedy και Eberhart βασίστηκαν στην θεωρία του C.W. Reynolds για την προσομοίωση της περίπλοκης κίνησης των μελών ενός σμήνους πτηνών: δημιουργία απόστασης από τους γείτονές τους αν πλησιάσουν, προσανατολισμός προς την μέση κατεύθυνση των γειτόνων τους και προσπάθεια τοποθέτησης τους εντός του σμήνους, σε μία μέση θέση δηλαδή. Σε αυτούς τους τρεις κανόνες προσέθεσαν μία φωλιά, ώστε κάθε μέλος να έλκεται προς αυτήν, να θυμάται πότε ήταν πλησιέστερα στη φωλιά και να ανταλλάσει

πληροφορίες με τους γείτονές του για την τρέχουσα απόσταση από αυτήν. Έτσι όλα τα μέλη του σμήνους καταλήγουν στην φωλιά.

Η λογική αυτή μπορεί να εφαρμοστεί για βελτιστοποίηση προβλημάτων με τις ακόλουθες μετατροπές:

- Ο αλγόριθμος χρησιμοποιεί ένα σμήνος σωματιδίων (πληθυσμός).
- Κάθε σωματίδιο καταλαμβάνει μια θέση στο χώρο σχεδιασμού (πιθανή λύση).
- Η καταλληλότητα κάθε σωματιδίου αντιστοιχεί στην τιμή της αντικειμενικής συνάρτησης ή της συνάρτησης ποιότητας σε περίπτωση ποινής λόγω περιορισμών.
- Τα σωματίδια πετούν στον διανυσματικό χώρο έχοντας κάποια ταχύτητα.
- Η κατεύθυνση και το μέτρο της ταχύτητας αυτής επηρεάζονται από την καλύτερη θέση που έχει βρει μέχρι τότε το σωματίδιο. Επομένως πραγματοποιείται τοπική αναζήτηση όπου χρησιμοποιείται η προσωπική μνήμη και εμπειρία του σωματιδίου.
- Η κατεύθυνση και το μέτρο της ταχύτητας αυτής επηρεάζονται από την καλύτερη θέση που έχουν βρει μέχρι τότε οι γείτονες του σωματιδίου. Επομένως πραγματοποιείται καθολική αναζήτηση, όπου χρησιμοποιείται η κοινωνική μνήμη και εμπειρία ολόκληρου του σμήνους.
- Τελικά όλα τα σωματίδια συγκλίνουν σε τοπικά ή ολικά βέλτιστες θέσεις.

Η ΒΣΣ έχει δοκιμαστεί σε ποικίλα προβλήματα βελτιστοποίησης (Bochenek και Foryś 2006, He και Wang 2007, Liang και Suganthan 2006, Mezura-Montes και Lopez-Ramirez 2007, Munoz-Zavala et al. 2006; Perez και Behdinan 2007, Ye et al. 2007) και αποδείχθηκε ιδιαίτερα αποδοτική συγκριτικά με άλλους μεταευριστικούς αλγορίθμους. Παρουσιάζει δηλαδή γρήγορη σύγκλιση και τελική λύση που είναι τουλάχιστον το ίδιο καλή. Επιπλέον είναι εύκολη στην εφαρμογή και τον προγραμματισμό της, αφού χρησιμοποιεί απλούς τελεστές (πρόσθεση και πολλαπλασιασμό διανυσμάτων) και απαιτεί τη ρύθμιση λίγων παραμέτρων. Λειτουργεί αποδοτικά για μη γραμμικούς, μη κυρτούς και ή συνεχείς χώρους σχεδιασμού, οπότε μπορεί να εφαρμοστεί σε προβλήματα μηχανικής και σχεδιασμού κατασκευών.

2.4.2 Ο βασικός αλγόριθμος ΒΣΣ

Για κάθε σωματίδιο j και επανάληψη t του αλγορίθμου ορίζουμε:

- $\boldsymbol{x}_j(t)$: Η τρέχουσα θέση του σωματιδίου στο διανυσματικό χώρο, το διάνυσμα δηλαδή που περιέχει τις τιμές των μεταβλητών σχεδιασμού.
- $\boldsymbol{v}_j(t)$: Η τρέχουσα ταχύτητα του σωματιδίου
- $\boldsymbol{x}_j^{Pbest}$: Η καλύτερη θέση που έχει βρει το σωματίδιο j μέχρι την επανάληψη t, δηλαδή η λύση με την βέλτιστη τιμή της αντικειμενικής συνάρτησης.
- $\boldsymbol{x}^{Gbest}$: Η καλύτερη θέση που έχει βρει ολόκληρο το σμήνος μέχρι την επανάληψη t.

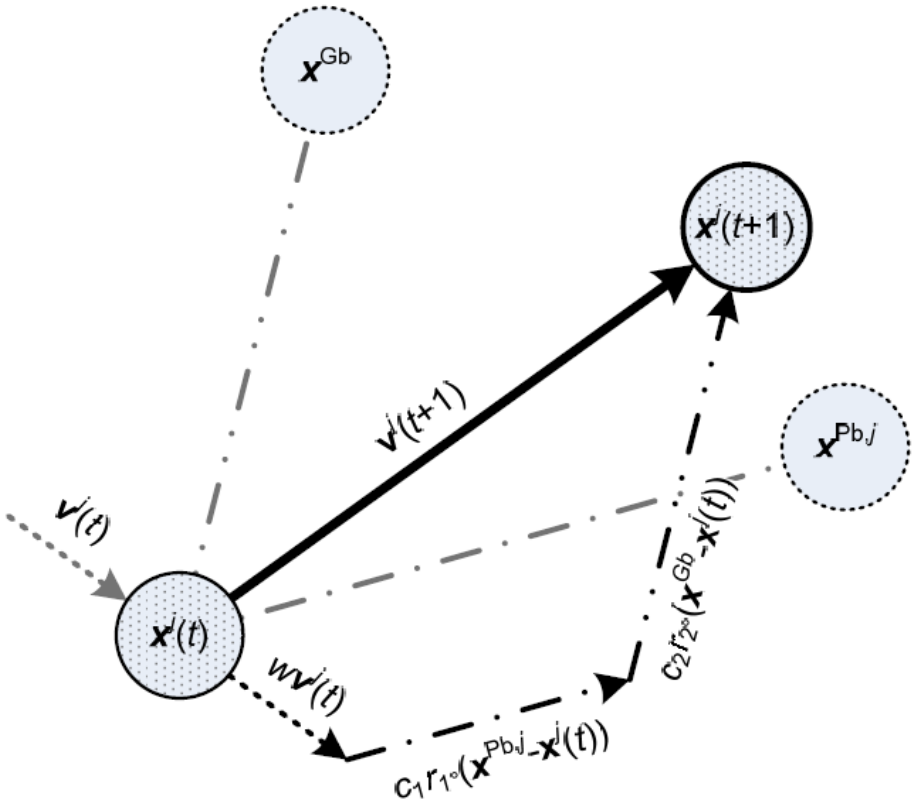
- c_1 : Συντελεστής επιτάχυνσης που εκφράζει την εμπιστοσύνη του σωματιδίου στην καλύτερη θέση που έχει βρει το ίδιο. «Προσωπική» παράμετρος.
- c_2 : Συντελεστής επιτάχυνσης που εκφράζει την εμπιστοσύνη του σωματιδίου στην καλύτερη θέση που έχει βρεθεί από ολόκληρο το σμήνος. «Κοινωνική» παράμετρος.
- $rand()$: Συνάρτηση που επιστρέφει τυχαίους αριθμούς, οι οποίοι υπακούν στην ομοιόμορφη κατανομή εντός του διαστήματος [0, 1].
- w: Συντελεστής αδράνειας που εκφράζει την τάση του σωματιδίου να διατηρεί την ταχύτητα που είχε στην προηγούμενη επανάληψη.

Οι βασικές εξισώσεις ανανέωσης της θέσης και ταχύτητας του σωματιδίου είναι:

$$\boldsymbol{v}_j(t+1) = w * \boldsymbol{v}_j(t) + c_1 * rand() * \left[\boldsymbol{x}_j^{Pbest} - \boldsymbol{x}_j(t)\right] + c_2 * rand() * \left[\boldsymbol{x}^{Gbest} - \boldsymbol{x}_j(t)\right] \quad \textbf{(2.8)}$$

$$\boldsymbol{x}_j(t+1) = \boldsymbol{x}_j(t) + \boldsymbol{v}_j(t+1) \quad \textbf{(2.9)}$$

Στο παρακάτω σχήμα φαίνεται μια γραφική αναπαράσταση των παραπάνω εξισώσεων στην περίπτωση του δισδιάστατου χώρου σχεδιασμού.



Σχήμα 2.6 Ανανέωση της θέσης και της ταχύτητας ενός σωματιδίου, στην περίπτωση δύο μεταβλητών σχεδιασμού.

Έτσι ο βασικός αλγόριθμος για συνεχή και χωρίς περιορισμούς προβλήματα βελτιστοποίησης είναι:

```
1.  Επανέλαβε για κάθε σωματίδιο i
2.      Εκκίνηση της θέσης διασκορπίζοντας τα σωματίδια τυχαία στο χώρο σχεδιασμού.
3.  Τέλος επανάληψης
4.  Επανέλαβε
5.      Επανέλαβε για κάθε σωματίδιο i
6.          Υπολογισμός της συνάρτησης ποιότητας (fitness) για την τρέχουσα θέση.
7.          Αν η τρέχουσα τιμή ποιότητας είναι καλύτερη από την προσωπικά
            βέλτιστη Pbest_i
8.              Θέσε την τρέχουσα τιμή ποιότητας ως τη νέα Pbest_i
9.              Θέσε την τρέχουσα θέση ως τη νέα x_i^Pbest
10.         Τέλος αν
11.     Τέλος επανάληψης
12.     Βρες την καλύτερη τιμή ποιότητας μεταξύ όλων των σωματιδίων.
13.     Αν είναι καλύτερη από την τρέχουσα Gbest
14.         Θέσε την ως τη νέα Gbest
15.         Θέσε την αντίστοιχη θέση ως τη νέα x^Gbest
16.     Τέλος αν
17.     Επανέλαβε για κάθε σωματίδιο i
18.         Υπολόγισε τη νέα ταχύτητα από την εξίσωση ( 2.8 )
19.         Υπολόγισε τη νέα θέση του σωματιδίου από την εξίσωση ( 2.9 )
20.     Τέλος επανάληψης
21. Μέχρις ότου επιτευχθεί σύγκλιση
```

Σχήμα 2.7 Ψευδό-κώδικας για τον βασικό αλγόριθμο ΒΣΣ, μόνο με συνεχείς μεταβλητές και χωρίς περιορισμούς.

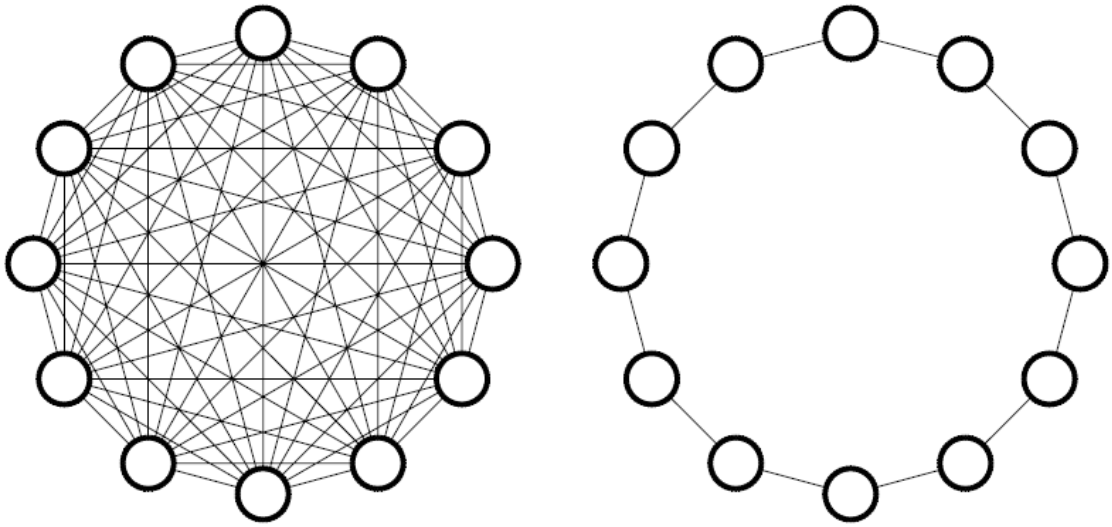
2.4.3 Παράμετροι της ΒΣΣ

2.4.3.1 Δομή Σμήνους

Κάθε σωματίδιο του σμήνους ανταλλάσσει πληροφορίες με άλλα σωματίδια στη γειτονιά του. Ο προσδιορισμός της γειτονιάς αυτής όμως αφήνεται στον χρήστη της μεθόδου ΒΣΣ. Οι δύο διαφορετικές δομές σμήνους είναι:

- Το σμήνος Gbest, όπου όλα τα σωματίδια του σμήνους θεωρείται ότι ανήκουν στην γειτονιά οποιουδήποτε από αυτά. Έτσι η θέση που χρησιμοποιείται στον κοινωνικό όρο της εξίσωσης ανανέωσης της ταχύτητας (2.8) αντιστοιχεί στην καλύτερη λύση που έχει βρεθεί από όλα τα σωματίδια μέχρι την τρέχουσα επανάληψη. Η τοπολογία αυτή φαίνεται στο **Σχήμα 2.8 α)**. Στις παρακάτω εφαρμογές θα χρησιμοποιηθεί αυτή η τοπολογία.

- Το σμήνος Lbest, όπου κάποια μόνο σωματίδια θεωρείται ότι ανήκουν στη γειτονιά κάποιου από αυτά. Έτσι λαμβάνεται η βέλτιστη λύση που έχει βρεθεί μόνο από αυτά. Η συσχέτιση αυτή μπορεί να γίνει με βάση τη γεωμετρική θέση των σωματιδίων, δηλαδή να λαμβάνονται τα σωματίδια που απέχουν την ελάχιστη απόσταση. Συχνότερα όμως απλά τίθενται αυθαίρετα οι γειτονιές των σωματιδίων στην αρχή του αλγορίθμου. Εξάλλου λόγω επικοινωνίας, τα σωματίδια εντός της ίδιας γειτονιάς θα τείνουν να πλησιάσουν και γεωμετρικά. Το συνηθέστερα χρησιμοποιούμενο σμήνος Lbest έχει τη μορφή δακτυλίου (Ring Lattice), όπου κάθε σωματίδιο επικοινωνεί μόνο με δύο γειτονικά του όπως φαίνεται στο **Σχήμα 2.8 β)**.



Σχήμα 2.8 Γραφική αναπαράσταση των δύο συνηθέστερα χρησιμοποιούμενων τοπολογιών.

Η τοπολογία του σμήνους εκφράζει το βαθμό συνδεσιμότητας των σωματιδίων και επηρεάζει σε μεγάλο βαθμό τη συμπεριφορά της μεθόδου. Το πλήρως συνδεδεμένο σμήνος (Fully Connected ή Fully Informed) Gbest πετυχαίνει την αμεσότερη και άρα ταχύτερη ανταλλαγή πληροφοριών. Έτσι τα σωματίδια συγκλίνουν γρήγορα στην βέλτιστη λύση. Παγιδεύεται όμως εύκολα σε τοπικά ελάχιστα, αφού μόλις ένα σωματίδιο παγιδευτεί, έλκει και τα υπόλοιπα. Στο άλλο άκρο βρίσκεται το σμήνος μορφής δακτυλίου. Σε αυτό οι πληροφορίες κινούνται με τον πλέον αργό ρυθμό, αφού πρέπει να περάσουν διαδοχικά από τα μισά σωματίδια μέχρι να φτάσουν στο αντιδιαμετρικό άκρο. Έτσι το σμήνος συγκλίνει πιο αργά, αλλά εξερευνεί περισσότερο μέρος του χώρου σχεδιασμού και δεν παγιδεύεται εύκολα σε τοπικά ελάχιστα.

Πολλές άλλες τοπολογίες έχουν σχεδιασθεί και χρησιμοποιηθεί. Οι J.Kennedy και R. Mendes δοκίμασαν και συνέκριναν διάφορες από αυτές (2002). Μπορούν επίσης να χρησιμοποιηθούν μη πάγιες δομές που προσαρμόζονται δυναμικά. Γενικά πάντως όσο στενότερη είναι η σύνδεση των σωματιδίων, τόσο αυξάνεται η τάση των σωματιδίων να συγκλίνουν στις καλύτερες λύσεις των πρώτων επαναλήψεων. Έτσι δεν εξερευνούν περιοχές του χώρου σχεδιασμού που

βρίσκονται πιο μακριά και πιθανόν να περιέχουν την καθολικά βέλτιστη λύση. Από την άλλη όσο χαλαρώνει η σύνδεση, τόσο μειώνεται η ταχύτητα, τα σωματίδια ψάχνουν στα τυφλά και σπαταλούν χρόνο και πόρους σε μάταιες δοκιμές.

Εξίσου σημαντικός με την μορφολογία του σμήνους είναι και ο πληθυσμός του. Μεγάλος πληθυσμός μειώνει την πιθανότητα παγίδευσης σε τοπικά ελάχιστα, αλλά αυξάνει τον αριθμό των δοκιμών. Για πολλά προβλήματα αρκεί πληθυσμός *10-40* σωματιδίων, ενώ για πιο πολύπλοκα μπορούν να χρησιμοποιηθούν και *50 - 100*.

2.4.3.2 Όρια χώρου σχεδιασμού

Όπως αναφέρθηκε στο κεφάλαιο 1, για κάθε συνεχή μεταβλητή σχεδιασμού x_i υπάρχουν ένα άνω και ένα κάτω επιτρεπόμενο όριο $l_i \le x_i \le u_i$. Κατά την κίνησή τους είναι πιθανόν τα σωματίδια να ξεπεράσουν το όριο αυτό στη συγκεκριμένη διάσταση *i*. Στην περίπτωση αυτή ως τιμή της μεταβλητής x_i τίθεται το όριο που μόλις παραβιάστηκε ($x_i = u_i$ ή $x_i = l_i$). Ταυτόχρονα μηδενίζεται και η ταχύτητα στην διεύθυνση αυτή ($v_i = 0$), ώστε το σωματίδιο να μην συνεχίσει να αναζητεί λύσεις εκτός του χώρου σχεδιασμού, λόγω της αδράνειας του.

2.4.3.3 Ταχύτητα

Αν οι ταχύτητα ενός σωματιδίου αυξηθεί ανεξέλεγκτα σε κάποια διεύθυνση i, τότε αυτό θα διασχίζει όλο το διαθέσιμο διάστημα $[l_i, u_i]$ σε λίγες επαναλήψεις ή ακόμα και σε μία μόνο, χωρίς να μπορεί να επισκεφθεί ενδιάμεσες τιμές. Γι' αυτό υπάρχει ένα άνω όριο στο μέτρο της ταχύτητας v_i και όταν αυτό ξεπεραστεί, θέτουμε την ταχύτητα στην διεύθυνση i ίση με $\pm v_{i,max}$ ανάλογα με τη φορά που είχε. Το μέγεθος αυτό είναι διαφορετικό για κάθε διεύθυνση και συνήθως τίθεται ίσο με το μισό της απόστασης $u_i - l_i$. Έτσι ορίζεται το διάνυσμα $\boldsymbol{v}_{max}$, που περιέχει τα όρια $v_{i,max}$ σε κάθε διεύθυνση.

2.4.3.4 Αδράνεια

Ας εξετάσουμε την εξίσωση (2.8). Το πρώτο μέρος της ($w * \boldsymbol{v}_j(t)$) εκφράζει το βαθμό εξερεύνησης των σωματιδίων. Μεγάλες τιμές του συντελεστή αδράνειας *w*, αυξάνουν το μέτρο της ταχύτητας των σωματιδίων και έτσι αυτά εξερευνούν περισσότερο το χώρο σχεδιασμού, πραγματοποιώντας πιο καθολική αναζήτηση. Αντίθετα μικρές τιμές του *w* συγκρατούν την ταχύτητα και αναγκάζουν τα σωματίδια να περιορίζονται σε περιοχές κοντά στους πόλους έλξης που είναι οι θέσεις των Gbest και Pbest. Στην περίπτωση μηδενικής αδράνειας, άρα και ταχύτητας, τα σωματίδια πραγματοποιούν τοπικές αναζητήσεις. Μάλιστα αυτή είναι μια αρκετά δημοφιλής παραλλαγή της βασικής μεθόδου ΒΣΣ (Accelerated PSO).

Η παράμετρος *w* δεν υπήρχε στην αρχική έκδοση του αλγορίθμου, αλλά προστέθηκε αργότερα (1998) από τους Y. Shi και R. Eberhart. Πραγματοποίησαν δοκιμές επί γνωστών προβλημάτων με διάφορες τιμές του συντελεστή αδρανείας και κατέληξαν ότι η μέθοδος κατάφερνε να βρει την (ήδη γνωστή) βέλτιστη λύση με μεγαλύτερη πιθανότητα αν ληφθούν τιμές αδρανείας στο διάστημα [0.9, 1.2]. Στατικές όμως τιμές του *w* επιβραδύνουν την σύγκλιση του αλγορίθμου στις τελευταίες επαναλήψεις. Μεγάλες τιμές αδράνειας βοηθούν την καθολική εξερεύνηση στα πρώτα βήματα, ώστε να εντοπιστεί η περιοχή του βέλτιστου. Ωστόσο μόλις βρεθεί η περιοχή αυτή, απαιτούνται μικρότερα βήματα για να προσεγγιστεί ακριβώς το βέλτιστο.

Γι' αυτό οι Eberhart και Shi πρότειναν τη χρήση δυναμικής αδράνειας, που μειώνεται γραμμικά ή μη γραμμικά όσο προχωρούν οι επαναλήψεις. Στην παρούσα εργασία θα χρησιμοποιηθεί γραμμικά μειούμενος συντελεστής αδράνειας σύμφωνα με την εξίσωση (2.10):

$$w(t+1) = w_{max} - \frac{w_{max} - w_{min}}{t_{max}} * t \qquad \textbf{(2.10)}$$

Όπου $t = 0, 1, \dots, t_{max}$ είναι ο αριθμός της τρέχουσας επανάληψης, , t_{max} είναι το μέγιστο πλήθος επαναλήψεων της μεθόδου και w_{max} και w_{min} είναι η μέγιστη και ελάχιστη τιμή του συντελεστή αδρανείας αντίστοιχα.

2.4.3.5 Εμπειρία και τυχαιότητα

Οι παράμετροι c_1 και c_2 είναι συντελεστές επιτάχυνσης που εκφράζουν το βαθμό έλξης του σωματιδίου προς την καλύτερη λύση που έχει συναντηθεί από την προσωπική εμπειρία του σωματιδίου ή από την κοινωνική εμπειρία των σωματιδίων στην γειτονιά του αντίστοιχα. Συγκριτικά μεγαλύτερη προσωπική παράμετρος προκαλεί τυχαία περιπλάνηση και απομόνωση των σωματιδίων, ενώ μεγαλύτερη κοινωνική παράμετρος προκαλεί πρόωρη σύγκλιση του σμήνους σε τοπικά ελάχιστα. Οι Kennedy και Eberhart προτείνουν και για τις δύο παραμέτρους την τιμή *2.0*, ώστε να εξισορροπηθούν οι δύο αυτές επιρροές και το σωματίδιο να έχει 50% πιθανότητα υπερπήδησης του στόχου του. Σύμφωνα με τους Perez και Behdinan (2007) μπορούν να χρησιμοποιηθούν και άλλες τιμές, αρκεί:

$$0 < c_1 + c_2 < 4 \qquad \textbf{(2.11)}$$

Η συνάρτηση *rand()* $\in [0,1]$ παρέχει έναν στοχαστικό παράγοντα της μεθόδου. Έτσι οι συνιστώσες της ταχύτητας προς την προσωπικά και καθολικά βέλτιστη θέση πολλαπλασιάζονται με έναν, διαφορετικό η καθεμία, τυχαίο αριθμό σε όλες τις διευθύνσεις τους. Αυτή η στοχαστική συνιστώσα της κίνησης των σωματιδίων είναι απαραίτητη ώστε να αποφευχθεί ο συγχρονισμός των σωματιδίων και η σταθερή και ομοιόμορφη κατεύθυνσή τους.

2.4.4 Κριτήρια τερματισμού

Ως στοχαστικός, μεταευριστικός αλγόριθμος, η ΒΣΣ δεν μπορεί να βρει την ακριβή λύση του προβλήματος. Οπότε απαιτούνται κάποια κριτήρια σύγκλισης, ώστε να μπορεί να τερματιστεί. Πλην της περίπτωσης δοκιμής της μεθόδου δε γνωρίζουμε εκ των προτέρων την λύση του προβλήματος, οπότε δεν ξέρουμε αν προσεγγίζεται ικανοποιητικά από την τρέχουσα καλύτερη θέση του σμήνους.

Ένα εύκολο στην εφαρμογή του κριτήριο σύγκλισης είναι ο μέγιστος αριθμός επαναλήψεων. Γενικά ο προσδιορισμός του απαραίτητου αριθμού επαναλήψεων εξαρτάται από το μέγεθος και την πολυπλοκότητα του προβλήματος. Επίσης, επειδή υπάρχει πάντα η πιθανότητα παγίδευσης σε τοπικά ελάχιστα, αύξηση των επαναλήψεων δεν συνεπάγεται αυτόματα και βελτίωση της λύσης. Παρόλα αυτά ένα όριο στις επαναλήψεις δεν κοστίζει να εφαρμοστεί ταυτόχρονα με κάποιο άλλο, πιο «έξυπνο» κριτήριο σύγκλισης. Στην παρούσα εργασία ο μέγιστος αριθμός επαναλήψεων είναι ένα από τα δύο κριτήρια που συνδυάζονται, δεδομένου ότι χρειάζεται και για τον προσδιορισμό της τρέχουσας αδράνειας, βλ. (2.10).

Το άλλο κριτήριο σύγκλισης που θα χρησιμοποιηθεί αφορά τον ελάχιστο ρυθμός βελτίωσης. Έστω $Gbest_t$ η βέλτιστη τιμή της αντικειμενικής συνάρτησης που έχει βρεθεί καθολικά από το σμήνος μέχρι την επανάληψη t και f_m το ελάχιστο ποσοστό βελτίωσης, για το οποίο η λύση συγκλίνει ικανοποιητικά. Τότε μπορούμε να θεωρήσουμε ότι η μέθοδος συνέκλινε, αν ο ρυθμός βελτίωσης του $Gbest_t$ κατά τις τελευταίες k_f επαναλήψεις είναι μικρότερος από τον απαραίτητο f_m :

$$\frac{Gbest_{t-kf+1} - Gbest_t}{Gbest_{t-kf+1}} \leq f_m \qquad \textbf{(2.12)}$$

2.4.5 Συγκεντρωτικός πίνακας

Στον παρακάτω πίνακα αναφέρονται συνοπτικά οι συνηθέστερες από τις χρησιμοποιούμενες παραμέτρους της μεθόδου ΒΣΣ και κάποιες ενδεικτικές τιμές που μπορούν να επιλεχθούν.

Πίνακας 2.1 Βασικές παράμετροι της Βελτιστοποίησης Σμήνους Σωματιδίων.

Σύμβολο	Περιγραφή	Λεπτομέρειες
n	Αριθμός μεταβλητών σχεδιασμού	Καθορίζεται από το πρόβλημα βελτιστοποίησης
NP	Πληθυσμός του σμήνους σωματιδίων	Γενικά 10 – 40. Για δύσκολα προβλήματα 50 - 100
$\mathbf{l}_i$, $\mathbf{u}_i$	Διανύσματα που περιέχουν το κάτω και άνω όριο των n μεταβλητών σχεδιασμού αντίστοιχα.	Καθορίζονται από το πρόβλημα βελτιστοποίησης.
$\mathbf{v}_{max}$	Διάνυσμα που περιέχει τις μέγιστες ταχύτητες που μπορούν να έχουν τα σωματίδια, σε καθεμία από τις n διευθύνσεις.	Συνήθως χρησιμοποιείται το μισό της απόστασης κάτω και άνω ορίου ανα διεύθυνση, δηλαδή: $v_i^{max} = (u_i - l_i)/2$, $i = 1,2, \dots, n$
w	Συντελεστής αδράνειας	Σταθερή τιμή 0.9 ως 1.2 . Διαφορετικά μπορεί να ανανεώνεται σε κάθε επανάληψη.
c_1,c_2	Συντελεστές επιτάχυνσης	Συνήθως $c_1 = c_2 = 2.0$. Για διαφορετικές τιμές πρέπει να ισχύει: $0 < c_1 + c_2 < 4$
t_{max}	Μέγιστος επιτρεπόμενος αριθμός επαναλήψεων.	Η επιλογή του εξαρτάται από το μέγεθος του προβλήματος n, και το πλήθος των σωματιδίων NP
k_f	Αριθμός επαναλήψεων για τον οποίο η σχετική βελτίωση της αντικειμενικής συνάρτησης είναι ικανοποιητική.	Αν η σχετική βελτίωση της αντικειμενικής συνάρτησης στη διάρκεια των k_f τελευταίων επαναλήψεων δεν ξεπερνά την ανοχή f_m, τότε έχει επιτευχθεί σύγκλιση.
f_m	Ελάχιστη σχετική βελτίωση της αντικειμενικής συνάρτησης ώστε να επιτευχθεί σύγκλιση.	

2.4.6 Ο Αλγόριθμος ΒΣΣ για το Βέλτιστο Σχεδιασμό Κατασκευών

Η μέθοδος ΒΣΣ αναπτύχθηκε για την αντιμετώπιση προβλημάτων βελτιστοποίησης χωρίς περιορισμούς. Για την αντιμετώπιση περιορισμών έχουν εφαρμοστεί αρκετές τεχνικές που εστιάζουν στην απόρριψη των μη εφικτών λύσεων (Hu et al., 2003) ή την διόρθωση τους και ανακατεύθυνση των σωματιδίων προς την εφικτή περιοχή (Venter και Sobieszczanski-Sobieski, 2004). Στις αριθμητικές εφαρμογές της παρούσας εργασίας θα χρησιμοποιηθεί η συνάρτηση ποινής που παρουσιάστηκε στην ενότητα 2.2.3.

Στην περίπτωση που κάποιες μεταβλητές είναι διακριτές, χρησιμοποιείται η τεχνική που περιγράφηκε στην ενότητα **2.3.3**, δηλαδή θα χρησιμοποιούνται συνεχείς μεταβλητές για την ανανέωση των διανυσμάτων ταχύτητας και θέσης, αλλά πριν το υπολογισμό της αντικειμενικής συνάρτησης και των συναρτήσεων περιορισμών, οι συνεχείς τιμές θα αντιστοιχίζονται στις πλησιέστερες διακριτές.

Με αυτές τις τροποποιήσεις ο αλγόριθμος που τελικά χρησιμοποιείται στις αριθμητικές εφαρμογές φαίνεται στο **Σχήμα 2.9**:

1. Επανέλαβε για κάθε σωματίδιο i
2. Εκκίνηση της θέσης διασκορπίζοντας τα σωματίδια τυχαία στο χώρο σχεδιασμού.
3. Ανάλυση FEM και έλεγχος περιορισμών, ώστε να βρεθεί αν η τυχαία θέση είναι εφικτή ή όχι.
4. Αν το πλήθος των μη εφικτών σωματιδίων έχει ξεπεράσει το μέγιστο πλήθος των μη εφικτών αρχικών σχεδιασμών.
5. Όσο αυτή η θέση δεν είναι εφικτή
6. Δοκιμή νέας τυχαίας θέσης.
7. Ανάλυση FEM και έλεγχος περιορισμών, ώστε να βρεθεί αν η νέα τυχαία θέση είναι εφικτή ή όχι.
8. Τέλος επανάληψης
9. Τέλος αν
10. Υπολογισμός αντικειμενικής συνάρτησης για την τρέχουσα θέση.
11. Υπολογισμός ποινής (penalty) από την εξίσωση (2.5)
12. Υπολογισμός συνάρτησης ποιότητας (fitness) από την εξίσωση (2.7)
13. Θέσε την τρέχουσα τιμή ποιότητας ως και την τρέχουσα θέση ως τη $\boldsymbol{Pbest_i}$
14. Θέσε την τρέχουσα θέση ως τη $\boldsymbol{x_i^{Pbest}}$
15. Τέλος επανάληψης
16. Βρες την καλύτερη τιμή ποιότητας, που αντιστοιχεί σε εφικτή λύση, μεταξύ όλων των σωματιδίων.
17. Θέσε την ως $\boldsymbol{Gbest}$ και θέσε την αντίστοιχη θέση ως $\boldsymbol{x^{Gbest}}$
18. Επανέλαβε
19. Επανέλαβε για κάθε σωματίδιο i
20. Υπολόγισε τη νέα ταχύτητα από την εξίσωση (2.8)
21. Αν σε κάποια διεύθυνση i : $v_i(t) < -v_i^{max}$ ή $v_i(t) > v_i^{max}$
22. Θέσε αντίστοιχα $v_i(t) = -v_i^{max}$ ή $v_i(t) = v_i^{max}$
23. Τέλος αν
24. Υπολόγισε τη νέα θέση του σωματιδίου από την εξίσωση (2.9)
25. Αν σε κάποια διεύθυνση i: $x_i(t) < l_i$ ή $x_i(t) > u_i$
26. Θέσε αντίστοιχα $x_i(t) = l_i$ ή $x_i(t) = u_i$
27. Τέλος αν
28. Αν η μεταβλητή x_j είναι διακριτή
29. Αντιστοίχησε την συνεχή τιμή στην πλησιέστερη διακριτή του συνόλου τιμών της μεταβλητής.
30. Τέλος αν

```
31.        Υπολογισμός αντικειμενικής συνάρτησης για την τρέχουσα θέση.
           Υπολογισμός ποινής (penalty) από την εξίσωση ( 2.5 ).
32.        Υπολογισμός συνάρτησης ποιότητας (fitness) από την εξίσωση ( 2.7 ).
33.        Αν η τρέχουσα τιμή ποιότητας είναι καλύτερη από την προσωπικά
34.        βέλτιστη Pbest_i
35.            Θέσε την τρέχουσα τιμή ποιότητας ως τη νέα Pbest_i
36.            Θέσε την τρέχουσα θέση ως τη νέα x_i^Pbest
37.        Τέλος αν
38.    Τέλος επανάληψης
39.    Βρες την καλύτερη τιμή ποιότητας μεταξύ όλων των σωματιδίων.
40.    Αν είναι καλύτερη από την τρέχουσα Gbest
41.        Θέσε την ως τη νέα Gbest
42.        Θέσε την αντίστοιχη θέση ως τη νέα x^Gbest
43.    Τέλος αν
44. Μέχρις ότου οι επαναλήψεις φθάσουν το μέγιστο αριθμό τους ή η σχετική
    βελτίωση του Gbest είναι μικρότερη από f_m για τις τελευταίες k_f επαναλήψεις.
```

Σχήμα 2.9 Ψευδό-κώδικας για την ΒΣΣ, όπως χρησιμοποιείται στις αριθμητικές εφαρμογές.

2.5 Μέθοδος Διαφορικής Εξέλιξης (ΔΕ)

2.5.1 Εισαγωγή

Η Διαφορική Εξέλιξη (Differnetial Evolution) είναι ένας απλός και εύκολος στην εφαρμογή του, αλλά απρόσμενα αποτελεσματικός μεταευριστικός αλγόριθμος καθολικής βελτιστοποίησης, που προτάθηκε από τους R. Storn & K. Price το 1997, για την επίλυση συνεχών προβλημάτων χωρίς περιορισμούς. Ανήκει στην κατηγορία των Εξελικτικών Αλγορίθμων (Evolutionary Algorithms), δηλαδή εμπνέεται από τη γενετική και την βιολογική εξέλιξη των οργανισμών. Όπως και άλλοι εξελικτικοί αλγόριθμοι (Γενετικοί Αλγόριθμοι, Εξελικτικές Στρατηγικές), χρησιμοποιεί ένα πληθυσμό από δυνατές λύσεις που σε κάθε γενιά, δηλαδή σε κάθε επανάληψη της μεθόδου, δίνουν νέες λύσεις ως απογόνους. Για τη γένεση των απογόνων χρησιμοποιούνται οι στοχαστικοί τελεστές της μετάλλαξης, του ανασυνδυασμού και της επιλογής, οι οποίοι όμως ορίζονται με διαφορετικό τρόπο από άλλους εξελικτικούς αλγόριθμους. Πιο συγκεκριμένα ο πληθυσμός κάθε γενιάς δημιουργεί δοκιμαστικές λύσεις προσθέτοντας την σταθμισμένη διαφορά δύο διανυσμάτων σχεδιασμού σε ένα τρίτο.

Η μέθοδος της Διαφορικής Εξέλιξης παρουσιάζει αρκετά πλεονεκτήματα έναντι άλλων στοχαστικών ή αιτιοκρατικών αλγορίθμων που την καθιστούν ιδιαίτερα ανταγωνιστική για πρακτικές εφαρμογές:

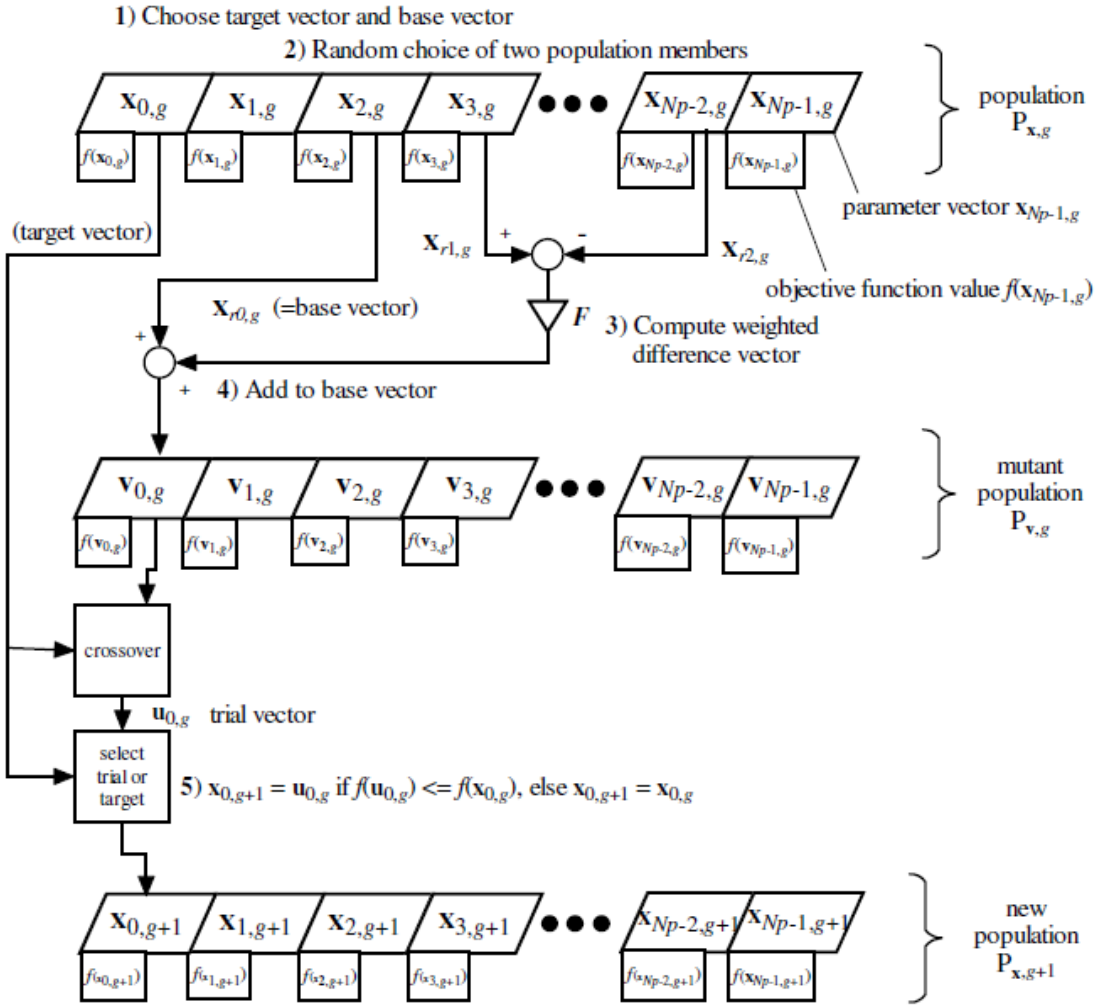
- Μπορεί να αντιμετωπίσει μη παραγωγίσιμες και μη γραμμικές αντικειμενικές συναρτήσεις με πολλαπλά τοπικά ελάχιστα.
- Λόγω της χρήσης πληθυσμού πιθανών λύσεων, μπορεί να εφαρμοστεί σε παράλληλο περιβάλλον, ώστε να μειωθεί ο υπολογιστικός χρόνος για απαιτητικές αντικειμενικές συναρτήσεις και συναρτήσεις περιορισμών.
- Ευκολία χρήσης, δηλαδή μικρός αριθμός παραμέτρων ελέγχου που πρέπει να επιλέξει ο χρήστης και που εξαρτώνται από το εκάστοτε πρόβλημα. Έτσι μπορεί να εφαρμοστεί χωρίς να χρειάζεται ειδίκευση στον κλάδο της βελτιστοποίησης.
- Σχετικά με άλλες μεθόδους, συγκλίνει στο καθολικά βέλτιστο σχεδιασμό γρήγορα, δηλαδή με μικρό αριθμό δοκιμαστικών λύσεων. Επιπλέον ο ρυθμός σύγκλισης είναι σταθερός όταν η μέθοδος επαναλαμβάνεται πολλές φορές για το ίδιο πρόβλημα, όπως φάνηκε σε πειραματικές δοκιμές.

Η Διαφορική Εξέλιξη έχει συγκριθεί με άλλους μεταευριστικούς αλγορίθμους (εξελικτικούς και μη) σε έναν αριθμό ερευνών. Οι εφευρέτες της ΔΕ (Storn &Price, 1997) την συνέκριναν με τις μεθόδους Annealed Nelder and Mead (Press 1992), Adaptive Simulated Annealing (Ingber 1993), Breeder Genetic Algorithm (Muehlenbein 1993), Evolutionary Algorithm with Soft Genetic Operators (Voigt 1995) και Stochastic Differential Equations (Aluffi-Pentini et al. 1985). Τα αποτελέσματα έδειξαν ότι η ΔΕ έβρισκε πάντα τη βέλτιστη λύση, συνήθως ταχύτερα από τις υπόλοιπες μεθόδους και σπάνια απαιτούσε προσαρμογή των λιγοστών παραμέτρων ελέγχου της. Σε άλλη έρευνα πάνω σε προβλήματα με περιορισμούς, φάνηκε ότι η ΔΕ υπερτερεί της Βελτιστοποίησης Σμήνους Σωματιδίων ως προς την ταχύτητα σύγκλισης και την

επαναληψιμότητα των αποτελεσμάτων. Στον πρώτο διεθνή διαγωνισμό Εξελικτικής Βελτιστοποίησης (ICEO 1996), η ΔΕ ήταν η ταχύτερη μέθοδος βελτιστοποίησης μετά από δύο ντετερμινιστικούς αλγορίθμους που πάσχουν από περιορισμένα πεδία εφαρμογής.

2.5.2 Ο βασικός αλγόριθμος ΔΕ

Η ΔΕ χρησιμοποιεί έναν πληθυσμό δυνατών λύσεων. Αρχικά ο πληθυσμός επιλέγεται τυχαία. Σε κάθε γενιά g (δηλαδή επανάληψη) και για κάθε μέλος-διάνυσμα του πληθυσμού $\boldsymbol{x}_{i,g}$, δημιουργείται ένα μεταλλαγμένο (mutant) διάνυσμα $\boldsymbol{v}_{i,g}$ χρησιμοποιώντας των τελεστή μετάλλαξης (mutation). Έπειτα δημιουργείται ένα δοκιμαστικό (trial) διάνυσμα $\boldsymbol{u}_{i,g}$ χρησιμοποιώντας τον τελεστή ανασυνδυασμού (recombination/crossover). Τέλος το νέο διάνυσμα του πληθυσμού $\boldsymbol{x}_{i,g+1}$ παράγεται με τον τελεστή επιλογής (selection) ανάμεσα στο αρχικό και το δοκιμαστικό διάνυσμα. Σχηματικά ο αλγόριθμος φαίνεται στο παρακάτω διάγραμμα ροής (Dobb 1997):



Σχήμα 2.10 Διάγραμμα ροής του κλασσικού αλγορίθμου Διαφορικής Εξέλιξης.

2.5.3 Οι γενετικοί τελεστές

Στη συνέχεια αναλύονται οι τελεστές με τους οποίους ανανεώνονται οι δυνατές λύσεις (ή διανύσματα) του πληθυσμού σε κάθε επανάληψη (ή γενιά όπως συνήθως αναφέρεται στους εξελικτικούς αλγόριθμους). Ορίζονται οι παρακάτω συμβολισμοί:

- NP: Το μέγεθος του πληθυσμού των διαφορετικών λύσεων σε κάθε γενιά.
- g: Η τρέχουσα γενιά, δηλαδή η τρέχουσα επανάληψη της μεθόδου.
- $\boldsymbol{x}_{i,g}$: Ένα διάνυσμα (πιθανή λύση) του πληθυσμού στη γενιά g. $i = 1, 2, \dots, NP$
- $\boldsymbol{v}_{i,g}$: Προσωρινό διάνυσμα μετά από μετάλλαξη (mutant vector).
- $\boldsymbol{u}_{i,g}$: Προσωρινό διάνυσμα μετά από ανασυνδυασμό (trial vector).
- F: Παράμετρος ελέγχου της μεθόδου που ονομάζεται διαφορικό βάρος. Συνήθως $F \in [0,2]$.
- CR: Παράμετρος ελέγχου της μεθόδου που ονομάζεται πιθανότητα crossover. Λαμβάνει τιμές στο διάστημα $CR \in [0,1]$.
- $f(\boldsymbol{x})$: Η αντικειμενική συνάρτηση.

Έχουν αναπτυχθεί αρκετές παραλλαγές για τους τελεστές της ΔΕ. Για λόγους ευκολίας κατά την περιγραφή τους χρησιμοποιείται η συντομογραφία DE/x/y/z. Οι πρώτοι δείκτες αναφέρονται στην μετάλλαξη: x είναι το διάνυσμα βάση που μεταλλάσσεται και y είναι ο αριθμός των αφαιρέσεων διανυσμάτων, που γίνονται για να προκύψει η μεταβολή του x. Ο δείκτης z αναφέρεται στον ανασυνδυασμό και συνήθως έχει την τιμή bin.

2.5.3.1 Τελεστής Μετάλλαξης (Mutation)

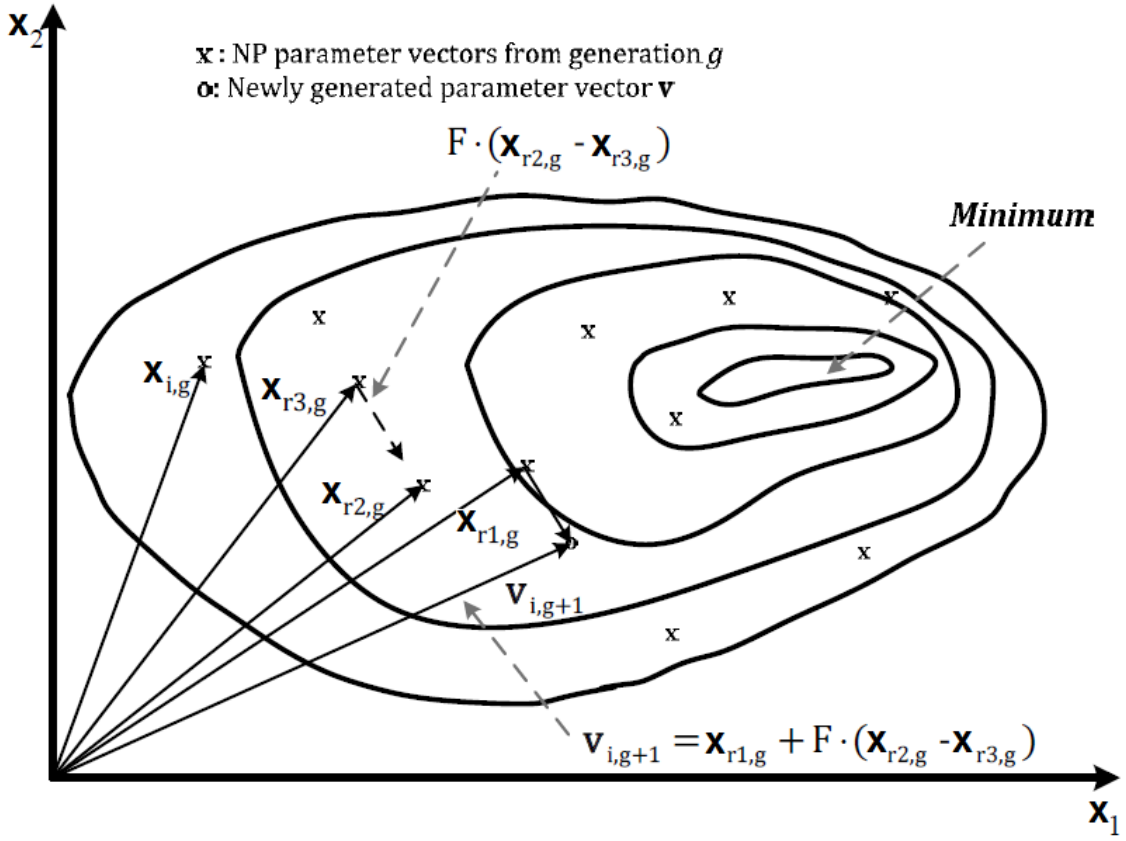
Η μετάλλαξη είναι ένας στοχαστικός τελεστής που παράγει ένα νέο διάνυσμα ελαφρά διαφορετικό από το αρχικό σε κάποιες ή όλες τις διαστάσεις του διανύσματος. Στους περισσότερους εξελικτικούς αλγόριθμους η μετάλλαξη πετυχαίνεται με μικρές διακυμάνσεις που υπακούν σε πιθανοτικές κατανομές για κάθε μεταβλητή. Στην ΔΕ όμως οι συνηθέστερες τελεστές μετάλλαξης που χρησιμοποιούνται είναι:

1) DE/rand/1/bin

$$\boldsymbol{v}_{i,g+1} = \boldsymbol{x}_{r1,g} + F * (\boldsymbol{x}_{r2,g} - \boldsymbol{x}_{r3,g}) \quad \textbf{(2.13)}$$

Όπου οι δείκτες $r1, r2, r3 \in \{1,2, \dots, NP\}$ είναι τυχαίοι ακέραιοι που πρέπει να είναι διαφορετικοί μεταξύ τους και από τον δείκτη i . Δηλαδή τα διανύσματα που συμμετέχουν στην μετάλλαξη είναι όλα διαφορετικά από το $\boldsymbol{x}_{i,g}$ (target vector) με το οποίο σχετίζεται το μεταλλαγμένο διάνυσμα $\boldsymbol{v}_{i,g+1}$. Επομένως ο πληθυσμός πρέπει να έχει τουλάχιστον τέσσερα μέλη: $NP \geq 4$. Για κάθε μέλος $\boldsymbol{x}_{i,g}$ του πληθυσμού χρησιμοποιούνται διαφορετικοί τυχαίοι

δείκτες $r1, r2, r3$. Η παράμετρος ελέγχου F καθορίζει την μεγέθυνση της διαφορικής μεταβολής $\boldsymbol{x}_{r2,g} - \boldsymbol{x}_{r3,g}$. Η εξίσωση (2.13) ήταν η αρχική μορφή του τελεστή μετάλλαξης που προτάθηκε από τους Storn & Price (1997). Στο παρακάτω σχήμα φαίνεται η γραφική αναπαράσταση της στην περίπτωση δύο μόνο μεταβλητών σχεδιασμού.



Σχήμα 2.11 Γραφική παράσταση της μετάλλαξης τύπου DE/rand/1/bin για ένα δυσδιάστατο χώρο σχεδιασμού.

2) DE/best/1/bin

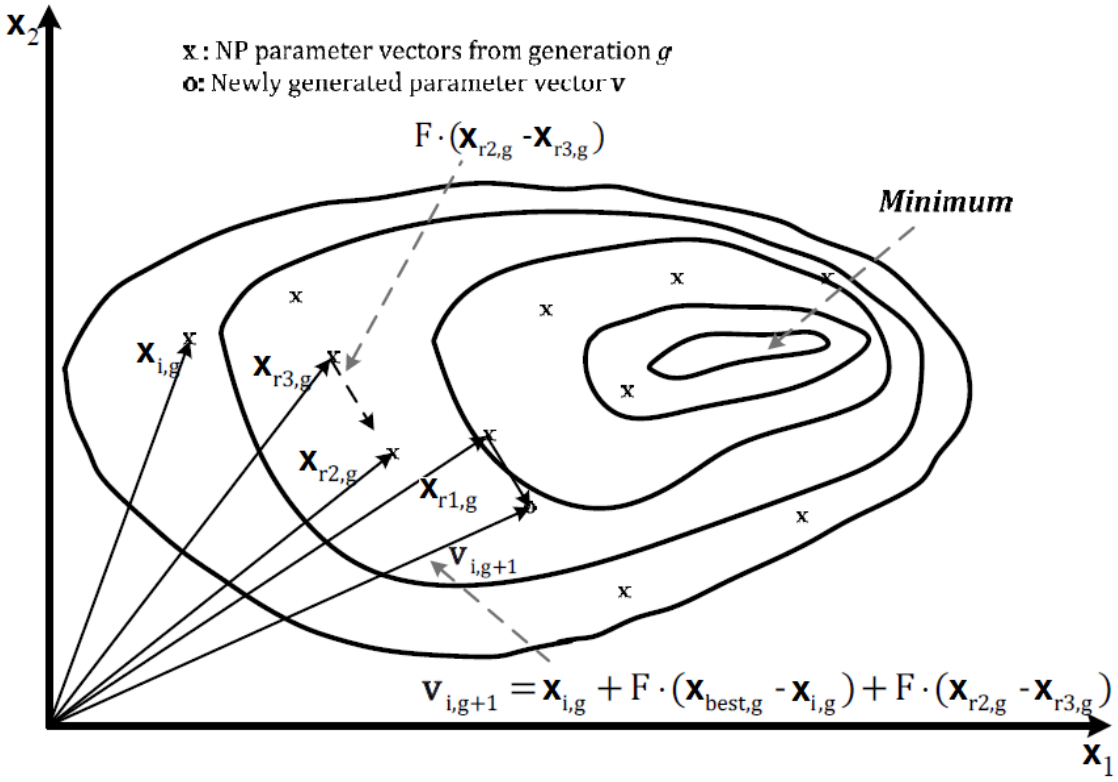
$$\boldsymbol{v}_{i,g+1} = \boldsymbol{x}_{best,g} + F * (\boldsymbol{x}_{r1,g} - \boldsymbol{x}_{r2,g}) \qquad \textbf{(2.14)}$$

Όπου το τυχαίο διάνυσμα-βάση της μετάλλαξης έχει αντικατασταθεί από τη βέλτιστη λύση της γενιάς g. Η συμμετοχή της βέλτιστης λύσης χαρακτηρίζει τον αλγόριθμο ως «άπληστο» (greedy). Η απληστία των αλγορίθμων βελτιστοποίησης έχει γενικά ως συνέπειες την ταχύτερη σύγκλιση, αλλά την ευκολότερη παγίδευση στα τοπικά ελάχιστα που βρίσκονται στις πρώτες επαναλήψεις (πρόωρη σύγκλιση).

3) DE/current-to-best/1/bin

$$\boldsymbol{v}_{i,g+1} = \boldsymbol{x}_{i,g} + F * (\boldsymbol{x}_{best,g} - \boldsymbol{x}_{i,g}) + F * (\boldsymbol{x}_{r1,g} - \boldsymbol{x}_{r2,g}) \quad \textbf{(2.15)}$$

Όπου ως βάση της μετάλλαξης χρησιμοποιείται το ίδιο το διάνυσμα $\boldsymbol{x}_{i,g}$ (target vector) που αντιστοιχεί στο μεταλλαγμένο $\boldsymbol{v}_{i,g+1}$. Επιπλέον εκτός από τη συνηθισμένη τυχαία διαφορά, χρησιμοποιείται και η πληροφορία της βέλτιστης λύσης στην μετάλλαξη. Ο τελεστής αυτός μοιάζει με τη λογική της ανταλλαγής πληροφοριών στη Βελτιστοποίηση Σμήνους Σωματιδίων. Στο παρακάτω σχήμα φαίνεται η γραφική αναπαράσταση του στην περίπτωση δύο μόνο μεταβλητών σχεδιασμού.



Σχήμα 2.12 Γραφική παράσταση της μετάλλαξης τύπου DE/current-to-best/1/bin για ένα δυσδιάστατο χώρο σχεδιασμού.

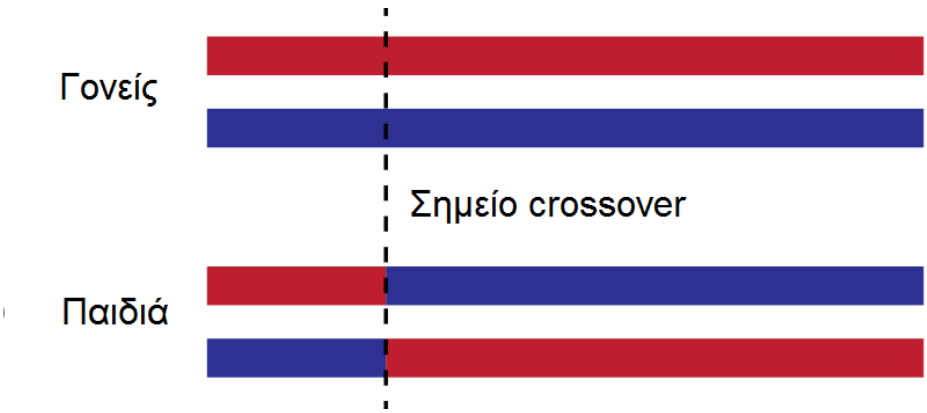
4) DE/best/2/bin

$$\boldsymbol{v}_{i,g+1} = \boldsymbol{x}_{best,g} + F * (\boldsymbol{x}_{r1,g} - \boldsymbol{x}_{r2,g} + \boldsymbol{x}_{r3,g} - \boldsymbol{x}_{r4,g}) \quad \textbf{(2.16)}$$

Όπου χρησιμοποιούνται δύο διαφορές τυχαίων διανυσμάτων. Η προσέγγιση αυτή βελτιώνει την ποικιλία του πληθυσμού αν το μέγεθός του NP είναι αρκετά μεγάλο.

2.5.3.2 Τελεστής Ανασυνδυασμού (Recombination)

Στην γενετική ανασυνδυασμός είναι η γέννηση ενός χρωμοσώματος-απογόνου από δύο χρωμοσώματα-γονείς. Αυτό επιτυγχάνεται με μία διαδικασία που λέγεται crossover (χιασμός ή επιχιασμός στα ελληνικά), κατά την οποία τα δύο χρωμοσώματα-γονείς διαχωρίζονται σε τμήματα και ο απόγονος δημιουργείται από το συνδυασμό γονιδίων ή σειρών γονιδίων των δύο γονέων. Η διαδικασία crossover αυτή φαίνεται στα παρακάτω σχήματα. Κατά τη προσομοίωση του βιολογικού μοντέλου από το μαθηματικό, τα γονίδια αντιστοιχίζονται στις μεταβλητές σχεδιασμού, τα χρωμοσώματα στα διανύσματα πιθανών λύσεων και ως σημεία για τον διαχωρισμό crossover επιλέγονται όλες οι μεταβλητές σχεδιασμού, ενώ δεν αποκλείεται να χρησιμοποιούνται παραπάνω από δύο γονείς.



Σχήμα 2.13 Crossover 1 σημείου.

Σχήμα 2.14 Crossover 2 σημείων.

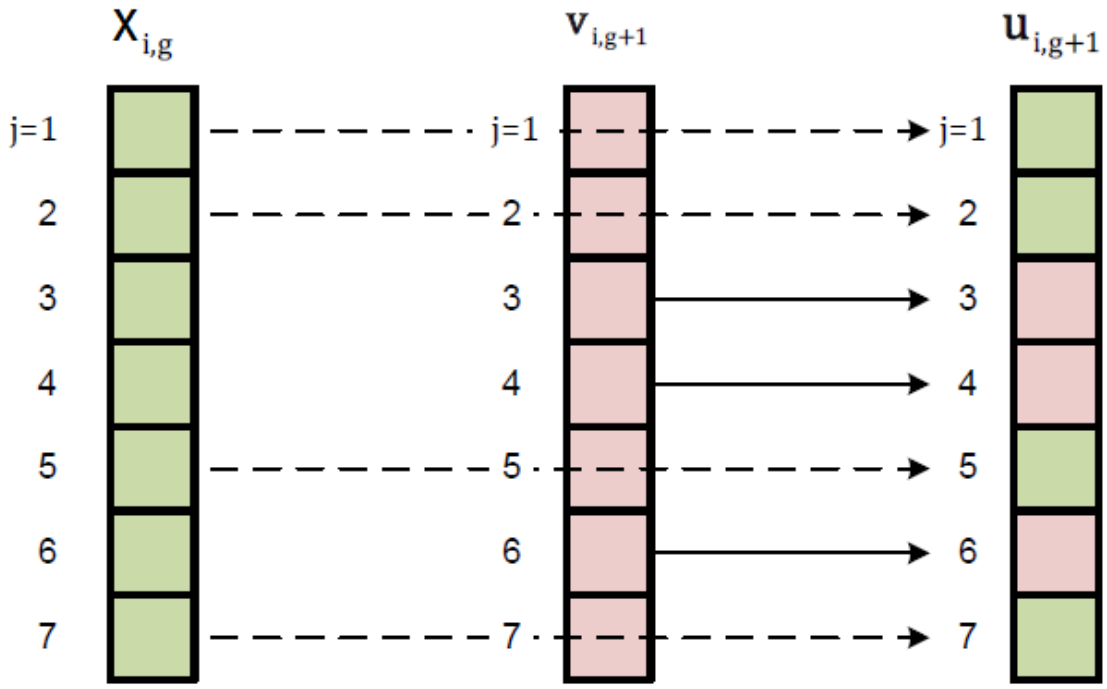
Ο τελεστής crossover που εφαρμόζεται στην Διαφορική Εξέλιξη ορίζεται ως εξής. Για κάθε ζεύγος αρχικού διανύσματος $\boldsymbol{x}_{i,g}$ (target vector) και μεταλλαγμένου διανύσματος $\boldsymbol{v}_{i,g+1}$ (mutant vector), δημιουργείται ένα νέο δοκιμαστικό διάνυσμα $\boldsymbol{u}_{i,g+1}$ (trial vector). Τα στοιχεία του $\boldsymbol{v}_{i,g+1}$ έχουν πιθανότητα CR (crossover probability) να εισέλθουν στο $\boldsymbol{u}_{i,g+1}$, σύμφωνα με την εξίσωση:

$$u_{j,i,g+1} = \begin{cases} v_{j,i,g+1} & \alpha\nu\ rand_{j,i} \leq CR\ \ \eta\ \ j = R_i \\ x_{j,i,g} & \alpha\nu\ rand_{j,i} > CR\ \ \kappa\alpha\iota\ \ j \neq R_i \end{cases} \quad \textbf{(2.17)}$$

Όπου:

- $i = 1, 2, \dots, NP$: είναι ο δείκτης των μελών του πληθυσμού.
- $j = 1, 2, \dots, D$: είναι ο δείκτης των μεταβλητών πλήθους D.
- $rand_{j,i}$: είναι ένας τυχαίος αριθμός που υπακούει στην ομοιόμορφη κατανομή $U[0,1]$. Για κάθε μεταβλητή κάθε διανύσματος χρησιμοποιείται ένας διαφορετικός τυχαίος αριθμός.
- R_i: είναι ένας τυχαίος ακέραιος αριθμός *1, 2, … ή D*. Παίρνει δηλαδή την τιμή του δείκτη μίας μεταβλητής. Για κάθε διάνυσμα χρησιμοποιείται διαφορετικός τυχαίος αριθμός. Ρόλος του είναι να εξασφαλίζει ότι το δοκιμαστικό διάνυσμα $\boldsymbol{u}_{i,g+1}$ θα διαφέρει από το αρχικό $\boldsymbol{x}_{i,g}$ τουλάχιστο σε μία μεταβλητή σχεδιασμού.

Ο τελεστής crossover αυξάνει την ποικιλία των λύσεων αλλά μπορεί να καταστρέψει αρκετές πληροφορίες κατεύθυνσης που παρέχονται από τη φάση της μετάλλαξης. Ωστόσο έχει φανεί ότι σε πρακτικές εφαρμογές η εφαρμογή του βελτιώνει τελικά την διαδικασία βελτιστοποίησης. Η παραπάνω μορφή του τελεστή είναι η πρωτότυπη που προτάθηκε από τους Storn & Price. Έχουν δημιουργηθεί και άλλες παραλλαγές, αλλά αυτή είναι η πλέον διαδεδομένη. Στο επόμενο σχήμα φαίνεται μια γραφική απεικόνισή της.



Σχήμα 2.15 Crossover στην Διαφορική Εξέλιξη με 7 μεταβλητές σχεδιασμού

2.5.3.3 Τελεστής Επιλογής (Selection)

Η επιλογή στην βιολογία αφορά την επιβίωση των καταλληλότερων χρωμοσωμάτων μιας γενιάς. Στην βελτιστοποίηση καταλληλότερες είναι οι λύσεις που παρουσιάζουν τη βέλτιστη τιμή αντικειμενικής συνάρτησης ή συνάρτησης ποιότητας (fitness), αν υπάρχουν περιορισμοί. Στην αρχική μέθοδο Διαφορικής Εξέλιξης ο τελεστής επιλογής είναι πολύ απλός. Συγκρίνονται τα διανύσματα $\boldsymbol{x}_{i,g}$ (target vector) και $\boldsymbol{u}_{i,g+1}$ (trial vector), ως εξής:

$$\boldsymbol{x}_{i,g+1} = \begin{cases} \boldsymbol{u}_{i,g+1} & \alpha\nu\ f(\boldsymbol{u}_{i,g+1}) \leq f(\boldsymbol{x}_{i,g}) \\ \boldsymbol{x}_{i,g} & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\acute{\alpha} \end{cases} \quad \textbf{(2.18)}$$

Όπου $i = 1, 2, \dots, NP$.

Άλλες μέθοδοι που έχουν προταθεί είναι:

- Η ελιτιστική *(μ+λ) επιλογή* που χρησιμοποιείται και στις εξελικτικές στρατηγικές. Σύμφωνα με αυτήν, αν μ είναι το μέγεθος του αρχικού πληθυσμού (διανύσματα $\boldsymbol{x}_{i,g}$) και λ το μέγεθος του ενδιάμεσου πληθυσμού (διανύσματα $\boldsymbol{u}_{i,g+1}$) επιβιώνουν οι μ καλύτερες λύσεις σε κάθε γενιά. Στην ΔΕ συνήθως $\mu = \lambda = NP$.
- Η επιλογή «τουρνουά» (tournament selection) με έναν προς έναν επιζώντες όπως στην κλασσική ΔΕ.

2.5.4 Αρχική επιλογή και κριτήρια τερματισμού

Ο αρχικός πληθυσμός που προτείνουν οι Storn & Price (1997), προκύπτει από ομοιόμορφα τυχαία διασπορά των διανυσμάτων-μελών στο χώρο σχεδιασμού σύμφωνα με την εξίσωση:

$$x_{j,i,0} = l_j + rand_{j,i} * (u_j - l_j) \quad \textbf{(2.19)}$$

Όπου

- $i = 1, 2, \dots, NP$: είναι ο δείκτης των μελών του πληθυσμού.
- $j = 1, 2, \dots, D$: είναι ο δείκτης των μεταβλητών πλήθους D.
- l_j, u_j: είναι το κάτω και άνω όριο της μεταβλητής x_j, αντίστοιχα.
- $rand_{j,i}$: είναι ένας τυχαίος αριθμός που υπακούει στην ομοιόμορφη κατανομή $U[0,1]$. Για κάθε μεταβλητή κάθε διανύσματος χρησιμοποιείται ένας διαφορετικός τυχαίος αριθμός.

Οι Storn & Price δεν αναφέρουν ρητά κάτι για τις συνθήκες σύγκλισης της μεθόδου. Στις αριθμητικές εφαρμογές της παρούσας εργασίας, ως κριτήρια σύγκλισης θα χρησιμοποιηθούν τα ίδια με τη Βελτιστοποίηση Σμήνους Σωματιδίων, δηλαδή α) ο μέγιστος αριθμός επαναλήψεων και β) ο ελάχιστος ρυθμός βελτίωσης της λύσης **(2.12)**.

2.5.5 Παράμετροι της ΔΕ

Η Διαφορική Εξέλιξη χρησιμοποιεί μόνο τρεις παραμέτρους. Μετά από έρευνες των Storn & Price, αλλά και άλλων έχουν εντοπιστεί κάποιες «καλές» αρχικές επιλογές, αν βέβαια δεν είναι γνωστή η συμπεριφορά της μεθόδου για το εκάστοτε πρόβλημα.:

- NP: Το πλήθος των δυνατών λύσεων-διανυσμάτων που συνυπάρχουν σε κάθε γενιά. Για τις ανάγκες του τελεστή μετάλλαξης πρέπει $NP \geq 4$ ή 5 ανάλογα με τον τύπο μετάλλαξης. Γενικά προτείνονται τιμές από $5 * D$ ως $10 * D$, όπου *D* είναι το πλήθος των μεταβλητών σχεδιασμού του προβλήματος.

- F: Παράμετρος ελέγχου της μεθόδου που ονομάζεται διαφορικό βάρος. Επιλέγεται ώστε $F \in [0, 2]$. Έρευνες έχουν δείξει, ότι στις περισσότερες περιπτώσεις η τιμή της F είναι προτιμότερο να περιορίζεται στο διάστημα $[0.5, 1.0]$. Οι εφευρέτες της μεθόδου προτείνουν αρχικά την τιμή 0.5 και αύξησή της αν η μέθοδος συγκλίνει πρόωρα. Η D. Zaharie απέδειξε ότι τιμές μικρότερες από μια κρίσιμη τιμή F_{crit} είναι ακατάλληλες, όπου

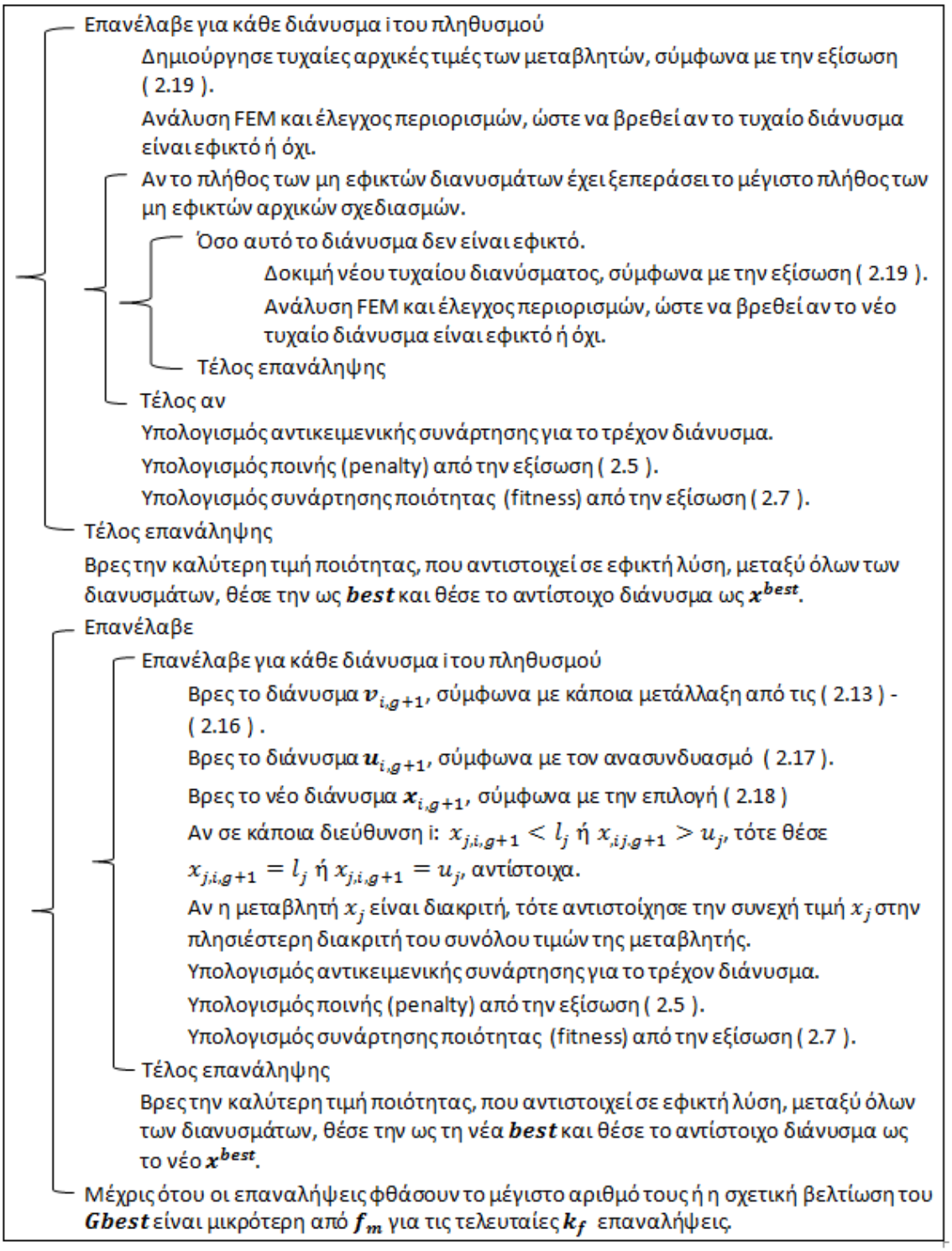
$$F_{crit} = \sqrt{\frac{1 - \frac{CR}{2}}{NP}} \quad \textbf{(2.20)}$$

  Άλλες έρευνες εστιάζουν στην χρήση διαφορικών βαρών *F* που δεν είναι σταθερά, αλλά μεταβάλλονται τυχαία σε κάθε επανάληψη. Έχει επίσης προταθεί η προσαρμογή των παραμέτρων *F* και *CR* ταυτόχρονα με τη βελτιστοποίηση, εντάσσοντάς τες στις μεταβλητές σχεδιασμού.

