# Μέθοδοι Διάδοσης Περιορισμών και Αναζήτησης στον Προγραμματισμό με Περιορισμούς

Νικόλαος Ποθητός*

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών,
Τμήμα Πληροφορικής και Τηλεπικοινωνιών,
Πανεπιστημιούπολη, 157 84 Αθήνα
pothitos@di.uoa.gr

**Περίληψη** Τα *προβλήματα ικανοποίησης περιορισμών* (ΠΙΠ) απαντώνται σε πολλούς επιστημονικούς τομείς, αλλά και στην καθημερινότητα, π.χ. κατά την κατάρτιση ενός ωρολογίου προγράμματος. Ο *Προγραμματισμός με Περιορισμούς* είναι μία δημοφιλής σύγχρονη μεθοδολογία Τεχνητής Νοημοσύνης, που στοχεύει στη δηλωτική διατύπωση ενός ΠΙΠ, σε συνδυασμό με την αποτελεσματική επίλυσή του. Στην εργασία αυτή μελετήσαμε βασικά συστατικά του Προγραμματισμού με Περιορισμούς, όπως η απεικόνιση ευρέων πεδίων τιμών των μεταβλητών ενός ΠΙΠ και η διάδοση (επιβολή) των περιορισμών, και προτείναμε καινούργιες τεχνικές, η αποδοτικότητα των οποίων ελέγχθηκε και πειραματικά.

**Λέξεις-κλειδιά:** διάδοση περιορισμών, συνέπεια ακμών, ευέλικτα πεδία

## 1 Εισαγωγή

Ο Προγραμματισμός με Περιορισμούς είναι ένας από τους βασικούς τομείς της Τεχνητής Νοημοσύνης. Άλλα, πόσο άμεση μπορεί να είναι η σχέση ενός είδους *«προγραμματισμού»* με τη *«νοημοσύνη»* που έχει ένας υπολογιστής;

Για να γίνει κατανοητή αυτή η σύνδεση, θα πρέπει να αναφέρουμε έναν από τους βασικούς στόχους του Προγραμματισμού με Περιορισμούς: *να αποσυνδέσει τη φάση διατύπωσης ενός προβλήματος από τον αλγόριθμο επίλυσής του.* Κατ΄ αυτόν τον τρόπο θα μπορεί ένας προγραμματιστής-χρήστης να δηλώνει εύκολα το πρόβλημά του και στη συνέχεια να καλείται ένα σύστημα επίλυσης για να κάνει την υπόλοιπη δουλειά. Όσο πιο «ευφυές» είναι αυτό το σύστημα, τόσο λιγότερη δουλειά θα έχει να κάνει ο χρήστης του [5].

### 1.1 Προβλήματα Ικανοποίησης Περιορισμών

Ο *Προγραμματισμός με Περιορισμούς* (Constraint Programming) εστιάζει στα *Προβλήματα Ικανοποίησης Περιορισμών* (ΠΙΠ – Constraint Satisfaction Problems

* Επιβλέπων Καθηγητής: Παναγιώτης Σταματόπουλος

– CSPs), ο μαθηματικός ορισμός των οποίων δίδεται παρακάτω. Παραδείγματα τέτοιων προβλημάτων απαντώνται σε όλους σχεδόν τους τομείς της Πληροφορικής. Π.χ. στη Θεωρητική Πληροφορική έχουμε το πρόβλημα ικανοποίησης μιας φόρμουλας που περιέχει λογικές (boolean) μεταβλητές [8], καθώς και άλλα NP-πλήρη προβλήματα· τελευταία, γίνεται επίσης πολλή συζήτηση όσον αφορά ΠΠΠ στην Υπολογιστική Βιολογία [1]. Ένα ΠΠΠ ορίζεται μέσα από το εξής τρίπτυχο [14]:

- ***Περιορισμένες μεταβλητές*** ή απλά *μεταβλητές* (variables) που αποτελούν το σύνολο $\mathscr{V} = \{V_1, V_2, \ldots, V_n\}$.
- ***Πεδία τιμών*** (domains) των μεταβλητών που αποτελούν το σύνολο $\mathscr{D} = \{D_1, D_2, \ldots, D_n\}$. Πρακτικά, μία μεταβλητή $V_i$ λέμε ότι «παίρνει τιμές» από το $D_i$, για $1 \leq i \leq n$. Σε αυτή την εργασία θεωρούμε ότι κάθε πεδίο τιμών είναι πεπερασμένο και αποτελείται από διακριτές ακέραιες τιμές.
- ***Περιορισμοί*** (constraints) που επιβάλλονται μεταξύ των μεταβλητών και αποτελούν το σύνολο $\mathscr{C} = \{C_1, C_2, \ldots, C_e\}$, όπου το $C_i$ περιέχει μια σχέση μεταξύ των πεδίων τιμών των μεταβλητών ενός συνόλου $S_i \subseteq \mathscr{V}$. Ορίζουμε $C_i = (S_i, T_i)$, με $T_i \subseteq D_{i_1} \times D_{i_2} \times \cdots \times D_{i_q}$ το σύνολο με τους δυνατούς συνδυασμούς για τις τιμές των μεταβλητών του συνόλου $S_i = \{V_{i_1}, V_{i_2}, \ldots, V_{i_q}\}$.

