Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μέση και Χειρότερη Επίδοση Online και Offline Αλγόριθμων για το Bin Packing

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Κ. Μαυρακάκης

Επιβλέπουσα: Φώτω Αφάτη
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Ιούνιος 2004

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μέση και Χειρότερη Επίδοση Online και Offline Αλγόριθμων για το Bin Packing

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Κ. Μαυρακάκης

Επιβλέπουσα: Φώτω Αφάτη
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Ιουλίου 2004.

....	....	....
Φώτω Άφρατη	Ευστάθιος Ζάχος	Γεώργιος Κολέτσος
Καθηγήτρια Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2004

....
Ιωάννης Κ. Μαυρακάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Θέλω να ευχαριστήσω θερμά την κ. Φώτω Αφράτη, καθηγήτρια του Εθνικού Μετσόβιου Πολυτεχνείου, για την ιδιαίτερη συμβολή της στη συγγραφή της παρούσας εργασίας. Οι γνώσεις και η εμπειρία της με βοήθησαν σημαντικά στην ολοκλήρωση της μελέτης αυτής και είναι τιμή μου που συνεργάστηκα μαζί της.

Να ευχαριστήσω επίσης τον Τίμο Ασλανίδη, ο οποίος αφιέρωσε πολύ χρόνο για την σωστή καθοδήγηση της μελέτης μου. Οι υποδείξεις και οι εύστοχες παρατηρήσεις του με βοήθησαν στην επίτευξη μίας ολοκληρωμένης εργασίας. Ομολογώ ότι ήταν ιδιαίτερα συνεργάσιμος μαζί μου και εκτός από το ότι έχει αρκετές γνώσεις, έδειξε επίσης ότι είναι άνθρωπος με ήθος.

Τέλος θέλω να ευχαριστήσω τους γονείς μου, οι οποίοι με τη σωστή διαπαιδαγώγηση που μου παρείχαν, συνέβαλαν σημαντικά στην πορεία μου μέχρι εδώ καθώς και στην αξιοπρεπή ολοκλήρωση των σπουδών μου.

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η μελέτη του *NP-Hard* προβλήματος του *Bin Packing*. Η εργασία ξεκινάει με μία εισαγωγή στη θεωρία της πολυπλοκότητας και τον ορισμό κάποιων πολύ βασικών εννοιών. Στη συνέχεια γίνεται μία εισαγωγή στο πρόβλημα και η αναφορά των κυριότερων κατηγοριών αλγορίθμων που το επιλύουν. Γίνεται αναλυτική μελέτη ενός πλήθους προσσεγγίστικων αλγορίθμων που επιλύουν το *Bin Packing*. Η μελέτη αυτή περιλαμβάνει ανάλυση των χειρότερων (*worst-case*) αλλά και των μέσων επιδόσεων (*average-case*) τους. Αναφέρονται πολλά θεωρήματα με τις αποδείξεις τους. Γίνεται επίσης εφαρμογή αρκετών αλγορίθμων για συγκεκριμένη είσοδο και σύγκριση των αποδόσεων τους. Υπάρχει αναλυτική αναφορά σε ένα *PTAS* με αποδείξεις όλων των θεωρημάτων που το συνοδεύουν. Τέλος αναφέρονται δύο πολύ πρόσφατοι αλγόριθμοι και μελετώνται οι επιδόσεις τους.

Λέξεις κλειδιά

Bin Packing, πολυπλοκότητα, προσσεγγιστικοί αλγόριθμοι, *average-case* ανάλυση, *worst-case* ανάλυση, *offline* αλγόριθμοι, *online* αλγόριθμοι, *bounded-space* αλγόριθμοι, *PTAS*, Αλυσίδες *Markov*.

Abstract

The scope of this thesis is the research of the NP-Hard problem called Bin Packing. The thesis begins with an introduction in complexity theory and the definitions of some basic related meanings. After that, there is an introduction in the problem and a reference in the basic categories of algorithms that solve it. There is an analytical research of a variety of approximation algorithms that solve Bin Packing. This research includes worst-case and average-case analysis of their behavior. There are a lot of theorems with their proofs. The most of the algorithms are practiced for a specific input and their performances are compared. There is an analytical research of a PTAS with all the proofs that accompany it. Finally, two recent algorithms are studied with their performances.

Keywords

Bin Packing, complexity, approximation algorithms, average-case analysis, worst-case analysis, offline algorithms, online algorithms, bounded-space algorithms, PTAS, Markov chains.

Περιεχόμενα

1 ΕΙΣΑΓΩΓΗ ΣΤΗ ΘΕΩΡΙΑ ΤΗΣ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ 10

1.1 ΤΟ ΚΟΣΤΟΣ ΕΝΟΣ ΑΛΓΟΡΙΘΜΟΥ 11

1.2 ΠΡΟΒΛΗΜΑΤΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ ΚΑΙ ΑΠΟΦΑΣΗΣ 12

1.3 ΟΙ ΚΛΑΣΕΙΣ P ΚΑΙ NP 14

1.4 Η ΕΝΝΟΙΑ ΤΗΣ ΑΝΑΓΩΓΗΣ 15

1.5 NP – Hard ΚΑΙ NP – Complete ΠΡΟΒΛΗΜΑΤΑ 16

1.6 ΠΡΟΣΕΓΓΙΣΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ 16

2 ΕΙΣΑΓΩΓΗ ΣΤΟ BIN PACKING 18

2.1 ΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ 18

2.2 ΑΝΑΛΥΣΗ ΧΕΙΡΟΤΕΡΗΣ ΕΠΙΔΟΣΗΣ 19

2.3 ΒΑΣΙΚΕΣ ΚΑΤΗΓΟΡΙΕΣ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ 19

2.3.1 ONLINE ΚΑΙ OFFLINE ΑΛΓΟΡΙΘΜΟΙ . . 19

2.3.2 BOUNDED – SPACE ΑΛΓΟΡΙΘΜΟΙ 21

3 ΜΕΛΕΤΗ ONLINE ΑΛΓΟΡΙΘΜΩΝ 23

3.1 NEXT FIT 23

3.2 FIRST FIT 28

3.3 BEST FIT 35

3.4 WORST FIT 39

3.5 ALMOST WORST FIT 42

3.6 ANY FIT ΑΛΓΟΡΙΘΜΟΙ 42

3.7 BOUNDED-SPACE ΑΛΓΟΡΙΘΜΟΙ 43

3.7.1 ΟΙ NF_K, BBF_K, ABF_K ΚΑΙ AFB_K ΑΛΓΟΡΙΘΜΟΙ 44

3.7.2 ΟΙ $Harmonic_K$ ΑΛΓΟΡΙΘΜΟΙ 48

3.7.3 ΣΥΓΚΡΙΣΗ 52

3.8 ΑΥΘΑΙΡΕΤΟΙ ONLINE ΑΛΓΟΡΙΘΜΟΙ 53

4 ΜΕΛΕΤΗ OFFLINE ΑΛΓΟΡΙΘΜΩΝ **58**
4.1 ΟΙ SEMI-ONLINE ΑΛΓΟΡΙΘΜΟΙ 58
4.2 Ο FIRST FIT DECREASING 60
4.3 Ο BEST FIT DECREASING 66
4.4 ΣΥΓΚΡΙΣΗ FFD ΜΕ BFD 67

5 ΜΕΛΕΤΗ ΜΕΣΩΝ ΕΠΙΔΟΣΕΩΝ **69**
5.1 ΟΡΟΛΟΓΙΑ 69
5.2 BOUNDED-SPACE ONLINE ΑΛΓΟΡΙΘΜΟΙ 71
5.3 ΑΠΟΤΕΛΕΣΜΑΤΑ ΓΙΑ ΑΛΛΟΥΣ ONLINE ΑΛΓΟΡΙΘΜΟΥΣ 73
5.3.1 ΣΥΝΕΧΕΙΣ ΚΑΤΑΝΟΜΕΣ 73
5.3.2 ΔΙΑΚΡΙΤΕΣ ΚΑΤΑΝΟΜΕΣ 78
5.4 OFFLINE ΑΛΓΟΡΙΘΜΟΙ 81

6 APPROXIMATION SCHEMES **83**
6.1 ΟΡΙΣΜΟΙ 83
6.2 ΕΝΑ ΑΣΥΜΠΤΩΤΙΚΟ PTAS 84

7 ΔΥΟ ΠΡΟΣΦΑΤΟΙ ΑΛΓΟΡΙΘΜΟΙ **89**
7.1 ΑΛΥΣΙΔΕΣ MARKOV 89
7.1.1 ΟΡΙΣΜΟΣ 89
7.1.2 ΠΡΟΣΕΓΓΙΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ . . 89
7.2 RANDOM FIT 91
7.2.1 ΜΕΛΕΤΗ ΧΕΙΡΟΤΕΡΗΣ ΕΠΙΔΟΣΗΣ . . . 92
7.2.2 RANDOM-FIT(d) 93
7.2.3 ΜΕΛΕΤΗ ΜΕΣΗΣ ΕΠΙΔΟΣΗΣ 93
7.3 SUM-OF-SQUARES 94
7.3.1 ΠΑΡΑΛΛΑΓΕΣ 97

8 ΣΥΜΠΕΡΑΣΜΑΤΑ **103**

Κεφάλαιο 1

ΕΙΣΑΓΩΓΗ ΣΤΗ ΘΕΩΡΙΑ ΤΗΣ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

Από το ξεκίνημα της ύπαρξής του, ο άνθρωπος βασανίζεται από διάφορα προβλήματα, των οποίων η λύση βοήθησε τη ζωή του να γίνει περισσότερο λειτουργική. Με το πέρασμα των αιώνων και την ανάπτυξη της μαθηματικής επιστήμης, περάσαμε στο στάδιο της απεικόνισης των προβλημάτων με μαθηματικούς όρους και της θεμελίωσης των κατάλληλων προυποθέσεων για την αποδοτική επίλυσή τους.

Έτσι επινοήθηκαν κατά καιρούς διάφοροι αλγόριθμοι για την επίλυση αυτών των προβλημάτων. Πώς ορίζεται όμως ένας αλγόριθμος; Αν και δεν υπάρχει αυστηρός ορισμός, αλγόριθμο ονομάζουμε συνήθως ένα πεπερασμένο σύνολο κανόνων, οι οποίοι περιγράφουν μία μέθοδο για να λυθεί ένα συγκεκριμένο πρόβλημα. Η μέθοδος την οποία περιγράφουν, αποτελείται από μία σειρά υπολογιστικών διεργασιών.

Όμως η έρευνα δεν σταματάει στην εύρεση απλά ενός αλγόριθμου που να επιλύει ένα συγκεκριμένο πρόβλημα, αλλά προχωράει στη μελέτη των μετρήσιμων ιδιοτήτων που χαρακτηρίζουν την αποδοτικότητα μίας υπολογιστικής μεθόδου. Τέτοια μετρήσιμα μεγέθη είναι, για παράδειγμα, ο χρόνος υπολογισμού, ο χώρος στη μνήμη του υπολογιστή και αρκετά άλλα που χαρακτηρίζουν την πολυπλοκότητα ενός αλγορίθμου. Όμως και ολόκληρα προβλήματα, εκτός των αλγορίθμων, χαρακτηρίζονται για την πολυπλοκότητά τους. Συγκεκριμένα, η πολυπλοκότητα ενός προβλήματος χαρακτηρίζεται από την πολυπλοκότητα του βέλτιστου αλγορίθμου που επιλύει το συγκεκριμένο πρόβλημα.

Η μελέτη της συμπεριφοράς ενός αλγορίθμου μπορεί να γίνει σε δύο πεδία. Μελετάται η μέση απόδοση (*average-case* ανάλυση) του και η χειρότερη δυνατή (*worst-case* ανάλυση), συμπεριλαμβάνοντας όλα τα πιθανά στιγμιότυπα του συγκεκριμένου προβλήματος που επιλύει.

Ένας αλγόριθμος μπορεί να είναι ντετερμινιστικός ή μη ντετερμινιστικός. Οι ντετερμινιστικοί αλγόριθμοι διακρίνονται από το ότι ο υπολογισμός που προτείνουν είναι γραμμικός. Για κάθε υπολογιστική τους διαμόρφωση υπάρχει ακριβώς μία νόμιμη επόμενη διαμόρφωση. Η υπολογιστική τους διαδικασία προχωράει βήμα προς βήμα και είναι σε θέση να σταματήσει για οποιαδήποτε δυνατή είσοδο. Οι μη ντετερμινιστικοί αλγόριθμοι διακρίνονται από το ότι για κάθε υπολογιστική τους διαμόρφωση μπορεί να υπάρχουν πολλές, μία ή και καμία νόμιμες επόμενες υπολογιστικές διαμορφώσεις. Επομένως οι ντετερμινιστικοί αλγόριθμοι αποτελούν μία υποπερίπτωση των αντίστοιχων μη ντετερμινιστικών. Οι μη ντετερμινιστικοί αλγόριθμοι αποτελούνται από δύο διαφορετικές φάσεις. Στην πρώτη φάση μαντεύουν μία ακολουθία από υπολογιστικές διαμορφώσεις, ενώ στη δεύτερη φάση, κάτω από μία καθαρά ντετερμινιστική διαδικασία ελέγχουν αν το αποτέλεσμα που δίνει η πρώτη φάση αποτελεί τη λύση του προβλήματος.

1.1 ΤΟ ΚΟΣΤΟΣ ΕΝΟΣ ΑΛΓΟΡΙΘΜΟΥ

Το κόστος ενός αλγόριθμου εξαρτάται από την της είσοδο του προβλήματος. Συνήθως αυξάνεται, όταν αυξάνεται το μέγεθος της εισόδου, έστω n.

Τις περισσότερες φορές είναι πιο πρακτικό να υπολογίζουμε την τάξη μεγέθους του κόστους ενός αλγόριθμου παρά την ακριβή τιμή του. Εκείνο που μας ενδιαφέρει είναι η ασυμπτωτική συμπεριφορά του. Με άλλα λόγια αναζητούμε την οριακή αυξητική τάση της συνάρτησης που εκφράζει την πολυπλοκότητα του αλγόριθμου καθώς αυξάνεται το μέγεθος της εισόδου.

Παρακάτω θα ορίσουμε κάποια σύμβολα που χρησιμοποιούνται στην μελέτη της αυξητικής τάσης μίας συνάρτησης και που χρησιμοποιούνται σε μεγάλο βαθμό στη συνέχεια, για τη μελέτη αλγορίθμων που επιλύουν το πρόβλημα του *Bin Packing*.

Έστω $f : N\ \{0\} \rightarrow N\ \{0\}$ και $g : N\ \{0\} \rightarrow N\ \{0\}$, όπου N οι φυσικοί αριθμοί. Τότε έχουμε τους παρακάτω συμβολισμούς:

$$O(g) = \{f \mid \exists c > 0, \exists n_0 : \forall n > n_0\ f(n) \leq cg(n)\}$$

$$o(g) = \{f \mid \forall c > 0, \exists n_0 : \forall n > n_0\ f(n) \leq cg(n)\}$$

$\Omega(g) = \{f \mid \exists c > 0, \exists n_0 : \forall n > n_0 \ f(n) \geq cg(n)\}$

$\omega(g) = \{f \mid \forall c > 0, \exists n_0 : \forall n > n_0 \ f(n) \geq cg(n)\}$

$\Theta(g) = \{f \mid \exists c_1 > 0, \ \exists c_2 > 0 \ \exists n_0 : \forall n > n_0 \ c_1 \leq \frac{f(n)}{g(n)} \leq c_2)\}$

Αν $g \in O(f)$, συνήθως γράφουμε $g(n) = O(f(n))$ και λέμε ότι η συνάρτηση g είναι τάξης μεγέθους f.

Μία συνάρτηση g είναι πολυωνυμικού βαθμού ως προς n, όταν $g \in O(poly) = \bigcup O(n^k)$.

Το βασικότερο κριτήριο για το αν ένας αλγόριθμος είναι αποδοτικός ή όχι, είναι το χρονικό κόστος που παρουσιάζει ή καλύτερα η χρονική πολυπλοκότητα του. Αυτήν την παριστάνουμε με μία συνάρτηση $T(n)$. Η $T(n)$ αποδίδει τον αριθμό των βημάτων που πραγματοποιεί ένας αλγόριθμος, συναρτήσει του μεγέθους της εισόδου n.

1.2 ΠΡΟΒΛΗΜΑΤΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ ΚΑΙ ΑΠΟΦΑΣΗΣ

Τα προβλήματα διακρίνονται σε διάφορες κατηγορίες ανάλογα με τα χαρακτηριστικά τους. Θα αναφέρουμε εδώ δύο από τις βασικότερες κατηγορίες, τα προβλήματα βελτιστοποίησης και τα προβλήματα απόφασης.

Στα προβλήματα βελτιστοποίησης, αναζητάμε τη λύση εκείνη που μεγιστοποιεί ή ελαχιστοποιεί κάποια αντικειμενική συνάρτηση. Για κάθε στιγμιότυπο ενός τέτοιου προβλήματος, υπάρχει ένα σύνολο εφικτών λύσεων $F(x)$. Για κάθε λύση $s \in F(x)$, αντιστοιχούμε, μέσω μίας αντικειμενικής συνάρτησης c, ένα θετικό ακέραιο $c(s)$ και ζητάμε τη λύση εκείνη s που βελτιστοποιεί[1] το $c(s)$. Το πρόβλημα του *Bin Packing* είναι ένα προβλημα βελτιστοποίησης και πιο συγκεκριμένα ένα πρόβλημα ελαχιστοποίησης.

Μία άλλη κατηγορία προβλημάτων είναι τα προβλήματα απόφασης. Στην κατηγορία αυτή οι πιθανές απαντήσεις είναι δύο και είναι της μορφής 'ναι' ή 'οχι'. Αν όλες οι απαντήσεις που δίνουν 'ναι', υποθέσουμε ότι ανήκουν σε ένα σύνο-

[1]Το κάνει μέγιστο ή ελάχιστο, ανάλογα με τον αν έχουμε πρόβλημα μεγιστοποίησης ή ελαχιστοποίησης.

λο Α, τότε τα προβλήματα αυτά, για κάθε είσοδο x, αποκτούν τη μορφή ΄$x \epsilon A$;΄

Έστω ότι έχουμε ένα πρόβλημα βελτιστοποίησης με είσοδο x και αναζητάμε μία εφικτή λύση s που να βελτιστοποιεί την αντικειμενική συνάρτηση $c(s)$. Αυτό το πρόβλημα, καθώς και οποιοδήποτε πρόβλημα βελτιστοποίησης, μπορούμε να το μετατρέψουμε στο αντίστοιχο πρόβλημα απόφασης: Για είσοδο x, n υπάρχει εφικτή λύση s για το x τέτοια ώστε το $c(s)$ να είναι καλύτερο του n; Για προβλήματα ελαχιστοποίησης εξετάζεται αν $c(s) < n$, ενώ για προβλήματα μεγιστοποίησης αν $c(s) > n$.

