ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΛΟΠΟΙΗΣΗ ΤΕΧΝΙΚΩΝ ΚΑΘΟΛΙΚΗΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ (BRANCH-AND-BOUND) ΣΕ ΠΕΡΙΒΑΛΛΟΝ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ. ΕΦΑΡΜΟΓΗ ΣΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ.

Κωνσταντίνος Νασιώτης

Επιβλέπων: Σταύρος Π. Αδάμ

Επίκουρος Καθηγητής

Άρτα, Σεπτέμβριος 2019

IMPLEMENTATION OF GLOBAL OPTIMIZATION TECHNIQUES (BRAND-AND-BOUND) IN A MULTITHREADED ENVIRONMENT. APPLICATIONS IN NEURAL NETWORKS.

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 11/9/2019

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής
   Σταύρος Π. Αδάμ,

2. Μέλος Επιτροπής
   Χρήστος Γκόγκος,

3. Μέλος Επιτροπής
   Ευριπίδης Γλαβάς

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Νασιώτης, Κωνσταντίνος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω όλους τους καθηγητές για το έργο τους αυτά τα 4 χρόνια και ιδιαίτερα τον επιβλέποντα καθηγητή μου, Σταύρο Αδάμ για την συνεργασία, την υποστήριξη αλλά και για την ευκαιρία που μου επέτρεψε να συγγράψω την συγκεκριμένη πτυχιακή.

Περίληψη

Τα **ill-defined** προβλήματα, όπως αυτό της **αντιστροφής συνόλου**, λόγω του σύνθετου ορισμού τους, απαιτούν την εξερεύνηση τεχνικών **καθολικής βελτιστοποίησης** για την επίλυσή τους. Αυτές οι μέθοδοι με τη σειρά τους, αποτελούν δύσκολα προς επίλυση υπολογιστικά προβλήματα διότι αφορούν την αναζήτηση λύσης σε πολύ μεγάλους χώρους αναζήτησης. Συνεπώς, απαιτούν την εξερεύνηση μεθόδων περιορισμού (**Branch & Bound**) και στη συνέχεια την εφαρμογή **παράλληλων διαδικασιών** με στόχο την μείωση του απαραίτητου χρόνου για την εύρεση λύσης. Οι παράλληλες μέθοδοι, πέρα από προβλήματα διαχείρισης του φόρτου εργασίας, προϋποθέτουν την εύρεση βέλτιστων τεχνικών για την διαχείριση της μνήμης κατά την εκτέλεση. Η παρούσα πτυχιακή αποτελεί μελέτη των προαναφερθέντων τεχνικών και στη συνέχεια, την συγχώνευσή τους σε ένα ενιαίο λογισμικό. Τέλος, θα εκτιμηθούν οι επιδόσεις του λογισμικού σε συναρτήσεις όπως αυτή του Griewank με τελικό στόχο τη μελέτη των προοπτικών της χρήσης του για την επίλυση πολυδιάστατων προβλημάτων, όπως αυτά των **νευρωνικών δικτύων**.

Λέξεις-κλειδιά: Branch-and-Bound, SIVIA, Παράλληλες αρχιτεκτονικές, Νευρωνικά δίκτυα

Abstract

Ill defined problems such as the **Set Inversion problem**, because of their complicated definition, require the use of **Global Optimization** methods for the search of a solution. These methods have proven to be very hard to solve because the solution, requires exhaustively searching very large search spaces. So, on their own, they require the application of **Branch & Bound** methods along with **Parallel Computation** solutions, so that the required time for a solution to be found is reduced. These parallel methods, besides workload management, also require the application of intelligent memory management techniques. This dissertation's purpose is the development of an application which applies the previously mentioned methods so that it can be used for tougher multidimensional problems, such as **Neural Networks**.

Keywords: Branch-and-Bound, SIVIA, Parallel Architectures, Neural Networks

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Στόχοι και δομή της πτυχιακής

Οι στόχοι της πτυχιακής αφορούν την επίδειξη των δυνατοτήτων που προκύπτουν από την εφαρμογή Branch-and-Bound αλγορίθμων σε μαθηματικά προβλήματα όπως αυτό της αντιστροφής συνόλου με τεχνικές ανάλυσης διαστημάτων, την ανάλυση, τη περιγραφή παράλληλων αρχιτεκτονικών καθώς και τις επιδόσεις που προκύπτουν από αυτές, τη προοπτική εφαρμογής τους σε πολυδιάστατα προβλήματα ώστε να δοκιμαστούν πάνω σε σύνθετα μοντέλα, όπως αυτά των νευρωνικών δικτύων.

Δομή της Πτυχιακής

Στο πρώτο κεφάλαιο περιγράφεται το θέμα που πραγματεύεται η πτυχιακή. Γίνεται αναφορά στη δομή της με στόχο την προεοτιμασία του αναγνώστη για τις θεματικές ενότητες που θα ακολουθήσουν.

Το δεύτερο κεφάλαιο αφορά το θεωρητικό υπόβαθρο της πτυχιακής. Περιγράφεται το πρόβλημα της βελτιστοποίησης και σε τι αναλύεται, παρουσιάζονται βασικές έννοιες των Branch-and-Bound αλγορίθμων, μαζί με προβλήματα διαχείρισης μνήμης που προκύπτουν σε τέτοια προβλήματα και που προέκυψαν κατά την ανάπτυξη των εργαλείων που χρησιμοποιήθηκαν για την εκπόνηση της πτυχιακής. Στη συνέχεια περιγράφεται ο κλάδος της ανάλυσης διαστημάτων, πως γίνονται μαθηματικές πράξεις σε διαστήματα, πως λειτουργεί η συνάρτηση εγκλεισμού, έχοντας σαν στόχο την παρουσίαση του προβλήματος της αντιστροφής συνόλου μέσω ανάλυσης διαστημάτων (**SIVIA**).

Στο τρίτο κεφάλαιο αναλύονται χρήσιμες παράλληλες διαδικασίες που χρησιμοποιήθηκαν για την επίλυση του **SIVIA**, οι απαιτήσεις, η λογική αλλά και οι

περιορισμοί τους. Προς το τέλος του κεφαλαίου θα δούμε πως εμπλέκονται κι άλλες έννοιες, όπως το πρόβλημα του knapsack.

Το τέταρτο κεφάλαιο αναφέρεται στην εφαρμογή όλων των προηγούμενων εννοιών. Παρουσιάζονται οι απαιτήσεις σε εξοπλισμό, τα δοκιμαστικά προβλήματα που χρησιμοποιήθηκαν σαν είσοδος στον αλγόριθμο της αντιστροφής συνόλου, ο οποίος σε αυτό το στάδιο της εργασίας έχει παραλληλοποιηθεί και στο τέλους του κεφαλαίου παρουσιάζονται τα αποτελέσματα της εφαρμογής.

Στο πέμπτο κεφάλαιο γίνεται μία ανασκόπηση της πτυχιακής αλλά και μία καταγραφή εννοιών που δυστυχώς δεν μπόρεσαν να ολοκληρωθούν, όπως η υλοποίηση της εφαρμογής του παράλληλου αλγορίθμου σε ένα εκπαιδευμένο νευρωνικό δίκτυο. Γύρω από αυτές τις έννοιες, παρουσίαζονται κι άλλες πιθανές προσεγγίσεις-επεκτάσεις της πτυχιακής ως ανοιχτά προβλήματα με στόχο την επίδειξη της προοπτικής βελτίωσης του ήδη ανεπτυγμένου λογισμικού.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Βελτιστοποίηση

Ο όρος βελτιστοποίηση αναφέρεται στη διαδικασία που αφορά την εύρεση βέλτιστων λύσεων σε διάφορα προβλήματα, στην μοντελοποίηση, την οικονομία, τη φυσική κ.α. Λύση μπορεί να είναι η μέγιστη ή, η ελάχιστη τιμή μίας συνάρτησης $f(x)$. Τα κύρια **χαρακτηριστικά** που συνθέτουν ένα πρόβλημα βελτιστοποίησης είναι τρία[9]:

1. Ένα σύνολο αγνώστων ή μεταβλητών
2. Ο τύπος της συνάρτησης που ζητείται να βελτιστοποιηθεί
3. Οι περιορισμοί που ενδέχεται να ικανοποιεί η βέλτιστη λύση

Οι μεταβλητές μπορεί να λαμβάνουν συνεχείς ή διακριτές τιμές και η συνάρτηση βελτιστοποίησης μπορεί επίσης να είναι συνεχής ή διακριτή.

2.2 Κατηγορίες Βελτιστοποίησης

Η βελτιστοποίηση μπορεί να χωριστεί στις εξής κατηγορίες[11]:

- Υπό περιορισμούς και Χωρίς Περιορισμούς
- Διακριτή και Συνεχής
- Αιτιοκρατική και Στοχαστική

- Τοπική και Καθολική με την τελευταία να αφορά και τον στόχο της συγκεκριμένης πτυχιακής
- Γραμμική και μη-Γραμμική

2.3 Καθολική Βελτιστοποίηση

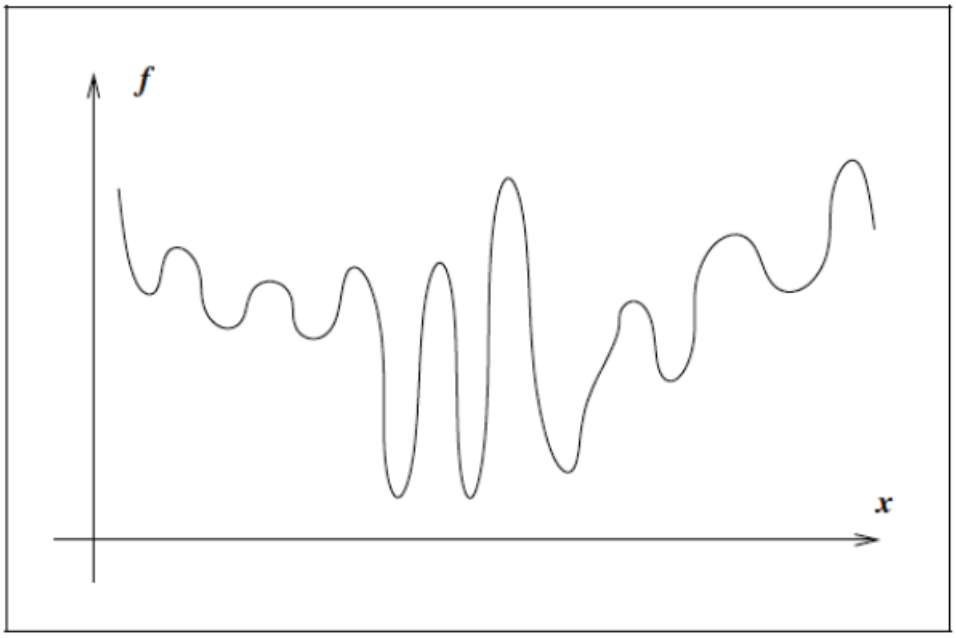
Συνήθως, οι αλγόριθμοι που χρησιμοποιούνται σε προβλήματα μη-γραμμικής βελτιστοποίησης ψάχνουν για τοπικές λύσεις, δηλαδή εκτελούνται σε περιορισμένους χώρους αναζήτησης. Πριν συνεχίσουμε, είναι απαραίτητο να αναφερθεί πως με τον όρο μη-γραμμική βελτιστοποίηση αναφερόμαστε σε προβλήματα βελτιστοποίησης όπου η αντικειμενική συνάρτηση δεν είναι γραμμική [17]. Για τα κυρτά προβλήματα προγραμματισμού και κυρίως για τα γραμμικα, οι τοπικές λύσεις είναι και ολικές [11].