- CR: Παράμετρος ελέγχου της μεθόδου που ονομάζεται πιθανότητα crossover. Λαμβάνει τιμές στο διάστημα $CR \in [0,1]$. Μεγάλες τιμές της παραμέτρου αυξάνουν την πιθανότητα επιβίωσης της καινούργιας τιμής που αποκτά κάθε μεταβλητή μετά την μετάλλαξη. Έτσι η σύγκλιση συχνά επιταχύνεται. Γενικά προτείνονται μεγάλες τιμές της παραμέτρου: $CR \in [0.9, 1.0]$. Σε κάποια προβλήματα όμως χαμηλές τιμές (~ 0.1) μπορεί να συμπεριφέρονται καλύτερα.

2.5.6 Ο Αλγόριθμος ΔΕ για το Βέλτιστο Σχεδιασμό Κατασκευών

Στις αριθμητικές εφαρμογές της παρούσας εργασίας θα χρησιμοποιηθεί η συνάρτηση ποινής που παρουσιάστηκε στην ενότητα **2.2.3** για το χειρισμό των περιορισμών. Αν κάποιες μεταβλητές είναι διακριτές, χρησιμοποιείται η τεχνική που περιγράφηκε στην ενότητα **2.3.3**, δηλαδή χρησιμοποιούνται συνεχείς τιμές που αντιστοιχίζονται στις πλησιέστερες διακριτές πριν την αξιολόγηση κάθε πιθανής λύσης. Έτσι ο τελικός αλγόριθμος γίνεται:



Σχήμα 2.16 Ψευδό-κώδικας για την ΔΕ, όπως χρησιμοποιείται στις αριθμητικές εφαρμογές.

2.6 Μέθοδος Στρατηγικών Εξέλιξης (ΣΕ)

2.6.1 Εισαγωγή

Οι Στρατηγικές Εξέλιξης (Evolutionary Strategies – ES) είναι μία στοχαστική μέθοδος βελτιστοποίησης που μιμείται την βιολογική εξέλιξη των οργανισμών. Αναπτύχθηκαν αρχικά από τον I. Rechenberg (1973) και στη συνέχεια από τον H.P. Schwefel (1981). Οι Εξελικτικές Στρατηγικές και οι Γενετικοί Αλγόριθμοι αποτελούν τους δύο δημοφιλέστερους Εξελικτικούς Αλγορίθμους, μια σημαντική υποκατηγορία των μεταευριστικών μεθόδων. Αρχικά εφαρμόστηκαν σε προβλήματα βελτιστοποίησης με συνεχείς μεταβλητές σχεδιασμού, αλλά σύντομα έγιναν προσπάθειες επέκτασης της μεθόδου για διακριτά και δυαδικά προβλήματα (Schwefel 1975).

Η βασική ιδέα πίσω από τις Στρατηγικές Εξέλιξης (και τους Εξελικτικούς Αλγόριθμους γενικότερα) είναι:

- Αναπαράσταση των μεταβλητών σχεδιασμού ως γονίδια και των διανυσμάτων που τις περιέχουν ως χρωμοσώματα.
- Χρήση ενός πληθυσμού τέτοιων χρωμοσωμάτων που ανανεώνεται σε κάθε γενιά, δηλαδή σε κάθε επανάληψη της μεθόδου.
- Η ανανέωση του πληθυσμού κάθε γενιάς γίνεται με τους στοχαστικούς γενετικούς τελεστές του ανασυνδυασμού, της μετάλλαξης και της επιλογής.
- Ο πληθυσμός γονέων κάθε γενιάς παράγει απογόνους εφαρμόζοντας ανασυνδυασμό (recombination) σε ζεύγη των υπαρχόντων μελών.
- Τα χρωμοσώματα των απογόνων τροποποιούνται ελαφρά και με τυχαίο τρόπο μέσω μετάλλαξης (mutation).
- Η καταλληλότητα κάθε χρωμοσώματος αξιολογείται με κριτήριο την αντικειμενική συνάρτηση και τους περιορισμούς του προβλήματος.
- Ο πληθυσμός της νέας γενιάς δημιουργείται μέσω επιλογής (selection), δηλαδή επιβιώνουν μόνο τα καταλληλότερα χρωμοσώματα από τους απογόνους ή και από τους γονείς.
- Η διαδικασία επαναλαμβάνεται μέχρι να επιτευχθεί ικανοποιητική σύγκλιση.

Οι Στρατηγικές Εξέλιξης μπορούν να αντιμετωπίσουν προβλήματα με μη κυρτές αντικειμενικές συναρτήσεις και πολύπλοκους περιορισμούς. Έχουν προσαρμοστεί για την επίλυση συνεχών, διακριτών ή μεικτών προβλήματα βελτιστοποίησης (J. Cai & G. Thierauf 1995-1996). Επομένως μπορούν και έχουν εφαρμοστεί σε προβλήματα βέλτιστου σχεδιασμού κατασκευών. Σε αρκετά προβλήματα φάνηκε να υπερτερούν μεθόδων Μαθηματικού Προγραμματισμού, που αναλώνουν πολύ χρόνο σε αναλύσεις ευαισθησίας. Σε σχέση με τους Γενετικούς Αλγόριθμους, ο εσωτερικός μηχανισμός προσαρμογής των ΣΕ θεωρείται πιο αποδοτικός ως προς την ταχύτητα σύγκλισης (Hoffmeister & Back, 1991) σε πρακτικές εφαρμογές και γενικά τείνει να καταλήγει στη βέλτιστη λύση συχνότερα, ειδικά όταν οι αρχικές τιμές των χρωμοσωμάτων δεν

είναι καλές. Επιπλέον μπορούν να εφαρμοστούν σε παράλληλο προγραμματιστικό περιβάλλον, για την μείωση του απαιτούμενου υπολογιστικού χρόνου.

Η αρχική μορφή του αλγορίθμου περιελάμβανε μόνο δύο μέλη, έναν γονέα και έναν απόγονο, σε κάθε επανάληψη και σημαίνεται ως (1+1)-ES. Στη συνέχεια ο Rechenberg εισήγαγε περισσότερους γονείς σε κάθε επανάληψη και πρόσθεσε τον ανασυνδυασμό, με αποτέλεσμα να αυξάνεται η ποικιλία των χρωμοσωμάτων και να προσαρμόζεται αυτόματα η ένταση της μετάλλαξης σε κάθε γενιά. Η παραλλαγή αυτή είναι γνωστή ως (μ+1)-ES. Οι Στρατηγικές Εξέλιξης πολλών μελών που χρησιμοποιούνται στην παρούσα εργασία, (μ+λ)-ES και (μ,λ)-ES, προτάθηκαν αργότερα από τον Schwefel και επιτρέπουν α) την ταυτόχρονη επεξεργασία των λ απογόνων σε παράλληλους επεξεργαστές και β) ακόμα μεγαλύτερη ποικιλία, άρα δυσκολότερη παγίδευση σε τοπικά ελάχιστα.

2.6.2 ΣΕ πολλών μελών για συνεχείς μεταβλητές σχεδιασμού

Έστω ότι στη γενιά g χρησιμοποιείται ένας πληθυσμός P_g^p χρωμοσωμάτων (πιθανών λύσεων) μεγέθους μ. Ο πληθυσμός αυτός θα αναφέρεται ως πληθυσμός γονέων (parents). Από τον πληθυσμό των γονέων παράγεται ένας ενδιάμεσος πληθυσμός απογόνων (offsprings) P_g^o μεγέθους λ, εφαρμόζοντας διαδοχικά τους τελεστές του ανασυνδυασμού και της μετάλλαξης. Έπειτα δημιουργείται ο πληθυσμός γονέων της επόμενης γενιάς P_{g+1}^p μεγέθους μ με τον τελεστή επιλογής και η διαδικασία επαναλαμβάνεται μέχρι να επιτευχθεί σύγκλιση.

Εκτός από τους παραπάνω συμβολισμούς, παρακάτω θα χρησιμοποιηθούν και οι επόμενοι:

- n : ο αριθμός των γονιδίων, δηλαδή μεταβλητών σχεδιασμού.
- $i = 1, 2, \dots, n$: ο δείκτης ενός γονιδίου.
- $l = 1, 2, \dots, \lambda$: ο δείκτης ενός χρωμοσώματος απογόνου.
- $m = 1, 2, \dots, \mu$: ο δείκτης ενός χρωμοσώματος γονέα.
- $x_{l,i}$ ή $x_{m,i}$: το γονίδιο i του χρωμοσώματος $\boldsymbol{x}_l$ ή $\boldsymbol{x}_m$, αντίστοιχα.

2.6.2.1 Ανασυνδυασμός (recombination)

Ο τελεστής ανασυνδυασμού παντρεύει δύο ή περισσότερα χρωμοσώματα-γονείς και γεννά ένα νέο που αποτελείται από γονίδια των γονέων. Αυτό γίνεται με μία διαδικασία που ονομάζεται crossover, κατά την οποία κάθε μεταβλητή-γονίδιο του χρωμοσώματος-απογόνου παίρνει τυχαία την τιμή του αντίστοιχου γονιδίου ενός από τους γονείς. Αυτό γίνεται για κάθε γονίδιο l ξεχωριστά. Η διαδικασία crossover αναλύθηκε περαιτέρω στην ενότητα 2.5.3.2 . Ο ανασυνδυασμός συμβάλλει στην αξιοπιστία της μεθόδου και στην αποφυγή τοπικών ελαχίστων χωρίς όμως να αυξάνει το υπολογιστικό κόστος, σε σχέση με την ΣΕ δύο μελών, όπου ο τελεστής αυτός απουσιάζει.

Για τον ανασυνδυασμό σε προβλήματα συνεχών τιμών μπορεί να χρησιμοποιηθεί μία από τις επόμενες εξισώσεις:

$$x_{l,i}^{temp} = \begin{cases} x_{a,i} \text{ ή } x_{b,i} \text{ στην τύχη} & (A) \\ 1/2 * (x_{a,i} + x_{b,i}) & (B) \\ x_{rand,i} & (C) \\ x_{a,i} \text{ ή } x_{rand,i} \text{ στην τύχη} & (D) \\ 1/2 * (x_{a,i} + x_{rand,i}) & (E) \end{cases} \quad \textbf{(2.21)}$$

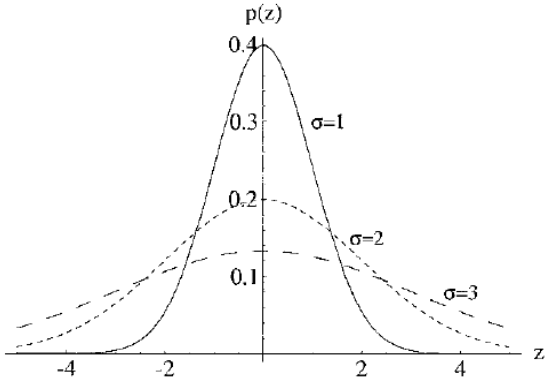
όπου $\boldsymbol{x}_{\alpha}$ και $\boldsymbol{x}_{b}$ είναι δύο διανύσματα που επιλέγονται τυχαία από των πληθυσμό των γονέων. Για κάθε προσωρινό απόγονο $\boldsymbol{x}_{l}^{temp}$ επιλέγονται δύο διαφορετικά $\boldsymbol{x}_{\alpha}$ και $\boldsymbol{x}_{b}$ αλλά διατηρούνται τα ίδια για όλα τα γονίδια αυτού του απογόνου. Αντίθετα το γονίδιο $x_{rand,i}$ επιλέγεται τυχαία από όλα τα γονίδια i των χρωμοσωμάτων του πληθυσμού γονέων και είναι διαφορετικό για κάθε γονίδιο $x_{l,i}^{temp}$ του ίδιου απογόνου. Στις περιπτώσεις (A) και (E) τα δύο γονίδια έχουν πιθανότητες επιλογής 50%-50%.

2.6.2.2 Μετάλλαξη (mutation)

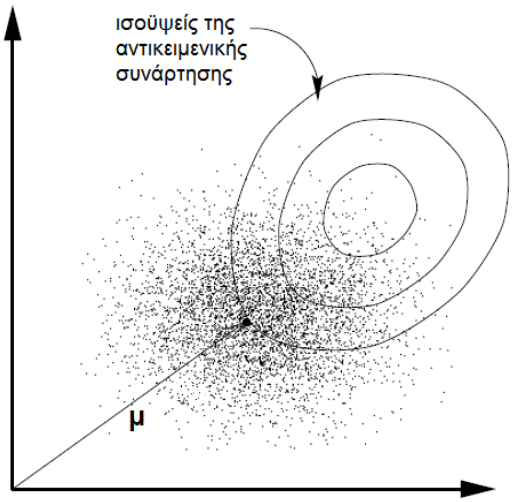
Το χρωμόσωμα κάθε απογόνου που προέκυψε από τον παραπάνω ανασυνδυασμό υφίσταται μετάλλαξη στη συνέχεια. Ο στοχαστικός τελεστής της μετάλλαξης παραμορφώνει ελαφρά και τυχαία τα γονίδια. Για την επιλογή της κατάλληλης διαδικασίας, πρέπει να ληφθούν υπόψη κάποιες απαιτήσεις. Η μετάλλαξη πρέπει να είναι τέτοια ώστε α) να μπορεί να δημιουργηθεί κάθε πιθανή λύση του χώρου σχεδιασμού μετά από έναν αριθμό γενιών, β) να είναι αμερόληπτη ως προς την καταλληλότητα (fitness) των χρωμοσωμάτων που παράγονται (άλλωστε για αυτό είναι υπεύθυνος ο τελεστής επιλογής), γ) να μπορεί να προσαρμοσθεί η ένταση της μετάλλαξης στις συνθήκες (ελάχιστα, μη εφικτές περιοχές) του χώρου σχεδιασμού. Επιπλέον, επιθυμούμε μικρές διαφοροποιήσεις με μεγάλη συχνότητα και μεγάλες διαφοροποιήσεις με μικρή συχνότητα, αλλά πάντα τυχαίες.

Έχει παρατηρηθεί ότι η επιβολή διακυμάνσεων που ακολουθούν την κανονική κατανομή, ικανοποιεί τις πρώτες δύο απαιτήσεις. Η κανονική κατανομή ή κατανομή Gauss καθορίζεται από την παράμετρο μ, που εκφράζει την αναμενόμενη μέση τιμή ενός τυχαίου δείγματος, και από την παράμετρο σ, που ονομάζεται τυπική απόκλιση και εκφράζει τη ρίζα της μέσης απόκλισης από τη μέση τιμή. Κάποιες γραφικές απεικονίσεις της φαίνονται στο **Σχήμα 2.17**. Λέμε ότι μια τυχαία μεταβλητή ζ ακολουθεί την κατανομή $N(\mu, \sigma^2)$, όταν έχει συνάρτηση πυκνότητας πιθανότητας:

$$p(\zeta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\zeta - \mu)^2}{2\sigma^2} \right) \quad \textbf{(2.22)}$$



α) Κανονικές κατανομές μίας μεταβλητής N(0,1), N(0,2^2), ...



β) Διασπορά 2D σημείων, που κάθε μεταβλητή υπακούει σε κανονική κατανομή

Σχήμα 2.17 Η κανονική κατανομή σε μία και δύο διαστάσεις.

Μία χρήσιμη ιδιότητα της κανονικής κατανομής είναι η ακόλουθη:

Αν η τυχαία μεταβλητή X ακολουθεί κανονική κατανομή με μέση τιμή μ και τυπική απόκλιση σ, τότε η τυχαία μεταβλητή $(X-\mu)/\sigma$ ακολουθεί την τυποποιημένη κανονική κατανομή $N(0,1)$ **(2.23)**

Έτσι ένας αριθμός που ισούται με την τυχαία τιμή $N(0,\sigma^2)$, μπορεί να γραφτεί ως $\sigma * N(0,1)$. Στη συνέχεια θα χρησιμοποιείται η δεύτερη μορφή, ώστε να απομονώνεται η τυπική απόκλιση που συνήθως είναι παράμετρος ελέγχου της μεθόδου, όπως θα εξηγηθεί στην επόμενη ενότητα.

Η μετάλλαξη επιτυγχάνεται προσθέτοντας στο χρωμόσωμα του απογόνου ένα διάνυσμα τυχηματικής αλλαγής $\boldsymbol{z}$:

$$\boldsymbol{x}_l = \boldsymbol{x}_l^{temp} + \boldsymbol{z}_l \qquad \textbf{(2.24)}$$

Η επιλογή του διανύσματος $\boldsymbol{z}$ έχει αποτελέσει αντικείμενο πολλών ερευνών, αφού ουσιαστικά καθορίζει τις ιδιότητες της μετάλλαξης. Όπως ήδη αναφέρθηκε, τα στοιχεία του ακολουθούν την κανονική κατανομή. Ο μέσος όρος επιλέγεται να είναι $\mu = 0$, γιατί διαφορετικά η μετάλλαξη θα μεροληπτούσε σε όφελος μεγαλύτερων ($\mu > 0$) ή μικρότερων ($\mu < 0$) τιμών, συγκριτικά με το τρέχον γονίδιο πάντα. Για την τυπική απόκλιση υπάρχουν οι εξής εναλλακτικές:

A. Ισότροπη μετάλλαξη:

$$\boldsymbol{z} = \sigma * [N_1(0,1),\ N_2(0,1),\ \ldots,\ N_n(0,1)]^T \quad \textbf{(2.25)}$$

όπου χρησιμοποιείται η ίδια τυπική απόκλιση για κάθε μεταβλητή. Φυσικά ένας διαφορετικός τυχαίος αριθμός $N_i(0,1)$ επιλέγεται για κάθε γονίδιο. Η μετάλλαξη αυτή είναι ισότροπη, δηλαδή παρουσιάζεται ο ίδιος βαθμός διασποράς προς όλες τις διευθύνσεις. Αυτός ο τύπος μετάλλαξης εισάγει μία μόνο παράμετρο ελέγχου της μεθόδου ΣΕ, την σ.

B. Ανισότροπη μετάλλαξη:

$$\boldsymbol{z} = [\sigma_1 * N_1(0,1),\ \sigma_2 * N_2(0,1),\ \ldots,\ \sigma_n * N_n(0,1)]^T \quad \textbf{(2.26)}$$

όπου χρησιμοποιείται διαφορετική τυπική απόκλιση για κάθε μεταβλητή. Έτσι η καμπύλη της κανονικής κατανομής μπορεί να είναι πιο «στενή» σε κάποιες διευθύνσεις. Στην περίπτωση αυτή οι παράμετροι ελέγχου της μεθόδου είναι το διάνυσμα

$\boldsymbol{\sigma} = [\sigma_1,\ \sigma_2,\ \ldots,\ \sigma_n]$.

C. Ανισότροπη μετάλλαξη με περιστροφή:

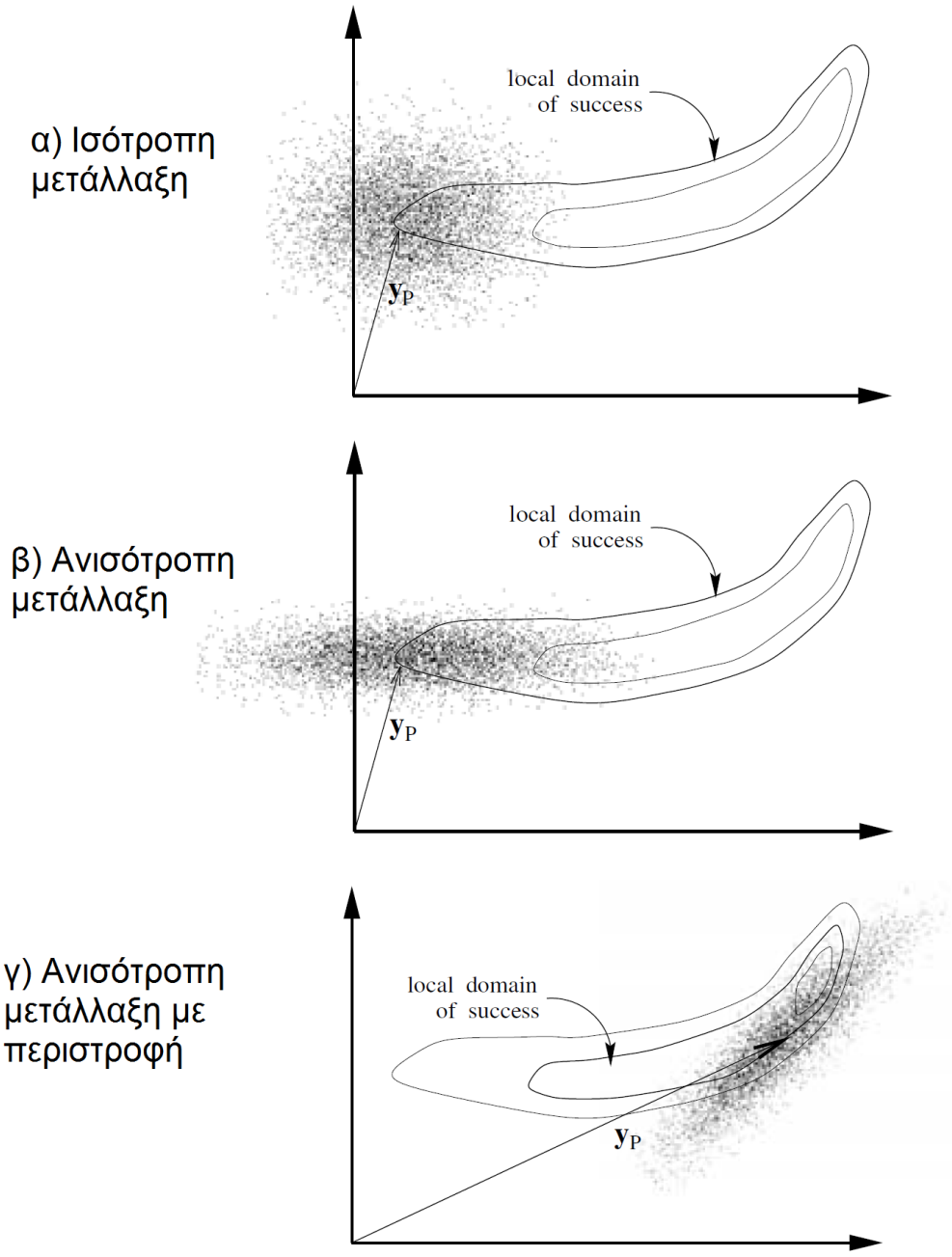
$$\boldsymbol{z} = \boldsymbol{M} * [\sigma_1 * N_1(0,1),\ \sigma_2 * N_2(0,1),\ \ldots,\ \sigma_n * N_n(0,1)]^T \quad \textbf{(2.27)}$$

όπου $\boldsymbol{M}$ είναι ένας ορθογώνιος πίνακας περιστροφής $\boldsymbol{M}^T\boldsymbol{M} = \boldsymbol{C}$. Στη γενική περίπτωση που ο $\boldsymbol{M}$ δεν είναι διαγώνιος, εισάγονται συσχετίσεις μεταξύ των στοιχείων του διανύσματος $\boldsymbol{z}$ και η συνάρτηση πυκνότητας πιθανότητας γίνεται:

$$p(\boldsymbol{z}) = \frac{1}{\left(\sqrt{2\pi}\right)^n}\frac{1}{\det|\boldsymbol{C}|}exp\left(-\frac{1}{2}\boldsymbol{z}^T\boldsymbol{C}\boldsymbol{z}\right) \quad \textbf{(2.28)}$$

Η χρήση πίνακα περιστροφής εισάγει άλλες $n(n-1)/2$ παραμέτρους ελέγχου της μεθόδου, οπότε μαζί με τις τυπικές αποκλίσεις απαιτείται η ρύθμιση $n(n+1)/2$ παραμέτρων από τον χρήστη της μεθόδου.

Η επιλογή μίας από τις τρεις αυτές μεταλλάξεις εξαρτάται από το εκάστοτε πρόβλημα. Στο παρακάτω σχήμα φαίνονται οι διασπορές που προκύπτουν από καθεμία μετάλλαξη, σε ένα δυσδιάστατο πρόβλημα. Στο συγκεκριμένο παράδειγμα προτιμότερη θα ήταν η μετάλλαξη C, λόγω του προσανατολισμού των ισοϋψών, ανεξάρτητα από το ότι στο σχήμα η μέση τιμή της C είναι μετατοπισμένη προς την βέλτιστη περιοχή.

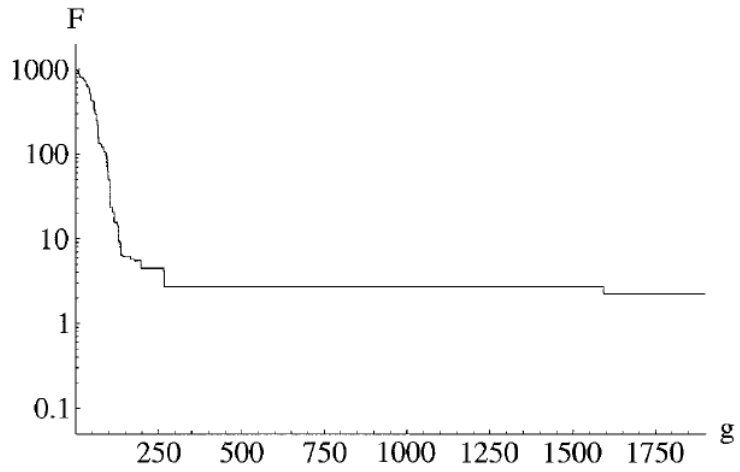


Σχήμα 2.18 Οι τρεις τύποι μετάλλαξης για ένα 2D πρόβλημα.

2.6.2.2.1 Προσαρμογή του βήματος μετάλλαξης

Όπως και σε πολλούς άλλους αλγόριθμους βελτιστοποίησης, τα μέλη του πληθυσμού στις Στρατηγικές Εξέλιξης συγκλίνουν προς κάποιο ελάχιστο διανύοντας κάποια απόσταση ή βήμα σε κάθε επανάληψη. Το βήμα αυτό μπορεί να είναι σταθερό, αλλά τότε η μέθοδος παρουσιάζει αργή σύγκλιση στις τελευταίες επαναλήψεις. Πράγματι ενώ στις πρώτες επαναλήψεις ένα μεγάλο βήμα είναι γενικά προτιμότερο για να εξερευνηθούν πολλές περιοχές του χώρου

σχεδιασμού, μόλις βρεθεί η περιοχή του ελαχίστου είναι καλύτερα μικρά βήματα ώστε η ακριβής προσέγγισή του να είναι ταχύτερη. Διαφορετικά οι δοκιμαστικές λύσεις θα το προσπερνούν. Βέβαια αν το συγκεκριμένο ελάχιστο δεν είναι τοπικό, τότε ένα μεγάλο βήμα έχει περισσότερες πιθανότητες να απεγκλωβίσει τον αλγόριθμο.



Σχήμα 2.19 Μείωση της ταχύτητας σύγκλισης για σταθερή τυπική απόκλιση. Ο πληθυσμός χάνει την ικανότητά του να προσαρμόζεται στις ιδιαιτερότητες του προβλήματος.

Στις Στρατηγικές Εξέλιξης τον ρόλο του βήματος παίζει η τυπική απόκλιση σ που εκφράζει την ένταση της μετάλλαξης. Ο Rechenberg παρατήρησε την επιβράδυνση της σύγκλισης όταν χρησιμοποιούσε σταθερή τυπική απόκλιση (**Σχήμα 2.19**). Για να την αντιμετωπίσει χρησιμοποίησε μία προσαρμογή της παραμέτρου σ η οποία έχει ντετερμινιστική φύση. Αργότερα ο Schwefel ενσωμάτωσε την τυπική απόκλιση στις μεταβλητές σχεδιασμού, ώστε να προσαρμόζεται ταυτόχρονα με την βελτιστοποίησή τους. Βέβαια υπάρχουν και άλλες προσεγγίσεις για την απαιτούμενη προσαρμογή. Οι δύο προαναφερθείσες αναλύονται παρακάτω.

- ***Ο κανόνας του 1/5***

Προτάθηκε από τον Rechenberg για τον αλγόριθμο (1+1)-ES. Ορίζεται ως συχνότητα επιτυχίας P_s, ο λόγος των επιτυχών μεταλλάξεων ως προς τις συνολικές. Επιτυχείς μεταλλάξεις θεωρούνται αυτές στις οποίες ο μεταλλαγμένος απόγονος αντικαθιστά κάποιο χρωμόσωμα του πληθυσμού γονέων κατά τη φάση επιλογής. Στην (1+1)-ES υπάρχει μόνο ένας γονέας και ένας απόγονος βέβαια, ενώ ο συνολικός αριθμός μεταλλάξεων ταυτίζεται με τον αριθμό της τρέχουσας γενιάς. Ο κανόνας του 1/5 λέει ότι: «Η συχνότητα των επιτυχών προς όλες τις μεταλλάξεις πρέπει να είναι 1/5. Αν είναι μεγαλύτερη ή μικρότερη του 1/5, τότε η διασπορά σ^2 πρέπει να μειωθεί ή να αυξηθεί, αντίστοιχα» ή με μαθηματικούς όρους:

$$\sigma^{g+n} = \begin{cases} \frac{\sigma^g}{C}, & \alpha\nu\ P_s > 1/5 \\ \sigma^g, & \alpha\nu\ P_s = 1/5 \\ \sigma^g * C, & \alpha\nu\ P_s < 1/5 \end{cases} \quad \textbf{(2.29)}$$

όπου η σταθερά $C \in [0.85, 1)$ και η προσαρμογή επαναλαμβάνεται κάθε n γενιές. Υπενθυμίζεται ότι n είναι το πλήθος των μεταβλητών σχεδιασμού.

Ο κανόνας του 1/5 αυξάνει την ταχύτητα σύγκλισης, εις βάρος όμως της ευρωστίας της μεθόδου. Υπάρχει πιθανότητα να οδηγήσει σε πρόωρη σύγκλιση αν υπάρχουν ενεργοί περιορισμοί ή ασυνέχειες. Επιπλέον, η συχνότητα 1/5 είναι η πλέον αποδοτική μόνο για την περίπτωση της (1+1)-ES. Για διαφορετικούς πληθυσμούς γονέων και απογόνων, η βέλτιστη συχνότητα επιτυχίας είναι διαφορετική. Τέλος, η προσαρμογή αυτή έχει νόημα αν χρησιμοποιείται μόνο μία παράμετρος σ για όλα τα γονίδια.

- **Αυτό-προσαρμογή**

Ο Schwefel (1975) προτείνει την ένταξη της παραμέτρου σ στις μεταβλητές σχεδιασμού. Έτσι υφίσταται και αυτή ανασυνδυασμό και μετάλλαξη (αλλά όχι επιλογή) σε κάθε γενιά και για κάθε χρωμόσωμα ξεχωριστά. Η φιλοσοφία της τεχνικής είναι να επιτρέψει στα ίδια τα χρωμοσώματα να προσαρμόζουν εξελικτικά την ένταση της μετάλλαξης. Έχει το πλεονέκτημα ότι μπορεί να εφαρμοστεί και στην περίπτωση που χρησιμοποιείται διαφορετική τυπική απόκλιση για κάθε γονίδιο κάθε μέλους του πληθυσμού. Τίθεται όμως το ερώτημα πως θα οριστούν ο ανασυνδυασμός και η μετάλλαξη της παραμέτρου σ ή των παραμέτρων σ_i.

Όσον αφορά τον ανασυνδυασμό, στην παρούσα εργασία θα χρησιμοποιηθεί μία από τις εξισώσεις **(2.21)**. Ωστόσο οι Schwefel και Beyer αναφέρουν ότι αυτοί οι ανασυνδυασμοί εισάγουν μεγάλες διακυμάνσεις που δεν είναι πάντα επιθυμητές για τις παραμέτρους ελέγχου σ. Για την μετάλλαξη προτείνουν τις ακόλουθες εξισώσεις, ανάλογα με τον τύπο της μετάλλαξης:

Α. Για ισότροπη μετάλλαξη, δηλαδή μία μόνο παράμετρο ελέγχου σ, βλέπε **(2.25)**:

$$\sigma^{mut} = \sigma * exp(\tau * N(0,1)) \quad \textbf{(2.30)}$$

όπου για την παράμετρο τ:

$$\tau = \frac{1}{\sqrt{n}}\ \gamma\varepsilon\nu\iota\kappa\acute{\alpha}$$
$$\tau = \frac{1}{\sqrt{2n}}\ \gamma\iota\alpha\ \acute{\varepsilon}\nu\tau o\nu\alpha\ \mu\eta\ \kappa\upsilon\rho\tau\acute{\varepsilon}\varsigma\ \alpha\nu\tau\iota\kappa\varepsilon\iota\mu\varepsilon\nu\iota\kappa\acute{\varepsilon}\varsigma\ \sigma\upsilon\nu\alpha\rho\tau\acute{\eta}\sigma\varepsilon\iota\varsigma \quad \textbf{(2.31)}$$

Κάθε μέλος του πληθυσμού, έχει ως γονίδια τις αρχικές μεταβλητές σχεδιασμού του προβλήματος και μία παράμετρο σ, διαφορετική για κάθε μέλος.

Β. Για ανισότροπη μετάλλαξη, δηλαδή ένα διάνυσμα με παραμέτρους σ_i, βλέπε **(2.26)**:

$$\boldsymbol{\sigma}^{mut} = \exp\left(\tau_0 N_0(0,1)\right) * \left[\sigma_1 \exp(\tau N_1(0,1)),\ \sigma_2 \exp(\tau N_2(0,1)),\ \ldots,\ \sigma_n \exp(\tau N_n(0,1))\right]^T \quad \textbf{(2.32)}$$

όπου για τις παραμέτρους τ:

$$\tau_0 = \frac{1}{\sqrt{2n}}, \qquad \tau = \frac{1}{\sqrt{2\sqrt{n}}} \quad \textbf{(2.33)}$$

Κάθε μέλος του πληθυσμού έχει δύο χρωμοσώματα. Το ένα περιέχει τις αρχικές μεταβλητές σχεδιασμού του προβλήματος. Το δεύτερο είναι ένα διάνυσμα $\boldsymbol{\sigma} = \left[\sigma_1,\ \sigma_2,\ \ \ldots,\ \sigma_n\right]$, διαφορετικό για κάθε μέλος.

C. Για ανισότροπη μετάλλαξη με περιστροφή, βλέπε **(2.26)**, εχούν επίσης αναπτυχθεί συσχετισμένες μεταλλάξεις (Schwefel 1995, Schwefel & Rudolf 1995, Hansen & Ostermeier 2001)

Σε όλες τις παραπάνω περιπτώσεις, ο τελεστής μετάλλαξης λειτουργεί σε δύο φάσεις:

- Μετάλλαξη της τυπικής απόκλισης σ ή του διανύσματος $\boldsymbol{\sigma}$, σύμφωνα με τις εξισώσεις **(2.30**) ή **(2.32)**, αντίστοιχα.
- Μετάλλαξη του χρωμοσώματος $\boldsymbol{x}_l$ χρησιμοποιώντας τις μεταλλαγμένες τυπικές αποκλίσεις.

2.6.2.3 Επιλογή (selection)

Η τεχνική με την οποία οι Στρατηγικές Εξέλιξης και γενικά οι Εξελικτικοί Αλγόριθμοι κατευθύνουν την αναζήτηση προς τις βέλτιστες περιοχές του χώρου σχεδιασμού είναι ο τελεστής επιλογής. Σύμφωνα με αυτόν σε κάθε γενιά επιβιώνουν μόνο τα καταλληλότερα χρωμοσώματα, δηλαδή αυτά με την βέλτιστη συνάρτηση ποιότητας (fitness). Μία απαίτηση που πρέπει να ικανοποιεί η επιλογή είναι συνήθως η διατήρηση του μεγέθους του πληθυσμού από γενιά σε γενιά. Στην παρούσα εργασία θα χρησιμοποιηθούν δύο εναλλακτικοί τελεστές επιλογής:

Έστω, ότι σε μία γενιά g ένας πληθυσμός μ γονέων παράγει λ απογόνους.

A. Η επιλογή ***(μ+λ)-ES***. Τα καλύτερα μ μέλη εκ των $\mu+\lambda$ συνολικών μελών επιλέγονται για να αποτελέσουν τους γονείς της επόμενης γενιάς. Δεν υπάρχει κάποιος περιορισμός για το μέγεθος λ του πληθυσμού των απογόνων, πέραν του να υπάρχει ($\lambda \geq 1$). Η παραλλαγή αυτή εγγυάται την επιβίωση του καλύτερου μέλους του πληθυσμού που έχει βρεθεί από την αρχή της αναζήτησης. Το καλύτερο αυτό χρωμόσωμα μπορεί να ζήσει επ' άπειρον, κάτι που δεν συναντάται στην φύση. Τέτοιες τεχνικές χαρακτηρίζονται ως «ελιτιστικές» και γενικά επιταχύνουν την σύγκλιση της μεθόδου, αφού δε χάνεται σε κάθε γενιά η πληροφορία της καλύτερης λύσης που έχει βρεθεί.

B. Η επιλογή ***(μ,λ)-ES***. Ως γονείς της επόμενης γενιάς επιλέγονται τα καλύτερα μ μέλη μόνο από τον πληθυσμό των λ απογόνων. Η διαδικασία αυτή προσομοιάζει καλύτερα την εξέλιξη στην φύση, αφού οι μ γονείς αναπαράγονται και πεθαίνουν, έχουν δηλαδή διάρκεια ζωής μία μόνο γενιά, ακόμα και αν είναι καταλληλότεροι από τους απογόνους τους. Έτσι θυσιάζεται η ταχύτητα σύγκλισης, αλλά αυξάνεται η τυχηματικότητα της μεθόδου, με αποτέλεσμα να απεγκλωβίζεται ευκολότερα από τοπικά ελάχιστα (οι απόγονοι τείνουν να δοκιμάζουν νέες «ιδέες», διαφορετικές από τους γονείς τους). Η (μ, λ) επιλογή έχει νόημα μόνο αν $\lambda > \mu$, ώστε να διατηρείται το μέγεθος του πληθυσμού. Η περίπτωση $\lambda = \mu$, όπου όλοι οι λ απόγονοι επιβιώνουν, δεν είναι αποτελεσματική. Αυτό συμβαίνει, επειδή ο τελεστής επιλογής αποτυγχάνει να παρέχει πληροφορίες που θα κατευθύνουν την αναζήτηση στον χώρο σχεδιασμού, η οποία γίνεται πλέον στα τυφλά.

2.6.2.4 Αρχική επιλογή

Οι Schwefel, Rudolf και Bäck προτείνουν δύο τρόπους επιλογής των αρχικών χρωμοσωμάτων:

A. Τυχαία και ομοιόμορφη διασπορά στο χώρο σχεδιασμού. Έχει περιγραφεί αναλυτικά στην ενότητα **2.5.4**

B. Προσδιορισμός ενός αρχικού χρωμοσώματος $\boldsymbol{x}_{spec}^{(0)}$ από τον χρήστη, το οποίο αντιστοιχεί σε έναν από τα μέλη του πληθυσμού γονέων της πρώτης γενιάς. Οι υπόλοιποι $\mu - 1$ γονείς παράγονται με μετάλλαξη του $\boldsymbol{x}_{spec}^{(0)}$, χρησιμοποιώντας αυξημένες τυπικές αποκλίσεις:

$$x_{m,i}^{(0)} = x_{spec,i}^{(0)} + c * \sigma_{m,i}^{(0)} * N(0,1) \quad \textbf{(2.34)}$$

όπου:

- Ο δείκτης $m = 1,2, \dots, \mu$ αναφέρεται στα μέλη του πληθυσμού των γονέων.
- Ο δείκτης $i = 1,2, \dots, n$ αναφέρεται στις μεταβλητές σχεδιασμού.

- Ο εκθέτης (0) επισημαίνει την πρώτη γενιά.
- Η τυπική απόκλιση $\sigma_{m,i}^{(0)}$ της πρώτης γενιάς είναι γενικά διαφορετική για κάθε γονίδιο και για κάθε μέλος του πληθυσμού (ισότροπη μετάλλαξη).
- c είναι ο συντελεστής μεγέθυνσης του βήματος μετάλλαξης. Μπορεί να πάρει τιμές ώστε $c > 1$, π.χ. $c > 10$.

2.6.2.5 Κριτήρια τερματισμού

Για τον τερματισμό της μεθόδου μπορούν να χρησιμοποιηθούν τα συνηθισμένα κριτήρια με βάση:

- Τους υπολογιστικούς πόρους(μέγιστος αριθμός επαναλήψεων, μέγιστος χρόνος εκτέλεσης στον υπολογιστή).
- Την σύγκλιση των καλύτερων λύσεων κάθε γενιάς.
- Την σύγκλιση των παραμέτρων ελέγχου (συνήθως για τις τυπικές αποκλίσεις της μετάλλαξης).

Ο Schwefel (1981) προτείνει τα εξής δύο κριτήρια, που ανήκουν στην δεύτερη κατηγορία από τις παραπάνω, για τις Στρατηγικές Εξέλιξης πολλών μελών. Η μέθοδος τερματίζει όταν:

1) Η σχετική διαφορά της καλύτερης και χειρότερης τιμής της αντικειμενικής συνάρτησης των γονέων στην τρέχουσα γενιά, είναι μικρότερη από μία δεδομένη μικρή τιμή ε_1. Τυπικά επιλέγεται $\varepsilon_1 \leq 10^{-4}$.
2) Η μέση τιμή των αντικειμενικών συναρτήσεων όλων των γονέων στις τελευταίες k_f γενιές έχει βελτιωθεί λιγότερο από μια δεδομένη μικρή τιμή ε_2. Τυπικά επιλέγονται $k_f \geq 2n$ όπου n είναι ο αριθμός των μεταβλητών σχεδιασμού, και $\varepsilon_2 \leq 10^{-3}$.

2.6.3 ΣΕ πολλών μελών για διακριτές μεταβλητές σχεδιασμού

Οι J. Cai και G. Thierauf (1996) πρότειναν μία παραλλαγή των Στρατηγικών Εξέλιξης για διακριτές μεταβλητές και την εφάρμοσαν σε προβλήματα βέλτιστου σχεδιασμού κατασκευών.

2.6.3.1 Ανασυνδυασμός (Recombination)

Ορίζεται όπως ακριβώς και στις συνεχείς μεταβλητές σχεδιασμού, με εξαίρεση τους τύπους που χρησιμοποιούσαν μέσους όρους, αφού ο μέσος όρος δύο διακριτών τιμών δεν ανήκει στο σύνολο των δυνατών τιμών της μεταβλητής. Έτσι έχουμε:

$$x_{l,i}^{temp} = \begin{cases} x_{a,i} \text{ ή } x_{b,i} \; στην \; τύχη & (A) \\ x_{a,i} \text{ ή } x_{best,i} \; στην \; τύχη & (B) \\ x_{rand,i} & (C) \\ x_{a,i} \text{ ή } x_{rand,i} \; στην \; τύχη & (D) \\ x_{best,i} \text{ ή } x_{rand,i} \; στην \; τύχη & (E) \end{cases} \quad \textbf{(2.35)}$$

όπου $\boldsymbol{x}_{\alpha}$ και $\boldsymbol{x}_{b}$ είναι δύο διανύσματα που επιλέγονται τυχαία από των πληθυσμό των γονέων. Για κάθε προσωρινό απόγονο $\boldsymbol{x}_{l}^{temp}$ επιλέγονται δύο διαφορετικά $\boldsymbol{x}_{\alpha}$ και $\boldsymbol{x}_{b}$ αλλά διατηρούνται τα ίδια για όλα τα γονίδια αυτού του απογόνου. Αντίθετα, το γονίδιο $x_{rand,i}$ επιλέγεται τυχαία από όλα τα γονίδια i των χρωμοσωμάτων του πληθυσμού γονέων και είναι διαφορετικό για κάθε γονίδιο $x_{l,i}^{temp}$ του ίδιου απογόνου. Το διάνυσμα $\boldsymbol{x}_{best}$ δεν επιλέγεται τυχαία, αλλά είναι το καλύτερο χρωμόσωμα του πληθυσμού γονέων της εκάστοτε γενιάς. Τα δύο γονίδια που εμφανίζονται σε κάθε περίπτωση, εκτός της (C), έχουν πιθανότητες επιλογής 50%-50%.

2.6.3.2 Μετάλλαξη (Mutation)

Η μετάλλαξη εφαρμόζεται με την προσθήκη ενός διανύσματος τυχηματικής αλλαγής Z στο χρωμόσωμα κάθε προσωρινού απογόνου, όπως ακριβώς στην περίπτωση των συνεχών μεταβλητών **(2.24)**. Για να επιλέξουμε όμως το διάνυσμα αυτό, πρέπει να λάβουμε υπόψη μια βασική διαφορά που παρουσιάζουν οι διακριτές μεταβλητές. Στην βιολογική εξέλιξη μικρές μεταλλάξεις συμβαίνουν με μεγάλη συχνότητα, ενώ μεγάλες μεταλλάξεις σπάνια. Η κανονική κατανομή που χρησιμοποιείται για συνεχή προβλήματα, προσομοιάζει με ικανοποιητική ακρίβεια την διακύμανση αυτή, εφόσον η τυπική απόκλιση είναι μικρή.

Στην περίπτωση διακριτών μεταβλητών, η μετάλλαξη ενός γονιδίου δεν είναι το ίδιο μικρή, ακόμα και αν το γονίδιο πάρει την αμέσως επόμενη/προηγούμενη δυνατή τιμή. Αυτό οφείλεται στην ασυνέχεια του χώρου σχεδιασμού, που επιβάλει μεγάλες μεταβολές της αντικειμενικής συνάρτησης. Γι' αυτό οι Thierauf & Cai προτείνουν την μετάλλαξη λίγων μόνο γονιδίων σε κάθε χρωμόσωμα. Έτσι σε κάθε χρωμόσωμα της εκάστοτε γενιάς επιλέγονται τυχαία q γονίδια προς μετάλλαξη σύμφωνα με την σχέση:

$$z_i = \begin{cases} \pm(\kappa + 1) * \delta\sigma_i, \; για \; τα \; q \; τυχαία \; επιλεγμένα \; γονίδια \\ 0, \; για \; τα \; υπόλοιπα \; n - q \; γονίδια \;\; (q < n) \end{cases} \quad \textbf{(2.36)}$$

όπου:

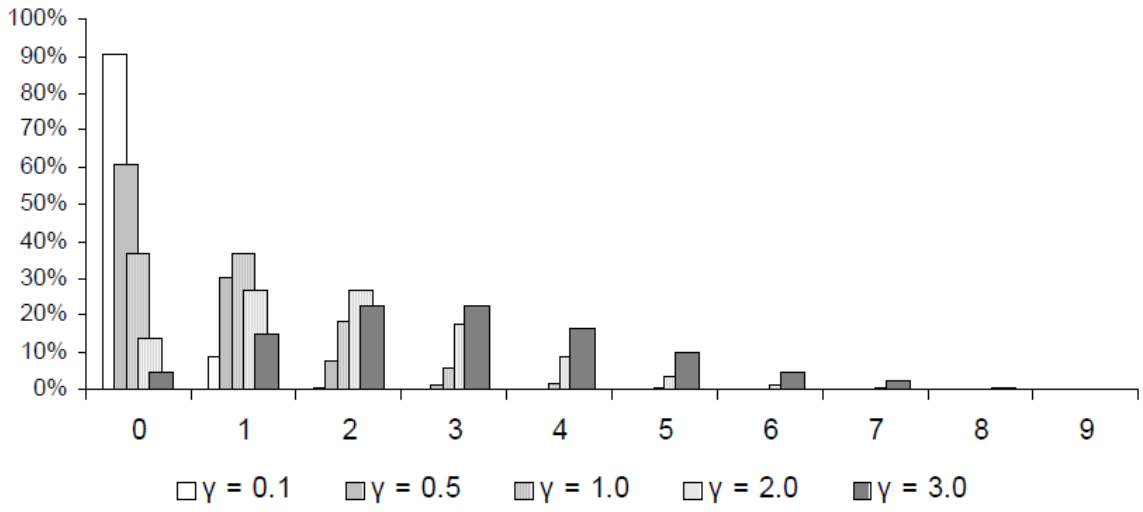
- $\delta\sigma_i$ είναι η διαφορά μεταξύ δύο γειτονικών τιμών του συνόλου των διακριτών τιμών της μεταβλητής x_i.

- τα q γονίδια επιλέγονται τυχαία και σύμφωνα με την ομοιόμορφη κατανομή από τα n συνολικά γονίδια. Μετά από δοκιμές σε προβλήματα βέλτιστου σχεδιασμού κατασκευών, προτείνονται τιμές q από $\frac{1}{10} * n$ ως $\frac{2}{5} * n$.
- κ είναι ένας ακέραιος τυχαίος αριθμός που ακολουθεί την κατανομή Poisson, όπως αναλύεται στη συνέχεια.

Μία τυχαία διακριτή μεταβλητή κ ακολουθεί την κατανομή Poisson με παράμετρο $\gamma > 0$, όταν έχει συνάρτηση μάζας πιθανότητας:

$$P(\kappa) = \frac{\gamma^{\kappa}}{\kappa!}\exp(-\gamma) \qquad \textbf{(2.37)}$$

Η παράμετρος γ ισούται με την μέση αναμενόμενη τιμή, επομένως πρέπει να παίρνει μικρές (θετικές) τιμές κοντά στο 0. Επίσης ισούται και με την διασπορά της κατανομής, με το τετράγωνο της τυπικής απόκλισης δηλαδή. Για τιμές του γ κοντά στο 0.001, ο αριθμός κ λαμβάνει την τιμή 0 με πιθανότητα μεγαλύτερη από $99{,}99\%$, ενώ για $\gamma = 0.05$, ο αριθμός κ λαμβάνει την τιμή 0 με πιθανότητα περίπου 95% και την τιμή 1 με πιθανότητα περίπου 5%.



Σχήμα 2.20 Κατανομές Poisson για μικρές τιμές της παραμέτρου γ.

2.6.3.2.1 Προσαρμογή του βήματος μετάλλαξης

Η διασπορά γ επηρεάζει την ένταση της μετάλλαξης σε διακριτά προβλήματα, παρόμοια με την τυπική απόκλιση σ σε συνεχή. Για να αποφευχθεί η στασιμότητα της σύγκλισης στις τελευταίες επαναλήψεις, πρέπει να εφαρμοστεί και εδώ κάποια προσαρμογή του βήματος της μετάλλαξης. Μπορεί να εφαρμοστεί ανάλογη διαδικασία με τον *κανόνα του 1/5* (βλέπε ενότητα **2.6.3.2.1**). Αρχικά επιλέγεται μία σχετικά μεγάλη τιμή της διασποράς (συνήθως $\gamma_0 = 0.01$). Αν μετά από τη δημιουργία της νέας γενιάς δεν έχει επιτευχθεί κάποια βελτίωση

στην τιμή της αντικειμενικής συνάρτησης, τότε η τιμή της διασποράς μειώνεται, ενώ σε αντίθετη περίπτωση η τιμή της διασποράς αυξάνεται. Η αύξηση ή μείωση είναι ποσοστιαία και μεγέθους $5-10\%$ της παλαιάς τιμής του γ.

2.6.3.3 Επιλογή (Selection)

Χρησιμοποιούνται και εδώ οι δύο τελεστές επιλογής που αναπτύχθηκαν στην ενότητα **2.6.2.3**, δηλαδή η *(μ+λ)-ES* και η *(μ,λ)-ES*. Οι Thierauf & Cai αναφέρουν ότι στις δοκιμές τους πάνω σε διακριτά προβλήματα βέλτιστου σχεδιασμού κατασκευών, παρατήρησαν ταχύτερη σύγκλιση και καλύτερες τελικές λύσεις για την επιλογή *(μ+λ)-ES.*

2.6.3.4 Τερματισμός

Σε διακριτά προβλήματα βέλτιστου σχεδιασμού, η μέθοδος μπορεί να τερματίσει όταν:

1) Η καλύτερη τιμή της αντικειμενικής συνάρτησης δεν έχει βελτιωθεί κατά τις τελευταίες $4*n*\mu/\lambda$ γενιές.
2) Η μέσης τιμή των αντικειμενικών συναρτήσεων όλων των γονέων κατά τις τελευταίες $2*n*\mu/\lambda$ γενιές παρουσιάζει ποσοστιαία βελτίωση μικρότερη από ένα όριο ε_b (π.χ. $\varepsilon_b = 10^{-4}$).
3) Η σχετική διαφορά μεταξύ της καλύτερης και μέσης τιμής της αντικειμενικής συνάρτησης για όλους τους γονείς της τρέχουσας γενιάς είναι μικρότερη από ένα όριο ε_c (π.χ. $\varepsilon_c = 10^{-4}$).
4) Ο λόγος μ_b/μ έχει φτάσει την τιμή $\varepsilon_d \in [0.5, 0.8]$, όπου μ_b είναι ο πλήθος των γονέων στην τρέχουσα γενιά, που έχουν τιμή αντικειμενικής συνάρτησης ίση με την καλύτερη που έχει επιτευχθεί.

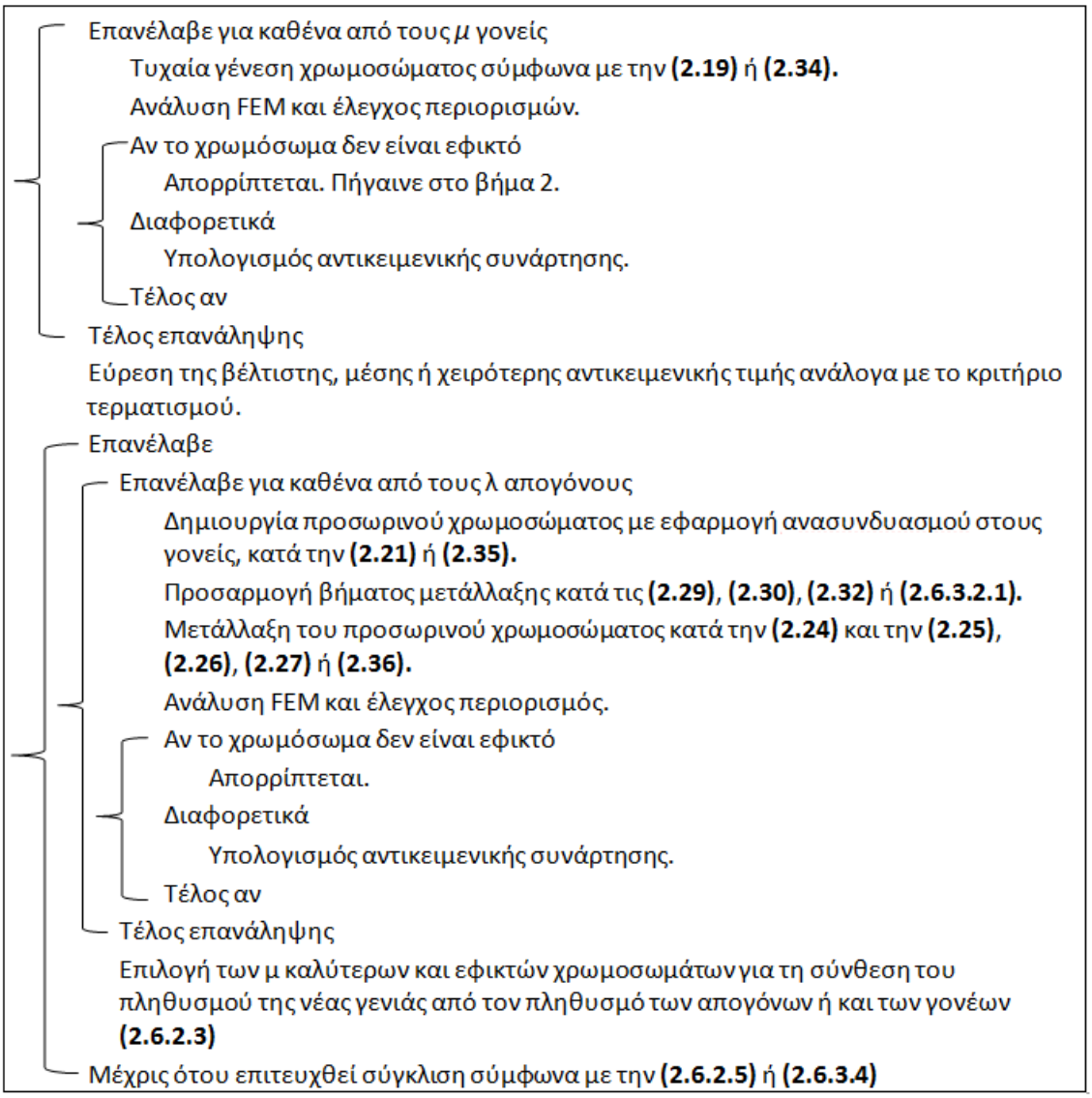
2.6.4 Ο αλγόριθμος ΣΕ για το βέλτιστο σχεδιασμό κατασκευών

Η παραδοσιακή τεχνική χειρισμού των περιορισμών στις Στρατηγικές Εξέλιξης είναι η *θανατική ποινή* (βλέπε ενότητα **2.2.1**). Συνήθως εφαρμόζεται ως εξής:

1) Τυχαία γένεση αρχικών λύσεων και έλεγχος περιορισμών, μέχρι να βρεθούν μ εφικτά χρωμοσώματα.
2) Δημιουργία του πληθυσμού απογόνων με ανασυνδυασμό και μετάλλαξη και έλεγχος περιορισμών.
3) Ο τελεστής της επιλογής λαμβάνει υπόψη μόνο τα εφικτά χρωμοσώματα για τη σύνθεση του πληθυσμού της επόμενης γενιάς

Έτσι διατηρείται η εφικτότητα των λύσεων από γενιά σε γενιά. Ωστόσο, οι πληροφορίες από τη δοκιμή μη εφικτών λύσεων χάνονται και περιοχές του χώρου σχεδιασμού, που πιθανόν να περιέχουν το ολικά βέλτιστο, μένουν ανεξερεύνητες. Επομένως, η μέθοδος μπορεί να

ωφεληθεί από τη χρήση συναρτήσεων ποινής (βλέπε ενότητα **2.2.2**). Στις αριθμητικές εφαρμογές της παρούσας εργασίας πάντως θα χρησιμοποιηθεί ο εξής αλγόριθμος:



Σχήμα 2.21 Ψευδό-κώδικας για τις ΣΕ, όπως χρησιμοποιούνται στις αριθμητικές εφαρμογές.

Κεφάλαιο 3

3 Αξιολόγηση Αλγορίθμων Βελτιστοποίησης

3.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζονται έλεγχοι των τριών αλγορίθμων βελτιστοποίησης, που αναπτύχθηκαν στο κεφάλαιο 2, σε λίγα από τα συνηθέστερα προβλήματα αξιολόγησης (benchmark problems). Γενικά για να αποτιμηθεί η αξιοπιστία και η ταχύτητα σύγκλισης ενός αλγορίθμου, αλλά και για να συγκριθεί με κάποιον άλλο, είναι απαραίτητη η εφαρμογή του σε ένα μεγάλο πλήθος προβλημάτων αξιολόγησης, καθένα από τα οποία εξετάζουν διαφορετικές ιδιότητες του αλγορίθμου. Ένας αλγόριθμος βελτιστοποίησης που προτείνεται για πρώτη φορά, σχεδόν πάντα συνοδεύεται από τέτοιους ελέγχους. Παρ' όλα αυτά δεν υπάρχει κάποια τυποποιημένη λίστα των προβλημάτων αξιολόγησης. Ωστόσο κάποια benchmarks χρησιμοποιούνται συχνότερα από άλλα.

Πρέπει να τονιστεί ότι σκοπός αυτού του κεφαλαίου δεν είναι να συγκρίνει τις τρεις μεθόδους βελτιστοποίησης. Κάτι τέτοιο θα απαιτούσε έναν εξαιρετικά μεγάλο αριθμό δοκιμών και μεγάλη ποικιλία προβλημάτων αξιολόγησης για κάθε επιστημονικό και μη κλάδο που οι αλγόριθμοι βρίσκουν εφαρμογή. Επίσης, κάθε αλγόριθμος βελτιστοποίησης μπορεί να λειτουργήσει τελείως διαφορετικά, ανάλογα με τις παραμέτρους που επιλέγονται (τοπολογία σμήνους στην ΒΣΣ, τιμή της πιθανότητας crossover στην ΔΕ, …). Η εύρεση κατάλληλων τιμών για τις παραμέτρους σε πολύ συγκεκριμένα είδη προβλημάτων, αποτελεί από μόνη της αντικείμενο πολλών ερευνών. Άλλωστε, μία μέθοδος δεν θα αναπτυσσόταν και χρησιμοποιούταν αν αποδεικνυόταν σταθερά χειρότερη από κάποια άλλη. Οι αξιολογήσεις, λοιπόν, που γίνονται στη συνέχεια αποσκοπούν στον έλεγχο του σωστού προγραμματισμού των αλγορίθμων από τον γράφοντα και στην εισαγωγή του αναγνώστη στην όλη διαδικασία.

Στις επόμενες ενότητες παρουσιάζεται ξεχωριστά κάθε πρόβλημα βελτιστοποίησης για συνεχείς μεταβλητές σχεδιασμού. Δίνονται η αντικειμενική συνάρτηση, οι μεταβλητές, ο χώρος σχεδιασμού και οι περιορισμοί, αν υπάρχουν. Έπειτα, λύνεται από τις μεθόδους ΒΣΣ, ΔΕ και ΣΕ, οι παράμετροι των οποίων δηλώνονται σε πίνακες.

3.2 Μεθοδολογία Αξιολόγησης

Κάθε μέθοδος βελτιστοποίησης εφαρμόζεται 1000 φορές για κάθε πρόβλημα αξιολόγησης, ώστε να βρεθεί η καλύτερη, η χειρότερη και η μέση συμπεριφορά της στο εκάστοτε πρόβλημα. Υπενθυμίζεται ότι χρησιμοποιούνται μεταευριστικοί αλγόριθμοι, που είναι στοχαστικές διαδικασίες και η αρχική επιλογή των λύσεων είναι τελείως τυχαία. Επομένως κάθε φορά που

εφαρμόζεται η μέθοδος, ακολουθεί διαφορετική πορεία. Οι βέλτιστες λύσεις των προβλημάτων είναι γνωστές. Μετρώνται:

- Το ποσοστό των εκτελέσεων ολόκληρης της διαδικασίας βελτιστοποίησης, που δεν κατάφεραν να συγκλίνουν. Μας ενδιαφέρουν οι περιπτώσεις στις οποίες η μέθοδος βελτιστοποίησης παγιδεύεται σε τοπικά ελάχιστα. Ως κριτήριο τερματισμού τίθενται 10.000 επαναλήψεις, ώστε η μέθοδος να προλάβει να συγκλίνει στο ολικό ελάχιστο ή έστω να παγιδευτεί σε τοπικά. Αυτό έχει ως συνέπεια να εκτελούνται υπερβολικά πολλές δοκιμές διαφορετικών σχεδιασμών, κάτι που δεν θα ήταν δυνατόν να γίνει στην πράξη, αφού η δοκιμή κάθε σχεδιασμού απαιτεί μια επίλυση FEM. Σε πρακτικές εφαρμογές χρησιμοποιείται και κάποιο άλλο κριτήριο τερματισμού.
- Ο μέσος αριθμός δοκιμών διαφορετικών σχεδιασμών, που χρειάστηκαν πριν συγκλίνει η μέθοδος. Για τον υπολογισμό του μέσου όρου αγνοούνται όσες εκτελέσεις δεν κατάφεραν να συγκλίνουν.
- Ο ελάχιστος αριθμός δοκιμών διαφορετικών σχεδιασμών, που χρειάστηκαν πριν συγκλίνει η μέθοδος, καθώς και η αλληλουχία των καθολικά βέλτιστων σχεδιασμών που έχουν βρεθεί σε κάθε επανάληψη.
- Ο μέγιστος αριθμός δοκιμών διαφορετικών σχεδιασμών, που χρειάστηκαν πριν συγκλίνει η μέθοδος, καθώς και η αλληλουχία των καθολικά βέλτιστων σχεδιασμών που έχουν βρεθεί σε κάθε επανάληψη.
- Από τις εκτελέσεις που δεν κατάφεραν να συγκλίνουν, αυτή που κατέληξε στη χειρότερη (μέγιστη) τιμή αντικειμενικής συνάρτησης, καθώς και η αλληλουχία των καθολικά βέλτιστων σχεδιασμών που έχουν βρεθεί σε κάθε επανάληψη.

Μία μέθοδος θα θεωρείται ότι συνέκλινε ικανοποιητικά αν:

$$\frac{\left|f\left(\boldsymbol{x}_t^{Gbest}\right) - f(\boldsymbol{x}^*)\right|}{f(\boldsymbol{x}^*)} \leq 10^{-2} \qquad \textbf{(3.1)}$$

Όπου $\boldsymbol{x}_t^{Gbest}$ είναι η καθολικά βέλτιστη λύση που έχει βρεθεί μέχρι την επανάληψη t και $\boldsymbol{x}^*$ είναι η γνωστή βέλτιστη λύση του προβλήματος. Αν για τη γνωστή λύση $f(\boldsymbol{x}^*) = 0$, τότε ο παρανομαστής 0 και χρησιμοποιείται ο επόμενος τύπος:

$$f\left(\boldsymbol{x}_t^{Gbest}\right) \leq 10^{-4} \qquad \textbf{(3.2)}$$

3.3 Προβλήματα αξιολόγησης χωρίς περιορισμούς

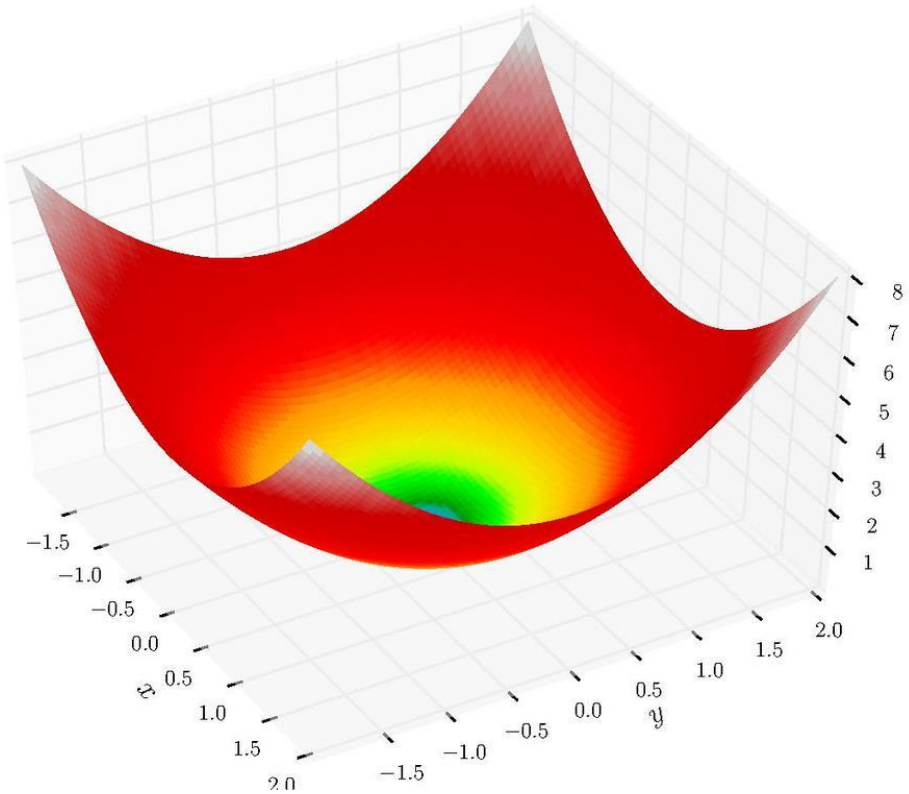
3.3.1 Συνάρτηση Σφαίρας

Είναι μία πολύ απλή αντικειμενική συνάρτηση. Παρουσιάζει ένα μόνο ολικό ελάχιστο και τόσα τοπικά, όσες οι διαστάσεις της, τα οποία βρίσκονται στα σύνορα του χώρου σχεδιασμού. Είναι συνεχής, κυρτή και μονοτροπική, δηλαδή παρουσιάζει μόνο ένα ολικό ελάχιστο.

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2 , \quad i = 1, 2, \dots, n \tag{3.3}$$

Ολικό ελάχιστο: $f(\boldsymbol{x}^*) = 0$ στο $\boldsymbol{x}^* = [0, 0, \dots, 0]^T$

Επιλέγονται: 10 μεταβλητές σχεδιασμού στα διαστήματα: $-50 \leq x_i \leq +50$



Σχήμα 3.1 Η συνάρτηση σφαίρας για 2 μεταβλητές σχεδιασμού.

3.3.1.1 ΒΣΣ

Πίνακας 3.1 Σφαίρα, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

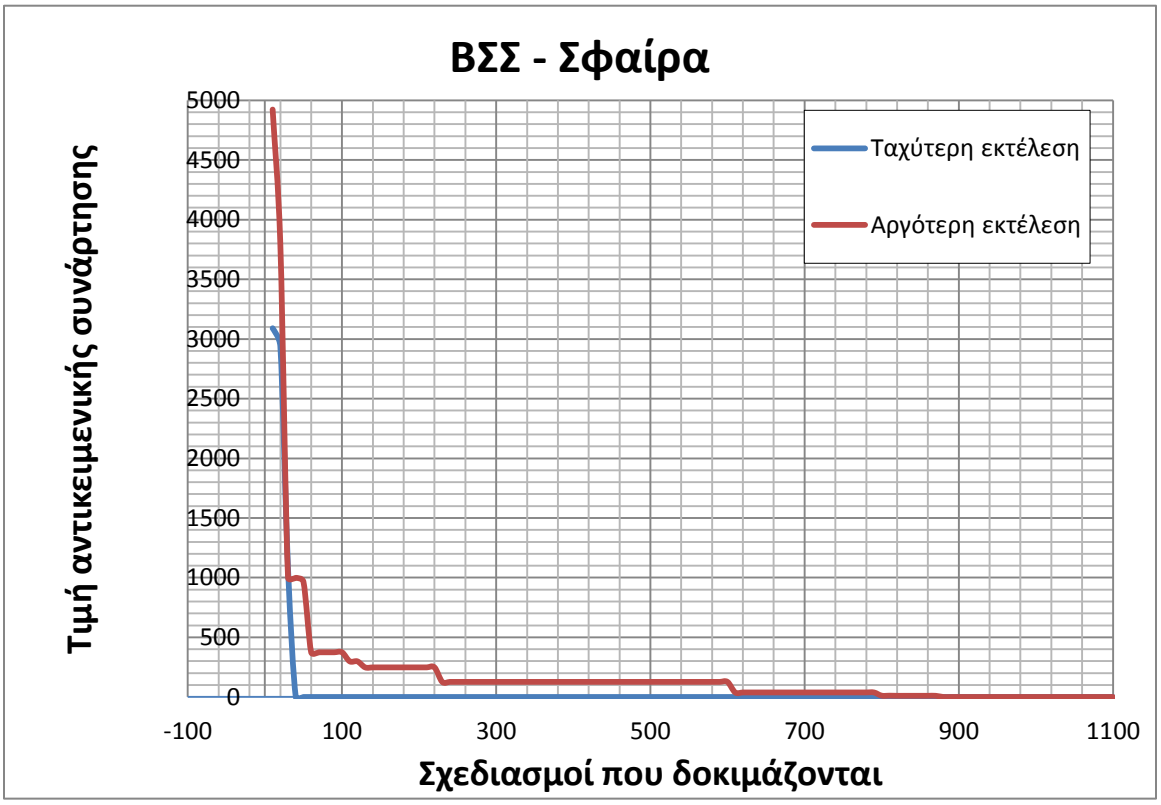
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}
10	Gbest	2.0	2.0	1.0	1.0	$(u_i - l_i)/2$

Πίνακας 3.2 Σφαίρα, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
40	1050	264.09	0%

Πίνακας 3.3 Σφαίρα, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[0, 0, \dots, 0]$	0.0
Αργότερη επίλυση	$[0, 0, \dots, 0]$	0.0
Χειρότερη επίλυση	-	-



Σχήμα 3.2 Σφαίρα, ΒΣΣ: Διάγραμμα σύγκλισης.

3.3.1.2 ΔΕ

Πίνακας 3.4 Σφαίρα, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

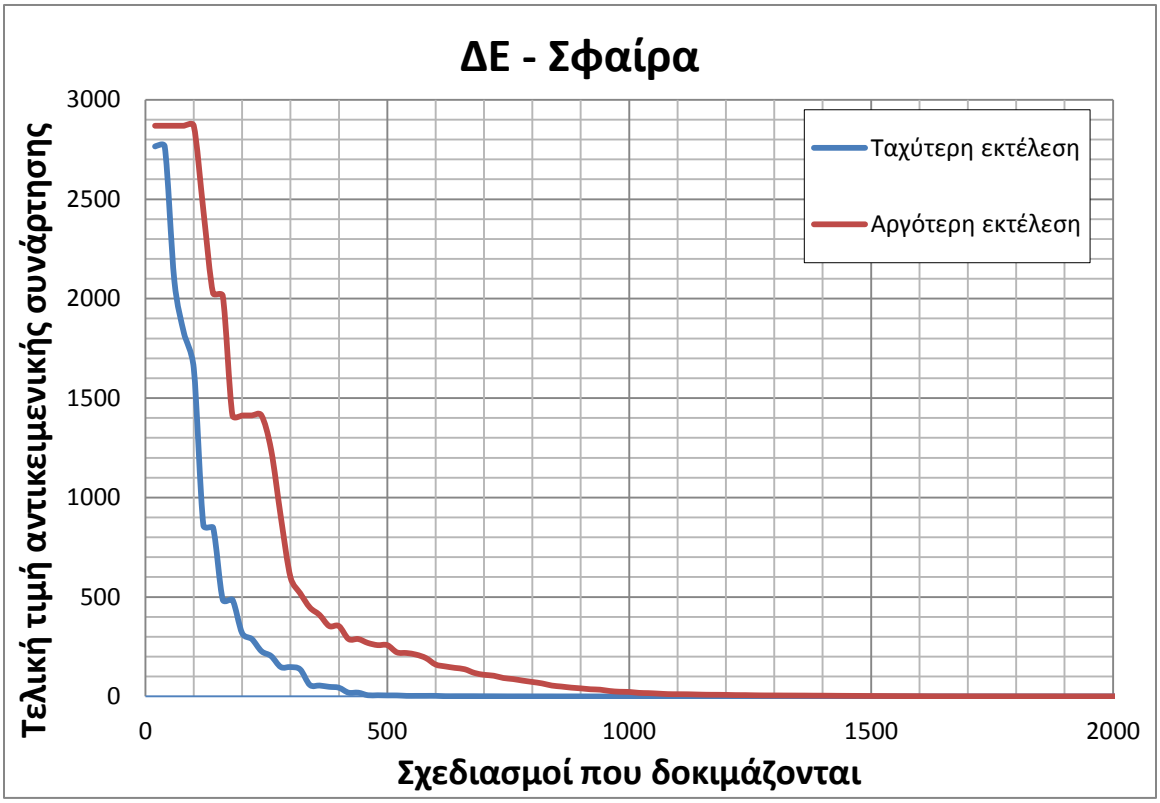
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR
20	best/1/bin	0.55	0.7

Πίνακας 3.5 Σφαίρα, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
1300	11000	1670.14	0%

Πίνακας 3.6 Σφαίρα, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[0, 0, \dots, 0]$	$1.11 * 10^{-174}$
Αργότερη επίλυση	$[0, 0, \dots, 0]$	$1.52 * 10^{-14}$
Χειρότερη επίλυση	-	-



Σχήμα 3.3 Σφαίρα, ΔΕ: Διάγραμμα σύγκλισης.