Γενικά είναι ιδιαίτερα πρακτικό να μετατρέπουμε όλες τις κατηγορίες προβλημάτων σε προβλήματα απόφασης, καθώς τα τελευταία μπορούν να μελετηθούν με ιδιαίτερα αποδοτικό τρόπο μέσω των τυπικών γλώσσων. Αν συνυπολογίσουμε το γεγονός ότι για να εισάγουμε ένα στιγμιότυπο ενός προβλήματος σε ένα υπολογιστικό μοντέλο, αυτό πρέπει να κωδικοποιηθεί από ένα πεπερασμένο σύνολο συμβόλων, τότε γίνεται κατανοητός ο λόγος που θέλουμε το πρόβλημά μας να έχει άμεση σχέση με τις τυπικές γλώσσες. Ας αναφέρουμε όμως κάποια βασικά στοιχεία γι΄ αυτές.

Έστω ότι έχουμε ένα πεπερασμένο σύνολο Σ από σύμβολα. Ορίζουμε ως Σ^* το σύνολο των πεπερασμένου μήκους συμβολοσειρών που αποτελούνται από σύμβολα του Σ. Αν για παράδειγμα, $\Sigma = \{0,1\}$ τότε $\Sigma^* = \{\varepsilon, 1, 0, 10, 101, 0111, \ldots\}$. Είναι προφανές ότι το σύνολο Σ^* περιέχει άπειρα στοιχεία, καθώς υπάρχουν απεριόριστοι συνδυασμοί των συμβόλων του Σ. Λέμε ότι η L είναι γλώσσα του αλφάβητου[2] Σ όταν $L \subseteq \Sigma^*$.

Για κάθε πρόβλημα απόφασης, υπάρχουν στιγμιότυπα για τα οποία η απάντηση είναι ΄ναι΄ και άλλα για τα οποία είναι ΄οχι΄. Ας υποθέσουμε ότι για ένα πρόβλημα Π, τα ΄ναι΄ στιγμιότυπα ανήκουν στο σύνολο N_Π, ενώ τα ΄οχι΄ ανήκουν στο O_Π. Αν μία γλώσσα L αποτελείται από τα *strings*, έστω x, που αποτελούν κωδικοποιήσεις του προβλήματος για τα οποία η απάντηση είναι ΄ναι΄, τότε αυτή ορίζεται ως εξής:

$$L(\Pi, k) = \{x \in \Sigma^* |\ x \text{ αναπαριστά ένα στιγμιότυπο } I \in N_\Pi\}$$

Με k συμβολίζουμε τον τρόπο κωδικοποίησης του προβλήματος. Ανάλογα με την κωδικοποίηση που επιλέγεται διαφοροποιούνται φυσικά και τα σύνολα N_Π και O_Π. Επομένως, ένα *string* x ανήκει στη γλώσσα L αν το κάποιο στιγμιότυπο του πρόβληματος απόφασης Π που παριστάνεται με το x μέσω

[2]Ένα πεπερασμένο σύνολο από σύμβολα όπως το Σ, το ονομάζουμε αλφάβητο.

μίας κωδικοποίησης k δίνει απάντηση 'ναι'.

1.3 ΟΙ ΚΛΑΣΕΙΣ P ΚΑΙ NP

Είπαμε προηγουμένως ότι κάθε πρόβλημα ανήκει σε μία συγκεκριμένη τάξη πολυπλοκότητας ανάλογα με την πολυπλοκότητα το βέλτιστου αλγόριθμου που το επιλύει. Θα αναφερθούμε εδώ σε δύο πολύ σημαντικές κλάσεις προβλημάτων, την P και την NP. Τα προβλήματα που ανήκουν στη κλάση P μπορούν να επιλυθούν με αποδοτικούς αλγόριθμους. Αντίθετα, τα αντίστοιχα της κλάσης NP θεωρούνται 'δύσκολα' προβλήματα και δεν έχει ανακαλυφθεί μέχρι στιγμής κάποιος αποδοτικός[3] αλγόριθμος που να τα επιλύει. Θα αναφερθούμε παρακάτω περισσότερο αναλυτικά για τις δύο αυτές κλάσεις.

Ονομάζουμε P ($polynomial$) την κλάση των προβλημάτων τα οποία επιλύονται σε πολυωνυμικό χρόνο από κάποιο ντετερμινιστικό αλγόριθμο.

Είπαμε ότι κάθε αλγόριθμος υλοποιείται σε κάποιο υπολογιστικό μοντέλο. Το μοντέλο στο οποίο θα αναφερόμαστε από δω και πέρα είναι η ντετερμινιστική μηχανή $Turing$ (DTM), στην οποία υποθέτουμε ότι υλοποιείται και ο ντετερμινιστικός αλγόριθμος του παραπάνω ορισμού. Ένα DTM πρόγραμμα με ένα αλφάβητο Σ αποδέχεται ένα $string$ x, όταν σταματάει σε κατάσταση αποδοχής q_N για το x. Επομένως ένας ισοδύναμος ορισμός για την κλάση P είναι:

$$P = \{L \mid \exists \text{ πολυωνυμικού χρόνου } DTM \text{ που αποδέχεται τη γλώσσα } L\ \}$$

Να διευκρινίσουμε εδώ ότι ο χρόνος μίας DTM για κάποιο n[4], ορίζεται σαν ο μέγιστος χρόνος για να αποκριθεί η DTM για οποιοδήποτε $string$ εισόδου, έστω x, που έχει μήκος ίσο με n. Μία DTM καλείται πολυωνυμικού χρόνου, αν υπάρχει πολυώνυμο p ώστε $\forall n \in Z^+$: χρόνος $(DTM) \leq p(n)$.

Ονομάζουμε $NP(Non-deterministicPolynomialTime)$ την κλάση των προβλημάτων τα οποία επιλύονται σε πολυωνυμικό χρόνο από κάποιο μη ντετερμινιστικό αλγόριθμο.

Αν θυμηθούμε τον ορισμό των μη-ντετερμινιστικών αλγόριθμων, μπορούμε να πούμε ότι ένα πρόβλημα ανήκει στην NP κλάση όταν, αφού 'μαντέψουμε' μία πιθανή λύση s για το συγκεκριμένο πρόβλημα, μπορεί να γίνει επαλήθευση

[3] Όταν λέμε αποδοτικός αλγόριθμος, εννοούμε αλγόριθμος πολυωνυμικού χρόνου.

[4] Όπου n είναι το μήκος του $string$ εισόδου.

της συγκεκριμένης λύσης σε πολυωνυμικό χρόνο.

Ένας ισοδύναμος ορισμός για την κλάση NP είναι:

$NP = \{L \mid \exists$ πολυωνυμικού χρόνου $NDTM$ που αποδέχεται τη γλώσσα $L\ \}$

Η $NDTM$ είναι μία $Non-deterministic$ μηχανή $Turing$. Μία $NDTM$ επιλέγει κάθε φορά το συντομότερο μονοπάτι, το οποίο οδηγεί σε κατάσταση αποδοχής για ένα συγκεκριμένο $string$ εισόδου, έστω x.

Ποιά είναι όμως η σχέση μεταξύ των κλάσεων P και NP ; Είναι προφανές ότι $P \subseteq NP$. Έχουμε πει ότι οι ντετερμινιστικοί αλγόριθμοι είναι ειδική περίπτωση των μη-ντετερμινιστικων. Επομένως, ένα πρόβλημα που λύνεται σε πολυωνυμικό χρόνο από έναν ντετερμινιστικό αλγόριθμο, λύνεται σε πολυωνυμικό χρόνο και από έναν μη-ντετερμινιστικο αλγόριθμο. Εκείνο που ερευνάται μέχρι σήμερα είναι αν τα προβλήματα που λύνονται σε πολυωνυμικό χρόνο από μία $NDTM$, μπορούν να λυθούν σε πολυωνυμικό χρόνο και από μία DTM. Δηλαδή αν $P = NP$.

1.4 Η ΕΝΝΟΙΑ ΤΗΣ ΑΝΑΓΩΓΗΣ

Τα προβλήματα που ανήκουν στις κλάσεις P και NP είναι εξαιρετικά πολλά σε αριθμό. Πολλές φορές τίθεται το ερώτημα, μεταξύ ορισμένων προβλημάτων ποιό είναι περισσότερο πολύπλοκο ή αν μπορεί να συσχετιστεί κάποιο πρόβλημα με ένα άλλο. Απαντήσεις σε τέτοιου είδους προβληματισμούς δίνει η έννοια της αναγωγής. Ας δούμε όμως τον ακριβή ορισμό της:

Λέμε ότι ένα πρόβλημα Β ανάγεται πολυωνυμικά σε ένα πρόβλημα Α και το συμβολίζουμε $B \preceq A$, αν υπάρχει μετασχηματισμός πολυωνυμικού χρόνου R, ο οποίος, για κάθε $x \in B$, να παράγει ένα ισοδύναμο $R(x) \in A$. Με τη λέξη ισοδύναμο εννοούμε ότι αν η είσοδος x του προβλήματος Β[5] οδηγεί σε απάντηση 'ναι', τότε και μόνο τότε η είσοδος $R(x)$ του προβλήματος Α θα οδηγεί επίσης σε απάντηση 'ναι'.Το ίδιο ισχύει και για τυχόν αρνητική απάντηση.

Με λίγα λόγια, αντί να επιλύσουμε το πρόβλημα Β για μία είσοδο x, μπορούμε να υπολογίσουμε το $R(x)$ και να επιλύσουμε το πρόβλημα Α για είσοδο $R(x)$. Και στις δύο περιπτώσεις θα καταλήξουμε στην ίδια λύση.

[5] Θεωρούμε ότι τα προβλήματα Α και Β είναι προβλήματα απόφασης.

1.5 NP – Hard ΚΑΙ NP – Complete ΠΡΟΒΛΗΜΑΤΑ

Όταν ένα πρόβλημα B ανάγεται σε ένα πρόβλημα A τότε μπορούμε να πούμε ότι το A είναι τουλάχιστον το ίδιο 'δύσκολο' πρόβλημα με το B. Κι αυτό ισχύει γιατί αν βρεθεί ένας αποδοτικός αλγόριθμος για το A, τότε αυτόματα μπορεί να λυθεί αποδοτικά και το B. Όσα περισσότερα προβλήματα ανάγονται σε ένα πρόβλημα έστω A, τόσο πιο 'δύσκολο' χαρακτηρίζεται αυτό.

$NP - Hard$ ονομάζονται τα προβλήματα στα οποία ανάγονται όλα τα προβλήματα που ανήκουν στην κλάση NP. Δηλαδή:

Αν $\forall \Pi' \in NP$ ισχύει $\Pi' \preceq \Pi$ τότε το Π είναι *NP-Hard*.

Τα *NP-Hard* προβλήματα που ανήκουν στην κλάση NP ονομάζονται *NP-Complete* προβλήματα.

Συμπεραίνουμε από τα παραπάνω ότι τα *NP-Hard* και τα *NP-Complete* προβλήματα χαρακτηρίζονται από την εξαιρετική τους δυσκολία. Για παράδειγμα αν βρεθεί αλγόριθμος πολυωνυμικού χρόνου που να λύνει ένα από αυτά τα προβλήματα, τότε μπορούν να λυθούν πολυωνυμικά όλα τα προβλήματα της κλάσης NP. Επομένως θα έχει αποδειχτεί ότι $P = NP$. Το πρόβλημα του *Bin Packing* είναι ένα *NP-Hard* πρόβλημα.

1.6 ΠΡΟΣΕΓΓΙΣΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ

Είπαμε στην προηγούμενη παράγραφο για δύο κατηγορίες προβλημάτων με εξαιρετικά υψηλό βαθμό δυσκολίας. Η πολυπλοκότητα τους μας αναγκάζει να ερευνάμε αλγόριθμους που δεν προσφέρουν την πολυτέλεια μίας αποτελεσματικής λύσης κάθε φορά που εφαρμόζονται. Οι αλγόριθμοι αυτοί (*heuristics*) επιστρέφουν εφικτές λύσεις οι οποίες όμως αποκλίνουν πολλές φορές από την *optimal* λύση. Μερικές φορές, ωστόσο, οι λύσεις που προσφέρουν τέτοιου είδους αλγόριθμοι δεν φαίνεται να απέχουν πολύ από την *optimal* λύση.

Ας υποθέσουμε ότι έχουμε ένα πρόβλημα βελτιστοποίησης A. Για κάθε είσοδο x, υπάρχει ένα σύνολο από εφικτές λύσεις $F(x)$. Τη βέλτιστη λύση του συνόλου αυτού την συμβολίζουμε $OPT(x)$. Έστω M ένας αλγόριθμος, ο οποίος για κάθε είσοδο x επιστρέφει μία εφικτή λύση $M(x) \in F(x)$. Ο M είναι ένας $\varepsilon - approximation$ αλγόριθμος[6], όπου $\varepsilon \geq 0$, αν για κάθε x ισχύει:

$$\frac{|c(M(x))-OPT(x)|}{max\{OPT(x),c(M(x))\}} \leq \varepsilon$$

Στην παραπάνω σχέση, η c είναι η αντικειμενική συνάρτηση του προβλήματος βελτιστοποίησης. Επομένως ένας *heuristic* αλγόριθμος είναι $\varepsilon - approximate$, αν το σχετικό σφάλμα της λύσης που προσφέρει είναι το πολύ ε.

Στη συνέχεια θα επικεντρώσουμε τη μελέτη μας στο *NP-Hard* πρόβλημα του *Bin Packing*. Θα μελετήσουμε ένα πλήθος αλγόριθμων που έχουν προταθεί για το συγκεκριμένο πρόβλημα. Η μελέτη μας αφορά τις χειρότερες αλλά και τις μέσες επιδόσεις τόσο κλασσικών αλγόριθμων του προβλήματος, οι οποίοι έχουν παρουσιαστεί εδώ και αρκετά χρόνια, όσο και αλγόριθμων που έχουν επινοηθεί μέχρι και την τελευταία πενταετία.

[6] *Approximation* αλγόριθμοι είναι οι προσεγγιστικοί αλγόριθμοι.

Κεφάλαιο 2

ΕΙΣΑΓΩΓΗ ΣΤΟ BIN PACKING

2.1 ΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Στο κλασσικό μονοδιάστατο *Bin Packing* μας δίνεται μια ακολουθία από $L = (a_1, a_2, ..., a_n)$ αντικείμενα, κάθε ένα με μέγεθος $s(a_i)$ ϵ $(0.1]$ και καλούμαστε να τα συσκευάσουμε σε έναν ελάχιστο αριθμό δοχείων χωρητικότητας ίσης με τη μονάδα (δηλ. να τα χωρίσουμε σε έναν ελάχιστο αριθμό m υποσυνόλων $B_1, B_2 ..., B_m$ έτσι ώστε $\sum_{a_i \epsilon B_j} s(a_i) \leq 1$, $1 \leq j \leq m$).

Με τον όρο ΄μονοδιάστατο΄ εννοούμε ότι τα αντικειμενα περιγράφονται μόνο από έναν αριθμό, ο οποίος μπορεί να συμβολίζει τον όγκο τους, το βάρος τους ή οποιοδήποτε άλλο μέγεθος μπορεί να χρησιμοποιηθεί ανάλογα με την εφαρμογή του προβλήματος. Στο διδιάστατο πρόβλημα τα αντικέιμενα περιγραφονται απο δύο αριθμούς (π.χ. μήκος, πλάτος), ενώ στο τρισδιάστατο από τρεις (π.χ. μήκος, πλάτος, ύψος). Ανάλογα με το πρόβλημα, οι κάδοι στους οποίους τοποθετούνται τα αντικείμενα είναι μονοδιάστατοι, διδιάστατοι, ή τρισδιάστατοι αντίστοιχα.

Εκτός απο το κλασσικό *Bin Packing* υπάρχουν πολλές άλλες παραλλαγές που έχουν μελετηθεί, ιδίως τα τελευταία χρόνια, και που έχουν δώσει διαφορετική διάσταση στο πρόβλημα.

2.2 ΑΝΑΛΥΣΗ ΧΕΙΡΟΤΕΡΗΣ ΕΠΙΔΟΣΗ-Σ

Το κριτήριο με το οποίο θα μελετήσουμε κάποιους συγκεκριμένους αλγόριθμους παρακάτω, είναι η απόδοση τους απέναντι στις λιγότερο βολικές λίστες αντικειμένων.

Ας υποθέσουμε ότι έχουμε μία λίστα αντικειμένων L και έναν $Bin\ Packing$ αλγόριθμο A. Τότε με $A(L)$ συμβολίζουμε τον αριθμό των κάδων που χρησιμοποιούνται όταν εφαρμόζεται ο αλγόριθμος A στη λίστα L. Επίσης με $R_A(L)$ συμβολίζουμε το πηλίκο $\frac{A(L)}{OPT(L)}$ ($R_A(L) = \frac{A(L)}{OPT(L)}$).

Θα ορίσουμε πρώτα το R_A (απόλυτο πηλίκο προσέγγισης χειρότερης επίδοσης[1]) ως εξής: $R_A = inf\{r \geq 1 : R_A(L) \leq r$ για όλες τις λίστες L $\}$

Το R_A^∞ (ασυμπτοτικό πηλίκο προσέγγισης χειρότερης επίδοσης[2]) ορίζεται ως εξής: $R_A^\infty = inf\{r \geq 1 :$ για κάποια $N > 0, R_A(L) \leq r$ για όλες τις λίστες L με $OPT(L) \geq N$ $\}$

Αν έχουμε λίστες που αποτελούνται από αντικείμενα που έχουν μέγεθος το πολύ α (με $a \leq 1$ φυσικά) μπορούμε να χρησιμοποιήσουμε τους συμβολισμούς $R_A(a)$, $R_A^\infty(a)$.

2.3 ΒΑΣΙΚΕΣ ΚΑΤΗΓΟΡΙΕΣ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ

2.3.1 ONLINE ΚΑΙ OFFLINE ΑΛΓΟΡΙΘΜΟΙ

Οι αλγόριθμοι οι οποίοι έχουν επινοηθεί κατα καιρούς για να λυθεί αποδοτικά το πρόβλημα αυτό, χωρίζονται σε δύο βασικες κατηγορίες:

online αλγόριθμοι: Εδώ τα αντικείμενα που αποτελούν την είσοδο του προβλήματος έρχονται ένα προς ένα. Δεν γνωρίζουμε όλοκληρη τη λίστα από την αρχη αλλά μόνο το εκάστωτε τρέχον αντικείμενο το οποίο πρέπει να τοποθετηθεί σε έναν κάδο πριν την άφιξη του επομένου.

offline αλγόριθμοι: Γνωρίζουμε ολόκληρη τη λίστα των αντικειμένων από

[1] *absolute worst-case performance ratio*
[2] *asymptotic worst-case performance ratio*

την αρχή και μπορούμε να τα τοποθετήσουμε με οποιαδήποτε σειρά επιθυμούμε ώστε να πετύχουμε μία αποδοτική λύση.

