Blum Complexity

Αλγόριθμοι και Πολυπλοκότητα II

Παναγιώτης Γροντάς

$\mu\Pi\lambda\forall$

Δεκέμβριος 2011

Ιστορικά Στοιχεία

- Manuel Blum (1938, Caracas Venezuela)
- Turing Award (1995)
  - Foundations Of Computational Complexity
  - Cryptography
  - Program Checking
- Δείγματα Δουλειάς (μεταξύ άλλων):
  - Median Of Medians Algorithm
  - Blum, Blum, Shub pseudo - number generator
  - Blum - Goldwasser cryptosystem
  - CAPTCHAs
  - Blum Complexity
- Γνωστοί μαθητές:
  - Michael Sipser
  - Leonard Adleman
  - Gary Miller
  - Vijay Vazirani
  - Ryan Williams

Blum Complexity

- Χαρακτηρισμός πολυπλοκότητας υπολογίσιμων συναρτήσεων (με top-down τρόπο).
- Ανεξαρτησία από μοντέλο μηχανής (δεν έχει σημασία πλήθος ταινιών, κωδικοποίηση εισόδου κτλ.)
- Ποιες είναι οι ’νόμιμες’ συναρτήσεις μέτρησης πολυπλοκότητας (πέρα από βήματα, κελιά);
  - **Αξιώματα Blum**
- Υπάρχουν προβλήματα για τα οποία δεν υπάρχει βέλτιστος αλγόριθμος!
  - **Speedup Theorem**

Συμβολισμός

- Ακολουθία μηχανών $Z_0, Z_1, Z_2 \ldots$
- Δεν έχουν σημασία τα χαρακτηριστικά τους (πλήθος ταινιών/κωδικοποίηση εισόδου)
- Συσχετίζουμε με κάθε μηχανή δύο συναρτήσεις:
  - Συνάρτηση που υπολογίζεται από την μηχανή Z_i : $\phi_i(n)$
  - Συνάρτηση *'καταμέτρησης βημάτων'* υπολογισμού: $\Phi_i(n)$

Άρα:

- Σύνολο partial recursive functions $\{\phi_i\}$
- Σύνολο partial recursive functions $\{\Phi_i\}$ το οποίο μπορει να ειναι εντελώς αυθαίρετο εκτός από τό ότι πρέπει να ισχύουν τα εξής αξιώματα:

Αξιώματα

1. Η $\phi_i(n)$ συγκλίνει $\longleftrightarrow$ $\Phi_i(n)$ συγκλίνει
   - Για να μετρήσουμε την πολυπλοκότητα μηχανών Turing αυτές πρέπει να τερματίζουν.
2. Η συνάρτηση

$$M(i,n,m) = \begin{cases} 1, \ if\ \Phi_i(n) = m \\ 0, otherwise \end{cases}$$

   είναι αναδρομική.
   - Τερμάτισε μια μηχανή με συγκεκριμένη είσοδο σε συγκεκριμένο πλήθος βημάτων;
   - Το πλήθος των βημάτων πρέπει να είναι υπολογίσιμο.
3. Μία συνάρτηση η οποία ικανοποιεί τα παραπάνω αξιώματα ονομάζεται μέτρο πολυπλοκότητας (complexity measure / computation measure)

Παράδειγματα

- Το πλήθος βημάτων μιας μηχανής Turing είναι complexity measure.
  Απόδειξη:
  - **Αξίωμα 1**: Το πλήθος βημάτων ορίζεται μόνο για μηχανές Turing που τερματίζουν
  - **Αξίωμα 2**: Η καθολική μηχανή Turing καθώς προσομοιώνει την M μετράει και τα βήματα. Αν ξεπεράσει το καθορισμένο πλήθος, τότε σταματάει.
- Το πλήθος τετραγώνων μιας μηχανής Turing είναι complexity measure.
- Η $\widehat{\Phi}_i(n) = \Phi_i(n)2^{i+n}$ είναι complexity measure αν και μόνο αν η $\Phi_i(n)$ είναι.
- Η $\Phi_i(n) = \phi_i(n)$ δεν είναι complexity measure
  - Ικανοποιείται το Αξίωμα 1 αλλά όχι το Αξίωμα 2:
- Η $\Phi_i(n) = 0$ δεν είναι complexity measure
  - Ικανοποιείται το Αξίωμα 2 αλλά όχι το Αξίωμα 1:

Speed Up Theorem

΄Εστω r μία total recursive function δύο μεταβλητών. Υπάρχει total recursive function f η οποία λαμβάνει τις τιμές 0 και 1 με την ιδιότητα για κάθε Φ_i υπάρχει Φ_j, που αφορούν την f, τέτοιο ώστε $\Phi_i(n) > r(n, \Phi_j(n))$ για σχεδόν όλα τα n.

Πρακτικά:

- Για μία πολύπλοκη συνάρτηση, σε κάθε μηχανή που την υπολογίζει αντιστοιχεί μία ακόμα πιο γρήγορη, η οποία για άπειρα n μπορει να λειτουργήσει με λιγότερα βήματα.
- Δεν μπορούμε να βρούμε το βέλτιστο προγραμμα για μια υπολογίσιμη συνάρτηση και για ένα complexity measure.
- Τα παραπάνω ισχύουν για οποιοδήποτε complexity measure (δηλ. και για χώρο και για χρόνο και για ό,τι άλλο πληρεί τα αξιώματα).

Speed Up Theorem

Για κάθε Φ_i υπάρχει Φ_j, που αφορούν την ίδια υπολογίσιμη συνάρτηση, τέτοια ώστε $\Phi_i(n) > 2\Phi_j(n)$.

Παράδειγμα

$$f(x) = \begin{cases} 1, x\ palindrome \\ 0, otherwise \end{cases}$$

Υπολογισμός του $f(372686273)$.

- Μπορούμε να σβήνουμε ένα χαρακτήρα από τα αριστερά και ένα από τα δεξιά (αν είναι ίδιοι), μέχρι να μείνει ένας χαρακτήρας ή η κενή συμβολοσειρά.

Υπολογισμός με μισά βήματα:

- Μπορούμε να ελέγχουμε για ισότητα και να σβήνουμε 2 χαρακτήρες την φορά.

Περίληψη Απόδειξης[4]

- Έστω $t(n)$ ένα complexity bound.
- Θα **κατασκευάσουμε** μια συνάρτηση $f : \{0, 1\}$ τέτοια ώστε αν η f μπορεί να υπολογιστει σε $O(t(n))$ βήματα μπορεί να υπολογιστεί σε $(t(n - i))$ βήματα $\forall i$.
- Έστω $S_i = \{Z_1, Z_2, ..Z_i\}$ οι πρώτες i μηχανές μιας απαρίθμησης.
- Έστω φυσικός n (είσοδος f).

Περίληψη Απόδειξης

for all $i \in \{1..n\}$ **do**
  Προσομοίωσε κάθε μη ακυρωμένη Z_k στο S_i για $t(n-i)$ βήματα.
  if καμία δεν τερματίσει **then**
    $f(i) \leftarrow 0$
  else
    $M_j \leftarrow$ η πρώτη μηχανή που τερματίζει.
    if $M_j = 0$ **then**
      $f(i) \leftarrow 1$
    else
      $f(i) \leftarrow 0$
    end if
    Ακύρωσε την M_j.
  end if
end for

Περίληψη Απόδειξης

- Υπολογισμός $f(n)$ σε $O(n^2 t(n))$
- Speed Up: Θα ενσωματώσουμε κάποιες τιμές της f στον αλγόριθμο για να γίνει πιο γρήγορος.
  - Ενσωματώνουμε τα M_j που ακυρώθηκαν στις πρώτες i επαναλήψεις.
  - Για να ορίσουμε την f ξεκινάμε από την επανάληψη $i+1$
  - Χρειάζονται $O(n^2 t(n-i))$ βήματα.
- Όσο πιο πολλά βήματα ενσωματώνουμε τόσο πιο γρήγορος γίνεται ο αλγόριθμος.
- Μένει να δειχθεί ότι ο παραπάνω τροπος κατασκευής της f δεν υπάρχει άλλος τρόπος κατασκευής της f.