3.3.1.3 ΣΕ

Πίνακας 3.7 Σφαίρα, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

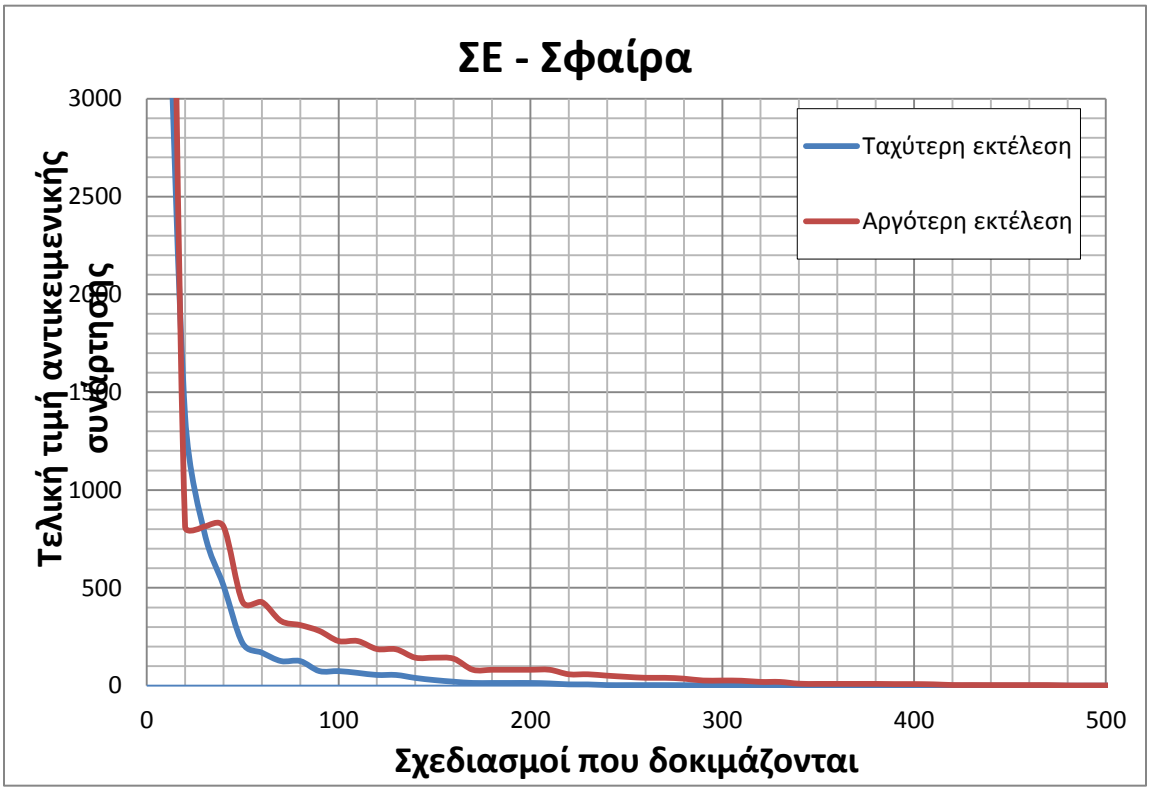
Επιλογή	Ανασυνδυασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ
μ+λ	$1/2 * (x_{a,i} + x_{rand,i})$	ισοτροπική	αυτοπροσαρμογή	10	10

Πίνακας 3.8 Σφαίρα, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
930	1880	1308.12	0%

Πίνακας 3.9 Σφαίρα, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$10^{-5} * [5.30, 4.65, 7.30, 6.99, -8.51, -0.67, 6.19, -1.88, 6.51, -3.91]$	$3.247 * 10^{-8}$
Αργότερη επίλυση	$10^{-5} * [3.05, -4.30, 9.78, 3.21, 1.61, -2.97, -3.19, 2.99, 4.93, 4.08]$	$2.052 * 10^{-8}$
Χειρότερη επίλυση	-	-



Σχήμα 3.4 Σφαίρα, ΣΕ: Διάγραμμα σύγκλισης.

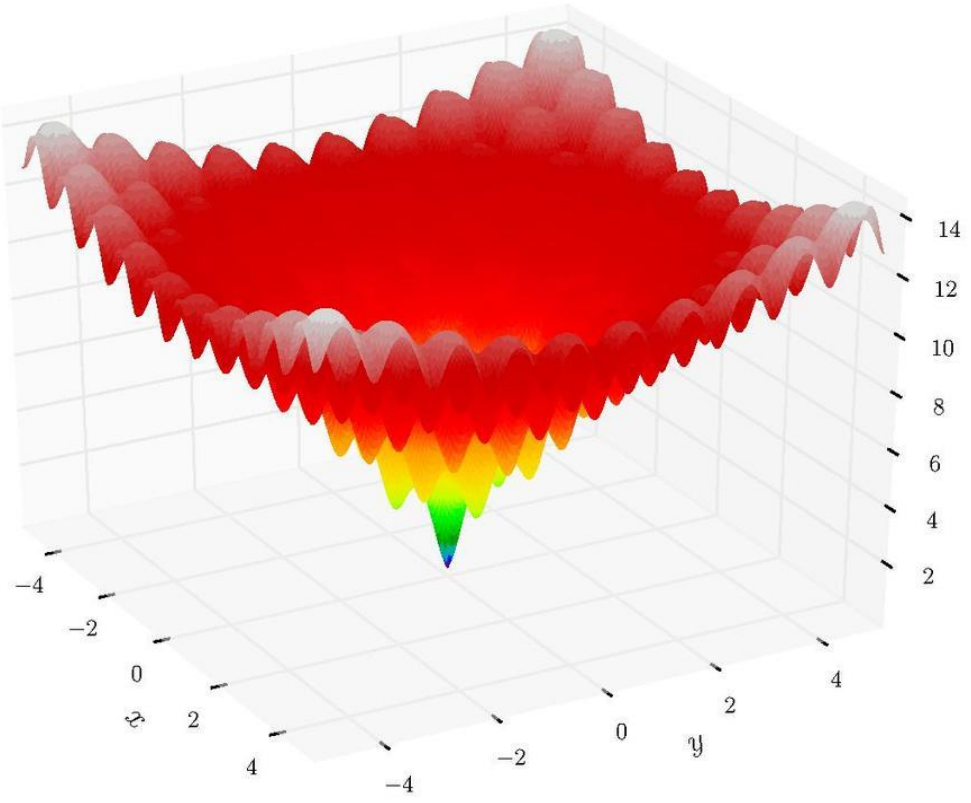
3.3.2 Συνάρτηση Ackley

Η συνάρτηση Ackley είναι αρκετά δημοφιλής για την αξιολόγηση αλγορίθμων βελτιστοποίησης. Είναι συνεχής, πολυτροπική (δηλ. περισσότερα από ένα τοπικά ελάχιστα), με ολικό ελάχιστο στο κέντρο του γραφήματος. Παρουσιάζει πολύ μεγάλο αριθμό τοπικών ελαχίστων, τα οποία παγιδεύουν τον αλγόριθμο βελτιστοποίησης.

$$f(\boldsymbol{x}) = -20exp\left(-0.2 * \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + \exp(1), \qquad i = 1, 2, \dots, n \tag{3.4}$$

Ολικό ελάχιστο: $f(\boldsymbol{x}^*) = 0$ στο $\boldsymbol{x}^* = [0, 0, \dots, 0]^T$

Επιλέγονται: 10 μεταβλητές σχεδιασμού στα διαστήματα: $-35 \leq x_i \leq +35$



Σχήμα 3.5 Η συνάρτηση Ackley για 2 μεταβλητές σχεδιασμού.

3.3.2.1 ΒΣΣ

Πίνακας 3.10 Ackley, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

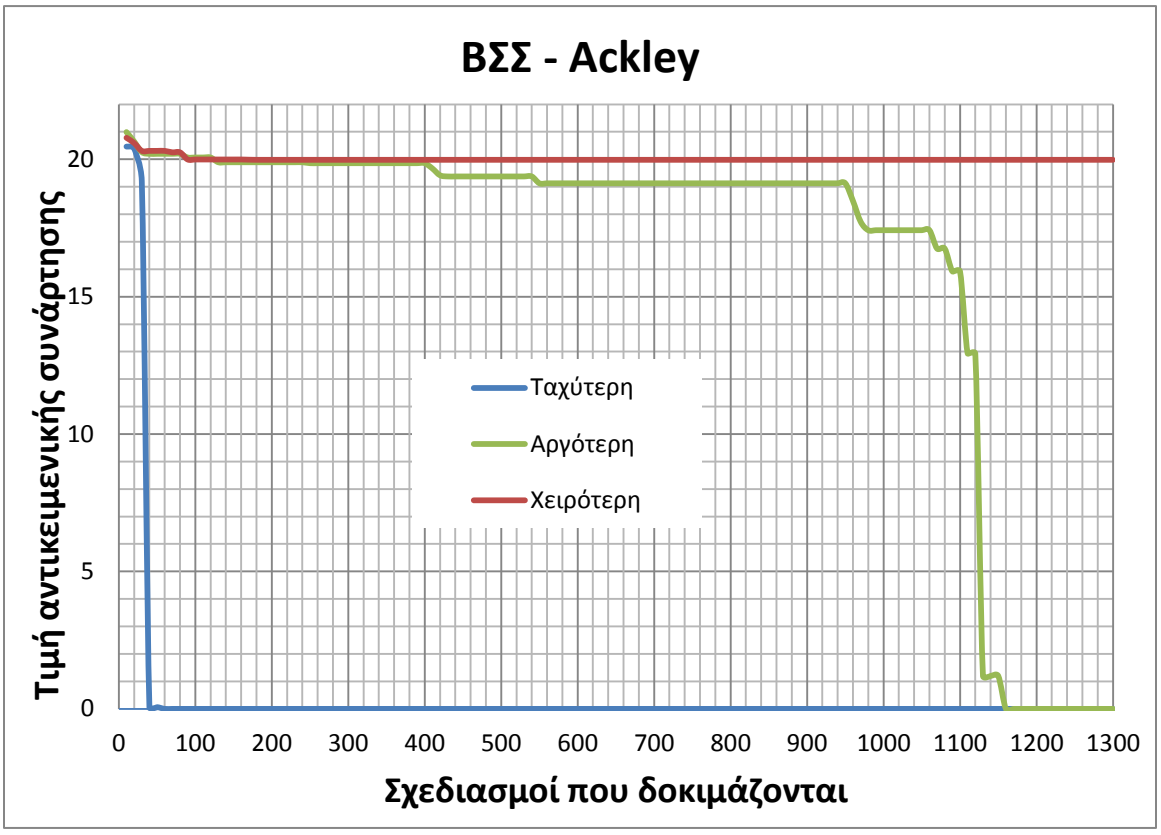
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}
10	Gbest	2.0	2.0	1.0	1.0	$(u_i - l_i)/2$

Πίνακας 3.11 Ackley, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
60	1160	277.8	3.1%

Πίνακας 3.12 Ackley, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[0, 0, \dots, 0]$	$4.44 * 10^{-16}$
Αργότερη επίλυση	$[0, 0, \dots, 0]$	$4.44 * 10^{-16}$
Χειρότερη επίλυση	$[-35.0, -35.0, \dots, -35.0]$	19.9817623



Σχήμα 3.6 Ackley, ΒΣΣ: Διάγραμμα σύγκλισης.

3.3.2.2 ΔΕ

Πίνακας 3.13 Ackley, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

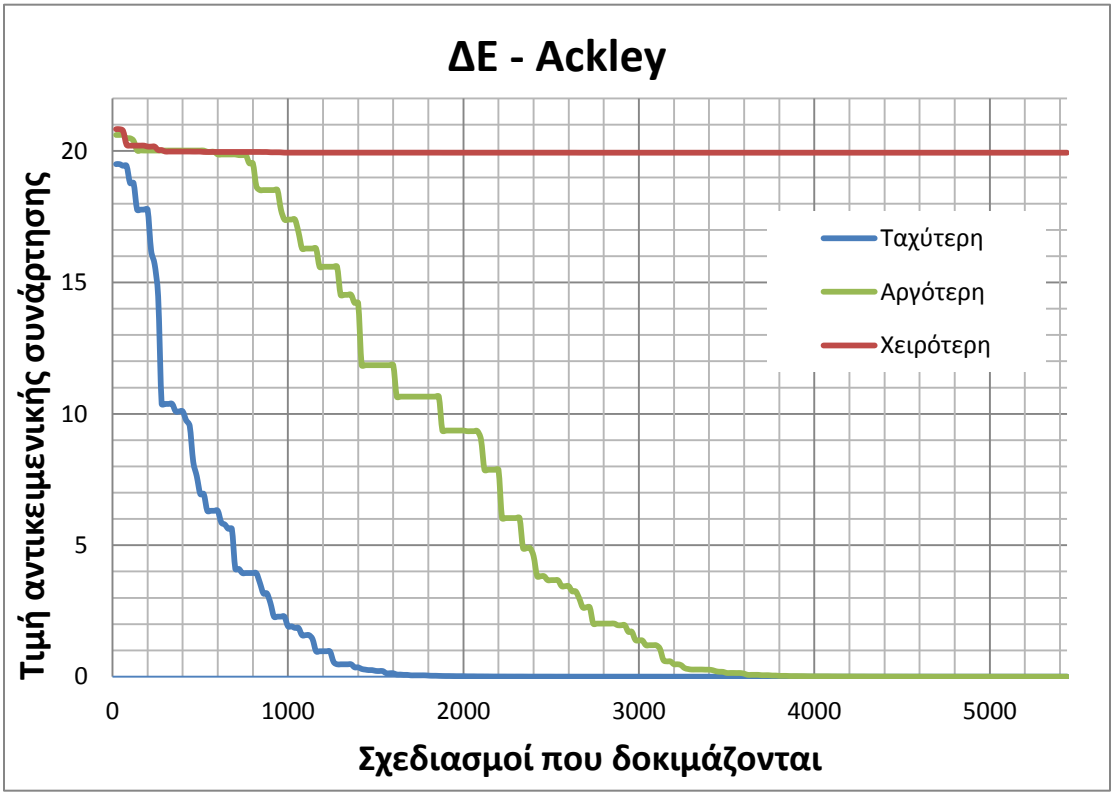
NP	μετάλλαξη, ανασυνδυασμός	F	CR
20	best/1/bin	0.6	0.4

Πίνακας 3.14 Ackley, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
3260	5440	3711.52	1.2%

Πίνακας 3.15 Ackley, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$10^{-16} * [5.54, 1.61, 16.01, 5.41, 3.52, 10.30, 7.63, -3.57, -6.90, 11.10]$	$3.997 * 10^{-15}$
Αργότερη επίλυση	$10^{-16} * [-5.55, -2.06, 23.2, -26.7, 8.22, 4.31, -14.9, 3.32, -1.72, -83.4]$	$1.52 * 10^{-14}$
Χειρότερη επίλυση	$[-3.999982, -35.0, -7.0, -25.970312, -35.0, 35.0, -1.999991, 35.0, 35.0, -35.0]$	19.937106



Σχήμα 3.7 Ackley, ΔΕ: Διάγραμμα σύγκλισης.

3.3.2.3 ΣΕ

Πίνακας 3.16 Ackley, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

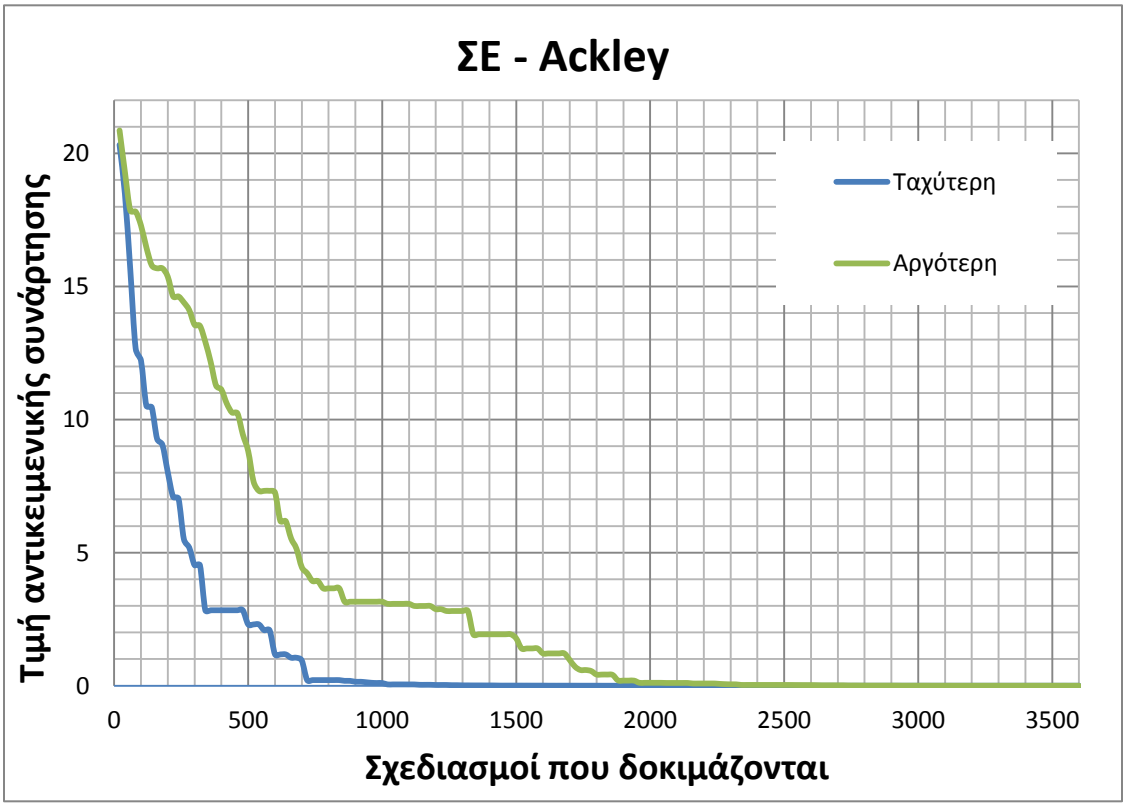
Επιλογή	Ανασυνδυασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ
μ+λ	$1/2 * (x_{a,i} + x_{rand,i})$	ισοτροπική	αυτοπροσαρμογή	10	20

Πίνακας 3.17 Ackley, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
1820	3520	2522.76	0%

Πίνακας 3.18 Ackley, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$10^{-5} * [3.37, -4.93, 0.52, 3.44, 7.54, 3.25, -8.00, -6.46, 5.15, 0.009]$	$1.992 * 10^{-4}$
Αργότερη επίλυση	$10^{-5} * [-4.97, -4.82, 1.27, 1.80, 1.29, -0.09, 5.42, -4.86, -5.86, -8.13]$	$1.827 * 10^{-4}$
Χειρότερη επίλυση	-	-



Σχήμα 3.8 Ackley, ΣΕ: Διάγραμμα σύγκλισης.

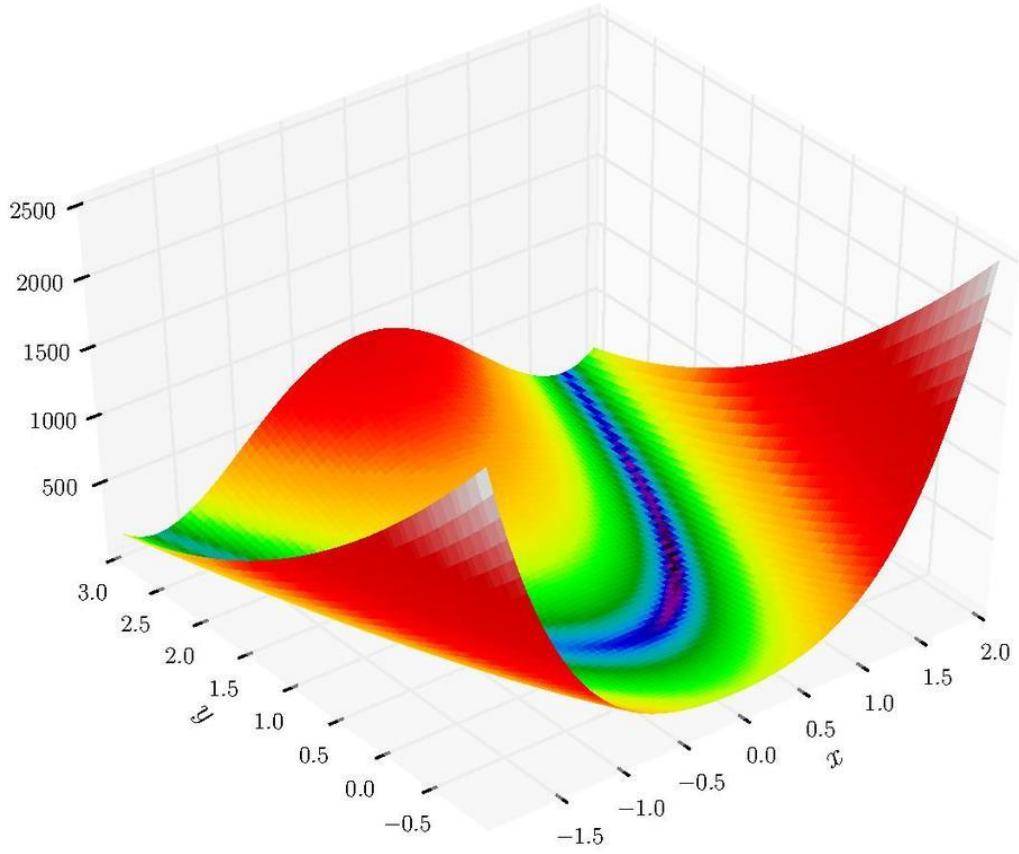
3.3.3 Συνάρτηση Rosenbrock

Η συνάρτηση Rosenbrock είναι συνεχής, κυρτή και μονοτροπική. Το ολικό ελάχιστο βρίσκεται σε μία στενή παραβολική κοιλάδα και είναι δύσκολο να προσεγγιστεί με ακρίβεια, παρότι η κοιλάδα βρίσκεται εύκολα.

$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right], \qquad i = 1, 2, \dots, n \qquad \textbf{(3.5)}$$

Ολικό ελάχιστο: $f(\boldsymbol{x}^*) = 0$ στο $\boldsymbol{x}^* = [1, 1, \dots, 1]^T$

Επιλέγονται: *10* μεταβλητές σχεδιασμού στα διαστήματα: $-10 \leq x_i \leq +10$



Σχήμα 3.9 Η συνάρτηση Rosenbrock για 2 μεταβλητές σχεδιασμού.

3.3.3.1 ΒΣΣ

Πίνακας 3.19 Rosenbrock, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

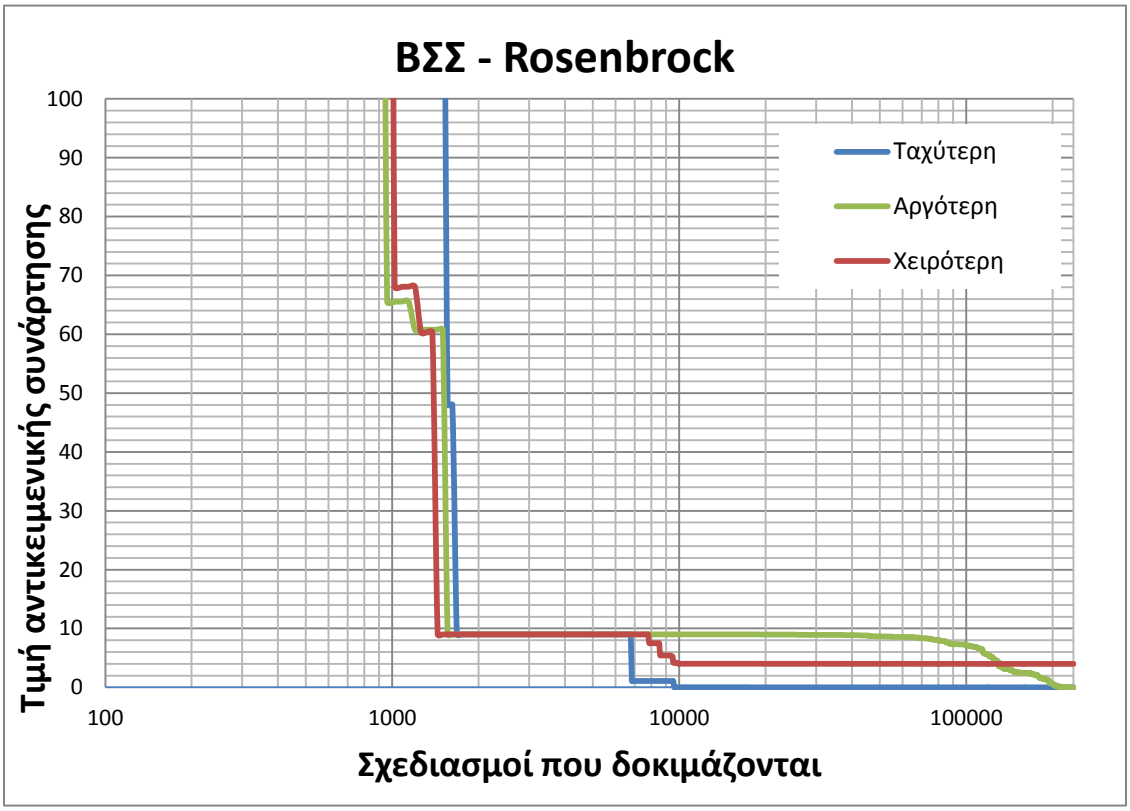
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}
60	Gbest	2.0	2.0	0.6	0.9	$(u_i - l_i)/2$

Πίνακας 3.20 Rosenbrock, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
18240	236640	190203.4	3.0%

Πίνακας 3.21 Rosenbrock, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1.0, 1.0, \dots, 1.0]$	0.0
Αργότερη επίλυση	$[1.0, 1.0, \dots, 1.0]$	0.0
Χειρότερη επίλυση	$[-0.99326, 0.99661, 0.99824, 0.99899, 0.99923, 0.99907, 0.99845, 0.99706, 0.99418, 0.98839]$	3.986579



Σχήμα 3.10 Rosenbrock, ΒΣΣ: Διάγραμμα σύγκλισης.

3.3.3.2 ΔΕ

Πίνακας 3.22 Rosenbrock, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

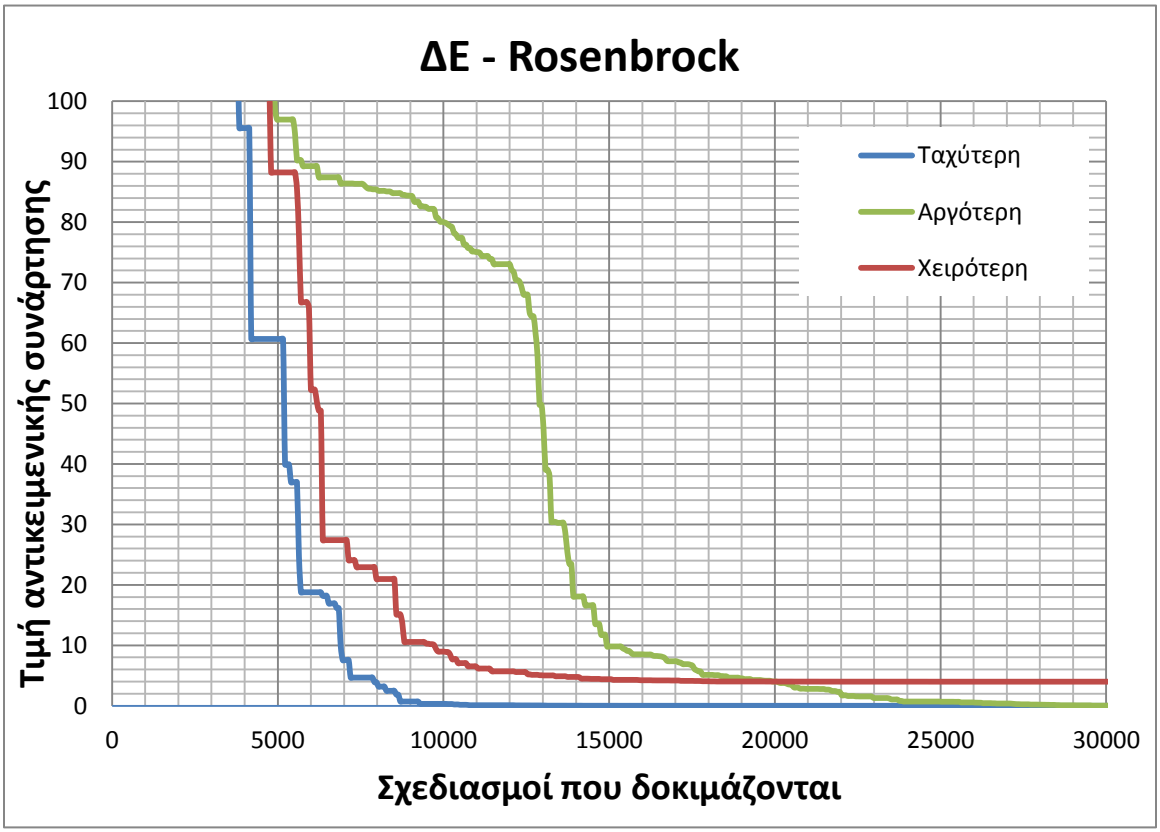
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR
60	best/2/bin	0.55	0.9

Πίνακας 3.23 Rosenbrock, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
20580	41100	29842.85	6.8%

Πίνακας 3.24 Rosenbrock, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1, 1, \dots, 1]$	0
Αργότερη επίλυση	$[1, 1, \dots, 1]$	0
Χειρότερη επίλυση	$[-0.99326, 0.99661, 0.998241, 0.998988, 0.999226, 0.999074, 0.998454, 0.997056, 0.994179, 0.988393]$	3.986579



Σχήμα 3.11 Rosenbrock, ΔΕ: Διάγραμμα σύγκλισης.

3.3.3.3 ΣΕ

Δοκιμάστηκαν πολλοί συνδυασμοί παραμέτρων για τις ΣΕ αλλά με κανέναν από αυτούς δεν συνέκλινε η μέθοδος.

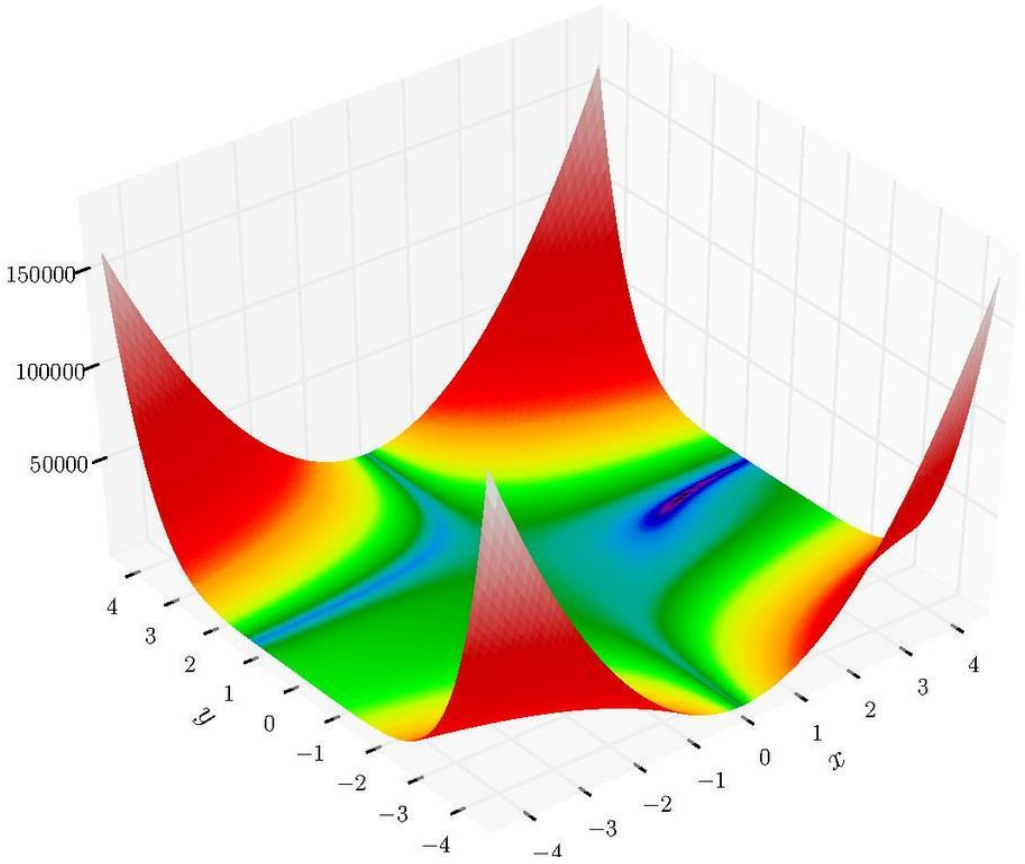
3.3.4 Συνάρτηση Beale

Η συνάρτηση Beale είναι συνεχής, πολυτροπική και ορίζεται μόνο για δύο μεταβλητές σχεδιασμού, σε αντίθεση με τις προηγουμένες.

$$f(\boldsymbol{x}) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2, \quad i = 1, 2, \dots, n \qquad \textbf{(3.6)}$$

Ολικό ελάχιστο: $f(\boldsymbol{x}^*) = 0$ στο $\boldsymbol{x}^* = [3,\ 0.5]^T$

Χώρος σχεδιασμού: $-4.5 \leq x \leq +4.5,\ -4.5 \leq y \leq +4.5$



Σχήμα 3.12 Η συνάρτηση Beale.

3.3.4.1 ΒΣΣ

Πίνακας 3.25 Beale, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

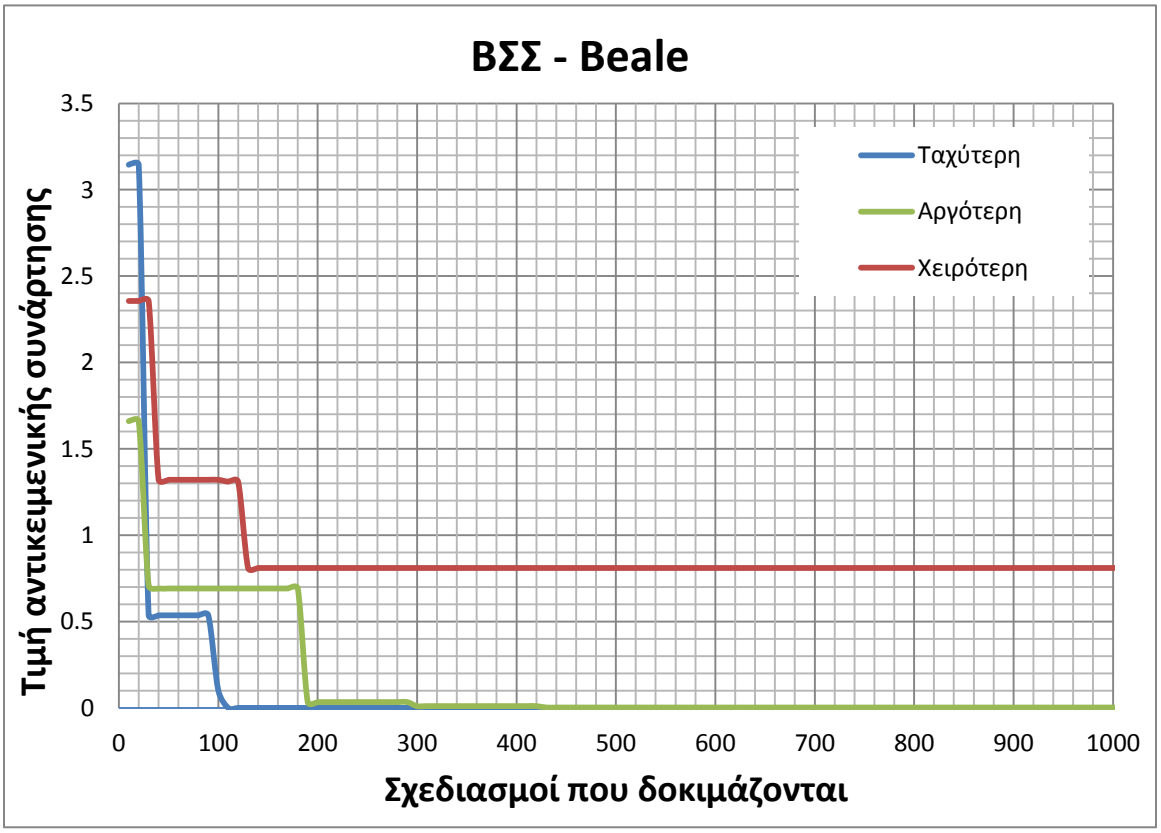
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}
10	Gbest	2.0	2.0	0.4	1.0	$(u_i - l_i)/2$

Πίνακας 3.26 Beale, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
110	25570	15891.8	0.6%

Πίνακας 3.27 Beale, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[3.0, 0.5]$	0.0
Αργότερη επίλυση	$[3.0, 0.5]$	0.0
Χειρότερη επίλυση	$[-4.5, 1.186429]$	0.76206965



Σχήμα 3.13 Beale, ΒΣΣ: Διάγραμμα σύγκλισης.

3.3.4.2 ΔΕ

Πίνακας 3.28 Beale, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

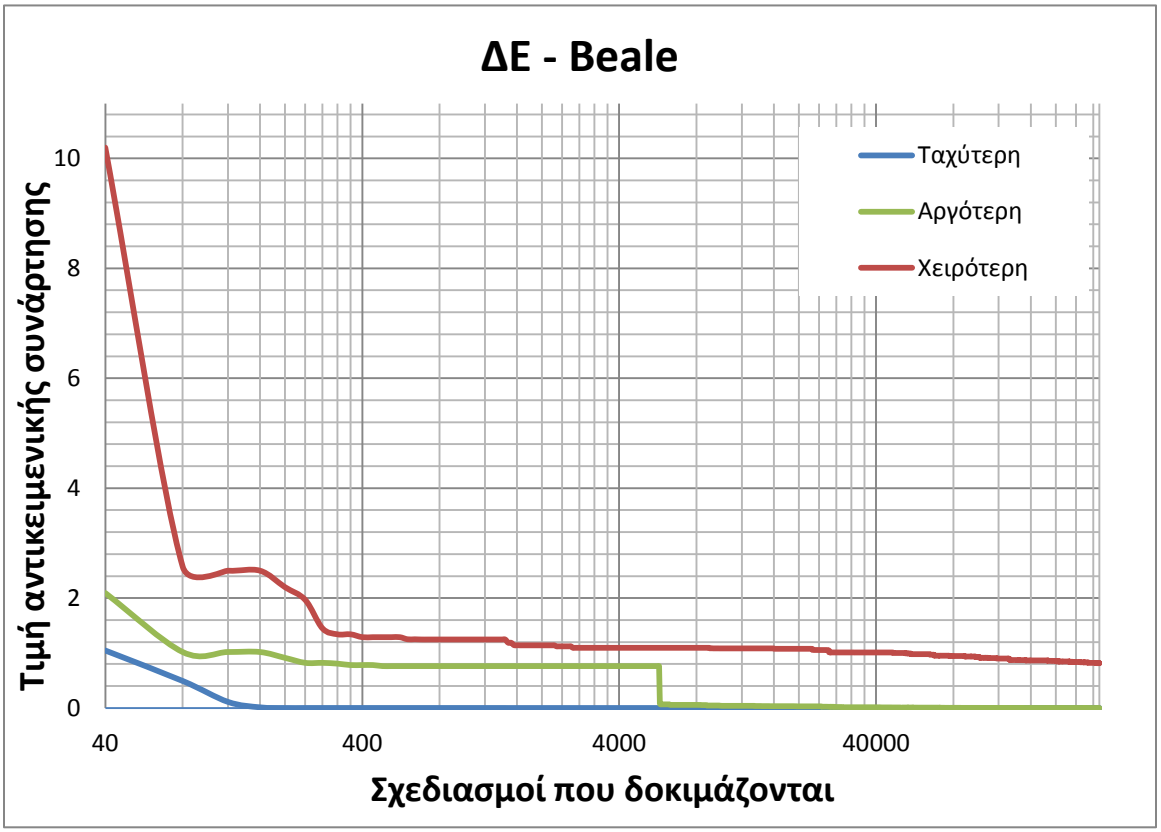
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR
40	best/1/bin	0.2	0.9

Πίνακας 3.29 Beale, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
200	295960	10736.5	1.1%

Πίνακας 3.30 Beale, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[3.000073, 0.500019]$	$8.778 * 10^{-10}$
Αργότερη επίλυση	$[3.017656, 0.504346]$	$4.89325 * 10^{-5}$
Χειρότερη επίλυση	$[-4.269853, 1.196846]$	0.7787722



Σχήμα 3.14 Beale, ΔΕ: Διάγραμμα σύγκλισης.

3.3.4.3 ΣΕ

Πίνακας 3.31 Beale, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

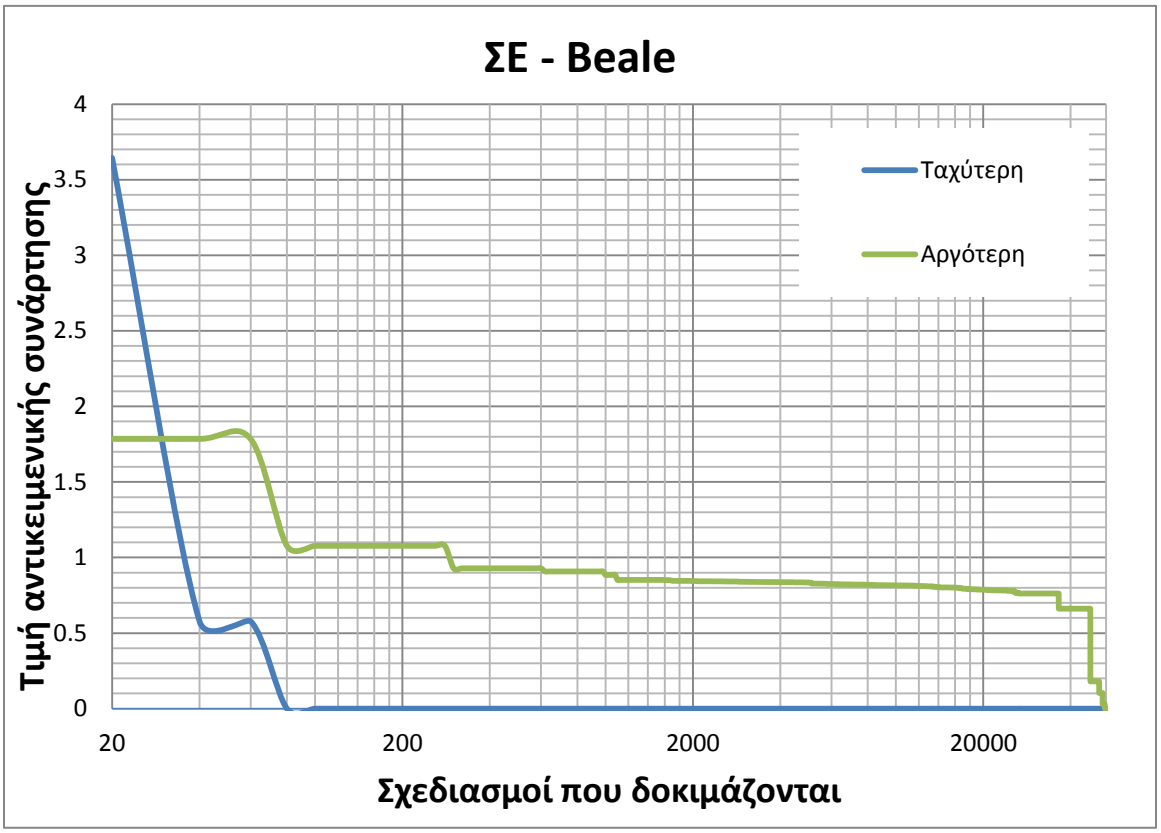
Επιλογή	Ανασυνδυασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ
μ+λ	$1/2 * (x_{a,i} + x_{rand,i})$	ισοτροπική	αυτοπροσαρμογή	20	20

Πίνακας 3.32 Beale, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
80	52840	970.86	0%

Πίνακας 3.33 Beale, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[2.99999999, 0.49999999]$	$8.293 * 10^{-11}$
Αργότερη επίλυση	$[3.00, 0.50]$	$7.311 * 10^{-11}$
Χειρότερη επίλυση	-	-



Σχήμα 3.15 Beale, ΣΕ: Διάγραμμα σύγκλισης.

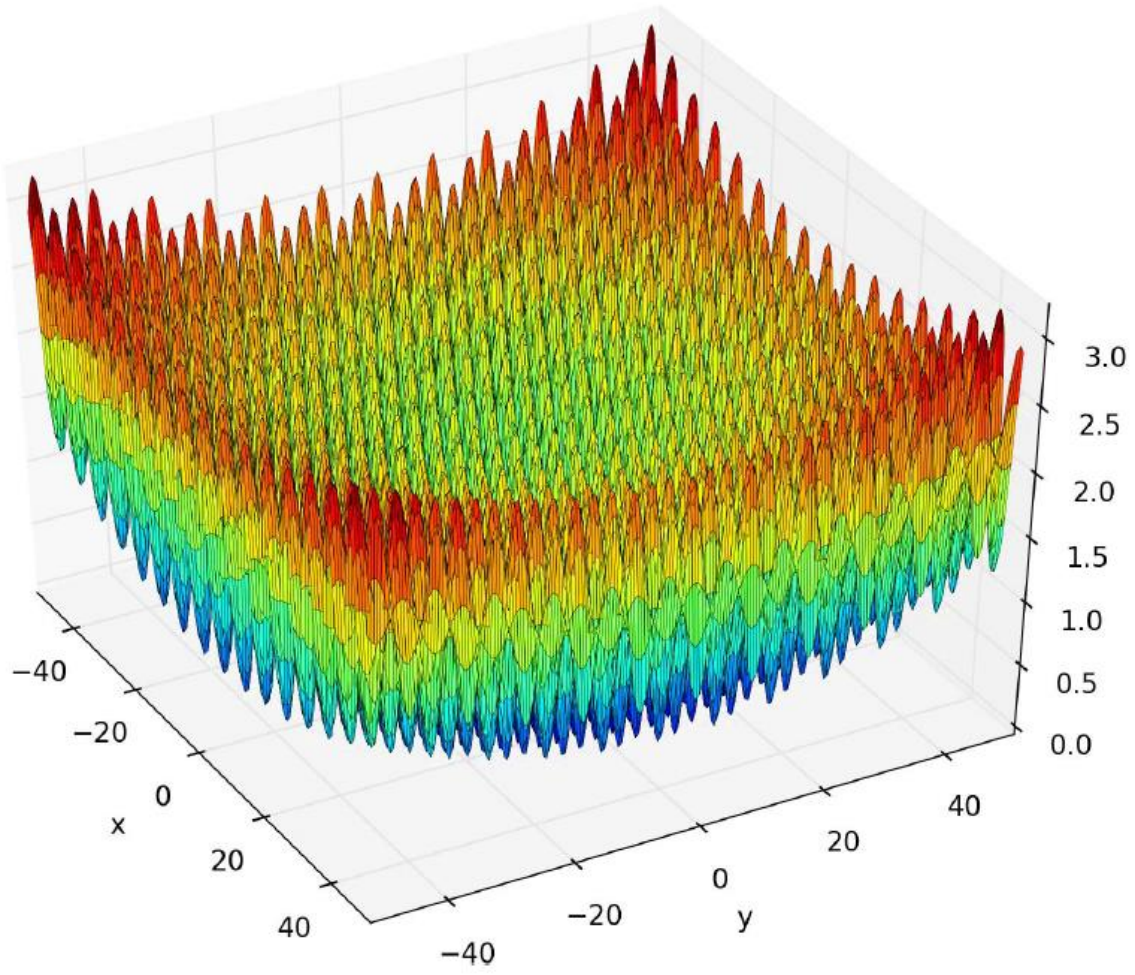
3.3.5 Συνάρτηση Griewank

Η συνάρτηση Griewank είναι συνεχής, πολυτροπική και παρουσιάζει πολλά τοπικά ελάχιστα, που είναι ομοιόμορφα κατανεμημένα.

$$f(\boldsymbol{x}) = 1 + \sum_{i=1}^{n}\left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^{n}\left(\cos\frac{x_i}{\sqrt{i}}\right), \qquad i = 1, 2, \dots, n \qquad \textbf{(3.7)}$$

Ολικό ελάχιστο: $f(\boldsymbol{x}^*) = 0$ στο $\boldsymbol{x}^* = [0, 0, \dots, 0]^T$

Επιλέγονται: *10* μεταβλητές σχεδιασμού στα διαστήματα: $-100 \leq x_i \leq +100$



Σχήμα 3.16 Η συνάρτηση Griewank για δύο μεταβλητές σχεδιασμού.

3.3.5.1 ΒΣΣ

Πίνακας 3.34 Griewank, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

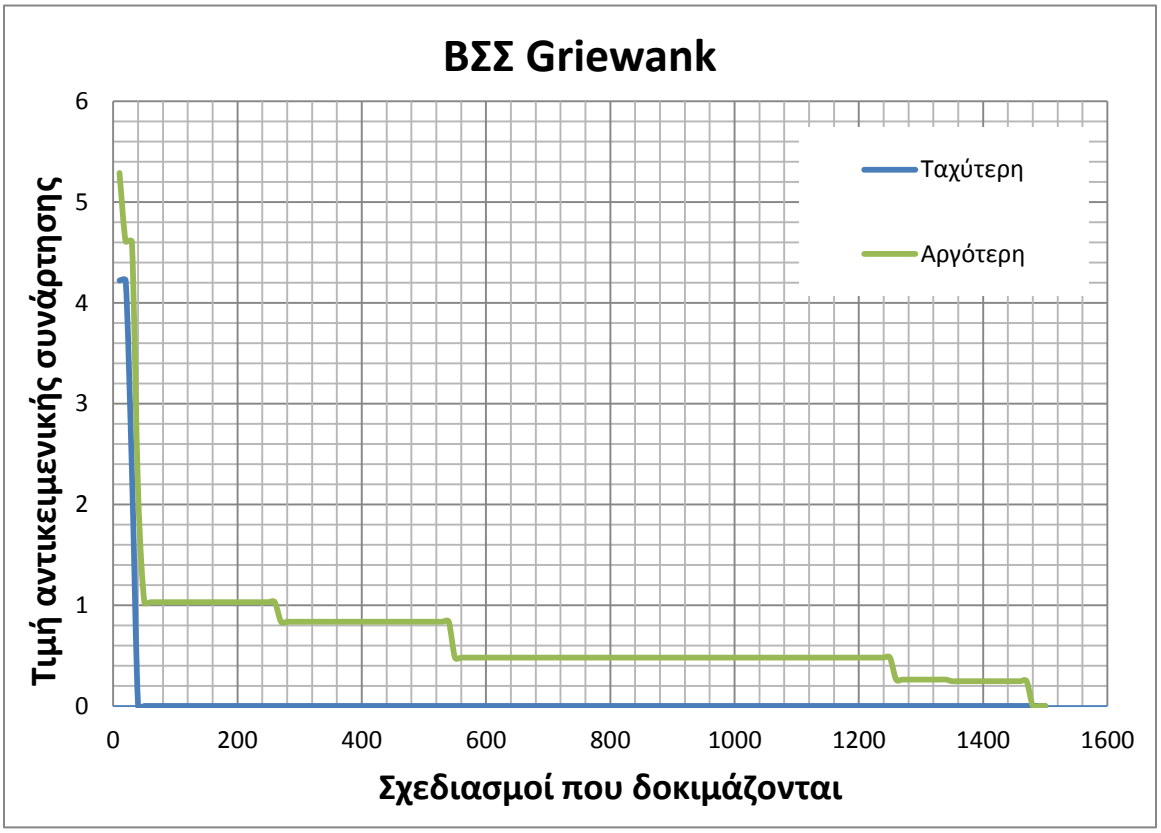
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}
10	Gbest	2.0	2.0	1.0	1.0	$(u_i - l_i)/2$

Πίνακας 3.35 Griewank, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
40	1480	303.58	0%

Πίνακας 3.36 Griewank, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[0, 0, \dots, 0]$	0.0
Αργότερη επίλυση	$[0, 0, \dots, 0]$	0.0
Χειρότερη επίλυση	–	-



Σχήμα 3.17 Griewank, ΒΣΣ: Διάγραμμα σύγκλισης.

3.3.5.2 ΔΕ

Πίνακας 3.37 Griewank, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

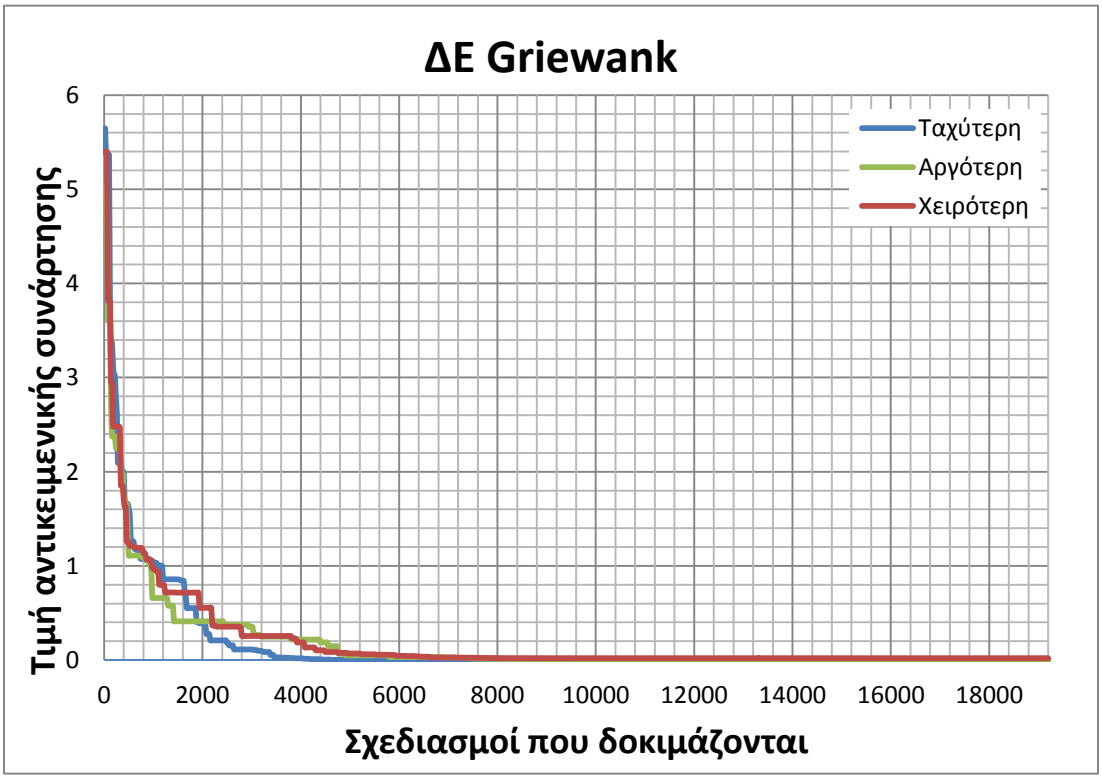
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR
20	rand/1/bin	0.5	0.3

Πίνακας 3.38 Griewank, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
5400	19140	9002.35	7.4%

Πίνακας 3.39 Griewank, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$10^{-9} * [6.6, -3.2, -5.2, -3.4, 11.9, -21.2, -21.2, 17.7, 24.8, -10.3]$	0.0
Αργότερη επίλυση	$10^{-9} * [-3334496.4, 8.7, 12.4, -5.0, -3.5, 11.6, -8.6, 6.2, 7.9, 0.98]$	$5.5622 * 10^{-6}$
Χειρότερη επίλυση	$10^{-8} * [314002255, 0.05, -1.4, 1.4, -1.3, 1.6, 828288217.4, 0.46, -0.13, -1.5]$	0.0196777346



Σχήμα 3.18 Griewank, ΔΕ: Διάγραμμα σύγκλισης.

3.3.5.3 ΣΕ

Πίνακας 3.40 Griewank, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

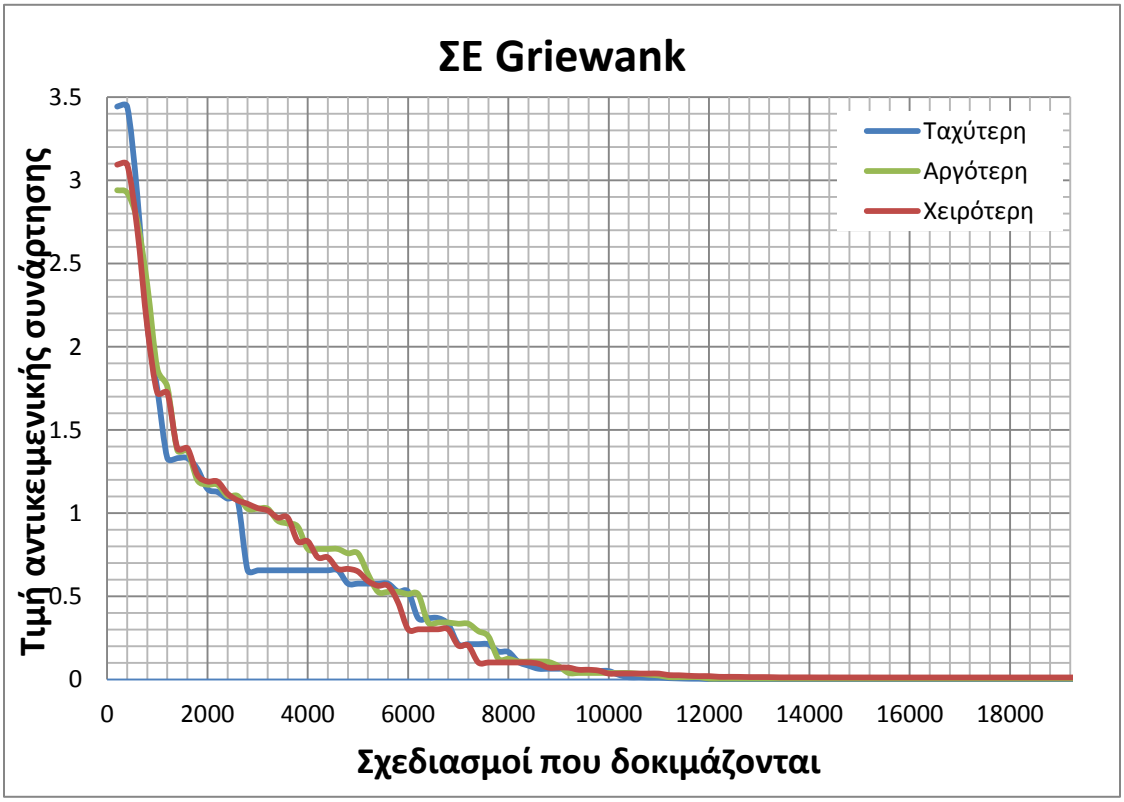
Επιλογή	Ανασυνδυασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ
μ+λ	$x_{rand,i}$	ισοτροπική	αυτοπροσαρμογή	200	200

Πίνακας 3.41 Griewank, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
14000	224000	18277.65	11.4%

Πίνακας 3.42 Griewank, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$10^{-5} - 1.37, -2.55, -2.30, 1.04, -2.43, 2.71, -0.83, 1.39, 7.30, -5.35]$	$9.405 * 10^{-10}$
Αργότερη επίλυση	$10^{-5}[-0.16, 1.34, -3.75, -0.91, 0.45, -0.52, 5.68, -0.38, -2.67, -0.41]$	$5.704 * 10^{-10}$
Χειρότερη επίλυση	$10^{-5}[0.46, -443847.30, 543326.07, 2.46, -1.92, -0.18, 1.07, 1.79, -1.42, 7.93]$	0.01232



Σχήμα 3.19 Griewank, ΣΕ: Διάγραμμα σύγκλισης.

3.3.6 Συνάρτηση Schwefel 2.26

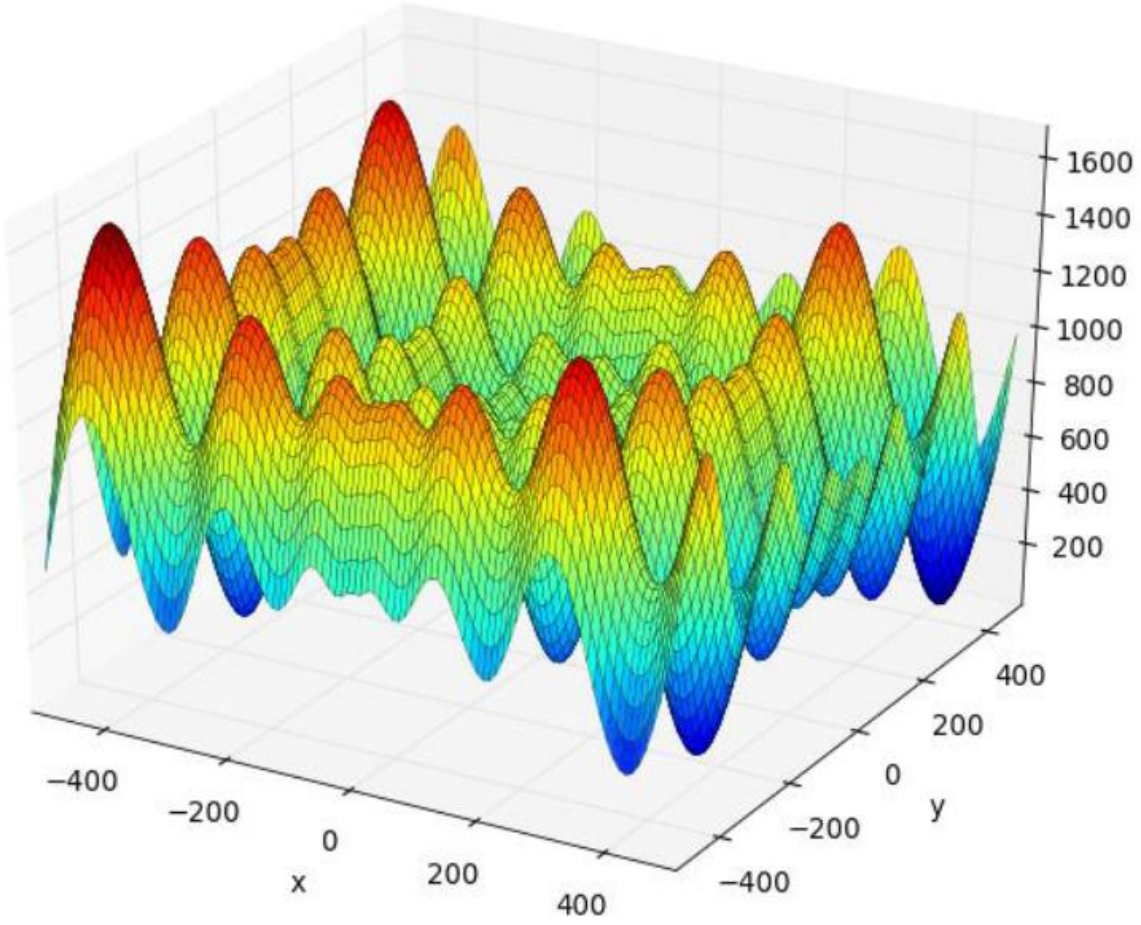
Η συνάρτηση αυτή ανήκει σε μια συλλογή συναρτήσεων benchmark που πρότεινε ο Schwefel. Είναι συνεχής και πολυτροπική.

$$f(\boldsymbol{x}) = -\frac{1}{n}\sum_{i=1}^{n} x_i * \sin\sqrt{|x_i|}\,, \qquad i = 1, 2, \dots, n \tag{3.8}$$

Ολικό ελάχιστο: $f(\boldsymbol{x}^*) = -418.98288727243379980791360 1398$
στο $\boldsymbol{x}^* = [420.968746,\ 420.968746, \dots, 420.968746]^T$

Προβλεπόμενος χώρος σχεδιασμού: $-500 \leq x_i \leq +500$

Επιλέγονται: *10* μεταβλητές σχεδιασμού



Σχήμα 3.20 Η συνάρτηση Schwefel 2.26 για 2 μεταβλητές σχεδιασμού.

3.3.6.1 ΒΣΣ

Πίνακας 3.43 Schwefel 2.26, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

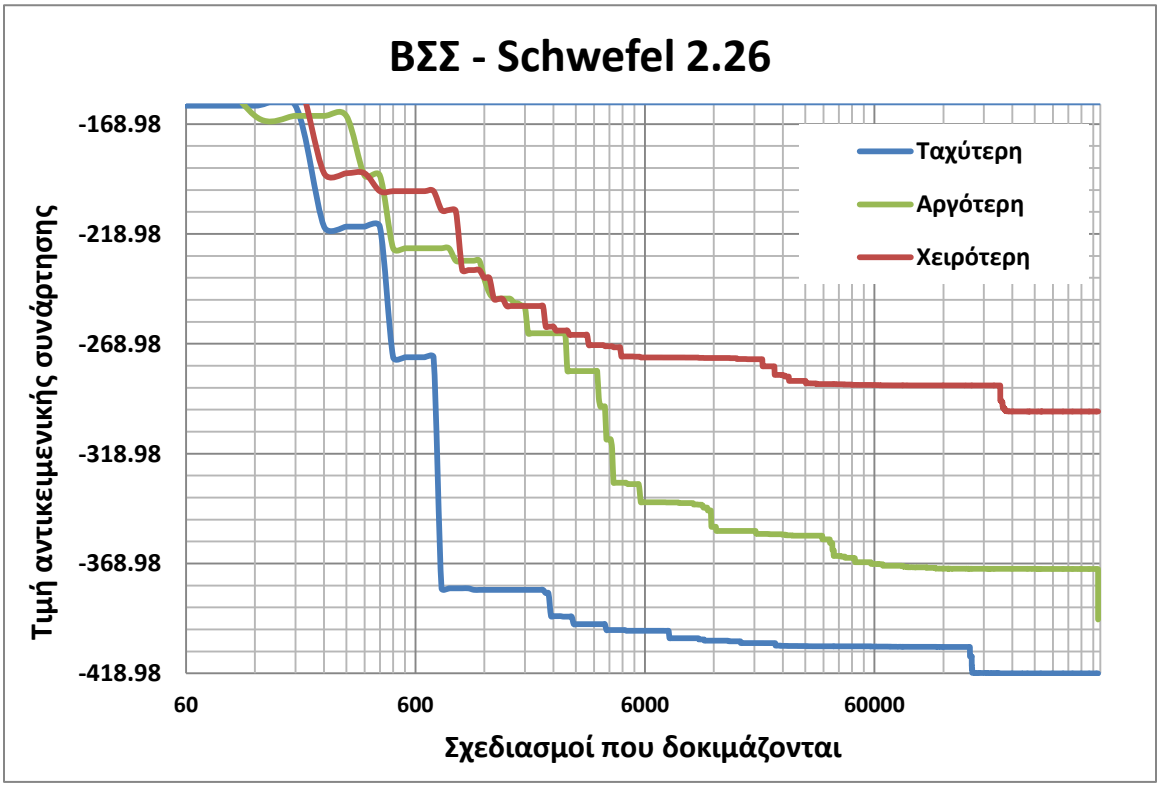
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}
60	Gbest	3.0	1.0	0.9	0.9	$(u_i - l_i)/2$

Πίνακας 3.44 Schwefel 2.26, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
780	566880	82990.7	26.5%

Πίνακας 3.45 Schwefel 2.26, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	[420.6926, 420.3627, 421.2874, 420.8808, 421.4258, 421.2716, 420.5510, 420.9447, 421.4455, 420.8287]	−418.966793
Αργότερη επίλυση	[420.3161, 421.0915, -500.0000, 420.8960, 420.9268, 420.7486, 420.6801, 420.9185, 421.3063, 421.1804]	−395.134164
Χειρότερη επίλυση	[−500.0, 420.9281, 421.0017, −500.0, −500.0, 420.9380, 420.9731, 420.9993, −500.0, −500.0]	−299.785964



Σχήμα 3.21 Schwefel 2.26, ΒΣΣ: Διάγραμμα σύγκλισης.

3.3.6.2 ΔΕ

Πίνακας 3.46 Schwefel 2.26, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

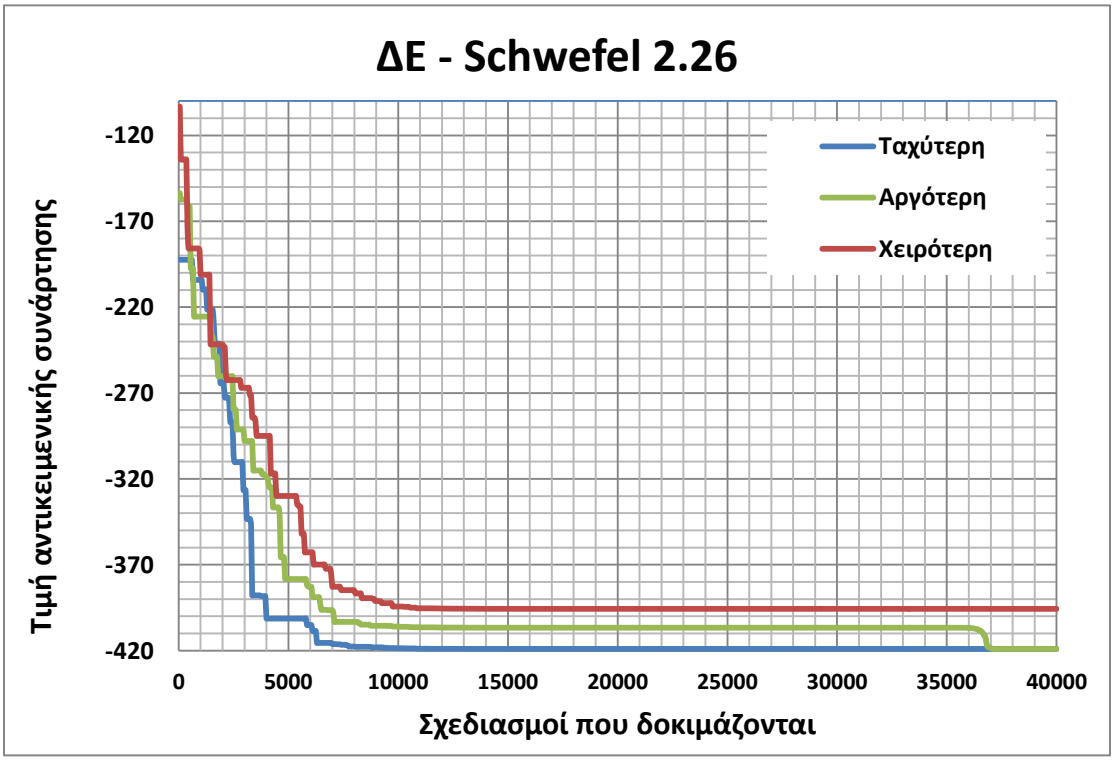
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR
40	best/1/bin	1.0	0.1

Πίνακας 3.47 Schwefel 2.26, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
6300	36800	8443.76	3.8%

Πίνακας 3.48 Schwefel 2.26, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	[420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687]	−418.982887
Αργότερη επίλυση	[420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687]	−418.982887
Χειρότερη επίλυση	[420.9687, 420.9687, 420.9687, 389.8503, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, −302.5249]	−395.659003



Σχήμα 3.22 Schwefel 2.26, ΔΕ: Διάγραμμα σύγκλισης.