Από τους παραπάνω ορισμούς αυτούς καθ' αυτούς μπορούμε να συμπεράνουμε ότι οι *offline* αλγόριθμοι είναι κατά βάση περισσότερο αποδοτικοί από τους *online*.

Λήμμα 2.3.1.1 Δεν υπάρχει *online* αλγόριθμος που να επιστρέφει πάντα μία *optimal* λύση.

Απόδειξη: Ας υποθέσουμε ότι $\varepsilon > 0$ είναι ένας πολύ μικρός πραγματικός αριθμός και m ένας θετικός ακέραιος τέτοιος ώστε $m \geq 2$. Ας υποθέσουμε επίσης ότι η είσοδος του προβλήματος, έστω I, αποτελείται από m αντικείμενα μεγέθους $\frac{1}{2} - \varepsilon$, τα οποία ακολουθούνται απο m αντικείμενα μεγέθους $\frac{1}{2} + \varepsilon$. Η *optimal* λύση αυτού του προβλήματος αποτελείται από m κάδους.

Ας υποθέσουμε τώρα ότι υπάρχει ένας *online* αλγόριθμος A που επιστρέφει πάντα την *optimal* λύση. Θα μελετήσουμε την συμπεριφορά του A στην είσοδο I. Μετά τα πρώτα m αντικείμενα, ο A έχει γεμίσει $\lceil \frac{m}{2} \rceil$ κάδους, γιατί αυτή είναι η *optimal* λύση για τα πρώτα m αντικείμενα. Τα επόμενα m αντικείμενα της εισόδου απαιτούν ακόμα m κάδους, εφόσον το m είναι άρτιος, γιατί όλοι οι προηγούμενοι κάδοι ήταν σχεδόν γεμάτοι και επομένως δεν μπορούν να δεχτούν καινούργια αντικείμενα. Αν ο m είναι περιττός θα χρειαζόμασταν $m-1$ καινούργιους κάδους γιατί το πρώτο αντικείμενο της δεύτερης ομάδας θα ταίριαζε στον τελευταίο κάδο της πρώτης ομάδας. Ετσι απαιτούνται συνολικά $\lfloor \frac{3}{2}m \rfloor > m$ κάδοι. Επομένως ο A δεν δίνει την *optimal* λύση. Άτοπο

Θα παραθέσουμε εδώ το Θεώρημα 2.3.1.2 που δείχνει ακριβώς πόσο περιοριστικό είναι για την απόδοση ενός αλγορίθμου, να ανήκει στην κατηγορία των *online*.

Θεώρημα 2.3.1.2 Δεν υπάρχει *online* αλγόριθμος που να επιστρέφει λύση κάτω των $\frac{4}{3}m^*$ κάδων[3].

Απόδειξη: Έστω ότι υπάρχει αλγόριθμος A που επιστρέφει λύση κάτω των $\frac{4}{3}m^*$ κάδων. Θεωρούμε ότι έχουμε την ίδια είσοδο στο πρόβλημά μας με αυτη στο προηγούμενο Λήμμα. Ας υποθέσουμε ότι ο m είναι άρτιος και ότι ο

[3] Το m^* δηλώνει τον βέλτιστο αριθμό κάδων για το πρόβλημα.

Α έχει γεμίσει b κάδους μετά τα πρώτα m αντικείμενα. Η βελτιστη λύση για τα πρώτα m αντικείμενα της εισόδου αποτελείται από $\frac{m}{2}$ κάδους. Επομένως: $b < \frac{4}{3} \cdot \frac{m}{2} \Rightarrow b < \frac{2}{3}m$.

Τώρα θα συνεχίσουμε με τα επόμενα m αντικείμενα. Στους πρώτους b κάδους υπάρχουν το πολύ 2 αντικείμενα ανά κάδο. Αν με τα επόμενα m αντικείμενα άνοιγε ο Α νέους κάδους, x από τους οποίους θα ήταν επίσης γέματοι, τότε σε αυτούς τους x κάδους θα υπάρχει ένα (μεγάλο) αντικείμενο στον καθένα. Επομένως $2b + x \geq 2m$(το μέγεθος της εισόδου). Επομένως $x \geq 2m - 2b$.

Ο συνολικός αριθμός των κάδων είναι τουλάχιστον $x + b \geq 2m - b$. Ο αριθμός των κάδων δεν πρέπει να ξεπερνάει τους $\frac{4}{3}m$, καθώς $m = m^*$ για την είσοδο Ι. Επομένως $2m - b < \frac{4}{3}m \Rightarrow b > \frac{2}{3}m$. Άτοπο.

Να πούμε εδώ ότι το $\frac{4}{3}$ είναι ένα όριο που μπορεί εύκολα να αποδειχτεί και γι' αυτό το παραθέσαμε σ' αυτό το σημείο. Έχει αποδειχτεί πολύ πιο στενό όριο για τους *online* αλγόριθμους στο οποίο θα αναφερθούμε παρακάτω.

2.3.2 BOUNDED – SPACE ΑΛΓΟΡΙΘΜΟΙ

Ορισμός: Ένας *online Bin Packing* αλγόριθμος είναι *K-bounded-space*, εάν σε κανένα χρονικό σημείο της εφαρμογής του, ο αριθμός των ανοικτών κάδων δεν ξεπέρνα τους Κ.

Είναι ένας περιορισμός για τους *online* αλγόριθμους σύμφωνα με τον οποίο μόνο ένας συγκεκριμένος αριθμός[4] μερικώς γεμισμένων κάδων μπορεί να είναι ανοιχτός να δέχεται αντικείμενα.

Όσοι αλγόριθμοι ανήκουν στην κατηγορία των *bounded-space*, μειονεκτούν σε απόδοση αλλα πλεονεκτούν στο χρόνο εφαρμογής τους. Ένας αλγόριθμος ο οποίος δεν εξετάζει όλους τους χρησιμοποιημένους κάδους σαν πιθανούς στόχους για το επόμενο αντικείμενο, αλλά μόνο μερικούς απο αυτούς, θα έχει λιγότερες πιθανότητες να βρει τη βέλτιστη λύση του συγκεκριμένου προβλήματος. Όμως, εφόσον σε κάθε στιγμιότυπο του δεν εξετάζει όλους τους κάδους αλλά μόνο κάποιους από αυτούς, λογικό είναι να ολοκληρώσει την εφαρμογή του σε λιγότερο χρόνο.

[4] Αυτός ο αριθμός ειναι σταθερός για κάθε αλγόριθμο, παίρνει τιμές από 1 εώς ∞ και υποδηλώνεται από τον Κ του ορισμού.

Κάθε αλγόριθμος περιλαμβάνει έναν *packing rule* που προσδιορίζει τον τρόπο με τον οποίο θα τοποθετήσει τα αντικείμενα της εισόδου στους κάδους. Στην περίπτωση όμως των *bounded-space* αλγόριθμων, υπάρχει άλλος ένας κανόνας που τους συνοδεύει και είναι ο *closing rule*. Κάθε φορά που έρχεται ένα καινούργιο αντικείμενο και δεν ταιριάζει στους, έστω Κ, ανοιχτούς κάδους τότε πρέπει να ανοίξει ένας νέος κάδος για να τοποθετηθεί. Πρέπει όμως, λόγω του *bounded-space* περιορισμού, να υπάρχουν το πολύ Κ κάδοι ανοιχτοί ταυτόχρονα. Έτσι για να ανοίξει ένας νέος κάδος πρέπει να κλείσει ένας από τους ήδη υπάρχοντες. Το ποιός ακριβώς θα κλείσει από αυτούς καθορίζεται απο τον *closing rule*.

Στα επόμενα κεφάλαια θα αναφέρουμε μερικούς γνωστούς *bounded-space* αλγόριθμους και θα μελετήσουμε τις αποδόσεις τους.

Κεφάλαιο 3

ΜΕΛΕΤΗ ONLINE ΑΛΓΟΡΙΘΜΩΝ

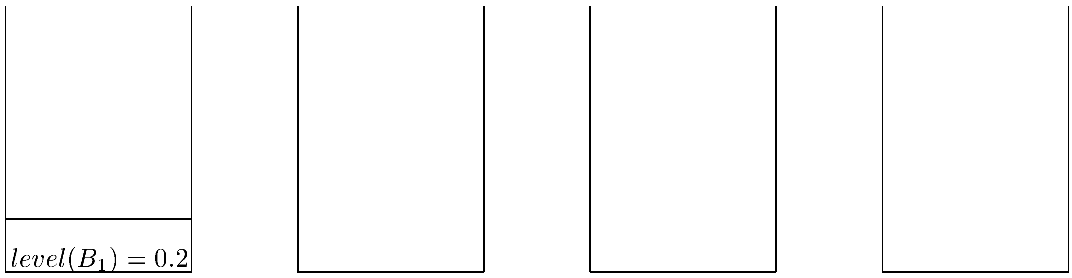
3.1 NEXT FIT

NF Packing Rule : Το πρώτο αντικείμενο a_1 τοποθετείται στον πρώτο κάδο B_1. Αν B_j είναι ο τελευταίος κάδος που έχει ανοίξει και έχει σειρά το αντικείμενο a_i τότε: Απονέμεται το a_i στον B_j, έφόσον αυτός έχει επαρκή χώρο, αλλιώς κλείνει μόνιμα ο B_j και τοποθετείται στον B_{j+1}.

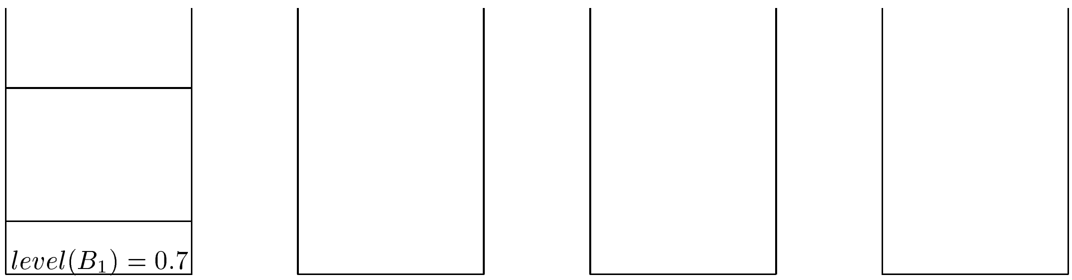
Να σημειώσουμε εδώ ότι ένας κάδος είναι ανοικτός όταν περιέχει ήδη κάποια αντικείμενα και μπορούμε να τοποθετήσουμε κι άλλα σε αυτόν. Αντίθετα ένας κάδος είναι κλειστός όταν έχει χρησιμοποιηθεί αλλά δεν εξετάζεται για οποιοδήποτε από τα επόμενα αντικείμενα.

Στον *Next Fit* είναι ανοικτός μόνο ο τελευταίος κάδος που έχει χρησιμοποιηθεί και οι προηγούμενοι απο αυτόν έχουν κλείσει μόνιμα. Αν το επόμενο αντικείμενο δεν ταιριάζει σε αυτόν, κλείνει κι αυτός μόνιμα και ανοίγει ο επόμενος για να το δεχτεί. Από τον *packing rule* εύκολα να συμπεραίνει κανείς ότι ο *Next Fit* (*NF*) είναι ένας *online* 1-*bounded-space* αλγόριθμος.

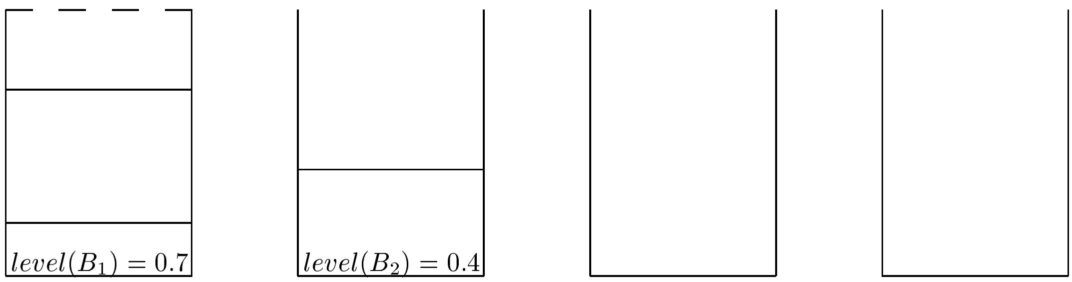
Παράδειγμα: Έστω ότι έχουμε σαν είσοδο τη λίστα $L = \{0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8\}$. Τη συγκεκριμένη λίστα θα χρησιμοποιούμε σαν είσοδο για όλους τους αλγόριθμους και η βέλτιστη λύση γι΄ αυτήν είναι: $B_1 = \{0.8, 0.2\}$, $B_2 = \{0.7, 0.3\}$, $B_3 = \{0.5, 0.4, 0.1\}$ δηλαδή 3 κάδοι. Ο *Next Fit* συμπεριφέρεται ως εξής:



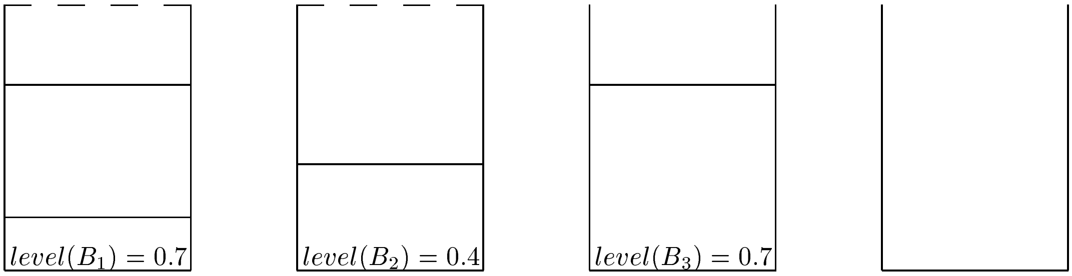
1ο ΒΗΜΑ Το $s(a_1) = 0.2$ τοποθετείται στον πρώτο κάδο.



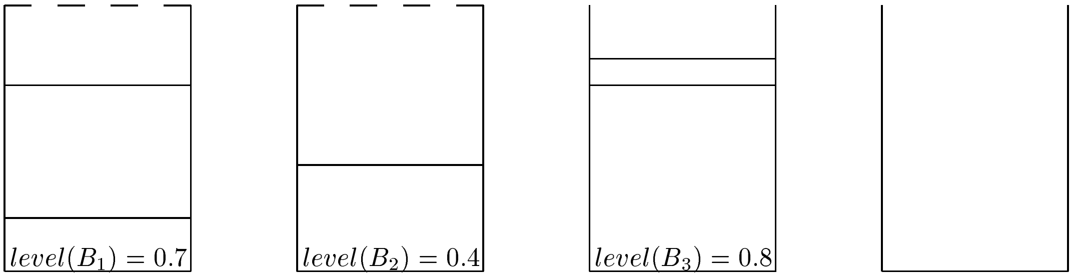
2ο ΒΗΜΑ Το $s(a_2) = 0.5$ τοποθετείται επίσης στον πρώτο κάδο.



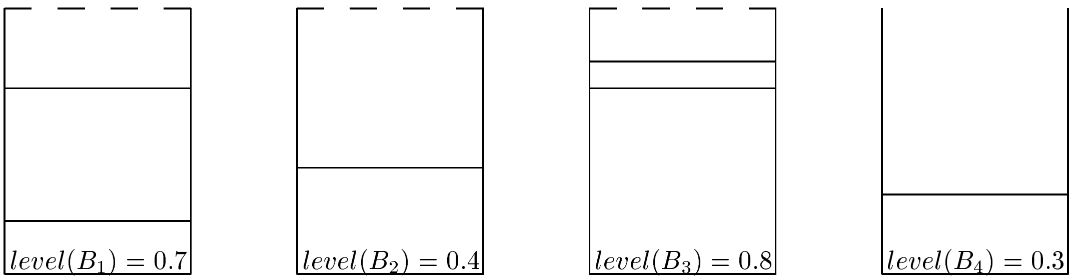
3ο ΒΗΜΑ Το $s(a_3) = 0.4$ τοποθετείται στον B_2 κάδο καθώς δεν ταιριάζει στον B_1. Ο B_1 κλείνει μόνιμα.



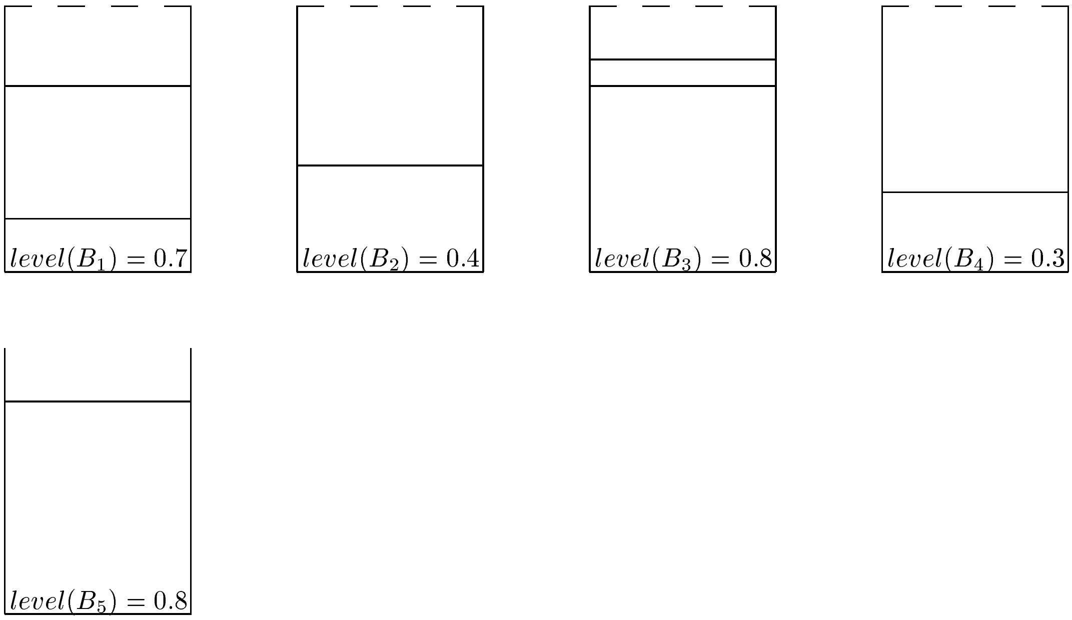
4ο ΒΗΜΑ Το $s(a_4) = 0.7$ τοποθετείται στον B_3 κάδο καθώς δεν ταιριάζει στον B_2. Ο B_2 κλείνει μόνιμα.