Speed Up Theorem

Η κατασκευή της f έγινε για μεμονωμένες τιμές της.
Υπάρχει διαδικασία με την οποία μπορούμε να βρούμε γρηγορότερη μηχανή για μια συνάρτηση f;
Όχι. Αυτό οφείλεται στο παρακάτω θεώρημα:
Έστω r total recursive function. Αν είναι αρκετά μεγάλη, τότε δεν μπορει να υπάρξει total recursive function k ώστε:

- Να απαριθμεί μόνο δείκτες της f.
- Σε κάθε δεικτη i για την f να αντιστοιχεί δείκτης $k(j)$ για την f ώστε $\Phi_i(n) > r(n, \Phi_{k(j)}(n))$ για σχεδόν όλα τα n.

Super Speed Up Theorem

Η συνάρτηση r του speed up theorem μπορεί να είναι όσο μεγάλη όσο η Φ_i.

Έστω g μια total recursive function. Υπάρχει total recursive function f τέτοια ώστε:

- Αν i ένας δείκτης για την f τότε $\Phi_i(n) > g(n)$ για σχεδόν όλα τα n.
- Σε κάθε δείκτη i για την f υπάρχει δείκτης j για την f ώστε $\Phi_i(n) > \Phi_j(\Phi_i(n))$ για σχεδόν όλα τα n.

Ισχύει και το αντίθετο:

- Έστω h, f total recursive functions. Υπάρχει δεικτης j για την f ώστε $\Phi_j(n) > h(n)$ για κάθε n.
- Μία 'άσχημα' σχεδιασμένη μηχανή μπορεί να σπαταλήσει έναν τεράστιο αριθμό βημάτων για να υπολογίσει μια απλή συνάρτηση.

Κριτική [3]

Η Blum complexity δεν έχει διαδοθεί ευρύτατα στις μέρες μας γιατί:

- Τα πραγματικά ενδιαφέροντα complexity measures ειναι το πλήθος βημάτων και το πλήθος μνήμης τα οποία έχουν μελετηθεί ανεξάρτητα.
- Τα measures τα οποία προκύπτουν από το speed up και τα λοιπά θεωρήματα είναι τεχνητά μεγάλα και δεν ανταποκρίνονται σε πραγματικές ανάγκες.
- Κάποια μέτρα πολυπλοκότητας που θα μας ενδιέφεραν, όπως πχ. το πλήθος τυχαίων νομισμάτων που πρέπει να χρησιμοποιήσει μια πιθανοτική μηχανή Turing, δεν πληρούν τα αξιώματα της θεωρίας του Blum.

Βιβλιογραφία

1. M. Blum. "A machine-independent theory of the complexity of recursive functions," J. ACM 14:2 (1967), 322-336.
2. J. Seiferas, "Machine-independent complexity theory," pp. 163-186 in The Handbook of Theoretical Computer Science, vol. I: Algorithms and Complexity,
3. Lance Fortnow, Blum Complexity Measures, `http://blog.computationalcomplexity.org/2004/04/blum-complexity-measures.html` (11/12/2011)
4. Scott Aaronson, Paleocomplexity, `http://www.scottaaronson.com/democritus/lec5.html` (11/12/2011)
5. Wikipedia entry On Manuel Blum, `http://en.wikipedia.org/wiki/Manuel_Blum` (11/12/2011)