3.3.6.3 ΣΕ

Πίνακας 3.49 Schwefel, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

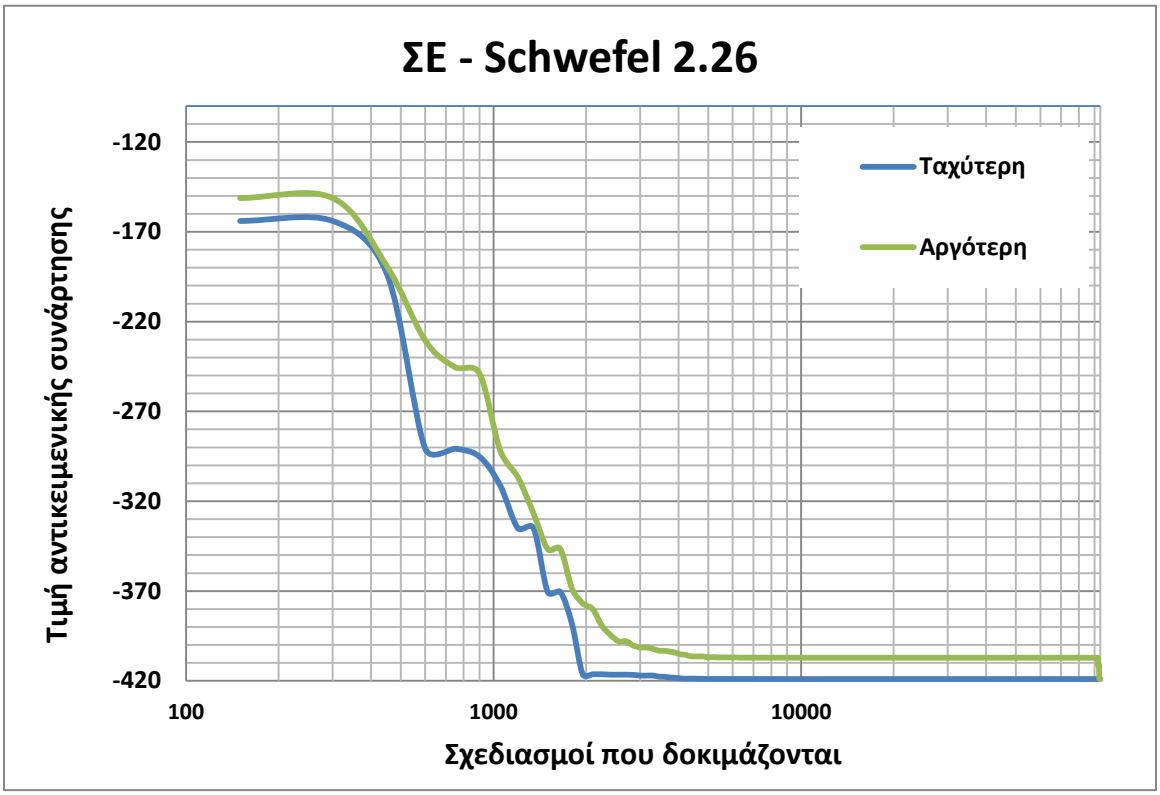
Επιλογή	Ανασυνδυασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ
μ+λ	$x_{rand,i}$	ανισοτροπική	αυτοπροσαρμογή	150	150

Πίνακας 3.50 Schwefel, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
1950	93300	3638.85	0%

Πίνακας 3.51 Schwefel, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	[420.9685, 420.9685, 420.9687, 420.9686, 420.9688, 420.9686, 420.9685, 420.9688, 420.9688, 420.9687]	−418.982887
Αργότερη επίλυση	[420.9687, 420.9688, 420.9688, 420.9687, 420.9687, 420.9688, 420.9687, 420.9688, 420.9687, 420.9688]	−418.982887
Χειρότερη επίλυση	-	-



Σχήμα 3.23 Schwefel, ΣΕ: Διάγραμμα σύγκλισης.

3.4 Προβλήματα αξιολόγησης με περιορισμούς

Για προβλήματα υπό περιορισμούς, δεν υπάρχει τόσο μεγάλη ποικιλία benchmarks. Από αυτά που υπάρχουν, πολύ λίγα τυγχάνουν ευρείας εφαρμογής. Βέβαια, πολλές πρακτικές εφαρμογές μπορούν να χρησιμοποιηθούν στη θέση τους, αφού τα πραγματικά προβλήματα βελτιστοποίησης (οικονομικών, μηχανικής, …) έχουν σχεδόν πάντα περιορισμούς. Παρακάτω αναφέρονται μερικά «μαθηματικά» benchmarks από τους Koziel & Michalewicz και ένα μηχανικής από τον Coello.

Σε κάθε πρόβλημα σημειώνεται και ο λόγος των εφικτών σχεδιασμών προς τους συνολικούς σχεδιασμούς που δοκιμάζονται τυχαία κατά την έναρξη της διαδικασίας. Λόγος 1.0 σημαίνει ότι όλοι οι αρχικοί σχεδιασμοί είναι εφικτοί. Τουλάχιστον ένας αρχικός σχεδιασμός πρέπει να είναι εφικτός.

Για την αντιμετώπιση των περιορισμών χρησιμοποιείται η γραμμική κατά κλάδους συνάρτηση ποινής της ενότητας **2.2.3**. Η θανατική ποινή δοκιμάστηκε στην μεθόδο ΣΕ, αλλά δεν έδωσε καλά αποτελέσματα.

3.4.1 Πρόβλημα G1

Αντικειμενική συνάρτηση:

$$G_1(\boldsymbol{x}) = 5 * \sum_{i=1}^{4} \left(x_i - x_i^2\right) - \sum_{i=5}^{13} x_i \qquad \textbf{(3.9)}$$

Χώρος σχεδιασμού:

$$\begin{aligned} 0 \le x_i \le 1, \qquad & i = 1, 2, \dots, 9, 13 \\ 0 \le x_i \le 100, \qquad & i = 10, 11, 12 \end{aligned} \qquad \textbf{(3.10)}$$

Περιορισμοί:

$$\begin{aligned} 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0 \\ 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0 \\ 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0 \\ -8x_1 + x_{10} \le 0 \\ -8x_2 + x_{11} \le 0 \\ -8x_3 + x_{12} \le 0 \\ -2x_4 - x_5 + x_{10} \le 0 \\ -2x_6 - x_7 + x_{11} \le 0 \\ -2x_8 - x_9 + x_{12} \le 0 \end{aligned} \qquad \textbf{(3.11)}$$

Το πρόβλημα έχει 13 μεταβλητές σχεδιασμού και 9 γραμμικούς περιορισμούς. Η αντικειμενική συνάρτηση είναι τετραγωνική. Παρουσιάζει ολικό ελάχιστο $G_1(\boldsymbol{x}^*) = -15$ στο

$\boldsymbol{x}^* = [1,1,1,1,1,1,1,1,1,3,3,3,1]^T$. Στο ολικό μέγιστο οι πρώτοι τρεις και οι τελευταίοι τρεις περιορισμοί είναι ενεργοί.

3.4.1.1 ΒΣΣ

Πίνακας 3.52 G1, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

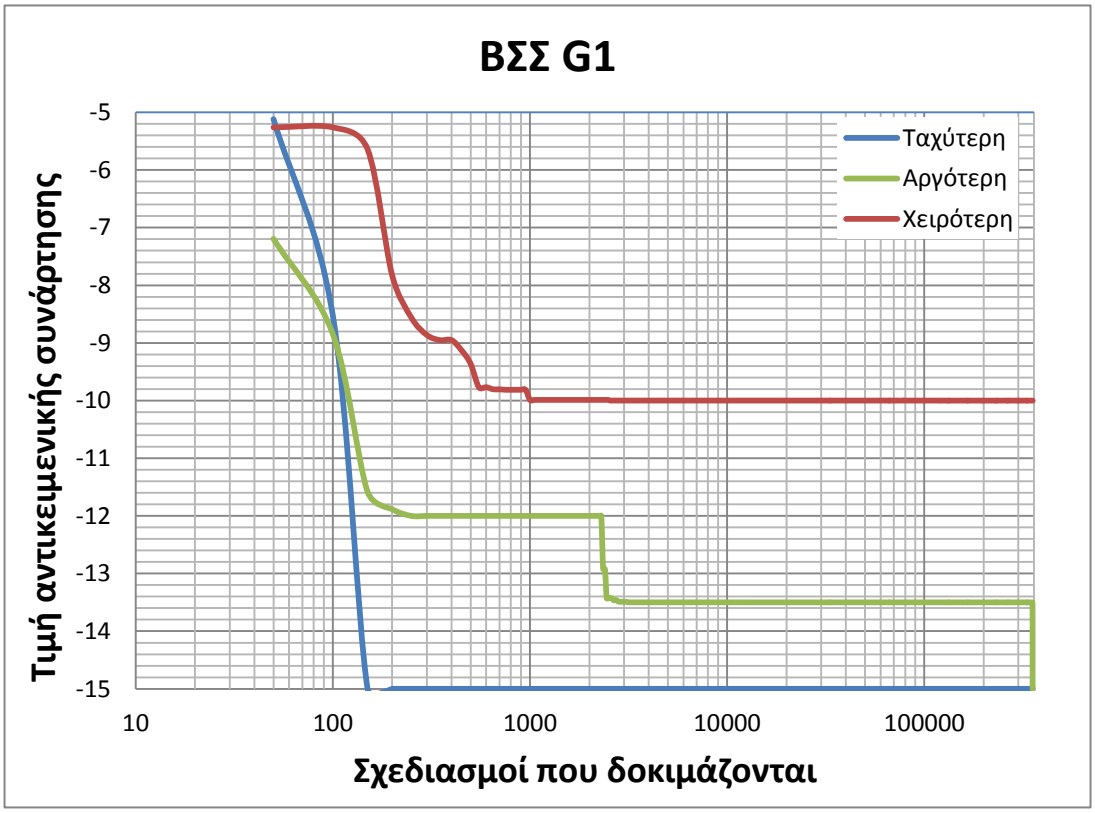
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
50	Gbest	2.0	2.0	0.4	1.6	$(u_i - l_i)/2$	1.0

Πίνακας 3.53 G1, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
150	354400	1139.3	20.6%

Πίνακας 3.54 G1, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1,1,1,1,1,1,1,1,1,3,3,3,1]$	-15.0
Αργότερη επίλυση	$[1,1,1,1,1,1,1,1,1,3,3,3,1]$	-15.0
Χειρότερη επίλυση	$[1,1,0,0,1,1,1,1,1,0.999999999,3,0,1]$	-10.0



Σχήμα 3.24 G1, ΒΣΣ: Διάγραμμα σύγκλισης.

3.4.1.2 ΔΕ

Πίνακας 3.55 G1, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

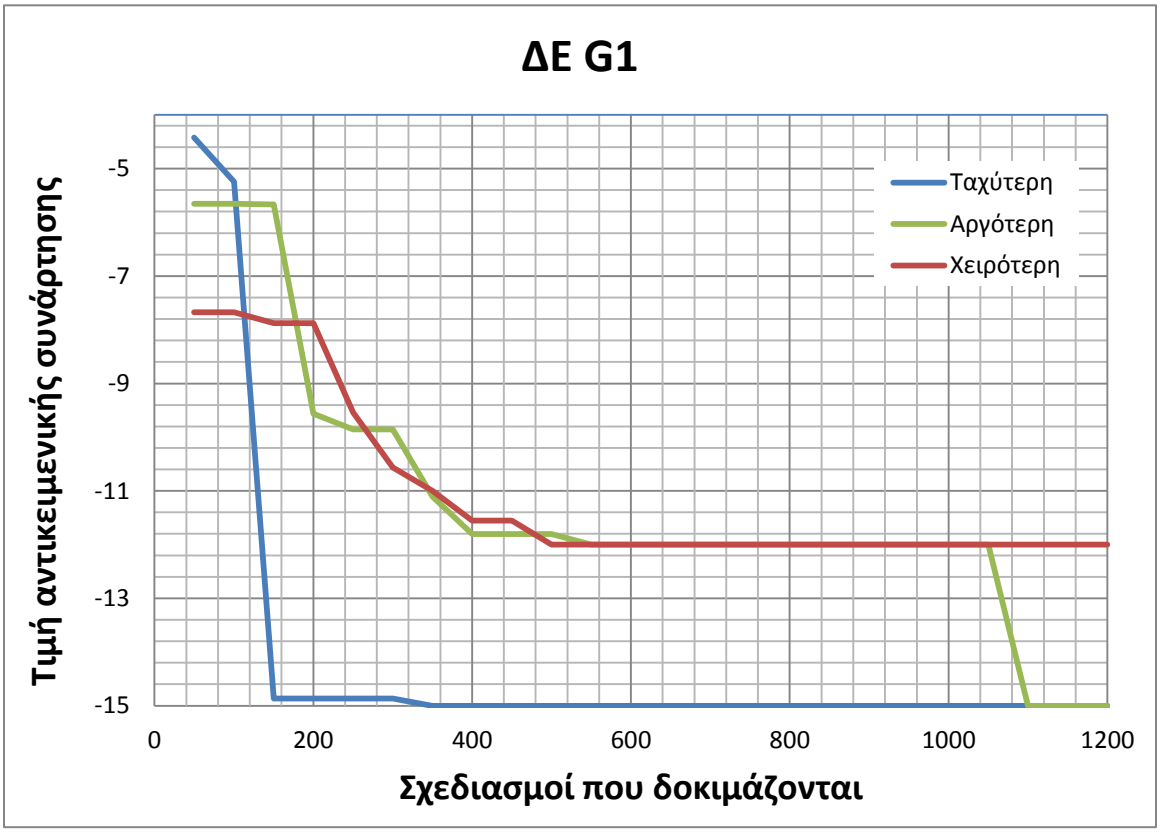
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
20	best/1/bin	1.5	0.9	1.0

Πίνακας 3.56 G1, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
150	1100	504.59	5.3%

Πίνακας 3.57 G1, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1]$	-15.0
Αργότερη επίλυση	$[1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1]$	-15.0
Χειρότερη επίλυση	$[1, 0, 1, 1, 1, 1, 1, 1, 1, 3, 0, 3, 1]$	-12.0



Σχήμα 3.25 G1, ΔΕ: Διάγραμμα σύγκλισης.

3.4.1.3 ΕΣ

Πίνακας 3.58 G1, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

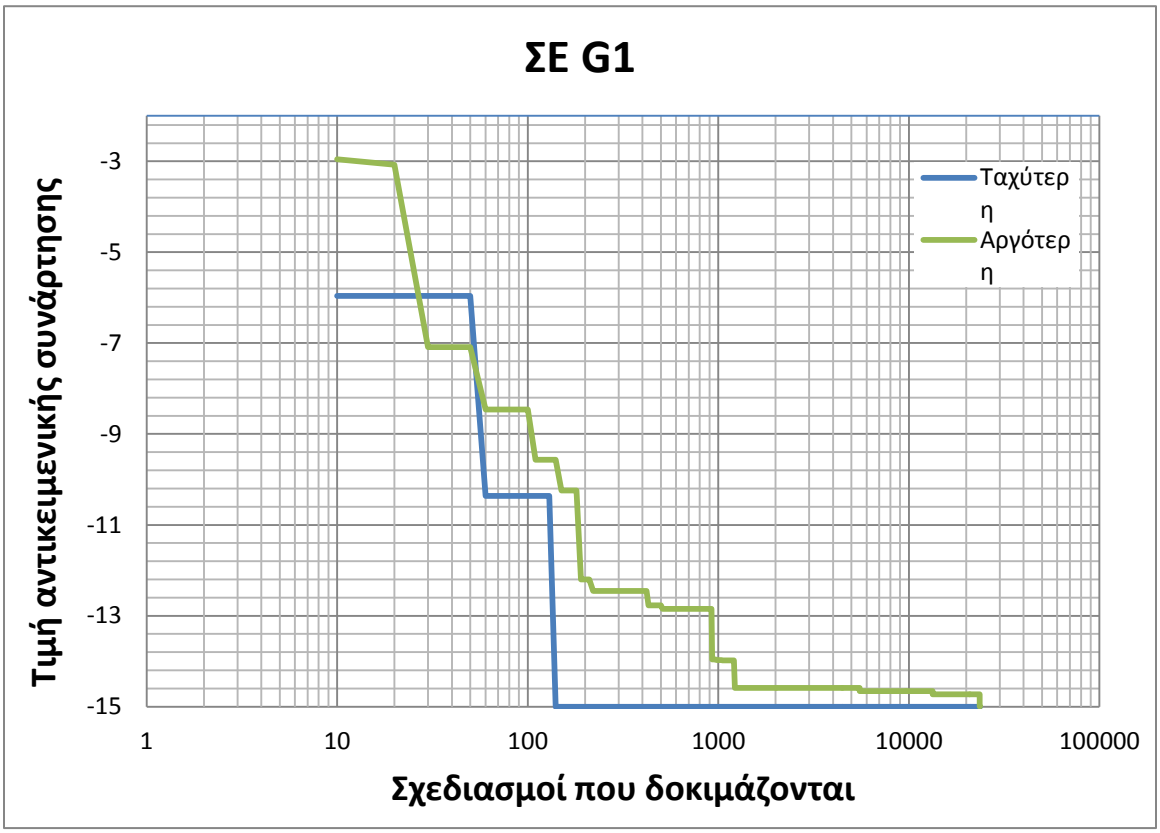
Επιλογή	Ανασυνδυ-ασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ	$\frac{\text{εφικτοί σχεδιασμοί}}{\text{σύνολο αρχικών σχεδιασμών}}$
μ+λ	$x_{rand,i}$	ανισο-τροπική	αυτο-προσαρμογή	10	10	0.5

Πίνακας 3.59 G1, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
140	23530	2966.78	0%

Πίνακας 3.60 G1, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1]$	-15.0
Αργότερη επίλυση	$[1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1]$	-15.0
Χειρότερη επίλυση	-	-



Σχήμα 3.26 G1, ΣΕ: Διάγραμμα σύγκλισης.

3.4.2 Πρόβλημα G6

Αντικειμενική συνάρτηση:

$$G_6(\boldsymbol{x}) = (x_1 - 10)^3 + (x_2 - 20)^3 \quad \textbf{(3.12)}$$

Χώρος σχεδιασμού:

$$\begin{gathered} 13 \leq x_1 \leq 100 \\ 0 \leq x_2 \leq 100 \end{gathered} \quad \textbf{(3.13)}$$

Περιορισμοί:

$$\begin{gathered} -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\ (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0 \end{gathered} \quad \textbf{(3.14)}$$

Το πρόβλημα έχει 3 μεταβλητές σχεδιασμού και 2 μη γραμμικούς περιορισμούς. Η αντικειμενική συνάρτηση είναι επίσης μη γραμμική και παρουσιάζει ολικό ελάχιστο $G_6(\boldsymbol{x}^*) = -6961.81381$ στο $\boldsymbol{x}^* = [14.095, 0.84296]^T$. Στο ολικό μέγιστο και οι δύο περιορισμοί είναι ενεργοί.

3.4.2.1 ΒΣΣ

Πίνακας 3.61 G6, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

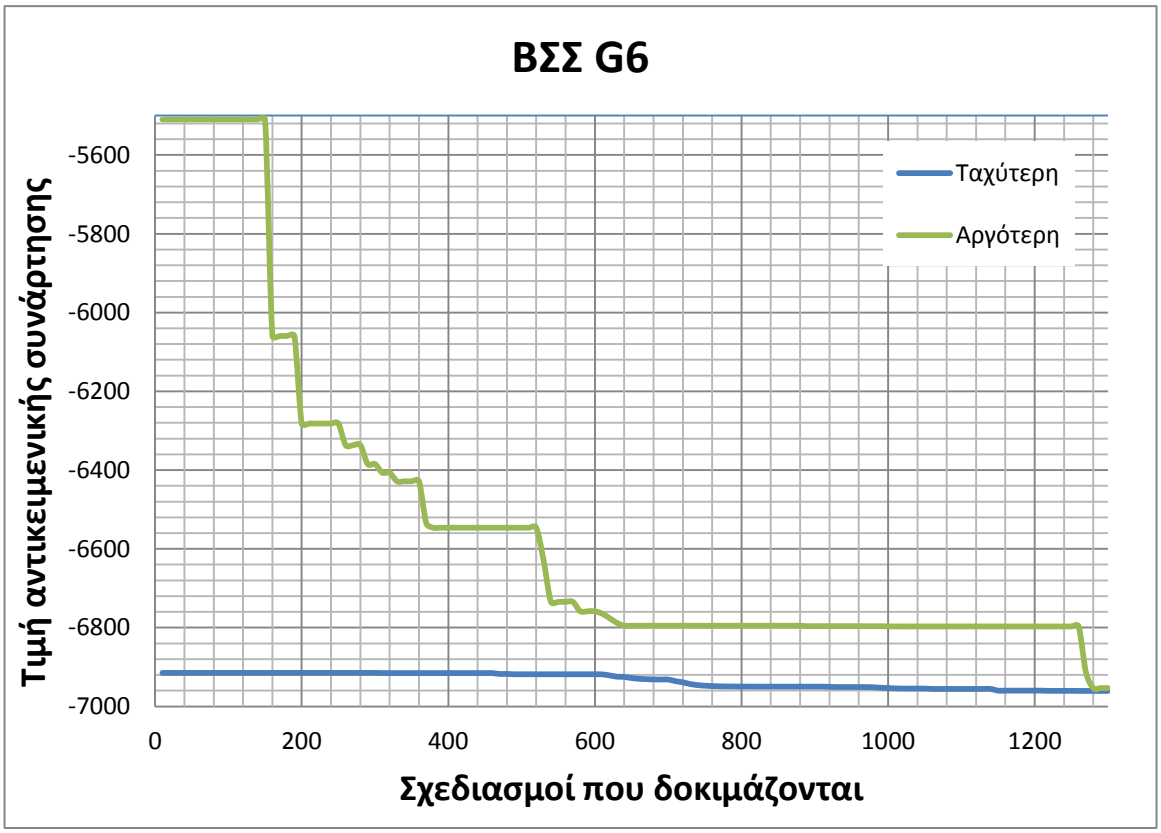
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
10	Gbest	2.0	2.0	0.4	0.6	$(u_i - l_i)/2$	1.0

Πίνακας 3.62 G6, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
10	1270	589.3	0%

Πίνακας 3.63 G6, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[14.095, 0.8429608]$	-6961.81387558
Αργότερη επίλυση	$[14.095, 0.8429608]$	-6961.81387558
Χειρότερη επίλυση	–	–



Σχήμα 3.27 G6, ΒΣΣ: Διάγραμμα σύγκλισης.

3.4.2.2 ΔΕ

Πίνακας 3.64 G6, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

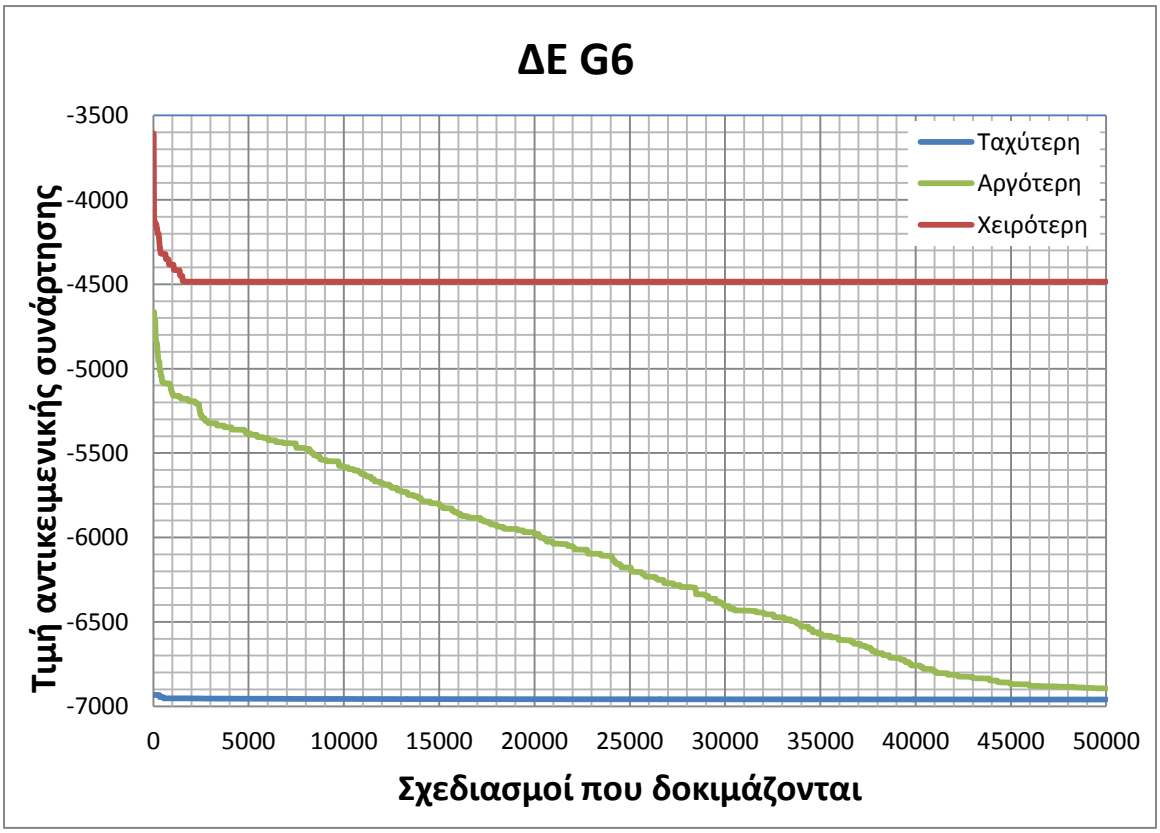
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
20	best/1/bin	0.01	1	0.5

Πίνακας 3.65 G6, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
20	49280	17067.97	1.1%

Πίνακας 3.66 G6, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[14.095160, 0.8433]$	-6961.432685
Αργότερη επίλυση	$[14.095130, 0.843236]$	-6961.5048685
Χειρότερη επίλυση	$[14.948508, 3.362008]$	-4484.5977283



Σχήμα 3.28 G6, ΔΕ: Διάγραμμα σύγκλισης.

3.4.2.3 ΣΕ

Πίνακας 3.67 G6, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

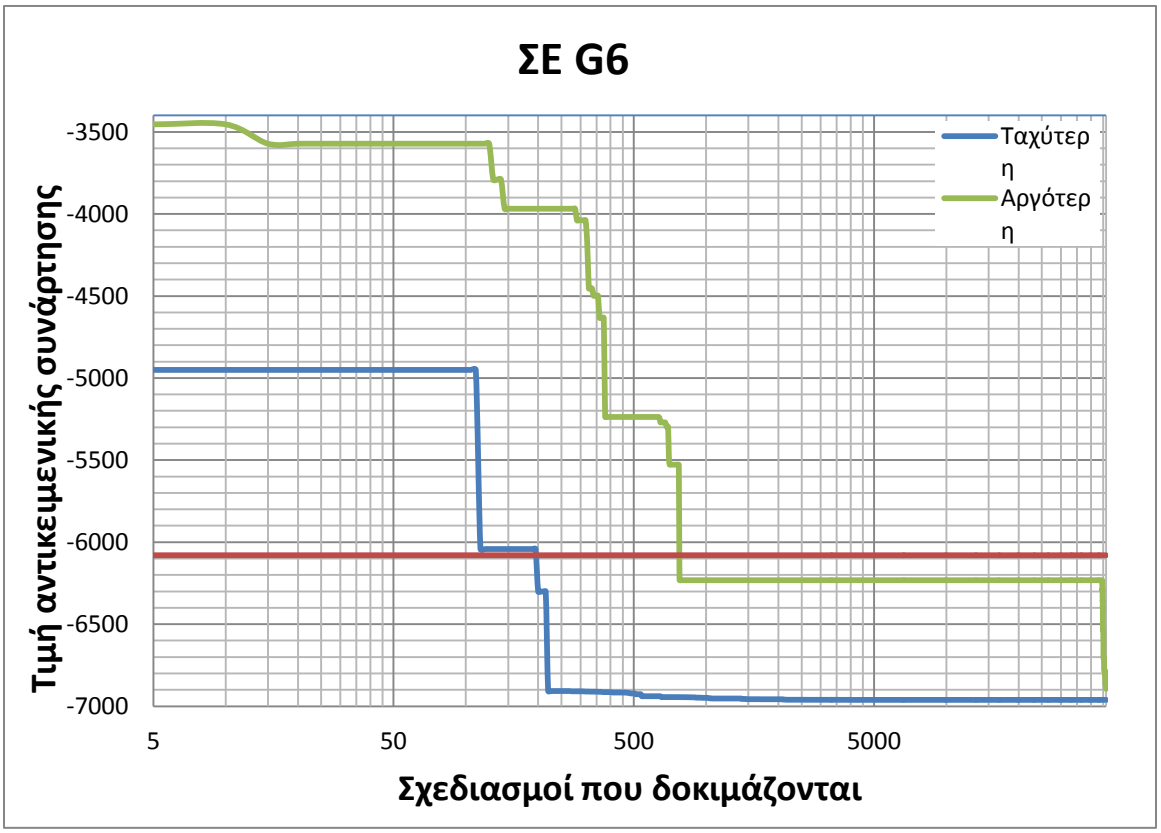
Επιλογή	Ανασυνδυασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ	$\frac{\text{εφικτοί σχεδιασμοί}}{\text{σύνολο αρχικών σχεδιασμών}}$
μ+λ	$1/2 * (x_{a,i} + x_{b,i})$	ισο-τροπική	αυτο-προσαρμογή	5	5	0.5

Πίνακας 3.68 G6, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
220	46285	2661.9	0.2%

Πίνακας 3.69 G6, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[14.095245, 0.843440]$	-6961.2737536
Αργότερη επίλυση	$[14.095498, 0.843959]$	-6960.6903556
Χειρότερη επίλυση	$[14.444309, 1.661097]$	-6079.8711930



Σχήμα 3.29 G6, ΣΕ: Διάγραμμα σύγκλισης.

3.4.3 Πρόβλημα G8

Αντικειμενική συνάρτηση:

$$G_8(\boldsymbol{x}) = -\frac{sin^3(2\pi x_1) * sin(2\pi x_2)}{x_1^3(x_1 + x_2)} \quad \textbf{(3.15)}$$

Χώρος σχεδιασμού:

$$\begin{matrix} 0 \leq x_1 \leq 10 \\ 0 \leq x_2 \leq 10 \end{matrix} \quad \textbf{(3.16)}$$

Περιορισμοί:

$$\begin{matrix} x_1^2 - x_2 + 1 \leq 0 \\ 1 - x_1 + (x_2 - 4)^2 \leq 0 \end{matrix} \quad \textbf{(3.17)}$$

Η συνάρτηση G_8 παρουσιάζει πολλά τοπικά ελάχιστα, τα χαμηλότερα από τα οποία βρίσκονται κοντά στον άξονα Χ. Στην εφικτή περιοχή το ελάχιστο είναι $G_8(\boldsymbol{x}^*) = -0.095825$ στη θέση $\boldsymbol{x}^* = [1.2279713, 4.2453733]^T$

3.4.3.1 ΒΣΣ

Πίνακας 3.70 G8, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

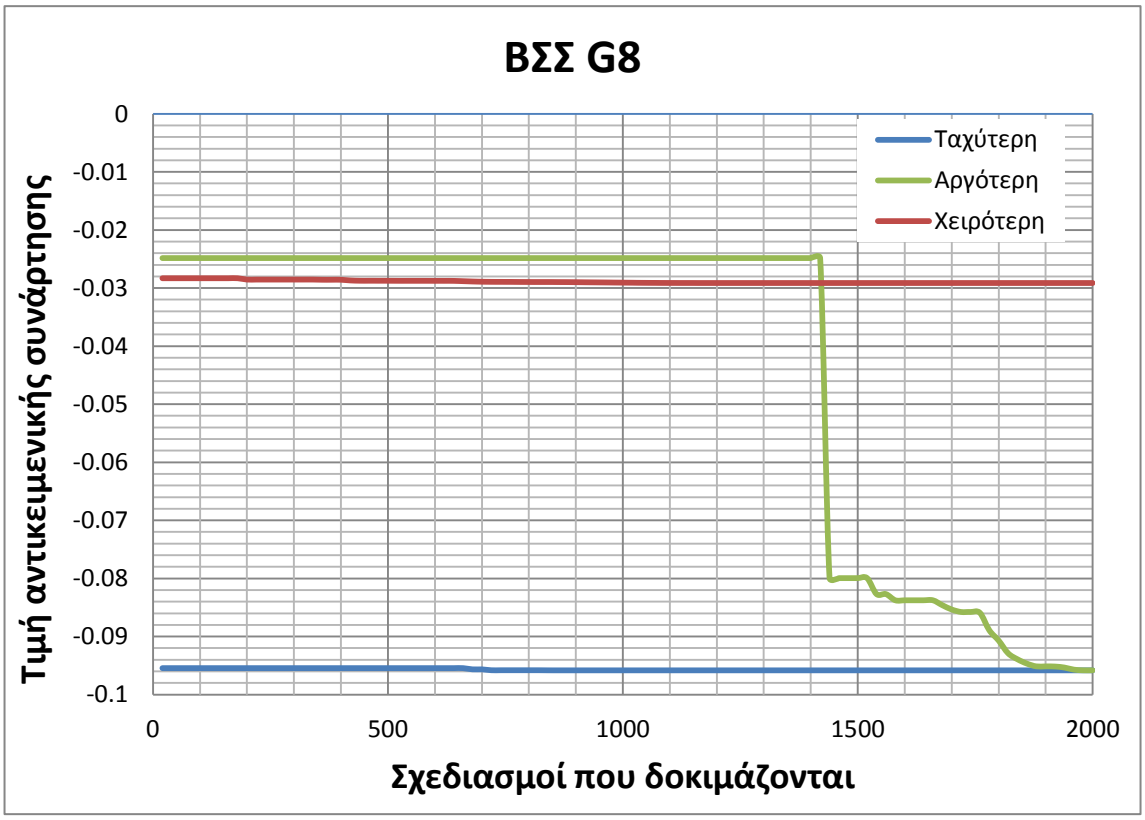
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
20	Gbest	2.0	2.0	0.4	0.6	$(u_i - l_i)/2$	0.5

Πίνακας 3.71 G8, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
20	1880	421	1%

Πίνακας 3.72 G8, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1.22797, 4.24537]$	-0.095825041418
Αργότερη επίλυση	$[1.22797, 4.24537]$	-0.095825041418
Χειρότερη επίλυση	$[1.73618, 4.74503]$	-0.029136002517



Σχήμα 3.30 G8, ΒΣΣ: Διάγραμμα σύγκλισης.

3.4.3.2 ΔΕ

Πίνακας 3.73 G8, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

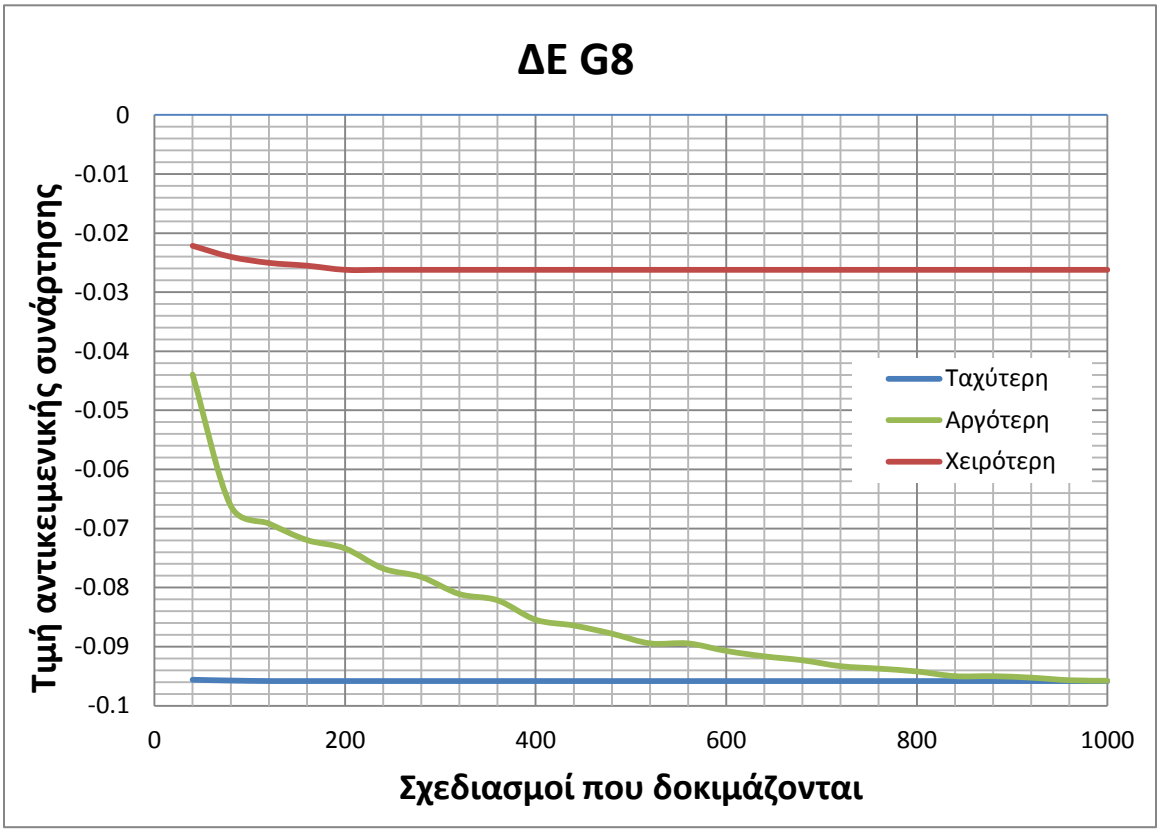
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
40	best/1/bin	0.01	0.9	1.0

Πίνακας 3.74 G8, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
40	840	230.56	1.3%

Πίνακας 3.75 G8, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1.227969, 4.245373]$	-0.09582504
Αργότερη επίλυση	$[1.227971, 4.245373]$	-0.09582504
Χειρότερη επίλυση	$[1.697918, 4.709862]$	-0.02621435



Σχήμα 3.31 G8, ΔΕ: Διάγραμμα σύγκλισης.

3.4.3.3 ΣΕ

Πίνακας 3.76 G8, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

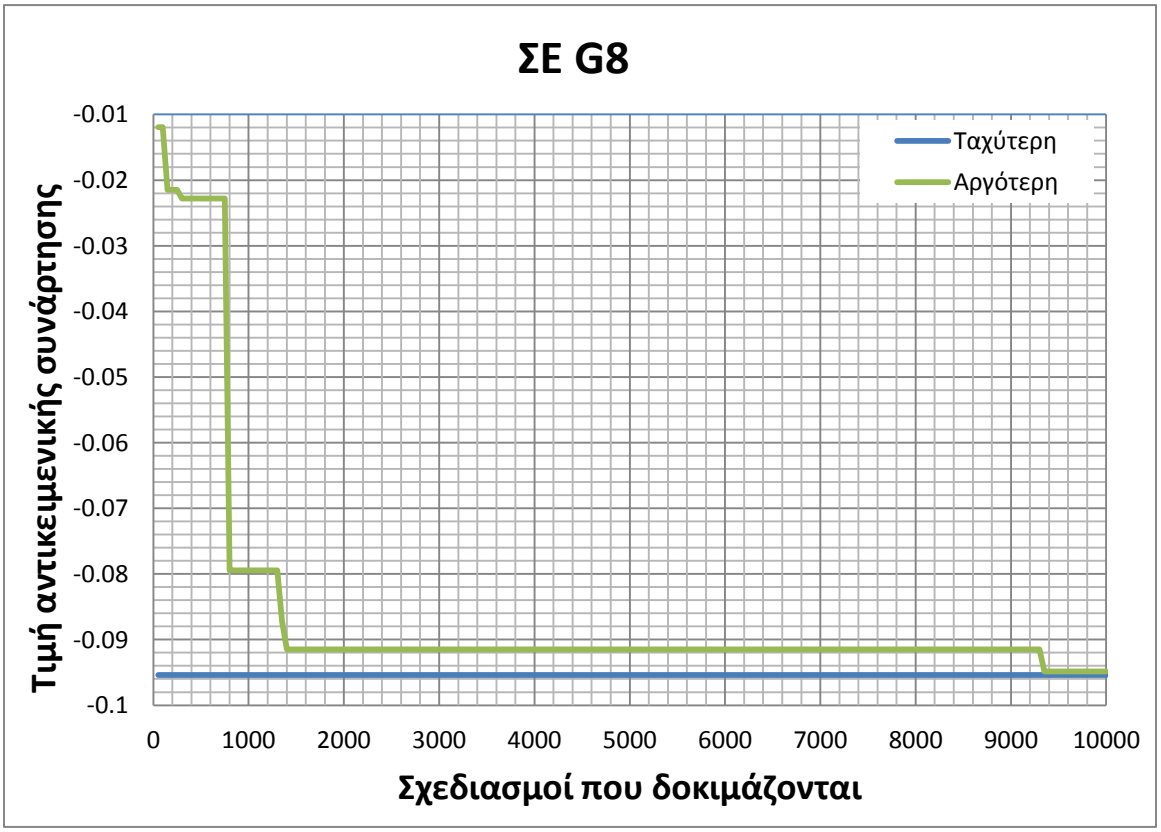
Επιλογή	Ανασυνδυ-ασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ	$\frac{\text{εφικτοί σχεδιασμοί}}{\text{σύνολο αρχικών σχεδιασμών}}$
μ+λ	$1/2 * (x_{a,i} + x_{b,i})$	ισο-τροπική	αυτο-προσαρμογή	10	50	1.0

Πίνακας 3.77 G8, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
50	257450	25731.55	0%

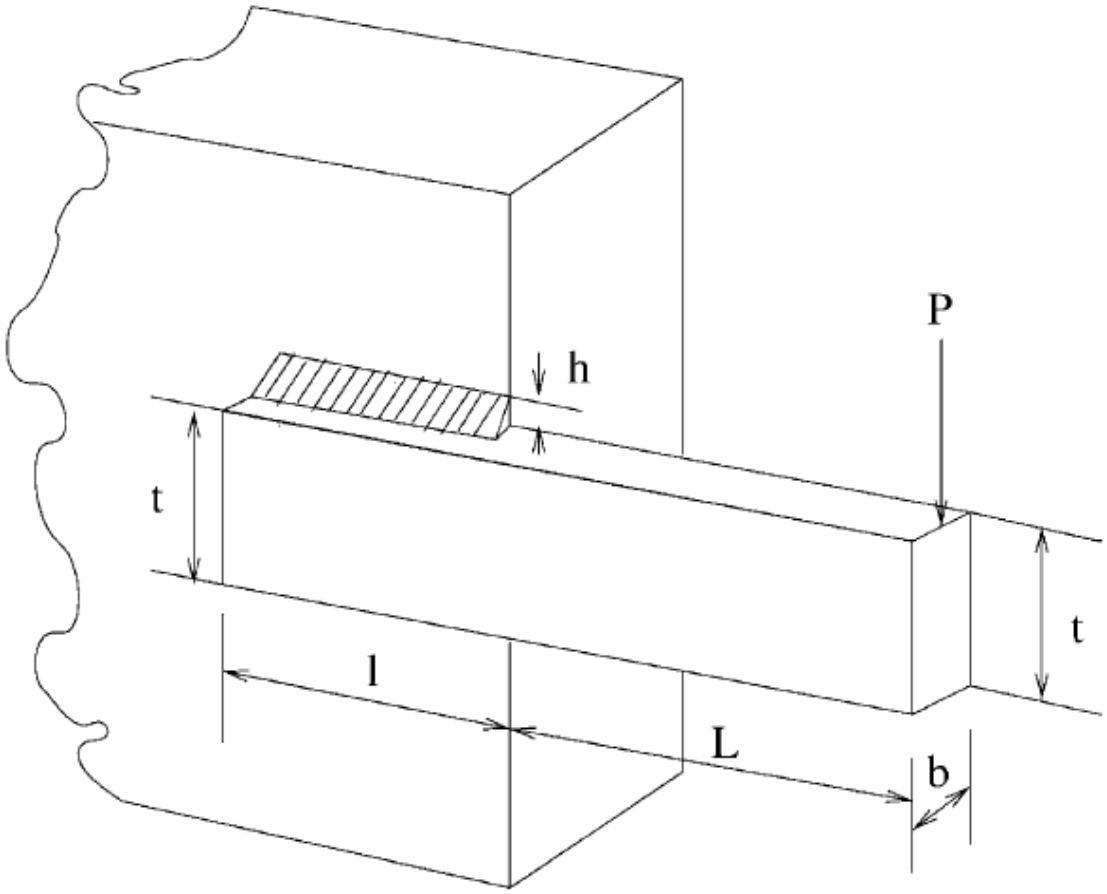
Πίνακας 3.78 G8, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	$[1.228474, 4.249683]$	-0.09578849
Αργότερη επίλυση	$[1.227836, 4.240902]$	-0.09578711
Χειρότερη επίλυση	–	–



Σχήμα 3.32 G8, ΣΕ: Διάγραμμα σύγκλισης.

3.4.4 Πρόβλημα σχεδιασμού συγκολλητής δοκού



Σχήμα 3.33 Συγκολλητή δοκός

Οι μεταβλητές σχεδιασμού είναι:

$$\begin{gathered} x_1 = h, \quad x_2 = l, \quad x_3 = t, \quad x_4 = b \\ x_i \geq 0.125, \quad i = 1,4 \\ x_i > 0, \quad i = 2,3 \end{gathered} \qquad \textbf{(3.18)}$$

Η αντικειμενική συνάρτηση εκφράζει το συνολικό κόστος της δοκού και της συγκόλλησης:

$$f(\boldsymbol{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (L + x_2) \qquad \textbf{(3.19)}$$

Οι περιορισμοί αφορούν τις διατμητικές τάσεις (τ), τις ορθές τάσεις από κάμψη (σ), το φορτίο λυγισμού (P_b), την βύθιση του άκρου (δ), και κάποιες κατασκευαστικές διατάξεις:

$$\begin{gathered}\tau(\boldsymbol{x}) \le \tau_{max} \\ \sigma(\boldsymbol{x}) \le \sigma_{max} \\ P_b(\boldsymbol{x}) \ge P \\ \delta(\boldsymbol{x}) \le \delta_{max} \\ x_1 - x_4 \le 0 \\ 0.10471 x_1^2 + 0.04811 x_3 x_4 (L + x_2) - 5.0 \le 0\end{gathered} \quad \textbf{(3.20)}$$

Όπου οι πρέπει να υπολογιστούν πρώτα οι ενδιάμεσες ποσότητες:

$$\begin{gathered}\tau(\boldsymbol{x}) = \sqrt{(\tau')^2 + 2\tau'\tau" \frac{x_2}{2R} + (\tau")^2} \\ \tau' = \frac{P}{\sqrt{2} x_1 x_2}, \qquad \tau" = \frac{MR}{J}, \qquad M = P\left(L + \frac{x_2}{2}\right) \\ R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \\ J = 2\left\{\sqrt{2} x_1 x_2 \left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} \\ \sigma(\boldsymbol{x}) = \frac{6PL}{x_4 x_3^2}, \qquad \delta(\boldsymbol{x}) = \frac{3PL^3}{E x_3^3 x_4} \\ P_b(\boldsymbol{x}) = \frac{4.013E}{6L^2} x_3 x_4^3 \left(1 - 0.25 \frac{x_3}{L}\sqrt{\frac{E}{G}}\right) \\ P = 6000\ lb, \qquad L = 14\ in \\ E = 30 * 10^6\ psi, \qquad G = 12 * 10^6\ psi \\ \tau_{max} = 13600\ psi, \qquad \sigma_{max} = 30000 sfhpsi, \\ \delta_{max} = 0.25 in\end{gathered} \quad \textbf{(3.21)}$$

Η βέλτιστη λύση των Coello, Mondes είναι $f(\boldsymbol{x}^*) = 1.728226$,
στο $\boldsymbol{x}^* = [0.205986,\ 3.471328,\ 9.020224,\ 0.206480]^T$. Πιθανόν όμως να υπάρχουν και καλύτερες λύσεις.

3.4.4.1 ΒΣΣ

Πίνακας 3.79 Συγκολλητή δοκός, ΒΣΣ: Παράμετροι της μεθόδου βελτιστοποίησης.

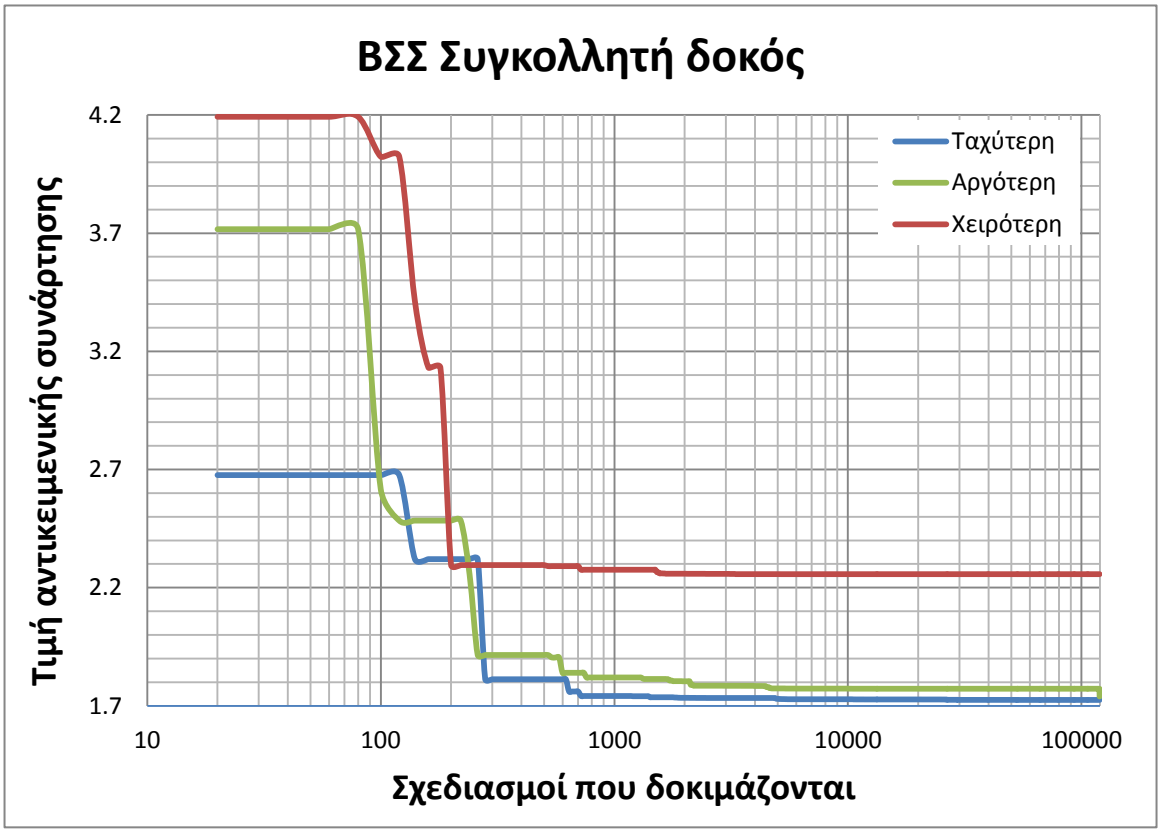
NP	Σμήνος	c_1	c_2	w_{min}	w_{max}	v_i^{max}	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
20	Gbest	2.0	2.0	0.4	0.6	$(u_i - l_i)/2$	0.5

Πίνακας 3.80 Συγκολλητή δοκός, ΒΣΣ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
720	119440	5758.1	2.1%

Πίνακας 3.81 Συγκολλητή δοκός, ΒΣΣ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	[0.205730, 3.470488, 9.036625, 0.205730]	1.724852434
Αργότερη επίλυση	[0.205712, 3.469318, 9.040676, 0.205712]	1.725227930
Χειρότερη επίλυση	[0.1 10.0 9.036624, 0.205730]	2.257063820



Σχήμα 3.34 Συγκολλητή δοκός, ΒΣΣ: Διάγραμμα σύγκλισης.

3.4.4.2 ΔΕ

Πίνακας 3.82 Συγκολλητή δοκός, ΔΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

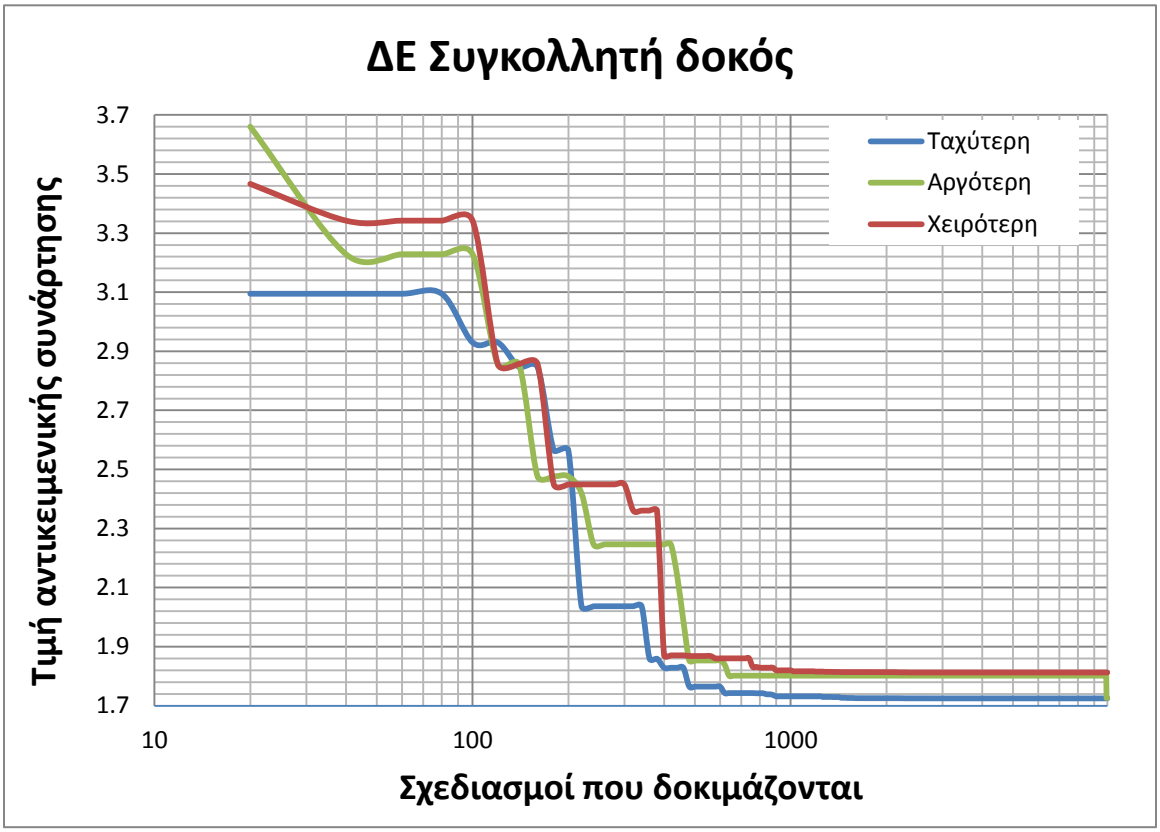
NP	μετάλλαξη, *ανασυνδυασμός*	F	CR	εφικτοί σχεδιασμοί / σύνολο αρχικών σχεδιασμών
20	best/1/bin	0.55	0.9	0.5

Πίνακας 3.83 Συγκολλητή δοκός, ΔΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
620	9840	1819.90	0.1%

Πίνακας 3.84 Συγκολλητή δοκός, ΔΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	[0.205730, 3.470489, 9.036624, 0.205730]	1.7248523
Αργότερη επίλυση	[0.205729, 3.470505, 9.036621, 0.205730]	1.7248541
Χειρότερη επίλυση	[0.201447, 3.236778, 9.980713, 0.201460]	1.8125138



Σχήμα 3.35 Συγκολλητή δοκός, ΔΕ: Διάγραμμα σύγκλισης.

3.4.4.3 ΣΕ

Πίνακας 3.85 Συγκολλητή δοκός, ΣΕ: Παράμετροι της μεθόδου βελτιστοποίησης.

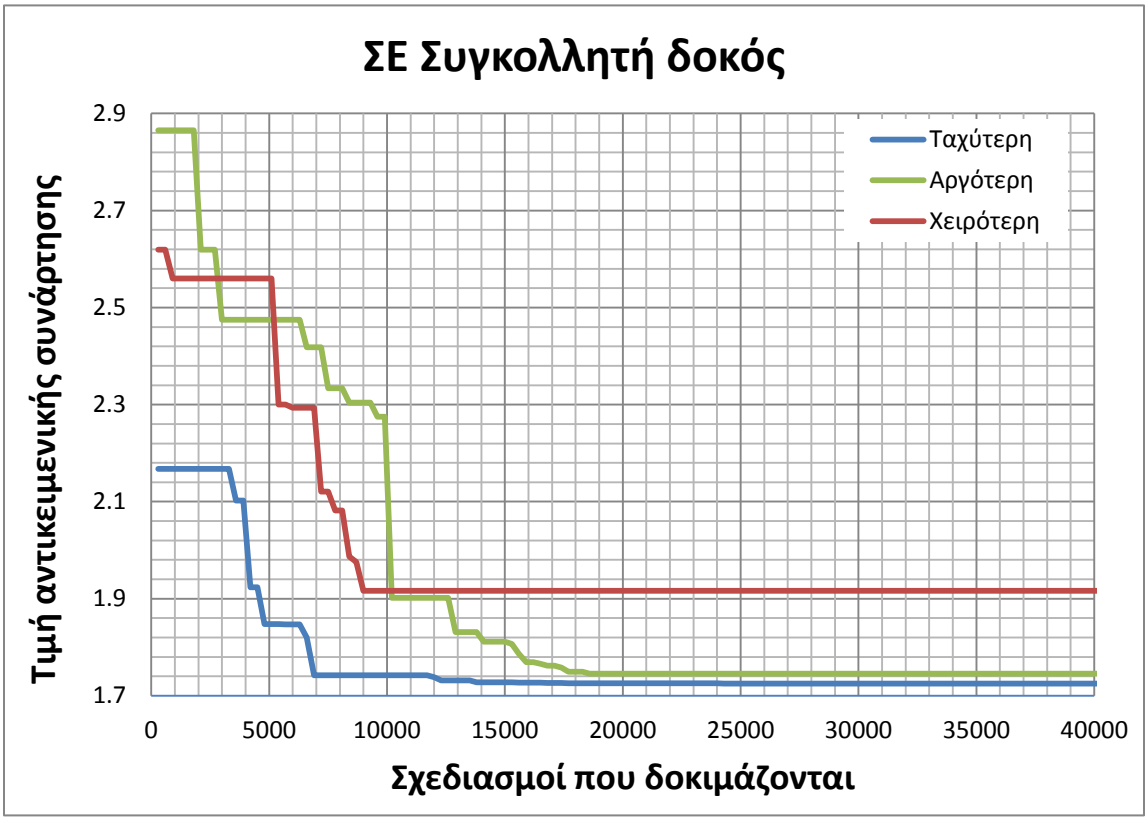
Επιλογή	Ανα-συνδυασμός	Μετάλλαξη	Προσαρμογή βήματος	μ	λ	$\frac{\text{εφικτοί σχεδιασμοί}}{\text{σύνολο αρχικών σχεδιασμών}}$
μ+λ	$x_{a,i}$ ή $x_{b,i}$ $στην\ τύχη$	ανισο-τροπική	αυτο-προσαρμογή	100	300	0.5

Πίνακας 3.86 Συγκολλητή δοκός, ΣΕ: Ταχύτητα σύγκλισης και και ποσοστό αποτυχίας.

Ελάχιστος αριθμός σχεδιασμών	Μέγιστος αριθμός σχεδιασμών	Μέσος αριθμός σχεδιασμών. Δεν προσμετρούνται οι αποτυχίες	Ποσοστό αποτυχημένων εκτελέσεων
6900	2217300	42061.66	10%

Πίνακας 3.87 Συγκολλητή δοκός, ΣΕ: Ταχύτερη, αργότερη και χειρότερη επίλυση.

	Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
Ταχύτερη επίλυση	[0.205698, 3.471017, 9.037531,1 0.205732]	1.72505022
Αργότερη επίλυση	[0.205731, 3.470938, 9.036672, 0.205737]	1.72498184
Χειρότερη επίλυση	[0.227131, 3.844005, 8.696829, 0.227365]	1.91658259



Σχήμα 3.36 Συγκολλητή δοκός, ΣΕ: Διάγραμμα σύγκλισης.

3.5 Συμπεράσματα

Η Βελτιστοποίηση Σμήνους Σωματιδίων συνέκλινε εξαιρετικά γρήγορα όταν το τοπίο παρουσίαζε «μακροσκοπική βύθιση» προς το ολικό ελάχιστο, ακόμα και αν παρεμβάλλονταν πολλά τοπικά ελάχιστα όπως στις Ackley και Griewank. Αντίθετα στη Schwefel 2.26, που τα ακρότατα ήταν τυχαία διασκορπισμένα, παγιδευόταν εύκολα σε τοπικά ελάχιστα. Απαιτήθηκαν ελάχιστες δοκιμές για να βρεθούν καλές παράμετροι στα ευκολότερα benchmarks. Στα δυσκολότερα, όπου έγιναν πολλές δοκιμές φάνηκε ότι κύριος παράγοντας επιτυχίας ήταν η επιλογή του συντελεστή αδράνειας w. Η τοπολογία σμήνους Gbest έδωσε καλύτερα αποτελέσματα ακόμα και σε προβλήματα που ήταν εύκολο να συγκλίνει πρόωρα η μέθοδος, λόγω τοπικών ελαχίστων.

Η Διαφορική Εξέλιξη δεν συνέκλινε το ίδιο γρήγορα στα benchmarks που η ΒΣΣ παρουσίασε τόσο καλή συμπεριφορά. Αντίθετα, παγιδευόταν πολύ δυσκολότερα στα τοπικά ακρότατα της Schwefel 2.26 και ήταν η μόνη μέθοδος που κατόρθωσε να εντοπίσει με ικανοποιητική ταχύτητα το ολικό ελάχιστο, που βρίσκεται εντός της κοιλάδας της Rosenbrock. Η επιλογή καλών τιμών για τις παραμέτρους ήταν γενικά δυσκολότερη από ότι στη ΒΣΣ, γιατί αν και είναι μόνο 4 (προσμετρείται και το είδος της εξίσωση της μετάλλαξης), η αλληλεπίδρασή τους είναι δυσκολότερο να κατανοηθεί από κάποιον που δεν έχει εμπειρία με τη μέθοδο. Επιπλέον, φάνηκε ότι η ύπαρξη μηχανισμού για την προσαρμογή του βήματος της μετάλλαξης, που διαθέτουν η ΒΣΣ και οι ΣΕ είναι πολύ χρήσιμη και λείπει από την ΔΕ.

Οι Στρατηγικές Εξέλιξης με μηχανισμό αυτό-προσαρμογής του βήματος μετάλλαξης κατά Scwhefel παρουσίασαν πολύ καλύτερη συμπεριφορά από τον κανόνα του 1/5 του Rechenberg. Η δυνατότητα κάθε μέλους να προσαρμόζει εξελικτικά το βήμα μετάλλαξης της έδωσε τη δυνατότητα να συγκλίνει γρηγορότερα από την ΔΕ στις μονοτροπικές συναρτήσεις και ξεφεύγει από τα τοπικά ακρότατα της Schwefel 2.26 ευκολότερα από τις δύο άλλες μεθόδους. Ωστόσο, η συνεργασία της με τις συναρτήσεις ποινής στα προβλήματα με περιορισμούς δεν ήταν το ίδιο καλή. Αρχικά δοκιμάστηκε η παραδοσιακή μέθοδος της θανατικής ποινής, αλλά έδωσε ακόμα χειρότερα αποτελέσματα. Στο benchmark του Rosenbrock, που είναι ιδιαίτερα δύσκολο για πολλές μεθόδους βελτιστοποίησης, οι ΣΕ απέτυχαν να βρουν το ολικό ελάχιστο. Πάντως δεν δοκιμάστηκε η ανισότροπη μετάλλαξη με περιστροφή (βλ **2.6.2.2**), που ίσως να επέτρεπε καλύτερη παρακολούθηση του προσανατολισμού της κοιλάδας, εντός της οποίας παγιδεύονταν όλα τα μέλη του πληθυσμού. Η επιλογή παραμέτρων ήταν πολύ εύκολη, καθώς αριθμητικές τιμές απαιτούνταν μόνο για τα μ και λ. Για τα υπόλοιπα, απλά επιλεγόταν κάποια παραλλαγή των τελεστών ανασυνδυασμού, μετάλλαξης και επιλογής. Η επιλογή $(\mu+\lambda)$ συμπεριφέρθηκε πολύ καλύτερα από την (μ,λ) σε όλα τα benchmarks.

Κεφάλαιο 4

4 Κάρτες Γραφικών και Παραλληλία

4.1 Εισαγωγή

Παραδοσιακά ένα πρόγραμμα υπολογιστή γράφεται για σειριακή εκτέλεση στην κεντρική μονάδα επεξεργασίας (Central Processing Unit – CPU) του υπολογιστή. Ο αλγόριθμος εφαρμόζεται ως μια σειρά εντολών που εκτελούνται σειριακά, δηλαδή η μία μετά την άλλη. Σε κάθε χρονική στιγμή εκτελείται μόνο μια εντολή και οι διάφορες θέσεις μνήμης που χρησιμοποιεί το πρόγραμμα διαβάζονται ή γράφονται μόνο από μία εντολή. Η ροή ελέγχου μπορεί να παρουσιάζει διακλαδώσεις και βρόγχους, αλλά η σειρά εκτέλεσης τους είναι προβλέψιμη και συνήθως ντετερμινιστική.

Από τη δεκαετία του 1980, που ξεκίνησαν να εμφανίζονται οι πρώτοι προσωπικοί υπολογιστές , ως το 2004 η βελτίωση της απόδοσης τους επιτυγχανόταν κυρίως με αύξηση της συχνότητας περιστροφής της κεντρικής μονάδας επεξεργασίας. Έτσι για να βελτιωθεί η απόδοση ενός προγράμματος, αρκούσε η εκτέλεσή του σε καινούργιους υπολογιστές με επεξεργαστές υψηλότερης συχνότητας. Βέβαια η απόδοση εξαρτάται και από την διαθέσιμη μνήμη RAM του συστήματος, αλλά και αυτή αυξανόταν ταχύτατα και συνεχίζει και σήμερα να αυξάνεται. Άλλωστε κάθε πρόγραμμα απαιτεί συνολικά ένα συγκεκριμένο μέγεθος μνήμης. Αύξηση της μνήμης του συστήματος πέρα από το όριο αυτό δεν επηρεάζει την απόδοση του προγράμματος. Επομένως ένα πρόγραμμα δεν χρειαζόταν καμία τροποποίηση προκειμένου να εκμεταλλευτεί πλήρως τις δυνατότητες του καινούργιου υλικού (hardware) και να παραμείνει αρκετά γρήγορο για την κάλυψη των αυξανόμενων απαιτήσεων χρήσης του.

Δυστυχώς η αύξηση της συχνότητας της CPU δεν είναι ανεξέλεγκτη. Η κατανάλωση ενέργειας ενός επεξεργαστή είναι ανάλογη της συχνότητας περιστροφής. Πέρα από το κόστος λειτουργίας, η αυξημένη κατανάλωση ενέργειας έχει ως συνέπεια και την παραγωγή περισσότερης θερμότητας. Έτσι τίθενται περιορισμοί αντοχής των υλικών από τα οποία αποτελούνται τα κυκλώματα των επεξεργαστών. Η αύξηση της συχνότητας της CPU άρχισε να επιβραδύνεται σημαντικά αναγκάζοντας τους παραγωγούς να καταφύγουν σε άλλους τρόπους βελτίωσης της απόδοσης. Η επόμενη επικρατούσα στρατηγική ήταν η χρήση περισσότερων από μία μονάδων επεξεργασίας στην ίδια CPU. Σήμερα σχεδόν όλοι νέοι σταθεροί και φορητοί υπολογιστές έχουν από 2 ως 10 πυρήνες (cores), δηλαδή ανεξάρτητες μονάδες επεξεργασίας τοποθετημένες στο ίδιο κύκλωμα. Η τάση αυτή εξαπλώνεται και στις φορητές συσκευές (π.χ. κινητά τηλέφωνα, music players) παρά τον περιορισμένο χώρο που διαθέτουν. Βέβαια η συχνότητα του κάθε πυρήνα συνεχίζει να αυξάνεται, αλλά πολύ πιο αργά.

Σε έναν υπολογιστή με δύο ή περισσότερους πυρήνες (multi-core), τα διάφορα προγράμματα εκτελούνται παράλληλα και το υπολογιστικό κόστος μοιράζεται στους διάφορους πυρήνες. Επιπλέον ένα πρόγραμμα μπορεί να διαιρεθεί σε ανεξάρτητα τμήματα και να εκτελεστεί παράλληλα, μειώνοντας το συνολικό απαιτούμενο χρόνο. Αυτό όμως απαιτεί την εφαρμογή τεχνικών του παράλληλου προγραμματισμού (parallel programming), που είναι σημαντικά δυσκολότερος, προκειμένου ένα πρόγραμμα να αξιοποιήσει πολλούς πυρήνες. Η παράλληλη χρήση πολλών μονάδων επεξεργασίας εφαρμοζόταν επί δεκαετίες στους υπέρ-υπολογιστές (supercomputers, mainframes) για απαιτητικά λογισμικά (high performance computing). Ωστόσο οι συνηθισμένες εμπορικές εφαρμογές δεν απαιτούν τόσο ακριβό εξοπλισμό, ενώ η δυσκολία ανάπτυξης παράλληλων προγραμμάτων καθιστούσε τον παράλληλο προγραμματισμό μη βιώσιμο οικονομικά. Έτσι οι προσωπικοί υπολογιστές διέθεταν μονοπύρηνους επεξεργαστές και τα λογισμικά που γράφονταν ήταν σειριακά.

Αυτό δε σημαίνει όμως ότι δεν υπάρχει καμία μορφή παραλληλίας. Οι επεξεργαστές των τελευταίων δεκαετιών εκτελούν ταυτόχρονα πολλές λειτουργίες. Προφανώς με έναν μόνο πυρήνα μπορεί να εκτελεστεί μόνο μία εντολή σε κάθε κύκλο του επεξεργαστή, αλλά οι εντολές διαφορετικών προγραμμάτων μπορούν να εναλλάσσονται. Έτσι η CPU εκτελεί κάποιες εντολές μιας διαδικασίας (process) και έπειτα την θέτει σε αναμονή και ασχολείται με μία άλλη, μοιράζοντας το χρόνο λειτουργίας μεταξύ ανεξάρτητων εργασιών (multitasking). Αυτό είναι απαραίτητο, διαφορετικά ο χρήστης θα έπρεπε να περιμένει να τελειώσουν όλα τα υπάρχοντα προγράμματα πριν ξεκινήσει ένα καινούργιο, ενώ αν κάποιο απαιτούσε είσοδο δεδομένων (π.χ. από τον χρήστη), ο επεξεργαστής απλά θα σταματούσε να λειτουργεί, αντί να στρέψει τη προσοχή του στα υπόλοιπα όσο περιμένει. Η ίδια λογική μπορεί να εφαρμοστεί εντός του ίδιου προγράμματος, χωρίζοντάς το σε δύο υπολειτουργίες, η μία από τις οποίες περιμένει κάποιο γεγονός που δεν εξαρτάται από το ίδιο το πρόγραμμα (asynchronous event), όπως κάποια εντολή του χρήστη, ενώ η άλλη συνεχίζει την εκτέλεση υπολογισμών.

Ακόμα και σε συστήματα που η CPU διαθέτει έναν μόνο πυρήνα, υπάρχει άλλος ένας επεξεργαστής: η μονάδα επεξεργασίας γραφικών (Graphics Processing Unit – GPU). Η GPU συνήθως τοποθετείται μαζί με κυκλώματα μνήμης RAM σε μία πλακέτα που ονομάζεται κάρτα γραφικών (video card). Από το 1985 και μετά αρχίζουν να παρασκευάζονται προσωπικοί υπολογιστές που χρησιμοποιούν GPU για τη σχεδίαση των γραφικών στην οθόνη. Μία GPU αποτελεί δευτερεύουσα μονάδα επεξεργασίας,, που λειτουργεί παράλληλα με την CPU και είναι αφιερωμένη στην απαιτητική διεργασία της σχεδίασης γραφικών. Με την εξάπλωση των γραφικών περιβαλλόντων σε προγράμματα, παιχνίδια και λειτουργικά συστήματα, όπως τα Windows, οι κάρτες γραφικών εξελίχθηκαν σε αναπόσπαστο μέρος των προσωπικών υπολογιστών. Σήμερα είναι συχνοί υπολογιστές με περισσότερες από μία κάρτες γραφικών.

Οι GPUs διαθέτουν εκατοντάδες πυρήνες, που είναι όμως απλούστεροι και αργότεροι από τον πυρήνα μιας CPU. Αρχικά σχεδιάστηκαν για απλές αριθμητικές και λογικές πράξεις, που χρειάζονται για τον προσδιορισμό και την αποτύπωση 2D και 3D γραφικών. Η επικοινωνία του

προγραμματιστή με την GPU γινόταν μόνο μέσω εξειδικευμένων προτύπων (Open GL, DirectX). Η GPU εκτελεί πράξεις πάνω στα δεδομένα που έχουν τη μορφή pixels, δηλαδή παριστάνονται ως συντεταγμένες (x,y). Ερευνητές διαπίστωσαν ότι αν μεταμφιέσουν τα δεδομένα ενός οποιουδήποτε, άσχετου με γραφικά, προβλήματος σε συντεταγμένες, τότε η GPU θα τα επεξεργαστεί και θα επιστρέψει αποτελέσματα. Η υπολογιστική ισχύς όλων αυτών των πυρήνων είναι εντυπωσιακή, όμως η μεταμφίεση των δεδομένων σαν στοιχεία γραφικών, η αδυναμία εξελιγμένων χειρισμών της μνήμης και η ανάγκη εκμάθησης των προτύπων που προορίζονται για την περιγραφή γραφικών, εμπόδιζαν την εφαρμογή της τεχνικής αυτής σε πρακτικά προβλήματα.

Το 2007 η NVidia κυκλοφόρησε την πρώτη κάρτα γραφικών με την αρχιτεκτονική CUDA που επιτρέπει τον έλεγχο των πυρήνων και της μνήμης της GPU. Η CUDA περιλαμβάνει επίσης και μια γλώσσα προγραμματισμού βασισμένη στην C, την C CUDA, ώστε ο προγραμματιστής να μην χρειάζεται να μεταμφιέζει τα δεδομένα του σαν γραφικά. Έτσι είναι δυνατόν να χρησιμοποιηθεί η GPU για γενικές εφαρμογές προγραμματισμού (General-purpose computing on graphics processing units – GPGPU). Σύντομα, παρόμοιες αρχιτεκτονικές αναπτύχθηκαν από τις ανταγωνιστικές εταιρίες της NVidia, την ATI (ATI Stream) και την Intel που παράγει κάρτες γραφικών ενσωματωμένες στο κύκλωμα του επεξεργαστή. Το 2009 ο όμιλος Khronos Group ανέπτυξε το OpenCL (Open Computing Language), ένα πλαίσιο για την επικοινωνία με CPU, GPU και άλλα είδη επεξεργαστών ανεξαρτήτως εταιρίας παραγωγής.