Η **Καθολική Βελτιστοποίηση** είναι μία κατηγορία βελτιστοποίησης η οποία αφορά την εύρεση της βέλτιστης λύσης σε ολόκληρο το χώρο αναζήτησης. Θεωρείται ένα από τα δυσκολότερα προβλήματα βελτιστοποίησης και αρκετοί αλγόριθμοι είναι αρκετά χρονοβόροι ενώ πολλές φορές δεν συγκλίνουν στη βέλτιστη λύση.

2.4 Branch & Bound Αλγόριθμοι

Ένας Branch-and-Bound αλγόριθμος αποτελεί μία στρατηγική που συχνά χρησιμοποιείται στην επίλυση προβλημάτων Καθολικής Βελτιστοποίησης με ντετερμινιστικό τρόπο [8]. Αυτή η στρατηγική διαιρεί διαδοχικά τον χώρο αναζήτησης και ορίζει νέα όρια μέσω του υπολογισμού της αντικειμενικής συνάρτησης. Υπάρχει ένα υποσύνολο τέτοιων αλγορίθμων που αποκαλούνται **Branch & Prune**, οι οποίοι απορρίπτουν τις περιοχές που είναι αδύνατο να βρεθεί καθολική λύση. Η διαδικασία παράγει ένα δέντρο αναζήτησης, η ρίζα του οποίου αποτελεί τον αρχικό χώρο αναζήτησης. Ο αλγόριθμος εξερευνεί κόμβους ή υποδέντρα, που αποτελούν υποσύνολα του αρχικού χώρου αναζήτησης. Αν ο κόμβος μπορεί να περιέχει καλύτερη λύση από την καλύτερη που έχει βρεθεί μέχρι στιγμής διακλαδίζεται σε περαιτέρω υποσύνολα, αλλιώς, απορρίπτεται[8].

Οι μέθοδοι (ή αλγόριθμοι) Branch-and-Bound (B&B) αποτελούν πολύ γνωστά εργαλεία για την επίλυση NP-hard προβλημάτων βελτιστοποίησης[4].



Σχήμα 2.1: Ενδεικτικός χώρος αναζήτησης.

2.4.1 Κατανάλωση μνήμης ή ευστοχία αναζήτησης;

Μία μέθοδος Brand-and-Bound κατασκευάζει ένα δέντρο που αποτελείται από τις υποδιαιρέσεις του αρχικού χώρου αναζήτησης, οι οποίες κατά την εκτέλεση, συνήθως, αποθηκεύονται στην κύρια μνήμη. Πιο λεπτομερώς, οι B&B αλγόριθμοι λειτουργούν με ένα σύνολο από εκκρεμείς χώρους(και υποχώρους) αναζήτησης. Ο κάθε υποχώρος αποθηκεύεται, έως ότου έρθει η σειρά του να ελεγχθεί. Ο αριθμός των στοιχείων και η δομή του συνόλου εργασίας (working set) συνήθως καθορίζονται από την μέθοδο επιλογής που χρησιμοποιείται από τον αλγόριθμο[8]. Μία δομή δεδομένων με στρατηγική LIFO (Last In First Out), όπως η στοίβα, είναι κατάλληλη για αναζήτηση σε βάθος. Για την αναζήτηση του καλύτερου κόμβου μπορουν να χρησιμοποιθούν ταξινομημένες συνδεδεμένες λίστες, AVL δέντρα κ.α. Η αναζήτηση σε βάθος δεσμεύει λιγότερη μνήμη από το σύστημα αλλά εμπεριέχει τον κίνδυνο να οδηγήσει την αναζήτηση σε περιοχές του χώρου αναζήτησης που δεν περιέχουν βέλτιστες τιμές για κάποιο χρονικό διάστημα[8].

2.5 Ανάλυση Διαστημάτων

Επειδή η **Ανάλυση Διαστημάτων** αφορά υπολογισμούς πάνω σε σύνολα, στηρίχθηκε στη Θεωρία Συνόλων [12]. Οι μέθοδοι διαστημάτων έχουν χρησιμοποιηθεί εκτενώς για την επίλυση τόσο θεωρητικών αλλά και πρακτικών προβλημάτων. Μια από τις εφαρμογές της θεωρίας διαστημάτων αφορά την **Καθολική Βελτιστοποίηση**, την επαλήθευση αριθμητικών υπολογισμών, τη μοντελοποίηση της αβεβαιότητας και στην αντιμετώπιση αβέβαιων συστημάτων. Ως αποτέλεσμα, οι υπολογισμοί με διαστήματα έχουν αποδειχθεί σημαντικοι σε μία πληθώρα προβλημάτων, στα οποία η αβεβαιότητα ή τα υπολογιστικά σφάλματα μπορούν να μοντελοποιηθούν με διαστήματα. Σύμφωνα με τον Kreinovich [13], η υπολογιστική διαστημάτων κατατάσσεται στα NP-hard προβλήματα και ο μόνος τρόπος να αντιμετωπιστούν είναι η χρήση παράλληλων αλγορίθμων, όπου είναι εφικτό.

2.5.1 Ενδεικτικοί τελεστές

Στην ανάλυση διαστημάτων, ένα κουτί ορίζεται ως χώρος που αποτελείται από διαστηματικές συνιστώσες. Το πλάτος ενός κουτιού δίνεται από :

$$w([x]) = \max_{i=1,\ldots,n} \{\overline{x}_i - \underline{\mathrm{x}}_i\} \tag{2.1}$$

Οι τέσσερις κλασικές πράξεις πραγματικών αριθμών, η πρόσθεση, η αφαίρεση, ο πολλαπλασιασμός και η διαίρεση μπορούν να χρησιμοποιηθούν και σε διαστήματα. Για κάθε τέτοιο διαδυκό τελεστή, ορισμένο ως $\diamond$, η εκτέλεση πράξης με το $\diamond$ στα διαστήματα $[x]$ και $[y]$ σημαίνει τον υπολογισμό του συνόλου: [12]

$$[x] \diamond [y] = \{\, x \diamond y \in \mathbb{R} \mid x \in [x], y \in [y]\} \tag{2.2}$$

Παραδείγματα πράξεων σε διαστήματα (κουτιών μίας διάστασης) είναι οι εξής:

Πρόσθεση

$$[x] + [y] = [\underline{\mathrm{x}} + \underline{\mathrm{y}}, \overline{x} + \overline{y}] \tag{2.3}$$

Αφαίρεση

$$[x] - [y] = [\underline{\mathrm{x}} - \overline{y}, \overline{x} - \underline{\mathrm{y}}] \tag{2.4}$$

Πολλαπλασιασμός

$$\begin{aligned} [x] - [y] &= [a, b] \\ a &= \min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\} \\ b &= \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\} \end{aligned} \tag{2.5}$$

Διαίρεση

$$\frac{1}{[y]} = [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}], 0 \notin [y] \tag{2.6}$$

2.5.2 Συνάρτηση εγκλεισμού

Σε αυτήν την ενότητα, θα αναφερθούμε ενδεικτικά στις Inclusion Functions για να μπορέσουμε να κατανοήσουμε τον αλγόριθμο SIVIA που θα ακολουθήσει.

Έστω συνάρτηση f που ορίζεται από το $\mathbb{R}^n$ στο $\mathbb{R}^m$. Η συνάρτηση διαστήματος $[f]$ από το $\mathbb{R}^n$ στο $\mathbb{R}^m$ είναι μία συνάρτηση εγκλεισμού (Inclusion Function) της f αν [12]

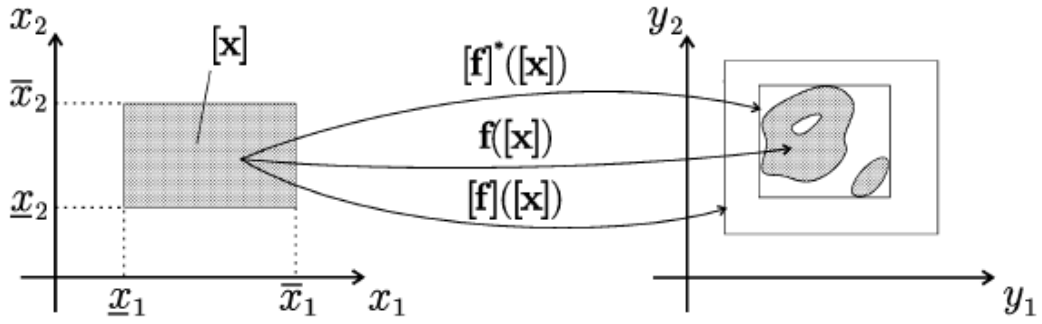
$$\forall [x] \in IR^n, f([x]) \subset [f]([x]). \tag{2.7}$$

Η συνάρτηση διαστήματος $[f]([x])$, για όλα τα $[x] \in IR^n$, αποτελεί παράδειγμα μίας inclusion function για όλες τις συναρτήσεις f από το R^n στο R^m. Ένας από τους σκοπούς της ανάλυσης διαστημάτων είναι η παροχή, για μία μεγάλη κατηγορία συναρτήσεων f, συναρτήσεων εγκλεισμού που μπορούν να εκτιμηθούν σχετικά γρήγορα και όπου η $[f]([x])$ προσεγγίζει βέλτιστα την εικόνα της. Η εικόνα της $f([x])$ μπορεί να έχει οποιοδήποτε σχήμα. Μπορεί να είναι μη-κυρτή ή ακόμα και μη συνεχής. Οποιοδήποτε σχήμα και να έχει η $f([x])$, μία inclusion function $[f]$ της f μας επιτρέπει να υπολογίσουμε ένα **κουτί** $[f]([x])$ που είναι εξασφαλισμένο πως θα εμπεριέχει την εικόνα της[12].
Στο σχήμα 2.2 βλέπουμε μία πολύ περίπλοκη περίπτωση της $f([x])$. Όμως, μπορούμε να δούμε επίσης πως περιλαμβάνονται όλα τα χαρακτηριστικά της f.

Συγκλίνουσα Συνάρτηση Εγκλεισμού

Μία Συνάρτηση Εγκλεισμού (Convergent Inclusion Function) $[f]$ είναι συγκλίνουσα, αν όλες οι συναρτησιακές της συντεταγμένες είναι συγκλίνουσες [12].

$$\lim_{k\to\infty} w([x](k)) = 0 \Rightarrow \lim_{k\to\infty} w([f]([x](k)) = 0. \tag{2.8}$$



Σχήμα 2.2: Οι εικόνες ενός κουτιού από μία διανυσματική συνάρτηση f και δύο από τις inclusion functions f, $[f]*$ [12].