Π.χ., για τις μεταβλητές $V_1$ και $V_2$ με πεδία τιμών έστω $\{0, 1, 2, 3\}$, ο περιορισμός της ισότητας μπορεί να δηλωθεί σαν $C_1(\{V_1, V_2\}, \{(0,0), (1,1), (2,2), (3,3)\})$. Παρότι αυτός ο συμβολισμός είναι όσο πιο γενικός γίνεται, στην πράξη χρησιμοποιούμε απλές σχέσεις για να περιγράψουμε τα δίκτυα περιορισμών. Στο παραπάνω παράδειγμα, ο περιορισμός μπορεί να γραφεί απλά σαν $V_1 = V_2$.

Όταν το πεδίο τιμών κάθε μεταβλητής γίνει μονοσύνολο, με άλλα λόγια όταν κάθε μεταβλητή ισούται με μία συγκεκριμένη τιμή, τότε έχουμε μία *ανάθεση* (assignment). Όταν μία ανάθεση είναι συνεπής ως προς τους περιορισμούς του προβλήματος, τότε μπορούμε να την ονομάσουμε και *λύση*.

### 1.2 Συνέπεια Ακμών

Έχοντας ορίσει τι είναι ένα ΠΠΠ, το ζητούμενο είναι τώρα να το επιλύσουμε όσο πιο σύντομα γίνεται. Στην κατεύθυνση αυτή μας βοηθά η *διάδοση περιορισμών* (constraint propagation). Ο σκοπός της μεθοδολογίας αυτής είναι η απαλοιφή όσο το δυνατόν περισσότερων αχρείαστων (no-goods) τιμών από τα πεδία των μεταβλητών, μέσω της επιβολής *συνέπειας ακμών* (arc consistency – AC).

Μια ακμή που ενώνει δύο μεταβλητές $(V_i, V_j)$ είναι *συνεπής*, αν για κάθε $x \in D_i$, υπάρχει $y \in D_j$ τέτοιο ώστε το ζεύγος τιμών $(x, y)$ να μην παραβιάζει τον περιορισμό για την ακμή αυτή. Π.χ. έστω $D_1 = \{7, 8, 9\}$ και $D_2 = \{17, 18\}$ με $V_2 = V_1 + 10$. Τότε η $(V_1, V_2)$ είναι ασυνεπής, αφού για $x = 9$ δεν υπάρχει $y \in D_2$ με $y = 9 + 10$. Για να γίνει η ακμή συνεπής, θα πρέπει να φύγει το 9 από το $D_1$.

## 2 Προηγμένες Μέθοδοι Διάδοσης Περιορισμών

Όταν κάνουμε μία ακμή $(V_1, V_2)$ συνεπή αφαιρώντας τιμές από το $D_1$, αυτό μπορεί να έχει σαν αποτέλεσμα το να γίνει ασυνεπής μία άλλη ακμή π.χ. $(V_3, V_1)$. Συνεπώς

μία αφαίρεση τιμής μπορεί να προκαλέσει ένα «ντόμινο» ασυνεπειών, γι' αυτό υπάρχουν εξειδικευμένοι αλγόριθμοι επιβολής συνέπειας ακμών, όπως ο AC-3 [11] και ο πιο πρόσφατος AC-2001 [2], οι οποίοι εισάγουν σε μία ουρά *τις μεταβλητές* τα πεδία των οποίων τροποποιήθηκαν, έτσι ώστε να διορθωθούν όλες οι ασυνέπειες.

### 2.1 Ο Αλγόριθμος AC-5

Ο αλγόριθμος AC-5 [9] που παρουσιάζεται παρακάτω αποτελεί εξέλιξη του AC-3, γιατί στην ουρά του δεν εισάγει μόνο τις μεταβλητές των οποίων τα πεδία τροποποιήθηκαν, αλλά *και τις τιμές* που αφαιρέθηκαν από αυτά:

---

**procedure** AC-5
  $Q \leftarrow \emptyset$
  **for each** $(i, j) \in \text{arcs}(G)$ **do**
    ArcCons$(i, j, \Delta)$
    $Q \leftarrow Q \cup \{((k, i), w) \mid (k, i) \in \text{arcs}(G), k \neq j\}$
    $D_i \leftarrow D_i - \Delta$
  **end for**
  **while** $Q \neq \emptyset$ **do**
    Pick and remove an $((i, j), w)$ out of $Q$
    LocalArcCons$(i, j, w, \Delta)$
    $Q \leftarrow Q \cup \{((k, i), w) \mid (k, i) \in \text{arcs(G)}, k \neq j\}$
    $D_i \leftarrow D_i - \Delta$
  **end while**
**end procedure**

---

Ο AC-5 χαρακτηρίζεται ως *παραμετρικός* εφόσον οι δύο υπορουτίνες ArcCons και LocalArcCons χρειάζεται να οριστούν για κάθε περιορισμό, ή, καλύτερα, για κάθε τύπο περιορισμού από τον χρήστη-προγραμματιστή.

- Η ArcCons$(i, j, \Delta)$ φέρνει σε συνέπεια την ακμή $(i, j)$. Όποιες τιμές αφαιρεθούν από το πεδίο $D_i$, θα πρέπει να μπουν στο σύνολο $\Delta$.
- Η LocalArcCons$(i, j, w, \Delta)$ οφείλει να φέρνει σε συνέπεια την ακμή $(i, j)$ *δεδομένης της αφαίρεσης της τιμής* $w$ *από το πεδίο* $D_j$.