5ο ΒΗΜΑ Το $s(a_5) = 0.1$ τοποθετείται επίσης στον B_3 κάδο.



6ο ΒΗΜΑ Το $s(a_6) = 0.3$ τοποθετείται στον B_4 κάδο καθώς δεν ταιριάζει στον B_3. Ο B_3 κλείνει μόνιμα.



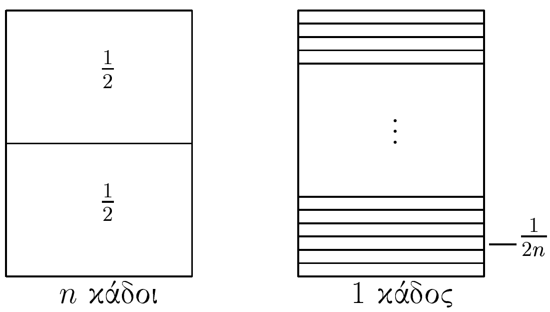
7ο ΒΗΜΑ Το $s(a_7) = 0.8$ τοποθετείται στον B_5 κάδο καθώς δεν ταιριάζει στον B_4. Ο B_4 κλείνει μόνιμα.

Για τη συγκεκριμένη λίστα L, ο NF χρησιμοποιεί 5 κάδους.

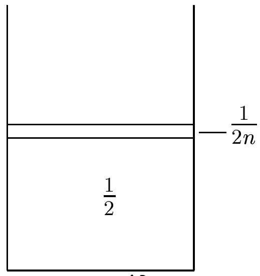
Ο NF μπορεί να τρέξει σε γραμμικό χρόνο. Να πούμε εδώ ότι υπάρχουν λίστες που οδηγούν τον $OPT(L)$ σε αυθαίρετα μεγάλες τιμές και για τις οποίες ισχύει η ισότητα $NF(L) = 2OPT(L) - 1$. Αυτές οι λίστες είναι της μορφής:

$L = \{\frac{1}{2}, \frac{1}{2n}, \frac{1}{2}, \frac{1}{2n}, \ldots, \frac{1}{2}, \frac{1}{2n}\}$

Συνολικά τα αντικείμενα είναι $4n$. Από αυτά, $2n$ αντικείμενα είναι μεγέθους $\frac{1}{2}$, ενώ τα υπόλοιπα $2n$ είναι μεγέθους $\frac{1}{2n}$. Η *optimal* λύση για τη συγκεκριμένη λίστα είναι:



Επομένως $OPT(L) = n+1$ κάδοι. Για την λίστα L, ο NF παραγματοποιεί την εξής τοποθέτηση:



$2n$ κάδοι

Έτσι όπως είναι διατεταγμένα τα αντικείμενα της λίστας L, ο NF τοποθετεί ένα αντικείμενο μεγέθους $\frac{1}{2}$ σε έναν κάδο, στη συνέχεια ένα μεγέθους $\frac{1}{2n}$ στον ίδιο κάδο και το επόμενο που είναι πάλι μεγέθους $\frac{1}{2}$ το τοποθετεί στον επόμενο κάδο καθώς δεν ταιριάζει στον ίδιο. Έτσι ο προηγούμενος κάδος κλέινει μόνιμα. Αυτό συμβαίνει για κάθε κάδο κι έτσι όλοι οι κάδοι που χρησιμοποιεί ο NF, είναι της μορφής του σχήματος. Επομένως $NF(L) = 2n$ κάδοι. Άρα όντως ισχύει $NF(L) = 2OPT(L) - 1$.

Θεώρημα 3.1.1 Για τον *Next Fit* ισχύει: $R^{\infty}_{NF} = 2$

Απόδειξη: Ας υποθέσουμε ότι για μία τυχαία είσοδο L, ο *Next Fit* χρησιμοποίησε h κάδους. Αν $OPT(L)$ είναι ο βέλτιστος αριθμός κάδων, τότε είναι προφανές ότι $OPT(L) \geq \lceil \sum a_i \rceil$, όπου $\sum a_i$ είναι το άθροισμα των αντικειμένων της λίστας L. Κάτι πολύ σημαντικό είναι ότι το άθροισμα των αντικειμένων δύο διαδοχικών κάδων είναι πάντα μεγαλύτερο της μονάδας, γιατί αλλίως ο *Next Fit* θα τα είχε τοποθετήσει στον ίδιο κάδο.

Αν ο h είναι άρτιος τότε έχουμε:

$$level(B_1) + level(B_2) \geq 1$$

$$level(B_3) + level(B_4) \geq 1$$

$$\vdots$$

$$level(B_{h-1}) + level(B_h) \geq 1$$

Αν αθροίσουμε τις παραπάνω ανισότητες κατά μέλη έχουμε: $\sum a_i \geq \frac{h}{2}$

Άρα: $h \leq 2\sum a_i \leq 2\lceil\sum a_i\rceil \leq 2OPT(L)$

Αν ο h είναι περιττός τότε:

$level(B_1) + level(B_2) \geq 1$

$level(B_3) + level(B_4) \geq 1$

$\vdots$

$level(B_{h-2}) + level(B_{h-1}) \geq 1$

Αν αθροίσουμε πάλι τις ανισότητες έχουμε:

$\sum a_i - level(B_h) \geq \frac{h-1}{2} \Rightarrow \sum a_i \geq level(B_h) + \frac{h-1}{2} \Rightarrow \lceil 2\sum a_i\rceil \geq \lceil 2level(B_h) + h - 1\rceil \Rightarrow \lceil 2\sum a_i\rceil \geq h \Rightarrow h \leq \lceil 2\sum a_i\rceil \leq 2\lceil\sum a_i\rceil \leq 2OPT(L)$

Για εισόδους με περιορισμό στα μεγέθη των αντικειμένων, ισχύει το παρακάτω θεώρημα [$Joh73$]:

Θεώρημα 3.1.2 Για τον NF ισχύει:

$R^{\infty}_{NF}(a) = 2,$ για $a \geq \frac{1}{2}$

$R^{\infty}_{NF}(a) = 1/(1-a),$ για $a \leq \frac{1}{2}$

Αξιοσημείωτο είναι ότι για $a \geq \frac{1}{2}$, η *worst-case* συμπεριφορά του NF παραμένει η ίδια, είτε έχουμε μέγιστο μέγεθος 1, είτε $\frac{1}{2}$. Για να πετύχουμε καλύτερη χειρότερη επίδοση πρέπει το μέγιστο δυνατό αντικείμενο της εισόδου να είναι μικρότερο του $\frac{1}{2}$.

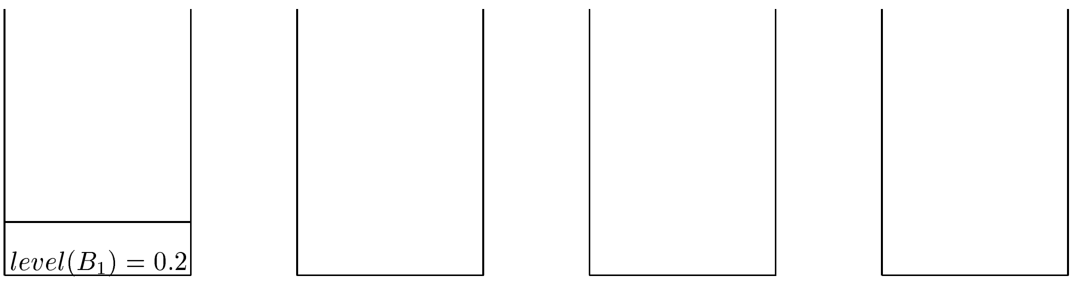
3.2 FIRST FIT

Είδαμε στην προηγούμενη παράγραφο ότι ο *Next Fit* δεν έχει ιδιαίτερα καλή συμπεριφορά λόγω του περιορισμού *bounded-space* ο οποίος είναι έμφυτος σ' αυτόν από τον ορισμό του. Θα αναφερθούμε τώρα σε έναν πιο ιστορικό αλγόριθμο, τον $First\ Fit$ (FF), στον οποίο ο παραπάνω περιορισμός αφαιρείται εξ΄ ολοκλήρου.

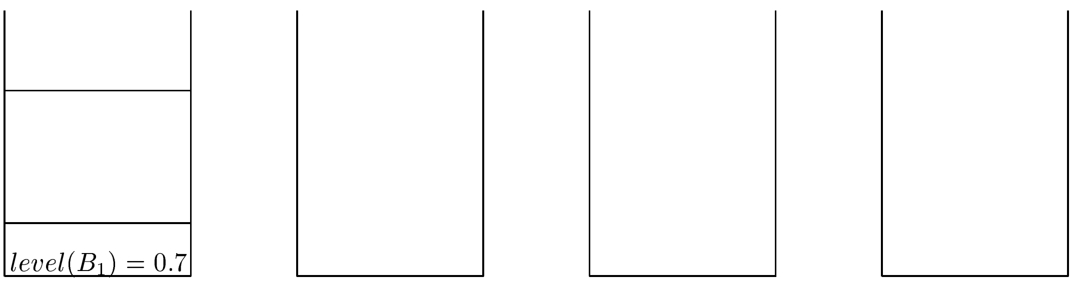
FF Packing Rule : Το τρέχον αντικείμενο a_i τοποθετείται στον πρώτο κάδο στον οποίο ταιριάζει, υποθέτοντας ότι όλοι οι κάδοι που έχουν χρησιμοποιηθεί είναι ανοικτοί.

Εδώ όλοι οι μερικώς γεμισμένοι κάδοι είναι πιθανοί προορισμοί για το επόμενο αντικείμενο. Ο κανόνας που ακολουθείται δίνει εξήγηση και για το όνομα του αλγόριθμου: Τοποθετούμε το κάθε αντικείμενο στον πρώτο κάδο (με το χαμηλότερο δείκτη) στον οποίο ταιριάζει. Δηλαδή το αντικείμενο a_i θα εισέλθει στον πρώτο κάδο για τον οποίο θα ισχύει η σχέση $level(B_j) + s(a_i) \leq 1$. Αν αυτή η σχέση δεν ισχύει για κανέναν μερικώς γεμισμένο (κι επομένως ανοικτό) κάδο, τότε ανοίγουμε έναν καινούργιο και το τοποθετούμε σάυτόν.

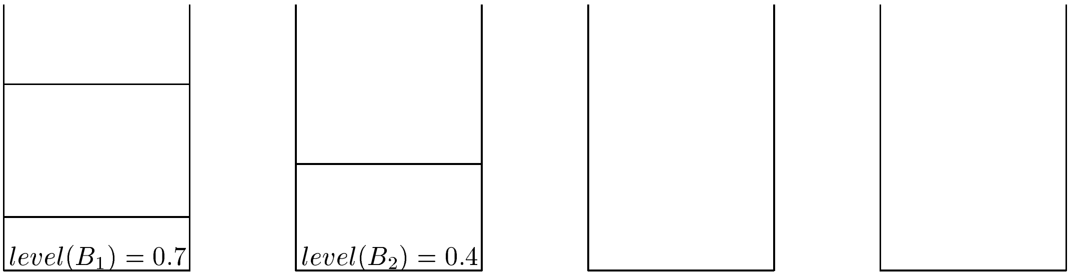
Παράδειγμα: Για τη λίστα $L = \{0.2,\ 0.5,\ 0.4,\ 0.7,\ 0.1,\ 0.3,\ 0.8\}$ με $OPT(L) = 3$ κάδοι, ο FF συμπεριφέρεται ως εξής:



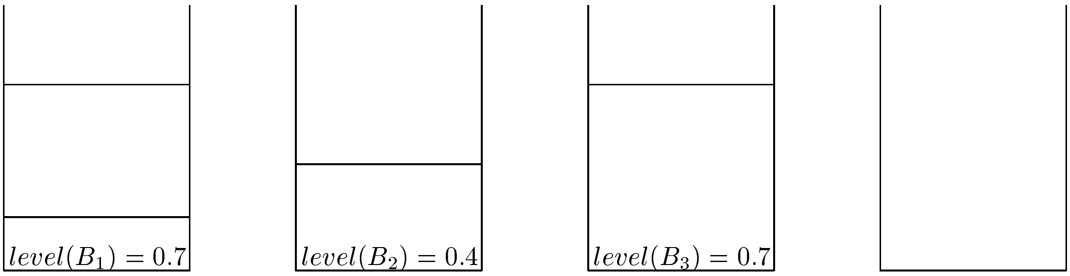
1ο ΒΗΜΑ Το $s(a_1) = 0.2$ τοποθετείται στον πρώτο κάδο.



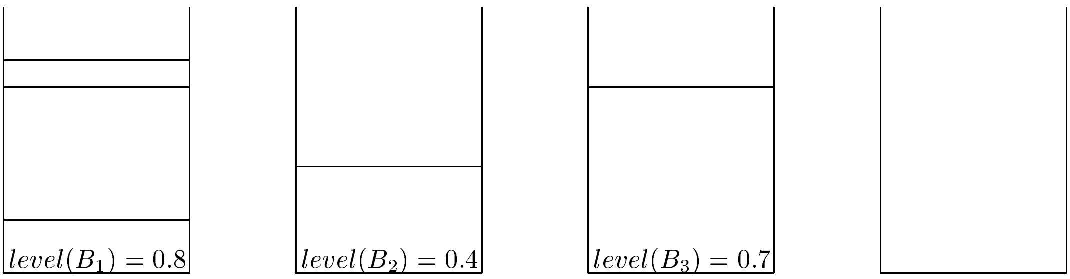
2ο ΒΗΜΑ Το $s(a_2) = 0.5$ τοποθετείται επίσης στον πρώτο κάδο.



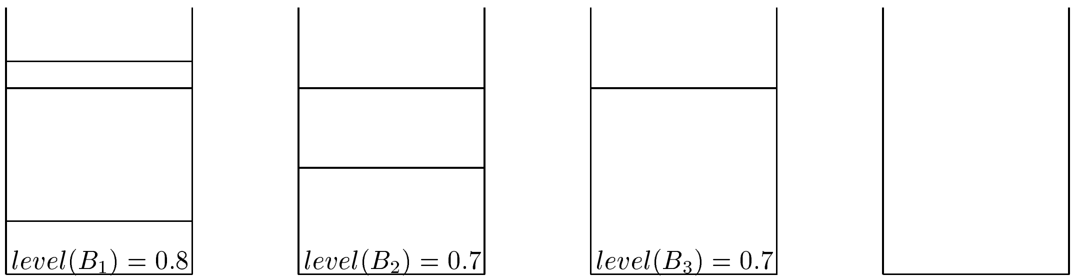
3ο ΒΗΜΑ Το $s(a_3) = 0.4$ τοποθετείται στον B_2 κάδο, καθώς δεν ταιριάζει στον B_1.



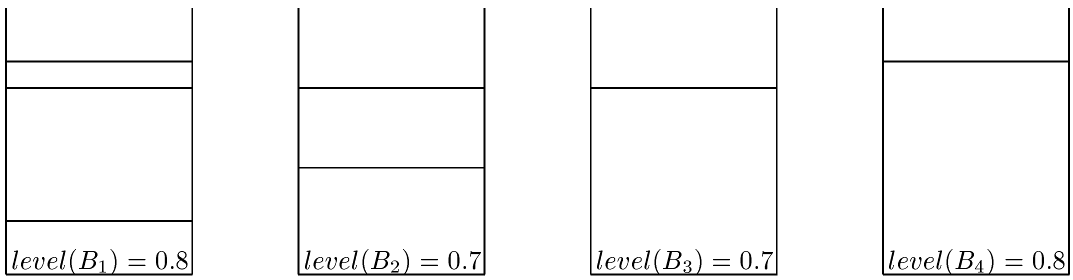
4ο ΒΗΜΑ Το $s(a_4) = 0.7$ τοποθετείται στον B_3 κάδο καθώς δεν ταιριάζει ούτε στον B_1, ούτε στον B_2.



5ο ΒΗΜΑ Το $s(a_5) = 0.1$ ταιριάζει στον B_1 κάδο και τοποθετείται σ' αυτόν.



6ο ΒΗΜΑ Το $s(a_6) = 0.3$ δεν ταιριάζει στον B_1, αλλά ταιριάζει στον B_2 και τοποθετείται σ' αυτόν.



7ο ΒΗΜΑ Το $s(a_7) = 0.8$ δεν ταιριάζει σε κανέναν από τους 3 πρώτους κάδους και τοποθετείται στον B_4.

Για τη συγκεκριμένη λίστα L, ο FF χρησιμοποιεί 4 κάδους.

Παρατηρούμε εδώ ότι αφαιρώντας τον περιορισμό *bounded-space*, χάνουμε το πλεονέκτημα(που είχε ο NF) να τρέχει σε γραμμικό χρόνο. Ο FF μπορεί να πακετάρει μία λίστα σε χρόνο $O(nlogn)$, αν χρησιμοποιήσουμε την κατάλληλη δομή δεδομένων. Όμως ο FF εξετάζοντας ένα ευρύτερο φάσμα από προορισμούς, έχει πολύ καλύτερη απόδοση.

Ακολουθεί ένα Λήμμα που μπορεί εύκολα να αποδειχτεί αλλά που δεν αποτελεί το στενότερο όριο για την απόδοση του FF.

Λήμμα 3.2.1 Ο *First Fit* χρησιμοποιεί το πολύ $2m^*$ κάδους.

Απόδειξη: Ας υποθέσουμε ότι έχουμε σαν είσοδο μία λίστα L, n αντικειμένων. Μετά την εφαρμογή του *First Fit* έστω ότι παίρνουμε σαν αποτέλεσμα m κάδους. Δηλαδή έστω ότι χρειάστηκαν m κάδοι $B_1, B_2, ..., B_m$ για την τοποθέτηση του συνόλου των αντικειμένων. Τέλος υποθέτουμε ότι το άθροισμα των αντικειμένων στον τελευταίο κάδο είναι x δηλαδή $level(B_m) = x$.