Οι GPUs διαθέτουν περισσότερους πυρήνες από τις CPU με αποτέλεσμα να μπορούν να εκτελούν υπολογισμούς με πολύ μεγαλύτερη ταχύτητα. Επιπλέον οι προσπελάσεις μνήμης από την GPU είναι ταχύτερες. Έτσι, πολλά προγράμματα μπορούν να ωφεληθούν αν αναθέσουν στην GPU συγκεκριμένα μέρη τους, τα οποία έχουν μεγάλο υπολογιστικό κόστος και είναι κατάλληλα για αντιμετώπιση από τους απλούστερους πυρήνες της GPU. Ένα υπολογιστικό περιβάλλον που χρησιμοποιεί παραπάνω από ένα είδος επεξεργαστή (π.χ. CPU και GPU) για την εκτέλεση της ίδιας εργασίας λέγεται ετερογενές (heterogeneous). Η ανάπτυξη προγραμμάτων που να χρησιμοποιούν τους εκατοντάδες πυρήνες της GPU ή περισσότερους από έναν επεξεργαστές απαιτεί γνώσεις παράλληλου προγραμματισμού.

Στη συνέχεια παρουσιάζονται οι δυνατότητες των καρτών γραφικών για την επιτάχυνση της εκτέλεσης διαδικασιών που χρησιμοποιούνται στην μέθοδο των πεπερασμένων στοιχείων. Γίνεται σύγκριση με την συμπεριφορά μιας συνηθισμένης CPU στα ίδια προβλήματα και αναφέρονται συνοπτικά κάποια εισαγωγικά στοιχεία για την ανάπτυξη κώδικα σε GPGPU. Πρώτα όμως εξηγούνται κάποιες βασικές έννοιες του παράλληλου προγραμματισμού.

4.2 Παράλληλος προγραμματισμός

Ο παράλληλος προγραμματισμός είναι σημαντικά πιο δύσκολος από τον σειριακό. Γι' αυτό, μέχρι πρόσφατα δεν αναπτύσσονταν παράλληλες εμπορικές εφαρμογές. Ωστόσο ένας παράλληλος αλγόριθμος, που εκτελείται σε πολλούς πυρήνες, μπορεί να είναι πολύ ταχύτερος από έναν ίδιο αλλά σειριακό αλγόριθμο. Όπως αναλύθηκε προηγουμένως, στο μέλλον η

αύξηση της απόδοσης των επεξεργαστών θα προέρχεται κυρίως από την χρήση πολλαπλών πυρήνων, οπότε η απόδοση των σειριακών αλγορίθμων θα αυξάνεται με πολύ πιο αργό ρυθμό.

Εκτός από τα οφέλη ταχύτητας, με τη διάσπαση ενός προγράμματος σε ανεξάρτητες διεργασίες μπορούμε να αντιμετωπίσουμε περιπτώσεις, όπου το πρόγραμμα σταματάει να λειτουργεί μέχρι να συμβεί κάποιο γεγονός (blocking operations). Αν υπάρχουν άλλες εντολές που το πρόγραμμα μπορεί να εκτελέσει όσο περιμένει να συμβεί το γεγονός, τότε έχει νόημα να τις αναθέσουμε σε μια παράλληλη διεργασία.

4.2.1 Ταχύτητα παράλληλων προγραμμάτων

Θεωρητικά η επιτάχυνση (speed-up) από την παράλληλη εφαρμογή ενός αλγορίθμου είναι ανάλογη του αριθμού των επεξεργαστών στους οποίους μοιράζεται το υπολογιστικό φορτίο. Στην πραγματικότητα όμως η σχέση μεταξύ αυτών των δύο δεν είναι γραμμική σχέση. Ένα πρόγραμμα αποτελείται από τμήματα που μπορούν να εκτελεστούν παράλληλα και από τμήματα που δεν μπορούν ή δεν συμφέρει να μετατραπούν σε παράλληλους αλγορίθμους. Αν ονομάσουμε T τον συνολικό χρόνο που απαιτείται για την σειριακή εκτέλεση όλων των τμημάτων του προγράμματος σε έναν πυρήνα, B τον συνολικό χρόνο που απαιτείται για την εκτέλεση των τμημάτων που δεν μπορούν να μετατραπούν σε παράλληλα και N τον αριθμό των πυρήνων, τότε ο χρόνος εκτέλεσης στους N πυρήνες είναι:

$$T(N) = B + \frac{T - B}{N} \qquad \textbf{(4.1)}$$

Έτσι σύμφωνα με τον νόμο του Amdahl (1967) η μέγιστη επιτάχυνση που μπορεί να επιτευχθεί είναι:

$$speedup = \lim_{N \to \propto} \frac{T}{\frac{T - B}{N} + B} = \frac{1}{b}, \qquad b = B/T \qquad \textbf{(4.2)}$$

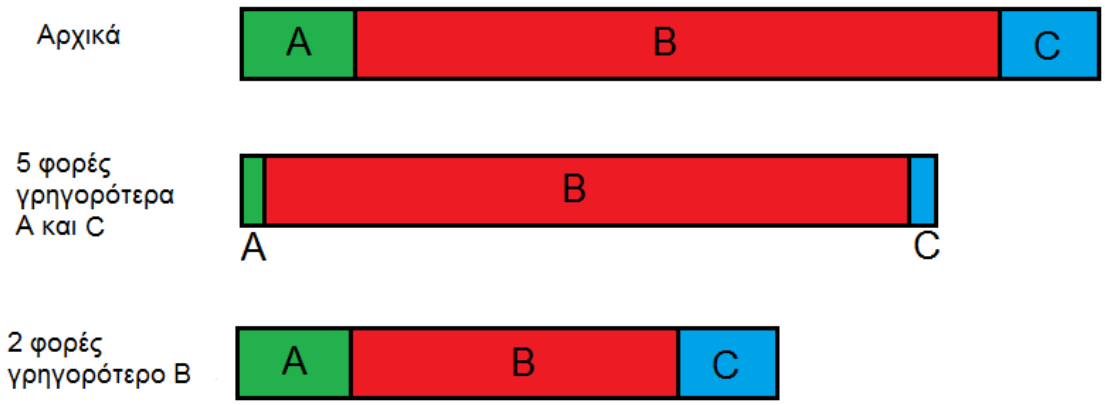
Δηλαδή εξαρτάται από το ποσοστό του προγράμματος που δεν μπορεί να εκτελεστεί παράλληλα. Προς το παρόν, ο μικρός αριθμός πυρήνων στις CPU δεν οδηγεί στην οριακή αυτή κατάσταση. Στις GPU όμως υπάρχουν εκατοντάδες ή χιλιάδες πυρήνες, με αποτέλεσμα ο περιορισμός αυτός να επηρεάζει σημαντικά.

Το ποσοστό ενός αλγορίθμου που εκτελείται παράλληλα είναι καθοριστικής σημασίας για το σχεδιασμό ενός προγράμματος. Ας πάρουμε για παράδειγμα ένα πρόγραμμα ανάλυσης με πεπερασμένα στοιχεία. Τυπικά αποτελείται από:

A. ένα υποπρόγραμμα (preprocessor) που επικοινωνεί με τον χρήστη και παράγει το δίκτυο των πεπερασμένων στοιχείων (mesh),

B. ένα υποπρόγραμμα (solver) που εκτελεί ολοκληρώσεις και διαδικασίες γραμμικής άλγεβρας για την δημιουργία και την επίλυση αντίστοιχα του γραμμικού συστήματος $[M] * \{\ddot{\Delta}\} + [C] * \{\dot{\Delta}\} + [K] * \{\Delta\} = \{P(t)\}$
C. ένα υποπρόγραμμα (postprocessor) που υπολογίζει παραμορφώσεις, τάσεις και τις παριστάνει γραφικά.

Το μεγαλύτερο μέρος του υπολογιστικού χρόνου αφιερώνεται στο B. Επομένως είναι πιο αποδοτικό να γίνουν οι λειτουργίες του solver παράλληλες, ακόμα και αν αυτό είναι δυσκολότερο, όπως φαίνεται στην επόμενη εικόνα.



Σχήμα 4.1 Ποιοτική κατανομή υπολογιστικού χρόνου σε λογισμικά FEM

4.2.2 Βασικές Έννοιες

Στη συνέχεια παρουσιάζονται κάποιες βασικές έννοιες που εμφανίζονται στα διάφορα μοντέλα παράλληλου προγραμματισμού που έχουν αναπτυχθεί. Καθεμία από τις διεργασίες που μπορούν να εκτελούνται ταυτόχρονα ονομάζεται thread (νήμα). Κάθε thread μπορεί να εκτελεί τις ίδιες εντολές σε άλλες θέσεις μνήμης ή εντελώς διαφορετικές εντολές από τα υπόλοιπα. Μπορεί να ξεκινάει και να τερματίζει σε άλλες χρονικές στιγμές και να έχει διαφορετική ροή ελέγχου. Τα threads ενός προγράμματος μπορεί να επικοινωνούν μεταξύ τους σε κάποια σημεία. Πολλές από τις βασικότερες δυσκολίες του παράλληλου προγραμματισμού προκύπτουν από την έλλειψη ελέγχου της ροής του προγράμματος: τα διάφορα threads εκτελούνται σε μη προβλέψιμες και μη ντετερμινιστικές χρονικές στιγμές.

Έστω ο αλγόριθμος:

$$\begin{aligned} &1:\ x \leftarrow 2 \\ &2:\ x \leftarrow x + 1 \\ &3:\ x \leftarrow 2 * x \\ &\textit{Αποτέλεσμα: } x\text{=6} \end{aligned} \qquad \textbf{(4.3)}$$

Αν αναθέσουμε τα βήματα 2 και 3 σε δύο διαφορετικά threads A και B αντίστοιχα, τότε δεν μπορούμε να ξέρουμε ποιο από τα δύο θα εκτελεστεί πρώτο:

$$A \text{ και μετά } B: \quad \begin{array}{l} 1:\ x \leftarrow 2 \\ \textit{Εκτέλεση Thread A} \\ 2:\ x \leftarrow x + 1 \\ \textit{Εκτέλεση Thread B} \\ 3:\ x \leftarrow 2 * x \\ \textit{Αποτέλεσμα: } x = 6 \end{array} \qquad \textbf{(4.4)}$$

$$B \text{ και μετά } A: \quad \begin{array}{l} 1:\ x \leftarrow 2 \\ \textit{Εκτέλεση Thread B} \\ 2:\ x \leftarrow 2 * x \\ \textit{Εκτέλεση Thread A} \\ 3:\ x \leftarrow x + 1 \\ \textit{Αποτέλεσμα: } x = 5 \end{array} \qquad \textbf{(4.5)}$$

$$1:\ x \leftarrow 2$$

$$A \text{ και } B \text{ ταυτόχρονα:} \quad \begin{array}{l|l} \textit{Thread A} & \textit{Thread B} \\ \textit{Ανάγνωση } x = 2 & \textit{Ανάγνωση } x = 2 \\ x \leftarrow x + 1 & x \leftarrow 2 * x \\ \textit{Εγγραφή } x = 3 & \textit{Εγγραφή } x = 4 \end{array}$$

$$\textit{Αποτέλεσμα } x = 3 \text{ ή } 4\textit{, όποιο εγγραφεί τελευταίο} \qquad \textbf{(4.6)}$$

Στην περίπτωση αυτή τα δύο threads «αγωνίζονται» για την ανάγνωση και την εγγραφή στις ίδιες θέσεις μνήμης (race conditions). Προκειμένου να αντιμετωπιστούν τέτοιες καταστάσεις, ο παράλληλος προγραμματισμός χρησιμοποιεί εντολές (atomic operations) που επιτρέπουν σε ένα thread να «κλειδώσει» (lock) τις απαραίτητες θέσεις μνήμης για όσο τις χρειάζεται. Τα υπόλοιπα threads, που αργούν να φτάσουν στις κλειδωμένες θέσεις μνήμης, αναγκάζονται να περιμένουν πότε θα ξεκλειδωθούν.

Ένας άλλος μηχανισμός ελέγχου της αλληλουχίας των threads είναι ο συγχρονισμός τους (barrier synchronization). Με την τεχνική αυτή προσδιορίζονται συγκεκριμένα σημεία του κώδικα που λειτουργούν ως φράγματα. Όταν ένα thread φτάνει στο φράγμα, αναγκάζεται να περιμένει μέχρι να φτάσουν και όλα τα υπόλοιπα. Έτσι μπορούν και να ανταλλάσσονται πληροφορίες μεταξύ τους σε προκαθορισμένα σημεία. Ο μηχανισμός αυτός έχει ιδιαίτερη σημασία στην GPGPU αφού εφαρμόζεται και για τον συγχρονισμό της CPU με την GPU. Υπάρχουν πολλές ακόμα τεχνικές που εφαρμόζονται στον παράλληλο προγραμματισμό, αλλά αυτές αρκούν για την κατανόηση των επόμενων ενοτήτων.

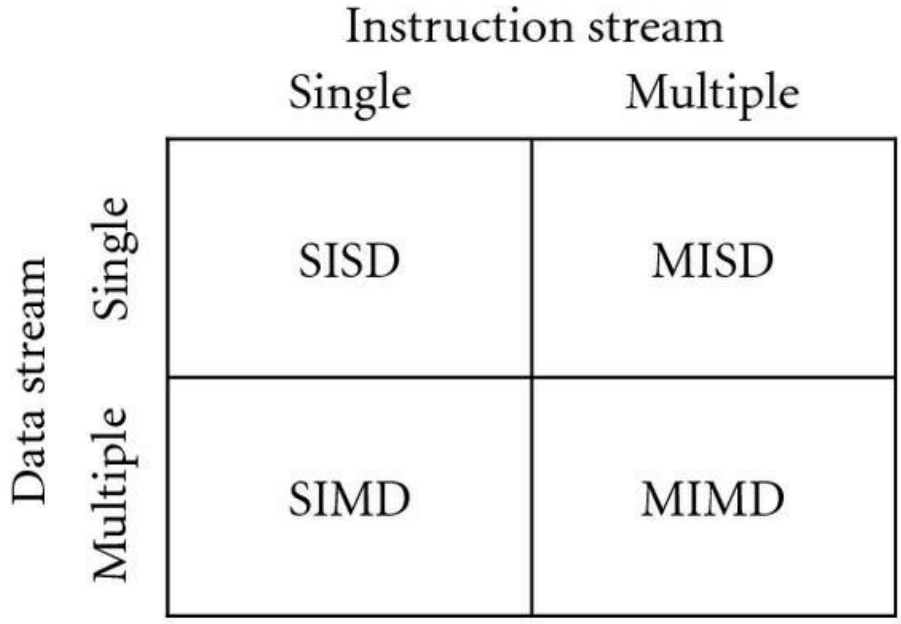
4.2.3 Κατηγορίες παραλληλίας

Ο βαθμός στον οποίον ένας αλγόριθμος μπορεί να μετατραπεί σε παράλληλο ποικίλει. Αλγόριθμοι που σε κάθε βήμα χρειάζονται τα αποτελέσματα των προηγούμενων, όπως π.χ. η μέθοδος Newton-Raphson, γενικά προορίζονται μόνο για σειριακή εκτέλεση (inherently serial). Αντίθετα, άλλοι αλγόριθμοι μπορούν να διαιρεθούν σε υπό εργασίες οι οποίες είναι τελείως ανεξάρτητες. Για παράδειγμα, η πρόσθεση δύο διανυσμάτων μήκους Ν μπορεί να διαιρεθεί σε Ν προσθέσεις των στοιχείων τους, οι οποίες μπορούν να γίνουν ταυτόχρονα πτωχαίνοντας πολύ μεγάλη επιτάχυνση. Αν ένα πρόγραμμα αποτελείται από διεργασίες που χρειάζεται να επικοινωνούν σπάνια ή καθόλου, τότε είναι πολύ εύκολο να εκτελεστεί παράλληλα (embarrassingly parallel). Οι μεταευριστικοί αλγόριθμοι βελτιστοποίησης, που αναλύθηκαν στο κεφάλαιο 2, χρησιμοποιούν πληθυσμούς πιθανών λύσεων, καθεμία από τις οποίες μπορεί να ανανεώνεται και δοκιμάζεται παράλληλα με τις άλλες. Επικοινωνία μεταξύ τους απαιτείται μόνο στο τέλος κάθε επανάληψης. Σε άλλα προγράμματα οι διεργασίες χρειάζεται να επικοινωνούν πολλές (fine-grained parallelism) ή λίγες (coarse-grained parallelism)φορές κάθε δευτερόλεπτο, οπότε και αναγκάζονται να περιμένουν η μία την άλλη.

Πολλοί αλγόριθμοι που δείχνουν σειριακοί εκ πρώτης όψης μπορούν να διασπαστούν σε ανεξάρτητα μέρη, τα οποία εκτελούνται παράλληλα και έπειτα συντίθενται για να δώσουν το τελικό αποτέλεσμα. Για παράδειγμα ο υπολογισμός του αθροίσματος ενός συνόλου αριθμών μπορεί να διαιρεθεί σε υπολογισμούς υποσυνόλων του και στο τέλος τα υπό-αθροίσματα να προστεθούν μεταξύ τους. Άλλο παράδειγμα είναι ο διαχωρισμός ενός μοντέλου πεπερασμένων στοιχείων σε υποφορείς που επιλύονται παράλληλα και στη συνέχεια συντονίζονται με τους γειτονικούς τους, μέσω επαναληπτικών μεθόδων.

Ένα από τα πρώτα συστήματα κατηγοριοποίησης παράλληλων και σειριακών προγραμμάτων είναι του M.J. Flynn (1966). Σύμφωνα με αυτό τα προγράμματα χωρίζονται σε κατηγορίες ανάλογα με τα αν threads τους εκτελούν την ίδια ή διαφορετικές εντολές, στην ίδια ή διαφορετικές θέσεις μνήμης. Σύμφωνα με αυτό έχουμε:

- Single Instruction, Single Data: Είναι οι παραδοσιακοί σειριακοί αλγόριθμοι
- Single Instruction, Multiple Data: Η ίδια εντολή εκτελείται ταυτόχρονα σε πολλές θέσεις μνήμης, από διαφορετικά threads. Το πρόγραμμα εκμεταλλεύεται την παραλληλία των δεδομένων (data parallelism). Οι κάρτες γραφικών λειτουργούν με αυτόν τον τρόπο, οπότε θα αναλυθεί εκτενώς στη συνέχεια.
- Multiple Instructions, Single Data: Δεν έχει πολλές πρακτικές εφαρμογές.
- Multiple Instructions, Multiple Data: Τα διάφορα threads εκτελούν ταυτόχρονα ίδιες ή διαφορετικές οδηγίες (task parallelism) σε κοινές ή μη θέσεις μνήμης. Οι CPU με πολλούς πυρήνες και τα διανεμημένα συστήματα ανήκουν σε αυτήν την κατηγορία.



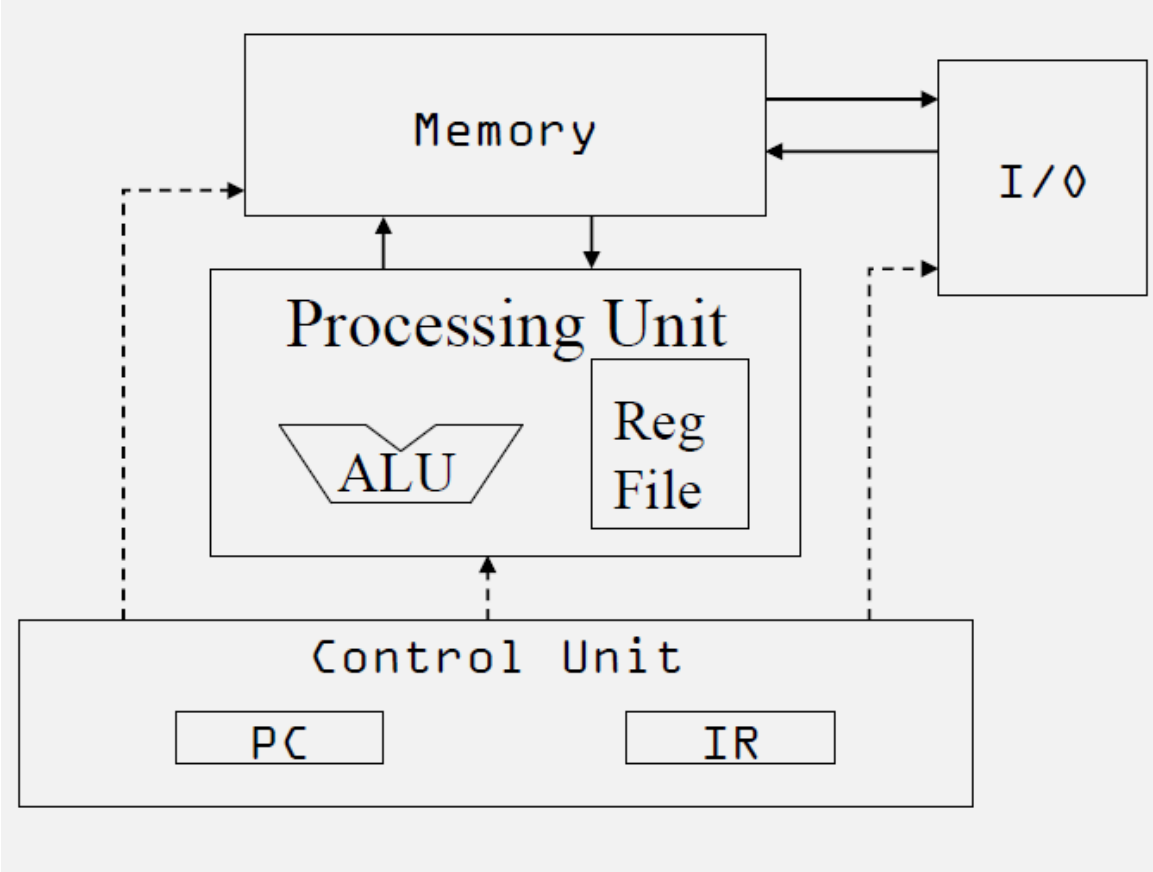
Σχήμα 4.2 Κατηγοριοποίηση παράλληλων προγραμμάτων κατά Flynn

Οι διεργασίες στις οποίες χωρίζεται το πρόγραμμα δεν είναι απαραίτητο να εκτελούνται στον ίδιο επεξεργαστή. Στα ετερογενή συστήματα (heterogeneous computing systems συνυπάρχουν στον ίδιο υπολογιστή παραπάνω από μια μονάδες επεξεργασίες, που εκτελούν ταυτόχρονα διαφορετικά μέρη του προγράμματος. Η απλούστερη περίπτωση είναι η χρήση CPU και GPU που καθεμία έχει τη δικιά της μνήμη RAM. Τα τελευταία χρόνια κυκλοφορούν μικρότερες GPU που είναι ενσωματωμένες στη πλακέτα της CPU και μοιράζονται την ίδια RAM. Πολλά συστήματα (συνήθως προσανατολισμένα για βιντεοπαιχνίδια) χρησιμοποιούν 2 ως 4 GPU. Η χρήση πολλών CPU ταυτόχρονα, δεν είναι ιδιαίτερα αποδοτική και δεν συνηθίζεται σε προσωπικούς υπολογιστές. Τέλος, είναι δυνατή η αξιοποίηση υπολογιστικών πόρων σε διαφορετικούς υπολογιστές (distributed systems). Στην περίπτωση αυτή επιστρατεύονται υπολογιστές που ανήκουν σε ένα δίκτυο και καθένας μπορεί να εκτελεί διαφορετικές εργασίες ενός προγράμματος. Η μνήμη προφανώς δεν είναι κοινή, οπότε απαιτείται ανταλλαγή δεδομένων μέσω του δικτύου.

4.3 Σύγκριση GPU με CPU

4.3.1 Αρχιτεκτονική της CPU

Για να γίνει κατανοητή η διαφορά των GPU με τις GPU, χρειάζεται γνώση της αρχιτεκτονικής των δύο επεξεργαστών. Στο παρακάτω σχήμα φαίνεται η αρχιτεκτονική Von Neumann:



Σχήμα 4.3 Επεξεργαστής Von Neumann

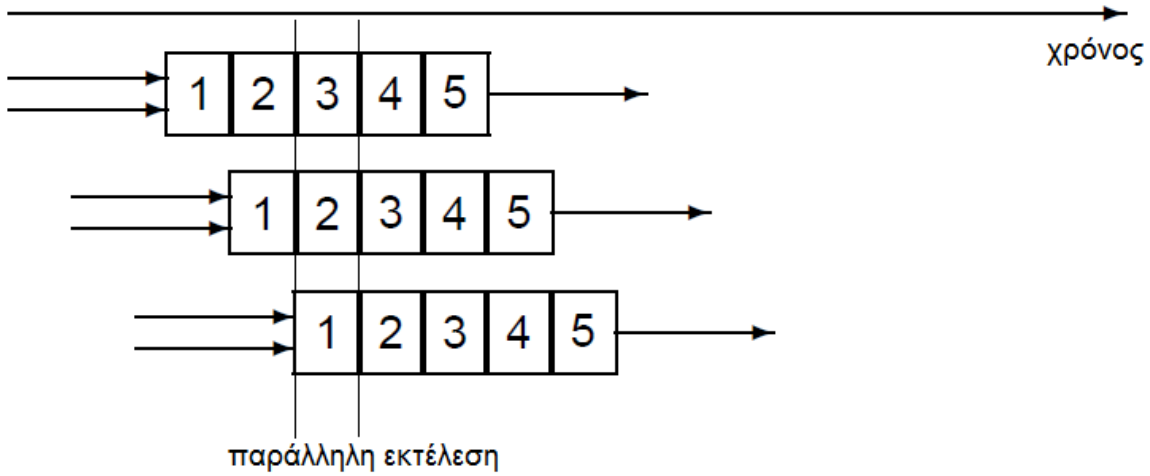
- I/O (Input/Output) είναι η είσοδος και έξοδος δεδομένων μεταξύ προγράμματος και περιφερειακών συσκευών ή αρχείων.
- Memory είναι η μνήμη RAM του συστήματος, όπου αποθηκεύονται τα δεδομένα αλλά και οι εντολές του προγράμματος.
- Processing Unit είναι π.χ. ένας πυρήνας της CPU. Αποτελείται από τις αριθμητικές-λογικές μονάδες (Arithmetic Logic Unit – ALU) που εκτελούν τις πράξεις του προγράμματος και τους καταχωρητές (Register File). Οι καταχωρητές είναι αποθηκευτικές μονάδες που βρίσκονται πάνω στο κύκλωμα του επεξεργαστή και γι' αυτό είναι πολύ πιο γρήγορες από την κύρια μνήμη RAM. Ο αποθηκευτικός τους χώρος όμως είναι πολύ μικρότερος.
- Control Unit είναι η μονάδα ελέγχου που συντονίζει την εκτέλεση των εντολών. Αποτελείται από τον Instruction Register (IR) που αποθηκεύει την τρέχουσα εντολή και τον Program Counter (PC) που δείχνει την θέση στην οποία βρίσκεται το πρόγραμμα κάθε δεδομένη στιγμή.

Κατά την εκτέλεση ενός προγράμματος, τυπικά διαβάζεται μία εντολή από τη μνήμη, αποθηκεύεται στον IR και ο PC αυξάνεται κατά 1. Η εντολή μπορεί να είναι ανάγνωση μνήμης (RAM), εγγραφή στην μνήμη ή εκτέλεση μίας πράξης. Κατά την ανάγνωση μνήμης, τα δεδομένα μεταφέρονται από την RAM στο Register File. Τα ALU εκτελούν πράξεις με αυτά τα δεδομένα

και στη συνέχεια τα αποτελέσματα μεταφέρονται από τους καταχωρητές πίσω στην κύρια μνήμη.

Οι σύγχρονες CPU χρησιμοποιούν εξελιγμένες τεχνικές για να αυξήσουν την υπολογιστική ισχύ τους:

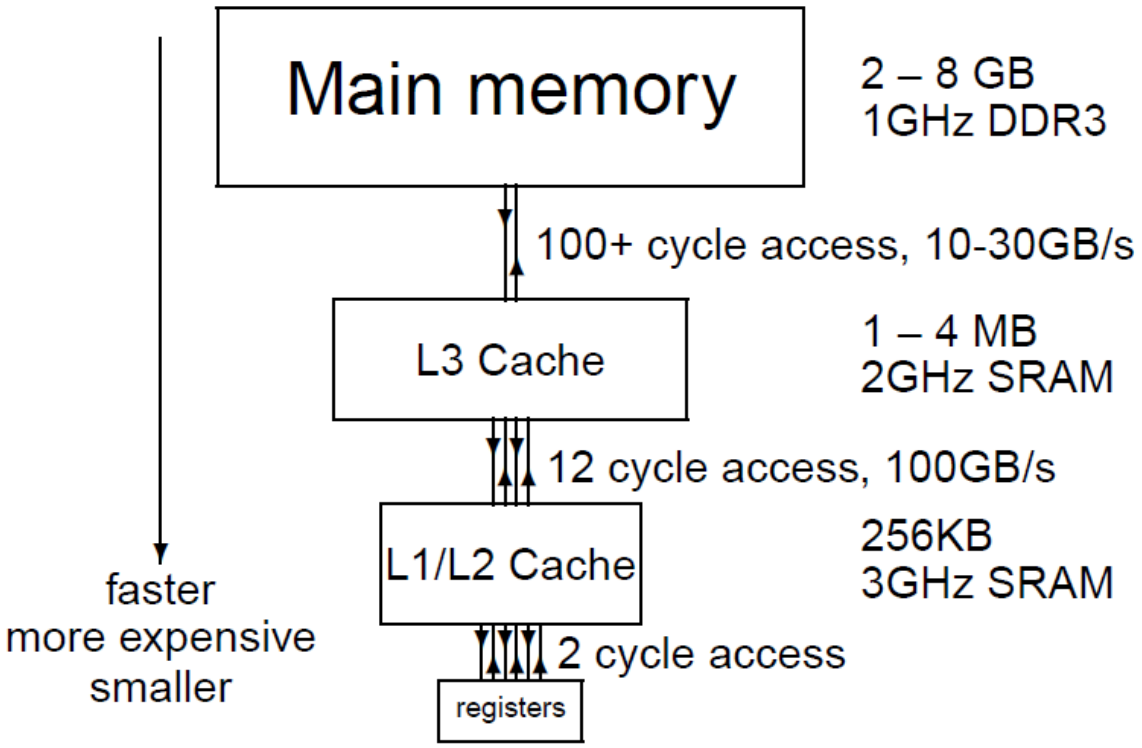
- Πυρήνες με υψηλή συχνότητα περιστροφής, τυπικά $2.000 - 3.000$ Mhz.
- Ταυτόχρονη εκτέλεση διαφορετικών εντολών (pipelining). Οι διεργασίες που περιγράφηκαν για τον επεξεργαστή Von Neumann μπορούν να εκτελεστούν παράλληλα (instruction level parallelism), ώστε να κερδίζεται χρόνος μες την αλληλοεπικάλυψη τους (overlap).



Σχήμα 4.4 Παραλληλία σε επίπεδο εντολής - pipelines

- Ισχυρά ALU που χρειάζονται λίγους κύκλους για την εκτέλεση πράξεων. Συνήθως χρησιμοποιούνται περισσότερα. Η χρήση περισσότερων ALU επιτρέπει επιπλέον μακρύτερα pipelines. Για παράδειγμα η αρχιτεκτονική Intel Core Duo υποστηρίζει 5 ALU και pipelines 14 σταδίων.
- Αναδιάταξη των εντολών για βέλτιστη απόδοση. Την αναδιάταξη την αναλαμβάνει συνήθως ο μεταγλωττιστής (compiler), οπότε αποτελεί περισσότερο αντικείμενο λογισμικού.
- Εκτέλεση εντολών εκτός σειράς ώστε να αποφεύγονται οι αναμονές για διάβασμα ή γράψιμο στη μνήμη.
- Πρόβλεψη διακλαδώσεων ώστε να ελαχιστοποιούνται οι καθυστερήσεις, οι οποίες προέρχονται από τον προσδιορισμό της πορείας του προγράμματος και την ανάγνωση των αντίστοιχων εντολών, σε περιπτώσεις διακλαδώσεων (if…then…else) και βρόγχων (for, while, …)
- Ιεράρχηση της μνήμης (memory hierarchy). Η λογική είναι ότι ο επεξεργαστής πρέπει να έχει όσο το δυνατόν γρηγορότερη πρόσβαση σε δεδομένα που πρόκειται να χρησιμοποιηθούν σύντομα. Αυτό επιτυγχάνεται με την χρήση μνήμης cache σε πολλά επίπεδα. Η μνήμη cache είναι αποθηκευτικός χώρος (SRAM) πάνω στη πλακέτα της CPU, ο

οποίος προσφέρει πολύ γρηγορότερη πρόσβαση από ότι η κύρια μνήμη (DRAM). Είναι όμως πολύ μικρότερος και ακριβότερος από αυτήν. Δεδομένα που μόλις διαβάζονται από την κεντρική μνήμη είναι πιθανόν να χρειαστούν ξανά σύντομα, οπότε τοποθετούνται στην cache. Μαζί με αυτά τοποθετούνται και γειτονικά τους δεδομένα.

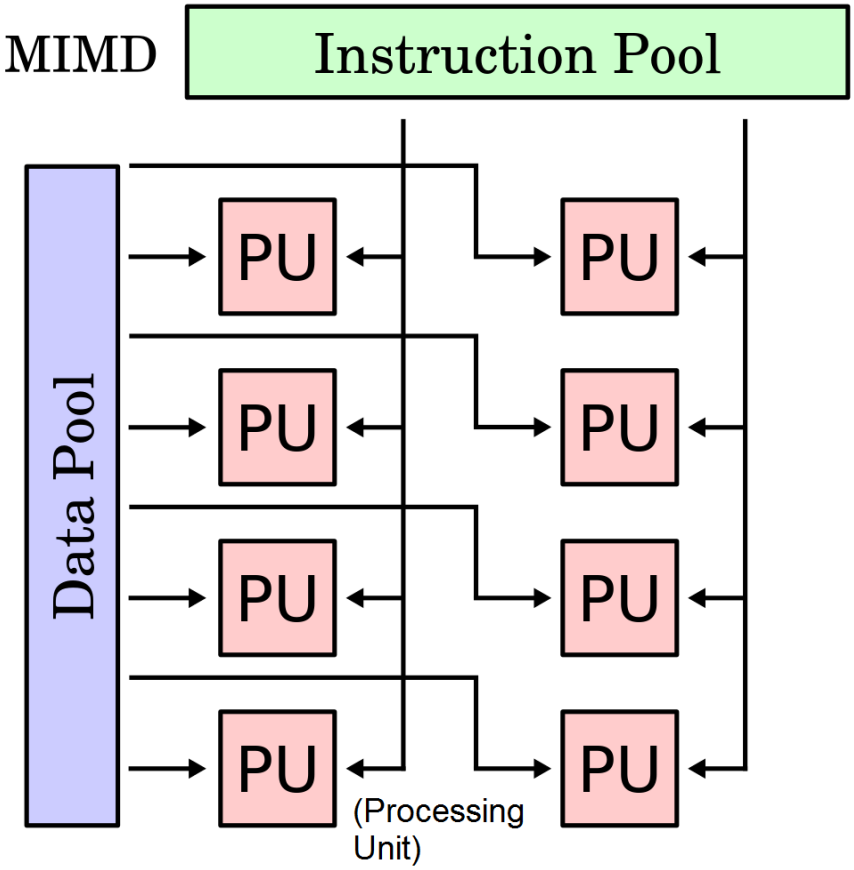


Σχήμα 4.5 Μνήμες και cache στην CPU

Από την οπτική γωνία του προγραμματιστή όλα αυτά γίνονται αυτόματα. Οι σύγχρονες γλώσσες προγραμματισμού για CPU προσφέρουν αυτήν την απλότητα, που είναι απαραίτητη για την ταχεία ανάπτυξη λογισμικών γενικής χρήσης. Ωστόσο αυτό σημαίνει ότι ο προγραμματιστής δεν μπορεί να βελτιστοποιήσει τον κώδικά του, ώστε να αξιοποιήσει τις παραπάνω τεχνολογίες με τον πιο αποδοτικό τρόπο. Στις GPU οι γλώσσες προγραμματισμού αναπτύχθηκαν τα τελευταία χρόνια και δεν έχουν προλάβει να ωριμάσουν σε τέτοιο επίπεδο, που να κρύβονται όλες αυτές οι λεπτομέρειες. Έτσι ο προγραμματιστής αφιερώνει μεγάλο μέρος της εργασίας του στην κατάλληλη χρήση των διαφόρων ειδών μνήμης και cache. Βέβαια αν σκεφτούμε ότι η GPU χρησιμοποιείται συνήθως για τα πλέον χρονοβόρα μέρη ενός προγράμματος, είναι λογικό να επιχειρείται η βελτιστοποίηση τους σε χαμηλό επίπεδο.

Στις σύγχρονες CPU με πολλαπλούς πυρήνες, η πλήρης εκμετάλλευση της υπολογιστικής ισχύος απαιτεί την παράλληλη εκτέλεση πολλών προγραμμάτων αλλά και κάθε προγράμματος ατομικά. Το πρώτο ρυθμίζεται αυτόματα από το λειτουργικό σύστημα. Η διάσπαση όμως ενός προγράμματος σε μικρότερα που εκτελούνται παράλληλα απαιτεί την εφαρμογή των τεχνικών

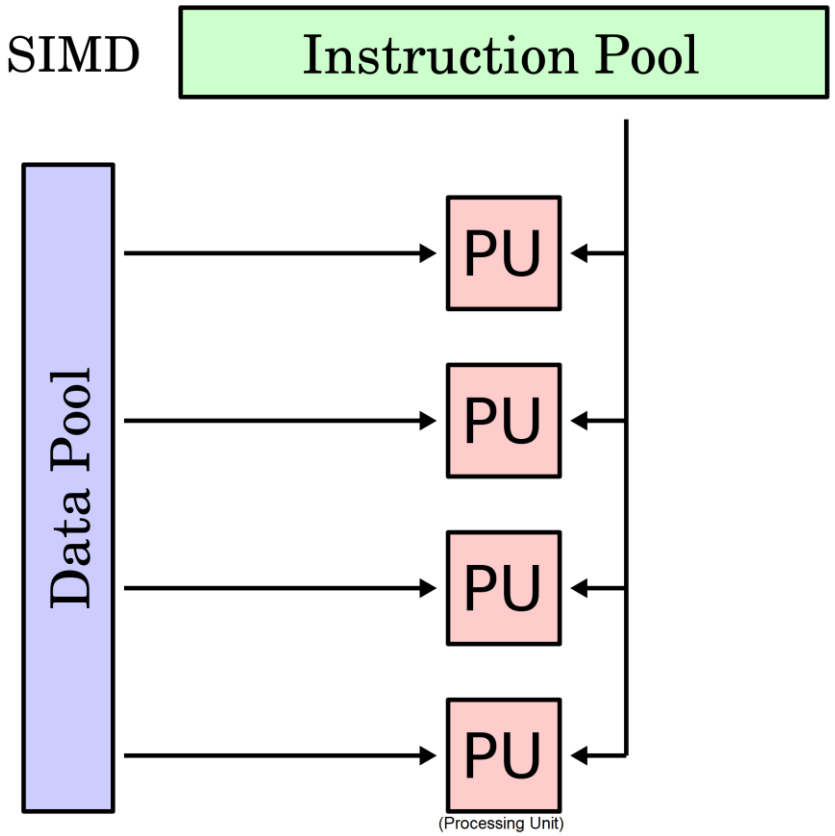
παράλληλου προγραμματισμού. Οι multicore CPU υποστηρίζουν το MIMD μοντέλο παραλληλίας, δηλαδή κάθε thread μπορεί να λειτουργεί ανεξάρτητα και να εκτελεί διαφορετικές εντολές σε διαφορετικές θέσεις μνήμης από τα υπόλοιπα, όπως φαίνεται στο επόμενο σχήμα.



Σχήμα 4.6 MIMD μοντέλο σε CPU με 8 πυρήνες

4.3.2 Αρχιτεκτονική της GPU

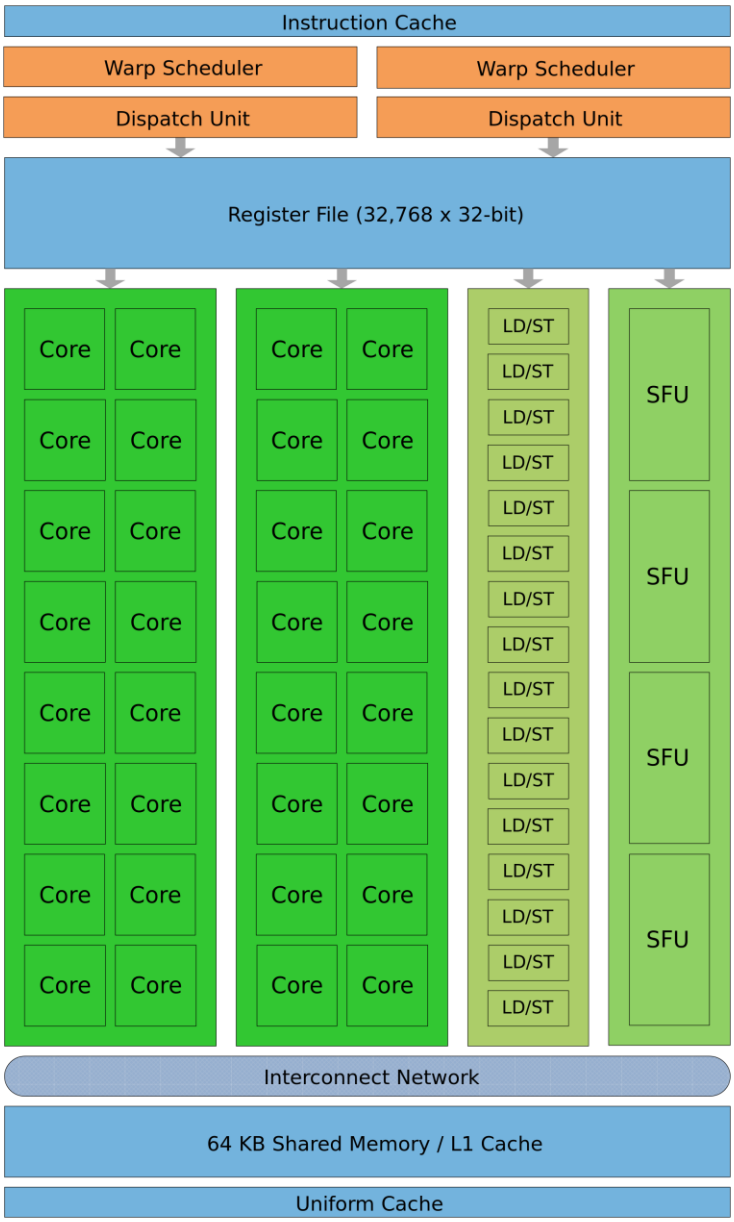
Η GPU είναι μαζικά παράλληλη μονάδα επεξεργασίας, που αποτελείται συνήθως από εκατοντάδες πυρήνες, καθένας από τους οποίους μπορεί να εκτελεί ταυτόχρονα εκατοντάδες threads. Η λειτουργία της βασίζεται στη SIMD μορφή παραλληλίας. Έτσι, από την οπτική γωνία του προγραμματιστή, όλα τα thread όλων των πυρήνων εκτελούν ταυτόχρονα την ίδια εντολή, καθένα σε διαφορετικές θέσεις μνήμης (data parallelism). Στην πραγματικότητα αυτό γίνεται ταυτόχρονα ανά ομάδες threads, όχι για όλα, αλλά ο συντονισμός των ομάδων γίνεται αυτόματα από το hardware. Για κάθε ομάδα thread απαιτείται να διαβαστούν και αποκωδικοποιηθούν οι εντολές του προγράμματος μόνο μία φορά, όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 4.7 SIMD μοντέλο για κάθε ομάδα πυρήνων.

Οι πυρήνες της GPU είναι οργανωμένοι σε ομάδες. Για παράδειγμα η NVidia GeForce GTX 580, που χρησιμοποιείται στις αριθμητικές εφαρμογές στη συνέχεια, αριθμεί 512 πυρήνες οργανωμένους σε 16 ομάδες των 32 πυρήνων. Η NVidia ονομάζει τις ομάδες αυτές Streaming Multiprocessors (SM). Κάθε SM:

- Συντονίζει τα threads που εκτελούνται σε αυτόν. Τα threads οργανώνονται σε ομάδες των 32, που στην αρχιτεκτονική CUDA της NVidia ονομάζονται warps. Όλα τα threads ενός warp εκτελούν ταυτόχρονα την ίδια εντολή.
- Έχει τις δικές του μονάδες ελέγχου (control units) και τα δικά του execution pipelines, που χρησιμεύουν για επικάλυψη (overlap) των εντολών διαφορετικών warps, όπως γίνεται και στη CPU (παραλληλία σε επίπεδο εντολών).
- Έχει τις δικές του αριθμητικές – λογικές μονάδες (ALU).
- Έχει δικούς του καταχωρητές (registers) και cache memories. Εκτός από αυτές υπάρχουν και κοινές caches, στις οποίες έχουν πρόσβαση όλοι οι SM της GPU.

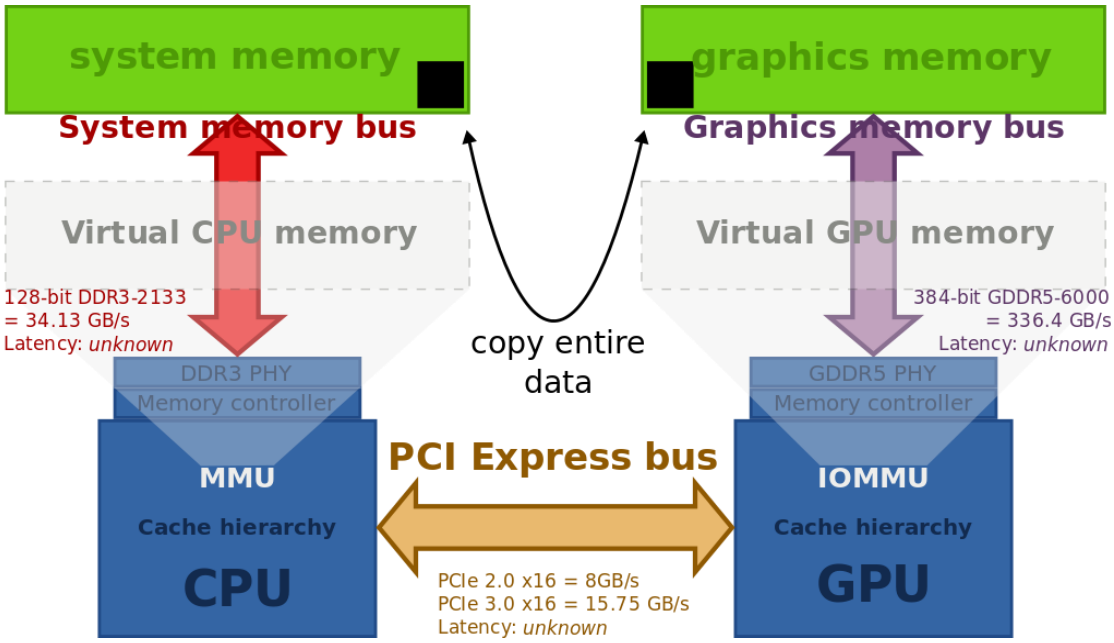


Σχήμα 4.8 Streaming Multiprocessor της αρχιτεκτονική CUDA

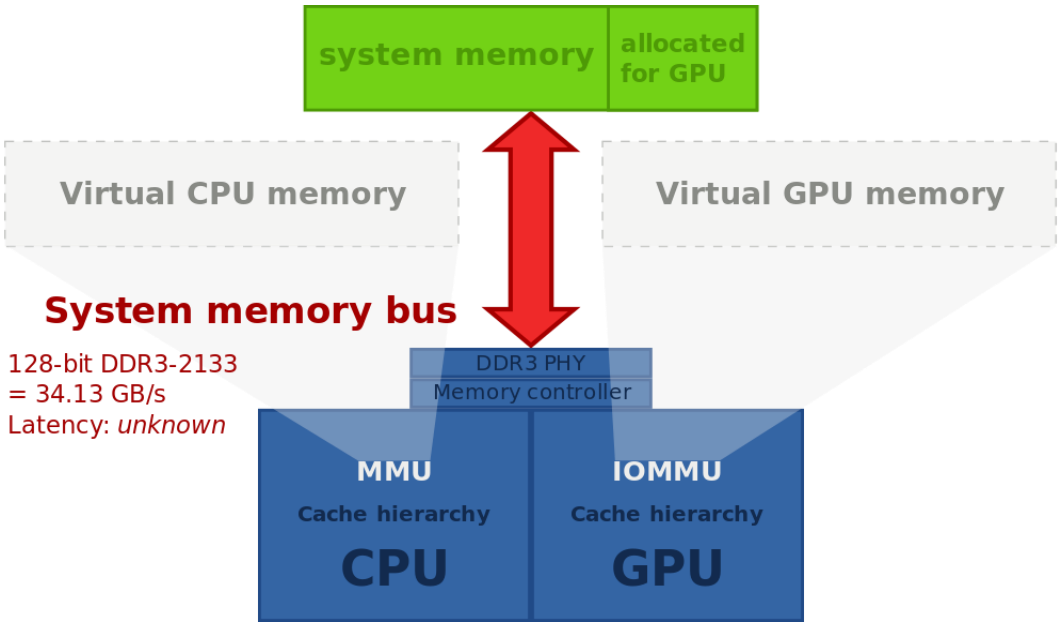
Η GPU δεν χρησιμοποιεί την κεντρική μνήμη RAM του συστήματος. Έχει τη δικιά της αντίστοιχη RAM, που είναι μικρότερη σε αποθηκευτικό χώρο (έως 6 GB), αλλά πολύ πιο γρήγορη (έως 200 GB/s). Οι registers είναι περισσότεροι από ότι στην CPU, αφού κάθε thread που εκτελείται ταυτόχρονα χρειάζεται τους δικούς του καταχωρητές. Αντίθετα η μνήμη cache είναι πιο μικρή, αλλά υπάρχουν και πάλι πολλαπλά επίπεδα, τα οποία αναλύονται στην επόμενη ενότητα.

Η χρήση διαφορετικής μνήμης από την CPU, δημιουργεί την ανάγκη μεταφοράς δεδομένων μεταξύ της κεντρικής μνήμης . Η GPU δεν βρίσκεται πάνω στη μητρική κάρτα, όπως η CPU και η κεντρική μνήμη, αλλά πάνω στην κάρτα γραφικών. Η μεταφορά δεδομένων γίνεται μέσω του διαύλου PCI-Express, που είναι πολύ πιο αργός (~8 GB/s) συγκριτικά με τις προσπελάσεις εντός

της ίδιας κάρτας. Έτσι τα προγράμματα υφίστανται σημαντικές καθυστερήσεις (overheads), όταν απαιτείται ανταλλαγή δεδομένων πολλών ή συχνών. Οι σύγχρονες GPU μπορούν να ταυτόχρονα να λαμβάνουν και να στέλνουν δεδομένα στην CPU, ενώ οι τελευταίες GPU μπορούν και να εκτελούν πράξεις ταυτόχρονα. Αντίθετα όσες GPU είναι ενσωματωμένες στο κύκλωμα της CPU (integrated GPU), δεν έχουν δικιά τους RAM, αλλά χρησιμοποιούν την κύρια μνήμη του συστήματος απευθείας. Δεν έχουν όμως την υπολογιστική ισχύ των συνηθισμένων dedicated GPU, αφού αποτελούν οικονομικές λύσεις χαμηλής κατανάλωσης, με περιορισμένες δυνατότητες.



Σχήμα 4.9 Μεταφορά δεδομένων μέσω PCI σε dedicated GPU



Σχήμα 4.10 Απευθείας μεταφορά δεδομένων σε integrated GPU

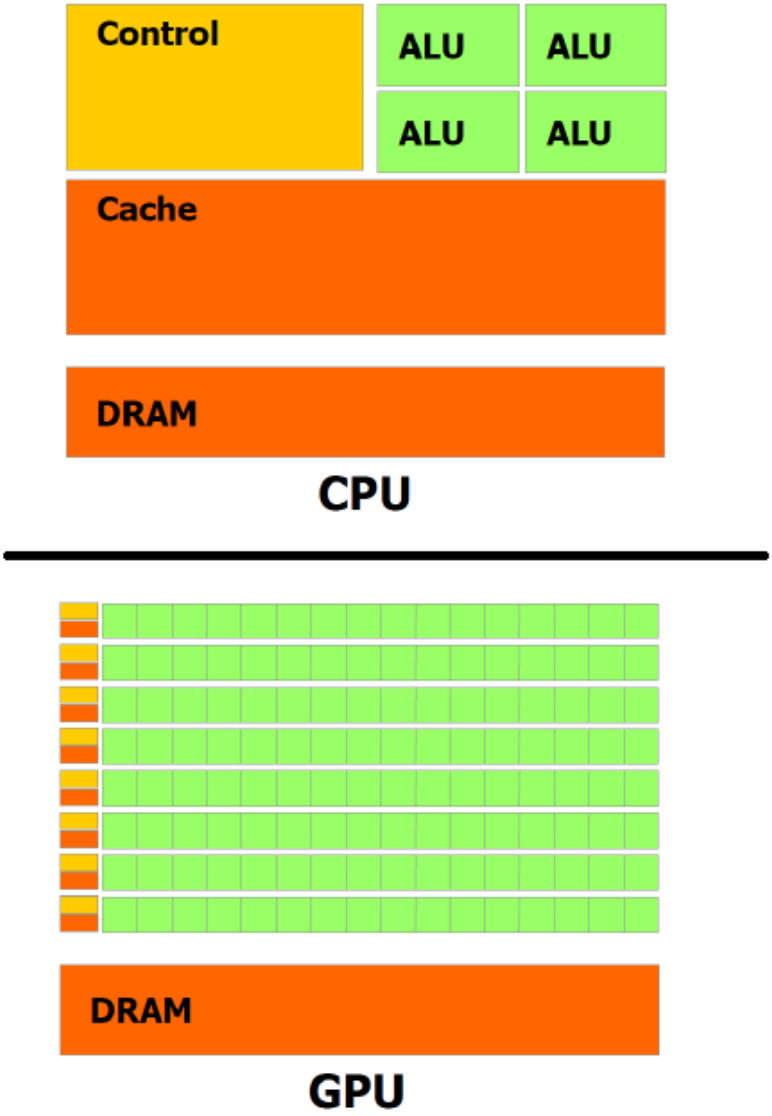
Οι μονάδες ελέγχου των GPU είναι πολύ απλές. Δεν υποστηρίζουν branch prediction ή out-of-order εκτέλεση εντολών. Έτσι σε κάθε κύκλο ενός πυρήνα GPU εκτελείται λιγότερη εργασία από ότι στους πυρήνες της CPU. Επιπλέον, η ταχύτητα περιστροφής των πυρήνων της GPU είναι σημαντικά χαμηλότερη. Παρ' όλα αυτά η υπολογιστική ισχύς της GPU είναι πολύ μεγαλύτερη λόγω των εκατοντάδων ή και χιλιάδων ALU που χρησιμοποιούνται. Τα ALU της GPU είναι απλούστερα και παρουσιάζουν μεγαλύτερους χρόνους καθυστέρησης. Για να είναι αποδοτικά χρησιμοποιείται μεγάλος αριθμός threads και έντονο pipelining.

4.3.3 Σύγκριση

Συγκεντρωτικά μπορούμε να πούμε ότι η αρχιτεκτονική της CPU είναι σχεδιασμένη ώστε να βελτιστοποιεί την απόδοση ενός Thread. Χρησιμοποιούνται λίγοι πυρήνες, τυπικά 2, 4, 6 ή 8, 10, 12 σε πολύ ακριβές CPU, υψηλής συχνότητας, τυπικά 2.000-3.000 Mhz. Κάθε πυρήνας έχει λίγα αλλά πανίσχυρα ALU. Το μεγαλύτερο μέρος της πλακέτας όμως καταλαμβάνεται από τη μονάδα ελέγχου (Control Unit) που βελτιστοποιεί τη ροή των προγραμμάτων και από τη μεγάλη μνήμη cache, π.χ. 8-15 MB L3 και 256-512 kB L1/L2. Η CPU χρησιμοποιεί την κεντρική μνήμη του συστήματος που είναι μεγαλύτερη από την μνήμη των GPU, αλλά με χαμηλότερη ταχύτητα προσπέλασης. Τυπικά 6-64 GB κεντρική μνήμη με ταχύτητα προσπέλασης 24-32 GB/s.

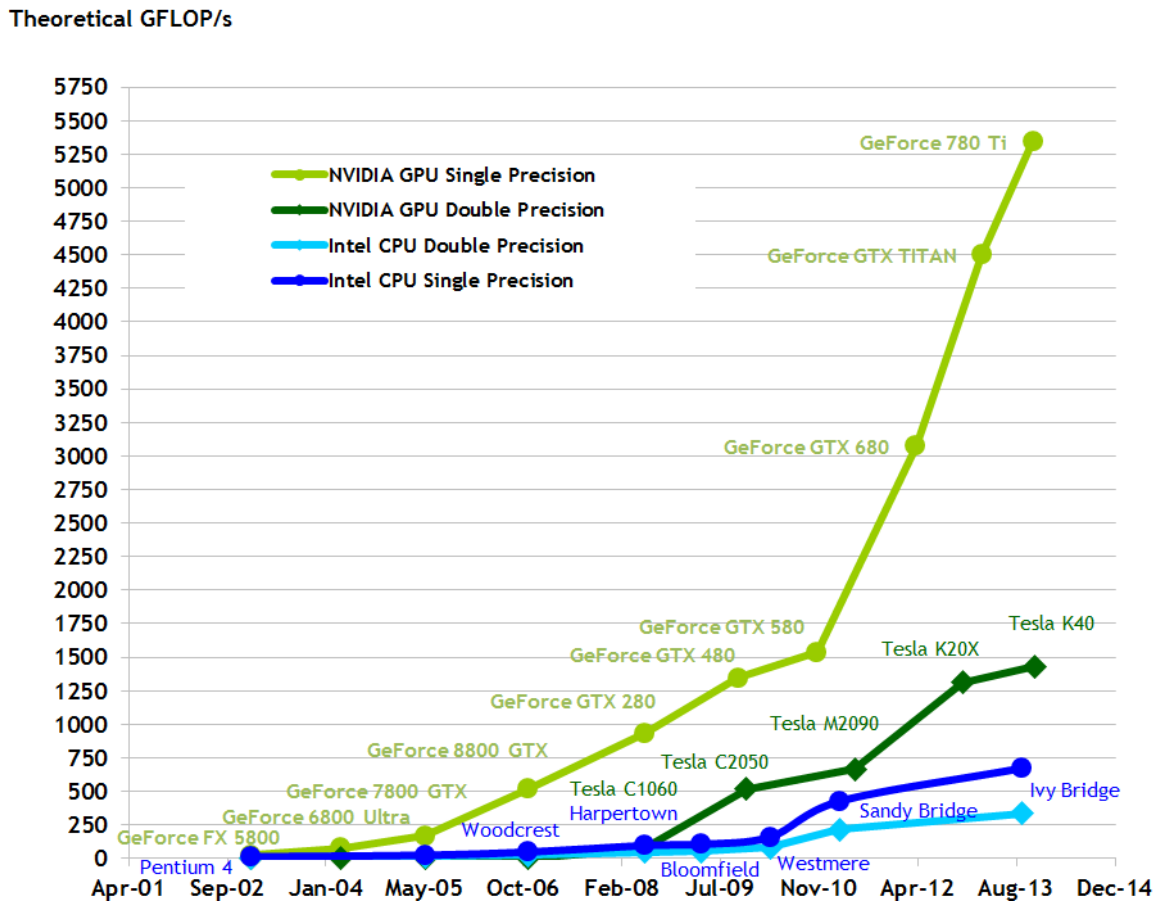
Αντίθετα η GPU είναι σχεδιασμένη ώστε να βελτιστοποιεί την εκτέλεση data parallel προγραμμάτων με πολύ μεγάλο αριθμό threads. Χρησιμοποιούνται εκατοντάδες ή χιλιάδες πυρήνες με χαμηλότερες συχνότητες (500 - 1500 MHz) από τους πυρήνες της CPU. Τα μεγέθη αυτά εξαρτώνται άμεσα από την εταιρία παραγωγής της GPU. Γενικά οι GPU της NVidia έχουν λιγότερους αλλά ταχύτερους πυρήνες από τις GPU της AMD. Για παράδειγμα η AMD Radeon HD 6990 έχει 3072 ALUs και συχνότητα 830 MHz, ενώ η Nvidia GTX 590 έχει 1024 ALUs και συχνότητα 1214 MHz. Σε κάθε περίπτωση το μεγαλύτερο μέρος της GPU καταλαμβάνεται από τα ALU και τους registers. Οι μονάδες ελέγχου έχουν απλούστερη λειτουργία και πιάνουν λιγότερο χώρο. Η μνήμη της κάρτας γραφικών είναι μικρότερη (768 MB – 6 GB), αλλά ταχύτερη (100 - 200 GB/s). Οι μνήμες cache είναι επίσης μικρότερες (512-768kB L2 και 16-48 kB L1).

Τα παραπάνω μεγέθη αφορούν πιο ακριβά μοντέλα CPU και GPU και φυσικά αυξάνονται ραγδαία κάθε χρόνο. Μια ποιοτική σύγκριση φαίνεται στο παρακάτω σχήμα. Γενικά οι CPU έχουν καλύτερη απόδοση (10+ φορές) για τα σειριακά μέρη του προγράμματος, ενώ οι GPU είναι 10+ φορές ταχύτερες για παράλληλο κώδικα. Για να αποφασιστεί αν αξίζει η χρήση GPU, πρέπει να ληφθεί υπόψη και η μεταφορά δεδομένων μεταξύ της κεντρικής μνήμης και της μνήμης της GPU. Η καθυστέρηση αυτή (overhead) συχνά είναι καθοριστικός παράγοντας.



Σχήμα 4.11 Ποιοτικές διαφορές CPU & GPU

Η υπολογιστική ισχύς των GPU είναι εντυπωσιακή. Στο επόμενο σχήμα φαίνεται η θεωρητική διαφορά σε GFLOP/s $(10^9 \text{Floating point Operations/s})$. Βέβαια, ακόμα και σε ιδανικές συνθήκες δεν πρόκειται να φανούν τέτοιες διαφορές στη πράξη, αφού η ταχύτητα με την οποία γίνονται οι προσπελάσεις και αντιγραφές μνήμης είναι πολύ μικρότερη και άρα αποτελεί τον κρισιμότερο παράγοντα του χρόνου εκτέλεσης. Επομένως τέτοια είδους διαγράμματα σύγκρισης εξυπηρετούν κυρίως διαφημιστικούς σκοπούς. Ομοίως, άρθρα και έρευνες που αναφέρουν επιταχύνσεις 100x ή και παραπάνω, όταν ο κώδικας μεταφέρεται στην GPU για παράλληλη εκτέλεση, είναι μη ρεαλιστικά.



Σχήμα 4.12 Θεωρητικές (και μόνο) αποδόσεις CPU και GPU

Για να γίνει κάποια αμερόληπτη σύγκριση, πρέπει να εξετασθούν CPU και GPU στο ίδιο εύρος τιμών και ο κώδικας που γράφεται για καθένα επεξεργαστή να είναι βελτιστοποιημένος. Μεταξύ άλλων πρέπει να χρησιμοποιούνται όλοι οι πυρήνες της CPU, αντί να παρουσιάζεται η απόδοση single threaded προγραμμάτων. Πιο αντικειμενικές έρευνες δείχνουν ότι οι παράλληλοι αλγόριθμοι σε GPU είναι συνήθως 2.5 έως 10 φορές πιο γρήγοροι από τους αντίστοιχους σε CPU. Αυτό βέβαια αφορά τα μόνο τα τμήματα του προγράμματος που προσφέρονται για SIMD παραλληλία. Υπάρχουν πολλά προβλήματα, που το μεγαλύτερο μέρος τους ή και ολόκληρο το πρόγραμμα συμφέρει να εκτελεστεί σειριακά ή με την MIMD παραλληλία που παρέχουν οι multicore CPU.

Υπάρχουν και άλλοι σημαντικοί παράγοντες που επηρεάζουν την απόδοση των δύο επεξεργαστών. Σε πραγματικές συνθήκες εκτέλεσης ενός προγράμματος, η CPU μάλλον θα εκτελεί και άλλες εργασίες, είτε αυτές είναι διεργασίες του λειτουργικού συστήματος είτε άλλα προγράμματα που χρησιμοποιεί ταυτόχρονα ο χρήστης. Από την άλλη πλευρά, οι διάφορες βελτιστοποιήσεις στον κώδικα που εκτελείται σε GPU αφορούν συγκεκριμένες μόνο αρχιτεκτονικές καρτών γραφικών. Δεν είναι απαραίτητο ότι το πρόγραμμα θα έχει την ίδια απόδοση σε κάρτα γραφικών άλλης γενιάς και κυρίως άλλης εταιρίας παραγωγής.

4.4 Προγραμματισμός GPGPU

Για τον προγραμματισμό σε ένα ετερογενές υπολογιστικό σύστημα μπορούν να χρησιμοποιηθούν τα πρότυπα παράλληλης εκτέλεσης που έχουν αναπτυχθεί για τις CPU. Αυτά δεν εξαρτώνται από το λειτουργικό σύστημα και τη γλώσσα προγραμματισμού, ενώ τα τελευταία χρόνια υποστηρίζουν και GPU computing:

- OpenMP (Open Multi-Processing). Χρησιμοποιεί εξειδικευμένες εντολές (directives) για να εκτελεί τμήματα του κώδικα σε παράλληλα threads. Μπορεί να εφαρμοστεί σχετικά εύκολα σε ένα υπάρχον πρόγραμμα. Υποστηρίζει SIMD και MIMD.
- OpenACC (Open Accelerators). Αποτελεί παραλλαγή του OpenMP που αναπτύχθηκε τα τελευταία χρόνια και λειτουργεί με παρόμοιο τρόπο. Προσανατολίζεται περισσότερο στον προγραμματισμό ετερογενών συστημάτων CPU/GPU, οπότε επιτρέπει μεγαλύτερη ευελιξία από το OpenMP για GPGPU προγράμματα. Είναι όμως ακόμα αρκετά νέο και δεν έχει βελτιστοποιηθεί στον ίδιο βαθμό.
- MPI (Message Passing Interface). Το πρωτόκολλο αυτό έχει αναπτυχθεί για την επικοινωνία πολλών επεξεργαστών – κόμβων, που δεν χρησιμοποιούν κοινή μνήμη. Πολλές πετυχημένες high performance computing (HPC) εφαρμογές χρησιμοποιούν το MPI. Για τον προγραμματισμό στους πυρήνες της GPU, που έχουν πρόσβαση σε κοινή μνήμη, υπάρχουν πιο αποδοτικοί τρόποι επικοινωνίας από το MPI. Ωστόσο μπορεί να χρησιμοποιηθεί για την επικοινωνία μεταξύ μιας ή περισσοτέρων CPU με μία ή περισσότερες GPU. Στην περίπτωση αυτή ένα άλλο πρότυπο παραλληλίας αναλαμβάνει την εκτέλεση του μέρους του κώδικα που αναλαμβάνεται από την GPU.

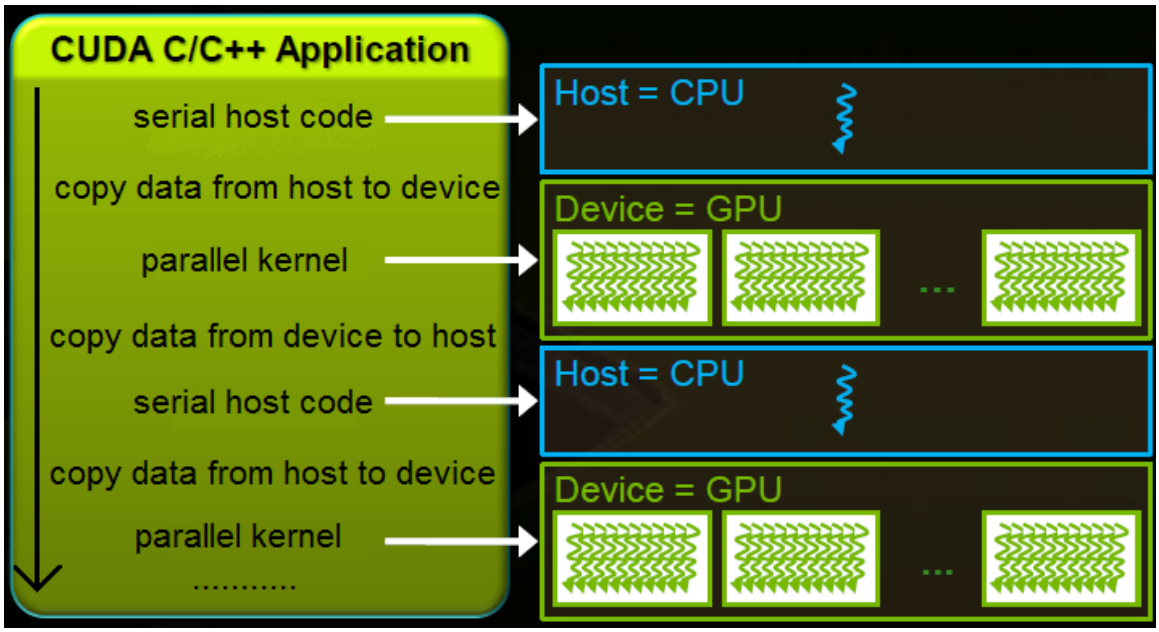
Με την ανάπτυξη της αρχιτεκτονικής CUDA (2006) η Nvidia εισήγαγε ένα ολοκληρωμένο μοντέλο προγραμματισμού με το ίδιο όνομα. Η CUDA προσφέρει μία διεπιφάνεια για την απόκρυψη (abstraction) πολλών τεχνικών λεπτομερειών της λειτουργίας της GPU. Ταυτόχρονα επιτρέπει μεγάλη ευελιξία στο χειρισμό των διαφόρων ειδών μνήμης (κανονικής και cache) και των πυρήνων της GPU. Έτσι ο προγραμματιστής εργάζεται σε χαμηλότερο προγραμματιστικό επίπεδο από ότι π.χ. στο OpenACC, οπότε μπορεί να εφαρμόσει περισσότερες βελτιστοποιήσεις. Έτσι αυξάνεται σημαντικά η απόδοση, αλλά ο προγραμματισμός είναι πιο πολύπλοκος. Η CUDA περιλαμβάνει ένα οικοσύστημα εργαλείων προγραμματισμού (development toolkit), όπως τη γλώσσα CUDA C που βασίζεται στις πλέον δημοφιλείς γλώσσες C/C++, ειδικό μεταγλωττιστή (compiler) που ονομάζεται NVCC, ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDE) με το όνομα NSight και βιβλιοθήκες για τις πλέον συνηθισμένες GPGPU εφαρμογές, π.χ. γραμμικής άλγεβρας. Τα προγράμματα που γράφονται για αρχιτεκτονικές CUDA κάποιας γενιάς χρειάζονται λίγες ή καθόλου αλλαγές για βελτιστοποιημένη εκτέλεση σε μεταγενέστερες γενιές. Έχουν δηλαδή καλό performance scaling.

Η CUDA όμως έχει ένα πολύ σημαντικό μειονέκτημα. Αφορά μόνο κάρτες γραφικών της Nvidia. Επιπλέον οι βιβλιοθήκες της είναι κλειστές (proprietary software). Για την γενίκευση των τεχνικών που εφαρμόζει η CUDA σε GPU άλλων εταιριών, αλλά και σε CPU και άλλα είδη επεξεργαστών, ο όμιλος Khronos ανέπτυξε το μοντέλο OpenCL (Open Computing Language) το 2009. Από την οπτική γωνία του προγραμματιστή, υπάρχουν πολλές ομοιότητες ανάμεσα σε CUDA και OpenCL. Είναι ελαφρώς πιο περίπλοκο, αφού χρειάζεται χαμηλότερη γνώση της αρχιτεκτονικής του επεξεργαστή για τον οποίο προορίζεται ένα πρόγραμμα. Η γλώσσα προγραμματισμού είναι βασισμένη στην C. Τα προγράμματα που γράφονται σε OpenCL για μία συσκευή (CPU ή GPU οποιαδήποτε εταιρίας κατασκευής) μπορούν να μεταφερθούν (port) με ελάχιστες τροποποιήσεις σε μία άλλη. Δυστυχώς αυτό αφορά μόνο την λειτουργία και όχι την απόδοσή τους. Κώδικας που έχει βελτιστοποιηθεί για μία συσκευή συνήθως πρέπει να βελτιστοποιηθεί από την αρχή για μία άλλη.

Στην εργασία αυτή χρησιμοποιείται η CUDA για τον προγραμματισμό σε ετερογενές σύστημα CPU/GPU. Στην συνέχεια παρουσιάζονται πολύ συνοπτικά τα σημαντικότερα χαρακτηριστικά της CUDA. Σκοπός είναι να πάρει ο αναγνώστης μία βασική ιδέα για τις λεπτομέρειες που υπεισέρχονται όταν γράφονται προγράμματα για GPU. Τέτοιες λεπτομέρειες αποκρύπτονται στις σύγχρονες υψηλές γλώσσες προγραμματισμού για CPU, οπότε το επίπεδο δυσκολίας είναι σημαντικά ανεβασμένο. Όλα τα επόμενα ισχύουν σχεδόν αυτούσια και για το μοντέλο OpenCL, ενώ υπάρχουν ανάλογες λειτουργίες στα OpenMP και OpenACC.