Η συνάρτηση η οποία καλείται τις περισσότερες φορές σε αυτόν τον αλγόριθμο είναι η LocalArcCons. Έχει ληφθεί μέριμνα έτσι ώστε να *πληροφορείται* για τιμές $w$ οι οποίες προκάλεσαν την *πυροδότησή* της και αυτό τη βοηθά στο να έχει μικρότερη πολυπλοκότητα από την ArcCons. Όταν όμως η υπορουτίνα αυτή *δεν* χρησιμοποιεί την πληροφορία $(w)$ που της δόθηκε, αυτό έχει σαν αποτέλεσμα να καλείται πολλές φορές άσκοπα, ενώ θα μπορούσε να κληθεί μία φορά για όλα τα διαφορετικά $w$ που την αφορούν και περιέχονται στην ουρά $Q$.

### 2.2 Η Μηχανή Διάδοσης Περιορισμών AC-5+

Για να περιοριστεί αυτή η σπατάλη, εξελίξαμε τον παραπάνω αλγόριθμο έτσι ώστε εν πολλοίς να προσαρμόζεται καλύτερα στις διάφορες κατηγορίες των αλγορίθμων.

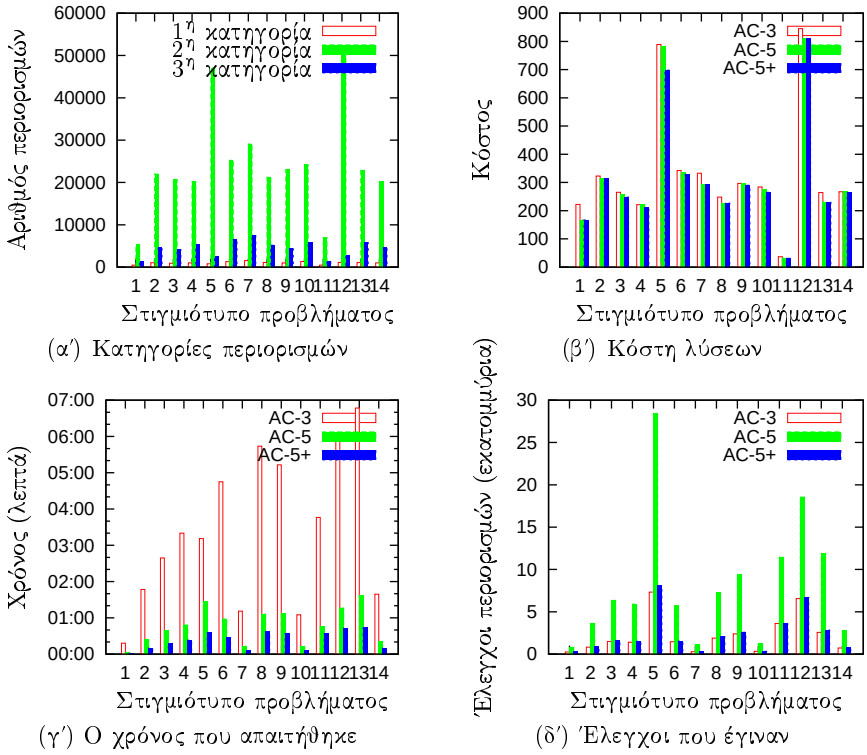
```
 1: procedure AC-5+
 2:     Q ← ∅
 3:     for each c ∈ 𝒞 do
 4:         c.ArcCons(Q)
 5:     end for
 6:     while Q ≠ ∅ do
 7:         Pick and remove a (V, W, (c_b, boundc, timec)) out of Q
 8:         for each c ∈ 𝒞, with V ∈ vars(c) do
 9:             if c.LocalArcCons category is 1st then
10:                 for each (c_f, w) ∈ W, with c ≠ c_f do
11:                     c.LocalArcCons(V, w, Q)
12:                 end for
13:             else if c.LocalArcCons category is 2nd then
14:                 if boundc = true and c ≠ c_b then
15:                     c.LocalArcCons(V, w, Q)          ▷ Το w δεν χρησιμοποιείται.
16:                 end if
17:             else                                     ▷ c.LocalArcCons category is 3rd
18:                 if boundc = true and c.last_check_time < timec then
19:                     c.LocalArcCons(V, w, Q) ▷ Τα V και w δεν χρησιμοποιούνται.
20:                 end if
21:             end if
22:             c.last_check_time ← time
23:         end for
24:     end while
25: end procedure
```

Στον παραπάνω αλγόριθμο έχει αλλάξει καταρχήν ο τύπος των στοιχείων της ουράς. Αντί για την τιμή $w$ υπάρχει τώρα το $W$ το οποίο περιέχει ένα *σύνολο* από τιμές· το σύνολο αυτό αφορά την $1^{\eta}$ κατηγορία LocalArcCons (γραμμή 11). Επίσης υπάρχει η μεταβλητή *boundc* η οποία είναι true αν έχει μεταβληθεί κάποιο *άκρο* του πεδίου τιμών· οι περισσότερες LocalArcCons έχουν φτιαχτεί για να καλούνται μόνο σε αυτή την περίπτωση ($2^{\eta}$ κατηγορία – γραμμή 15).