2.5.3 Αντιστροφή συνόλου μέσω ανάλυσης διαστημάτων-SIVIA

Έστω οτι έχουμε μία συνάρτηση f, που ενδέχεται να μην είναι γραμμική, που ορίζεται από το R^n στο R^m και έστω Y υποσύνολο του R^m. Το πρόβλημα της αντιστροφής συνόλου **Set Inversion** χαρακτηρίζεται ως:

$$X = \{x \in R^n | f(x) \in Y\} = f^{-1}(Y). \tag{2.9}$$

Για οποιοδήποτε $Y \subset R^m$ και για οποιαδήποτε συνάρτηση f για την οποία ορίζεται μία συνάρτηση εγκλεισμού (inclusion function) $[f]()$, δύο κανονικά υποδιαστήματα $\underline{\mathrm{X}}$ και $\overline{X}$ όπως:

$$\underline{\mathrm{X}} \subset X \subset \overline{X} \tag{2.10}$$

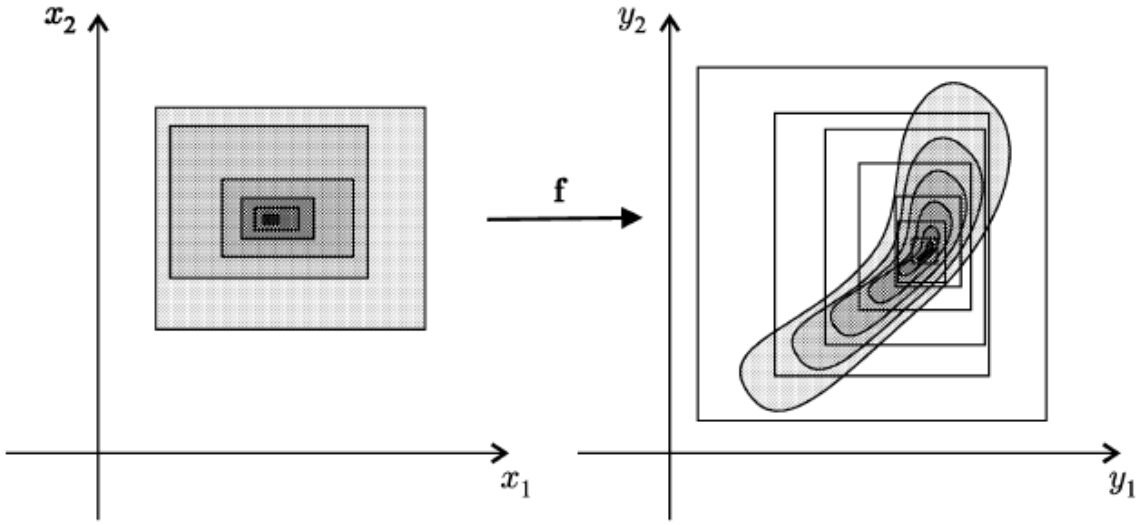
μπορούν να παρθούν από τον αλγόριθμο SIVIA (**S**et **I**nverter **V**ia **I**nterval **A**nalysis, Jaulin and Walter, 1993a and 1993c) [14]

Algorithm 1 SIVIA(in:$f, Y, [x], \epsilon$; inout:, $\underline{\mathrm{X}}, \overline{X}$)

---

if $[f]([x]) \cap Y = \emptyset$ **then return** ;
else if $[f]([x]) \subset Y$ **then**
$\{\underline{\mathrm{X}} := \underline{\mathrm{X}} \cup [x]; \overline{X} := \overline{X} \cup [x]$; return$\}$;
else if $w([x]) < \epsilon$ **then** $\{ \overline{X} := \overline{X} \cup [x]$; return; $\}$;
end if
SIVIA($f, Y, L[x], \epsilon, \underline{\mathrm{X}}, \overline{X}$); SIVIA($f, Y, R[x], \epsilon, \underline{\mathrm{X}}, \overline{X}$)

Ο **SIVIA**, στη βασική του εκδοχή, είναι ένας αναδρομικός αλγόριθμος ο οποίος ξεκινάει ορίζοντας ένα πολύ μεγάλο **κουτί** αναζήτησης $[x](0)$ του οποίου $\overline{X}$ είναι σίγουρο πως θα περιέχεται. Η γενική διαδικασία είναι η εξής [12]:



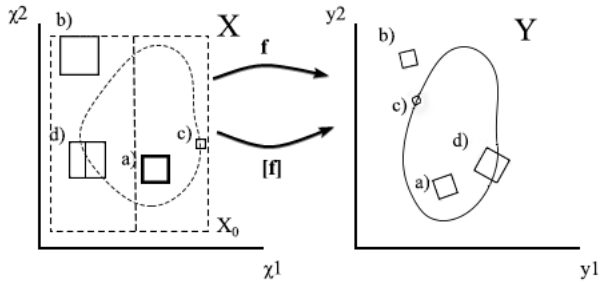
Σχήμα 2.3: Μία συγκλίνουσα συνάρτηση εγκλισμού [12].

1. Αν η $[f]([x])$ έχει μία μη-κενή τομή με το Y, αλλά δεν ανήκει εξ'ολοκλήρου στο Y, Τότε το $[x]$ μπορεί να περιέχει μέρος του συνόλου λύσης. Το $[x]$ ορίζεται ως μη-καθορισμένο. Αν έχει πλάτος w μεγαλύτερο από μία ορισμένη παράμετρο ϵ, τότε θα διχοτομηθεί και ο ελέγχος θα επαναληφθεί στα δύο νέα κουτιά που παράχθηκαν.

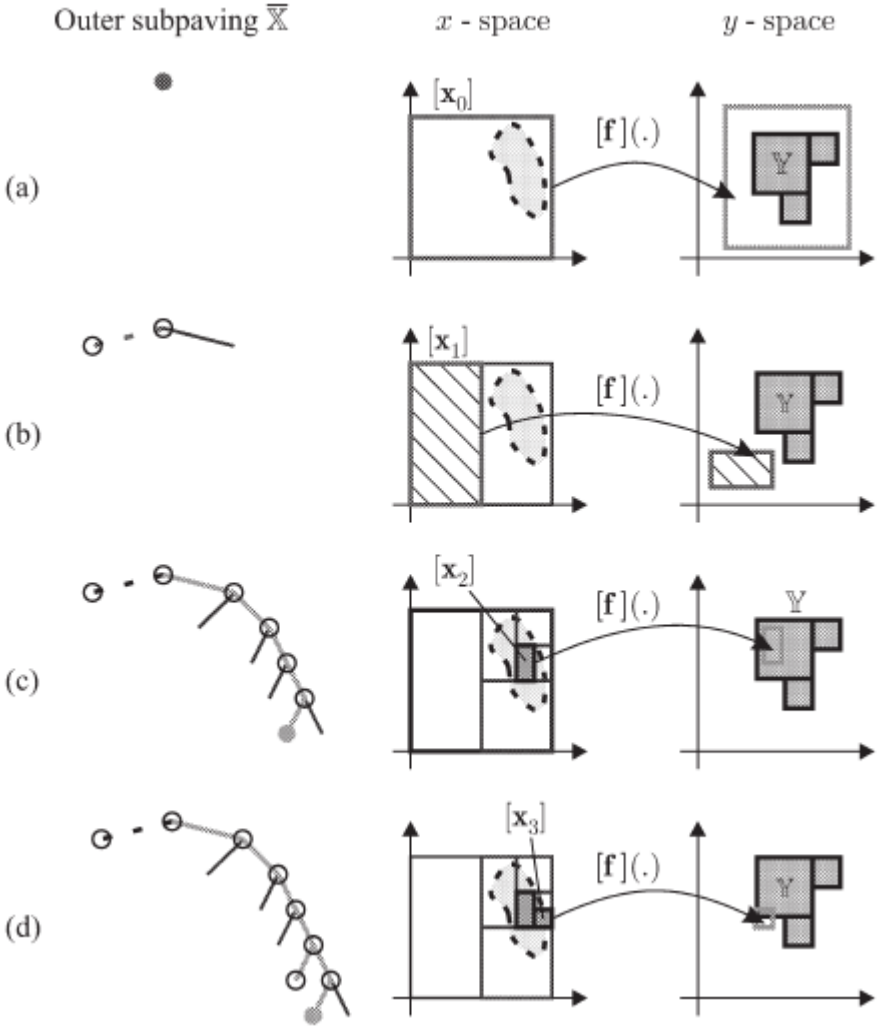
2. Αν η $[f]([x])$ έχει κενή τομή με το Y, τότε το $[x]$ δεν ανήκει στο X και μπορεί να αφαιρεθεί από το δέντρο λύσεων.

3. Αν η $[f]([x])$ ανήκει εξ'ολοκλήρου στο Y, τότε το $[x]$ ανοίκει στο υποδιάστημα λύσεων X, και αποθηκεύεται και στο $\underline{\mathrm{X}}$ αλλά και στο $\overline{Y}$.

4. Αν το κουτί θεωρείται μη-καθορισμένο, και το πλάτος του είναι μικρότερο από το ϵ, τότε θεωρείται αρκετά μικρό για να αποθηκευτεί στην εξωτερική προσέγγιση του X, $\overline{X}$.



Σχήμα 2.4: Όλες οι περιπτώσεις ενός κουτιού στο πεδίο ορισμού και στο πεδίο λύσεων.



Σχήμα 2.5: Μία πιο αναλυτική περιγραφή στις τέσσερις περιπτώσεις που μπορεί να αντιμετωπίσει ο SIVIA. [12]

Algorithm 2 SIVIA(($[f]$, $[x_0]$, $[y]$, ϵ)) //without recursion

Require:

f, ▷ Model function
$[X_0] \in I^n$, ▷ Search space
$y \in Y$, ▷ Image set
ϵ ▷ Elimination criterion

Ensure:

L_{in} ▷ $[f]([x \in L_{in}]) \subseteq [y]$.
L_{out} ▷ $[f]([x \in L_{out}]) \cap [y] = \emptyset$
L_{border} ▷ $\omega([x \in L_{border}]) \leq \epsilon$

$L_{work} \leftarrow [X_0]$ ▷ Working list
while $L_{work} \neq \emptyset$ **do** ▷ Termination criterion
    pop $[x]$ from L_{work} ▷ Selection
    if $[f]([x]) \in [y]$ **then** ▷ Bounding rule
        add $[x] \rightarrow L_{in}$
        continue;
    else if $y < [f]([x])$ or $y > [f]([x])$ **then**
        add $[x] \rightarrow L_{out}$ ▷ Elimination
        continue;
    else
        Bisect $[x] = \{[x_1], [x_2]\}$ ▷ Division
    end if
    if $\omega([x_1]) < \epsilon$ **then** ▷ Elimination
        add $[x_1] \rightarrow L_{border}$
    else
        push $[x_1] \rightarrow L_{work}$
    end if
    if $\omega([x_2]) < \epsilon$ **then** ▷ Elimination
        add $[x_2] \rightarrow L_{border}$
    else
        push $[x_2] \rightarrow L_{work}$
    end if
    return L_{in}, L_{out}, L_{border}
end while

Κεφάλαιο 3

Παράλληλοι αλγόριθμοι και η εφαρμογή τους στο πρόβλημα της αντιστροφής συνόλου

Στόχος αυτού του κεφαλαίου είναι η περιγραφή μερικών ήδη σχεδιασμένων παράλληλων B&B αλγορίθμων. Οι παράλληλοι Branch-and-Bound αλγόριθμοι εκτελούν ανώμαλη και απρόβλεπτη αναζήτηση [7] και ορισμένες φορές θα δούμε πως είναι πιο χρονοβόροι από αντίστοιχους σειριακούς.