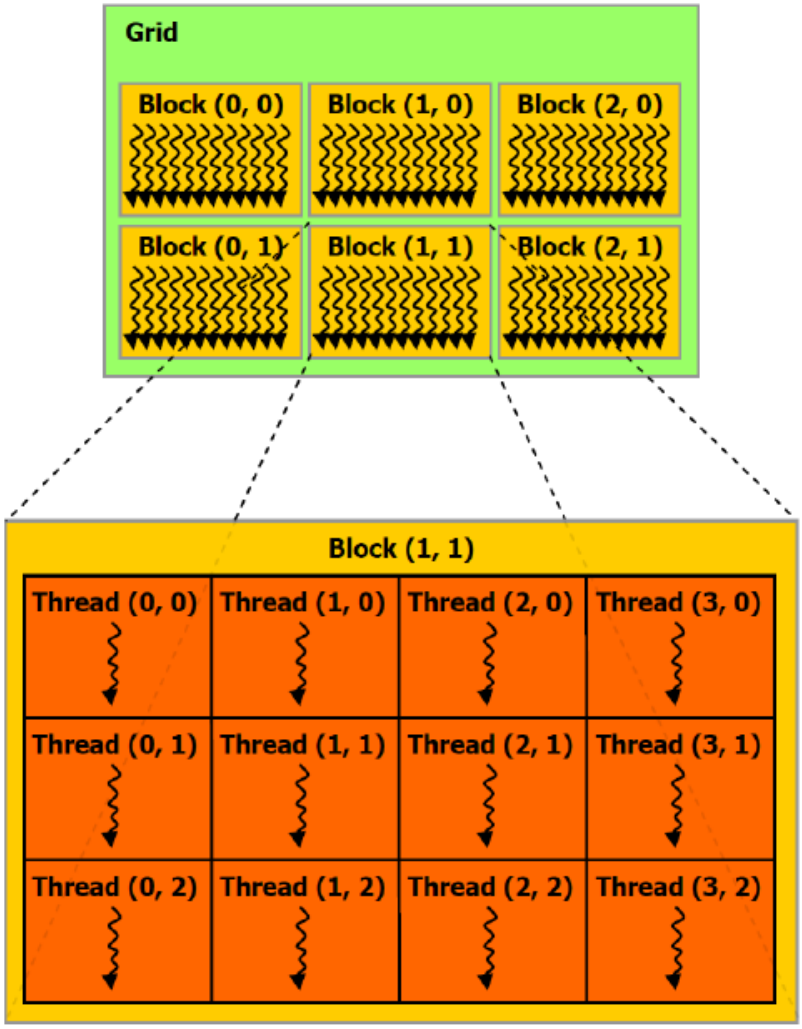
4.4.1 Ροή ελέγχου σε προγράμματα CUDA

Σε ένα ετερογενές υπολογιστικό σύστημα υπάρχει ο επεξεργαστής ***host***, που είναι υπεύθυνος για την κύρια ροή του προγράμματος, και ένας ή περισσότεροι επεξεργαστές ***devices***, που εκτελούν συμπληρωματικά μέρη του προγράμματος σε διαφορετικά threads από αυτό του host. Στην περίπτωση 1CPU & 1GPU, host θεωρείται η CPU και device η GPU. Τα μέρη του προγράμματος που εκτελούνται από το host και το device αναφέρονται ως ***host code*** και ***kernel*** αντίστοιχα Στην CUDA C η μεταγλώττιση και των δύο αναλαμβάνεται από τον NVCC compiler. Το host code μεταγλωττίζεται τελικά από τον C compiler που έχει εγκαταστήσει ο χρήστης (π.χ. GCC για linux), ενώ οι kernels μεταγλωττίζονται από τον Just-In-Time Compiler του NVCC. Στο OpenCL η μεταγλώττιση των kernels είναι πιο περίπλοκη, αφού δεν γίνεται με τον ίδιο τρόπο σε όλες τις GPU ή τα άλλα είδη επεξεργαστών που υποστηρίζονται. Κατά την εκτέλεση ενός τυπικού προγράμματος: εκτελείται σειριακά το host code, μεταφέρονται δεδομένα από την κεντρική RAM του συστήματος στην DRAM της κάρτας γραφικών, εκτελείται ο kernel με πολλά παράλληλα threads και μεταφέρονται τα αποτελέσματα από την κάρτα γραφικών στην κεντρική μνήμη. Τα παραπάνω βήματα επαναλαμβάνονται μέχρι να τελειώσει το πρόγραμμα.



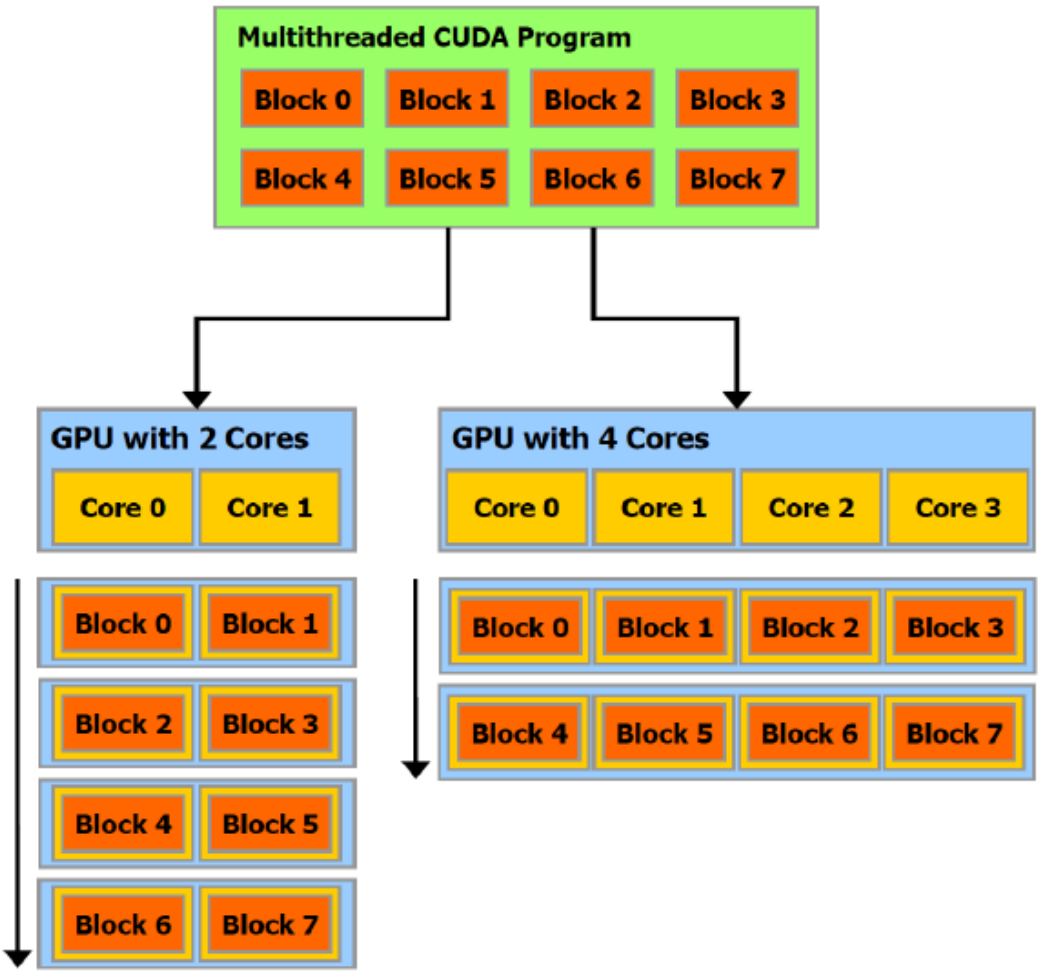
Σχήμα 4.13 Εκτέλεση προγράμματος σε ετερογενές σύστημα CPU & GPU

Μία βασική ακολουθία εντολών που εκτελούνται στην GPU ονομάζεται ***thread*** όπως και στα περισσότερα μοντέλα παράλληλου προγραμματισμού. Κάθε εντολή εκτελείται παράλληλα από χιλιάδες threads (massively parallel), καθένα από τα οποία την εφαρμόζει σε διαφορετικές θέσεις μνήμης σύμφωνα με την SIMD παραλληλία. Τα threads οργανώνονται σε ομάδες που ονομάζονται ***blocks***. Κάθε block μπορεί να έχει 1, 2 ή 3 διαστάσεις ώστε να οργανώνει τα threads με τρόπο που διευκολύνει τον προγραμματισμό της εκάστοτε ιδέας. Για παράδειγμα, ένα 1D block εξυπηρετεί διανύσματα, ενώ ένα 2D block εξυπηρετεί πίνακες. Το σύνολο όλων των block ονομάζεται ***grid***. Το grid μπορεί να αποτελείται από blocks σε 1, 2 ή 3 διαστάσεις στις πιο καινούργιες αρχιτεκτονικές, ανεξάρτητα από την διάσταση του κάθε block. Ο μέγιστος αριθμός των thread που μπορούν να υπάρχουν σε κάθε block είναι 512 ή 1024 ανάλογα με το πόσο σύγχρονη είναι η GPU. Ο μέγιστος αριθμός των block ανά διεύθυνση του grid είναι 65535. Στην πραγματικότητα όμως ο μέγιστος αριθμός των thread, ώστε να επιτευχθεί βέλτιστη απόδοση, καθορίζεται από τους διαθέσιμους πόρους της κάθε GPU. Βέβαια παραμένει πολύ μεγάλος. Τα threads του ίδιου block έχουν πρόσβαση σε κοινές θέσεις cache μνήμης, διαφορετικές για κάθε block. Η επικοινωνία των threads που δεν ανήκουν στο ίδιο block γίνεται μέσω της DRAM της κάρτας γραφικών που είναι πιο αργή.



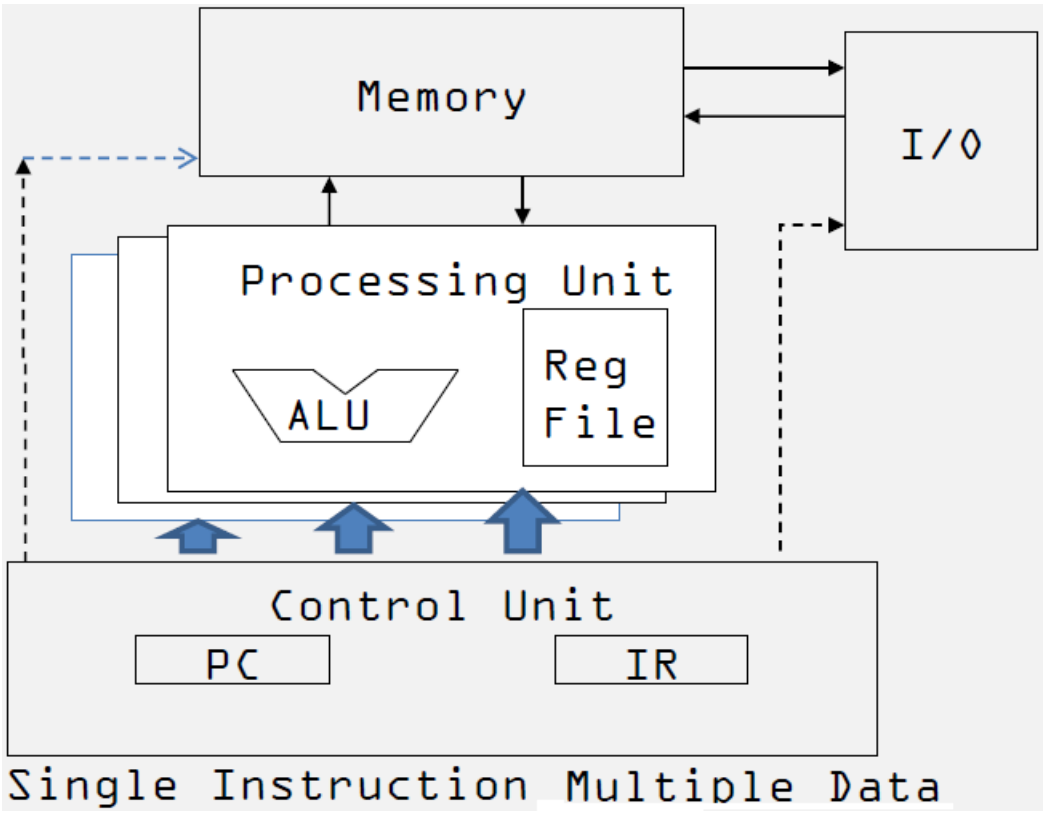
Σχήμα 4.14 Threads, Blocks, Grid

Ο πολύ μεγάλος αριθμός των thread δεν αντιστοιχεί σε εξίσου μεγάλο αριθμό πυρήνων της GPU. Κάθε block ανατίθεται σε κάποιον Streaming Multiprocessor, ανάλογα με την διαθεσιμότητα πόρων και με σκοπό την ελαχιστοποίηση των αναμονών. Έτσι ο κάθε SM των πλέον σύγχρονων GPU μπορεί να έχει έως 32 blocks και 2048 threads, κάθε χρονική στιγμή. Σε άλλες χρονικές στιγμές καινούργια blocks και threads ανατίθενται στον SM. Η διαδικασία αυτή γίνεται αυτόματα, χωρίς τον έλεγχο του προγραμματιστή. Ωστόσο μπορούν να γίνουν βελτιστοποιήσεις σχετικά με την μνήμη cache και τους registers του SM. Πάντως η αυτόματη ανάθεση οσοδήποτε μεγάλου grid (software) στους πυρήνες της GPU (hardware), επιτρέπει την αυτόματη αύξηση της απόδοσης των προγραμμάτων, όταν αυξάνεται ο αριθμός των πυρήνων σε νέες GPU (transparent performance scaling). Έτσι, εγγυάται η συνεχής βελτίωση των GPGPU λογισμικών ακόμα και αν δεν γίνει καμία βελτιστοποίηση που να αξιοποιεί τις νέες τεχνολογικές δυνατότητες μεταγενέστερων καρτών γραφικών.



Σχήμα 4.15 Αυτόματος καταμερισμός των thread

Ακόμα και εντός του ίδιου block, όλα τα threads δεν εκτελούνται παράλληλα. Χωρίζονται σε ομάδες των 32 που ονομάζονται ***warps***. Τα warps αυτά συντονίζονται από τον Streaming Multiprocessor σε επίπεδο hardware μέσω των warp scheduler. Έτσι δημιουργούνται pipelines, στα οποία π.χ. οι προσπελάσεις μνήμης των threads ενός warp εκτελούνται ταυτόχρονα με τις αριθμητικές πράξεις σε ένα άλλο, αντίστοιχα με την παραλληλία σε επίπεδο εντολών που εφαρμόζουν οι CPU. Και τα 32 threads ενός warp εκτελούνται την ίδια ακριβώς χρονική στιγμή. Έτσι οι προσπελάσεις μνήμης που γίνονται από αυτά τα threads είναι επίσης ταυτόχρονες και μπορούν να γίνουν βελτιστοποιήσεις όπως θα δούμε στη συνέχεια. Κάθε warp μπορεί να θεωρηθεί ως ένας επεξεργαστής Von Neumann που λειτουργεί σε SIMD, δηλαδή η κάθε εντολή του προγράμματος διαβάζεται μία φορά από τη μνήμη και εκτελείται ταυτόχρονα από τα 32 threads.



Σχήμα 4.16 Θεώρηση warp ως SIMD επεξεργαστή Von Neumann

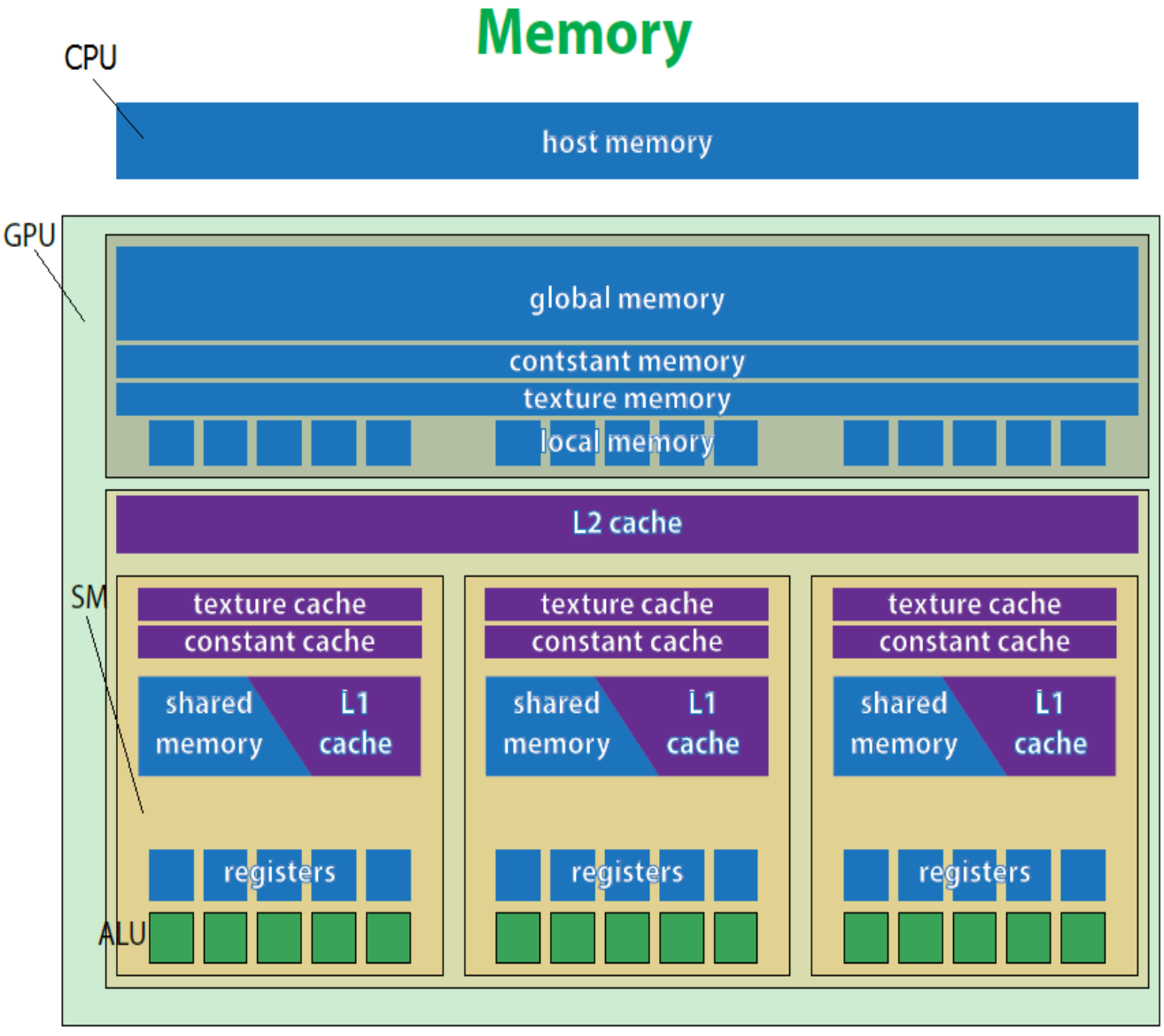
Αυτό δημιουργεί ένα σημαντικό πρόβλημα: τι γίνεται όταν υπάρχει κάποια διακλάδωση στη ροή του προγράμματος, δηλαδή κάποιο branch *if…then…else…* ή κάποιος βρόγχος. Για παράδειγμα, δίνεται ο κώδικας **(4.7)**. Τα threads όλου του grid αριθμούνται με κάποιον φυσικό αριθμό που λέγεται thread id. Τα threads που έχουν id<100 εκτελούν παράλληλα την ομάδα εντολών Α. Αφού τελειώσουν, τα υπόλοιπα threads εκτελούν την ομάδα εντολών Β.

$$\begin{aligned} &if\ (thread\ id < 100)\ then \\ &\{ομάδα\ εντολών\ A\} \\ &else \\ &\{ομάδα\ εντολών\ B\} \end{aligned} \qquad \textbf{(4.7)}$$

Έχουμε λοιπόν διαχωρισμό των threads του warp σε ομάδες. Η κατάσταση αυτή ονομάζεται ***control divergence***. Κάθε ομάδα thread εκτελείται παράλληλα, αλλά οι δύο ομάδες εκτελούνται η μία μετά την άλλη, δηλαδή σειριακά. Όσο περισσότερες διακλαδώσεις έχει ένα πρόγραμμα, τόσο περισσότερο σειριακός γίνεται ο αλγόριθμος, με αποτέλεσμα να υπάρχουν σοβαρές καθυστερήσεις. Βέβαια, η ύπαρξη διακλάδωσης ή βρόγχου στον κώδικα δεν συνεπάγεται αυτόματα control divergence. Είναι πολύ πιθανό, όλα τα 32 threads ενός warp να ακολουθούν την ίδια ροή ελέγχου, π.χ. όλα να ακολουθούν το *else*. Έτσι, ο προγραμματιστής προσπαθεί να περιορίσει όσο το δυνατόν περισσότερο τον αριθμό των warp στα οποία υπάρχει πραγματικά control divergence.

4.4.2 Χειρισμός μνήμης σε προγράμματα CUDA

Η υπολογιστική ισχύς που προσφέρει ο μεγάλος αριθμός threads, σημαίνει ότι ο κρισιμότερος παράγοντας καθυστέρησης (bottleneck) είναι οι προσπελάσεις μνήμης. Για τον λόγο αυτό η CUDA επιτρέπει την χρήση διαφόρων ειδών μνήμης cache, που μπορούν να ελαττώνουν σημαντικά το συνολικό χρόνο προσπελάσεων. Ο βέλτιστος χειρισμός αυτών των cache είναι απαραίτητος για την ανάπτυξη GPGPU προγραμμάτων. Παρακάτω παρουσιάζονται συνοπτικά τα διάφορα είδη μνήμης στην αρχιτεκτονική CUDA.



Σχήμα 4.17 Το μοντέλο μνήμης της αρχιτεκτονικής CUDA

1) Global memory

Είναι η κύρια μνήμη της κάρτας γραφικών και υλοποιείται με DRAM. Σε αυτή έχουν πρόσβαση όλα τα threads όλων των Streaming Multiprocessors της GPU. Η πρόσβαση αυτή μπορεί να είναι απευθείας ή μέσω L2 cache, για τις πιο πρόσφατες κάρτες γραφικών. Η CPU δεν έχει απευθείας πρόσβαση στα περιεχόμενά της, αλλά πρέπει πρώτα να μεταφέρει τα δεδομένα στην host memory. Η global memory έχει συνήθως μέγεθος 768 MB έως 6 GB και ταχύτητες προσπέλασης 100 – 200 GB/s. Αποτελεί την μεγαλύτερη και αργότερη μνήμη της κάρτας γραφικών. Δεν βρίσκεται πάνω στο κύκλωμα της GPU, αλλά περιμετρικά της πάνω στην κάρτα γραφικών. Επειδή μπορεί να προσπελαστεί από όλα τα threads, είναι πιθανόν το εύρος (bandwidth) των διαύλων μεταφοράς δεδομένων να μην επαρκεί για όλα τα threads που προσπαθούν ταυτόχρονα να αναγνώσουν ή να γράψουν σε αυτήν. Έτσι κάποια από αυτά αναγκάζονται να περιμένουν. Γενικά χρειάζεται προσοχή στην κατανάλωση του bandwidth της global memory.

Ο προγραμματιστής μπορεί να διαβάσει ή να γράψει σε μία οποιαδήποτε θέση μνήμης της global memory μέσα από οποιοδήποτε thread. Από την οπτική γωνία του προγραμματιστή, μπορεί να είναι στατική (static memory) ή δυναμική μνήμη (dynamic memory). Σε ένα πρόγραμμα στατική μνήμη είναι αυτή που το μέγεθός της είναι γνωστό την στιγμή της μεταγλώττισης (compile time) και σταθερό κατά τη διάρκεια εκτέλεσης (runtime). Το πρόγραμμα μπορεί να την προσπελάσει μέσα από τις μεταβλητές του, οι οποίες καταλαμβάνουν τον αντίστοιχο χώρο στη RAM. Η δυναμική μνήμη αντίθετα μπορεί να έχει οποιοδήποτε μέγεθος το οποίο δεν είναι γνωστό τη στιγμή της μεταγλώττισης. Αυτό επιτρέπει στο πρόγραμμα να χρησιμοποιεί όση μνήμη χρειάζεται κάθε χρονική στιγμή και μετά να την αποδεσμεύει. Η πρόσβαση στην dynamic memory γίνεται μέσω ειδικών μεταβλητών που λέγονται ***pointers***. Αυτοί πιάνουν ελάχιστο χώρο και περιέχουν «βέλη» που «δείχνουν» σε διευθύνσεις μνήμης της RAM, οι οποίες αντιστοιχούν στα δεδομένα της δυναμικής μνήμης. Στην περίπτωση της global memory, κάθε thread έχει μεταβλητές για την στατική μνήμη και pointers για την δυναμική. Στην CUDA C μία μεταβλητή που αφορά την global memory δηλώνεται με το πρόθεμα ***__device__.*** Οι pointers είναι πολύ πιο σύνθετο θέμα από ότι παρουσιάστηκαν εδώ και για να τους καταλάβει κανείς χρειάζεται να ξέρει τη γλώσσα C ή κάποια άλλη που έχει βασιστεί στην C.

2) Host memory

Είναι η κεντρική μνήμη του συστήματος και χρησιμοποιείται από τη CPU. Συνήθως έχει μέγεθος 4-16 GB, με τους πιο ακριβούς υπολογιστές να φτάνουν τα 64 GB, και ταχύτητα προσπέλασης 24-32 GB/s. Είναι μεγαλύτερη και αργότερη από την global memory της GPU. Βρίσκεται πάνω στη μητρική κάρτα, οπότε η CPU έχει απευθείας πρόσβαση σε αυτήν. Η GPU όμως πρέπει να μεταφέρει τα δεδομένα που χρειάζεται από την host memory στην global memory πριν τα χρησιμοποιήσει. Οι μεταφορές δεδομένων μεταξύ των host και global memory δηλώνονται στο host code, ανεξαρτήτως φοράς (host→device ή device→host). Σε επίπεδο υλικού γίνονται μέσω του διαύλου PCI-Express, αφού η GPU δεν βρίσκεται πάνω στη motherboard. Η ταχύτητα

προσπέλασης του PCI-Express είναι πολύ μικρότερη (8-15 GB/s) και υπόκειται σε σημαντικές καθυστερήσεις (overheads), οπότε ο προγραμματιστής πρέπει να αποφεύγει τη μεταφορά μεγάλου όγκου δεδομένων και να σχεδιάζει τις μεταφορές ώστε να μην είναι διεσπαρμένες σε πολλά σημεία του κώδικα. Οι μνήμες cache, που χρησιμοποιεί η CPU, βρίσκονται πάνω στο κύκλωμά της, οπότε τα threads της GPU δεν έχουν καμία πρόσβαση σε αυτές.

3) Registers

Οι registers (καταχωρητές) αποτελούν το ταχύτερο είδος μνήμης στην GPU όπως και στην CPU. Βρίσκονται πάνω στο κύκλωμα της GPU σε αντίθεση με την global memory που βρίσκεται σε άλλο σημείο της κάρτας γραφικών. Σε μία τυπική GPU, η ταχύτητα προσπέλασής τους είναι τουλάχιστον 2 τάξεις μεγέθους μεγαλύτερη από την global memory. Κάθε SM έχει τους δικούς του registers και τους διανέμει στα διάφορα threads που εκτελούνται σε αυτόν. Για την εξυπηρέτηση τόσο μεγάλου αριθμού threads απαιτούνται πολλοί registers, οπότε καταλαμβάνουν αναλογικά μεγαλύτερο μέρος του κυκλώματος από τους registers της CPU.

Αντιστοιχούν στο Register File του επεξεργαστή Von Neumann. Τα δεδομένα που διαβάζονται από την global memory, τοποθετούνται προσωρινά στους registers πριν εκτελεστούν αριθμητικές ή λογικές πράξεις πάνω σε αυτά. Τα αποτελέσματα των πράξεων τοποθετούνται επίσης στους registers και στη συνέχεια μεταφέρονται πίσω στην global memory. Οι μεταφορές αυτές δεν καταναλώνουν άσκοπα χρόνο, αφού γίνονται μέσω pipelines. Η όλη διαδικασία γίνεται αυτόματα, χωρίς κανέναν έλεγχο από τον προγραμματιστή, όπως και στη CPU. Κάθε thread έχει τους δικούς του registers στους οποίους μόνο αυτό έχει πρόσβαση. Εκτός από προσωρινές θέσεις μνήμης, χρησιμοποιούνται και για την αποθήκευση των ιδιωτικών μεταβλητών του thread, που δεν όμως δεν είναι arrays. Για παράδειγμα, αν σε ένα thread δηλώνεται “int a;”, χωρίς κανένα πρόθεμα που να σηματοδοτεί άλλο είδος μνήμης (π.χ. __shared__), τότε η ακέραια μεταβλητή ‘a’ αποθηκεύεται στους registers του thread. Με τον τρόπο αυτό, ο προγραμματιστής αλληλεπιδρά με τους registers, κάτι που δεν γινόταν στην CPU. Χρειάζεται λοιπόν προσοχή να μην δηλώνονται πολλές ιδιωτικές μεταβλητές σε κάθε thread, γιατί ο αποθηκευτικός χώρος των registers εξαντλείται γρήγορα.

4) Local memory

Η local memory αποτελεί λογική αφαίρεση και όχι κάποιο πραγματικό είδος μνήμης από πλευράς hardware. Στις παλαιότερες κάρτες γραφικών, η local memory υλοποιούταν εντός της global memory, οπότε η ταχύτητα προσπέλασής της ήταν το ίδιο αργή και συνεισέφερε στο bottleneck που περιγράφηκε παραπάνω: πάρα πολλά threads προσπαθούν ταυτόχρονα να μεταφέρουν περισσότερα δεδομένα από το bandwidth της global memory. Παρ’ όλα αυτά ήταν γρηγορότερη από την global memory, γιατί οι θέσεις μνήμης των δεδομένων των threads ήταν συνεχόμενες (coalesced), κάτι που εξηγείται στη συνέχεια. Στις νεότερες αρχιτεκτονικές CUDA (≥2.0) η local memory υλοποιείται πάνω στο κύκλωμα (on-chip) της GPU και αποτελεί μέρος της L1 cache του SM στον οποίον εκτελείται το κάθε thread. Έτσι είναι πολύ γρηγορότερη.

Όπως και στους registers, κάθε thread έχει την δικιά του local memory, στην οποία μόνο αυτό έχει πρόσβαση. Η χρήση της είναι διπλή. Ο χώρος των registers μπορεί εύκολα να καταναλωθεί από δεδομένα, οπότε να μην μπορούν να αποθηκευτούν νέα. Όταν απαιτηθεί η προσωρινή αποθήκευση νέων δεδομένων, τότε κάποια από τα υπάρχοντα μεταφέρονται στην local memory προσωρινά πάλι. Η διαδικασία αυτή ονομάζεται register spill και γίνεται αυτόματα, αλλά μπορεί να προκαλέσει σημαντικές καθυστερήσεις ιδιαίτερα αν η local memory βρίσκεται off-chip. Επιπλέον, η local memory χρησιμοποιείται, όπως και οι registers, για την αποθήκευση των ιδιωτικών μεταβλητών του κάθε thread. Σε αυτήν όμως αποθηκεύονται τα ***arrays*** (που πιάνουν περισσότερο χώρο), ενώ στους registers αποθηκεύονταν μόνο οι απλές μεταβλητές. Επομένως, ο προγραμματιστής αλληλεπιδρά και με την local memory. Η local memory είναι στατική μνήμη από πλευράς προγραμματισμού (βλέπε παραπάνω τη εξήγηση στην global memory).

5) Shared memory

Η shared memory είναι μνήμη κοινή για όλα τα threads του ίδιου block. Είναι η δεύτερη πιο συχνά χρησιμοποιούμενη από τον προγραμματιστή μνήμη, μετά την global memory. Η βασική χρήση της είναι για την ανταλλαγή δεδομένων μεταξύ των threads του ίδιου block. Επιπλέον όσα δεδομένα χρησιμοποιούνται πολλές φορές από τα threads του ίδιου block, μπορούν να μεταφερθούν στην shared memory για γρηγορότερη προσπέλαση και μείωση της κατανάλωσης του bandwidth της global memory. Λειτουργεί δηλαδή ως μνήμη cache, η οποία όμως ρυθμίζεται με απόλυτο έλεγχο του προγραμματιστή. Πράγματι, η shared memory υλοποιείται στην ίδια περιοχή του κυκλώματος της GPU με την L1 cache. Η δήλωση των μεταβλητών και arrays της shared memory γίνεται με το πρόθεμα __shared__. Μπορεί να είναι στατική ή δυναμική, αλλά στη δεύτερη περίπτωση υπάρχουν κάποιες δυσχέρειες χειρισμού της.

Η shared memory υλοποιείται πάνω στο κύκλωμα της GPU. Κάθε Streaming Multiprocessor έχει τη δικιά του shared memory και την διανέμει στα block των threads που εκτελούνται σε αυτόν. Κάθε block έχει πρόσβαση μόνο στο δικό του μέρος από την συνολική shared memory του SM. Είναι η δεύτερη πιο γρήγορη μνήμη της GPU, μετά τους registers. Συνήθως η ταχύτητά της είναι 15x υψηλότερη από την global memory, αλλά υπό συγκεκριμένες συνθήκες μπορεί να φτάσει κοντά στην ταχύτητα των registers. Είναι όμως περιορισμένη σε μέγεθος: 16 – 96 kB σε κάθε SM. Το μέρος της shared memory που δεν χρησιμοποιεί ο προγραμματιστής ενσωματώνεται αυτόματα στην L1 cache.

6) L1 & L2 caches

Οι L1 και L2 caches λειτουργούν όπως ακριβώς οι caches των CPU: τα δεδομένα που διαβάζονται από την global memory τοποθετούνται στην L1/L2 cache, ώστε αν χρειαστούν ξανά στη συνέχεια να προσπελαστούν πολύ γρηγορότερα. Όταν ένα thread προσπαθεί να προσπελάσει δεδομένα της global memory, αυτά αναζητούνται πρώτα στην L1 cache. Αν δε βρεθούν εκεί (cache miss), αναζητούνται στην L2 cache και τέλος στην global memory. Τόσο η L1 όσο και η L2 βρίσκονται πάνω στο κύκλωμα της GPU. Είναι γρηγορότερες από την global memory και η χρήση τους μειώνει την χρήση του bandwidth της global memory.

Η L2 cache είναι κοινή για όλα τα threads, ενώ κάθε SM έχει την δικιά του L1 cache. Έχουν μικρό μέγεθος: 16 – 48 kB L1 cache και 512 – 768 kB L2 cache. Ο προγραμματιστής έχει ελάχιστο έλεγχο: μπορεί να διαλέγει τον τρόπο συμπεριφοράς τους κατά τη στιγμή της μεταγλώττισης. Ειδικά για την L1, μπορεί να καθορίζει το μέγεθός της σε σχέση με την shared memory: π.χ 16 kB shared memory και 48 kB L1 cache ή 48 kB shared memory και 16 kB L1 cache.

7) Constant memory, Constant cache και Read-only data cache

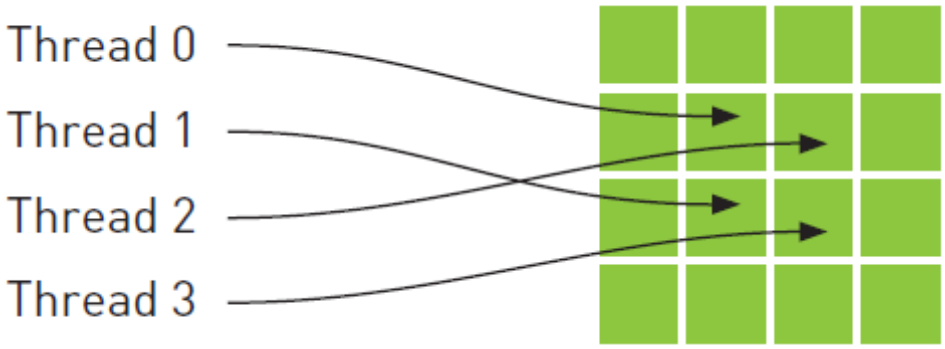
Η constant memory είναι read only μνήμη που δεν αλλάζει κατά την εκτέλεση ενός kernel. Χρησιμοποιείται μόνο για είσοδο δεδομένων στους kernels. Ο προγραμματιστής δηλώνει την constant memory στο host code με το πρόθεμα __constant__. Είναι στατική μνήμη, δηλαδή έχει προκαθορισμένο μέγεθος, αλλά το περιεχόμενό της μπορεί να αλλάξει, αρκεί η αλλαγή αυτή να γίνει στο host code, όχι μέσα σε kernel. Από πλευράς hardware, η constant memory είναι πιο αργή από την shared memory. Υλοποιείται ως ένα τμήμα της global memory, εκτός του κυκλώματος της GPU, και έχει μέγεθος 64 kB. Στην constant memory έχουν πρόσβαση (πιθανόν ταυτόχρονη) όλα τα threads. Η αύξηση της απόδοσης της επιτυγχάνεται μέσω της constant cache. Κάθε Streaming Multiprocessor διαθέτει 8 kB δικής του constant cache. Όταν κάποιο thread προσπαθεί να αναγνώσει για πρώτη φορά δεδομένα από την constant memory, αυτά μεταφέρονται στην constant cache του SM που φιλοξενεί το thread. Οι επόμενες αναγνώσεις των δεδομένων αυτών γίνεται από την constant cache, που είναι πολύ γρηγορότερη. Επιπλέον, αποφεύγεται η προσπέλαση της constant memory που θα αύξανε την κατανάλωση του bandwidth της global memory, όπου και βρίσκεται.

Η χρήση της constant memory έχει ένα ακόμα πλεονέκτημα. Τα δεδομένα που διαβάζονται από αυτήν μεταφέρονται ταυτόχρονα σε όλα τα threads ενός half warp. Ένα half warp αποτελείται από 16 ομαδοποιημένα threads ενός warp. Εδώ χρειάζεται πολλή προσοχή από τον προγραμματιστή. Αν και τα 16 threads ζητήσουν τα ίδια δεδομένα τότε θα αποφευχθούν 15 αναγνώσεις. Αν όμως καθένα από τα 16 threads ζητήσει διαφορετικά δεδομένα, τότε θα γίνουν 16 αναγνώσεις σειριακά επιβραδύνοντας σε μεγάλο βαθμό αυτό το κομμάτι του kernel. Αυτό συμβαίνει επειδή η ανάγνωση της cache memory μπορεί να γίνει μόνο από ένα thread κάθε χρονική στιγμή, σε αντίθεση με την global memory(και άλλες μνήμες) που έχει αρκετά μεγάλο bandwidth, ώστε να προσπελαστεί ταυτόχρονα από πολλά threads.

Η Read-only data cache είναι ανεξάρτητη από τις constant memory και constant cache. Κάθε SM (αρχιτεκτονικής CUDA 3.5 και μετά) διαθέτει 48 kB cache στο κύκλωμα της GPU. Αυτή χρησιμοποιείται όταν ο compiler αποφασίσει ότι κάποια μεταβλητή δεν θα αλλάξει κατά την διάρκεια εκτέλεσης του kernel. Ο προγραμματιστής συχνά χρειάζεται να χρησιμοποιήσει λέξεις κλειδιά όπως *const* ή *__restrict__* στο signature του kernel ή το directive *__ldg()* στο εσωτερικό του, ώστε να κατευθύνει τον compiler. Τα υπόλοιπα αναλαμβάνονται από τον compiler. Η read-only data cache χρησιμοποιείται λοιπόν ευκολότερα από την constant memory, αλλά δεν είναι το ίδιο γρήγορη.

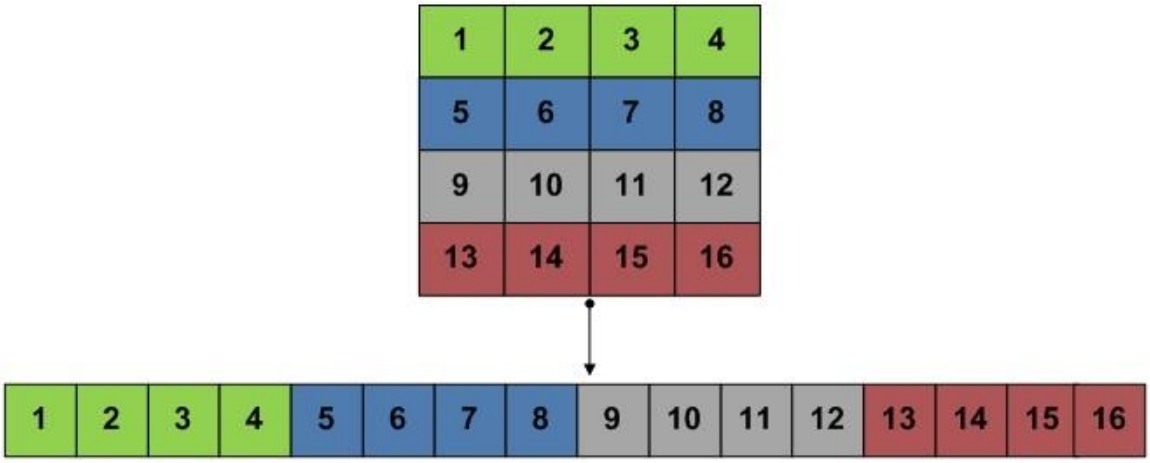
8) Texture memory & Texture cache

Η texture memory είναι και αυτή read-only μνήμη, που μπορεί να χρησιμοποιηθεί για να βελτιώσει την απόδοση, όταν διαβάζονται θέσεις μνήμης που παρουσιάζουν κάποια «γεωμετρική» συγγένεια. Αναπτύχθηκε αρχικά για τη χρήση σε γραφικά, αλλά υπάρχουν αρκετές άλλες περιπτώσεις που μπορεί να χρησιμοποιηθεί. Συνήθως η texture memory επιστρατεύεται για την ανάγνωση κοντινών δεδομένων όπως στο επόμενο 2D σχήμα:



Σχήμα 4.18 Περίπτωση που προσφέρεται η texture memory

Ας θεωρήσουμε ότι το παραπάνω σχήμα παριστάνει έναν πίνακα. Τα στοιχεία του αποθηκεύονται στη μνήμη του υπολογιστή ως 1D array με πολύ διαφορετική δομή. Η C, άρα και CUDA C, χρησιμοποιεί την row major απεικόνιση, στην οποία τα στοιχεία της ίδιας γραμμής αποθηκεύονται σε διαδοχικές θέσεις, αλλά τα στοιχεία της ίδιας στήλης όχι. Έτσι τα στοιχεία 2D και 3D arrays δεν αποθηκεύονται σε συνεχόμενες θέσεις μνήμης, κάτι που όπως θα δούμε στη συνέχεια αυξάνει σημαντικά την ταχύτητα προσπέλασης της global memory, αλλά και των cache. Αντίθετα, η texture memory έχει σχεδιαστεί ώστε να βελτιστοποιεί προσπελάσεις που παρουσιάζουν την παραπάνω εγγύτητα.



Σχήμα 4.19 Row major απεικόνιση ενός 2D array.

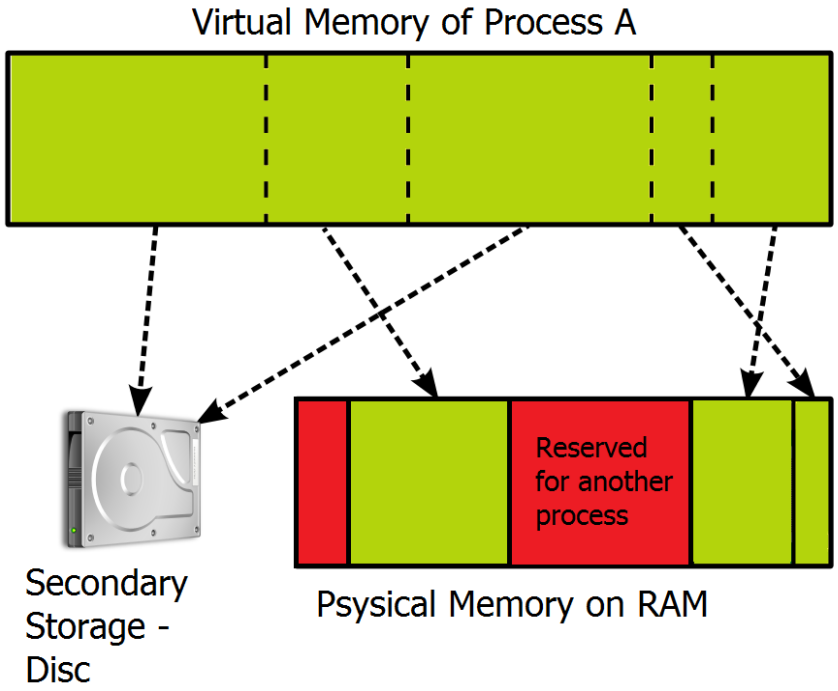
Κάθε Streaming Multiprocessor διαθέτει μία texture cache μεγέθους 6-48 kB. Η ανάγνωση της texture memory γίνεται μέσω της texture cache, όπως και στην constant memory/cache.

Επίσης τα threads του ίδιου warp, που διαβάζουν κοντινές θέσεις μνήμης, έχουν αυξημένη απόδοση όπως γίνεται και με την constant memory.

9) Pinned memory

Η pinned memory ή page-locked memory δεν αφορά την κάρτα γραφικών αλλά την host memory του συστήματος, Κάθε πρόγραμμα έχει το δικό του χώρο μνήμης χωρίς να έχει πρόσβαση στη μνήμη ενός άλλου προγράμματος. Την μνήμη αυτήν την βλέπει ως συνεχή, π.χ. τα διαδοχικά στοιχεία ενός array έχουν διαδοχικές διευθύνσεις μνήμης. Έτσι εξυπηρετείται η χρήση της και απομονώνονται οι χώροι μνήμης κάθε προγράμματος. Η μνήμη αυτή όμως είναι εικονική (virtual memory). Σε επίπεδο υλικού, ο χώρος μνήμης RAM (psysical memory) που διατίθεται σε κάθε πρόγραμμα είναι κερματισμένος. Ειδικό hardware (Memory Management Unit) επί της CPU μεταφράζει τις διευθύνσεις virtual memory σε διευθύνσεις physical memory πριν τις προσπελάσει.

Επιπλέον, η virtual memory μπορεί να χρησιμοποιηθεί από το λειτουργικό σύστημα για να αυξήσει εικονικά την πραγματικά διαθέσιμη μνήμη RAM. Έτσι η συνολική virtual memory που διατίθεται στα προγράμματα είναι μεγαλύτερη από την physical memory που έχει εγκατασταθεί. Για την επιπλέον virtual memory χρησιμοποιείται δευτερεύων αποθηκευτικός χώρος, γεγονός που επιβραδύνει την πρόσβαση σε αυτές τις θέσεις μνήμης. Σήμερα χρησιμοποιούνται κυρίως δύο είδη δευτερεύοντος αποθηκευτικού χώρου, οι Hard Disk Drives (HDD) με ταχύτητες προσπέλασης 50-120MB/s και οι Solid State Drive (SSD) με ταχύτητες ανάγνωσης και εγγραφής 200-600 MB/s. Σε σχέση με τις ταχύτητες προσπέλασης της RAM (20-30 GB/s για τις τελευταίες τεχνολογίες), οι προσπελάσεις αυτές είναι πολύ αργές και φθείρουν τους δίσκους, καθώς δεν έχουν σχεδιαστεί για πολλές εγγραφές δεδομένων, όπως η RAM.



Σχήμα 4.20 Virtual Memory.

Η virtual memory διαιρείται σε pages, που είναι μπλοκ από συνεχόμενες διευθύνσεις μνήμης. Οι διευθύνσεις virtual memory κάθε σελίδας αντιστοιχίζονται σε διευθύνσεις physical memory μέσω των page tables. Κάθε διεύθυνση της virtual memory αναζητείται πρώτα σε αυτόν τον πίνακα, όπου αναγράφεται αν η αντίστοιχη page βρίσκεται στην physical memory. Αν δεν βρίσκεται, τότε ολόκληρο το αντίστοιχο page μεταφέρεται από τον σκληρό δίσκο στην RAM (page in) και έπειτα τα δεδομένα της διεύθυνσης αυτής μεταφέρονται από την RAM στους registers και τις μνήμες cache της CPU. Η page που μόλις διαβάστηκε, πιθανότατα εκτοπίζει κάποια από τις αυτές που βρίσκονταν στην physical memory και τώρα θα μεταφερθεί στον δίσκο (page out). Τα λειτουργικά συστήματα έχουν την δυνατότητα να εξασφαλίζουν ότι για συγκεκριμένες pages που πρέπει να παραμείνουν στην physical memory, δεν θα πραγματοποιηθεί page-out, για όσο χρόνο αυτές χρειάζονται. Οι σελίδες αυτές ονομάζονται pinned, locked, fixed ή wired.

Η μεταφορά δεδομένων από την host (motherboard) στην global memory (κάρτα γραφικών) γίνεται μέσω hardware που ονομάζεται Direct Memory Access (DMA). Σε κάθε μεταφορά τα δεδομένα χωρίζονται σε ομάδες, των οποίων οι διευθύνσεις στην virtual memory μεταφράζονται σε διευθύνσεις physical memory όλες μαζί. Η μετάφραση αυτή γίνεται μόνο μία φορά πριν την εκκίνηση της μεταφοράς αυτής της ομάδας δεδομένων, ώστε να αποφεύγονται επαναλαμβανόμενες μεταφράσεις και αναζητήσεις στον page table και τελικά να εξοικονομείται χρόνος. Έτσι όμως είναι πιθανόν το λειτουργικό σύστημα να κάνει page-out τις σελίδες κάποιων δεδομένων πριν τελειώσει η μεταφορά όλης της ομάδας. Για να αποφευχθεί αυτό, κάθε φορά που μεταφέρονται δεδομένα από την host στην global memory:

- Όσες από τις σελίδες, όπου βρίσκονται οι virtual διευθύνσεις των δεδομένων, δεν υπάρχουν στην physical memory, μεταφέρονται από τον δίσκο στην RAM.
- Όλες οι pages των δεδομένων γίνονται pinned.
- Τα αντίστοιχα δεδομένα μεταφέρονται από την RAM του συστήματος (host memory) στην global memory της κάρτας γραφικών.
- Όταν τελειώσει η αντιγραφή, απελευθερώνονται οι pinned pages αυτές.

Έτσι κάθε αντιγραφή μπορεί να γίνει δύο φορές, οι οποίες ευτυχώς εκτελούνται παράλληλα. Ακόμα και έτσι όμως ο συνολικός χρόνος καθορίζεται από την ταχύτητα προσπέλασης του δίσκου, που είναι πολύ χαμηλότερη από αυτή του PCI-Express (motherboard → κάρτα γραφικών). Ο προγραμματιστής μπορεί να πετύχει επιτάχυνση περίπου 2x, δηλώνοντας τα δεδομένα της host memory που θέλει να μεταφέρει ως pinned memory. Έτσι εξασφαλίζει ότι το λειτουργικό σύστημα θα δεσμεύσει physical memory για τις pages των δεδομένων και δε θα πραγματοποιούνται περιττές page-in αντιγραφές. Αυτό γίνεται στο host code και αφορά την dynamic memory, οπότε η pinned memory μπορεί να ελευθερωθεί σε επόμενα βήματα, όταν ο προγραμματιστής είναι σίγουρος ότι δεν την χρειάζεται. Αυτό είναι απαραίτητο, γιατί όταν αυξάνεται το ποσοστό της pinned memory, η physical memory εξαντλείται πολύ γρήγορα, ακινητοποιώντας έτσι μεταγενέστερα κομμάτια του προγράμματος ή άλλα προγράμματα που εκτελούνται ταυτόχρονα.

Memory	Location on/off chip	Cached	Access	Scope	Lifetime
Register	On	n/a	R/W	1 thread	Thread
Local	Off	†	R/W	1 thread	Thread
Shared	On	n/a	R/W	All threads in block	Block
Global	Off	†	R/W	All threads + host	Host allocation
Constant	Off	Yes	R	All threads + host	Host allocation
Texture	Off	Yes	R	All threads + host	Host allocation
† Cached only on devices of compute capability 2.x.					

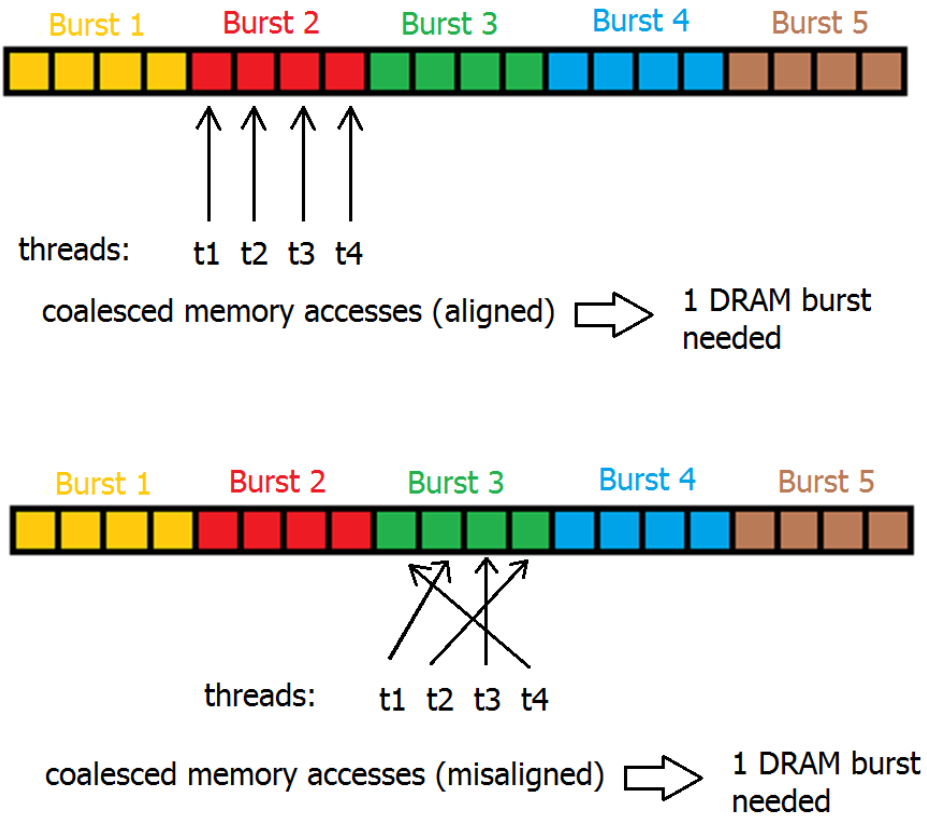
Σχήμα 4.21 Χρήση των διαφόρων ειδών μνήμης της CUDA

Στο παραπάνω σχήμα φαίνονται τα συνηθέστερα είδη μνήμης, το μέρος του κώδικα που έχει πρόσβαση σε αυτά (scope) και η διάρκεια διατήρησης των δεδομένων τους πριν ελευθερωθούν οι αντίστοιχοι πόροι (lifetime). Η γενική λογική του χειρισμού τους είναι να μεταφέρονται τα δεδομένα που χρησιμοποιούνται συχνά στις πιο γρήγορες μνήμες («κοντά» στα ALU), μεριμνώντας να μην αυτές εξαντληθούν από άποψη χώρου ή bandwidth.

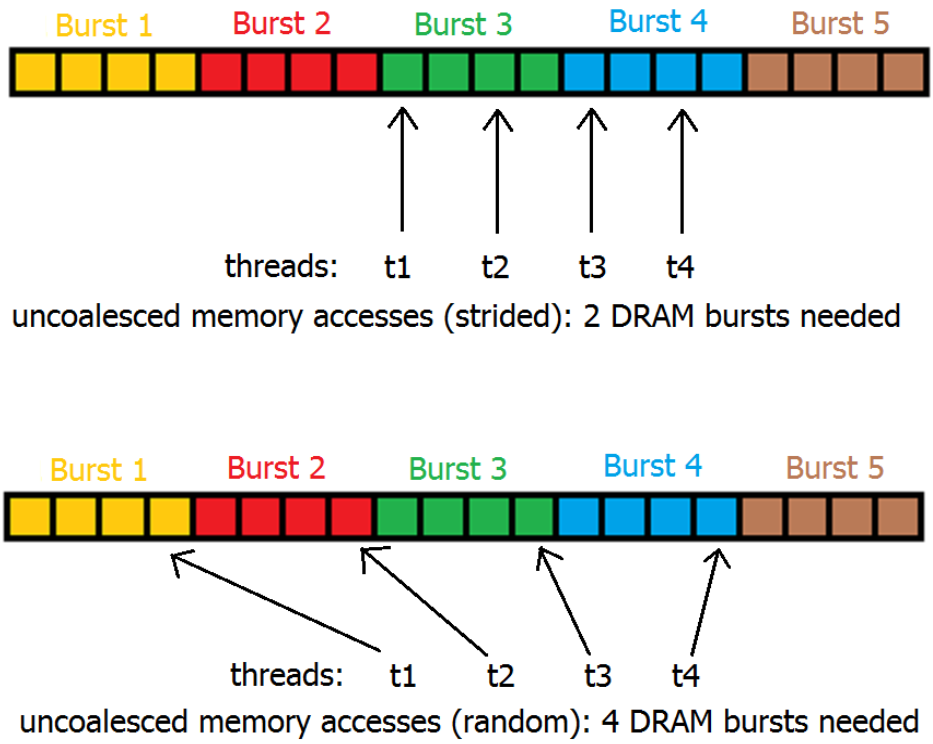
Memory Coalescing

Η global memory υλοποιείται με DRAM πάνω στην κάρτα γραφικών. Η ταχύτητα ανάγνωσης δεδομένων από τα transistor στο εσωτερικό της DRAM είναι N φορές πιο μικρή από την ταχύτητα της διεπιφάνειας, από την οποία διέρχονται πριν μεταφερθούν στην GPU. Ο παράγοντας N μάλιστα μεγαλώνει στις νεότερες τεχνολογίες DRAM. Ως αντιστάθμισμα χρησιμοποιείται η τεχνική DRAM bursting: Σε κάθε ανάγνωση διαβάζονται N φορές περισσότερα δεδομένα από το bandwidth της διεπιφάνειας και τοποθετούνται προσωρινά σε ένα εσωτερικό buffer που είναι πολύ γρηγορότερο. Τα N αυτά δεδομένα είναι συνεχόμενα. Αν οι επόμενες αναγνώσεις που ζητήσει η GPU αφορούν γειτονικά δεδομένα, αυτά ήδη βρίσκονται στον buffer και η μεταφορά γίνεται πολύ ταχύτερα. Αλλιώς τα περιττά δεδομένα που διαβάστηκαν απορρίπτονται και ο buffer γεμίζει ξανά με νέες αναγνώσεις από τα transistor. Έτσι δεν επιτυγχάνεται η ονομαστική ταχύτητα 100-200 GB/s της διεπιφάνειας και τελικά οι προσπελάσεις της global memory καταλήγουν να έχουν περίπου το 1/10 της ταχύτητας.

Τα 32 threads ενός warp προσπελαύνουν ταυτόχρονα δεδομένα από την global memory. Αν τα δεδομένα αυτά είναι αποθηκευμένα σε διαδοχικές θέσεις μνήμης της DRAM, όπως π.χ. διαδοχικά στοιχεία ενός 1D array, τότε χρειάζονται λίγα bursts ώστε να διαβαστούν όλα τα δεδομένα από την DRAM, αφού κάθε burst καλύπτει τις ανάγκες πολλών threads. Λέμε τότε ότι οι προσπελάσεις μνήμης είναι ***coalesced***. Αν όμως τα threads του warp, ζητούν δεδομένα σε μακρινές θέσεις μνήμης, τότε απαιτούνται πολλά bursts, 32 στη χειρότερη περίπτωση, πριν ολοκληρωθεί η ανάγνωση και μπορέσει το warp να συνεχίσει στην επόμενη εντολή.



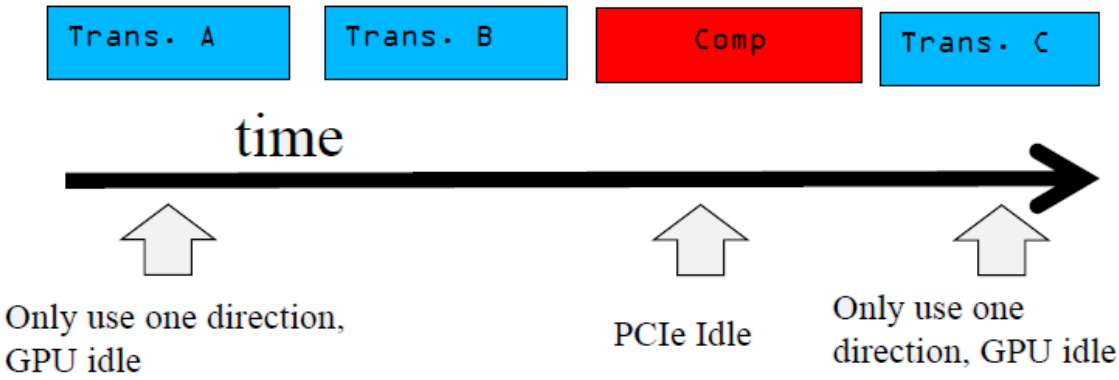
Σχήμα 4.22 Coalesced προσπελάσεις μνήμης (απλούστευση για 4 threads ανά warp)



Σχήμα 4.23 Uncoalesced προσπελάσεις μνήμης (απλούστευση για 4 threads ανά warp)

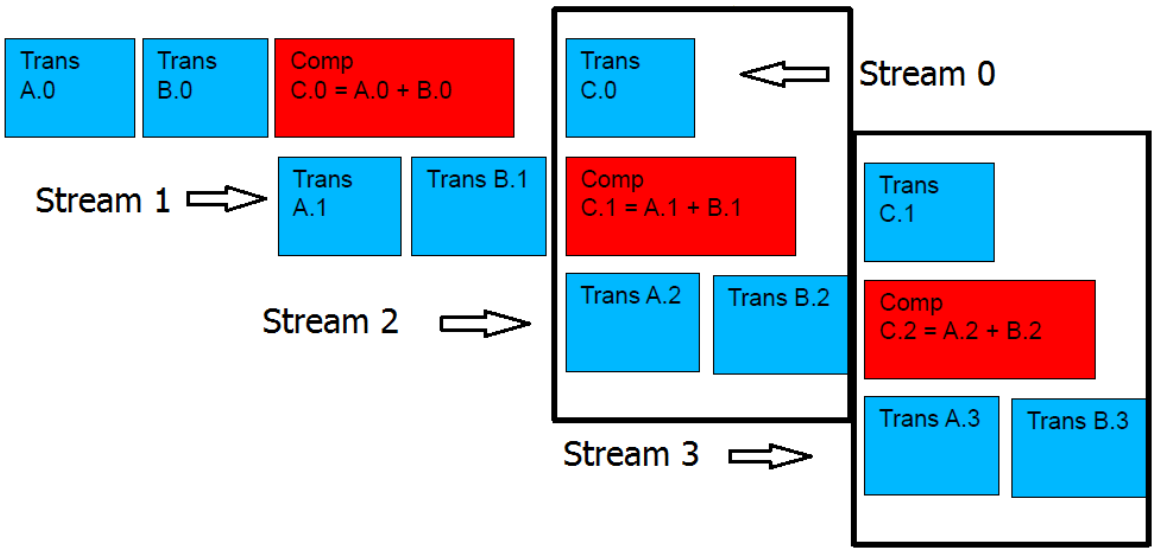
4.4.3 Streams

Σε όσα αναφέρθηκαν προηγουμένως υποτίθεται ότι η αντιγραφή από host στη global memory, η εκτέλεση του kernel και η αντιγραφή των αποτελεσμάτων από τη global στη host memory εκτελούνται η μία μετά την άλλη, σειριακά. Μία ποιοτική απεικόνιση της σειριακής εκτέλεσης αυτών των φάσεων φαίνεται στο παρακάτω σχήμα.



Σχήμα 4.24 Σειριακή εκτέλεση kernel και μεταφορές δεδομένων

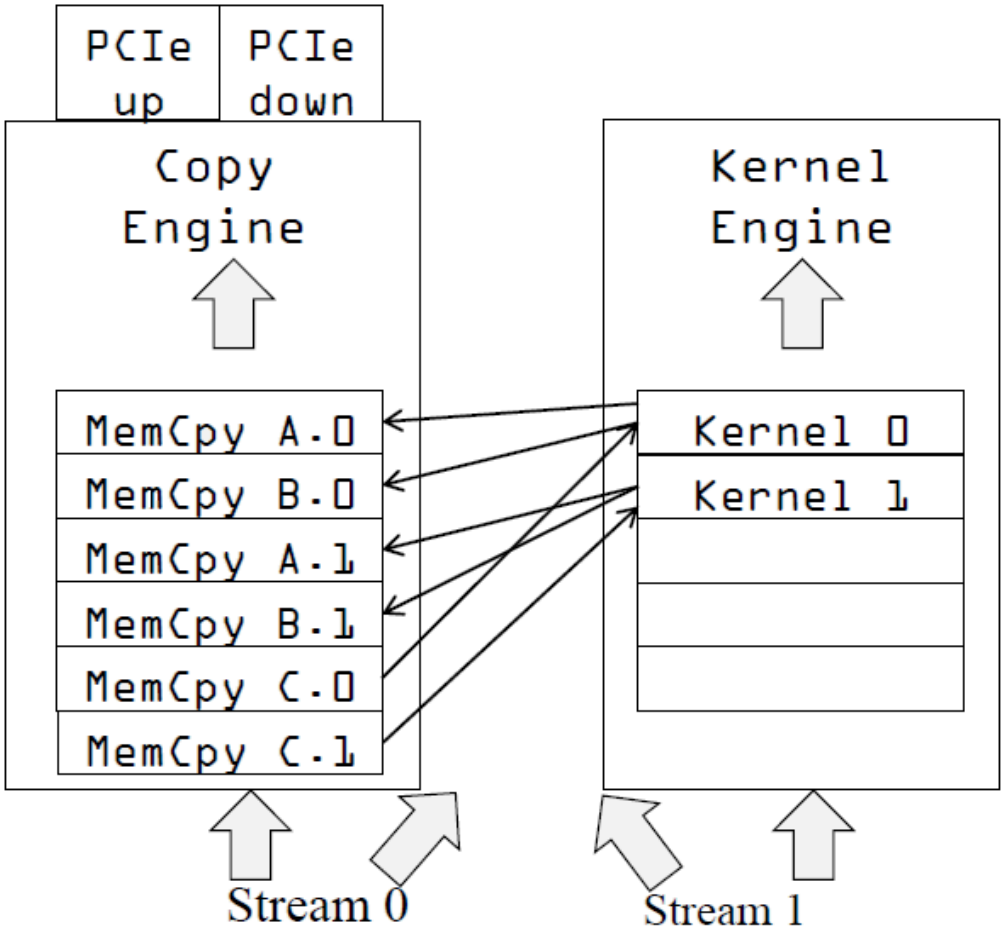
Στην πραγματικότητα η CUDA επιτρέπει την παράλληλη μεταφορά δεδομένων από και προς την host memory. Στις πιο σύγχρονες αρχιτεκτονικές CUDA επιτρέπεται η ταυτόχρονη εκτέλεση και των 3 φάσεων. Έτσι μπορούν να δημιουργηθούν pipelines τριών επιπέδων ώστε να εξαχθεί ακόμα περισσότερη παραλληλία και να ελαττωθεί ο συνολικός χρόνος εκτέλεσης. Για να γίνει αυτό βέβαια, πρέπει το σύνολο των δεδομένων να χωριστεί σε 3 ή περισσότερες ομάδες, κάτι που είναι εύκολο καθώς ισχύει η παραλληλία δεδομένων. Κάθε ακολουθία των 3 φάσεων: μεταφορά host→global memory, εκτέλεση kernel, μεταφορά global → host memory, που γίνεται σε αυτές τις ομάδες (segments) δεδομένων, ονομάζεται stream.



Σχήμα 4.25 Pipelined εκτέλεση kernel και μεταφορές δεδομένων μέσω streams

Μία εντολή ή συνάρτηση στον host code ονομάζεται blocking όταν η εκτέλεση των επόμενων εντολών δεν μπορεί να γίνει πριν αυτή τελειώσει. Αντίθετα μία εντολή ονομάζεται non-blocking όταν μόλις συναντηθεί, ξεκινήσει να εκτελείται, αλλά το πρόγραμμα συνεχίσει με την επόμενη εντολή, χωρίς να περιμένει. Επομένως, μια non-blocking εντολή εκτελείται παράλληλα με τα επόμενα βήματα του host code. Οι kernels είναι non-blocking εντολές. Η CUDA παρέχει blocking , αλλά και non-blocking εντολές για “asynchronous” μεταφορές δεδομένων μεταξύ host και global memory.

Η εκτέλεση των kernel και οι αντιγραφές δεδομένων γίνονται από διαφορετικό hardware της GPU. Έτσι οι αντιγραφές δεδομένων εκτελούνται από το “copy engine” και η εκτέλεση των kernel από το “kernel engine”. Ο προγραμματιστής πρέπει να καθορίζει την αλληλουχία των τριών φάσεων ώστε να επιτυγχάνεται όσο δυνατόν περισσότερη αλληλοεπικάλυψη στα pipelines. Αυτό εξαρτάται και από τις διαφορές στις διάρκειες εκτέλεσης της κάθε φάσης.



Σχήμα 4.26 Αλληλουχία των τριών φάσεων και εκτέλεσή τους από διαφορετικό hardware της GPU

4.5 Εφαρμογές GPGPU

Οι κάρτες γραφικών χρησιμοποιούνται σταδιακά σε όλο και περισσότερους τομείς. Εκτός από τον κλάδο των πολυμέσων και των βιντεοπαιχνιδιών, από όπου ξεκίνησε η χρήση τους, έχουν απήχηση στην βιοπληροφορική, στα ιατρικά μηχανήματα, στην υπολογιστική οικονομική, σε μοντέλα προσομοίωσης καιρικών φαινομένων, κ.τ.λ. Ιδιαίτερο ενδιαφέρον για τους μηχανικούς έχει η εφαρμογή της GPGPU στις επιστήμες των Computation Fluid Dynamics (CFD) και Computational Structural Mechanics. Στην επόμενη ενότητα περιγράφεται το πρόγραμμα που αναπτύχθηκε στα πλαίσια αυτής της εργασίας για το βέλτιστο σχεδιασμό κατασκευών. Για την εκτέλεση των αναλύσεων FEM συνεργάζονται η CPU και η GPU ενός ετερογενούς υπολογιστικού συστήματος.

Κεφάλαιο 5

5 Βελτιστοποίηση σε Ετερογενή Συστήματα

5.1.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται εποπτικά ο κώδικας που αναπτύχθηκε από το γράφοντα για την επίλυση αναλύσεων FEM σε ετερογενές σύστημα CPU & GPU, καθώς και για την εκτέλεση της διαδικασίας βελτιστοποίησης. Τα διάφορα μέρη αυτού του κώδικα συνεργάζονται με άλλα λογισμικά τα οποία επίσης περιγράφονται συνοπτικά. Τέλος, χρησιμοποιούνται τα λογισμικά και ο κώδικας του γράφοντα σε αριθμητική εφαρμογή, που αφορά το βέλτιστο σχεδιασμό μιας κατασκευής.

Τα μέρη του κώδικα που εκτελούνται σε CPU γράφτηκαν σε Java. Η Java είναι η πλέον διαδεδομένη γλώσσα προγραμματισμού και προσφέρει υψηλό επίπεδο αφαίρεσης, πολλές απλουστεύσεις και τη δυνατότητα αντικειμενοστραφούς προγραμματισμού (Object Oriented Programming – OOP). Έτσι, η ανάπτυξη προγραμμάτων σε Java είναι απλούστερη και γρηγορότερη, π.χ. από την πιο περίπλοκη αλλά επίσης αντικειμενοστραφή και ταχύτατη C++. Ο αντικειμενοστραφής προγραμματισμός είναι πολύτιμο βοήθημα στην ανάπτυξη λογισμικών και καθιστά την Java προτιμότερη από την εξίσου απλή και γενικά πιο γρήγορη Fortran. Το βασικό μειονέκτημα της Java είναι ότι είναι συγκριτικά πιο αργή και καταναλώνει περισσότερη μνήμη από γλώσσες όπως η C++ και η Fortran. Αυτό οφείλεται κυρίως στο μεγάλο αριθμό απλουστεύσεων (π.χ. ο προγραμματιστής δεν χρειάζεται να αποδεσμεύει ο ίδιος μνήμη που δε χρειάζεται πλέον - γίνεται αυτόματα) και της αυξημένης ασφάλειας σε προγραμματιστικά σφάλματα (π.χ η Java ελέγχει αυτόματα αν ένα πρόγραμμα επιχειρεί να γράψει σε θέσεις εκτός της επιτρεπόμενης μνήμης ενός array).

Τα μέρη του κώδικα που απαιτούν πολλούς υπολογισμούς και καταναλώνουν το περισσότερο κλάσμα του συνολικού χρόνου, εκτελούνται στην GPU και γράφονται σε CUDA C. Έτσι, οι χρονικές απώλειες που θα επέφερε η χρήση της Java, εξαφανίζονται αφού η CUDA C προσφέρει υψηλό βαθμό ελευθερίας στο χειρισμό της μνήμης και πολλές άλλες βελτιστοποιήσεις. Το κυριότερο όμως πλεονέκτημα είναι η εκτέλεση σε GPU, που είναι κατά πολύ ταχύτερη από την CPU. Βέβαια ο κώδικας που γράφεται για GPU (device code) αφορά διεργασίες που προσφέρονται για SIMD παραλληλία (βλέπε **4.2.3**). Στην ανάλυση FEM που μας ενδιαφέρει, η επίλυση γραμμικών συστημάτων είναι η πλέον χρονοβόρα διαδικασία και πράγματι προσφέρεται για παράλληλη εκτέλεση κατά SIMD. Για την σύνδεση του κυρίως προγράμματος σε Java (host code) με το device code σε CUDA C, χρησιμοποιείται η ανοιχτή βιβλιοθήκη JCuda (http://www.jcuda.org/).

Για τις αναλύσεις με τη μέθοδο FEM χρησιμοποιείται το λογισμικό ***Solverize***, που αναπτύσσεται τα τελευταία χρόνια στο τμήμα Πολιτικών Μηχανικών του ΕΜΠ, από μια ομάδα ατόμων υπό την εποπτεία του κ. Παπαδρακάκη. Το Solverize μορφώνει και λύνει τα συστήματα γραμμικών εξισώσεων που προκύπτουν κατά τη FEM, αποτελεί δηλαδή το μέρος ενός λογισμικού FEM που ονομάζεται ***processor***. Μπορεί να εκτελέσει γραμμικές ή μη γραμμικές, στατικές ή δυναμικές αναλύσεις σε φορείς με έναν ή περισσότερους υποφορείς. Οι βιβλιοθήκες γραμμικής άλγεβρας και ολοκληρώσεων, που χρησιμοποιεί, έχουν γραφτεί ειδικά για την εφαρμογή στα προβλήματα μαθηματικών που προκύπτουν στη FEM και είναι πλήρως βελτιστοποιημένες. Για τις ανάγκες της παρούσας εργασίας, προστέθηκε από το γράφοντα η δυνατότητα επίλυσης γραμμικών συστημάτων σε GPU για γραμμικές στατικές αναλύσεις με ένα υποφορέα (βλ. ενότητα **5.3**Error! Reference source not found.). Επιπλέον προστέθηκαν βασικές δυνατότητες post-processing για τον υπολογισμό τάσεων.