Η $3^{\eta}$ κατηγορία LocalArcCons (γραμμή 19) έχει να κάνει με τις υπορουτίνες εκείνες που δίνουν το ίδιο αποτέλεσμα, ανεξάρτητα από το όρισμα $V$ που δέχονται. Μέσω της μεταβλητής *timec* είμαστε σε θέση να γνωρίζουμε αν η συγκεκριμένη συνάρτηση έχει ξανακληθεί προηγουμένως (για κάποια άλλη μεταβλητή $V' \neq V$), έτσι ώστε να μην χρειαστεί να την ξανακαλέσουμε.

### 2.3 Πειραματικά Αποτελέσματα

Πέρα από τη θεωρητική μελέτη που έγινε όσον αφορά τη μειωμένη πολυπλοκότητά του, ο AC-5+ ενσωματώθηκε και πρακτικά στο σύστημα Προγραμματισμού με Περιορισμούς Naxos Solver [12]. Ο Naxos είναι μία βιβλιοθήκη για το αντικειμενοστραφές περιβάλλον προγραμματισμού της C++, με τον οποίο επιλύσαμε τα προβλήματα κατάρτισης ωρολογίων προγραμμάτων για εκπαιδευτικά ιδρύματα του αντίστοιχου διεθνούς διαγωνισμού (International Timetabling Competition




(α′) Κατηγορίες περιορισμών (β′) Κόστη λύσεων

(γ′) Ο χρόνος που απαιτήθηκε (δ′) Έλεγχοι που έγιναν

**Σχήμα 1.** Πειραματικά αποτελέσματα στην κατάρτιση ωρολογίων προγραμμάτων

2007) [6] –τα οποία προσφέρουν μεγάλη ποικιλία περιορισμών– σε έναν υπολογιστή με διπύρηνο επεξεργαστή Intel Core στα 2.8 GHz και 1 GB μνήμη.[1]

Χρησιμοποιήθηκε η μέθοδος *αναζήτησης με φραγμένη κατά βάθος ασυμφωνία* (depth-bounded discrepancy search – DDS) [17] η οποία διατίθεται μαζί με άλλες μεθόδους σαν επέκταση του Naxos [19]. Σε κάθε βήμα της αναζήτησης επιβάλλεται συνέπεια ακμών [15] μέσω ενός διαφορετικού κάθε φορά αλγορίθμου AC.

Στο Σχήμα 1(α′) παρουσιάζεται η κατανομή των περιορισμών στις τρεις κατηγορίες για το κάθε στιγμιότυπο του προβλήματος· οι περισσότεροι ανήκουν στη $2^η$ κατηγορία. Στο Σχήμα 1(β′) φαίνεται η υπεροχή του AC-5+, καθώς τα κόστη των λύσεων που βρίσκονται μέσω αυτού είναι τα μικρότερα.

Πριν φτάσουν όμως στις βέλτιστες λύσεις, οι μεθοδολογίες αυτές έχουν βρει κάποιες κοινές λύσεις, σαν ενδιάμεσες. Στα Σχήματα 1(γ′) και 1(δ′) προβάλλονται ο χρόνος που χρειάστηκε και οι έλεγχοι περιορισμών που πραγματοποίησε καθεμία από τις τρεις μεθοδολογίες έτσι ώστε να φτάσει στην καλύτερη *κοινή* λύση. Ο AC-5 είναι γρηγορότερος από τον AC-3, αλλά πραγματοποιεί περισσότερους ελέγχους

[1] Ο διαθέσιμος χρόνος μέσα στον οποίο οφείλουμε να βρούμε λύση έχει οριστεί από τον διαγωνισμό για τον συγκεκριμένο υπολογιστή ίσος με 7 λεπτά και 20 δευτερόλεπτα.

περιορισμών, οι οποίοι όμως είναι ταχύτεροι. Ο AC-5+ μειώνει τους υπεράριθμους αυτούς ελέγχους του AC-5, επιταχύνοντας έτσι την επίλυση.

# 3 Ευέλικτη Διαχείριση Μεγάλων Πεδίων Τιμών [13]

Το κύριο πρόβλημα κατά την επίλυση ενός ΠΙΠ είναι ο εκθετικός *χρόνος* που απαιτείται, στη γενική περίπτωση. Ο *χώρος* που θα χρειαστεί να δεσμεύσουμε στη μνήμη του υπολογιστή έρχεται σε δεύτερη μοίρα, αφού είναι πολυωνυμική συνάρτηση του μεγέθους του μέγιστου πεδίου τιμών (το οποίο συνήθως συμβολίζεται με $d$). Σε αυτή την ενότητα σχεδιάζουμε και υλοποιούμε αλγορίθμους και δομές δεδομένων έτσι ώστε να αναδείξουμε τα οφέλη όταν η πολυπλοκότητα διαχείρισης μνήμης είναι καλύτερη από πολυωνυμική.