3.1 Παραλληλοποίηση B&B Αλγορίθμων και η κατάληξη στον Parallel AMIGO

Στην αρχή, ένας B&B αλγόριθμος, σε σχέση με τη παραλληλοποίησή του, φαίνεται σαν μία αρκετά εύκολη διαδικασία καθώς, κάθε ένα από τα υποπροβλήματα που παράγονται, μπορούν να επιλυθούν ανεξάρτητα από διαφορετικές υπολογιστικές μονάδες. Όμως, περαιτέρω ανάλυση μας δείχνει πως οι B&B αλγόριθμοι πρέπει να κάνουν πιο βέλτιστη αναζήτηση του χώρου λύσεων. Τα υποπροβλήματα παράγονται και απορρίπτονται με απρόβλεπτο και δυναμικό τρόπο. Μερικές φορές ίσως χρειάζεται να ταξινομούνται σε σειρά προτεραιότητας. Ο Parallel AMIGO είναι ένας δοκιμασμένος παράλληλος αλγόριθμος που έχει χρησιμοποιηθεί σε παρόμοια προβλήματα βελτιστοποίησης.[5]

Οι δύο στόχοι του **Parallel AMIGO** είναι οι εξής:

- Να είναι όλες οι υπολογιστικές μονάδες απασχολημένες. Αυτό μπορεί να αποδειχθεί δύσκολο, καθώς τα υποπροβλήματα που βρίσκονται σε αναμο-

νή ποικίλουν σε απαιτούμενο χρόνο επεξεργασίας και εξαρτώνται από το ίδιο το πρόβλημα. Είναι δύσκολο να καθορίσουμε από την αρχή τον μέγιστο αριθμό των μονάδων επεξεργασίας που θα επιλύσουν το κάθε πρόβλημα, ώστε να αποφύγουμε το να βρίσκονται σε αναμονή. Επίσης, ανάλογα με το μοντέλο παραλληλισμού, ένας δυναμικός σταθεροποιητής φόρτου (**Dynamic Load Balancing**) ίσως χρειαστεί για την αποφυγή αδρανών επεξεργαστών, ενώ υπάρχει αρκετή δουλειά για όλους.

- Να συμβαίνει χρήσιμη επεξεργασία. Η επεξεργασία δηλαδή υποπροβλημάτων που οδηγούν σε λύση. Επίσης, καλό είναι να μην δέχονται επεξεργασία υποπροβλήματα που έτσι κι αλλιώς δεν θα επεξεργάζονταν σε έναν διαδοχικό(**sequential**) αλγόριθμο.

3.1.1 Local vs Global-PAMIGO

Global-PAMIGO

Ο Global-PAMIGO μπορεί να κατηγοριοποιηθέι ως ένα (ASP - Asynchronous Single Pool) [5] [4]. Η περιγραφή του έχει ως εξής:

- Η βέλτιστη λύση ορίζεται ως καθολική (global) μεταβλητή.
- Ορίζουμε 2 δομές δεδομένων L και Q οι οποίες είναι κι αυτές globals. Για να αποκτήσουν τα νήματα πρόσβαση σε αυτές θα χρειαστεί αμοιβαίος αποκλεισμός.
- Ο αριθμός των νημάτων (threads) θα είναι ίσος με έναν αριθμό p, ο οποίος καθορίζει τις (λογικές) υπολογιστικές μονάδες που θα χρησμοποιηθούν για να επιλύσουν το πρόβλημα.
- Κάθε νήμα επεξεργάζεται ένα υποπρόβλημα και αποθηκεύει το υποπρόβλημα που ενδέχεται να παραχθεί στο σύνολο L ή το Q. Τα παραγόμενα υποπροβλήματα δεν μπορούν να είναι πάνω από δύο.
- Ένα νήμα που θέλει να λάβει ένα πρόβλημα από το $L = \{\}$, κλειδώνει μέσω αμοιβαίου αποκλεισμού.
- Αν ένα νήμα που εκχωρεί δύο υποπροβλήματα στο L εντοπίσει πως ένα άλλο νήμα βρίσκεται σε κλειδωμένη κατάσταση, το ξεκλειδώνει.

- Όταν ένα νήμα προσπαθήσει να λάβει ένα υποπρόβλημα από το $L = \{\}$ και εντοπίσει πως όλα τα υπόλοιπα νήματα είναι κλειδωμένα, τα ξεκλειδώνει και απαιτείται ο τερματισμός τους.

Τα νήματα του Global-PAMIGO αποκτούν το επόμενο υποπρόβλημα από τη καθολική δομή δεδομένων L. Επίσης, ο συγκεκριμένος παράλληλος αλγόριθμος διανύει το δέντρο αναζήτησης με περίπου την ίδια σειρά όπως και μία διαδοχική-μονοπύρηνη υλοποίηση. Έτσι, όλα τα νήματα λειτουργούν στα υποπροβλήματα με τη μεγαλύτερη προοπτική λύσης.[5]. Το μεγαλύτερο αρνητικό χαρακτηριστικό του συγκεκριμένου μοντέλου, είναι ο ανταγωνισμός των νημάτων στην προσπέλαση της μνήμης, όταν η μεταβλητή $L(Q)$ ενημερώνεται, επειδή δύο νήματα δεν γίνεται να το επεξεργαστούν ταυτόχρονα με ασφάλεια.

Local-PAMIGO

Ο Local-PAMIGO μπορεί να κατηγοριοποιηθεί ως ένα (AMP- Asynchronous Multiple Pool).

Σε αντίθεση με τον Global-PAMIGO, ο συγκεκριμένος αλγόριθμος διαχειρίζεται τοπικές δομές δεδομένων L και Q. Τα χαρακτηριστικά του είναι τα εξής:

- Η βέλτιστη λύση ορίζεται ως global μεταβλητή όπως και στον προηγούμενο αλγόριθμο.

- Ορίζεται μία παράμετρος p η οποία συμβολίζει τον άριθμο των μονάδων επεξεργασίας που θα επιλύσουν το πρόβλημα.

- Ορίζεται μία παράμετρος NTh η οποία δείχνει τον αριθμό των νημάτων που ήδη εκτελούνται. Ο αλγόριθμος προσπαθεί να διατηρήσει σε ισχύ την ισότητα $NTh = p$.

- Ορίζεται ένα καθολικό διάνυσμα VPU (Virtual Process Units) με στοιχεία $VPU[i], i = 1, ..., p$, στο οποίο αποθηκεύεται η ταυτότητα του νήματος που έχει οριστεί στη θέση i ($VPU[i].idthread$) καθώς και την λύση ($VPU[i].Q$). Κάθε νήμα i αποκτά τη δική του τοπική δομή δεδομένων L_i κατά τη δημιουργία του.

- Η εκτέλεση του αλγορίθμου ξεκινά με ένα νήμα που ανατίθεται στο $VPU[1], L_1 = \{S\}, VPU[1].Q = \{\}$ και $NTh = 1$.

- Ένα νήμα που ανατίθεται σε ένα $VPU[i]$ με $L_i =$ θέτει το $NTh = NTh - 1$, ορίζει τη κατάσταση της θέσης i ως ελεύθερη $VPU[i].idthread = -1$ και τερματίζει την εκτέλεση.
- Η συνθήκη $NTh < p$ ελέγχεται από όλα τα νήματα σε κάθε επανάληψη. Όταν ένα νήμα, για παράδειγμα $VPU[i].idthread$, εντοπίσει πως $NTh < p$ και πως υπάρχουν περισσότερα από δύο υποπροβλήματα στην αναμονή μέσα στο L_i, τότε θα εντοπίσει για μία ελεύθερη θέση στο VPU (για παράδειγμα $VPU[j]$) και μεταφέρει το δεύτερο, τέταρτο και ούτω καθεξής υποπροβλήματα από το L_i στο L_j. Τότε, κατασκευάζει ένα νέο νήμα, του αναθέτει στο $VPU[j]$ και θέτει $NTh = NTh + 1$
- Ο συνολικός αριθμός των κατασκευασμένων νημάτων ποικίλει από εκτέλεση σε εκτέλεση και εξαρτάται από το νήμα που θα εντοπίσει πρώτο τη συνθήκη δημιουργίας νέου νήματος ($NTh < p$).
- Ο Αλγόριθμος τερματίζει όταν $NTh = 0$. Το σύνολο λύσεων αποτελείται από το τοπικά αποθηκευμένο σύνολο λύσεων του κάθε νήματος $\{VPU[i].Q, i = 1, ..., p\}$.

Ακολουθώντας αυτή τη μέθοδο, όταν μία εικονική διεργασία (Virtual Process) είναι αδρανής και υπάρχει αρκετή «δουλειά» σε αναμονή, ένα νέο νήμα κατασκευάζεται. Ο χρόνος που μία αδρανής υπολογιστική μονάδα περιμένει για ένα νέο νήμα, δίνεται από τον χρόνο:

- να ελεγχθεί η συνθήκη $NTh < p$
- που χρειάζεται η διαίρεση του χώρου αναζήτησης
- κατασκευής ενός νέου νήματος
- της μεταφοράς του νέου νήματος στην αδρανή υπολογιστική μονάδα.

Ο Local-PAMIGO σχεδιάστηκε για να μειώσει τις συγκρούσεις προσπέλλασης μνήμης, που είναι και η αδυναμία του Global-PAMIGO, εξαιτίας των δομών δεδομένων (L,Q) που είναι κοινά προσβάσιμες από όλους τους επεξεργαστές.

3.2 Το knapsack πρόβλημα της επιλογής κουτιών

Όπως θα δούμε στην συνέχεια, σε μία από τις υλοποιήσεις των παράλληλων αλγορίθμων θα χρειαστεί να αναπτύξουμε μία διαδικασία η οποία θα έχει την ευθύνη της μεταφοράς κουτιών μεταξύ νημάτων. Δηλαδή θα χρειαστεί μία λογική η

οποία θα αποφασίζει ποια κουτιά είναι πιο κατάλληλα για την μεταφορά. Με άλλα λόγια, ποια από τα αποθηκευμένα κουτιά που εκμεταλλεύεται το νήμα αξίζει να επιλέξουμε για μεταφορά, έτσι ώστε ο χρόνος που απαιτείται από τη μέθοδο για την επιλογή τους να είναι ασήμαντος μπροστά στον ολικό χρόνο εκτέλεσης της ρουτίνας.