Αν $x \leq \frac{1}{2}$ τότε πρέπει για όλους τους προηγούμενους κάδους να ισχύει: $level(B_i) \geq 1 - x$, $\forall\ i\ \epsilon\ \{1{,}m-1\}$. Στην αντίθετη περίπτωση τα αντικείμενα του τελευταίου κάδου θα ταίριαζαν σε κάποιον από τους προηγούμενους κάδους που δεν θα ικανοποιούσε την παραπάνω σχέση. Επομένως έχουμε τα εξής:

$$\forall i, level(B_i) \geq 1-x \Rightarrow \forall i, level(B_i) \geq \frac{1}{2} \Rightarrow \sum_{i=1}^{m-1} level(B_i) \geq \frac{1}{2} \cdot (m-1) \quad (3.1)$$

Όμως $\sum_{i=1}^{m-1} level(B_i) = sum(L) - x$, όπου $sum(L)$ είναι το συνολικό αθροισμα των αντικειμένων της λίστας.

Άρα:
$\frac{1}{2} \cdot (m-1) \leq sum(L) - x \leq m^* - x < m^* \Rightarrow \frac{1}{2} \cdot (m-1) < m^* \Rightarrow m - 1 < 2m^* \Rightarrow m \leq 2m^*$

Αν $x > \frac{1}{2}$ τότε απο τους προηγούμενους κάδους υπάρχει το πολύ ένας, τέτοιος ώστε $level(B_i) \leq \frac{1}{2}$. Ας εξηγήσουμε όμως εδώ γιατί συμβαίνει αυτό. Αν υπήρχαν δυο ή περισσότεροι κάδοι με $level \leq \frac{1}{2}$ τότε αν οι πρώτοι δύο απο αυτούς ήταν οι B_i και B_j με $i < j$, ο *First Fit* θα είχε τοποθετήσει τα αντικείμενα του B_j στον B_i.

Αν δεν υπάρχει κανένας κάδος τέτοιος ώστε $level(B_i) \leq \frac{1}{2}$ τότε για όλους τους κάδους ισχύει $level(B_i) > \frac{1}{2}$ και καταλήγουμε εύκολα στο ότι $\frac{1}{2} \cdot m < m^*$. Αν υπάρχει ένας κάδος με $level \leq \frac{1}{2}$ τότε ακολουθούμε ακριβώς τα ίδια βήματα με την περίπτωση που $x \leq \frac{1}{2}$ και καταλήγουμε πάλι ότι $m \leq 2m^*$.

Θεώρημα 3.2.2 $[GGJ76]$ Για κάθε λίστα, $FF(L) \leq \lceil 1.7m^* \rceil$.

Το Θεώρημα αυτό, που δείχνει την ανωτερότητα του του FF έναντι του NF, στενεύει λίγο το όριο που είχε αποδειχτεί από τον $Ullman$ το 1971 $[Ull71]$

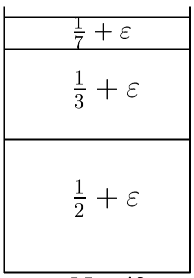
και ήταν $\frac{17}{10}OPT(L)$[1] $+3$. Ο πρόσθετος αυτός όρος (3) μειώθηκε αργότερα στο 2 [$JDU74$] και στη συνέχεια στο 1 [$GGJ76$]. Να σημειωθεί ότι μέχρι σήμερα δεν έχει βρεθεί στενότερο όριο.

Το άνω όριο του Θεωρήματος 2.2.2 είναι ασυμπτωτικά στενό, ώστε για αυθαίρετα μεγάλες τιμές του N να μπορούν να κατασκευαστούν λίστες L_N τέτοιες ώστε $FF(L_N) > \frac{17}{10}OPT(L_N) - 2$. Υπάρχουν απλά παραδείγματα από τέτοιες λίστες που δημιουργούν πολλά προβλήματα στον αλγόριθμο και είναι εύκολα κατασκευάσιμες.

Θα δούμε εδώ μία οικογένεια από λίστες, τέτοιες ώστε $\frac{FF(L)}{OPT(L)} = \frac{5}{3} = 1.\overline{666}$. Ένας λόγος που δεν αποτελούν την πιο ιδανική είσοδο για τον FF είναι ότι τα αντικείμενα τους είναι ταξινομημένα κατά αύξουσα σειρά μεγέθους. Συγκεκριμένα, οι λίστες αυτές είναι της μορφής:

$$L = \{\underbrace{\frac{1}{7} + \varepsilon \ldots \frac{1}{7} + \varepsilon}_{6N}, \underbrace{\frac{1}{3} + \varepsilon \ldots \frac{1}{3} + \varepsilon}_{6N}, \underbrace{\frac{1}{2} + \varepsilon \ldots \frac{1}{2} + \varepsilon}_{6N}\}$$
.

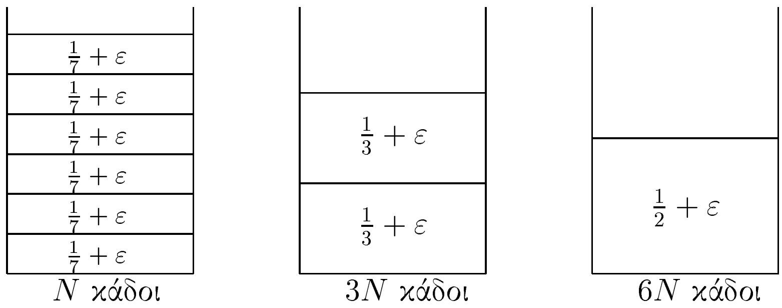
Το μεγεθός τους είναι $18N$ αντικείμενα, ενώ το ε παίρνει πολύ μικρές τιμές. Η *optimal* τοποθέτηση για τις λίστες αυτές είναι η εξής:



$6N$ κάδοι

Επομένως $OPT(L) = 6N$ κάδοι. Ο FF πραγματοποιεί την εξής τοποθέτηση:

[1] $OPT(L) \equiv m^*$



Άρα $FF(L) = 10N$ κάδοι. Επομένως όντως ισχύει $\frac{FF(L)}{OPT(L)} = \frac{5}{3} = 1.\overline{666}$.

Με τον παραπάνω τρόπο μπορούν να επινοηθούν λίστες που είναι ακόμα περισσότερο μη βολικές. Για παράδειγμα, υποθέτοντας ότι το ε είναι αρκετά μικρό, θα μπορούσε να συμπεριληφθεί ένα πρόσθετο αντικείμενο μεγέθους $\frac{1}{43} + \epsilon$ σε κάθε κάδο της βέλτιστης λύσης. Αν τοποθετηθούν N τέτοια αντικείμενα στην αρχή της λίστας, θα αναγκαστεί ο $First\ Fit$ να τα τοποθετήσει σε επιπλέον $\frac{N}{42}$ κάδους[2].

Ας γενικέψουμε την ιδέα και ας θεωρήσουμε την ακολουθία $t_1 = 2$, $t_{i+1} = t_i(t_i - 1) + 1$, για $i > 1$. Τότε για κάθε $k < \infty$ μπορούμε να κατασκευάσουμε παραδείγματα τέτοια ώστε οι βέλτιστοι κάδοι να περιέχουν ο καθένας k αντικείμενα, το καθένα από τα οποία θα έχει μέγεθος $\frac{1}{t_i} + \varepsilon$, με $1 \leq i \leq k$. Όταν δίνονται στον $First\ Fit$ αυτά τα αντικείμενα ταξινομημένα κατά μη φθίνουσα σειρά, θα απαιτηθούν κατά προσέγγιση $\sum_{i=1}^{k} \frac{1}{t_i - 1}$ κάδοι. Εκείνο που μπορούμε να συμπεράνουμε από αυτό είναι ότι:

$$R_{FF}^{\infty} \geq T_{\infty} \equiv \sum_{i=1}^{\infty} \frac{1}{t_i - 1} = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{42} + \frac{1}{1805} + \ldots \approx 1.69103$$

Η σχέση αυτή δεν αντικατοπτρίζει πλήρως τη *worst-case* συμπεριφορά του FF καθώς υπάρχουν πολύ πιο σύνθετα παραδείγματα στα οποία έχει ακόμα χαμηλότερη απόδοση. Να σημειωθεί εδώ ότι η σταθερά T_{∞} δεν είναι τυχαία καθώς σχετίζεται και με άλλους αλγόριθμους.

Όπως και στον $Next\ Fit$, η συμπεριφορά του $First\ Fit$ στις χειρότερες περιπτώσεις βελτιώνεται δραματικά όσο το μέγεθος του μεγαλύτερου αντικειμένου της εισόδου μειώνεται. Επιπλέον διατηρεί το πλεονέκτημά του απέναντι στο NF σε αυτές περιπτώσεις, παρ΄ όλο που το μέγεθος του πλεονεκτήματος αυτού εξαρτάται από την ακριβή τιμή του α[3] και στενεύει μαζί με το μέγεθος του

[2] εφόσον το N διαιρείται με το 42.
[3] το α δηλώνει το μέγεθος του μεγαλύτερου αντικειμένου της εισόδου.

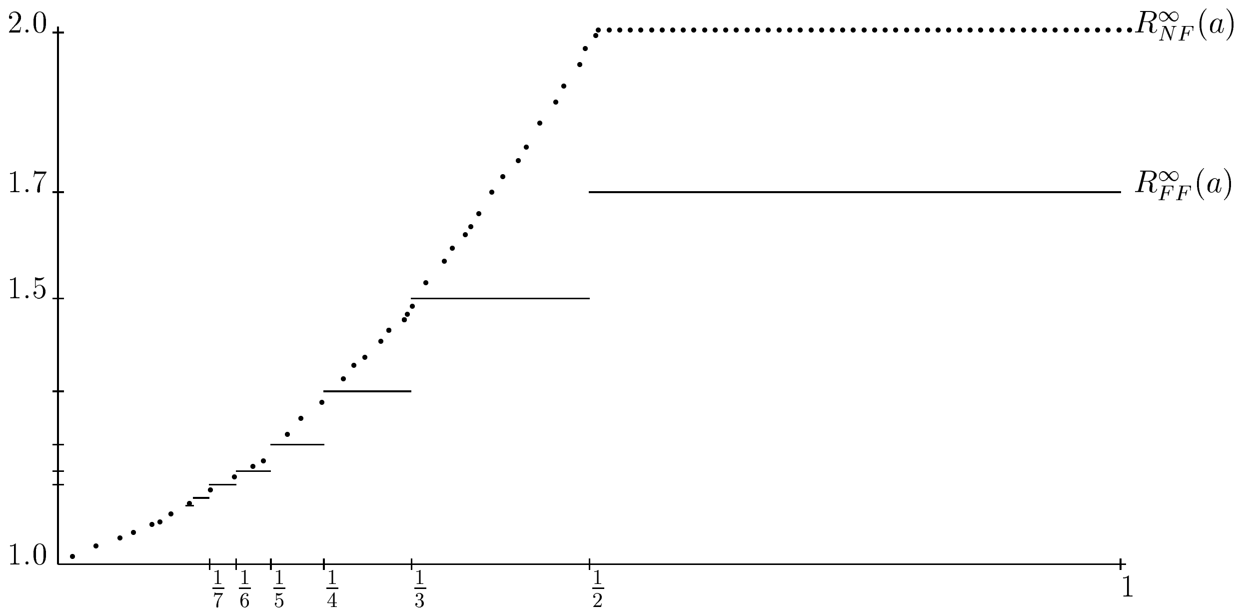
μεγαλύτερου αντικειμένου.

Θεώρημα 3.2.3 $[Joh73][JDU74]$ Ας υποθέσουμε ότι υπάρχει $m \epsilon Z$ τέτοιο ώστε $\frac{1}{m+1} < a \leq \frac{1}{m}$. Τότε:

A. Για $m = 1$, $R^{\infty}_{FF}(a) = \frac{17}{10}$

B. Για $m \geq 2$, $R^{\infty}_{FF}(a) = 1 + \frac{1}{m}$

Το παρακάτω σχήμα απεικονίζει τις $R^{\infty}_{NF}(a)$, $R^{\infty}_{FF}(a)$ σαν συναρτήσεις του α. Να σημειώσουμε ότι, παρ'ολο που η απόδοση του NF πλησιάζει αυτήν του FF κάθε φορά που το α πλησιάζει το $\frac{1}{t}$ των ακεραίων του Θεωρήματος, ποτέ δεν την πιάνει, δεδομένου ότι η $R^{\infty}_{FF}(a)$ είναι μία βηματική συνάρτηση που αλλάζει τιμή μόνο σ΄ αυτά τα σημεία, παραμένοντας σταθερή στα υπόλοιπα διαστήματα.

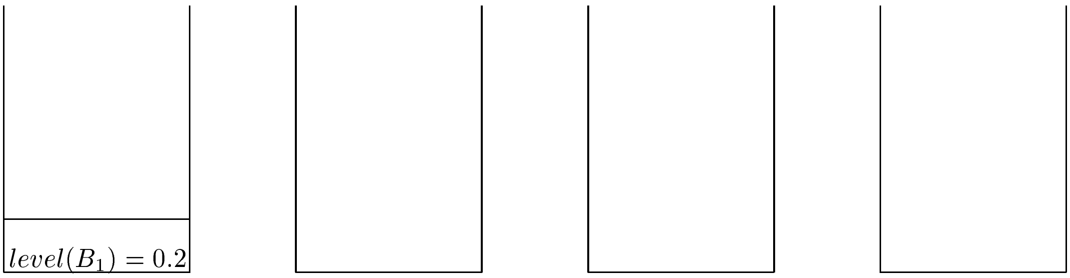


3.3 BEST FIT

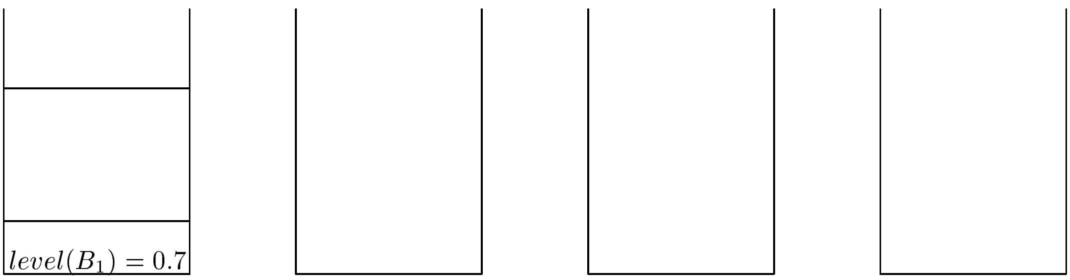
BF Packing Rule : Το πρώτο αντικείμενο της εισόδου τοποθετείται στον κάδο B_1. Στη συνέχεια το τρέχον αντικείμενο a_i τοποθετείται στον κάδο με το υψηλότερο δυνατό *level* στον οποίο ταιριάζει. Αν δεν ταιριάζει σε κανέναν τότε τοποθετείται σε νέο κάδο.

Η φιλοσοφία του αλγορίθμου $Best\ Fit(BF)$ είναι να τοποθετεί κάθε αντικείμενο στους όσο το δυνατόν πιο γεμάτους κάδους. Εξετάζει τα $level$ όλων των ανοικτών κάδων και βρίσκει αυτόν με το υψηλότερο(έστω B_j). Στη συνέχεια εξετάζει αν $s(a_i) \leq 1 - level(B_j)$ δηλαδή αν ταιριάζει το τρέχον a_i στον κάδο B_j. Α όχι τότε εξετάζει τον κάδο με το δεύτερο υψηλότερο $level$ και συνεχίζει έτσι φτάνοντας μέχρι και τον τελευταίο αν χρειαστεί. Σε περίπτωση που δύο κάδοι έχουν ίδιο $level$, θα προτιμηθεί αυτός που έχει το χαμηλότερο δείκτη.

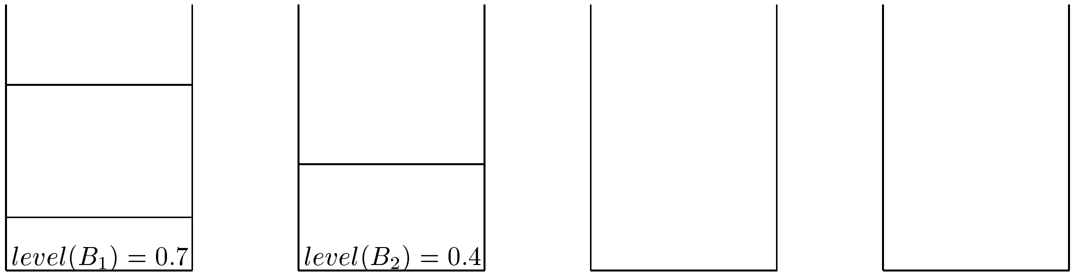
Παράδειγμα: Για τη λίστα $L = \{0.2,\ 0.5,\ 0.4,\ 0.7,\ 0.1,\ 0.3,\ 0.8\}$ με $OPT(L) = 3$ κάδοι, ο BF συμπεριφέρεται ως εξής:



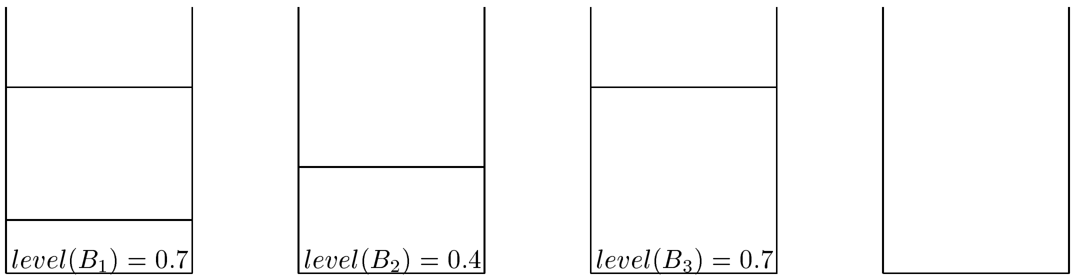
1ο ΒΗΜΑ Το $s(a_1) = 0.2$ τοποθετείται στον πρώτο κάδο.



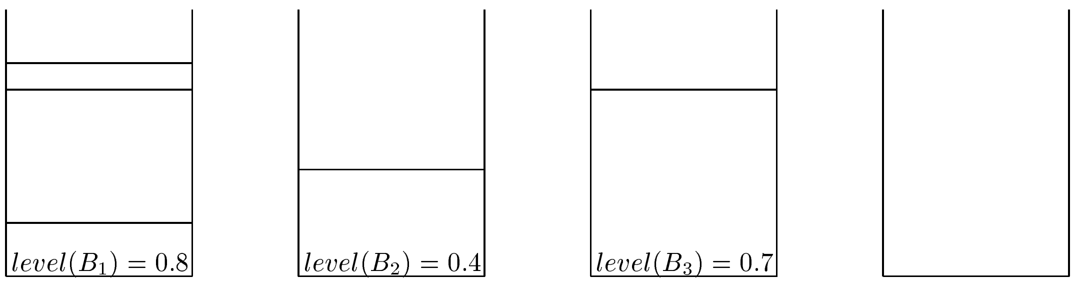
2ο ΒΗΜΑ Το $s(a_2) = 0.5$ τοποθετείται επίσης στον πρώτο κάδο.