## 3.1 Σχετικές Εργασίες

Υπάρχουν εργασίες πάνω στις δομές δεδομένων για την αποδοτική *διάδοση περιορισμών* [7], αλλά από όσο γνωρίζουμε η απεικόνιση των πεδίων τιμών δεν έχει αποτελέσει το *κύριο* θέμα κάποιας δημοσίευσης. Παρόλα αυτά, οι Schulte και Carlsson μέσα σε μία επισκόπηση των συστημάτων Προγραμματισμού με Περιορισμούς [16] παραθέτουν τυπικούς ορισμούς για τις δύο δημοφιλέστερες δομές δεδομένων που αναπαριστούν πεπερασμένα σύνολα ακεραίων:

**Πίνακας Bit.** Χωρίς βλάβη της γενικότητας υποθέτουμε ότι ένα πεδίο τιμών $D$ περιέχει μόνο *θετικές* ακέραιες τιμές. Έστω $a$ ένας *πίνακας από bit* (bit vector). Τότε η τιμή $v$ ανήκει στο $D$, αν και μόνο αν $a[v] = 1$. Παραλλαγές αυτής της δομής υλοποιούνται σε πολλούς επιλυτές Προγραμματισμού με Περιορισμούς [3,4].

**Ακολουθία Διαστημάτων.** Μία άλλη προσέγγιση είναι η απεικόνιση μίας *ακολουθίας διαστημάτων* (range sequence). Τυπικά, το $D$ «αποσυντίθεται» στα στοιχεία του συνόλου $\{[a_1, b_1], \ldots, [a_n, b_n]\}$, έτσι ώστε $\cup_i [a_i, b_i] = D$. Είναι επιθυμητό αυτή η ακολουθία να είναι ταξινομημένη και η μικρότερη δυνατή, δηλαδή $[a_i, b_i] \cap [a_j, b_j] = \emptyset$, $\forall i \neq j$.

Μία πιο απλή δομή από τις δύο προαναφερθείσες, αποθηκεύει μόνο τα *άκρα* του $D$. Π.χ., για το πεδίο τιμών $[1\,..\,100000]$[2] αποθηκεύουμε στη μνήμη δύο μόνο ακεραίους: τον 1 και τον 100000. Προφανώς αυτή είναι μία ατελής αναπαράσταση για τα ασυνεχή πεδία τιμών, όπως το $[1\,..\,3\;5\,..\,9]$. Αυτό σημαίνει ότι η συγκεκριμένη δομή είναι ασύμβατη με τους περισσότερους αλγορίθμους για ΠΙΠ· μόνο συγκεκριμένες μεθοδολογίες μπορούν να την εκμεταλλευτούν [18].

Από την άλλη πλευρά, για την απεικόνιση του παραπάνω πεδίου $[1\,..\,100000]$ ένας πίνακας bit θα δέσμευε 100000 bit στη μνήμη, παρότι το πεδίο θα μπορούσε να αναπαρασταθεί μέσω μίας ακολουθίας διαστημάτων, χρησιμοποιώντας δύο μόλις λέξεις μνήμης.

[2] Με $[a\,..\,b]$ συμβολίζουμε το σύνολο $\{a, a+1, \ldots, b\}$.

### 3.2 Αποδοτική Απεικόνιση Πεδίων Τιμών

Πέρα από την απόπειρα για μείωση της *χωρικής* πολυπλοκότητας, δεν θα πρέπει να αμελήσουμε τη *χρονική* πολυπλοκότητα, ιδίως των παρακάτω δύο λειτουργιών που εκτελούνται πάρα πολλές φορές πάνω στα πεδία τιμών:

1. *Αναζήτηση* αν ένα εύρος τιμών περιλαμβάνεται στο πεδίο.
2. *Διαγραφή* ενός εύρους τιμών από το πεδίο τιμών.

Σημειώνεται ότι η προσθήκη τιμών είναι περιττή· τα πεδία πάντα *μειώνονται* είτε λόγω της διάδοσης περιορισμών, είτε λόγω αναθέσεων τιμών.

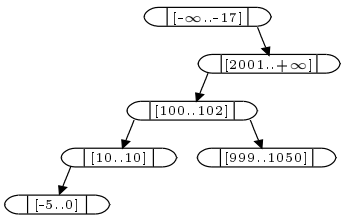
Η αναζήτηση ή η αφαίρεση ενός εύρους από $k$ τιμές παίρνει $O(k)$ βήματα σε έναν *πίνακα bit*. Οι ίδιες λειτουργίες σε μία *ακολουθία διαστημάτων* που έχει υλοποιηθεί σαν *συνδεδεμένη λίστα* [16] απαιτούν $O(\delta)$ βήματα, όπου $\delta$ ο αριθμός των διακριτών διαστημάτων του πεδίου. Ο χώρος που δεσμεύεται είναι κατά πολύ λιγότερος ($O(\delta)$ επίσης) σε σχέση με τον αντίστοιχο του πίνακα bit ($O(d)$). Μία πιο «σοφή» επιλογή θα ήταν να υλοποιήσουμε την ακολουθία διαστημάτων σαν ένα *δυαδικό δένδρο αναζήτησης*, το οποίο έχει μέση πολυπλοκότητα αναζήτησης/διαγραφής $O(\log \delta)$, χωρίς να απαιτείται παραπάνω χώρος.