Στην δεύτερη υλοποίηση που θα ακολουθήσει, σε κάθε θέση του διανύσματος $VPU[i]$, πιο συγκεκριμένα στο L, περιέχεται ένα δυαδικό δέντρο όπου ο κάθε κόμβος του έχει αποθηκευμένη μία λίστα κουτιών ισάξιου φόρτου (σχήμα 4.4, περισσότερες λεπτομέρειες σχετικά με αυτό θα ακολουθήσουν στο επόμενο κεφάλαιο). Αν χρειαστεί η μεταφορά κουτιών μεταξύ θέσεων του VPU θα μπορούσαμε να αδειάσουμε τα κουτιά από τη θέση i και να τα μεταφέρουμε στη θέση j του διανύσματος. Αυτό θα είχε σαν αποτέλεσμα να τερματίσει το νήμα που είχε αναλάβει το $VPU[i].L$ καθώς δεν θα είχε άλλη δουλειά και, σαν να μην έφτανε αυτό, ουσιαστικά θα έχουμε αποτύχει να διαμοιράσουμε τον χώρο επεξεργασίας. Μία άλλη προσέγγιση θα ήταν να ορίσουμε ένα όριο ή καλύτερα ένα άθροισμα που θα αποτελείται από τα πλάτη w των μισών αποθηκευμένων κουτιών $\sum_{i=1}^{N/2} w$. Αν χρησιμοποιήσουμε αυτό το άθροισμα σαν ανώτατο όριο επιλογής κουτιών(ή αλλιώς σαν έναν σάκο), και επειδή μπορούμε να έχουμε περισσότερα από ένα κουτιά σε έναν κόμβο του δυαδικού δέντρου, τότε δεν έχουμε τίποτα άλλο παρά ένα **Bounded Knapsack problem**. Επειδή όμως το knapsack problem είναι ολόκληρο θέμα από μόνο του και επειδή ο διαθέσιμος χρόνος μου ήταν περιορισμένος, στην πράξη, υλοποίησα μία **naive**-απλοϊκή μέθοδο η οποία επιλέγει κουτιά ξεκινώντας από το δεξί φύλλο του δέντρου μέχρι να γεμίσει ο σάκος ή μέχρι να διανυθεί ολόκληρο το δέντρο (σε περίπτωση που τα περισσότερα κουτιά έχουν πολύ μεγάλο πλάτος). Δεν μπορώ να αποδείξω πως είναι η βέλτιστη μέθοδος, συνεπώς αναφερόμαστε σε ένα ανοιχτό πρόβλημα.

Κεφάλαιο 4

Υλοποίηση

Η προτεινόμενη υλοποίηση συνδιάζει τον αλγόριθμο **SIVIA** με τον παράλληλο Branch-and-Bound αλγόριθμο, τον **Local-PAMIGO** διότι σύμφωνα με το [5], οι λιγότερες συγκρούσεις που επιτυγχάνονται επιτρέπουν την κλιμάκωση του αλγορίθμου, δηλαδή μπορεί πιο αποδοτικά να εκτελεστεί σε περισσότερους πυρήνες. Ο στόχος μου ήταν να επιταχύνω τη διαδικασία του Set Inversion, ώστε στο μέλλον, να γίνει εφικτή η ταχύτερη εφαρμογή του σε νευρωνικά δίκτυα.

4.1 Απαραίτητα εργαλεία

4.1.1 Περιβάλλον

Οι δύο υλοποιήσεις που θα ακολουθούν, αναπτύχθηκαν σε γλώσσα **C/C++**, σε λειτουργικό Linux και ειδικότερα, σε Ubuntu 18.04 bionic LTS. Η βιβλιοθήκη που χρησιμοποιήθηκε για τον υπολογισμό των διαστημάτων είναι η C-XSC.

4.1.2 Hardware

Το hardware που χρησιμοποιήθηκε για τον υπολογισμό της επιτάχυνσης που θα δούμε στη συνέχεια ήταν ένα Node του υπερυπολογιστή του εργαστηρίου **Supercomputadoras Algoritmos Laboratorio - SAL** που βρίσκεται στο Πανεπιστήμιο της Αλμερίας (Universidad de Almería) [15]. Πιο συγκεκριμένα, το Node, με όνομα ibbullion, ήταν εξοπλισμένο με:

Hardware			
OS (On the HUB)	CPU	RAM	Storage
Ubuntu 18.04.2 LTS	8 Intel Xeon E7 8860v3 @ 2.2GHz	2.3 TB DDR3	2x 300 GB SAS

Σχήμα 4.1: Ο υπολογιστής που χρησιμοποιήθηκε για τη λήψη των αποτελεσμάτων

4.2 Παραδείγματα

Οι δύο συναρτήσεις που θα κάνω invert μέσω του PAMIGO, είναι η εξίσωση κύκλου με:

$$\begin{aligned} f(x) &= x^2 + y^2 \\ [x]_0 &= [-1.5, 1.5]^2, \\ [y] &= [1, 2] \end{aligned} \tag{4.1}$$

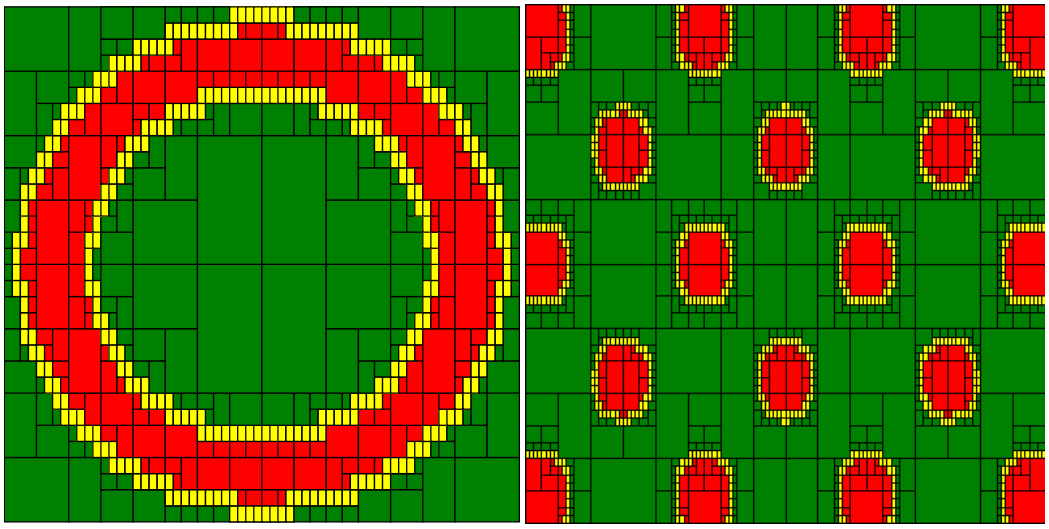
και η εξίσωση του Griewank:

$$\begin{aligned} f(x) = \sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{2} cos(\frac{x_i}{\sqrt{i}}) + 1 \\ [x]_0 = [-10, 10]^2 \\ [y] = [1.5, 3] \end{aligned} \tag{4.2}$$

4.3 Πρώτη Υλοποίηση

Η δομή δεδομένων που χρησιμοποιήθηκε αρχικά είναι μία στοίβα (stack) επειδή ο προγραμματισμός που χρειαζόταν ήταν λιγότερος χάρη στις έτοιμες βιβλιοθήκες της C++ αλλά και για να καταλαμβάνει λιγότερη μνήμη κατά την εκτέλεση [8]. Ο αλγόριθμος, όπως φαίνεται στη συνέχεια, είναι ο ίδιος ο Local-PAMIGO με καθορισμένη δομή δεδομένων και αντί για την εύρεση του μεγίστου ή του ελαχίστου, προσπαθούμε να προσεγγίσουμε την είσοδο X της συνάρτησης f.

- Ορίζουμε το διάνυσμα VPU με θέσεις (slots) p, το οποίο p ορίζεται κατά την εκτέλεση.



(α΄) Circle (β΄) Griewank

Σχήμα 4.2: Το αποτέλεσμα προσέγγισης του κύκλου 4.2α΄ και Griewank 4.2β΄ με $\epsilon = 1e^{-2}$.

- Το τοπικό σύνολο λύσεων που περιέχεται στο VPU, αποτελείται από τρεις λίστες, ας τις πούμε Q_{in}, Q_{out} και Q_{ϵ}.
- Στο Q_{in} αποθηκεύονται οι λύσεις, τα κουτιά δηλαδή που εμπεριέχονται στο Y (Σημείωση: Τα κουτιά είναι υλοποιημένα ως Interval Vectors). Στο Q_{out}, Q_{ϵ} αποθηκεύονται οι λύσεις που απορρίπτονται και τα πολύ μικρά κουτία αντίστοιχα. Ένα κουτί που δεν ανήκει στο Q διχοτομείται. Αν τα κουτιά που προκύπτουν δεν ανήκουν σε κάποιο από τα Q, αποθηκεύονται ξανά στο L.
- Ανατίθεται στο L του $VPU[0]$ το πρώτο κουτί $[X_0]$. Το L είναι υλοποιημένο ως στοίβα. Το $VPU[0]$ ορίζεται ως **busy**.
- Εκτελείται το inclusion check, δηλαδή το $[X_0]$ δίνεται σαν είσοδος σε μία συνάρτηση f.
- Στο τέλος του ελέγχου, αν δεν υπάρχουν κουτιά στο L το νήμα τερματίζει, αλλιώς, ψάχνει για κάποιο νήμα που να μην βρίσκεται σε κατάσταση ”busy”. Αν το βρει, μεταφέρεται από το L_0 στο $L_{OtherSlot}$ ένα κουτί, ορίζεται ως busy, και του ανατίθεται ένα νήμα.

- Κάθε νήμα που ξεκινά, αυξάνει την global μεταβλητή NTh κατά μία μονάδα και την μειώνει όταν καταστρέφεται. Για την αποφυγή συγκρούσεων, το NTh είναι υλοποιημένο ως atomic.
- Ο αλγόριθμος ολοκληρώνει όταν $NTh = 0$.
- Η λύση αποτελείται από τον συνδυασμό όλων των τοπικών Q_{in} και Q_{ϵ}.

4.4 Δεύτερη Υλοποίηση

Οι βασικότερες αλλαγές στον δεύτερο αλγόριθμο αφορούν κυρίως τη δομή δεδομένων L που χρησιμοποιείται για την αποθήκευση των κουτιών. Ήταν απαραίτητος ο σχεδιασμός μίας μεθόδου που θα μας επέτρεπε να επιλέξουμε αρκετά κουτιά, έτσι ώστε αντί να μεταφέρουμε ένα κουτί στο διαθέσιμο slot, να μεταφέρουμε την μισή «δουλειά» του νήματος. Αλλά πριν φτάσουμε σε αυτό το σημείο, έπρεπε με κάποιο τρόπο να αξιολογήσουμε τη δουλειά ενός κουτιού. Εδώ υπεισέρχεται ο υπολογιστής φόρτου εργασίας(**workload estimator**) που παρουσιάζεται στη συνέχεια.