Για τις ανάγκες pre-processing χρησιμοποιείται το ανοιχτό (open source) λογισμικό ***Gmsh*** (http://geuz.org/gmsh). Το Gmsh επιτρέπει στο χρήστη να περιγράφει τη γεωμετρία του φορέα μέσα από ένα γραφικό περιβάλλον και στη συνέχεια δημιουργεί το δίκτυο πεπερασμένων στοιχείων (mesh generation). Εκτός από το γραφικό περιβάλλον, η χρήση του μπορεί να γίνει μέσω της ψευδογλώσσας που διαθέτει. Έτσι ο χρήστης περιγράφει τη γεωμετρία και τις προτιμήσεις του για τη γένεση του δικτύου μέσα σε ***scripts,*** τα οποία το Gmsh διαβάζει και εκτελεί τις αντίστοιχες εντολές (με παρόμοιο τρόπο λειτουργεί το Matlab). Η λειτουργία αυτή είναι πολύτιμη για τη διαδικασία της βελτιστοποίησης, επειδή επιτρέπει την αυτόματη ανανέωση των τιμών των μεταβλητών σχεδιασμού και τη γένεση νέων δικτύων για κάθε δοκιμαζόμενο σχεδιασμό. Ταυτόχρονα ο προσδιορισμός της αρχικής γεωμετρίας γίνεται πριν την έναρξη της βελτιστοποίησης, μέσω του γραφικού περιβάλλοντος που είναι ευκολότερο στη χρήση. Το Gmsh είναι multiplatform, δηλαδή διατίθεται για τα λειτουργικά συστήματα Windows, Linux και Mac OS X. Προφανώς, ως open source λογισμικό, διατίθεται και το source code του, που είναι γραμμένο σε C++.

Το υπολογιστικό σύστημα στο οποίο εκτελούνται οι διάφορες αναλύσεις και η διαδικασία της βελτιστοποίησης διαθέτει:

- CPU: Intel Pentium G3240. 2 πυρήνες στα 3.10 GHz.
- Κεντρική μνήμη RAM: 4GB
- Κάρτα γραφικών: Nvidia GeForce GTX 580. 512 πυρήνες στα 1544 MHz. 1.5GB RAM με 192.4 GB/s bandwidth.

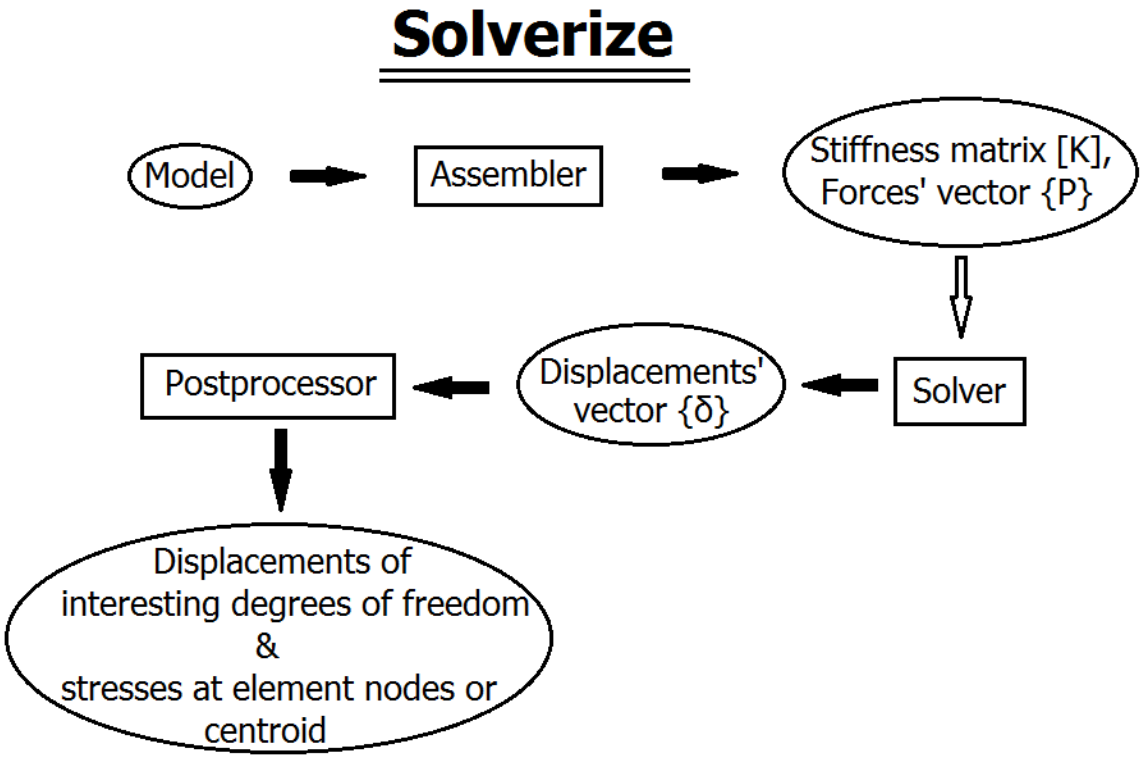
Η CPU που χρησιμοποιείται ανήκει στη φθηνή κατηγορία CPU, ενώ η κάρτα γραφικών ανήκει στη μεσαία προς ακριβή αντίστοιχη κατηγορία. Επομένως σε όσες συγκρίσεις ακολουθούν, αναμένεται καλύτερη απόδοση της GPU. Πάντως, οι αλγόριθμοι που γράφονται σε CPU είναι σειριακοί και άρα χρησιμοποιούν μόνο ένα πυρήνα. Η απόδοση της συγκεκριμένης CPU σε σειριακά προγράμματα (single threaded performance) είναι ιδιαίτερα υψηλή, παρά την χαμηλή τιμή της, οπότε η χρήση ακριβότερης CPU για τους σειριακούς αυτούς αλγόριθμους δεν θα επέφερε σημαντική βελτίωση. Βέβαια, μια πιο αντικειμενική σύγκριση μεταξύ CPU και GPU, θα

αφορούσε την απόδοση της CPU σε παράλληλους αλγορίθμους με 4 ή περισσότερα threads . Στην περίπτωση αυτή, η φθηνή CPU που χρησιμοποιήθηκε εδώ θα ήταν σαφώς πιο αργή από άλλες CPU στην ίδια κατηγορία τιμών με τη συγκεκριμένη κάρτα γραφικών. Το κόστος του hardware είναι ένας από τους σημαντικότερους παράγοντες και πρέπει να λαμβάνεται υπόψη στις συγκρίσεις. Οι συγκρίσεις που γίνονται στα επόμενα έχουν περισσότερο ποιοτική αξία. Αν τα προγράμματα εκτελεστούν σε ένα συνηθισμένο υπολογιστή, αναμένεται καλύτερη απόδοση των αλγορίθμων που εκμεταλλεύονται την GPU, αλλά οι διαφορές δεν θα είναι τόσο έντονες.

5.2 Ροή προγράμματος βελτιστοποίησης

Παρακάτω φαίνονται διαγράμματα που δείχνουν εποπτικά τη ροή της λειτουργίας των διαφόρων μερών του κώδικα που συμμετέχουν στη διαδικασία της βελτιστοποίησης, όπως αυτή εφαρμόζεται στην παρούσα εργασία. Γενικά, με έλλειψη σηματοδοτούνται δεδομένα και με ορθογώνια παραλληλόγραμμα σηματοδοτούνται διεργασίες. Βέβαια, στον αντικειμενοστραφή προγραμματισμό ένα ***object*** περιέχει δεδομένα και ταυτόχρονα εκτελεί διεργασίες. Επομένως όλα τα επόμενα σχήματα παριστάνουν objects, που ανάλογα με τη βασική λειτουργία τους σηματοδοτούνται με έλλειψη ή ορθογώνιο παραλληλόγραμμο.

Η λειτουργία του λογισμικού Solverize φαίνεται στο παρακάτω διάγραμμα:

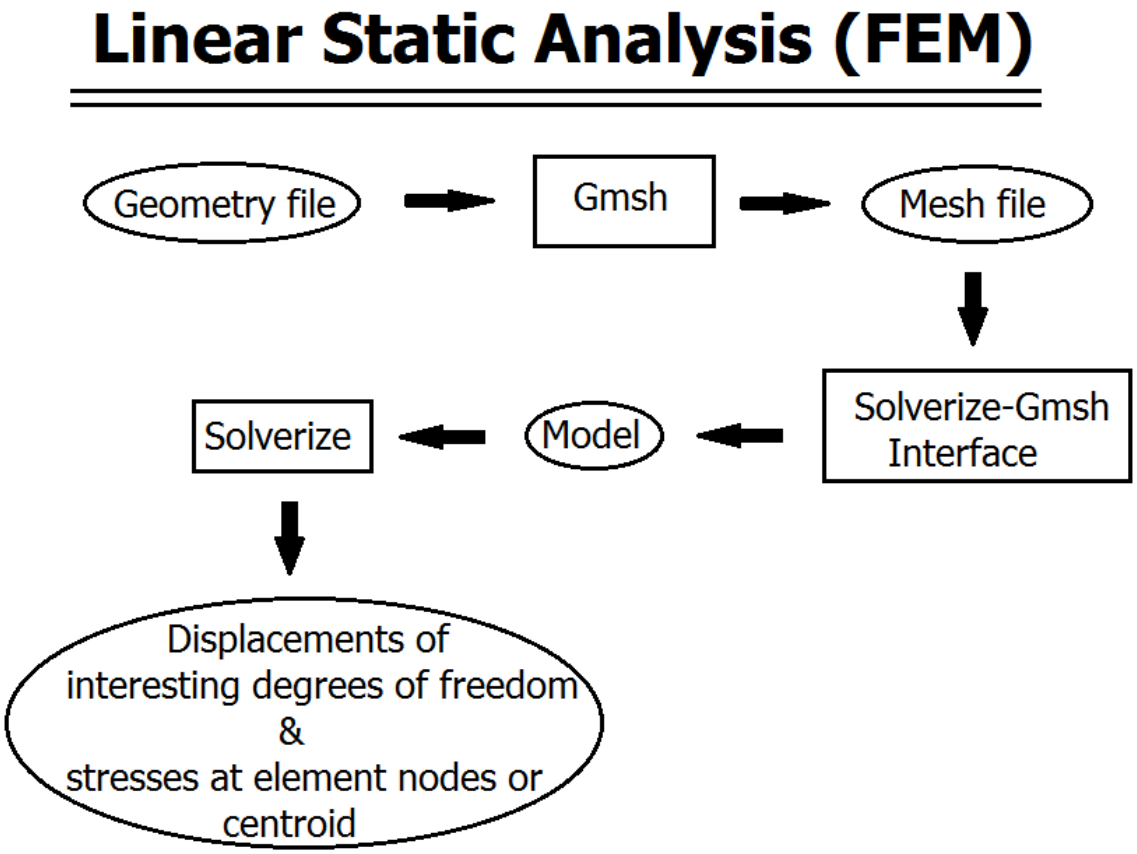


Σχήμα 5.1 Τυπική λειτουργία του Solverize για γραμμικές στατικές αναλύσεις FEM

Όπου:

- *Model* είναι το μοντέλο προσομοίωσης του φορέα και περιλαμβάνει τα πεπερασμένα στοιχεία, τους κόμβους, τα φορτία και τις δεσμεύσεις βαθμών ελευθερίας.
- *Assembler* είναι object που παράγει το μητρώο στιβαρότητας (και μάζας, απόσβεσης σε δυναμικές αναλύσεις) κάθε πεπερασμένου στοιχείου, με ολοκλήρωση Gauss. Στη συνέχεια συνθέτει τα επιμέρους μητρώα όλων των στοιχείων και δημιουργεί το καθολικό μητρώο στιβαρότητας $[K]$ του φορέα. Επίσης δημιουργεί το διάνυσμα επικόμβιων δράσεων $\{P\}$. Έτσι προσδιορίζεται το γραμμικό σύστημα $[K] * \{\Delta\} = \{P\}$ που θα επιλυθεί στη συνέχεια.
- *Solver* είναι object που επιλύει το παραπάνω γραμμικό σύστημα και υπολογίζει το διάνυσμα επικόμβιων μετατοπίσεων $\{\Delta\}$. Το Solverize διαθέτει διάφορους Solvers: Skyline, PCG, FETI (για υποφορείς), … . Για τις ανάγκες της παρούσας εργασίας, προστέθηκε η δυνατότητα επίλυσης γραμμικών συστημάτων σε GPU με τη μέθοδο Conjugate Gradient.
- *Postprocessor* είναι object που προστέθηκε για τις ανάγκες της παρούσας εργασίας και διευκολύνει τον υπολογισμό τάσεων στα πεπερασμένα στοιχεία, αφού έχει βρεθεί το διάνυσμα επικόμβιων μετατοπίσεων $\{\Delta\}$. Προς το παρόν, οι τάσεις μπορούν να υπολογιστούν στους κόμβους ή το κέντρο κάθε στοιχείου.

Με την προσθήκη του Gmsh ως preprocessor, μία πλήρης εκτέλεση γραμμικής, στατικής ανάλυσης με έναν υποφορέα γίνεται ως εξής:

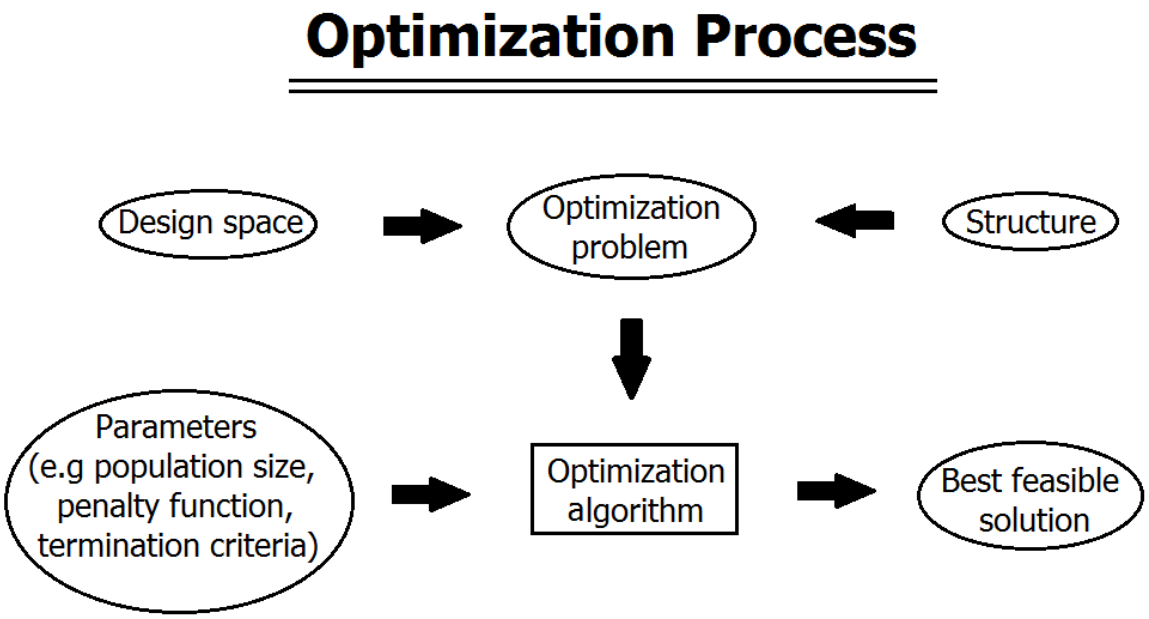


Σχήμα 5.2 Πλήρης εκτέλεση μίας ανάλυσης FEM με Solverize & Gmsh

Όπου:

- Τα *Geometry file* και *Mesh file* είναι αρχεία κειμένου που χρησιμοποιούνται για είσοδο και έξοδο δεδομένων στο Gmsh και περιέχουν την περιγραφή της γεωμετρίας του φορέα και τη διακριτοποίησή του σε πεπερασμένα στοιχεία και κόμβους αντίστοιχα.
- Το *Solverize-Gmsh interface* αποτελεί τη διεπιφάνεια επικοινωνίας των δύο λογισμικών. Διαβάζει Mesh file και δημιουργεί το Model που μπορεί πλέον να επεξεργαστεί το Solverize.
- Επειδή η επικοινωνία των δύο προγραμμάτων γίνεται μέσω αρχείων εισόδου/εξόδου, εισάγονται σημαντικές καθυστερήσεις. Πραγματοποιούνται συνολικά 2 αναγνώσεις και 2 εγγραφές των αρχείων που αποθηκεύονται στον σκληρό δίσκο. Όπως αναλύθηκε στο προηγούμενο κεφάλαιο, η μεταφορά δεδομένων από το σκληρό δίσκο στη RAM και αντιστρόφως, είναι πολύ αργές συγκριτικά με τα υπόλοιπα μέρη του προγράμματος. Έτσι σε μικρούς φορείς, η δημιουργία του Model απαιτεί δυσανάλογα μεγάλο κλάσμα του συνολικού χρόνου εκτέλεσης. Αντίθετα σε φορείς με πολλούς βαθμούς ελευθερίας, η καθυστέρηση αυτή είναι αμελητέα, αφού ο απαιτούμενος χρόνος της επίλυσης του γραμμικού συστήματος αυξάνεται δραματικά.

Η διαδικασία της βελτιστοποίησης έχει την ακόλουθη ροή:

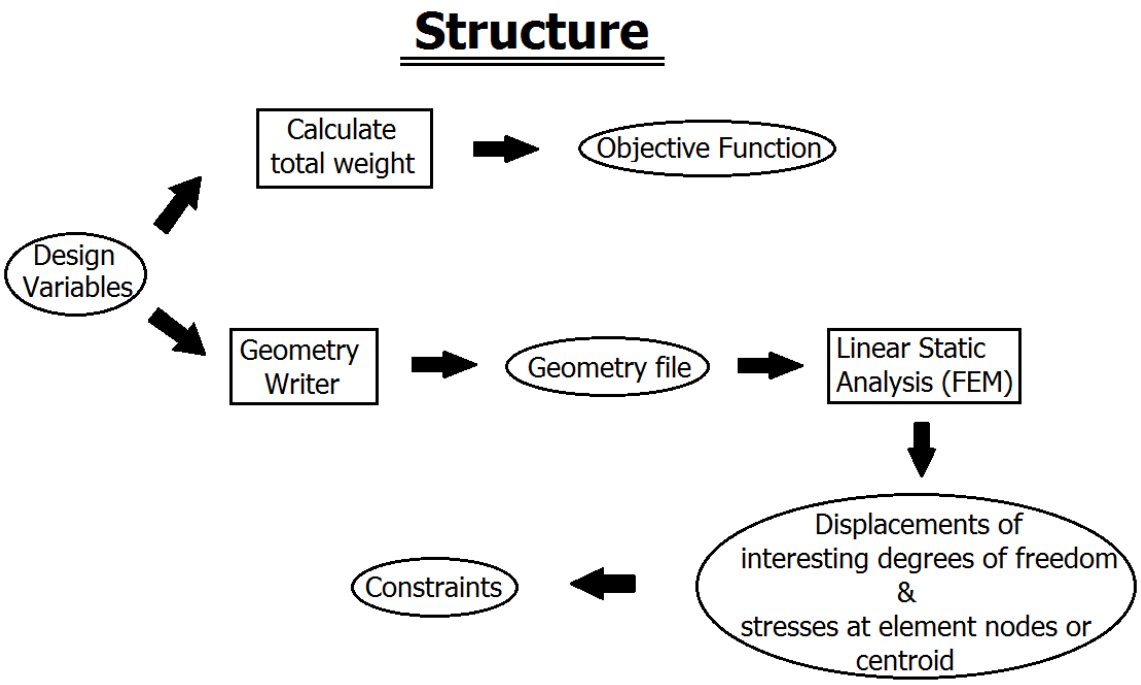


Σχήμα 5.3 Ροή της διαδικασίας βελτιστοποίησης

Όπου:

- Το *Optimization problem* περιέχει όλα τα δεδομένα που προσδιορίζουν πλήρως το πρόβλημα βελτιστοποίησης προς επίλυση. Τέτοια δεδομένα είναι οι μεταβλητές σχεδιασμού και ο χώρος δυνατών τιμών τους (Design space). Η αντικειμενική συνάρτηση και οι συναρτήσεις περιορισμών περιέχονται στο object *Structure*, που αναλύεται στο επόμενο σχήμα.
- *Optimization algorithm* είναι το object που εκτελεί την βελτιστοποίηση σύμφωνα με κάποιον αλγόριθμο όπως η Βελτιστοποίηση Σμήνους Σωματιδίων, η Διαφορική Εξέλιξη και οι Στρατηγικές Εξέλιξης που περιγράφηκαν στο κεφάλαιο **2**. Ως είσοδο δεδομένων χρειάζεται τις παραμέτρους της εκάστοτε μεθόδου, π.χ. το μέγεθος του πληθυσμού, τα κριτήρια τερματισμού και τη συνάρτηση στην περίπτωση της αντιμετώπισης των περιορισμών με την τεχνική αυτή.
- Μετά την εκτέλεση του Optimization algorithm επιστρέφεται η καλύτερη από τις διάφορες εφικτές λύσεις που δοκιμάστηκαν.

Παρακάτω φαίνεται η δομή και λειτουργία του object Structure.



Σχήμα 5.4 Η διαδικασία δοκιμής κάθε πιθανού σχεδιασμού.

Το object ***Structure*** είναι υπεύθυνο για την δοκιμή κάθε πιθανής λύσης που προκύπτει κατά τη διάρκεια της βελτιστοποίησης. Στο βέλτιστο σχεδιασμό κατασκευών, ο υπολογισμός των συναρτήσεων περιορισμών γίνεται μετά από ανάλυση FEM, οπότε αυτό το μέρος του προγράμματος είναι το πλέον περίπλοκο και χρονοβόρο. Ο υπολογισμός της αντικειμενικής συνάρτησης είναι συνήθως απλός, αφού αυτή αφορά το συνολικό βάρος του φορέα, το οποίο δεν απαιτεί κάποια ενδιάμεση επεξεργασία για να υπολογιστεί. Οι δύο αυτές λειτουργίες απομονώνονται από την υπόλοιπη διαδικασία βελτιστοποίησης, η οποία χρειάζεται απλά τιμές για τις συναρτήσεις περιορισμών και αντικειμένων. Το όνομα *Structure* εμπνέεται από το *φορέα* που δοκιμάζεται, αλλά το object αυτό μπορεί να χρησιμοποιηθεί για την βελτιστοποίηση συστημάτων που δεν είναι απαραίτητα κατασκευές, οπότε το *Structure* ερμηνεύεται ως *δομή*.

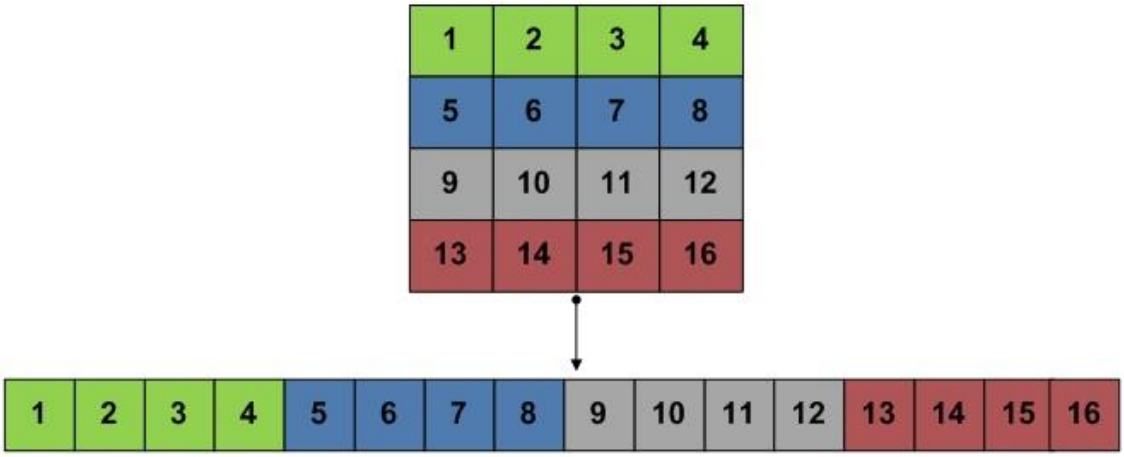
Το Structure που χρησιμοποιείται σε αυτή την εργασία:

- Χρησιμοποιεί ως περιορισμούς τις μετατοπίσεις κόμβων και τις τάσεις πεπερασμένων στοιχείων. Αυτά τα μεγέθη προκύπτουν μετά από γραμμική στατική ανάλυση FEM με ένα υποφορέα. Για την ανάλυση FEM χρησιμοποιείται ο συνδυασμός Solverize & Gmsh που αναλύθηκε προηγουμένως.
- Για την εκτέλεση αυτής της διαδικασίας χρειάζεται ο προσδιορισμός της γεωμετρίας του φορέα που αντιστοιχεί στις μεταβλητές σχεδιασμού κάθε πιθανής λύσης. Αυτό αναλαμβάνεται από το object *Geometry writer*, το οποίο γράφει τα αρχεία Geometry, που στη συνέχεια διαβάζονται από το Gmsh.
- Ο υπολογισμός της αντικειμενικής συνάρτησης γίνεται άμεσα από τις τιμές των μεταβλητών σχεδιασμού. Στην αριθμητική εφαρμογή που ακολουθεί, ως αντικειμενική συνάρτηση χρησιμοποιείται το συνολικό βάρος του φορέα.

5.3 Επίλυση γραμμικών συστημάτων με GPU

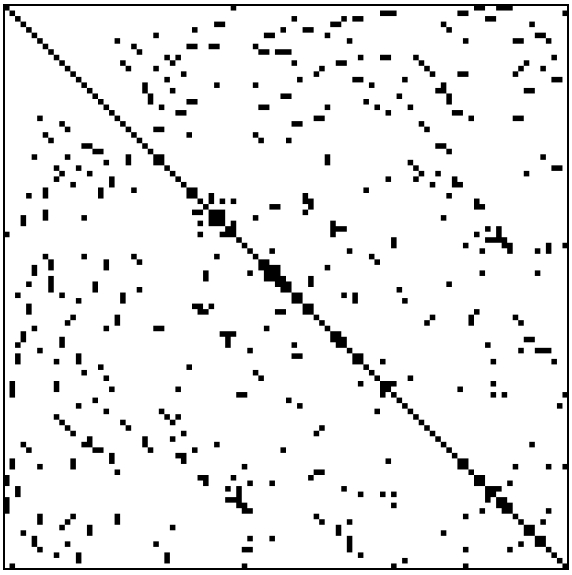
5.3.1 Απεικόνιση πινάκων

Ο συνηθισμένος τρόπος θεώρησης ενός πίνακα είναι ως ένα ορθογωνικό πλέγμα από γραμμές (rows) και στήλες (columns). Βέβαια, η μνήμη στον υπολογιστή δεν είναι 2D, αλλά μια σειρά από διαδοχικές θέσεις. Έτσι γλώσσες όπως η C και οι παράγωγές της (C++, Java, C#, CUDA C, OpenCL, …), οι οποίες σήμερα έχουν επικρατήσει στο χώρο του προγραμματισμού, παριστάνουν ένα πίνακα ως πολλές σειρές (arrays) στοιχείων, καθεμία από τις οποίες αποτελεί μια γραμμή του πίνακα, τοποθετημένες η μία μετά την άλλη. Έτσι στοιχεία της ίδιας γραμμής βρίσκονται σε διαδοχικές θέσεις μνήμης, αλλά στοιχεία της ίδιας στήλης όχι. Αυτή η μορφή απεικόνισης χαρακτηρίζεται ως row major. Ένας πίνακας, στον οποίο αποθηκεύονται όλα τα στοιχεία του, ονομάζεται ***dense matrix***. Γενικά, χρειάζεται η αποθήκευση όλων των στοιχείων ενός πίνακα, για να μην χάνονται πληροφορίες, οπότε ο dense matrix είναι η πιο παραδοσιακή μορφή πίνακα.



Σχήμα 5.5 Row major απεικόνιση ενός Dense Matrix.

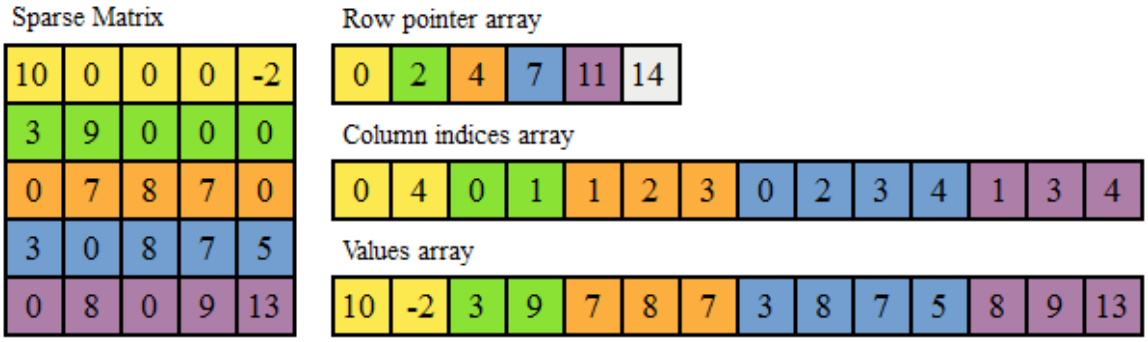
Στους πίνακες που χρησιμοποιούνται στη μέθοδο πεπερασμένων στοιχείων, ελάχιστα στοιχεία είναι μη μηδενικά. Οι πίνακες αυτοί είναι τετραγωνικοί και έχουν τόσες γραμμές και στήλες όσοι οι βαθμοί ελευθερίας του μοντέλου προσομοίωσης. Στα συνηθέστερα προβλήματα, όμως, υπάρχουν χιλιάδες ή εκατομμύρια βαθμοί ελευθερίας κάτι που οδηγεί σε πολύ μεγάλους πίνακες. Ένα άλλο χαρακτηριστικό των πινάκων στη FEM, είναι ότι κάθε βαθμός ελευθερίας αλληλεπιδρά με πολύ μικρό αριθμό άλλων βαθμών. Έτσι, τα περισσότερα στοιχεία του πίνακα είναι 0, εκτός από την κύρια διαγώνιο και κάποια μεμονωμένα στοιχεία, που συνήθως βρίσκονται γύρω της. Η αποθήκευση ενός τέτοιου πίνακα με τη μορφή dense matrix δαπανά υπερβολικά πολλή μνήμη, οπότε δεν είναι οικονομική ή δεν είναι καν εφικτή για πολύ μεγάλα μοντέλα. Για να αντιμετωπιστούν αυτές οι δυσκολίες, δεν αποθηκεύονται τα μηδενικά στοιχεία, που απλά καταναλώνουν μνήμη χωρίς να προσφέρουν κάποια χρήσιμη πληροφορία. Έτσι αποθηκεύονται μόνο τα μη μηδενικά στοιχεία, που στα προβλήματα FEM είναι ελάχιστα, καθώς και κάποιες οδηγίες για τον προσδιορισμό της θέσης αυτών. Αυτή η μορφή αποθήκευσης ονομάζεται ***sparse matrix***.



Σχήμα 5.6 Sparse πίνακας που συναντάται στην FEM

Υπάρχουν διάφορες μορφές αποθήκευσης ενός sparse matrix. Στην παρούσα εργασία χρησιμοποιείται μία από τις πλέον διαδεδομένες, η οποία ονομάζεται ***CSR*** (Compressed Row Storage). Σύμφωνα με την CSR ένας πίνακας $m \times n$ αποθηκεύεται με 3 arrays:

- Values array: περιέχει τα nnz μη μηδενικά στοιχεία σε row major απεικόνιση, δηλαδή πρώτα τα μη μηδενικά στοιχεία της πρώτης γραμμής, μετά της δεύτερης, κτλ. Έχει μήκος nnz.
- Column indexes array: περιέχει τους αναγνωριστικούς αριθμούς (index) των στηλών στις οποίες ανήκουν τα μη μηδενικά στοιχεία (που βρίσκονται στην values array). Έχει επίσης μήκος nnz.
- Row pointers array: αποθηκεύει και αυτό indexes. Για το πρώτο μη μηδενικό στοιχείο κάθε γραμμής του πίνακα, αποθηκεύεται η θέση που έχει στην values array. Το array αυτό περιέχει m στοιχεία (1 για κάθε στήλη) και ένα στο τέλος που ισούται με: $nnz - \tau o\ \pi\rho\acute{\omega}\tau o\ \sigma\tau o\iota\chi\epsilon\acute{\iota}o\ \tau\eta\varsigma\ row\ pointers\ array$. Έτσι έχει συνολικό μήκος $m + 1$. Γενικά το πρώτο στοιχείο είναι 1 ή 0. Το 0 είναι για τις γλώσσες που βασίζονται στη C, όπου η αρίθμηση ξεκινάει πάντα από το 0 (0,1,2,3,…).



Σχήμα 5.7 CSR πίνακας

Σε έναν dense matrix οι θέσεις μνήμης προσπελαύνονται διαδοχικά, κατά τις συνηθέστερες μεθόδους της γραμμικής άλγεβρας (πρόσθεση πινάκων, πολλαπλασιασμός πίνακα με αριθμό, πολλαπλασιασμός πίνακα με πίνακα, πολλαπλασιασμός πίνακα με διάνυσμα). Στην GPU αυτό ισοδυναμεί με coalesced memory access (βλέπε **4.4.2**), οπότε παρουσιάζεται πολύ καλή απόδοση που μπορεί να φτάνει 50-100x την απόδοση της CPU, αν εξεταστεί μόνο αυτό το μέρος του κώδικα. Ωστόσο η μεταφορά ενός μεγάλου dense matrix από την κεντρική μνήμη του συστήματος στην μνήμη της κάρτας γραφικών είναι πολύ αργή και συχνά αδύνατη, αν δεν επαρκεί κάποια από τις δύο μνήμες.

Σε ένα sparse matrix, οι προσπελάσεις μνήμης είναι τυχαίες (uncoalesced), οπότε η διαφορά στην απόδοση CPU-GPU δεν είναι τόσο μεγάλη. Ωστόσο, η μεταφορά ενός sparse matrix από τη host στην device memory, είναι πολύ ταχύτερη λόγω του μικρού μεγέθους του. Επιπλέον πολλές βασικές μέθοδοι της γραμμικής άλγεβρας είναι ταχύτερες σε sparse matrices, αν εφαρμοστούν με τρόπο που να αποφεύγει περιττές πράξεις με τα μηδενικά στοιχεία. Για αυτούς τους λόγους, συνήθως προτιμώνται οι sparse matrices σε CPU και GPU.

5.3.2 Η μέθοδος Conjugate Gradient

Για την επίλυση συστημάτων εξισώσεων χρησιμοποιούνται δύο κατηγορίες μεθόδων:

- Άμεσες (direct) μέθοδοι, που επιχειρούν να υπολογίσουν τη λύση με ένα πεπερασμένο πλήθος πράξεων και απόλυτη ακρίβεια. Τέτοιες μέθοδοι είναι η απαλοιφή Gauss, η παραγοντοποίηση LU, κτλ. Οι άμεσες μέθοδοι δεν μπορούν να εφαρμοστούν σε μη γραμμικά συστήματα εξισώσεων.
- Επαναληπτικές (iterative) μέθοδοι, που επιχειρούν να προσεγγίσουν τη λύση του συστήματος με διαδοχικές δοκιμές πιθανών λύσεων-διανυσμάτων, οι οποίες συνεχίζονται μέχρι να επιτευχθεί ικανοποιητική σύγκλιση ή να διακοπεί η μέθοδος, οπότε και αποτυγχάνει. Οι επαναληπτικές μέθοδοι μπορούν να επιλύσουν και μη γραμμικά συστήματα.

Οι επαναληπτικές μέθοδοι χρησιμοποιούνται συχνά σε γραμμικά συστήματα. Στον κλάδο των πεπερασμένων στοιχείων παρουσιάζουν δύο σημαντικά πλεονεκτήματα έναντι των συνεχών:

- Σε πολύ μεγάλους πίνακες μπορεί να εντοπίζουν την λύση με ικανοποιητική ακρίβεια πολύ γρηγορότερα από τις άμεσες.
- Οι περισσότερες άμεσες μέθοδοι τροποποιούν σε κάθε βήμα τον πίνακα του συστήματος (π.χ. τριγωνοποίηση). Σε πολύ μεγάλα προβλήματα χρησιμοποιούνται αναγκαστικά sparse πίνακες, οι οποίοι προοδευτικά «γεμίζουν» λόγω της τροποποίησης αυτής. Έτσι απαιτείται σημαντικά περισσότερη μνήμη που συνήθως δεν διατίθεται στο σύστημα. Μπορεί επίσης να απαιτούνται και αντιγραφές ολόκληρου του πίνακα ή στοιχείων του από βήμα σε βήμα. Αυτό βέβαια εξαρτάται και από τη μορφή του πίνακα. Οι dense matrixes δεν έχουν τέτοιο θέμα.

Κατά τη διάρκεια της παρούσας εργασίας γράφτηκε κώδικας που εφαρμόζει τη μέθοδο Conjugate Gradient (CG) σε CSR πίνακες. Ο κώδικας αυτός γράφτηκε σε CUDA C για εκτέλεση ολόκληρης της επίλυσης του γραμμικού συστήματος στην GPU. Έστω το γραμμικό σύστημα:

$$\boldsymbol{A} * \boldsymbol{x} = \boldsymbol{b} \qquad \textbf{(5.1)}$$

Όπου:

- $\boldsymbol{A}$ είναι ένας συμμετρικός ($\boldsymbol{A}^T = \boldsymbol{A}$) και θετικά ορισμένος ($\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0$, $για\ κάθε\ μη\ μηδενικό\ \boldsymbol{x}$) πίνακας $n \times n$
- $\boldsymbol{b}$ είναι ένα γνωστό διάνυσμα $n \times 1$
- $\boldsymbol{x}$ είναι η λύση - διάνυσμα $n \times 1$, που αναζητείται

Η μέθοδος CG (Hestenes M., Stiefel E.) επιχειρεί να προσεγγίσει την ακριβή λύση $\boldsymbol{x}$ ξεκινώντας από μία αρχική δοκιμή $\boldsymbol{x_0}$ (συνήθως $\boldsymbol{x_0} = \boldsymbol{0}$), την οποία βελτιώνει σε κάθε επανάληψη χρησιμοποιώντας ένα διάνυσμα που καθορίζει τη διεύθυνση αναζήτησης και ένα πραγματικό αριθμό που καθορίζει το βήμα αναζήτησης σε αυτή τη διεύθυνση. Η απόκλιση/ υπόλοιπο (residual) του γινομένου $\boldsymbol{Ax}$ από το γνωστό στόχο $\boldsymbol{b}$, παριστάνεται από ένα τρίτο διάνυσμα $\boldsymbol{r}$. Η μέθοδος τερματίζει όταν το υπόλοιπο αυτό είναι αρκούντως μικρό ή έχει εξαντληθεί το επιτρεπόμενο πλήθος επαναλήψεων. Στην παρούσα εργασία, το υπόλοιπο μπορεί να θεωρηθεί αρκούντως μικρό αν:

$$norm2(\boldsymbol{r}) = \sqrt{\sum r_i^2} \leq \varepsilon, \qquad \varepsilon = 10^{-6} \tag{5.2}$$

Η κατάλληλη επιλογή της αρχικής δοκιμαστικής λύσης $\boldsymbol{x_0}$ μπορεί να μειώσει σημαντικά το πλήθος των επαναλήψεων εξοικονομώντας πολύ χρόνο, αλλά απαιτεί γνώση των χαρακτηριστικών του συστήματος. Γι' αυτό θα χρησιμοποιηθεί η τυπική $\boldsymbol{x_0} = \boldsymbol{0}$ στην παρούσα εργασία. Ο πλήρης αλγόριθμος της Conjugate Gradient φαίνεται στο επόμενο σχήμα:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$

$\mathbf{p}_0 := \mathbf{r}_0$

$k := 0$

while (k ≤ kmax) repeat

$\alpha_k := \dfrac{\mathbf{r}_k^{\mathsf{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathsf{T}}\mathbf{A}\mathbf{p}_k}$

$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$

$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$

if $\dfrac{\text{norm2}(\mathbf{r}_{k+1})}{\text{norm2}(\mathbf{b})} \leq \varepsilon$, then exit loop

$\beta_k := \dfrac{\mathbf{r}_{k+1}^{\mathsf{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathsf{T}}\mathbf{r}_k}$

$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$

$k := k + 1$

end repeat

The result is $\mathbf{x}_{k+1}$

Σχήμα 5.8 Η μέθοδος Conjugate Gradient

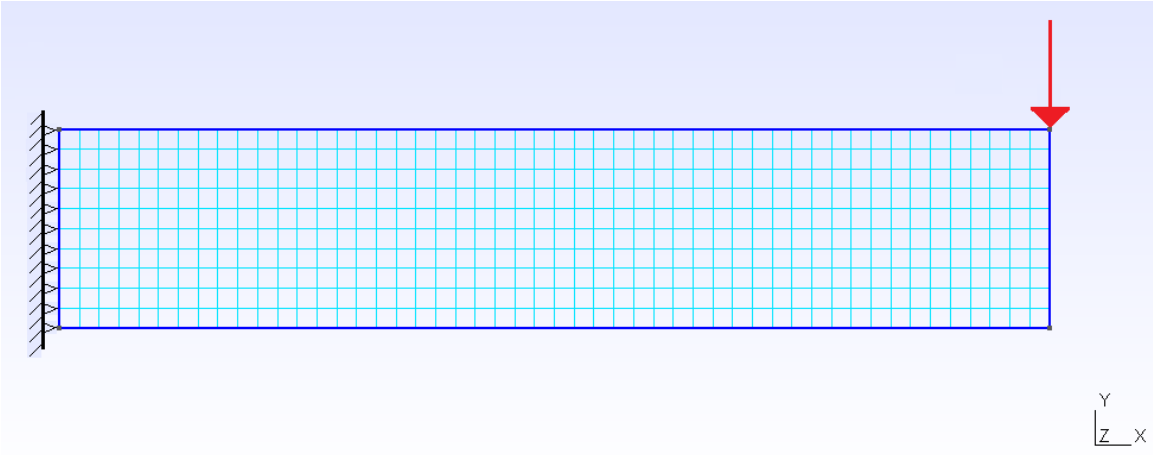
Σε κάθε επανάληψη εκτελείται ένας πολλαπλασιασμός πίνακα με διάνυσμα, εσωτερικά γινόμενα και προσθέσεις διανυσμάτων και πολλαπλασιασμός διανύσματος με πραγματικό αριθμό (scale). Οι πράξεις των διανυσμάτων, κοστίζουν χρονικά, αλλά η πλέον χρονοβόρα είναι ο πολλαπλασιασμός πίνακα με διάνυσμα. Στην περίπτωση των sparse πινάκων, ο πολλαπλασιασμός αυτός μπορεί να επιταχυνθεί σημαντικά, αγνοώντας τα πάρα πολλά μηδενικά στοιχεία. Οι πίνακες που προκύπτουν στην FEM είναι συμμετρικοί, θετικά ορισμένοι και αναγκαστικά sparse, οπότε μπορεί να εφαρμοστεί η CG και μάλιστα με αυξημένη απόδοση.

Γενικά η Conjugate Gradient απαιτεί πολλές επαναλήψεις για να συγκλίνει. Στην πράξη, συχνά χρησιμοποιείται μια παραλλαγή της, η Preconditioned Conjugate Gradient (PCG), που τροποποιεί τον αρχικό πίνακα A πριν ξεκινήσουν οι επαναλήψεις. Με την κατάλληλη τροποποίηση (preconditioning), δεν αυξάνονται τα μη μηδενικά στοιχεία (διατήρηση του sparsity pattern) και τελικά η μέθοδος συγκλίνει με πολύ λιγότερες επαναλήψεις. Ωστόσο, το θέμα του preconditioning είναι σύνθετο και ξεφεύγει από το πλαίσιο αυτής της εργασίας, οπότε στα προγράμματα και τις εφαρμογές χρησιμοποιείται η απλή CG.

Μία ενδιαφέρουσα παρατήρηση είναι ότι η Conjugate Gradient εκτελεί βελτιστοποίηση της δοκιμαστικής λύσης, δοκιμάζοντας διάφορα πιθανά διανύσματα και ελέγχοντας αν έχει επιτύχει ικανοποιητική ακρίβεια. Πράγματι η CG αποτελεί μαθηματικό αλγόριθμο βελτιστοποίησης και χρησιμοποιείται σε προβλήματα χωρίς περιορισμούς.

5.3.3 Σύγκριση απόδοσης της CG σε CPU & GPU

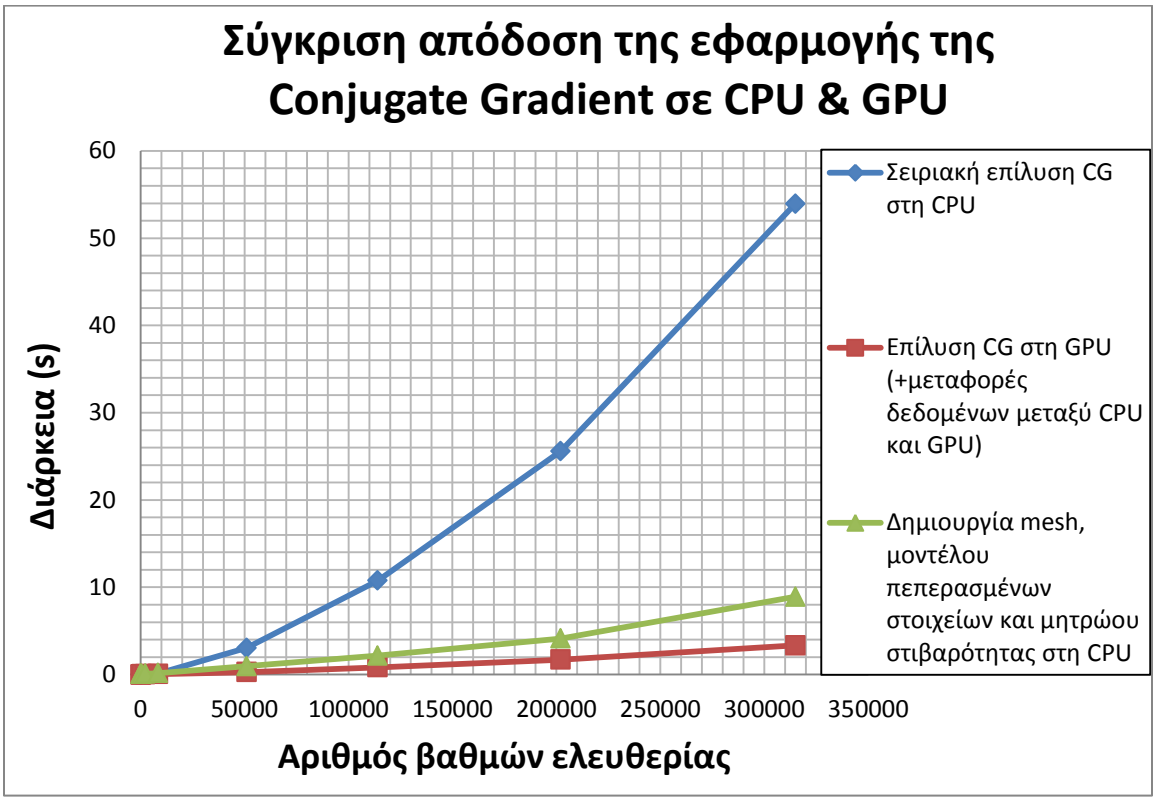
Για να συγκριθεί η απόδοση της Conjugate Gradient σε CPU και GPU γράφτηκε ο παραπάνω αλγόριθμος και σε Java, για επίλυση στην CPU. Αυτή η εκδοχή του αλγορίθμου είναι τελείως σειριακή. Στη συνέχεια εκτελέστηκε στατική γραμμική ανάλυση FEM σε ένα απλό δισδιάστατο φορέα που φαίνεται στο επόμενο σχήμα. Η διαδικασία της ανάλυσης φαίνεται στο **Σχήμα 5.2**. Σαν Solver χρησιμοποιήθηκαν οι εφαρμογές της Conjugate Gradient σε CPU και σε GPU. Ο πίνακας του γραμμικού συστήματος είχε τη μορφή CSR.



Σχήμα 5.9 Απλός 2D πρόβολος

Στο παρακάτω διάγραμμα φαίνεται:

- η συνολική διάρκεια των διαδικασιών δημιουργίας των input/output files, γένεσης του δικτύου πεπερασμένων στοιχείων, μόρφωσης του μοντέλου που χρησιμοποιεί το Solverize και υπολογισμό του μητρώου στιβαρότητας για το γραμμικό σύστημα.
- Η διάρκεια της επίλυσης του γραμμικού συστήματος με τη μέθοδο CG στην CPU, σειριακά.
- Η συνολική διάρκεια της ίδιας επίλυσης στην GPU και των μεταφορών δεδομένων μεταξύ της κύριας μνήμης του συστήματος και της μνήμης της κάρτας γραφικών.



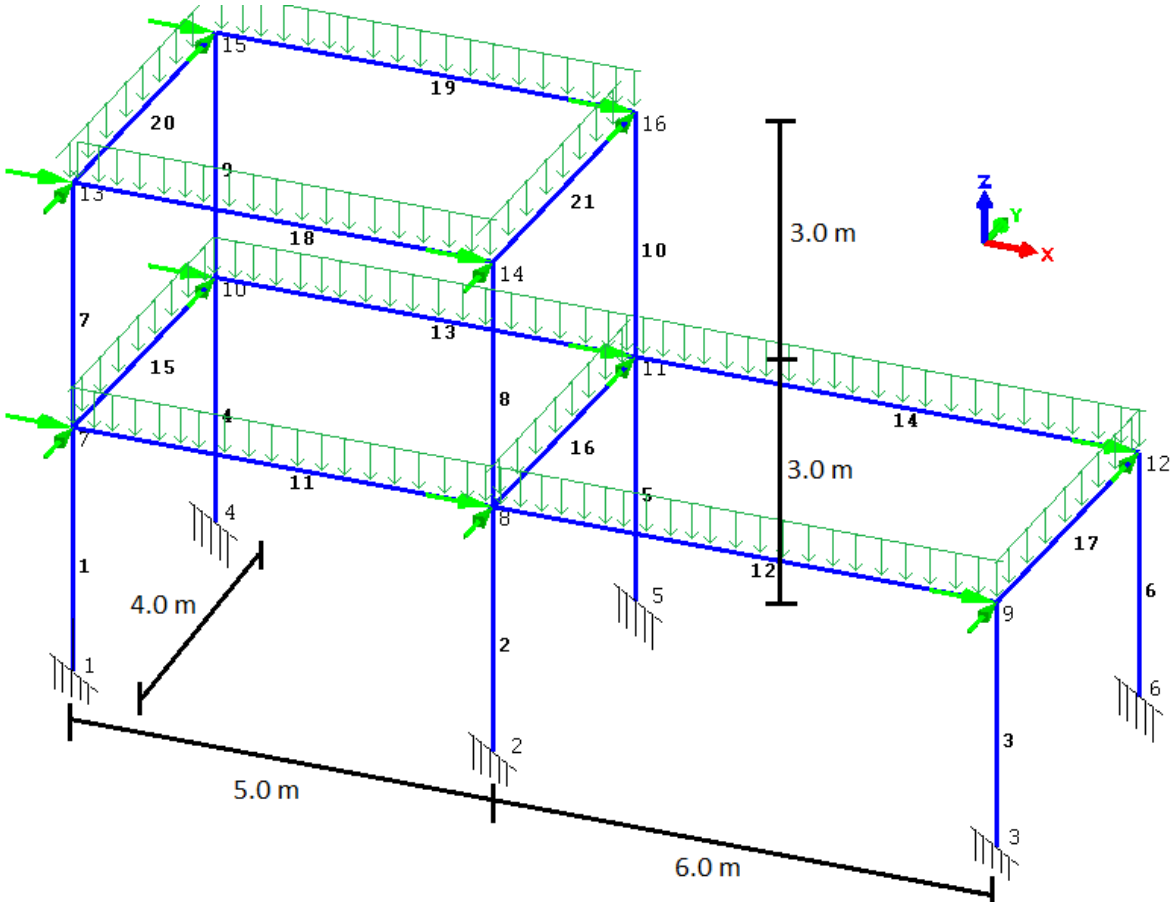
Σχήμα 5.10 Απόδοση της Conjugate Gradient σε CPU και GPU

Τα παραπάνω μεγέθη αποτελούν μέσους όρους 100 επαναλήψεων. Όπως αναφέρθηκε και στην εισαγωγή του κεφαλαίου, η σύγκριση αυτή δεν είναι αντικειμενική αφού η συγκεκριμένη CPU που χρησιμοποιήθηκε είναι πολύ χαμηλότερης ποιότητας από την αντίστοιχη GPU. Επιπλέον, η εφαρμογή της CG στην CPU είναι σειριακή και δεν εκμεταλλεύεται τους επιπλέον πυρήνες που διατίθενται (η συγκεκριμένη CPU έχει 2 πυρήνες, που είναι το ελάχιστο από όσες κυκλοφορούν). Επομένως η διαφορά της απόδοσης σε έναν υπολογιστή, ο οποίος έχει CPU και GPU παρόμοιας ποιότητας, δε θα είναι τόσο μεγάλη. Αξίζει πάντως να τονιστεί ότι η απόδοση της CPU εξαρτάται άμεσα και από τις υπόλοιπες διεργασίες που εκτελούνται παράλληλα, είτε αυτές είναι του λειτουργικού συστήματος είτε άλλα προγράμματα που χρησιμοποιεί ταυτόχρονα ο χρήστης. Στις παραπάνω δοκιμές, έγινε προσπάθεια να ελαχιστοποιηθεί η κατανάλωση υπολογιστικής ισχύος της CPU από άλλες διεργασίες. Στην πράξη όμως, ένα πρόγραμμα που χρησιμοποιεί όλους τους διαθέσιμους πυρήνες της CPU, θα ανταγωνίζεται με τις λοιπές διεργασίες που εκτελούνται ταυτόχρονα.

5.4 Αριθμητική εφαρμογή

Παρακάτω εφαρμόζεται η μέθοδος Βελτιστοποίησης Σμήνους Σωματιδίων για το βέλτιστο σχεδιασμό ενός τρισδιάστατου πλαισιωτού μεταλλικού φορέα. Γίνεται βελτιστοποίηση μεγέθους διατομών (sizing optimization), οι διαστάσεις των οποίων αποτελούν συνεχείς μεταβλητές. Η αντικειμενική συνάρτηση προς ελαχιστοποίηση είναι το βάρος της κατασκευής. Οι περιορισμοί του προβλήματος βελτιστοποίησης χωρίζονται σε δύο ομάδες: α) οι τάσεις στα πεπερασμένα στοιχεία να μην υπερβαίνουν τα όρια αντοχής του υλικού και β) οι οριζόντιες μετατοπίσεις των κόμβων του φορέα και τα κατακόρυφα βέλη των δοκών του να μην υπερβαίνουν τα όρια του Ευρωκώδικα (EN1993-1-1). Η κατασκευή αυτή φαίνεται στο επόμενο σχήμα.

Για τον υπολογισμό των παραπάνω μεγεθών προηγείται διακριτοποίηση του φορέα και γραμμική στατική ανάλυση FEM (με έναν μόνο υποφορέα). Η επίλυση του γραμμικού συστήματος, που αποτελεί το πλέον χρονοβόρο μέρος της ανάλυσης, γίνεται στην GPU με τη μέθοδο Conjugate Gradient και χρησιμοποιώντας CSR πίνακες. Όλα τα υπόλοιπα μέρη της ανάλυσης και της διαδικασίας βελτιστοποίησης εκτελούνται στη CPU, όπως περιγράφηκαν στην ενότητα **5.2**.



Σχήμα 5.11 Το 3D πλαίσιο που θέλουμε να βελτιστοποιήσουμε.

5.4.1 Μοντελοποίηση του φορέα

Στο παραπάνω σχήμα τα οριζόντια μήκη είναι από άξονα σε άξονα, ενώ τα κατακόρυφα μήκη είναι τα μεικτά ύψη κάθε ορόφου, δηλαδή συμπεριλαμβάνουν την κρέμαση των δοκών και το πάχος των πλακών, οι οποίες δεν προσομοιώνονται. Οι διατομές των γραμμικών στοιχείων είναι ορθογωνικές. Στην πραγματικότητα τα γραμμικά στοιχεία του πλαισίου είναι όγκοι σχήματος ορθογωνίου παραλληλεπιπέδου, με την μία διάσταση πολύ μεγαλύτερη από τις άλλες δύο. Οι όγκοι αυτοί συμβάλλουν στους κόμβους του πλαισίου, που είναι επίσης ορθογώνια παραλληλεπίπεδα, αλλά και οι τρεις διαστάσεις τους είναι παραπλήσιες, οπότε στο επόμενο σχήμα φαίνονται ως κύβοι.

Το πλαίσιο φορτίζεται από κατακόρυφα φορτία κατανεμημένα στις δοκούς, που αντιπροσωπεύουν μόνιμα και κινητά φορτία και οριζόντια φορτία, που παριστάνουν τη δύναμη αδράνειας στη διάρκεια ενός σεισμού (υπενθυμίζεται ότι δε γίνεται δυναμική ανάλυση). Οι πλάκες δεν προσομοιώνονται, αλλά η διαφραγματική λειτουργία τους λαμβάνεται υπόψη ως εξής: τα οριζόντια φορτία δεν εντείνουν κάθε δοκό καμπτικά, όπως τα κατακόρυφα, αλλά μεταφέρονται στους κόμβους του πλαισίου. Έτσι σε κάθε κόμβο του πλαισίου εφαρμόζονται οριζόντια φορτία κατά x και κατά y. Τα φορτία αυτά αντιστοιχούν στην αδράνεια της κάθε πλάκας και είναι ανάλογα του βάρους της άρα και των διαστάσεών της, που φαίνονται στο προηγούμενο σχήμα. Ο σεισμός σχηματίζει γωνία $30°$ με τον άξονα x, οπότε α φορτία κατά x είναι μεγαλύτερα από τα αντίστοιχα κατά y. Στους επόμενους πίνακες συνοψίζονται οι τιμές των οριζόντιων φορτίων σε κάθε κόμβο του πλαισίου και της συνισταμένης των κατακόρυφων κατανεμημένων φορτίων κάθε δοκαριού.

Κόμβος	Fx (kN)	Fy (kN)
7	42	24
8	91	53
9	50	29
10	42	24
11	91	53
12	50	29
13	42	24
14	42	24
15	42	24
16	42	24

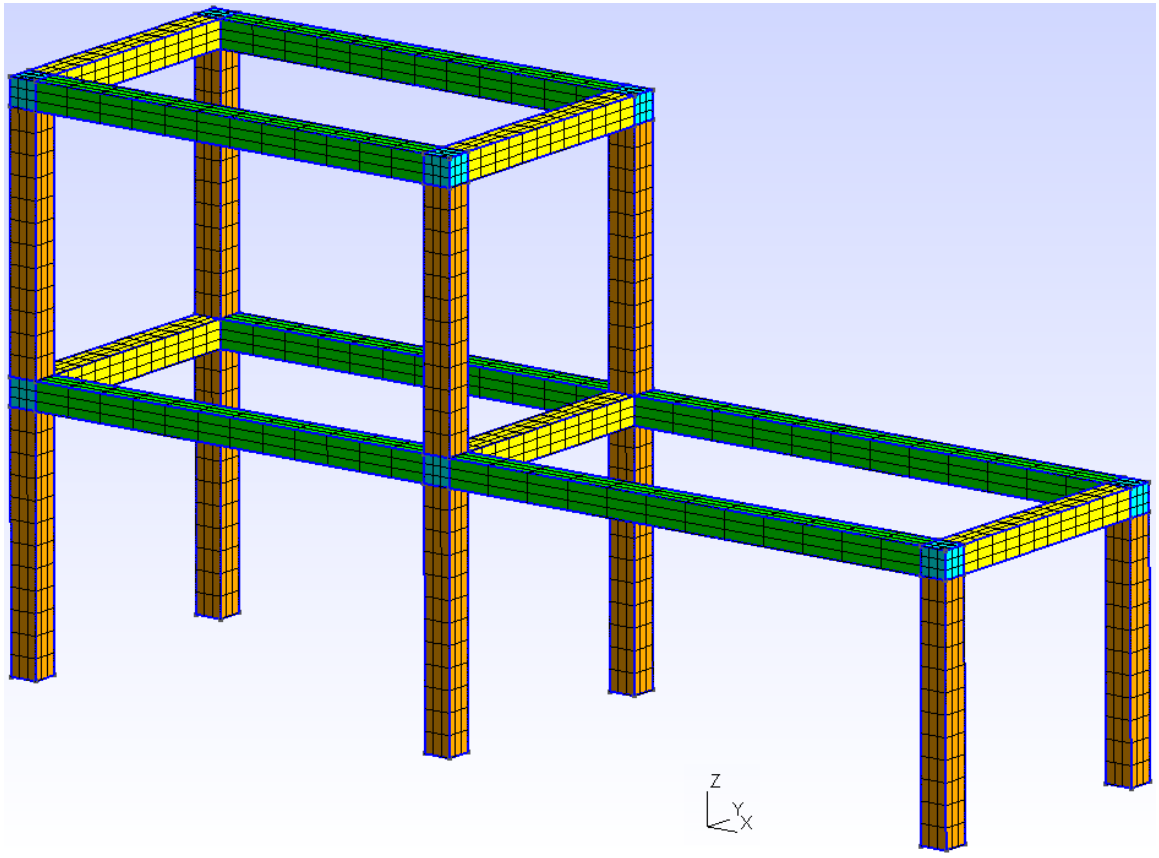
Πίνακας 5.1 Οριζόντια φορτία στους κόμβους του πλαισίου.

Δοκός	ΣFz (kN)
11	222
12	267
13	222
14	267
15	178
16	391
17	213
18	222
19	222
20	178
21	178

Πίνακας 5.2 Κατακόρυφα φορτία στις δοκούς του πλαισίου.

Το δίκτυο των πεπερασμένων στοιχείων δημιουργείται από το λογισμικό Gmsh. Οι κόμβοι του πλαισίου είναι οι πλέον καταπονούμενοι, οπότε το δίκτυο είναι πυκνότερο σε αυτούς από ότι στα υποστυλώματα και τις δοκούς. Το κατακόρυφο φορτίο σε κάθε δοκό εφαρμόζεται ως ένα σύνολο από επικόμβια φορτία στα πεπερασμένα στοιχεία, που αθροιστικά ισοδυναμούν με το συνολικό κατακόρυφο φορτίο της. Με τον ίδιο τρόπο μοιράζονται τα οριζόντια φορτία κατά x και κατά y στους κόμβους των πεπερασμένων στοιχείων, που συνθέτουν κάθε «κόμβο» του πλαισίου. Τα πεπερασμένα στοιχεία που χρησιμοποιούνται είναι ισοπαραμετρικά εξαπλευρικά οκτακομβικά στοιχεία συνεχούς μέσου (***Hexa 8***). Περισσότερες πληροφορίες σχετικά με αυτά δίνονται στο **Παράρτημα Α**.

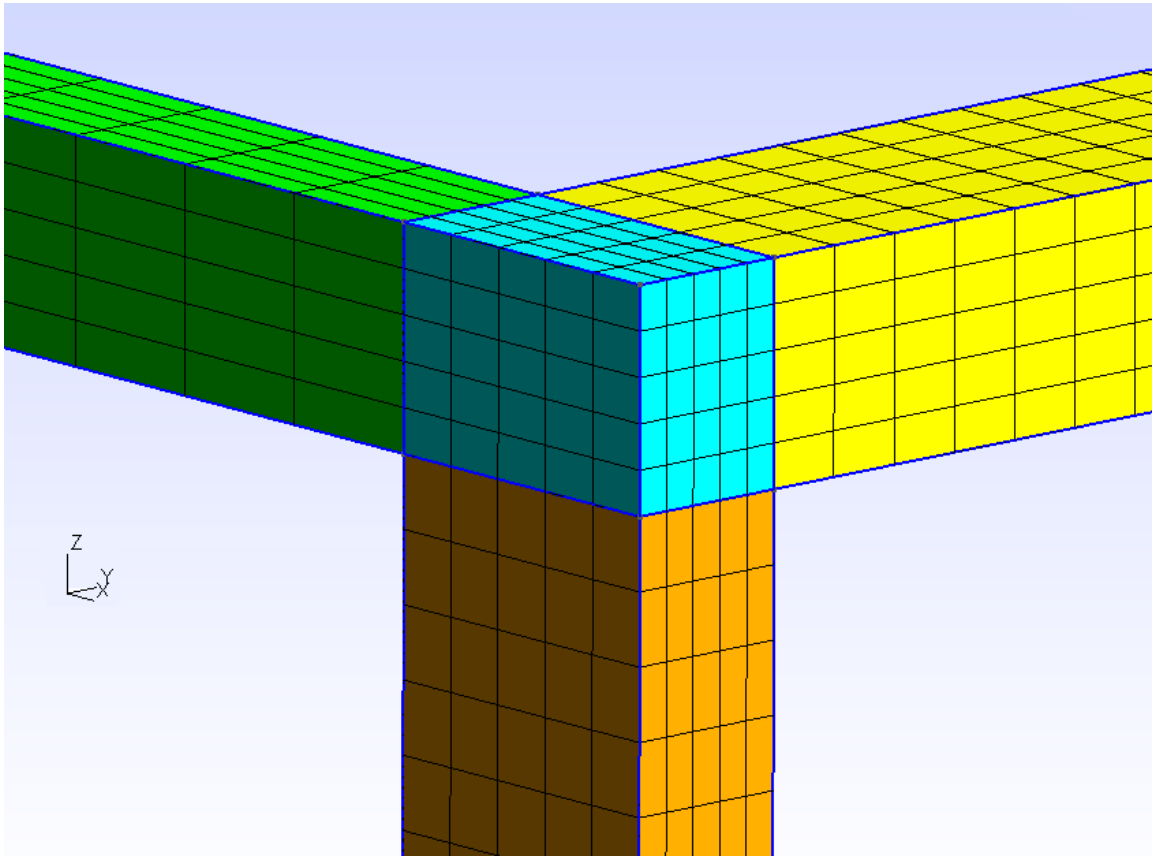
Το πλαίσιο στηρίζεται με πακτώσεις στα σημεία επαφής με το έδαφος. Για την προσομοίωση των πακτώσεων εφαρμόζονται δεσμεύσεις των 3 μεταφορικών βαθμών ελευθερίας κάθε κόμβου του mesh που έχει $z = 0$. Χρησιμοποιούνται στοιχεία συνεχούς μέσου, οπότε οι στροφικοί βαθμοί ελευθερίας είναι ανενεργοί και δεν έχει νόημα να δεσμευτούν.



Σχήμα 5.12 Finite element mesh της κατασκευής με χρήση του Gmsh. Οι κόμβοι, τα υποστυλώματα και οι δοκοί του πλαισίου φαίνονται με διαφορετικό χρώμα.

5.4.2 Προσδιορισμός προβλήματος και μεθόδου βελτιστοποίησης

Ως μεταβλητές σχεδιασμού επιλέγονται οι διαστάσεις των ορθογωνικών διατομών των δοκών και υποστυλωμάτων του πλαισίου. Πραγματοποιείται δηλαδή βελτιστοποίηση μεγέθους διατομών (βλέπε **1.5.1**). Για την απλούστερη γένεση του δικτύου πεπερασμένων στοιχείων, όλες οι δοκοί και τα υποστυλώματα, που συμβάλλουν σε οποιοδήποτε κόμβο, έχουν ίδιες διαστάσεις σε κάθε διεύθυνση, όπως φαίνεται στο επόμενο σχήμα. Αυτό μειώνει τον αριθμό των ανεξάρτητων μεταβλητών σχεδιασμού και την ποιότητα της βέλτιστης λύσης που θα βρεθεί. Ωστόσο, η σύνδεση μη ευθυγραμμισμένων πεπερασμένων στοιχείων αυξάνει την πολυπλοκότητα της δημιουργίας του mesh σε μη επιθυμητά επίπεδα για τις ανάγκες μιας απλής αριθμητικής εφαρμογής.



Σχήμα 5.13 Λεπτομέρεια κόμβου του πλαισίου. Οι αντίστοιχες διαστάσεις των γραμμικών στοιχείων είναι ίδιες.

Έτσι έχουμε συνολικά 7 συνεχείς μεταβλητές σχεδιασμού:

Μεταβλητή	Ελάχιστη επιτρεπόμενη τιμή (m)	Μέγιστη επιτρεπόμενη τιμή (m)	Περιγραφή
x_1	0.05	0.40	Πλάτος κατά x των υποστυλωμάτων 1, 4, 7, 9 και των δοκών 15, 20
x_2	0.05	0.40	Πλάτος κατά x των υποστυλωμάτων 2, 5, 8, 10 και των δοκών 16, 21
x_3	0.05	0.40	Πλάτος κατά x των υποστυλωμάτων 3, 6 και της δοκού 17
x_4	0.05	0.40	Πλάτος κατά y των υποστυλωμάτων 1, 2, 3, 7, 8 και των δοκών 11, 12, 18
x_5	0.05	0.40	Πλάτος κατά y των υποστυλωμάτων 4, 5, 6, 9, 10 και των δοκών 13, 14, 19
x_6	0.05	0.40	Κρέμαση (κατά -z) των δοκών 11, 12, 13, 14, 15, 16, 17
x_7	0.05	0.40	Κρέμαση (κατά -z) των δοκών 18, 19, 20, 21

Πίνακας 5.3 Μεταβλητές σχεδιασμού.

Ζητούμενο είναι η ελαχιστοποίηση του συνολικού βάρους της κατασκευής. Η αντικειμενική συνάρτηση είναι:

$$F(\boldsymbol{x}) = \gamma_s * [\sum_{beams}(A_{section} * L_{clear}) + \sum_{columns}(A_{section} * L_{total})] \quad \textbf{(5.3)}$$

Ο έλεγχος των περιορισμών του προβλήματος περιλαμβάνει:

1) Για κάθε κόμβο του πλαισίου, δοκό ή υποστύλωμα ελέγχεται η μέγιστη τάση Von Mises από αυτές που υπολογίζονται στα κέντρα όλων των πεπερασμένων στοιχείων του αντίστοιχου «όγκου»:

$$max\{\sigma^{VonMises}_{\text{στο κέντρο του στοιχείου}}\} \leq fy \quad \textbf{(5.4)}$$

2) Για κάθε κόμβο του πλαισίου ελέγχεται η μέγιστη οριζόντια μετατόπιση όλων των βαθμών ελευθερίας του κόμβου αυτού:

$$max\{\delta_h\} = max\left\{\sqrt{\delta_x^2 + \delta_y^2}\right\} \leq h_{ορόφου}/150 \quad \textbf{(5.5)}$$

3) Για κάθε δοκό ελέγχεται το βέλος της, δηλαδή η μέγιστη κατακόρυφη μετατόπιση όλων των βαθμών ελευθερίας που ανήκουν σε αυτή τη δοκό:

$$max\{\delta_v\} \leq L_{δοκού}/250 \quad \textbf{(5.6)}$$

Ο τρόπος υπολογισμού των ισοδύναμων τάσεων Von Mises στο κέντρο κάθε πεπερασμένου στοιχείου Hexa 8, αναλύεται στο **Παράρτημα Α**.

Το υλικό του φορέα είναι δομικός χάλυβας S235 με τις ακόλουθες ιδιότητες:

Ειδικό βάρος: $\gamma_s = 78.5\ kN/m^3$
Όριο διαρροής: $f_y = 235\ MPa$
Μέτρο ελαστικότητας: $E_s = 210\ GPa$
Λόγος Poisson: $v = 0.3$

Πίνακας 5.4 Ιδιότητες υλικού (χάλυβα) κατασκευής.

Για τη βελτιστοποίηση επιλέγεται η μέθοδος Βελτιστοποίησης Σμήνους Σωματιδίων, με τις ακόλουθες παραμέτρους:

NP	**Σμήνος**	$\mathbf{c_1}$	$\mathbf{c_2}$	$\mathbf{w_{min}}$	$\mathbf{w_{max}}$	$\mathbf{v_i^{max}}$
10	Gbest	2.0	2.0	0.0	0.9	$(u_i - \ l_i)/2$

Πίνακας 5.5: Παράμετροι της μεθόδου ΒΣΣ για την αριθμητική εφαρμογή.

Η αντιμετώπιση των περιορισμών γίνεται με τη μέθοδο συνάρτησης ποινής που περιγράφεται στην ενότητα **2.2.3**. Για τον τερματισμό της μεθόδου επιλέγονται τα κριτήρια που περιγράφονται στην ενότητα **2.4.4**, δηλαδή η μέθοδος τερματίζει όταν ο ρυθμός βελτίωσης των τελευταίων $\mathrm{k_f}$ επαναλήψεων είναι μικρότερος από ένα όριο $\mathrm{f_m}$ ή αν ξεπεραστεί ο επιτρεπόμενος αριθμός επαναλήψεων t_{max}.

Συνάρτηση ποινής	**Αρχική επιλογή**	$\mathbf{t_{max}}$	$\mathbf{k_f}$	$\mathbf{f_m}$
Γραμμική αύξηση ανάλογα με το βαθμό υπέρβασης του περιορισμού	Τυχαία στο διάστημα $[0.2, 0.4]$ για κάθε μεταβλητή	100	20	10^{-5}

Πίνακας 5.6: Αντιμετώπιση περιορισμών και κριτήρια τερματισμού της ΒΣΣ.