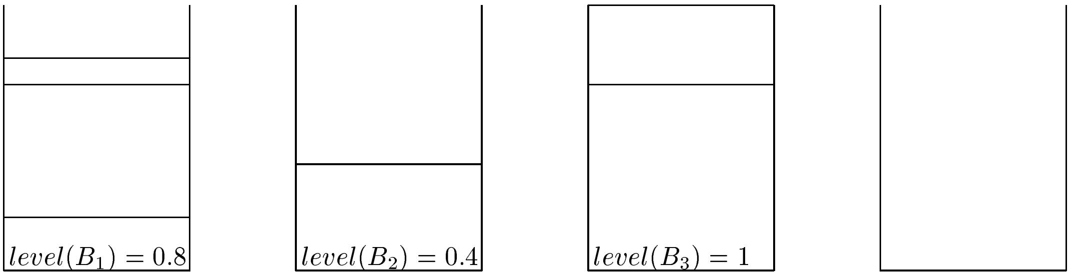
3ο ΒΗΜΑ Το $s(a_3) = 0.4$ τοποθετείται στον B_2 κάδο, καθώς δεν ταιριάζει στον B_1.



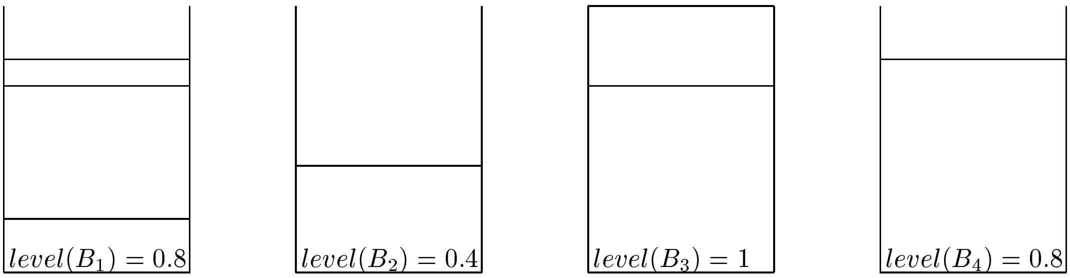
4ο ΒΗΜΑ Το $s(a_4) = 0.7$ τοποθετείται στον B_3 κάδο καθώς δεν ταιριάζει ούτε στον B_1 με το υψηλότερο $level$, ούτε στον B_2.



5ο ΒΗΜΑ Οι κάδοι με τα υψηλότερα $level$, είναι οι B_1 και B_3 στους οποίους ταιριάζει το $s(a_5) = 0.1$. Οι δύο αυτοί κάδοι έχουν το ίδιο $level$, αλλά το a_5 τοποθετείται στον B_1, επειδή έχει χαμηλότερο δείκτη.



6ο ΒΗΜΑ Το $s(a_6) = 0.3$ δεν ταιριάζει στον B_1 με το υψηλότερο $level$, αλλά ταιριάζει στον B_3 με το δεύτερο υψηλότερο και τοποθετείται σ' αυτόν. Ο B_3 κλείνει, καθώς είναι γεμάτος.



7ο ΒΗΜΑ Το $s(a_7) = 0.8$ δεν ταιριάζει σε κανέναν από τους 3 πρώτους κάδους και τοποθετείται στον B_4.

Για τη συγκεκριμένη λίστα L, ο BF χρησιμοποιεί 4 κάδους.

Ο $Best\ Fit$ έχει αρκετές ομοιότητες στη συμπεριφορά του με τον $First\ Fit$. Για παράδειγμα ο BF τρέχει επίσης σε χρόνο $O(nlogn)$. Οι BF και FF μπορούν να δώσουν θαυμάσιες λύσεις για μεμονωμένες λίστες. Υπάρχουν παραδείγματα με λίστες L[4] για τις οποίες προκύπτει ότι $BF(L) = \frac{4}{3}FF(L)$ και υπάρχουν και άλλες για τις οποίες ισχύει $FF(L) = \frac{3}{2}BF(L)$ $[Joh73]$. Εντούτοις όλα τα αποτελέσματα που αναφέρονται στο $R^{\infty}_{FF}(a)$,και για τα οποία μιλήσαμε στο προηγούμενο κεφάλαιο, ισχύουν και για το $R^{\infty}_{BF}(a)$ $[Joh73][Joh74][JDU74]$.

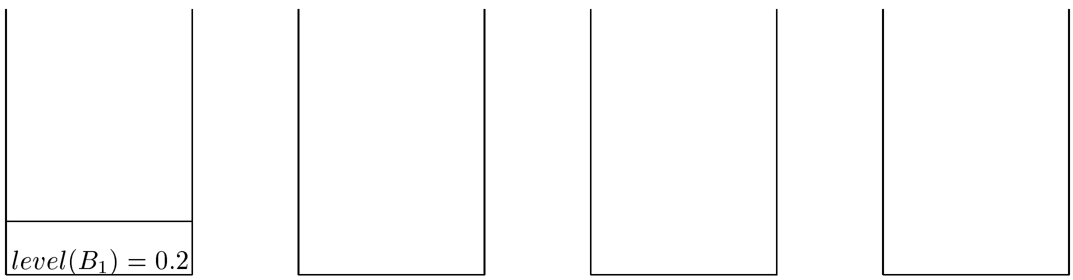
[4] Οι λίστες αυτές ωθούν τον $OPT(L)$ σε αυθαίρετα μεγάλες τιμές.

3.4 WORST FIT

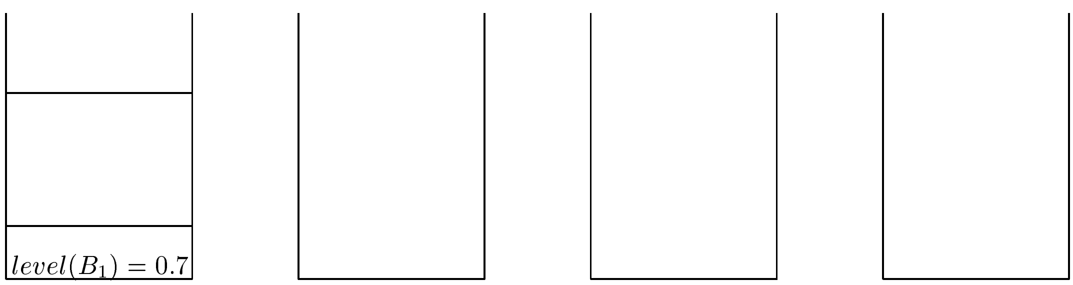
WF Packing Rule : Το τρέχον αντικείμενο a_i τοποθετείται στον ανοιχτό εκείνο κάδο με το χαμηλότερο *level*. Αν δεν ταιριάζει σ′ αυτόν ανοίγει νέος κάδος και τοποθετείται εκεί.

Όπως στον BF, έτσι και στον $Worst\ Fit(WF)$ αν έχουμε δύο κάδους οι οποίοι έχουν το ίδιο *level* και αυτό είναι το χαμηλότερο που υπάρχει τότε θα προτιμηθεί εκείνος με το χαμηλότερο δείκτη.

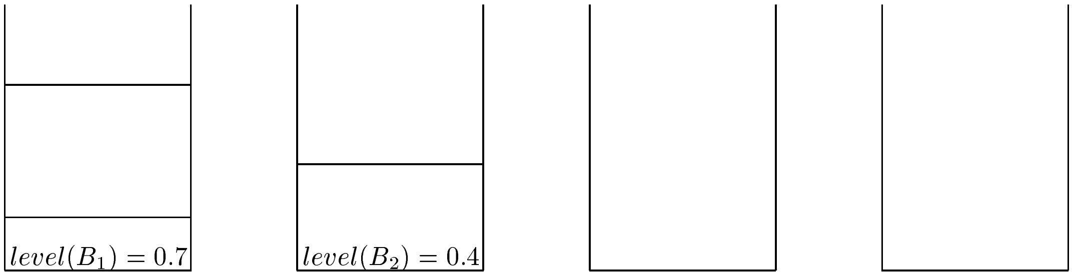
Παράδειγμα: Για τη λίστα $L = \{0.2,\ 0.5,\ 0.4,\ 0.7,\ 0.1,\ 0.3,\ 0.8\}$ με $OPT(L) = 3$ κάδοι, ο WF συμπεριφέρεται ως εξής:



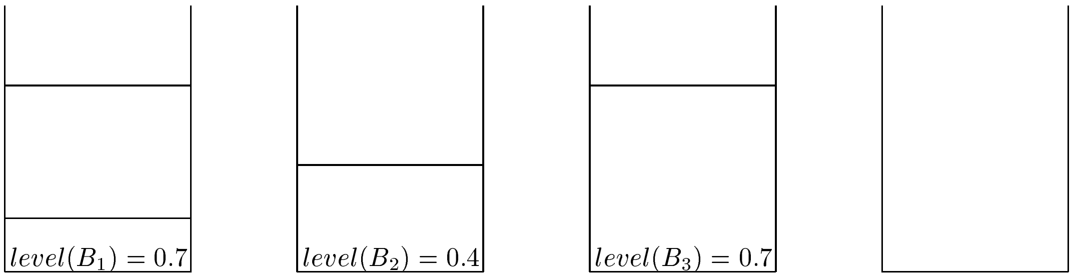
1ο ΒΗΜΑ Το $s(a_1) = 0.2$ τοποθετείται στον πρώτο κάδο.



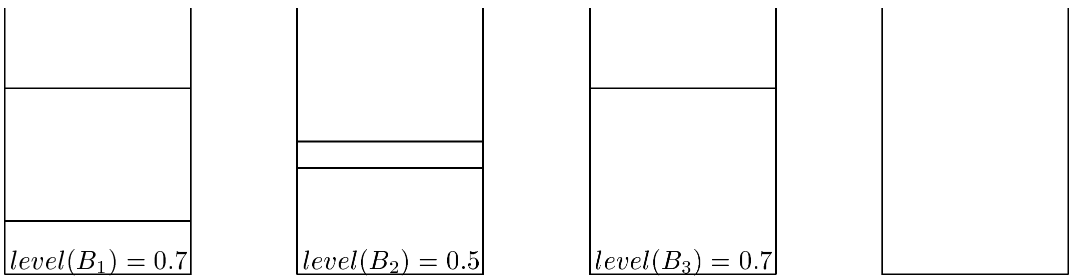
2ο ΒΗΜΑ Το $s(a_2) = 0.5$ τοποθετείται επίσης στον πρώτο κάδο.



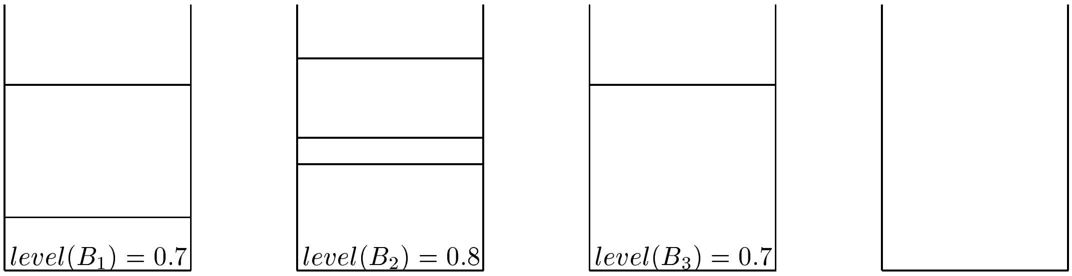
3ο ΒΗΜΑ Το $s(a_3) = 0.4$ τοποθετείται στον B_2 κάδο, καθώς δεν ταιριάζει στον B_1.



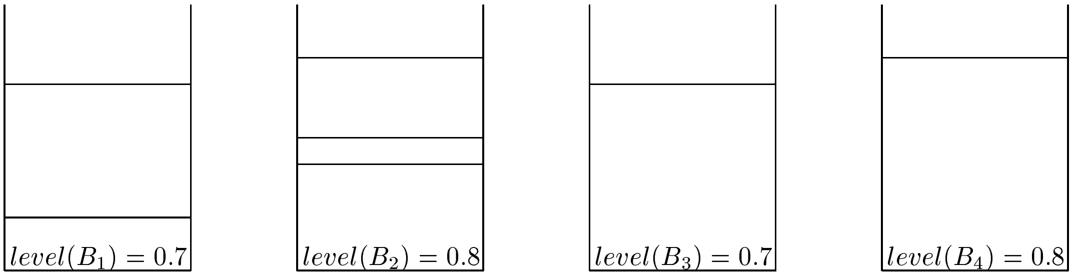
4ο ΒΗΜΑ Το $s(a_4) = 0.7$ τοποθετείται στον B_3 κάδο καθώς δεν ταιριάζει ούτε στον B_2 με το χαμηλότερο *level*, ούτε στον B_1.



5ο ΒΗΜΑ Το $s(a_5) = 0.1$ τοποθετείται στον B_2, ο οποίος έχει το χαμηλότερο *level*. $level(B_1) = 0.7$, $level(B_2) = 0.5$, $level(B_3) = 0.7$



6ο ΒΗΜΑ Το $s(a_6) = 0.3$ τοποθετείται στον B_2 με το χαμηλότερο $level$.



7ο ΒΗΜΑ Το $s(a_7) = 0.8$ δεν ταιριάζει στους B_1, B_3, οι οποίοι έχουν το χαμηλότερο $level$, αλλά ούτε και στον B_2. Επομένως τοποθετείται στον B_4.

Για τη συγκεκριμένη λίστα L, ο WF χρησιμοποιεί 4 κάδους.

Θεώρημα 3.4.1

Για $a \leq \frac{1}{2}$, $R^{\infty}_{WF}(a) = \frac{1}{1-a}$

Για $a \geq \frac{1}{2}$, $R^{\infty}_{WF}(a) = 2$

Παρατηρούμε ότι ο WF έχει την ίδια απόδοση με τον NF κάτι που σημαίνει ότι δεν είναι ιδιαίτερα αποτελεσματικός. Επίσης έχει το ελλάτωμα ότι σε πολλές περιπτώσεις δεν μπορεί να κλείσει κανέναν κάδο. Κι αυτό συμβαίνει γιατί ο ορισμός του ο ίδιος του καθιστά δύσκολο να γεμίσει τους κάδους.

3.5 ALMOST WORST FIT

O $Worst\ Fit$ δεν έχει την καλύτερη δυνατή απόδοση, ωστόσο με μία παραλλαγή στον ορισμό του μπορούμε να κατασκευάσουμε έναν άλλο αλγόριθμο πολύ πιο αποτελεσματικό. Αυτός δεν είναι άλλος από τον $Almost\ Worst\ Fit$.

AWF Packing Rule : Το τρέχον αντικείμενο a_i τοποθετείται στον κάδο με το δεύτερο χαμηλότερο $level$[5], εκτός αν υπάρχει μόνο ένας κάδος στον οποίο να ταιριάζει οπότε θα εισέλθει σε αυτόν. Αν το a_i δεν ταιριάζει σε κανένα ανοικτό κάδο τότε θα ανοίξει ένας καινούργιος.

Η χειρότερη επίδοση του AWF είναι εξίσου καλή όσο των $First$ και $Best\ Fit$. Ωστόσο μπορούν να υπάρξουν σημαντικές διαφοροποιήσεις για μεμονωμένες λίστες. Υπάρχουν παραδείγματα για τα οποία ισχύει $AWF(L) = \frac{5}{4}FF(L)$ και άλλα για τα οποία $FF(L) = \frac{9}{8}AWF(L)$ $[Joh73]$.

3.6 ANY FIT ΑΛΓΟΡΙΘΜΟΙ

Θα ονομάσουμε έναν $online\ bin\ packing$ αλγόριθμο σαν $Any\ Fit(AF)$, εάν ανοίγει ένα καινούργιο κάδο μόνο όταν το αντικείμενο που πρέπει να τοποθετηθεί δεν ταιριάζει σε κανέναν χρησιμοποιημένο κάδο.

Ένας αντίστοιχος αλγόριθμος θα ονομάζεται $Almost\ Any\ Fit(AAF)$, εάν δεν πακετάρει ποτέ ένα αντικείμενο στον μερικώς γεμισμένο κάδο με το χαμηλότερο επίπεδο εκτός αν υπάρχουν περισσότεροι του ενός τέτοιοι κάδοι ή αυτός ο κάδος είναι ο μοναδικός που έχει επαρκή χώρο.

Από τους παραπάνω ορισμούς είναι φανερό ότι οι $First\ Fit$, $Best\ Fit$, $Worst\ Fit$ και $Almost\ Worst\ Fit$ είναι $Any\ Fit$ αλγόριθμοι. Ο $Next\ Fit$ δεν ανήκει σ' αυτήν την κατηγορία καθώς ο ορισμός[6] του τον εμποδίζει να εξετάσει όλους τους χρησιμοποιημένους κάδους γιατί κρατάει μόνο έναν από αυτούς ανοικτό. Οι $Best\ Fit$ και $Almost\ Worst\ Fit$ ανήκουν επίσης στη κατηγορία των $Almost\ Any\ Fit$ αλγόριθμων.

Για αυτές τις παραπάνω δύο κατηγορίες ισχύει το εξής Θεώρημα $[Joh73][Joh74]$:

[5] Για ισότητα σ' εκείνον με το χαμηλότερο δείκτη.

[6] Ο ορισμός του περιέχει 1-*bounded space* περιορισμό

Θεώρημα 3.6.1 Για όλα τα α, τέτοια ώστε $0 < a \leq 1$ ισχύει:

Α. Αν ο Α είναι ένας AF αλγόριθμος, τότε $R^{\infty}_{FF}(a) \leq R^{\infty}_{A}(a) \leq R^{\infty}_{NF}(a)$

Β. Αν ο Α είναι ένας AAF αλγόριθμος, τότε $R^{\infty}_{A}(a) = R^{\infty}_{FF}(a)$

Εδώ φαίνεται και ο λόγος που κανένας από τους αλγορίθμους που έχουμε μελετήσει μέχρι τώρα δεν ξεπέρασε την απόδοση του *First Fit*. Εφ' όσον σχεδόν όλοι είναι *Any Fit* δεν μπορούν από τον ορισμό τους να το καταφέρουν αυτό. Μόνο δύο από αυτούς απλά έφτασαν την απόδοση του FF. Αυτοί είναι οι BF και AWF οι οποίοι είναι *Almost Any Fit* αλγόριθμοι.