Υπολογιστής Φόρτου Εργασίας

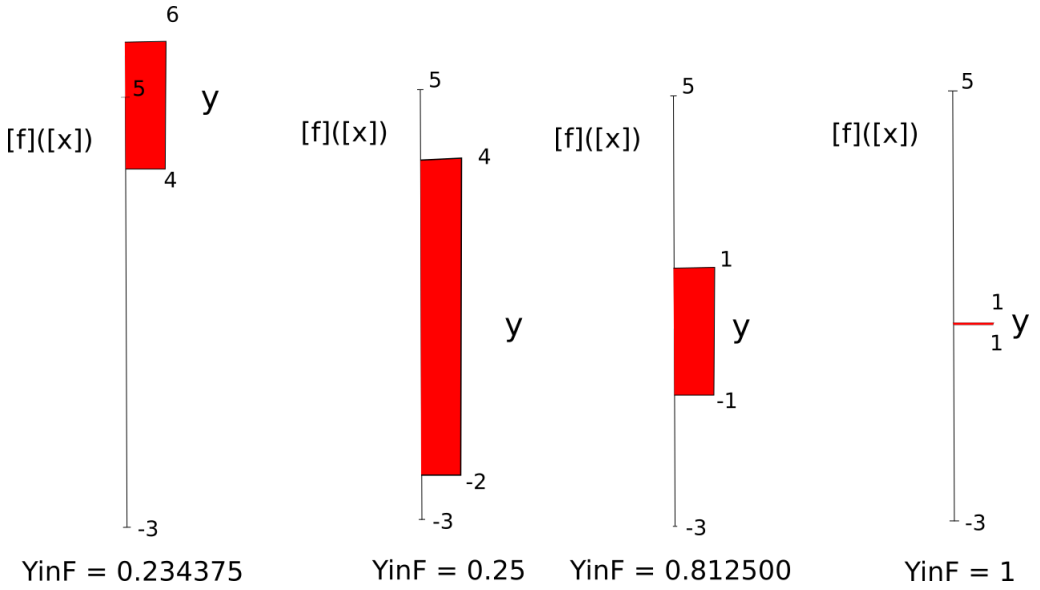
Το workload WL ενός κουτιού, ορίζεται από το το ποσοστό συμπερίληψης του αποτελέσματος $f([x])$, το οποίο προκύπτει από το inclusion test στον SIVIA, από το διάνυσμα λύσεων Y, που μας είναι γνωστό πριν την εκτέλεση του αλγορίθμου αλλά και από το πλάτος ω του κουτιού.

$$\text{WL}(f, [x], [y]) = \text{YinF}(f, [x], [y]) \times \omega([x]) \tag{4.3}$$

Η σχέση $YinF$ μας δίνει τη ποσότητα εγκλεισμού της λύσης(βλ. σχήμα 4.3) Y εντός του διαστήματος $f([x])$, όπως στο 4.4:

$$\text{YinF}(f, [x], [y]) = (1 - \frac{|(\overline{f}([x] - \overline{y}) - (\underline{y} - \underline{f}([x]))|}{\omega([f]([x]))}) \times (1 - \frac{\omega([y])}{\omega([f]([x]))}) \tag{4.4}$$

Όσο το YinF τείνει προς το 1, τόσο περισσότερη δουλειά χρειάζεται το κουτί μέχρι να ταξινομηθεί στο Q.



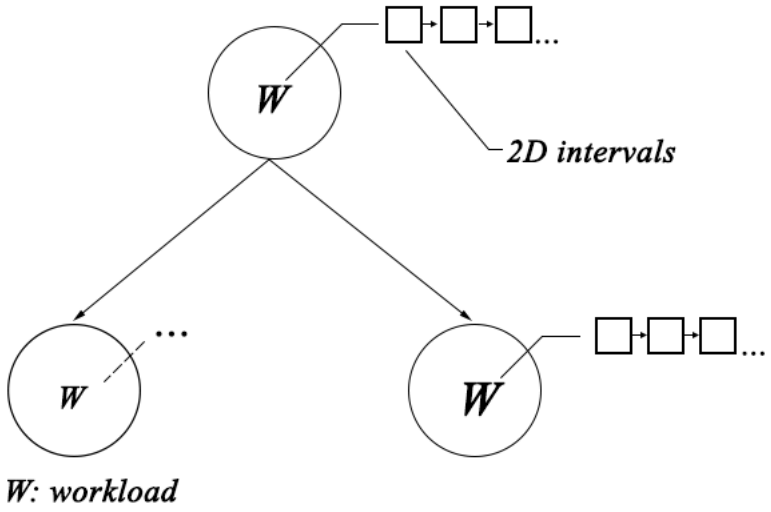
Σχήμα 4.3: Η συμπεριφορά της συνάρτησης YinF.

4.4.1 Ο αλγόριθμος

Ορίζουμε τα ακόλουθα:

- Ένα διάνυσμα VPU με θέσεις (slots) p, το οποίο p ορίζεται κατά την εκτέλεση, όπως ορίζεται από το PAMIGO.
- Το τοπικό σύνολο λύσεων που περιέχεται στο VPU, αποτελείται από τρεις λίστες, έστω Q_{in}, Q_{out} και Q_{ϵ} όπως και στον προηγούμενο αλγόριθμο.
- Στο Q_{in} αποθηκεύονται οι λύσεις, τα κουτιά δηλαδή που εμπεριέχονται στο Y. Στο Q_{out}, Q_{ϵ} αποθηκεύονται οι λύσεις που απορρίπτονται και τα πολύ μικρά κουτία αντίστοιχα.Ένα κουτί που δεν ανήκει στο Q, διχοτομείται. Αν τα κουτιά που προκύπτουν δεν είναι έτοιμα για αποθήκευση στο Q, δηλαδή ούτε έχουν λύση αλλά ούτε είναι όσο μικρά όσο ϵ, τότε υπολογίζεται το WL τους σύμφωνα με την σχέση 4.3 και αποθηκεύονται στο L.
- Στο L του $VPU[0]$ ανατίθεται το πρώτο κουτί $[X_0]$. Το L είναι υλοποιημένο ως ένα AVL δυαδικό δέντρο. Το $VPU[0]$ ορίζεται ως **busy**.
- Εκτελείται το inclusion check, δηλαδή το $[X_0]$ δίνεται σαν είσοδος στην συνάρτηση f.

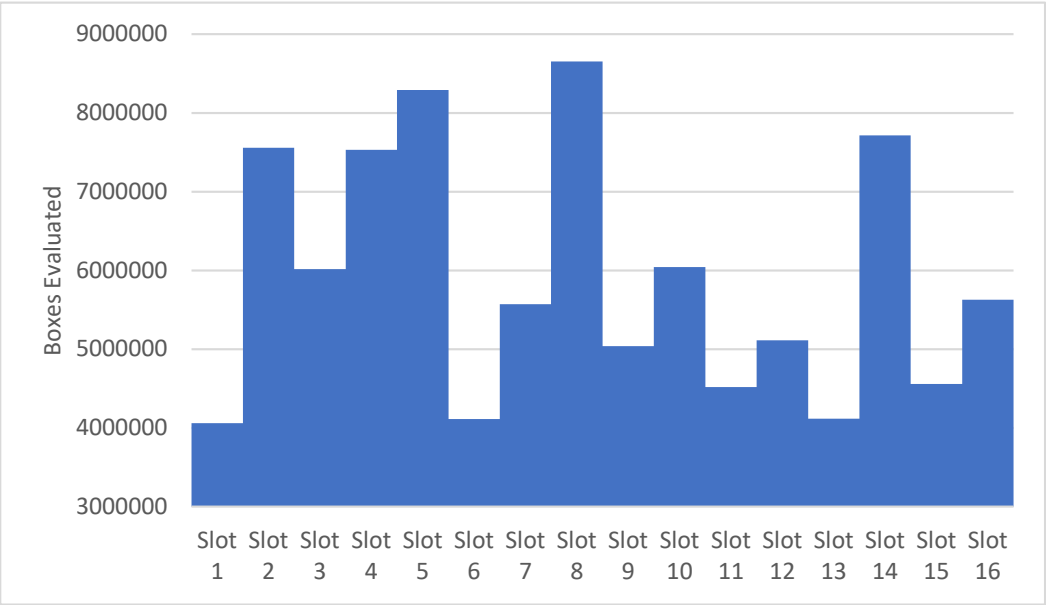
- Στο τέλος του ελέγχου, αν δεν υπάρχουν κουτιά στο L το νήμα τερματίζει, αλλιώς, ψάχνει για κάποιο νήμα που να μην βρίσκεται σε κατάσταση **busy**. Αν το βρει, μεταφέρεται από το L_0 στο $L_{OtherSlot}$ ένα κουτί, ορίζεται ως busy, και του ανατίθεται ένα νήμα.
- Κάθε νήμα που ξεκινά, αυξάνει την global μεταβλητή NTh κατά μία μονάδα και την μειώνει όταν καταστρέφεται. Για την αποφυγή συγκρούσεων, το NTh είναι υλοποιημένο ως atomic.
- Ο αλγόριθμος ολοκληρώνει όταν $NTh = 0$.
- Η λύση αποτελείται από τον συνδυασμό όλων των τοπικών Q_{in} και Q_{ϵ}.



Σχήμα 4.4: Δομή δυαδικού δέντρου. Κάθε κόμβος χρησιμοποιεί ως κλειδί τον υπολογισμένο φόρτο εργασίας του w και σε κάθε κόμβο βρίσκεται αποθηκευμένη μία λίστα με κουτιά ίσου φόρτου.

4.5 Αποτελέσματα

Όπως φαίνεταιε στα σχήματα 4.5, 4.6, 4.7 και 4.8, στην δεύτερη υλοποίηση, όπου χρησιμοποιείται το δυαδικό δέντρο ως L, ο διαμοιρασμός των κουτιών, ή

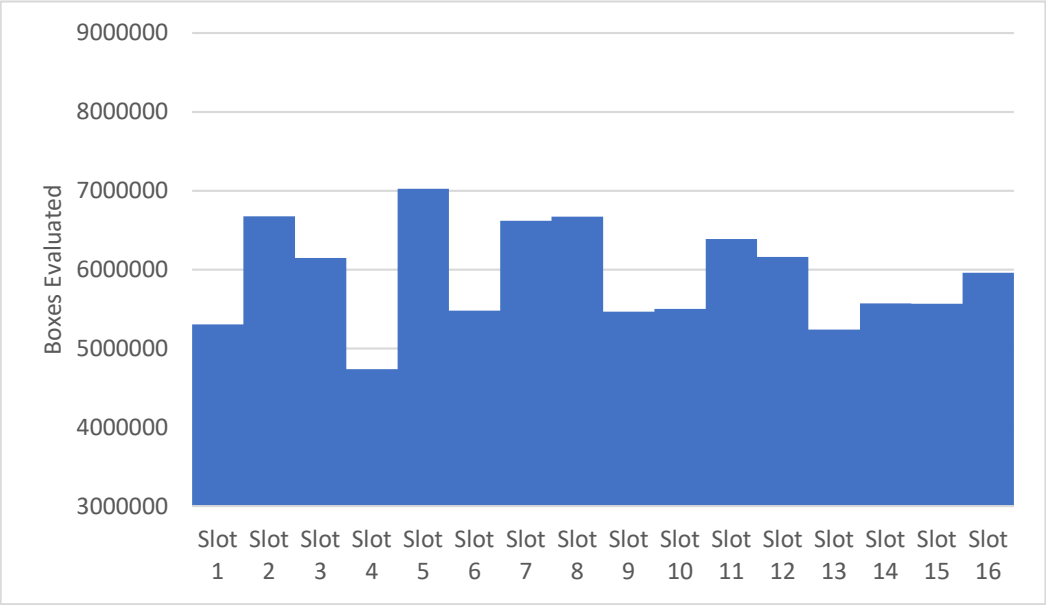


Σχήμα 4.5: **Πρώτη Υλοποίηση** - Ο αριθμός τον κουτιών που εκτιμήθηκαν per-slot για το παράδειγμα του κύκλου για $p = 16$.