5.4.3 Αποτελέσματα διαδικασίας βελτιστοποίησης

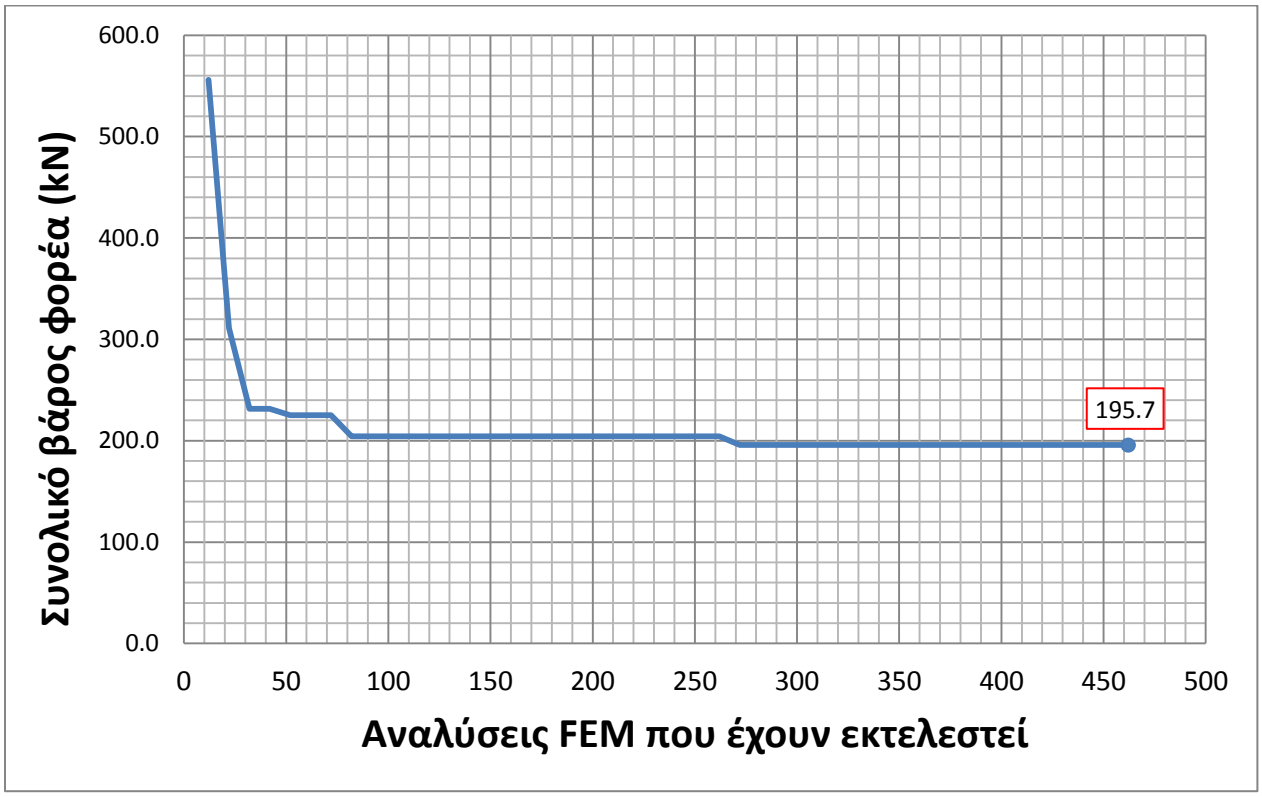
Τα αποτελέσματα της διαδικασίας βελτιστοποίησης με τη μέθοδο ΒΣΣ συνοψίζονται παρακάτω

Τελική Λύση	Αντίστοιχη τιμή αντικειμενικής συνάρτησης
[0.211536 , 0.234429 , 0.4 , 0.217789, 0.161147 , 0.05 , 0.189455]	195.695625

Πίνακας 5.7 Αποτελέσματα βελτιστοποίησης στο πρόβλημα της αριθμητικής εφαρμογής.

Αναλύσεις FEM που χρειαστήκαν	Συνολικός χρόνος εκτέλεσης (min)
$12\ (εκκίνηση) + 45\ (επαναλήψεις) * 10\ (πληθυσμός) = \mathbf{462}$	18.04

Πίνακας 5.8 Ταχύτητα της διαδικασίας βελτιστοποίησης.



Σχήμα 5.14 Διάγραμμα σύγκλισης της ΒΣΣ για το πρόβλημα της αριθμητικής εφαρμογής.

5.5 Παράλληλα μοτίβα για βελτιστοποίηση

5.5.1 Το μοντέλο pipeline

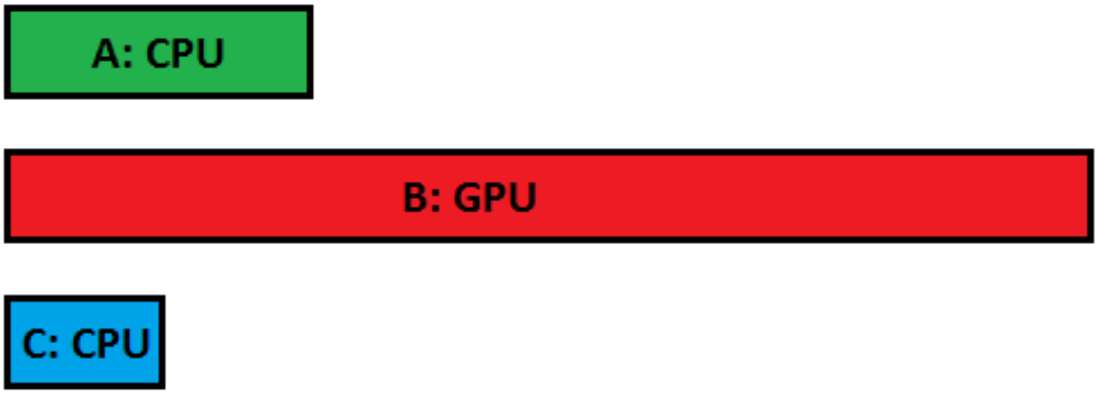
Στην διαδικασία που εφαρμόστηκε, για τον έλεγχο κάθε πιθανής λύσης από τη βελτιστοποίηση οι υπό-εργασίες μπορούν να χωριστούν σε 3 βήματα:

A) Εύρεση του νέου διανύσματος σχεδιασμού ανάλογα με τη μέθοδο βελτιστοποίησης, γένεση του δικτύου πεπερασμένων στοιχείων, δημιουργία του μοντέλου στο Solverize, υπολογισμός του μητρώου δυσκαμψίας και του διανύσματος δράσεων. Οι εργασίες αυτές εκτελούνται στην CPU.

B) Επίλυση του συστήματος γραμμικών εξισώσεων στην GPU.

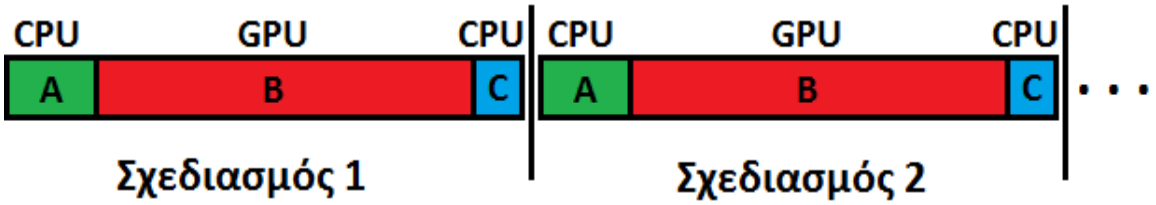
C) Υπολογισμός των μετατοπίσεων στους κόμβους ενδιαφέροντος και των τάσεων των πεπερασμένων στοιχείων (post-processing). Οι τιμές αυτές συγκρίνονται με τα όρια των περιορισμών του προβλήματος βελτιστοποίησης για να προκύψει η τιμή της συνάρτησης ποινής. Επίσης υπολογίζεται με αμελητέο κόστος και η αντικειμενική συνάρτηση. Οι εργασίες αυτές εκτελούνται στην CPU.

Η διάρκεια κάθε βήματος εξαρτάται από το πρόβλημα και την υπολογιστική ισχύ της CPU και της GPU. Γενικά το βήμα B είναι κατά πολύ πιο χρονοβόρο, γι' αυτό και αναλαμβάνεται από την GPU. Μια ποιοτική κατανομή της διάρκειας των επιμέρους βημάτων φαίνεται στο παρακάτω σχήμα.



Σχήμα 5.15 Τα επιμέρους βήματα της διαδικασίας βελτιστοποίησης σε CPU και GPU.

Αν και η επίλυση του γραμμικού συστήματος γίνεται με παράλληλο αλγόριθμο, η όλη διαδικασία είναι σειριακή. Για κάθε πιθανή λύση εκτελούνται με τη σειρά τα παραπάνω βήματα. Μόλις τελειώσουν, εκτελούνται τα βήματα για την επόμενη λύση κτλ.



Σχήμα 5.16 Σειριακή εξέταση των πιθανών λύσεων κατά τη βελτιστοποίηση.

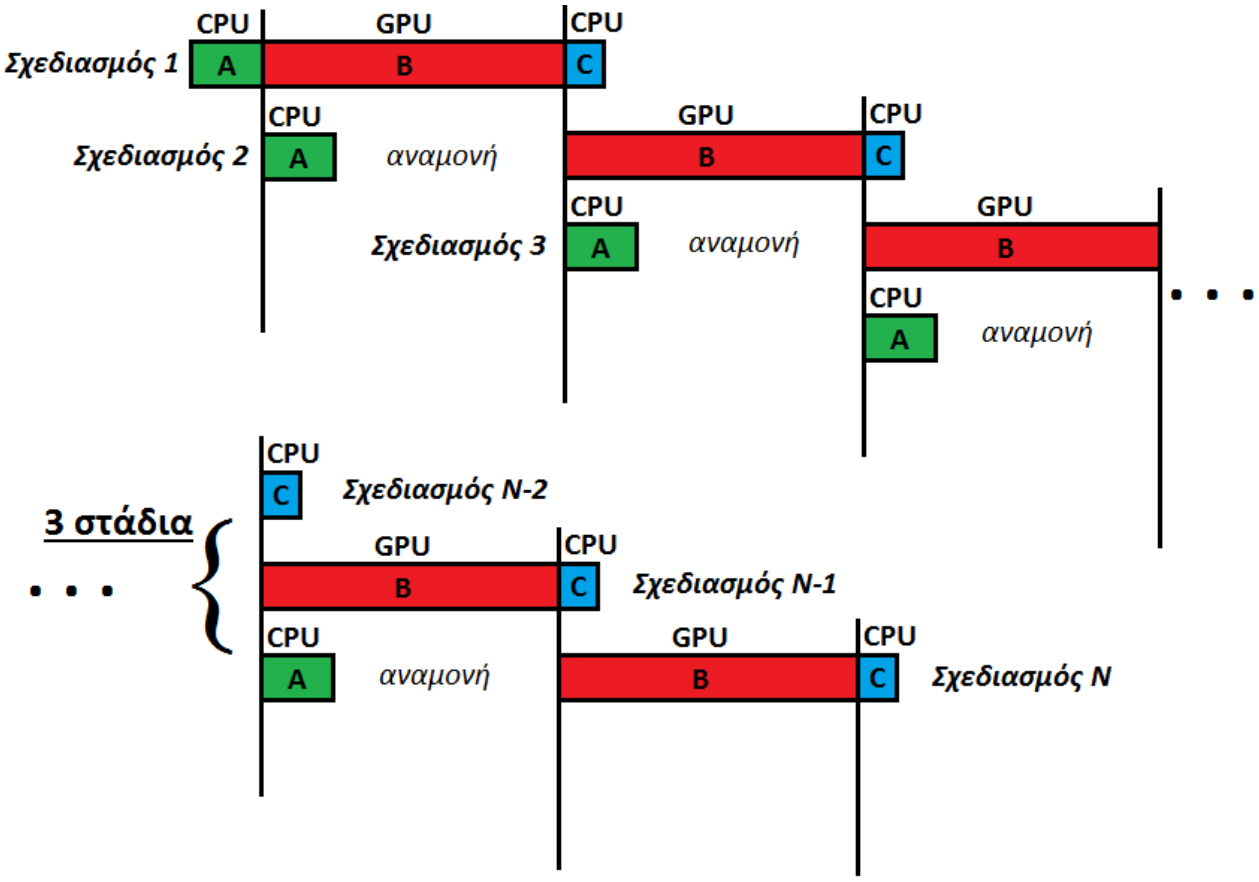
Μπορούμε να ελαττώσουμε τον υπολογιστικό χρόνο χρησιμοποιώντας τεχνικές του παράλληλου προγραμματισμού. Αυτό είναι δυνατό αφού:

- Χρησιμοποιούμε μεταευριστικούς αλγορίθμους βελτιστοποίησης. Για κάθε πιθανό σχεδιασμό τα 3 βήματα αναγκαστικά εκτελούνται το ένα μετά το άλλο. Οι σχεδιασμοί όμως που δοκιμάζονται σε κάθε γενιά/επανάληψη της μεθόδου βελτιστοποίησης είναι ανεξάρτητοι μεταξύ τους. Επομένως μπορούν να εκτελούνται παράλληλα τα βήματα δύο ή περισσότερων σχεδιασμών. Στην περίπτωση μαθηματικών αλγορίθμων που καθοδηγούν την αναζήτηση χρησιμοποιώντας παραγώγους δεν θα ήταν δυνατό κάτι τέτοιο.
- Σε ένα ετερογενές υπολογιστικό σύστημα υπάρχουν δύο επεξεργαστές, η CPU και η GPU που μπορούν να λειτουργούν ανεξάρτητα. Στην μέχρι τώρα διαδικασία κάθε χρονική στιγμή αξιοποιούταν η υπολογιστική ισχύς μόνο του ενός, ενώ ο άλλος ήταν αδρανής και περίμενε. Προφανώς, η CPU δεν ήταν εντελώς αδρανής, αφού εκτελούσε και άλλες διεργασίες του λειτουργικού συστήματος.
- Οι σύγχρονες CPU έχουν τουλάχιστον δύο πυρήνες που μπορούν να λειτουργούν ταυτόχρονα, αν το πρόγραμμα είναι παράλληλο. Μέχρι τώρα χρησιμοποιήθηκε μόνο ένας πυρήνας καθ' όλη τη διάρκεια του σειριακού προγράμματος.

Για να αξιοποιήσουμε τις παραπάνω δυνατότητες πρέπει να διαχωρίσουμε κατάλληλα τη σειριακή διαδικασία βελτιστοποίησης και να εφαρμόσουμε κάποιο μοντέλο παράλληλου προγραμματισμού. Σκοπός είναι η να μειωθεί στο ελάχιστο ο νεκρός χρόνος της GPU και των πυρήνων της CPU. Προφανώς χρειαζόμαστε κάποιο μοντέλο που υποστηρίζει MIMD παραλληλία (δηλαδή task parallelism), αφού κάθε αυτόνομη υπολογιστική μονάδα θα εκτελεί διαφορετικές εντολές.

Ως πρώτη προσέγγιση επιλέχθηκε η εφαρμογή της τεχνικής των pipelines. Η τεχνική αυτή εφαρμόζεται εδώ και δεκαετίες σε επίπεδο υλικού των CPU για την παράλληλη εκτέλεση στοιχειωδών εντολών (διάβασμα εντολής, αποκωδικοποίηση, προσπέλαση μνήμης, εκτέλεση αριθμητικής ή λογικής πράξης). Επίσης η ίδια λογική βρίσκεται πίσω από τη λειτουργίας της γραμμής παραγωγής. Σύμφωνα με αυτήν:

- Η συνολική εργασία που απαιτείται για την ολοκλήρωση ενός αντικειμένου χωρίζεται σε επιμέρους τμήματα. Στην περίπτωσή μας αντικείμενο είναι ο έλεγχος ενός σχεδιασμού και τα επιμέρους τμήματα είναι τα βήματα που περιγράφηκαν παραπάνω.
- Κατά τη διάρκεια της εκτέλεσης π.χ. του βήματος C ενός σχεδιασμού, εκτελείται παράλληλα το βήμα B του προηγούμενου και το βήμα A του προ-προηγούμενου. Επομένως τα pipelines έχουν 3 στάδια.
- Κάθε βήμα εκτελείται από διαφορετική μονάδα επεξεργασίας. Τα βήματα A και C εκτελούνται από διαφορετικούς πυρήνες της CPU, ενώ το βήμα B εκτελείται από την GPU. Οι πυρήνες της CPU και η GPU μπορούν να λειτουργούν ανεξάρτητα οπότε τα επιμέρους βήματα εκτελούνται παράλληλα.

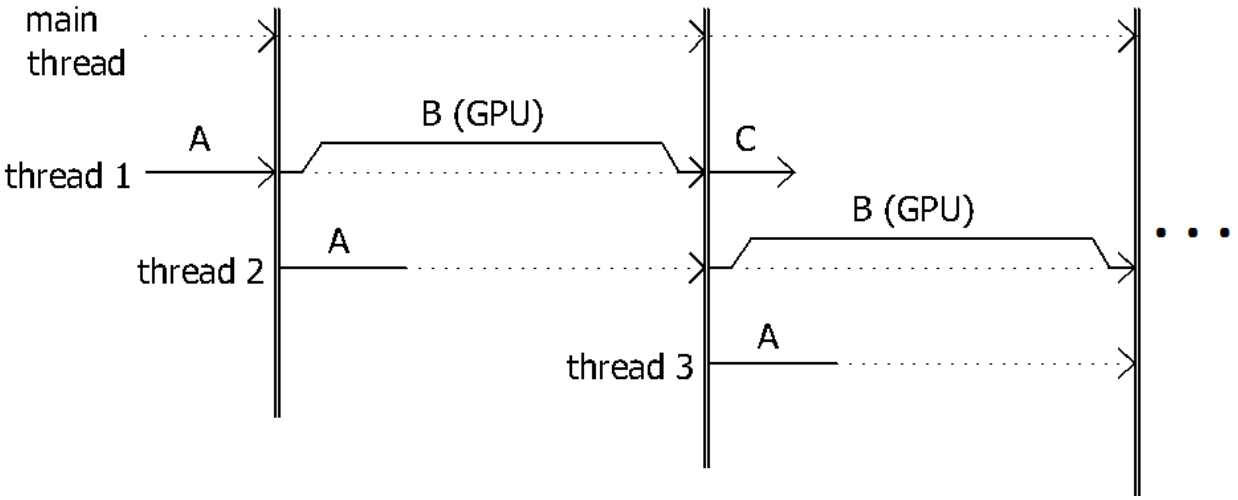


Σχήμα 5.17 Βελτιστοποίηση με pipelines.

Η απόδοση του μοντέλου pipeline εξαρτάται από αρκετούς παράγοντες που τελικά καθορίζουν την διάταξή του:

- Η διάρκεια κάθε επανάληψης του pipeline ταυτίζεται με την διάρκεια του πλέον χρονοβόρου βήματος. Εδώ το πλέον χρονοβόρο είναι το βήμα B, ακόμα και όταν η επίλυση γίνεται στην GPU. Επομένως η μείωση του υπολογιστικού χρόνου προέρχεται από την εκτέλεση των βημάτων A και C όσο η GPU είναι απασχολημένη με το B. Δεδομένου ότι η εκτέλεση των A και C μαζί διαρκεί συνήθως λιγότερο από την εκτέλεση του B, η χρήση δύο πυρήνων της CPU αντί για έναν μόνο δεν έχει κάποιο όφελος.
- Στα πρώτα και στα τελευταία στάδια δεν έχουν ενεργοποιηθεί όλοι οι υπολογιστικοί πόροι, οπότε λέμε ότι το pipeline «γεμίζει» ή «αδειάζει» αντίστοιχα. Ο συνολικός υπολογιστικός χρόνος αυξάνεται από την διάρκεια αυτών των σταδίων. Γενικά όσο περισσότεροι σχεδιασμοί δοκιμάζονται σε κάθε γενιά/επανάληψη της μεθόδου βελτιστοποίησης, τόσο μικρότερη είναι η επίδραση των πρώτων και τελευταίων σταδίων.
- Γενικά είναι επιθυμητό να χωρίζεται το πρόβλημα σε όσο το δυνατόν περισσότερα και ίσα σε διάρκεια τμήματα. Θεωρητικά αν χρησιμοποιηθούν N ίσα τμήματα που εκτελούνται ταυτόχρονα, ο συνολικός χρόνος μειώνεται στο 1/N του αρχικού. Η ιδανική αυτή περίπτωση δεν είναι εφικτή αφού ο διαχωρισμός σε ίσα τμήματα είναι δύσκολος ή αδύνατος. Ακόμα και αν επιτευχθεί όμως, η διάρκεια κάθε τμήματος δεν είναι ντετερμινιστική, επομένως η καθυστέρηση κάποιου τμήματος επηρεάζει όλο το στάδιο. Επιπλέον, ο κερματισμός σε πολλά τμήματα συνεπάγεται αυξημένες ανάγκες επικοινωνίας μεταξύ τους, που εισάγουν πιθανόν σημαντικές καθυστερήσεις.
- Γενικά είναι επιθυμητό τα επιμέρους τμήματα να εκτελούνται όλα ταυτόχρονα, δηλαδή ο αριθμός των σταδίων να είναι ίσος με τον αριθμό των τμημάτων. Ο στόχος αυτός περιορίζεται από τους διαθέσιμους υπολογιστικούς πόρους. Για παράδειγμα, η χρήση περισσότερων σταδίων από τους πυρήνες της CPU δεν μειώνει περισσότερο το συνολικό χρόνο υπολογισμού. Στην pipeline που εφαρμόστηκε, κρίσιμη είναι η GPU που μπορεί να επιλύει ένα γραμμικό σύστημα κάθε φορά.
- Επιπλέον η χρήση πολλών σταδίων σημαίνει, ότι συνυπάρχουν ταυτόχρονα πολλά μοντέλα πεπερασμένων στοιχείων και κυρίως γραμμικά συστήματα στη μνήμη. Για μεγάλο αριθμό βαθμών ελευθερίας, είναι πολύ εύκολο η απαίτηση σε μνήμη να ξεπεράσει την διαθέσιμη RAM του συστήματος και το πρόγραμμα να διακοπεί. Στην pipeline που εφαρμόστηκε, χρησιμοποιούνται 3 στάδια, ένα για κάθε βήμα A, B ή C. Το βήμα C απαιτεί συγκριτικά λίγη μνήμη αφού δεν αποθηκεύει κάποιο γραμμικό σύστημα. Επιπλέον στο βήμα B, το γραμμικό σύστημα μεταφέρεται αρχικά στη μνήμη της κάρτας γραφικών, οπότε η αντίστοιχη κεντρική μνήμη μπορεί να ελευθερωθεί, μόλις τελειώσει η αντιγραφή. Έτσι η συνολική απαίτηση σε μνήμη ισοδυναμεί περίπου με αυτή 2 γραμμικών συστημάτων.

Για την εφαρμογή των pipelines γράφτηκε multi-threaded κώδικας σε περιβάλλον Java, ώστε τα pipelines να ενσωματωθούν εύκολα στην υπάρχουσα διαδικασία βελτιστοποίησης. Η Java παρέχει μεγάλη ελευθερία στη χρήση threads και πολλές βοηθητικές classes για παράλληλο προγραμματισμό (στο πακέτο concurrency). Ο multi-threaded αλγόριθμος περιγράφεται παρακάτω:



Σχήμα 5.18 Εφαρμογή του pipeline με threads. Διακεκομμένη γραμμή σημαίνει ότι το thread περιμένει χωρίς να κάνει κάτι.

- Κάθε πιθανή λύση αναλαμβάνεται από ένα thread. Κάθε χρονική στιγμή είναι ενεργά τρία threads (3 στάδια) και το κεντρικό thread του προγράμματος. Όταν κάποιο thread περιμένει, δεν καταναλώνει υπολογιστικούς πόρους. Όταν δύο ή τρία threads είναι ταυτόχρονα ενεργά, μοιράζονται την υπολογιστική ισχύ των πυρήνων της CPU. Αυτό γίνεται αυτόματα από το λειτουργικό σύστημα (thread scheduling), επομένως αν εκτελούνται ταυτόχρονα και άλλες εφαρμογές, είναι πιθανόν να μην διατίθενται όλοι οι πυρήνες στα threads.
- Η επίλυση του γραμμικού συστήματος στην GPU (βήμα B) απαιτεί την μεταφορά δεδομένων μεταξύ της κεντρικής μνήμης και της μνήμης στην κάρτα γραφικών και κάποιους σποραδικούς ελέγχους σύγκλισης που εκτελούνται στη CPU. Επομένως το thread που ελέγχει το βήμα αυτό δεν μπορεί να συνεχίσει αν δεν ολοκληρωθεί το βήμα B (blocking operation). Έτσι παραμένει ανενεργό κατά το μεγαλύτερο μέρος του βήματος B και καταναλώνει ελάχιστους κύκλους της CPU.
- Για τον συντονισμό των threads χρησιμοποιείται φράγμα που αναγκάζει όσα threads φτάνουν σε αυτό πρώτα, να περιμένουν και την άφιξη των υπολοίπων, πριν συνεχίσουν στο επόμενο βήμα (barrier synchronization). Τα φράγματα αυτά φαίνονται με διπλή γραμμή στο παραπάνω σχήμα.

- Μόλις δοκιμαστούν όλοι οι σχεδιασμοί μιας γενιάς, το pipeline τερματίζει και το main thread συνεχίζει τις εργασίες βελτιστοποίησης, πριν ξεκινήσει το pipeline της επόμενης γενιάς.

5.5.2 Τροποποίηση για καλύτερη εκμετάλλευση της CPU

Η τεχνική pipeline όπως περιγράφηκε στην προηγούμενη ενότητα πετυχαίνει την εξάλειψη του νεκρού χρόνου της GPU, αφού τα βήματα της CPU ξεκινούν και τελειώνουν κατά τη διάρκεια της επίλυσης του γραμμικού συστήματος. Αυτό βέβαια ισχύει, εφόσον ο χρόνος εκτέλεσης των βημάτων Α και C από 2 πυρήνες της CPU είναι μικρότερος από τον συνολικό χρόνο εκτέλεσης του βήματος Β από την GPU. Αν η GPU είναι πολύ ισχυρότερη από την CPU και η επαναληπτική μέθοδος επίλυσης (εδώ χρησιμοποιείται η Conjugate Gradient) χρειάζεται λίγες επαναλήψεις, τότε είναι πιθανό το βήμα Β να τελειώσει πρώτο, οπότε η GPU θα περιμένει την CPU. Στη γενική περίπτωση όμως πλέον χρονοβόρο είναι το βήμα Β, ειδικά όταν η ανάλυση είναι μη γραμμική ή δυναμική.

Επομένως στη γενική περίπτωση η υπολογιστική ισχύς της CPU δεν αξιοποιείται πλήρως. Το προηγούμενο μοντέλο pipeline μπορεί να βελτιωθεί, αν επιτρέψουμε στο βήμα Α να ξεκινήσει τις επαναλήψεις της Conjugate Gradient (ή οποιασδήποτε άλλης επαναληπτικής μεθόδου) στην CPU, όσο η GPU είναι απασχολημένη με το βήμα Β του προηγούμενου σχεδιασμού. Μόλις η GPU ολοκληρώσει το βήμα Β, τότε διακόπτονται και οι επαναλήψεις της Conjugate Gradient (CG), που εκτελούνται στη CPU από το βήμα Α. Τα threads μπορούν πλέον να συνεχίσουν με την επόμενη επανάληψη του pipeline, αλλά τώρα η GPU θα έχει μειωμένο υπολογιστικό φόρτο, αφού μέρος των συνολικών επαναλήψεων της CG έχουν ήδη εκτελεστεί από το προηγούμενο βήμα. Έτσι η επανάληψη αυτή θα χρειαστεί λιγότερο χρόνο.

Παρατηρώντας τον αλγόριθμο της Conjugate Gradient φαίνεται ότι είναι εύκολο, να διακοπεί στο τέλος κάποιας επανάληψης και να συνεχιστεί μεταγενέστερα από την ίδια ή διαφορετική μονάδα επεξεργασίας, αν αποθηκευτεί η πρόοδος που έχει σημειωθεί μέχρι την επανάληψη αυτή. Η πρόοδος εκφράζεται από 3 διανύσματα: $\boldsymbol{x}$ (δοκιμαστική λύση), $\boldsymbol{r}$ (υπόλοιπο), p (κατεύθυνση αναζήτησης) και από το εσωτερικό γινόμενο $\boldsymbol{r^T * r}$ της προηγούμενης επανάληψης. Αφού οι επόμενες επαναλήψεις θα εκτελεστούν στην GPU, πρέπει να μεταφερθούν στην μνήμη της κάρτας γραφικών τα μεγέθη αυτά, καθώς και ο πίνακας του γραμμικού συστήματος **A** και ο νόρμα που χρησιμοποιείται για τον έλεγχο σύγκλισης $\boldsymbol{norm_2(r^T)}$. Ωστόσο δεν χρειάζεται να μεταφερθεί το διάνυσμα του δεξιού μέλους $\boldsymbol{b}$. Στο επόμενο σχήμα επαναλαμβάνεται ο αλγόριθμος της CG με σημειωμένα τα μεγέθη που πρέπει να αντιγραφούν. Σε σχέση με την απευθείας εκτέλεση της CG στην GPU, η αντιγραφή μνήμης είναι αυξημένη κατά δύο διανύσματα, αφού το κόστος μεταφοράς των βαθμωτών μεγεθών είναι πρακτικά μηδενικό και ο πίνακας θα χρειαζόταν να μεταφερθεί έτσι και αλλιώς. Όμως ακόμα και η προσαύξηση αυτή είναι αμελητέα συγκριτικά με τις υπόλοιπες εργασίες που εκτελούνται.

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$$
$$\mathbf{p}_0 := \mathbf{r}_0$$
$$k := 0$$
while (k ≤ kmax) repeat
$$\alpha_k := \frac{\mathbf{r}_k^{\mathsf{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathsf{T}}\mathbf{A}\mathbf{p}_k}$$
$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$$
$$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$$
$$\text{if } \frac{\text{norm2}(\mathbf{r}_{k+1})}{\text{norm2}(\mathbf{b})} \leq \varepsilon, \text{ then exit loop}$$
$$\beta_k := \frac{\mathbf{r}_{k+1}^{\mathsf{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathsf{T}}\mathbf{r}_k}$$
$$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$$
$$k := k + 1$$
end repeat
The result is $\mathbf{x}_{k+1}$

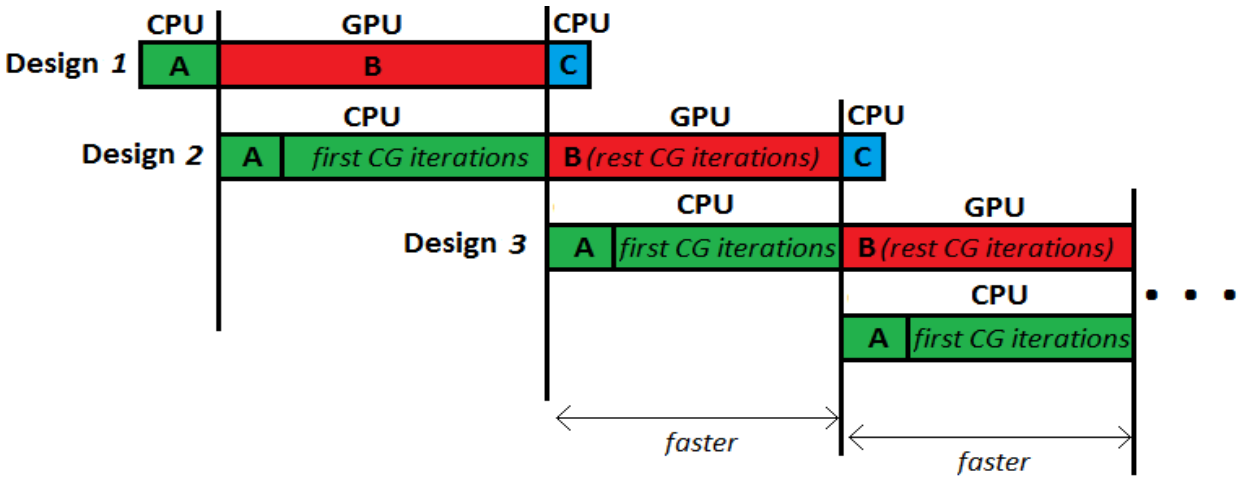
Σχήμα 5.19 Η μέθοδος Conjugate Gradient και τα μεγέθη που πρέπει να μεταφερθούν στην κάρτα γραφικών για να συνεχίσουν οι επαναλήψεις

Με τις αλλαγές αυτές, ο αλγόριθμος για κάθε thread τροποποιείται ως εξής:

Βήμα Α (CPU): Εύρεση νέας πιθανής λύσης, δημιουργία δικτύου και μοντέλου πεπερασμένων στοιχείων και υπολογισμός γραμμικού συστήματος. Επιπλέον: εκκίνηση της Conjugate Gradient στην CPU. Στο τέλος κάθε επανάληψης ελέγχεται αν η GPU χρησιμοποιείται από κάποιο άλλο thread. Μόλις η GPU ελευθερωθεί, διακοπή των επαναλήψεων της CG στην CPU και μετάβαση στο φράγμα.

Βήμα Β (GPU): Επίλυση του συστήματος γραμμικών εξισώσεων στην GPU, η οποία δεσμεύεται από αυτό το thread. Μεταφορά των απαραίτητων δεδομένων στη μνήμη της κάρτας γραφικών και εκτέλεση της CG από την αρχή ή από την επανάληψη που διακόπηκε στο βήμα Α. Μόλις ολοκληρωθεί η CG, ελευθέρωση της GPU και μετάβαση στο φράγμα. Στη σπάνια περίπτωση που η CG έχει ολοκληρωθεί από το βήμα Α, μετάβαση απευθείας στο φράγμα.

Βήμα C (CPU): Post-processing και υπολογισμός της αντικειμενικής συνάρτησης, των περιορισμών και της συνάρτησης ποινής.



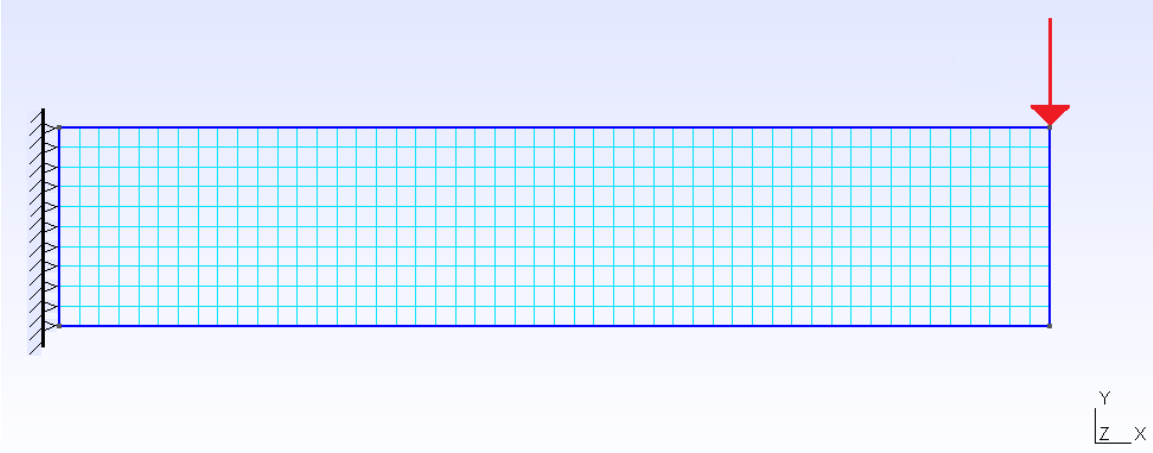
Σχήμα 5.20 Βελτιστοποίηση με pipelines χωρίζοντας την Conjugate Gradient

Με τις τροποποιήσεις αυτές εξαλείφεται ο νεκρός χρόνος του πυρήνα που εκτελεί το βήμα A σε κάθε σχεδιασμό, αφού η το διάστημα αναμονής που φαίνεται στο **Σχήμα 5.20** συμπληρώνεται με την εκτέλεση επαναλήψεων της Conjugate Gradient (ή οποιασδήποτε άλλης επαναληπτικής μεθόδου). Ταυτόχρονα η GPU χρειάζεται λιγότερες επαναλήψεις για να ολοκληρώσει την CG στο βήμα B. Ωστόσο οι επαναλήψεις της CG που εκτελούνται στην CPU είναι γενικά λιγότερες από αυτές που εκτελούνται στην GPU. Η βελτίωση στην απόδοση από το βασικό μοντέλο pipeline είναι μεγαλύτερη όσο αυξάνεται η υπολογιστική ισχύς της CPU συγκριτικά με την GPU. Έτσι η συνολική μέθοδος είναι αποτελεσματική και όταν η CPU είναι το ίδιο ή περισσότερο ισχυρή από την GPU. Υπενθυμίζεται ότι κατά τη μετάβαση από τον σειριακό αλγόριθμο σε CPU στον αλγόριθμο που εκμεταλλεύεται και την GPU, η βελτίωση της απόδοσης ήταν μεγαλύτερη, όταν η CPU ήταν λιγότερο ισχυρή από την GPU. Επομένως με το τροποποιημένο μοντέλο pipeline επιτυγχάνεται σημαντική βελτίωση της απόδοσης ανεξαρτήτως της σχετικής ισχύος των CPU και GPU.

Ωστόσο η CPU συνεχίζει να υποχρησιμοποιείται, αφού μόνο ένας πυρήνας είναι διαρκώς ενεργός. Ο πυρήνας που εκτελεί το βήμα C, ολοκληρώνει γρήγορα τις εργασίες του και είναι υποχρεωμένος να περιμένει. Επίσης οι υπόλοιποι πυρήνες, που μπορεί να έχει η CPU, δεν χρησιμοποιούνται καθόλου. Για αυτό πρέπει να αναζητηθούν άλλα παράλληλα μοτίβα που να εκμεταλλεύονται και την υπόλοιπη υπολογιστική ισχύ της CPU.

5.5.3 Εφαρμογή και σύγκριση απόδοσης

Στην ενότητα αυτή συγκρίνεται η απόδοση της βελτιστοποίησης σε CPU μόνο σειριακά, σε GPU χωρίς τη συνεργασία της CPU, καθώς και οι δύο τεχνικές παραλληλίας, που αναπτύχθηκαν προηγουμένως. Ως εφαρμογή επιλέχτηκε ένα απλό παράδειγμα δισδιάστατου προβόλου που προσομοιώνεται με τετραπλευρικά πεπερασμένα στοιχεία επίπεδης έντασης. Μεταβλητές σχεδιασμού είναι η κρέμαση της δοκού και το πάχος των στοιχείων, ενώ το μήκος του προβόλου είναι σταθερό. Ο πρόβολος καταπονείται από γνωστό σημειακό φορτίο στο ελεύθερο άκρο του. Το υλικό κατασκευής του είναι χάλυβας με σταθερές γνωστές ιδιότητες. Αντικειμενική συνάρτηση είναι το βάρος του προβόλου, ενώ οι συναρτήσεις περιορισμών αφορούν τη βύθιση του ελεύθερου άκρου και τις ισοδύναμες τάσεις Von Mises που αναπτύσσονται στα πεπερασμένα στοιχεία, όπως στην παράγραφο **5.4**.



Σχήμα 5.21 Δισδιάστατος πρόβολος

Λόγω της απλής γεωμετρίας, η γένεση του δικτύου δεν απαιτεί τη χρήση ειδικού προγράμματος, οπότε εξαλείφονται οι μεγάλες καθυστερήσεις από την επικοινωνία μεταξύ GMSH (preprocessor) και Solverize (processor). Έτσι η σύγκριση των μεθόδων είναι περισσότερο αντικειμενική, αφού ο χρόνος που δαπανάται στο βήμα Α είναι πιο ρεαλιστικός.

Εδώ πρέπει να ληφθεί υπόψη, ότι οι επαναληπτικές μέθοδοι επίλυσης όπως η Conjugate Gradient είναι ευαίσθητες στην αριθμητική κατάσταση του συστήματος. Δηλαδή, αν ένα γραμμικό σύστημα δεν έχει καλή κατάσταση, τότε η CG θα χρειαστεί πολύ περισσότερες επαναλήψεις. Η αξιολόγηση της κατάστασης ενός συστήματος ξεπερνά το πλαίσιο αυτής της εργασίας. Ποιοτικά, όσο πιο λυγηρός είναι ο πρόβολος, δηλαδή όσο μεγαλύτερο είναι το μήκος του σε σχέση με τις άλλες δύο διαστάσεις, τόσο χειρότερη είναι η κατάσταση του συστήματος.

Αυτό σημαίνει ότι ανάλογα με τις τιμές που παίρνουν οι μεταβλητές σχεδιασμού, το βήμα Β θα έχει διαφορετική διάρκεια. Δεδομένου ότι θα εφαρμοστεί ένας μεταευριστικός αλγόριθμος με στοχαστικούς τελεστές, η αλληλουχία των σχεδιασμών που δοκιμάζονται θα είναι πάντα διαφορετική. Επομένως, η σύγκριση των τεσσάρων τεχνικών δεν έχει νόημα, αφού ο αριθμός των επαναλήψεων της CG και άρα ο συνολικός υπολογιστικός φόρτος κάθε διαδικασίας βελτιστοποίησης δεν είναι ίδιος. Γι' αυτό, οι παράμετροι της μεθόδου βελτιστοποίησης επιλέγονται ώστε να εξαφανίζεται οποιαδήποτε τυχαιότητα και να δοκιμάζονται πάντα οι ίδιοι σχεδιασμοί. Έτσι καταστρέφεται η λειτουργία της μεθόδου βελτιστοποίησης, αλλά σε αυτό το στάδιο, δεν μας ενδιαφέρει η τελική λύση, αλλά η απόδοση του αλγορίθμου. Έχοντας υπόψη αυτές τις παρατηρήσεις, εφαρμόζεται η μέθοδος Βελτιστοποίησης Σμήνους Σωματιδίων με τις ακόλουθες παραμέτρους:

NP	**Σμήνος**	$\mathbf{c_1}$	$\mathbf{c_2}$	$\boldsymbol{w}$	**Αρχική επιλογή**	$\mathbf{t_{max}}$
10	Gbest	2.0	2.0	0.0	Όλοι οι σχεδιασμοί έχουν $x_1 = x_2 = 0.05$	10

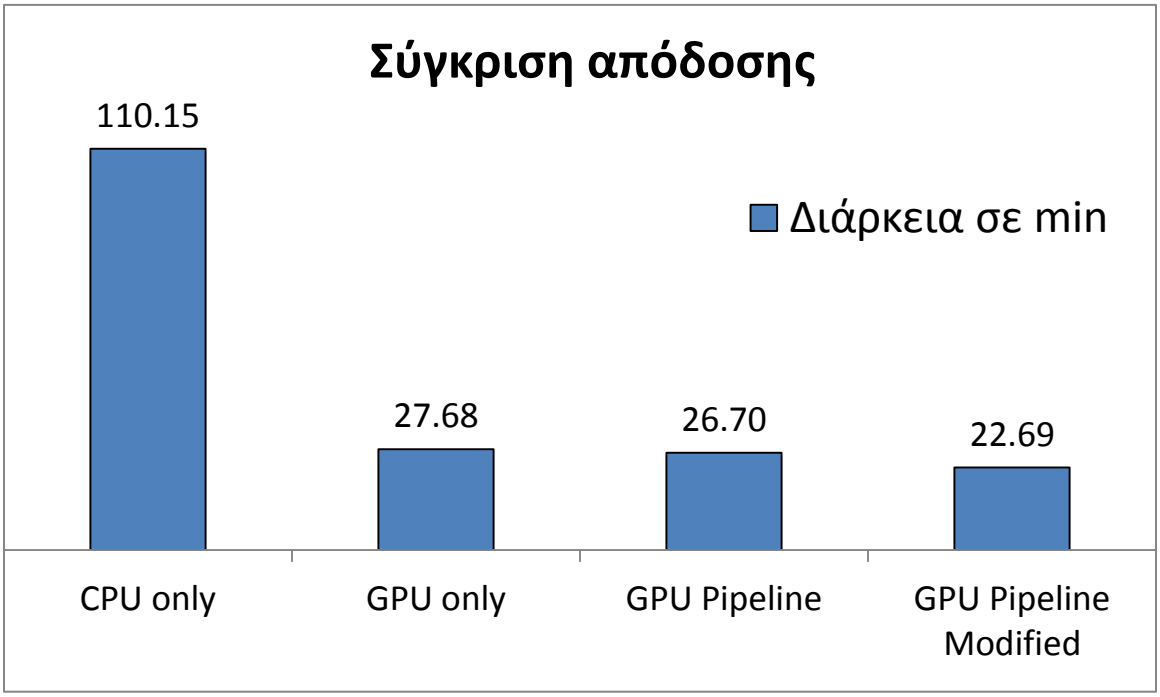
Πίνακας 5.9: Παράμετροι της μεθόδου ΒΣΣ για την αριθμητική εφαρμογή.

Το περιβάλλον εκτέλεσης είναι διαφορετικό από αυτό που περιγράφτηκε στην ενότητα **5.1.1** και χρησιμοποιήθηκε μέχρι τώρα. Τώρα χρησιμοποιείται τετραπύρηνος επεξεργαστής CPU με συχνότητα 4.4 GHz, και καλύτερη κάρτα γραφικών. Η αλλαγή περιβάλλοντος δοκιμών έγινε, εν μέρει για να χρησιμοποιηθούν πολλοί πυρήνες ώστε η οι διαδικασίες του λειτουργικού συστήματος να μην επηρεάζουν την απόδοση του προγράμματος, όπως θα γινόταν στη διπύρηνη CPU όταν θα επιστρατεύονταν και οι δύο πυρήνες του. Η υπολογιστική ισχύς της CPU αυξήθηκε περισσότερο από την ισχύ της GPU, οπότε οι διαφορές στις αποδόσεις που περιγράφονται στην παράγραφο **5.3.3** αναμένεται να είναι λιγότερο έντονες. Το νέο υπολογιστικό σύστημα είναι αρκετά ρεαλιστικό.

Τα αποτελέσματα της εκτέλεσης των τεσσάρων διαδικασιών βελτιστοποίησης παρουσιάζονται παρακάτω. Φαίνεται ότι η βασική βελτίωση στην απόδοση έναντι του παραδοσιακού σειριακού αλγορίθμου στη CPU, προέρχεται από τη χρήση GPU ακόμα και όταν η CPU είναι αξιόλογη. Η χρήση των παράλληλων μοτίβων, που επιστρατεύουν δύο πυρήνες της CPU, βελτιώνει την απόδοση, ειδικά όταν ο ένας πυρήνας είναι διαρκώς ενεργός. Η βελτίωση δεν είναι το ίδιο έντονη, αλλά δεν μπορεί να αμεληθεί. Να σημειωθεί ότι το σύστημα που επιλύεται είναι πολύ άσχημης κατάστασης (ill-conditioned). Σε ένα σύστημα καλύτερης κατάστασης, θα δαπανηθεί λιγότερος χρόνος στην επίλυσή του και τα βήματα Α, Β και C θα έχουν πιο κοντινές διάρκειες , επομένως η βελτίωση λόγω των pipelines θα είναι εντονότερη. Όμως αυτό αφορά γραμμική, στατική ανάλυση FEM. Σε μη γραμμικές ή δυναμικές αναλύσεις, το βήμα Β απαιτεί πολύ περισσότερο χρόνο από τα Α και C, οπότε τα μεγέθη που φαίνονται παρακάτω είναι αρκετά αντιπροσωπευτικά.

Βαθμοί Ελευθερίας	Αναλύσεις FEM	Επαναλήψεις CG στη CPU	Επαναλήψεις CG στη GPU
108000	100	28372	30289

Πίνακας 5.10 Μέγεθος προβλήματος



Σχήμα 5.22 Σύγκριση απόδοσης των διαφόρων τεχνικών βελτιστοποίησης σε ετερογενές υπολογιστικό περιβάλλον

Ποσοστιαία Μείωση Απαιτούμενου Χρόνου	
Σειριακός αλγόριθμος CPU → Σειριακός αλγόριθμος GPU	75%
Σειριακός αλγόριθμος GPU → Pipelines (βασική εκδοχή)	4%
Σειριακός αλγόριθμος GPU → Pipelines (βελτιωμένη εκδοχή)	18%
Σειριακός αλγόριθμος CPU → Pipelines (βελτιωμένη εκδοχή)	79%

Πίνακας 5.11 Ποσοστιαία μείωση απαιτούμενου χρόνου

Ανακεφαλαίωση και ιδέες για περεταίρω έρευνα

Στην παρούσα εργασία παρουσιάστηκαν οι μέθοδοι Βελτιστοποίησης Σμήνους Σωματιδίων, Διαφορικής Εξέλιξης και Στρατηγικών Εξέλιξης, δοκιμάστηκαν σε έναν αριθμό προβλημάτων αξιολόγησης και εφαρμόστηκαν για το βέλτιστο σχεδιασμό κατασκευών. Ταυτόχρονα έγινε προσπάθεια να αξιοποιηθεί η συνήθως ανεκμετάλλευτη αλλά εντυπωσιακή υπολογιστική ισχύς των GPU, για τη δραστική μείωση του χρόνου ανάλυσης με τη μέθοδο των πεπερασμένων στοιχείων. Έπειτα, ενσωματώθηκαν στο λογισμικό Solverize οι δυνατότητες βελτιστοποίησης και επίλυσης γραμμικών συστημάτων με χρήση GPU. Τέλος εφαρμόστηκε ο συνδυασμός των παραπάνω προγραμμάτων σε ένα πρόβλημα βελτιστοποίησης μεγέθους διατομών και έγινε σύγκριση με την αντίστοιχη σειριακή αντιμετώπιση σε CPU που χρησιμοποιείται παραδοσιακά.

Φυσική συνέχεια της εργασίας αυτής είναι η αξιοποίηση της GPU για μη γραμμικές, δυναμικές αναλύσεις FEM με τη χρήση υποφορέων. Στον κλάδο της βελτιστοποίησης, ενδιαφέρον έχει να ερευνηθεί η απόδοση των μαθηματικών μεθόδων βελτιστοποίησης, όπου το μεγαλύτερο υπολογιστικό κόστος έχει η ανάλυση ευαισθησίας, σε ετερογενή υπολογιστικά συστήματα CPU & GPU. Όσον αφορά τις μεταευριστικές μεθόδους, η εύρεση παράλληλων μοτίβων που να εκμεταλλεύονται πλήρως την υπολογιστική ισχύ του ετερογενούς συστήματος είναι το επόμενο αντικείμενο που θα ασχοληθεί ο γράφων.

Βιβλιογραφία

➢ Ελληνική

1. Πλεύρης Β. (2001). "Βέλτιστος σχεδιασμός κατασκευών με πολλαπλά κριτήρια με χρήση Στρατηγικών Εξέλιξης". *Μεταπτυχιακή διατριβή, Επιβλέπων Μ. Παπαδρακάκης, ΕΜΠ, Αθήνα*
2. Καρλαύτης Μ.Γ., Λαγαρός Ν.Δ. (2010). "Επιχειρησιακή έρευνα και βελτιστοποίηση για μηχανικούς", *Εκδόσεις Συμμετρία.*
3. Παπαδρακάκης Μ. (2001). "Ανάλυση φορέων με τη μέθοδο των πεπερασμένων στοιχείων", *Εκδόσεις Παπασωτηρίου.*
4. Πλεύρης Β., Λαγαρός Ν., Παπαδρακάκης Μ. (2008). "Εύρωστος βέλτιστος αντισεισμικός σχεδιασμός μεταλλικών κατασκευών". *3ο Πανελλήνιο Συνέδριο Αντισεισμικής Μηχανικής & Τεχνικής Σεισμολογίας.*
5. Μπούτσης Κ. (2014). "Βέλτιστος σχεδιασμός μεταλλικών κατασκευών με τη μέθοδο σμήνους σωματιδίων». *Διπλωματική εργασία, Επιβλέπων Μ. Παπαδρακάκης, ΕΜΠ, Αθήνα*
6. Κουμούσης Β. (1998). "Βέλτιστος σχεδιασμός των κατασκευών".

➢ Ξένη

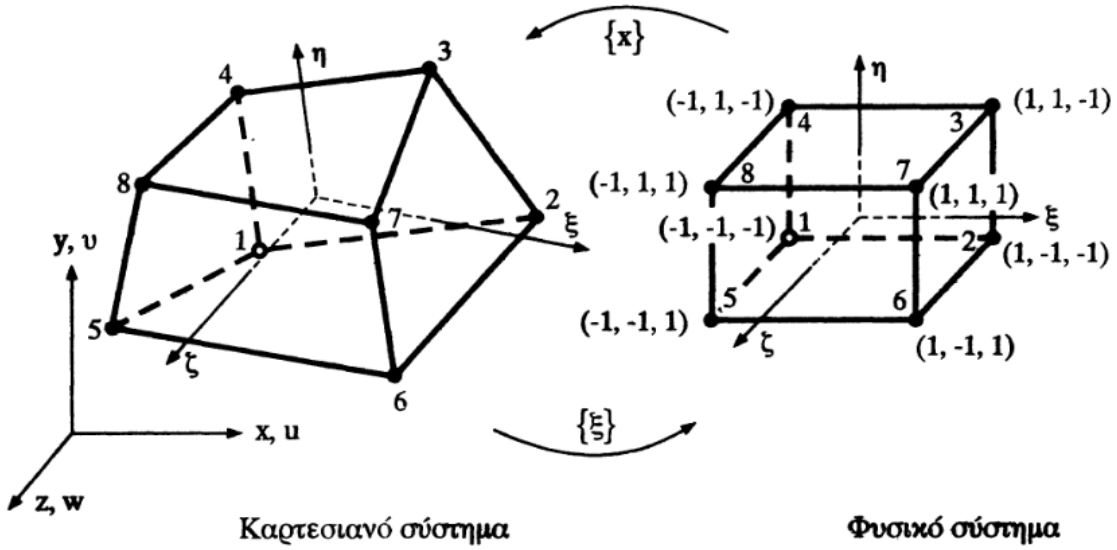
7. Pleuris V. (2009). "Innovative computational techniques for the optimum structural design considering uncertainties", *Διδακτορική διατριβή, Επιβλέπων Μ. Παπαδρακάκης, ΕΜΠ, Αθήνα.*
8. Mitropoulou C. (2011), "Advanced computational methods for seismic design and assessment of reinforced concrete structures", *Διδακτορική διατριβή, Επιβλέπων Μ. Παπδρακάκης, ΕΜΠ, Αθήνα.*
9. Kennedy J., Eberhart R. (1995). "Particle Swarm Optimization" *Proceedings of IEEE International Conference on Neural Networks.*
10. Shi Y., Eberhart R. (1998). "A modified particle swarm optimizer". *Proceedings of IEEE International Conference on Evolutionary Computation.*
11. Kennedy,, J., Mendes R. (2002). "Population structure and particle swarm performance". Proceedings of the 2002 Congress on Evolutionary Computation.
12. Browne P. A. (2013). "Topology optimization of linear elastic structures", *PhD thesis, University of Bath.*
13. Vanderplaats, G. B. (2006). "Structural optimization for statics, dynamics and beyond". *Proceedings of DINAME International Symposium on Dynamic Problems of Mechanics*.

14. Ahrari A., Atai A. A., Deb K. (2014). “Simultaneous topology, shape and size optimization of truss structures by fully stressed design based on evolution strategy”. *Engineering Optimization.*
15. Papadrakakis M., Lagaros N.D., Tsompanakis Y., Plevris V. (2001). “Large Scale Structural Optimization: Computational Methods and Optimization Algorithms”
16. Lagaros N.D., Papadrakakis M., Kokossalakis G. (2002). “Structural optimization using evolutionary algorithms”. Computers & Structures.
17. Venkayya V.B. (1971). “Design of Optimum structures”. *Computers & Structures.*
18. Michalewicz Z. (1995). “A survey of constraint handling techniques in evolutionary computation methods”. *Proceedings of the 4th Annual Conference on Evolutionary Programming*.
19. Koziel S., Michalewicz Z. (1999). “Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization, Evolutionary Computation”. *Evolutionary Computation.*
20. Coello C. A. (2007). “Constraint-handling techniques used with evolutionary algorithms”. *Proceedings of the 9th Annual Genetic and Evolutionary Computation Conference, GECCO.*
21. Coello C. A., Mentos E. M. (2002). “Constraint-handling in genetic algorithms through the use of dominance-based tournament selection”. *Advanced Engineering Informatics*
22. Parkinson A.R., Balling R., Hedengren J.D. (2013). “Optimization Methods for Engineering Design”. *Brigham Young University.*
23. Storn R., Price K. (1997). “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”. *Journal of Global Optimization.*
24. Storn R. (2008). “Differential evolution research - Trends and open questions”. *Studies in Computational Intelligence.*
25. Papadrakakis M., Lagaros N.D., Thierauf G., Cai J. (1998). ” Advanced solution methods in structural optimization based on evolutionary strategies”. *Engineering Computations*
26. Cai J., Thierauf G. (1996). “Evolution strategies for solving discrete optimization problems”. *Advances in Engineering Software.*
27. Beyer H. G., Schwefel H. P. (2002). “Evolution strategies: A comprehensive introduction”. *Natural computing*.
28. Bäck T., Schwefel H. P. (1991). “A survey of evolutionary strategies”. *Proceedings of the Fourth International Conference on Genetic Algorithms*.
29. Schwefel H. P., Rudolf G., Bäck T. (1995). “Contemporary evolution strategies”. *Proceedings of the Third European Conference on Artificial Life.*

30. Jamil M., Yang X. S. (2013). “A literature survey of benchmark functions for global optimization problems”. *Int. Journal of Mathematical Modelling and Numerical Optimization.*
31. Sanders J., Kandrot E. (2010). “CUDA by example. An introduction to general-purpose GPU programming.”
32. Kirk D.B., Hwu W.W. (2012). “Programming Massively Parallel Processors, Second Edition: A Hands-on Approach”.
33. Saad Y. (2003). “Iterative Methods for Sparse Linear Systems, Second Edition”. *Society for Industrial and Applied Mechanics.*
34. Mattson T. G., Sanders B. A., Massingill B.L. (2004). “Patterns for Parallel Programming (Software Patters Series), First Edition”. *Addison-Wesley Professional*.

Παράρτημα Α: Ισοπαραμετρικά εξαεδρικά στοιχεία οκτώ κόμβων

Στο παρακάτω σχήμα φαίνεται ένα εξαεδρικό ισοπαραμετρικό στοιχείο οκτώ κόμβων (Hexa 8) στο Καρτεσιανό και στο φυσικό σύστημα συντεταγμένων.



Σχήμα Α Εξαεδρικό ισοπαραμετρικό στοιχείο τρισδιάστατης ελαστικότητας οκτώ κόμβων

Συναρτήσεις σχήματος:

$$
\begin{aligned}
N_1 &= \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)\\
N_2 &= \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)\\
N_3 &= \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)\\
N_4 &= \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)\\
N_5 &= \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)\\
N_6 &= \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)\\
N_7 &= \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)\\
N_8 &= \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)
\end{aligned}
\qquad \textbf{(A.1)}
$$

Οι συντεταγμένες x, y, z και οι μετατοπίσεις u, v, w ενός σημείου του πεπερασμένου στοιχείου στο Καρτεσιανό σύστημα συντεταγμένο εκφράζονται συναρτήσει των αντίστοιχων επικόμβιων μεγεθών (του Καρτεσιανού συστήματος) με τη βοήθεια των συναρτήσεων σχήματος.

$$\begin{aligned} x &= \sum_{i=1}^{8} N_i x_i \\ y &= \sum_{i=1}^{8} N_i y_i \\ z &= \sum_{i=1}^{8} N_i z_i \\ u &= \sum_{i=1}^{8} N_i u_i \\ v &= \sum_{i=1}^{8} N_i v_i \\ w &= \sum_{i=1}^{8} N_i w_i \end{aligned} \qquad \textbf{(A.2)}$$

Ιακωμβιανό μητρώο της απεικόνισης:

$$[J] = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \\ \dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix} \qquad \textbf{(A.3)}$$

Και το αντίστροφο:

$$[J]^{-1} = \begin{bmatrix} J_{11}^{*} & J_{12}^{*} & J_{13}^{*} \\ J_{21}^{*} & J_{22}^{*} & J_{23}^{*} \\ J_{31}^{*} & J_{32}^{*} & J_{33}^{*} \end{bmatrix} \qquad \textbf{(A.4)}$$

Μητρώο παραμόρφωσης:

$$\underset{(6x24)}{[B]} = \underset{(6x9)}{[B_1]}\underset{(9x24)}{[B_2]} \quad \textbf{(A.5)}$$

Όπου:

$$[B_1] = \begin{bmatrix} J_{11}^* & J_{12}^* & J_{13}^* & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & J_{21}^* & J_{22}^* & J_{23}^* & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & J_{31}^* & J_{32}^* & J_{33}^* \\ J_{21}^* & J_{22}^* & J_{23}^* & J_{11}^* & J_{12}^* & J_{13}^* & 0 & 0 & 0 \\ 0 & 0 & 0 & J_{31}^* & J_{31}^* & J_{33}^* & J_{21}^* & J_{22}^* & J_{23}^* \\ J_{31}^* & J_{31}^* & J_{33}^* & 0 & 0 & 0 & J_{11}^* & J_{12}^* & J_{13}^* \end{bmatrix} \quad \textbf{(A.6)}$$

Και

$$[B_2] = \begin{bmatrix} N_{1,\xi} & 0 & 0 & N_{2,\xi} & 0 & 0 & \dots & N_{8,\xi} & 0 & 0 \\ N_{1,\eta} & 0 & 0 & N_{2,\eta} & 0 & 0 & \dots & N_{8,\eta} & 0 & 0 \\ N_{1,\zeta} & 0 & 0 & N_{2,\zeta} & 0 & 0 & \dots & N_{8,\zeta} & 0 & 0 \\ 0 & N_{1,\xi} & 0 & 0 & N_{2,\xi} & 0 & \dots & 0 & N_{8,\xi} & 0 \\ 0 & N_{1,\eta} & 0 & 0 & N_{2,\eta} & 0 & \dots & 0 & N_{8,\eta} & 0 \\ 0 & N_{1,\zeta} & 0 & 0 & N_{2,\zeta} & 0 & \dots & 0 & N_{8,\zeta} & 0 \\ 0 & 0 & N_{1,\xi} & 0 & 0 & N_{2,\xi} & \dots & 0 & 0 & N_{8,\xi} \\ 0 & 0 & N_{1,\eta} & 0 & 0 & N_{2,\eta} & \dots & 0 & 0 & N_{8,\eta} \\ 0 & 0 & N_{1,\zeta} & 0 & 0 & N_{2,\zeta} & \dots & 0 & 0 & N_{8,\zeta} \end{bmatrix} \quad \textbf{(A.7)}$$

Οι όροι $N_{i,\xi}, N_{i,\eta}, N_{i,\zeta}$ του μητρώου $[B_2]$ αποτελούν τις παραγώγους των συναρτήσεων σχήματος και δίνονται από τους τύπους:

$$\begin{aligned} N_{i,\xi} &= \pm\frac{1}{8}(1 \pm \eta)(1 \pm \zeta) \\ N_{i,\eta} &= \pm\frac{1}{8}(1 \pm \xi)(1 \pm \eta) \\ N_{i,\zeta} &= \pm\frac{1}{8}(1 \pm \xi)(1 \pm \eta) \end{aligned} \quad \textbf{(A.8)}$$

Μητρώο Στιβαρότητας: $[\mathrm{k}]\ (24 \times 24)$

$$[k] = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} [B(\xi,\eta,\xi)]^T * [E] * [B(\xi,\eta,\xi)] * \det[J] * d\xi\ d\eta\ d\zeta \qquad \textbf{(A.9)}$$

Όπου το μητρώο ελαστικότητας $[\mathrm{E}]\ (6 \times 6)$ είναι:

$$[E] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \qquad \textbf{(A.10)}$$

Αριθμητική ολοκλήρωση:

Για τον υπολογισμό του μητρώου στιβαρότητας χρησιμοποιείται η αριθμητική ολοκλήρωση κατά Gauss. Το ολοκλήρωμα της τρισδιάστατης συνάρτησης της σχέσης **(A.9)** προσεγγίζεται από το σταθμισμένο άθροισμα των τιμών της συνάρτησης σε κατάλληλα επιλεγμένα σημεία.

$$I = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta,\zeta)\, d\xi\ d\eta\ d\zeta = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \alpha_i * \alpha_j * \alpha_k * f(\xi_i,\eta_j,\zeta_k) \qquad \textbf{(A.11)}$$

Όπου:

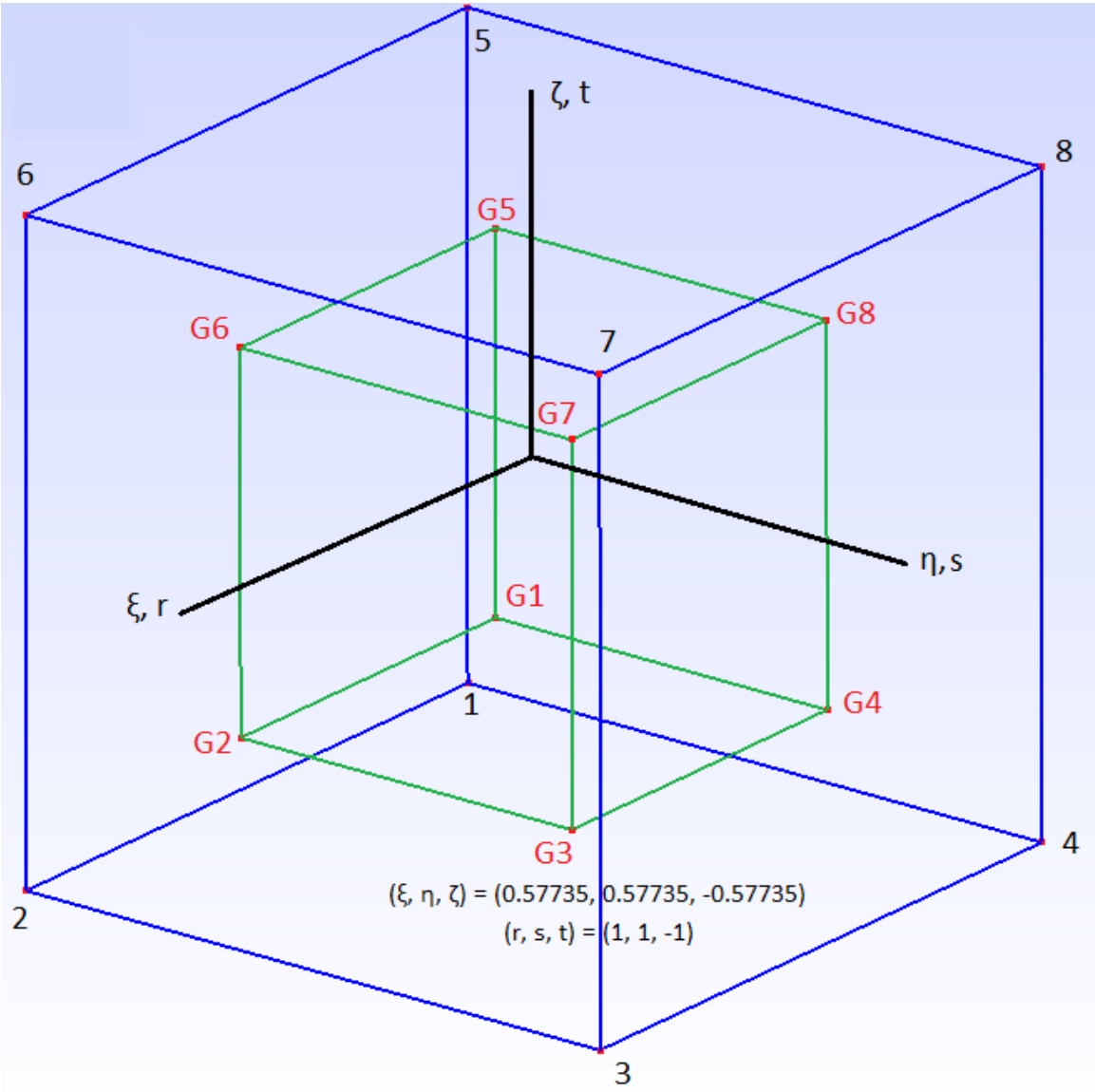
1) f είναι η υπό ολοκλήρωση συνάρτηση της σχέσης **(A.9)**
2) (ξ_i, η_j, ζ_k) είναι τα σημεία Gauss, στα οποία υπολογίζεται η τιμή της f
3) Το γινόμενο $\alpha_i * \alpha_j * \alpha_k$ είναι ο συντελεστής βάρους καθενός από τα σημεία Gauss
4) n είναι η τάξη της ολοκλήρωσης Gauss.

Για τα εξαεδρικά οκτακομβικά στοιχεία η απαιτούμενη τάξη είναι $n = 2$, ώστε να ελαχιστοποιούνται τα σφάλματα ολοκλήρωσης. Ο κανόνας ολοκλήρωσης Gauss 2x2 στις 3 διαστάσεις χρησιμοποιεί συνολικά 8 σημεία Gauss, στις θέσεις:

$$(\xi_i, \eta_j, \zeta_k) = \left(\pm\frac{1}{\sqrt{3}}, \pm\frac{1}{\sqrt{3}}, \pm\frac{1}{\sqrt{3}}\right) \quad \textbf{(A.12)}$$

με τον ίδιο συντελεστή βάρους σε όλα τα σημεία:

$$\alpha = \alpha_i * \alpha_j * \alpha_k = 1.0 * 1.0 * 1.0 = 1.0 \quad \textbf{(A.13)}$$



Σχήμα Β Σημεία Gauss στο Hexa 8

Υπολογισμός τάσεων:

Αφού επιλυθεί το γραμμικό σύστημα και βρεθούν οι μετατοπίσεις σε κάθε βαθμό ελευθερίας, το διάνυσμα των τάσεων υπολογίζεται από τη σχέση:

$$\{\sigma\} = [E]\,[B]\,\{d\} \quad \textbf{(A.14)}$$

Όπου

- $[E]\ (6 \times 6)$ και $[B]\ (6 \times 24)$ είναι τα προηγούμενα μητρώα ελαστικότητας και παραμόρφωσης αντίστοιχα
- $\{d\}\ (24 \times 1)$ είναι το διάνυσμα των επικόμβιων μετατοπίσεων του στοιχείου στο Καρτεσιανό σύστημα.
- $\{\sigma\}\ (6 \times 1)$ είναι το διάνυσμα των τάσεων σε ένα σημείο του στοιχείου.

$$\{\sigma\} = \{\sigma_{xx}\,,\quad \sigma_{yy}\,,\quad \sigma_{zz}\,,\quad \tau_{xy}\,,\quad \tau_{yz}\,,\quad \tau_{xz}\}^T \quad \textbf{(A.15)}$$

Η παραπάνω σχέση μπορεί να υπολογίσει τις τάσεις σε οποιοδήποτε σημείο του στοιχείου, αντικαθιστώντας στο μητρώο $[B]$ τις συντεταγμένες του σημείου στο φυσικό σύστημα (ξ, η, ζ). Ωστόσο το πεδίο τάσεων (όπως και το πεδίο παραμορφώσεων) προέρχεται από παραγώγιση του πεδίου μετατοπίσεων, οπότε έχει μειωμένη ακρίβεια. Γενικά οι τάσεις υπολογίζονται με μεγαλύτερη ακρίβεια στα σημεία Gauss. Έπειτα χρησιμοποιώντας τις συναρτήσεις σχήματος, γίνεται παρεμβολή στα σημεία που χρειάζονται στο εκάστοτε πρόβλημα.

Βέβαια οι συναρτήσεις σχήματος της ισοπαραμετρικής θεώρησης υπολογίζουν ένα οποιοδήποτε μέγεθος (θέση, μετατόπιση, τάση) σε κάποιο σημείο του στοιχείου, χρησιμοποιώντας τα αντίστοιχα επικόμβια μεγέθη, δηλαδή στα σημεία $(\xi, \eta, \zeta) = (\pm 1, \pm 1, \pm 1)$. Τα σημεία Gauss όμως βρίσκονται στο εσωτερικό του στοιχείου. Πριν χρησιμοποιηθούν οι συναρτήσεις σχήματος, ορίζεται ένα νέο σύστημα συντεταγμένων (r, s, t), στο οποίο τα σημεία Gauss έχουν συντεταγμένες $(r, s, t) = (\pm 1, \pm 1, \pm 1)$ και οι κόμβοι του στοιχείου $(r, s, t) = \left(\pm\sqrt{3}, \pm\sqrt{3}, \pm\sqrt{3}\right)$. Στο νέο αυτό σύστημα τα σημεία Gauss θεωρούνται κόμβοι ενός νέου «στοιχείου Gauss» στο εσωτερικό του ισοπαραμετρικού, όπως φαίνεται στο **Σχήμα Β**. Στο στοιχείο Gauss μπορούν να χρησιμοποιηθούν οι συναρτήσεις σχήματος του ισοπαραμετρικού στοιχείου που έχει τον ίδιο αριθμό κόμβων. Έτσι οι τάσεις στα σημεία Gauss παρεμβάλλονται σε σημεία του εσωτερικού του (interpolation) ή του εξωτερικού του (extrapolation), αρκεί να μην ξεπεραστούν τα όρια του ισοπαραμετρικού στοιχείου.

Τα εξαεδρικά ισοπαραμετρικά στοιχεία με 8 κόμβους έχουν 8 σημεία Gauss, άρα οι συναρτήσεις σχήματος είναι οι ίδιες στο ισοπαραμετρικό στοιχείο και στο «στοιχείο Gauss". Για τις ανάγκες της παρούσας εργασίας θα υπολογίζονται οι τάσεις στο κέντρο του στοιχείου:

$$(r_\kappa, s_\kappa, t_\kappa) = \sqrt{3}(\xi_\kappa, \eta_\kappa, \zeta_\kappa) = (0,0,0)$$
$$N_i(0,0,0) = \frac{1}{8}, \qquad i = 1,2,\dots 8 \tag{A.16}$$

Αφού βρεθεί το διάνυσμα των τάσεων στο κέντρο κάθε στοιχείου, υπολογίζεται η ισοδύναμη τάση Von Mises και ελέγχεται με το όριο διαρροής του υλικού. Για την περίπτωση της τρισδιάστατης ελαστικότητας ο έλεγχος αυτός γίνεται:

$$\sigma_V = \sqrt{\frac{1}{2}\left[\left(\sigma_{xx} - \sigma_{yy}\right)^2 + \left(\sigma_{yy} - \sigma_{zz}\right)^2 + (\sigma_{zz} - \sigma_{xx})^2 + 6\left(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2\right)\right]} \tag{A.17}$$
$$\leq fy$$