Το συμπέρασμα που βγαίνει είναι ότι εάν θέλουμε να λάβουμε καλύτερη *worst-case* συμπεριφορά από αυτήν του *First Fit* θα πρέπει να εγκαταλείψουμε τον περιορισμό *Any Fit* που ορίσαμε παραπάνω ή τον ίδιο τον *online* περιορισμό.

3.7 BOUNDED-SPACE ΑΛΓΟΡΙΘΜΟΙ

Το χαρακτηριστικό του περιορισμού *Any Fit* είναι ότι κανένας κάδος δεν μπορεί να κλείσει μόνιμα για περαιτέρω αντικείμενα. Συνεπώς κανένας *Any Fit* αλγόριθμος δεν μπορεί ταυτόχρονα να είναι και *bounded-space* αλγόριθμος. Είδαμε ότι ο περιορισμός *bounded-space* λέει ότι μόνο ορισμένοι κάδοι (των οποίων καθορίζεται ο αριθμός) είναι ανοικτοί κάθε φορά να δέχονται αντικείμενα.

Έχουμε ήδη δει έναν *bounded-space online* αλγόριθμο, τον *Next Fit*, ο οποίος πληρώνει γι' αυτόν τον περιορισμό στη χειρότερη επίδοση του, σε σχέση με τους *unbounded-space* αλγόριθμους *First* και *Best Fit*. Μάλιστα ο NF είναι 1-*bounded space*[7] δηλαδή έχει τον μεγαλύτερο δυνατό περιορισμό για αυτό το είδος των αλγορίθμων.

Έχουμε πει σε προηγούμενο κεφάλαιο ότι οι *bounded-space* αλγόριθμοι εκτός απο τον *packing rule* ορίζονται κι από έναν *closing rule*, ο οποίος καθορίζει ποιός κάδος θα κλείσει κάθε φορά που ανοίγει ένας καινούργιος, ώστε να έχουμε συνέχεια σταθερό αριθμό ανοικτών κάδων. Παρακάτω θα αναφερθούμε αναλυτικά σε αρκετούς *bounded-space* αλγόριθμους οι οποίοι παρουσιάζουν εξαιρετικό ενδιαφέρον.

[7] Ένας ανοικτός κάδος καθε φορά.

3.7.1 ΟΙ $\mathbf{NF_K}$, $\mathbf{BBF_K}$, $\mathbf{ABF_K}$ ΚΑΙ $\mathbf{AFB_K}$ ΑΛΓΟΡΙΘΜΟΙ

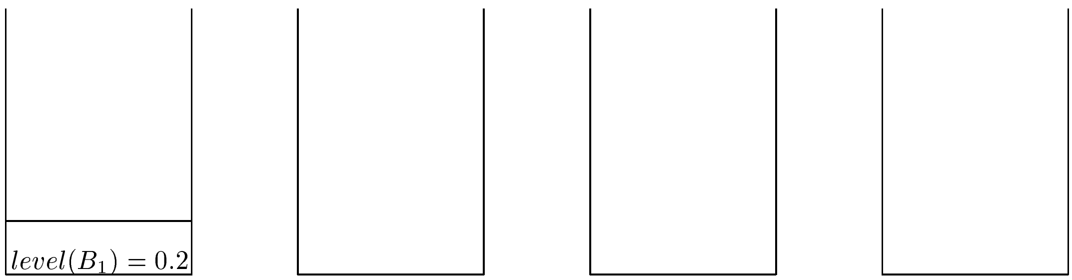
Σε προηγούμενα κεφάλαια ορίσαμε τους *packing rules* του *First Fit* και του *Best Fit*. Εδώ θα ορίσουμε τους αντίστοιχους *closing rules*.

First Fit Closing Rule : Κάθε φορά που πρέπει να ανοίξει ένας καινούργος κάδος, κλείνει εκείνος από τους ανοικτούς κάδους ο οποίος έχει το χαμηλότερο δείκτη.

Best Fit Closing Rule : Κάθε φορά που πρέπει να ανοίξει ένας καινούργος κάδος, κλείνει εκείνος από τους ανοικτούς κάδους ο οποίος έχει το υψηλότερο *level*[8].

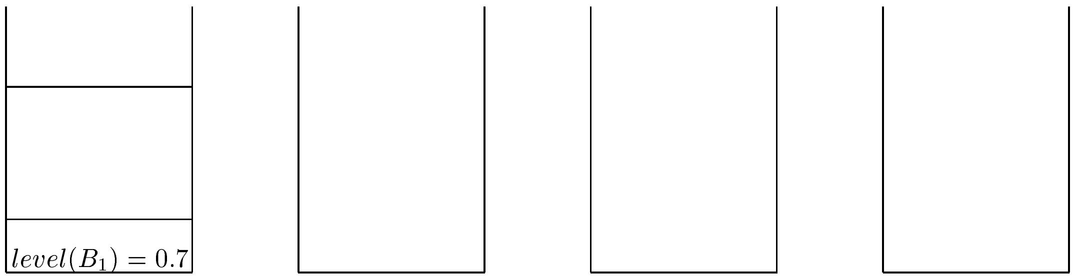
Ο *Next-K Fit* αλγόριθμος (NF_K) συνδυάζει τον *packing* και τον *closing rule* του *First Fit*. Ο *K-Bounded Best Fit* (BBF_K) συνδυάζει τους αντίστοιχους κανόνες του *Best Fit*. Επίσης υπάρχουν και οι ABF_K και AFB_K. Ο πρώτος από αυτούς συνδυάζει τον *packing rule* του BF και τον *closing rule* του FF και αναλύθηκε αρχικά με το όνομα *Best-K Fit*. Ο δεύτερος (AFB_K) χρησιμοποιεί τον *packing rule* του FF και τον *closing rule* του BF. Και οι τέσσερις αλγόριθμοι συμπεριφέρονται όπως ο NF για $K = 1$ και όλοι μπορούν να τρέξουν σε γραμμικό χρόνο για οποιοδήποτε K.

$\mathbf{NF_2}$ Παράδειγμα: Θα δούμε σ' αυτό το σημείο ένα παράδειγμα για τον NF_2, ώστε να γίνει κατανοητός και πρακτικά ο *bounded-space* περιορισμός. Για τον NF_2, οι ανοικτοί κάδοι δεν πρέπει ποτέ να ξεπερνούν τους 2. Σαν είσοδο θα έχουμε την λίστα $L = \{0.2,\ 0.5,\ 0.4,\ 0.7,\ 0.1,\ 0.3,\ 0.8\}$ με $OPT(L) = 3$ κάδοι.

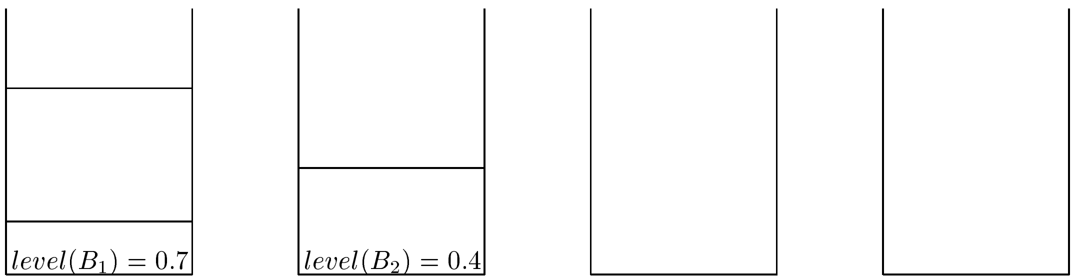


[8] Σε περίπτωση που έχουμε δύο κάδους με το ίδιο *level*, κλείνει εκείνος με τον χαμηλότερο δείκτη.

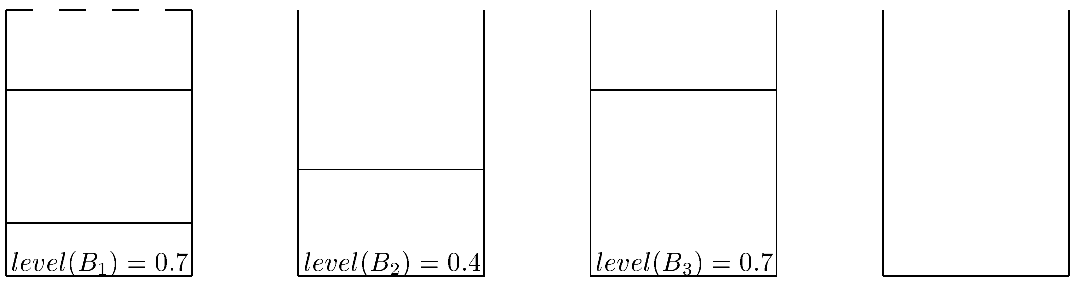
1o ΒΗΜΑ Το $s(a_1) = 0.2$ τοποθετείται στον πρώτο κάδο.



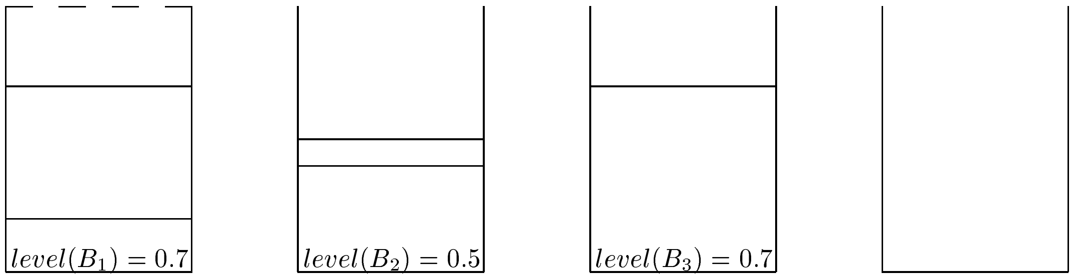
2o ΒΗΜΑ Το $s(a_2) = 0.5$ τοποθετείται επίσης στον πρώτο κάδο.



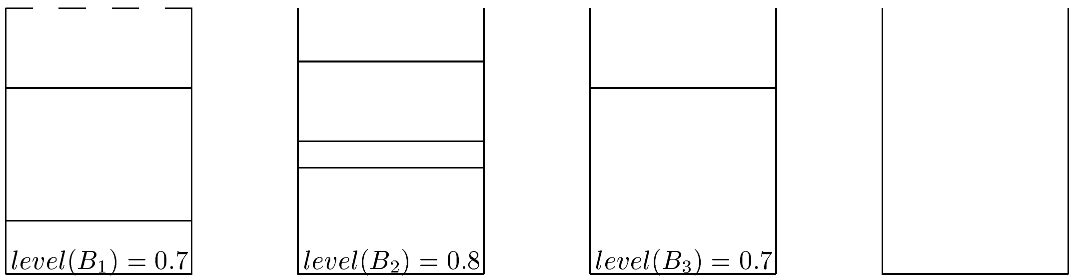
3o ΒΗΜΑ Το $s(a_3) = 0.4$ τοποθετείται στον B_2 κάδο καθώς δεν ταιριάζει στον B_1.



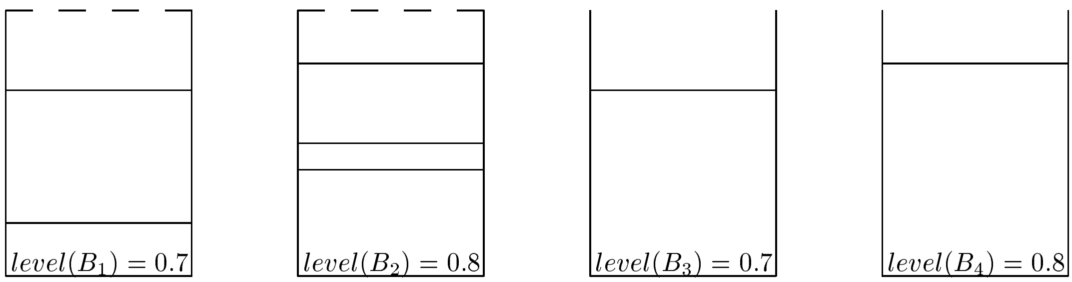
4ο ΒΗΜΑ Το $s(a_4) = 0.7$ τοποθετείται στον B_3 κάδο καθώς δεν ταιριάζει ούτε στον B_1, ούτε στον B_2. Επομένως άνοιξε ένας νέος κάδος, ο B_3. Έτσι, ο B_1 κλείνει μόνιμα σύμφωνα με τον *packing rule* του FF ώστε να έχουμε 2 ανοικτούς κάδους.



5ο ΒΗΜΑ Το $s(a_5) = 0.1$ τοποθετείται στον B_2 κάδο καθώς είναι ο πρώτος κάδος που εξετάζεται σύμφωνα με τον FF και ταιριάζει σ' αυτόν.



6ο ΒΗΜΑ Το $s(a_6) = 0.3$ τοποθετείται επίσης στον B_2 κάδο.



7ο ΒΗΜΑ Το $s(a_7) = 0.8$ δεν ταιριάζει στον B_2 αλλά ούτε και στον B_3. Έτσι τοποθετείται στον B_4. Ο B_2 κλείνει μόνιμα σύμφωνα με τον 2-*bounded-space* περιορισμό.

Για τη συγκεκριμένη λίστα L, ο NF_2 χρησιμοποιεί 4 κάδους.

Το επόμενο θεώρημα μας ενημερώνει σχετικά με τα ασυμπτωτικά προσεγγιστικά πηλίκα χειρότερης επίδοσης των παραπάνω τεσσάρων αλγόριθμων, για $K \geq 2$ $[CI89][Mao93b]$ $[Mao93a][Zha94][CJ91]$:

Θεώρημα 3.7.1.1 Για $K \geq 2$ έχουμε:

A. $R^{\infty}_{NF_K} = R^{\infty}_{AFB_K} = 1.7 + \frac{3}{10(K-1)}$

B. $R^{\infty}_{ABF_K} = 1.7 + \frac{3}{10K}$

C. $R^{\infty}_{BBF_K} = 1.7$

Οι παραπάνω τύποι αποδείχτηκαν κατά καιρούς από διαφορετικά πρόσωπα όπως οι $Crisik$, Mao, $Zhang$ και άλλοι. Απ΄ αυτούς μπορούμε να βγάλουμε αρκετά ενδιαφέροντα συμπεράσματα.

Κανένας απο τους τέσσερεις αλγόριθμους δεν σπάει το φράγμα του 1.7. Αυτό ήταν κατά κάποιο τρόπο αναμενόμενο καθώς θα ήταν αδύνατο να ξεπεράσουν σε απόδοση τους BF και FF λόγω του *bounded-space* περιορισμού τους. Επίσης βλέπουμε ότι ο ABF_K έχει καλύτερη απόδοση από τους NF_K και AFB_K καθώς $\frac{3}{10K} < \frac{3}{10(K-1)}$. Παρατηρούμε ότι καθώς το Κ αυξάνεται, η απόδοση και των τριών παραπάνω αλγορίθμων βελτιώνεται. Αυτό συμβαίνει για όλους τους *bounded-space* αλγόριθμους καθώς όσο μεγαλύτερο είναι το Κ, τόσο περισσότεροι κάδοι είναι ανοικτοί κάθε φορά να δεκτούν το επόμενο αντικείμενο. Να σημειωθεί ότι όταν το Κ τείνει στο ∞ και οι τρείς παραπάνω αλγόριθμοι τείνουν να έχουν απόδοση στο 1.7. Ουσιαστικά όμως όταν $K \to \infty$ καταργείται ο *bounded-space* περιορισμός και καταλήγουμε στους αντίστοιχους *unbounded-space* αλγόριθμους[9].

Το εντυπωσιακό όμως στο παραπάνω Θεώρημα είναι ότι ο BBF_K έχει σταθερή απόδοση για $K \geq 2$ και προσφέρει τις ίδιες εγγυήσεις στην εκτέλεση

[9] Ουσιαστικά καταργείται ο *closing rule* και παραμένει σε ισχύ μόνο ο *packing rule*.

του με αυτές των $First\ Fit$ και $Best\ Fit$.

Συνοψίζοντας, από την παραπάνω τετράδα καλύτερος σε απόδοση είναι ο BBF_K, ακολουθεί ο ABF_K και τελευταίοι έρχονται οι NF_K και AFB_K.

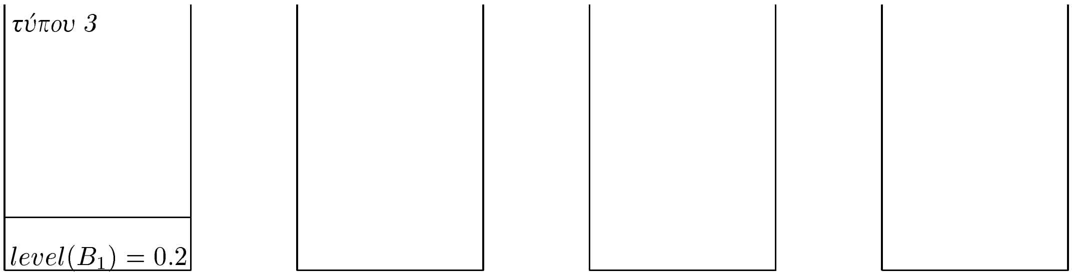
3.7.2 ΟΙ $\mathbf{Harmonic_K}$ ΑΛΓΟΡΙΘΜΟΙ

Θα αναφερθούμε σ' αυτό το κεφάλαιο σε μία σειρά από *bounded-space* αλγόριθμους, τους οποίους συνέστησε ο *Lee* και ονομάζονται $Harmonic_K(H_K)$ αλγόριθμοι. Η δομή τους είναι αρκετά πολύπλοκη αλλά η απόδοσή τους είναι εξαιρετικά καλή.