Ωστόσο η διαγραφή ενός εύρους τιμών από ένα δένδρο είναι περίπλοκη. (Για να γίνει απαιτούνται δύο περίπου διασχίσεις του δένδρου και μία συνένωση δύο υποδένδρων.) Αυτό είναι ανεπιθύμητο, όχι μόνο για τον χρόνο που δαπανάται, αλλά και για τις πολλαπλές τροποποιήσεις που επιφέρει στη δομή. Ο αριθμός των τροποποιήσεων είναι κρίσιμος, αφού καταγράφονται έτσι ώστε να *αναιρεθούν* όταν ένα σύστημα Προγραμματισμού με Περιορισμούς *οπισθοδρομεί*, δηλαδή όταν επαναφέρει τα πεδία τιμών σε μία πρότερη (ή στη αρχική) κατάστασή τους, σε μία προσπάθεια να επανεκκινήσει τη διαδικασία επίλυσης, ψάχνοντας σε άλλα μονοπάτια.

### 3.3 Αναπαράσταση Δένδρου με Κενά Διαστήματα.

Για να γίνουν τα πράγματα πιο απλά και γρήγορα, υλοποιήσαμε ένα δυαδικό δένδρο αναζήτησης στους κόμβους του οποίου «κρεμάσαμε» τα *κενά διαστήματα* του πεδίου τιμών. Ένα σημαντικό πλεονέκτημα αυτής της υλοποίησης είναι ότι η διαγραφή ενός εύρους τιμών γίνεται ταχύτερα, αφού επηρεάζει μόνο έναν κόμβο του δένδρου (δηλαδή είτε εισάγεται, είτε τροποποιείται ένας μόνο κόμβος).

Π.χ., το πεδίο $[9\,..\,17\;44\,..\,101]$ περιγράφεται από τα τρία κενά: $[-\infty\,..\,8]$, $[18\,..\,43]$ και $[102\,..\,+\infty]$. Στο Σχήμα 2 απεικονίζονται τα κενά ενός πεδίου τιμών διατεταγμένα ως ένα δένδρο αναζήτησης.




**Σχήμα 2.** Το δένδρο που περιέχει τα κενά διαστήματα του πεδίου τιμών $[-16\,..\,-6\;1\,..\,9\;11\,..\,99\;103\,..\,998\;1051\,..\,2000]$

Ένα ακόμα πλεονέκτημα αυτής της προσέγγισης είναι ότι οι δύο βασικές λειτουργίες (αναζήτηση/διαγραφή) εκτελούνται μέσω του ίδιου αλγορίθμου.[3] Ο αλγό-

[3] Παρουσιάζεται στο `http://www.di.uoa.gr/~pothitos/setn2010/algo.pdf`

ριθμος αυτός εκτελείται σε λογαριθμικό χρόνο, αφού υλοποιεί τις βασικές λειτουργίες για ένα δυαδικό δένδρο αναζήτησης. Πέραν τούτου, επιχειρεί να κάνει όσο το δυνατόν περισσότερες συγχωνεύσεις κόμβων για να διατηρείται το δένδρο μικρό.

### 3.4 Πειραματικά Αποτελέσματα

Παρότι η παραπάνω υλοποίηση είναι συμβατή με τον τυπικό ορισμό ενός ΠΠΠ και τις συνήθεις μεθοδολογίες που το επιλύουν [15], συνιστάται ιδίως για προβλήματα με μεγάλα μη συνεχή πεδία τιμών, όπως αυτά που συναντάμε στη Βιοπληροφορική.

**Ένα Απλό Πρόβλημα Ακολουθιών DNA.** Το γενετικό υλικό κάθε ανθρωπίνου κυττάρου περιέχει 46 χρωμοσώματα, καθένα από τα οποία δομείται από αλυσίδες 247.2 εκατομμυρίων περίπου νουκλεοτιδίων DNA. Υπάρχουν τέσσερα είδη τέτοιων νουκλεοτιδίων: η αδενίνη (A), η θυμίνη (T), η γουανίνη (G) και η κυτοσίνη (C).

Ας υποθέσουμε ότι επιθυμούμε να «χωρέσουμε» μέσα σε ένα χρωμόσωμα μία ακολουθία από τέσσερις κυτοσίνες $C_1$, $C_2$, $C_3$, $C_4$ και μία ακολουθία από τέσσερις γουανίνες $G_1$, $G_2$, $G_3$, $G_4$ επίσης. Τα $C_i$ και $G_i$ συμβολίζουν τις *θέσεις* των αντίστοιχων νουκλεοτιδίων στην αλυσίδα του DNA· το αρχικό πεδίο τιμών για κάθε θέση είναι το $[1\,..\,247200000]$. Έστω ότι η πρώτη ακολουθία είναι γεωμετρική με $C_i = \lfloor C_{i+1}/99 \rfloor$ και η δεύτερη ακολουθία είναι αριθμητική με $G_{i+1} = G_i + 99$.

**Προβλήματα κατά την Επίλυση.** Αυτό το απλοϊκό ΠΠΠ, με οκτώ μόλις περιορισμένες μεταβλητές, μπορεί να γίνει... δύσκολο αν δεν φροντίσουμε να διαχειριστούμε κατάλληλα τα εκατομμύρια των τιμών που περιέχουν τα πεδία τους.[4]

*Naxos.* Καταρχήν ενσωματώσαμε το δυαδικό δένδρο με τα κενά διαστήματα που περιγράφηκε στην προηγούμενη ενότητα στον επιλυτή Naxos Solver [12] –ο οποίος χρησιμοποιήθηκε και στη § 2.3. Η λύση[5] για το απλό ΠΠΠ που περιγράφτηκε παραπάνω βρέθηκε ακαριαία, χρησιμοποιώντας 3 MB μνήμη.