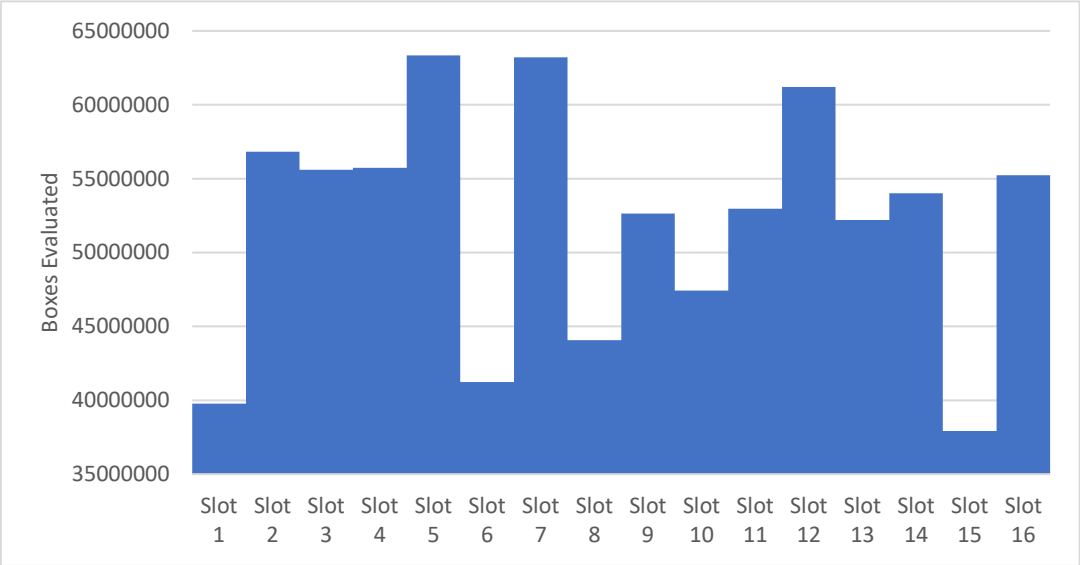
καλύτερα του φόρτου εργασίας, είναι πιο ίσος. Είναι ακόμα πιο ξεκάθαρο στο παράδειγμα του Griewank (Σχ. 4.8). Η επιτάχυνση δίνεται από τη σχέση:

$$S = \frac{t_1}{t_2} \tag{4.5}$$

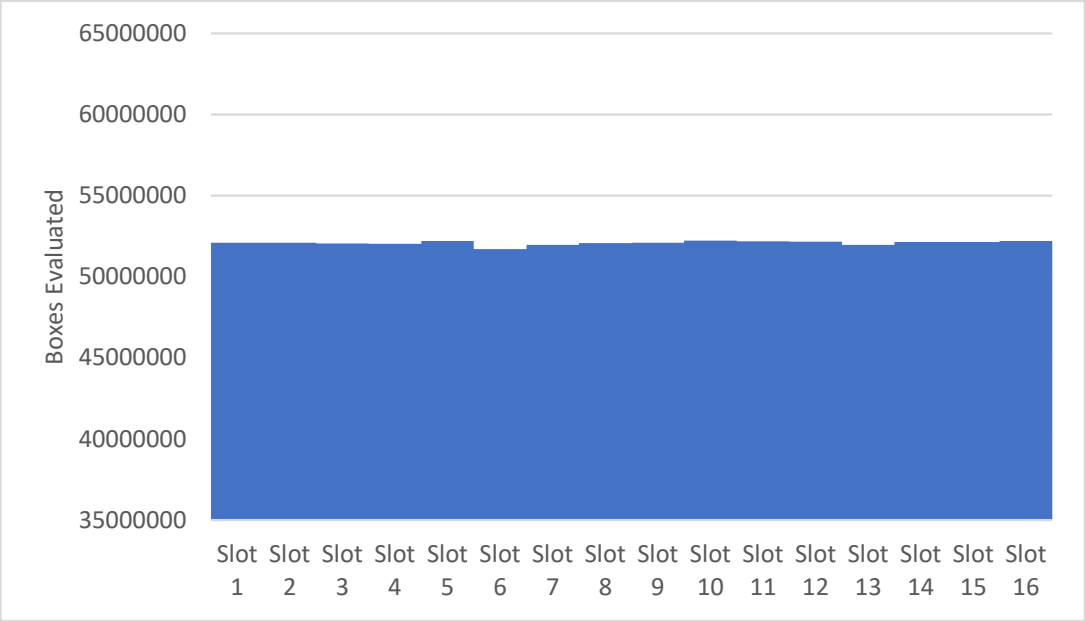
Όπου t_1 είναι η χρόνος εκτέλεσης του αλγορίθμου σε έναν πυρήνα και t_2 είναι ο χρόνος που έκανε για να ολοκληρωθεί ο αλγόριθμος για p νήματα. Στο Σχήμα 4.10, για την υπολογισμό της "πραγματικής" επιτάχυνσης ως t_1 χρησιμοποιήθηκε ο ίδιος αλγόριθμος χωρίς την κατασκευή νημάτων, με την πιο γρήγορη σε υπολογισμούς δομή δεδομένων, την στοίβα, λόγω της άμεσης εισαγωγής και εξαγωγής κουτιών από και προς σε αυτή.



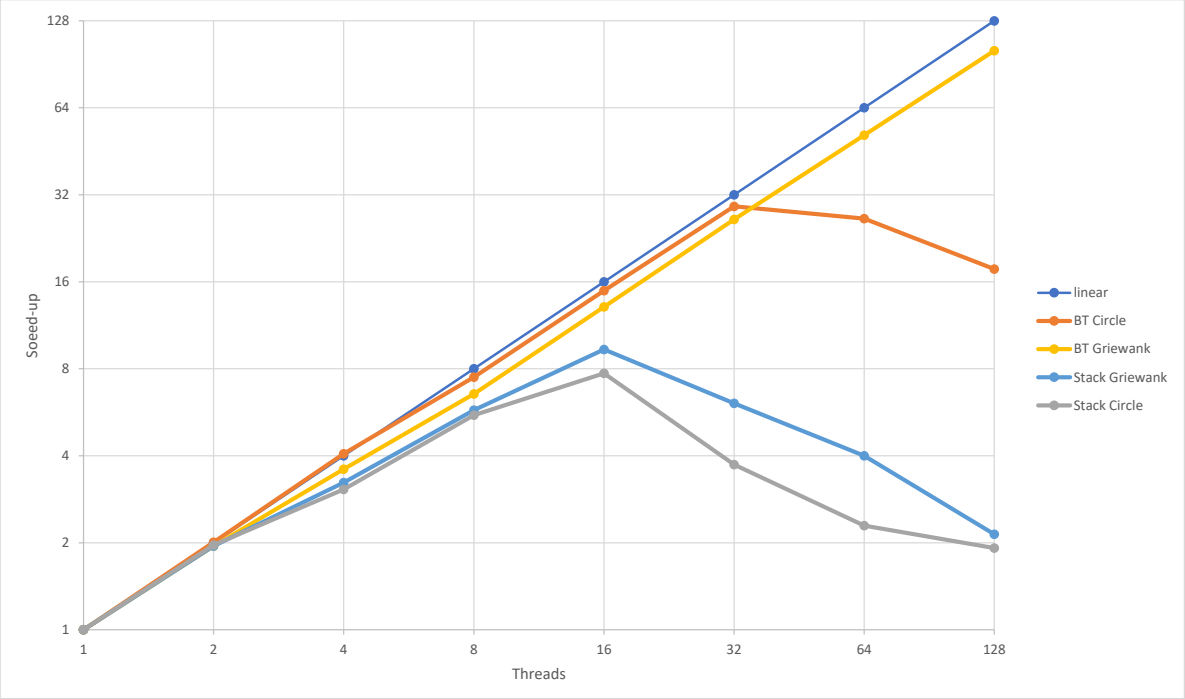
Σχήμα 4.6: **Δεύτερη Υλοποίηση** - Ο αριθμός τον κουτιών που εκτιμήθηκαν per-slot για το παράδειγμα του κύκλου για $p = 16$.



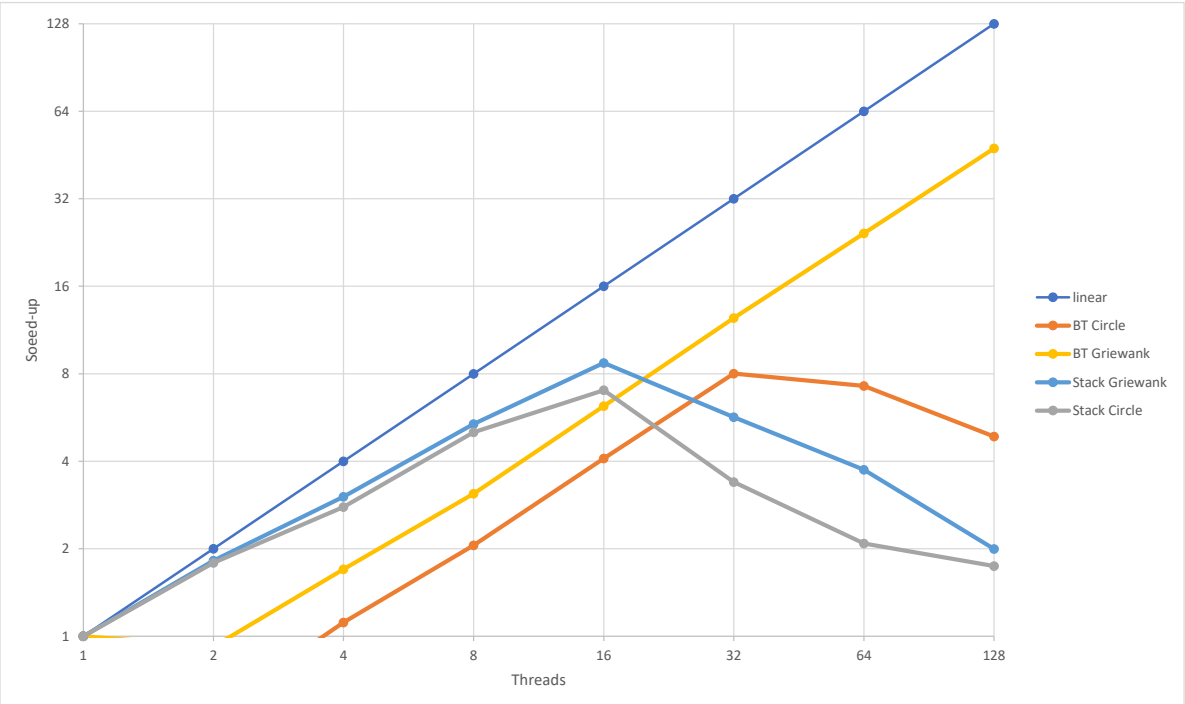
Σχήμα 4.7: **Πρώτη Υλοποίηση** - Ο αριθμός τον κουτιών που εκτιμήθηκαν per-slot για το παράδειγμα του Griewank για $p = 16$.