*$ECL^iPS^e$.* Στο ίδιο μηχάνημα, ωστόσο, χρειάστηκαν να περάσουν τρία δευτερόλεπτα για να βρει το σύστημα Λογικού Προγραμματισμού με Περιορισμούς $ECL^iPS^e$ 5.10[6] [4] την ίδια λύση, δεσμεύοντας 125 MB μνήμη, αφού υλοποιεί μία παραλλαγή του πίνακα bit για να αποθηκεύει τα πεδία τιμών.

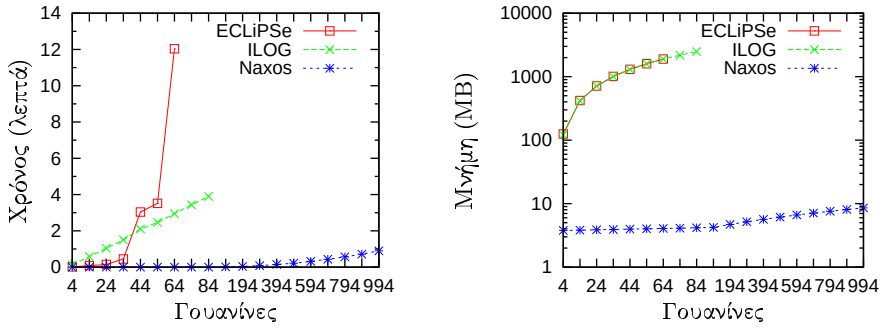
*Ilog.* Ο Ilog Solver 4.4 [10], μία δημοφιλής βιβλιοθήκη Προγραμματισμού με Περιορισμούς σε C++, χρειάζεται τριπλάσιο χρόνο (περίπου δέκα δευτερόλεπτα) σε σχέση με την $ECL^iPS^e$ για να επιλύσει το πρόβλημα, αλλά καταναλώνει σχεδόν την ίδια μνήμη.

---

[4] Τα πειράματα έγιναν σε έναν υπολογιστή Sun Blade ο οποίος διαθέτει έναν επεξεργαστή SPARC στα 1.5 GHz και 1 GB μνήμη. Όλα τα πειραματικά δεδομένα και ο κώδικας για τα προβλήματα Βιοπληροφορικής που επιλύσαμε διατίθενται στη διεύθυνση `http://www.di.uoa.gr/~pothitos/setn2010`

[5] Η πρώτη λύση περιλαμβάνει τις αναθέσεις $C_1 = 1$, $C_2 = 99$, $C_3 = 9801$, $C_4 = 970299$, $G_1 = 2$, $G_2 = 101$, $G_3 = 200$ και $G_4 = 299$.

[6] Χρησιμοποιήσαμε τη βιβλιοθήκη της $ECL^iPS^e$ ονόματι `ic` (Interval Constraints).




(α′) Ο *χρόνος* για να βρεθεί μία λύση (β′) Ο *χώρος* που δεσμεύτηκε

**Σχήμα 3.** Οι πόροι που ζητούν οι επιλυτές ΠΠ όσο το πρόβλημα κλιμακώνεται

**Κλιμάκωση του Προβλήματος.** Ένας απλός τρόπος για να κλιμακώσουμε το πρόβλημα είναι το να προσθέσουμε ακόμα περισσότερες γουανίνες στην αντίστοιχη ακολουθία. Στο Σχήμα 3 υπάρχουν οι γραφικές παραστάσεις με τους πόρους που δαπανά καθένα από τα τρία συστήματα έτσι ώστε να φτάσει σε λύση.

Πριν ακόμα φτάσουμε τα εκατό νουκλεοτίδια, η ECL$^i$PS$^e$ και ο ILOG SOLVER μένουν από πόρους, αφού έχουν ήδη χρησιμοποιήσει όλη τη διαθέσιμη πραγματική και εικονική μνήμη. Από την άλλη πλευρά, ο NAXOS προσαρμόζεται ομαλά, επειδή εκμεταλλευόμενος την προτεινόμενη απεικόνιση πεδίου τιμών, απαιτεί τάξεις μεγέθους λιγότερη μνήμη.

## 4 Συμπεράσματα και Μελλοντικές Κατευθύνσεις

Η απεικόνιση ενός πεδίου τιμών μέσα από ένα δυαδικό δένδρο αναζήτησης που περιέχει τα *κενά* του, είδαμε ότι είναι εξαιρετικά χρήσιμη όταν έχουμε να κάνουμε με μεγάλα πεδία. Είναι ενδιαφέρον να εξετάσουμε στη συνέχεια την απόδοση ενός υβριδικού δένδρου αναζήτησης, οι κόμβοι του οποίου θα περιέχουν πίνακες bit.

Όσον αφορά τη μηχανή διάδοσης περιορισμών AC-5+ προέκυψε από το «πάντρεμα» της θεωρίας στην οποία στηρίζονται οι αλγόριθμοι διάδοσης περιορισμών που χρησιμοποιούνται στους σύγχρονους επιλυτές και της πιο «κλασικής» θεωρίας των πρώτων αλγορίθμων επιβολής συνέπειας ακμών –συγκεκριμένα του AC-5. Θα ήταν ενδιαφέρον να δούμε μελλοντικά το πώς μπορεί να γίνει αυτή η μεθοδολογία κατανεμημένη, έτσι ώστε να αυξήσουμε την αποδοτικότητά της, αξιοποιώντας όλους τους υπολογιστικούς πόρους που έχουμε στη διάθεσή μας.

**Ευχαριστίες.** Θα ήθελα να ευχαριστήσω τον επιβλέποντα Επίκουρο Καθηγητή Παναγιώτη Σταματόπουλο για τις συμβουλές του και τη συνεχή βοήθειά του σε όλη την πορεία της εκπόνησης της εργασίας, σε όποιο μέρος της Ελλάδας και αν βρέθηκα, ειδικά κατά τη διάρκεια της στρατιωτικής μου θητείας. Επίσης ευχαριστώ θερμά τον Επίκουρο Καθηγητή Σταύρο Κολλιόπουλο για τη συμμετοχή του στην

επιτροπή εξέτασης της εργασίας, καθώς και για το γεγονός ότι με έκανε να δω πολλά πράγματα από διαφορετική (θεωρητική) σκοπιά· ευχαριστώ και όσους άλλους παραβρέθηκαν στην παρουσίαση της εργασίας. Τέλος ευχαριστώ όσους κατά τη στρατιωτική μου θητεία ενδιαφέρθηκαν για τούτη τη δουλειά και με βοήθησαν ποικιλοτρόπως σε αυτήν· ευχαριστώ ιδιαίτερα τον Ανθυπολοχαγό Απόστολο Πουρνάρα, τον Συνταγματάρχη Γιώργο Πόθο και τον Λοχαγό Βασίλη Αναστόπουλο.

## Αναφορές

1. Backofen, R., Gilbert, D.: Bioinformatics and constraints. Constraints 6(2-3), 141–156 (2001)
2. Bessière, C., Régin, J.C., Yap, R.H.C., Zhang, Y.: An optimal coarse-grained arc consistency algorithm. Artificial Intelligence 165(2), 165–185 (2005)
3. Codognet, P., Diaz, D.: Compiling constraints in `clp(FD)`. The Journal of Logic Programming 27(3), 185–226 (1996)
4. ECL$^i$PS$^e$ constraint programming system. `http://eclipse-clp.org` (2008)
5. Freuder, E.C.: In pursuit of the holy grail. ACM Comput. Surv. 28(4es), 63 (1996)
6. di Gaspero, L., McCollum, B., Schaerf, A.: The 2$^{\text{nd}}$ int'l timetabling competition (ITC-2007): Curriculum-based course timetabling (Track 3). Tech. rep. (2007)
7. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Data structures for generalised arc consistency for extensional constraints. In: AAAI '07: 22$^{\text{nd}}$ National Conf. on Artificial Intelligence, Vancouver. pp. 191–197. AAAI Press, Menlo Park (2007)
8. Gu, J., Purdom, P.W., Franco, J., Wah, B.W.: Algorithms for satisfiability (SAT) problem: A survey. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 35, 19–152 (1996)
9. van Hentenryck, P., Deville, Y., Teng, C.M.: A generic arc-consistency algorithm and its specializations. Artificial Intelligence 57(2-3), 291–321 (1992)
10. ILOG S.A.: ILOG Solver 4.4: User's Manual (1999)
11. Mackworth, A.K.: Consistency in networks of relations. Artificial Intelligence 8(1), 99–118 (1977)
12. Pothitos, N.: Naxos Solver. `http://www.di.uoa.gr/~pothitos/naxos` (2010)
13. Pothitos, N., Stamatopoulos, P.: Flexible management of large-scale integer domains in CSPs. In: SETN 2010: 6$^{\text{th}}$ Hellenic Conf. on Artificial Intelligence, Athens. LNAI, vol. 6040, pp. 405–410. Springer, Heidelberg (2010), to appear
14. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, chap. 6, pp. 202–233. Prentice Hall, New Jersey, 3$^{\text{rd}}$ edn. (2009)
15. Sabin, D., Freuder, E.C.: Contradicting conventional wisdom in constraint satisfaction. In: PPCP '94: 2$^{\text{nd}}$ Int'l Workshop on Principles and Practice of Constraint Programming, Washington. LNCS, vol. 874, pp. 125–129 (1994)
16. Schulte, C., Carlsson, M.: Finite domain constraint programming systems. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constr. Programming, chap. 14, pp. 495–526. Foundations of Artif. Intell., Elsevier Science, Amsterdam (2006)
17. Walsh, T.: Depth-bounded discrepancy search. In: IJCAI'97: 15$^{\text{th}}$ Int'l Joint Conf. on Artif. Intell., Nagoya. pp. 1388–1393. Morgan Kaufmann, San Francisco (1997)
18. Zytnicki, M., Gaspin, C., Schiex, T.: A new local consistency for weighted CSP dedicated to long domains. In: SAC '06: 2006 ACM Symposium on Applied Computing, Dijon. pp. 394–398. ACM, New York (2006)
19. Θεοχάρης, Φ.: Προηγμένες Μέθοδοι Αναζήτησης για Προβλήματα Ικανοποίησης Περιορισμών. In: Επιλογή Διπλωματικών και Πτυχιακών Εργασιών, vol. 5, pp. 33–42. Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Ε.Κ.Π.Α. (2008)