Σχήμα 4.8: **Δεύτερη Υλοποίηση** - Ο αριθμός τον κουτιών που εκτιμήθηκαν per-slot για το παράδειγμα του Griewank για $p = 16$.



Σχήμα 4.9: Η παρατηρημένη επιτάχυνση στην εκτέλεση των υλοποιήσεων για όλα τα παραδείγματα.



Σχήμα 4.10: Σύγκριση με τον sequential χρόνο εκτέλεσης του αλγορίθμου. Η πραγματική παρατηρημένη επιτάχυνση στην εκτέλεση των υλοποιήσεων για όλα τα παραδείγματα.

Κεφάλαιο 5

Συμπεράσματα

Όπως είδαμε στα παραπάνω σχήματα, μέχρι τα 16 ή τα 32 νήματα, έχουμε βελτίωση στους χρόνους εκτέλεσης σε σχεδόν όλα τα παραδείγματα. Ειδικά στην περίπτωση της δεύτερης υλοποίησης με τον αλγόριθμο του Griewank, παρατηρήθηκε ένα Speedup $S = 48$. Αυτό σημαίνει πως αν ο sequential αλγόριθμος χρειαζόταν 30 ημέρες για να ολοκληρωθεί, τώρα θα χρειαστούν περίπου 15 ώρες. Όσο για το παράδειγμα του κύκλου, ενδέχεται να μην παρατηρείται μεγάλο Speedup, επειδή το δέντρο, που παράγεται από τον Branch & Bound αλγόριθμο, δεν είναι αρκετά μεγάλο. Με λίγα λόγια, ενδέχεται να μην υπάρχει αρκετή δουλειά στο L για να κερδίσει χρόνο από τα πολλαπλά νήματα. Όχι απλά δεν προσφέρει σε επιτάχυνση, αλλά οι πράξεις που αφορούν την μεταφορά φόρτου σε άλλα νήματα, καθώς και οι εντολές που χρειάζονται για την εξαγωγή και εξαγωγή κουτιών από την δομή δεδομένων, επιβαρύνουν την εκτέλεση. Με λίγα λόγια, η μείωση του χρόνου που επιτεύχθηκε, μπορεί να οδηγήσει σε πιο εύστοχες εκτιμήσεις του χώρου εισόδου . Μπορεί η επιτάχυνση να μην είναι γραμμική, αλλά ο χρόνος που μειώθηκε είναι σημαντικός.

5.1 Ανοιχτά προβλήματα

Ενώ παρατηρήθηκε βελτίωση, υπάρχουν διάφοροι τομείς που θα μπορούσαν, υποθετικά, να επεκταθούν οι συγκεκριμένες υλοποιήσεις.

5.1.1 Βελτιστοποίηση του αλγορίθμου

Ο αλγόριθμος, ενώ έχει έναν στοιβαρό σχεδιασμό, βασισμένο σε προηγούμενες εργασίες [5],[13], στην υλοποίηση του όμως υπάρχουν σημεία που θα μπο-

ρούσαν να γραφούν καλύτερα. Καλύτερη χρήση δεικτών, περισσότερη έρευνα στις δομές δεδομένων που χρησιμοποιούνται (όπως red-black trees αντί για παραδοσιακά avl trees) κ.α. Τι μπορούν να προσφέρουν τέτοιες μετατροπές στον αλγόριθμο;

5.1.2 Υποστήριξη περισσότερων διαστάσεων

Αυτό αφορά κυρίως το κομμάτι της εκτίμησης των κουτιών μέσω του SIVIA. Μία επέκταση των υλοποιήσεων θα ήταν η υποστήριξη πολυδιάστατων κουτιών, πολυδιάστατων χώρων λύσεων Y, η υποστήριξη πολλαπλών συναρτήσεων χωρίς να είναι hard-coded κ.λ.π

5.1.3 Εφαρμογή σε νευρωνικά δίκτυα

Αν ένα νευρωνικό δίκτυο το ορίσουμε σαν μία διανυσματική συνάρτηση f, μπορούμε να χρησιμοποιήσουμε τον SIVIA για να προσεγγίσουμε την γενίκευσή του, όπως στο [16]. Ο PAMIGO με τον SIVIA όμως επεκτείνει τις δυνατότητες επιτρέποντας καλύτερες προσεγγίσεις λόγω του μεγαλύτερου ϵ.

Υποθετικό πρόβλημα σε ένα εκπαιδευμένο perceptron

Έστω διανυσματική συνάρτηση $f : \mathbb{R}^2 \to \mathbb{R}^3$ με:

$$f(x_1, x_2) = \begin{cases} g(f_1(x_1, x_2)) \\ g(f_2(x_1, x_2)) \\ g(f_3(x_1, x_2)) \end{cases}$$

Όπου:

$$g(f_i(x_1, x_2)) = \frac{1}{1 + e^{-f_i(x_1, x_2)}} i = 1, 2, 3$$

και:
$f_1(x_1, x_2) = 0.5655 * x_1 + 13.1506 * x_2 - 0.7661$
$f_2(x_1, x_2) = 10.2437 * x_1 - 11.3511 * x_2 - 11.2658$
$f_3(x_1, x_2) - 10.1487 * x_1 - 6.1276 * x_2 - 6.1399$
Το πρόβλημα: Ο ορισμός της περιοχής του $[-1, 1]$x$[-1, 1]$ όπου
$x_1 \in [-1, 1], x_2 \in [1, 1]$ έτσι ώστε $f(x_1, x_2) \in [0.8, 1]$x$[0, 0.2]$x$[0, 0.2]$.

5.1.4 Χρήση MPI

Αν μπορούμε να επεκτείνουμε περισσότερο τον PAMIGO, τότε αυτό θα γινόταν μόνο μέσω MPI. Χάρη σε αυτό, θεωρητικά, ο αλγόριθμος μπορεί να εκμεταλλευτεί περισσότερους πυρήνες που ανήκουν σε περισσότερους από ένα κόμβους υπολογιστικών συστημάτων. Αυτό προϋποθέτει επανασχεδιασμό του αλγορίθμου σε κάποια σημεία, καθώς και την ανάπτυξη μίας μεθόδου (metric), η οποία θα καθορίζει πότε το κουτί (ή κουτιά) θα μεταφερθούν σε άλλο κόμβο αντί για άλλο νήμα. Το ανοιχτό πρόβλημα είναι αν αυτό θα προσφέρει όντως μεγαλύτερη επιτάχυνση.

5.1.5 Μελέτη του knapsack προβλήματος επιλογής κουτιών

Όπως προαναφέρθηκε, η τωρινή μέθοδος επιλογής κουτιών είναι υλοποιημένη αρκετά απλοϊκά. Θεωρώ πως ο PAMIGO θα μας έδειχνε ενδιαφέροντα αποτελέσματα σε πολυδιάστατα προβλήματα. Με την σωστή μελέτη και εφαρμογή λύσεων τέτοιων προβλημάτων ίσως κερδίσουμε σε χρόνο εκτέλεσης, ίσως όμως το κάθε νήμα να έχει μεγάλες απαιτήσεις μνήμης και να μην αξίζει η προσπάθεια.

5.1.6 Χρήση GPU με NVIDIA CUDA® API

Ίσως και η πιο ”τραβηγμένη” ιδέα. Είναι εφικτή η μεταφορά του προβλήματος σε ένα περιβάλλον που αφορά την αρχιτεκτονική των GPUs; Θα μπορούσαν να επιτευχθούν μειωμένοι χρόνοι επεξεργασίας; Πόσο επεκτάσιμη μία τέτοια υλοποίηση;

Βιβλιογραφία

[1] Konstantinos Nasiotis, Daniel López, Stavros P. Adam y Leocadio G. Casado. On parallelizing Set Inversion by Interval Analysis, SARTECO, 2019.

[2] K. A. Nasiotis, D. López, S. P. Adam, and L. G. Casado. Set Inversion Via Interval Analysis - A Study on Parallel Processing Implementation, SWIM, 2019.

[3] TÖRN, Aimo; ŽILINSKAS, Antanas. Global optimization. Berlin: Springer-Verlag, 1989.

[4] GENDRON, Bernard; CRAINIC, Teodor Gabriel. Parallel branch-and-branch algorithms: Survey and synthesis. Operations research, 1994, 42.6: 1042-1066.

[5] CASADO, Leocadio G., et al. Branch-and-bound interval global optimization on shared memory multiprocessors. Optimization Methods & Software, 2008, 23.5: 689-701.

[6] MEHLHORN, Kurt; SANDERS, Peter. Algorithms and data structures: The basic toolbox. Springer Science & Business Media, 2008.

[7] SANJUAN-ESTRADA, J. F.; CASADO, Leocadio G.; GARCÍA, Inmaculada. Adaptive parallel interval branch and bound algorithms based on their performance for multicore architectures. The Journal of Supercomputing, 2011, 58.3: 376-384.

[8] HERRERA, Juan FR, et al. On parallel branch and bound frameworks for global optimization. Journal of Global Optimization, 2017, 69.3: 547-560.

[9] ΧΑΤΖΗΣ, Σωτήρης. Δειγματοληψία με χρήση Κανονικών Κατανομών και εφαρμογή στην Καθολική Βελτιστοποίηση, 2008.

[10] KONTOZOUDIS, Theodoros; ΚΟΝΤΟΖΟΎΔΗΣ, Θεόδωρος. Βελτιστοποίηση ελεγκτών ρομποτικών οχημάτων με τη μέθοδο Βελτιστοποίησης Σμήνους Σωματιδίων. 2017.

[11] ΡΙΖΟΥ, Αρσινόη Ειρήνη. Επέκταση του αλγορίθμου" Dog-Leg" με βελτιστοποίηση εντός υποχώρων. 2018.

[12] JAULIN, Luc, et al. Applied Interval Analysis. Springer, London, 2001.

[13] KREINOVICH, Vladik; BERNAT, Andrew. Parallel algorithms for interval computations: an introduction. Interval Computations, 1994, 3: 3-6.

[14] JAULIN, Luc; WALTER, Eric. Set inversion via interval analysis for nonlinear bounded-error estimation. Automatica, 1993, 29.4: 1053-1064.

[15] Supercomputación Algoritmos laboratory of Ual, *https://hpca.ual.es/es/*.

[16] ADAM, Stavros P.; LIKAS, Aristidis C.; VRAHATIS, Michael N. Evaluating generalization through interval-based neural network inversion. Neural Computing and Applications, 2019, 1-20.

[17] SCALES, L. E. Introduction to non-linear optimization. Macmillan International Higher Education, 1985.

[18] CORMEN, Thomas H., et al. Introduction to algorithms. MIT press, 2009.