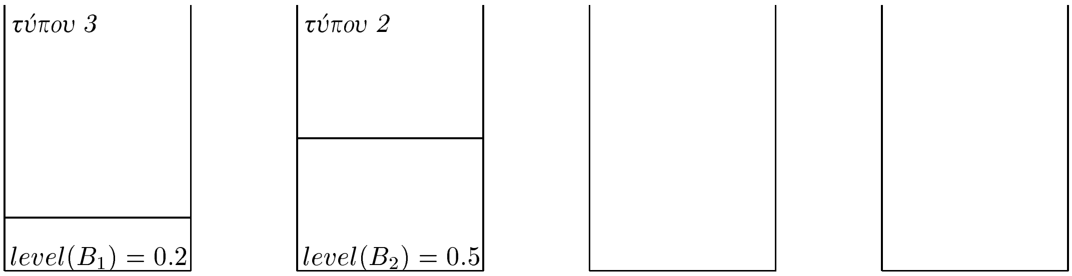
$\mathbf{H_K}$ Packing Rule : Το διάστημα από 0 έως 1 διαιρείται σε K υποδιαστήματα I_j, $1 \leq j \leq K$, όπου $I_j = (\frac{1}{j+1}, \frac{1}{j}]$, $1 \leq j < K$ και $I_K = (0, \frac{1}{K}]$. Ένα αντικείμενο α θα έχει τύπο j εάν $s(a)\ \epsilon\ I_j$. Παρόμοια οι κάδοι θα διαιρεθούν σε K τύπους, έτσι ώστε κάδοι του τύπου j να δέχονται μόνο αντικείμενα του ίδιου τύπου, και μόνο ένας κάδος από κάθε τύπο να μπορεί να είναι ανοικτός κάθε στιγμή. Έτσι το τρέχον αντικείμενο α θα πακετάρεται ως εξής: Ας υποθέσουμε ότι ο τύπος του συγκεκριμένου αντικειμένου είναι j. Αν υπάρχει ανοικτός κάδος τύπου j και έχει χώρο για να δεχτεί το α, τότε θα τοποθετηθεί εκεί. Αλλιώς κλείνουμε τον κάδο αυτό, εφόσον υπάρχει, και ανοίγουμε ένα καινούργιο κάδο τύπου j, όπου και θα εισέλθει το α.

Οι $Harmonic_K$ είναι οικογένεια αλγορίθμων και όχι ένας αλγόριθμος γιατί για κάθε διαφορετικό K έχουμε και διαφορετική περίπτωση.

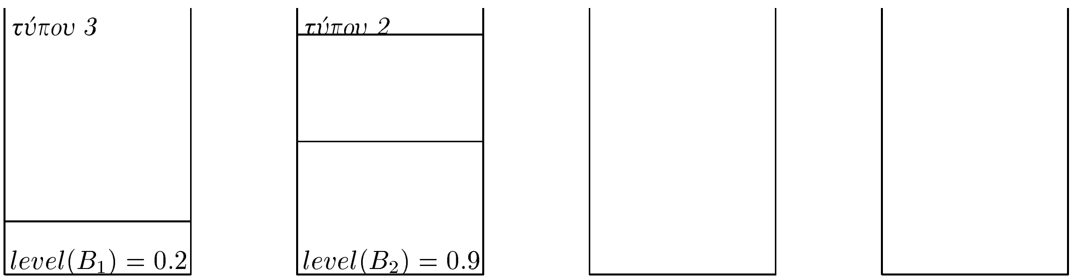
Παράδειγμα: Θα αναφέρουμε σαν παράδειγμα τον $Harmonic_3$, ο οποίος είναι ένας από τους πιο απλούς αλγόριθμους της κατηγορίας. Εδω έχουμε 3 υποδιαστήματα είναι τα εξής: $I_1 = (\frac{1}{2}, 1]$, το $I_2 = (\frac{1}{3}, \frac{1}{2}]$ και το $I_3 = (0, \frac{1}{3}]$. Έτσι, θα έχουμε 3 τύπους αντικειμένων και 3 τύπους κάδων. Ο $Harmonic_3$ είναι προφανώς ένας 3-*bounded-space* αλγόριθμος. Η λίστα εισόδου είναι η γνωστή λίστα $L = \{0.2,\ 0.5,\ 0.4,\ 0.7,\ 0.1,\ 0.3,\ 0.8\}$.



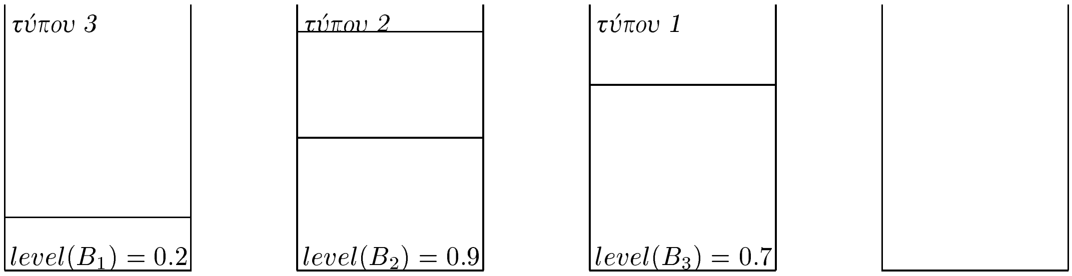
1ο ΒΗΜΑ Το $s(a_1) = 0.2 \in I_3$. Άρα το a_1 είναι τύπου 3. Τοποθετείται στον πρώτο κάδο, ο οποίος αυτόματα γίνεται τύπου 3.



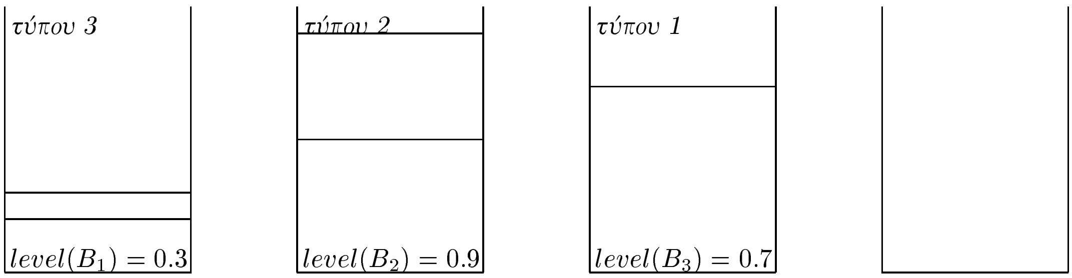
2ο ΒΗΜΑ Το $s(a_2) = 0.5 \in I_2$. Άρα το a_2 είναι τύπου 2. Επομένως δεν μπορεί να τοποθετηθεί στον B_1 που είναι τύπου 3. Τοποθετείται στον δεύτερο κάδο, ο οποίος γίνεται τύπου 2.



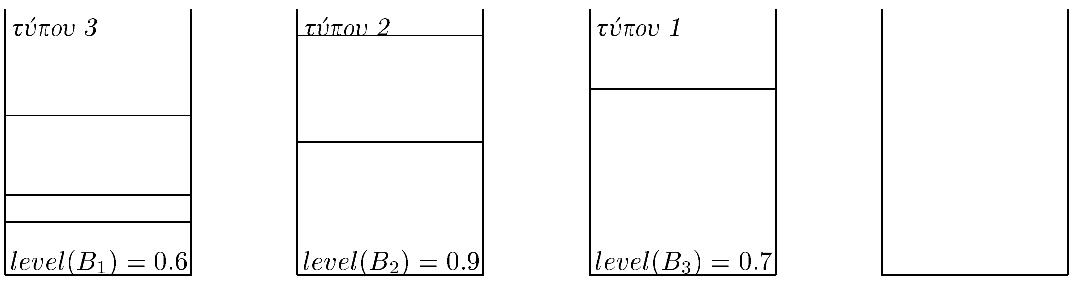
3ο ΒΗΜΑ Το $s(a_3) = 0.4 \in I_2$. Άρα το a_3 είναι επίσης τύπου 2. Επομένως, εφ' όσον ταιριάζει στον B_2 που είναι του ίδιου τύπου, τοποθετείται σ' αυτόν.



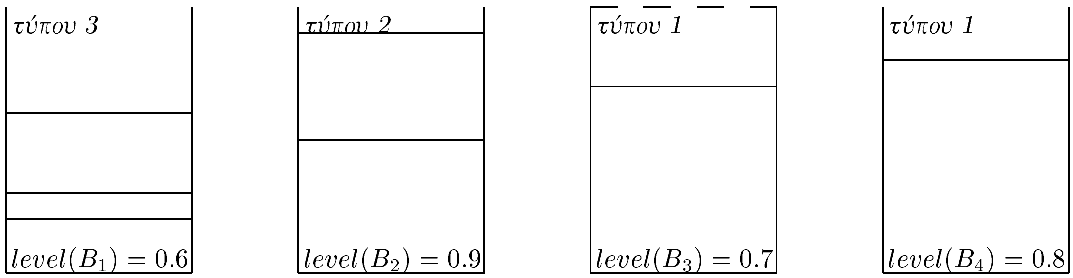
4ο ΒΗΜΑ Το $s(a_4) = 0.7 \in I_1$. Άρα το a_4 είναι τύπου 1. Δεν υπάρχει ανοικτός κάδος αυτού του τύπου, επομένως θα ανοίξει ο B_3 στον οποίο τοποθετείται το a_4. Ο B_3 γίνεται τύπου 1.



5ο ΒΗΜΑ Το $s(a_5) = 0.1 \in I_3$. Άρα το a_5 είναι τύπου 3. Ταιριάζει στον ανοικτό κάδο(B_1) τύπου 3 και τοποθετείται σ' αυτόν.



6ο ΒΗΜΑ Το $s(a_6) = 0.3 \in I_3$. Επομένως και το a_6 είναι τύπου 3. Ταιριάζει στον B_1 και τοποθετείται σ' αυτόν.



7ο ΒΗΜΑ Το $s(a_7) = 0.8 \in I_1$. Επομένως το a_7 είναι τύπου 1. Ο κάδος B_3 είναι επίσης τύπου 1. Όμως το a_7 δεν ταιριάζει σ' αυτόν. Άρα πρέπει να κλείσει μόνιμα ο B_3 και να ανοίξει ένας νέος κάδος τύπου 1. Αυτός θα είναι ο B_4 όπου θα τοποθετηθεί και το a_7.

Για τη συγκεκριμένη λίστα L, ο H_3 χρησιμοποιεί 4 κάδους.

Η χειρότερη επίδοση των $Harmonic_K$ συνδέεται άμεσα με την ακολουθία[10] $k_1 = 1$, $k_{i+1} = k_i(k_i + 1)$ για $i > 1$. Αν $t_j = \sum_{i=1}^{j} \frac{1}{k_i}$ τότε $t_1 = 1$, $t_2 = \frac{3}{2}$, $t_3 = \frac{5}{3}$, $t_4 = 1,6904\ldots$ και $t_\infty = T_\infty = lim_{j\to\infty} t_j = 1,69103\ldots$. Η T_∞ είναι η γνωστή σταθερά που συναντήσαμε και και σε προηγούμενα κεφάλαια.

Θεώρημα 3.7.2.1 [$CJ01$] Αν $k_i < K \leq k_{i+1}$ για κάποια $i \geq 1$, τότε: $t_i + \frac{1}{k_{i+1}-1} \leq R^{\infty}_{H_K} \leq t_i + \frac{K}{k_{i+1}(K-1)}$

Τα όρια του $R^{\infty}_{H_K}$ συμπίπτουν σε μία τιμή για $K = k_{i+1}$. Έτσι για συγκεκριμένα K, το $R^{\infty}_{H_K}$ είναι γνωστο και ίσο με $t_i + \frac{1}{k_{i+1}-1}$. Όμως και για $K = k_{i+1} + 1$ μπορούμε να βρούμε ακριβείς αποδόσεις, καθώς έχουν εφευρεθεί τα κατάλληλα παραδείγματα που δείχνουν ότι και σ' αυτές τις περιπτώσεις μπορούμε να βρούμε την τιμή του $R^{\infty}_{H_K}$. Επομένως για να υπολογίσουμε το $R^{\infty}_{H_K}$ με ακρίβεια πρέπει το K να πέρνει συγκεκριμένες τιμές. Για όλες τις υπόλοιπες γίνονται έρευνες για κάθε τιμή ξεχωριστά. Μάλιστα για τις περισσότερες τιμές του K δεν έχουμε ακόμα ακριβή αποτελέσματα. Πάντως πρόσφατα ο $Vliet$ πέτυχε να υπολογίσει τις ακριβείς αποδόσεις του H_K για $K = 4, 5$. Αυτές, όπως και αρκετές άλλες, φαίνονται στον πίνακα 2.7.3.1 που παρατίθεται στο επόμενο κεφάλαιο.

[10] $< k_i >= 1, 2, 6, 42, \ldots$

Όσο πιο μεγάλο είναι το K, και επομένως όσο πιο πολλά υποδιαστήματα έχουμε, τόσο καλύτερη απόδοση θα μας δίνει ένας $Harmonic_K$ αλγόριθμος. Για K αρκετά μεγάλο, οι αλγόριθμοι αυτοί μπορούν να σπάσουν το φράγμα του 1.7 [$LL85$]:

Θεώρημα 3.7.2.2 $lim_{K \to \infty} R_{H_K}^{\infty} = T_{\infty} = 1.69103\ldots$

Ύστερα από πολλές μετρήσεις, έχει παρατηρηθεί ότι για $K \leq 7$, έχουμε $R_{H_K}^{\infty} \leq 1.695$.

3.7.3 ΣΥΓΚΡΙΣΗ

Υπάρχει μία παραλλαγή του γνήσιου $Harmonic$ αλγόριθμου, η οποία συστήθηκε από τον $Woegigner$ [$Woe93$] και προσφέρει εγγυήσεις για απόδοση κάτω του 1.7 όταν το K είναι περίπου στο 6. Οι αλγόριθμοι του $Woegigner$, τους οποίους ονομάζει $Simplified\ Harmonic\ algorithms$ (SH_K), στηρίζονται σε μία πιο πολύπλοκη δομή διαστημάτων, στην οποία δεν θα αναφεθούμε εδώ[11]. Παρ'ολα αυτά αναφέρουμε τους SH_K γιάτι παρουσιάζουν πολύ υψηλές αποδόσεις για τα δεδομένα της κατηγορίας των *bounded space*.

Στον παρακάτω πίνακα δίνονται συγκριτικά οι τιμές του R_A^{∞}, σαν συνάρτηση του K, για τους πιο αποδοτικούς *bounded space* αλγόριθμους που έχουμε αναφέρει. Τα ακριβή όρια για τον H_K δεν είναι γνωστά, γι' αυτό ο πίνακας δίνει άνω και κάτω οριακές τιμές που γνωρίζουμε για αυτόν.

[11] Δεν γίνεται λεπτομερέστερη αναφορά λόγω της εξαιρετικά πολύπλοκης δομής που παρουσιάζουν οι συγκεκριμένοι αλγόριθμοι.

K	NF_K	ABF_K	BBF_K	$H_K \geq$	$H_K \leq$	SH_K
2	2.00000	1.85000	1.70000	2.00000	2.00000	2.00000
3	1.85000	1.80000	1.70000	1.75000	1.75000	1.75000
4	1.80000	1.77500	1.70000	1.71429	1.71429	1.72222
5	1.77500	1.76000	1.70000	1.70000	1.70000	1.70000
6	1.76000	1.75000	1.70000	1.70000	1.70000	1.69444
7	1.75000	1.74286	1.70000	1.69444	1.69444	1.69338
8	1.74286	1.73750	1.70000	1.69377	1.69388	1.69106
9	1.73750	1.73333	1.70000	1.69326	1.69345	1.69104
10	1.73333	1.73000	1.70000	1.69287	1.69312	1.69104
∞	1.70000	1.70000	1.70000	1.69103	1.69103	1.69103

Παρατηρούμε εδώ ότι την καλύτερη εγγύηση για $K\epsilon\{2,3,4\}$ μας την προσφέρει ο BBF_K , ενώ για K=5 μοιράζονται τα πρωτεία οι BBF_K, H_K, SH_K. Τέλος για τις υπόλοιπες τιμές του K, ο SH_K είναι ο καλύτερος. Βέβαια οι εγγυήσεις που προσφέρει ο SH_K δεν έχουν διαφορά από αυτές του H_K άνω του 0.3%. Και όσο το $K \to \infty$ αυτή η διαφορά τείνει να μηδενιστεί. Ασυμπτοτικά και οι δύο αυτοί αλγόριθμοι είναι περίπου κατά 0.5% καλύτεροι από τους FF, BF, BBF_K. Επομένως βλέπουμε εδώ ότι δύο *bounded space* αλγόριθμοι[12] τείνουν να αποκτήσουν καλύτερη απόδοση από γνωστούς *unbounded space* αλγόριθμους όπως ο FF και ο BF. Βέβαια η διαφορά τους είναι αρκετά μικρή αλλά, όπως φαίνεται και στο παρακάτω θεώρημα, μέχρι εκει φτάνουν οι δυνατότητες ένος *bounded space* αλγόριθμου [$LL85$]:

Θεώρημα 3.7.3.1 Αν ο A είναι ένας *bounded-space online bin packing* αλγόριθμος, τότε $R_A^\infty \geq T_\infty = 1.6910\ldots$

Να πούμε σ' αυτό το σημείο ότι παρ' όλο που οι SH_K και H_K είναι ανώτεροι από κάθε άλλο γνωστό *bounded-space* αλγόριθμο για $K > 5$, υστερούν αρκετά στην μέση επίδοση τους η οποία είναι πιο φτωχή κι από αυτήν του NF_K. Αντίθετα οι BBF_K έχουν μία ικανοποιητική *worst-case* συμπεριφορά, αλλά και μία εξίσου καλή *average-case*.

3.8 ΑΥΘΑΙΡΕΤΟΙ ONLINE ΑΛΓΟΡΙΘΜΟΙ

Στα προηγούμενα κεφάλαια μελετήσαμε πολλούς αλγόριθμους οι οποίοι υπακούουν στον περιορισμό *Any Fit*. Όλοι όμως οι *Any Fit* αλγόριθμοι είναι καταδικασμένοι να έχουν αποδόσεις άνω του 1.7. Η λύση δεν επιτυγχάνεται

[12] Οι οποίοι όμως έχουν εξαιρετικά πολύπλοκη δομή.

ούτε με τους *bounded-space* αλγόριθμους καθώς η απόδοσή τους δεν μπορεί να πέσει κάτω από τη σταθερά $T_\infty = 1.6910\ldots$.

Εδώ θα μελετήσουμε ορισμένους *online* αλγόριθμους που δεν υπακούουν σε κανέναν από τους παραπάνω δύο περιορισμούς και τους ονομάζουμε αυθαίρετους *online* αλγόριθμους.

Οι πρώτοι *unbounded-space* αλγόριθμοι που προτάθηκαν και δεν υπακούουν στο περιορισμό *Any Fit* ήταν οι *Group-X Fit* (*GXF*) του *Johnson* [$Joh73$][$Joh74$]. Μπορούμε να πάρουμε διαφορετικές εκδοχές των αλγορίθμων αυτών ανάλογα με την επιλογή του X, το οποίο είναι μία αύξουσα ακολουθία από πραγματικούς αριθμούς τέτοια ώστε $0 = x_0 < x_1 < \ldots < x_p = 1$. Κάθε χρονική στιγμή, ένας μερικώς γεμισμένος κάδος με κενό χώρο g θα φαίνεται σα να έχει ένα $de\ facto$ κενό χώρο ίσο με το $max\{x_i \leq g : 0 \leq i < p\}$. Αυτό ουσιαστικά χωρίζει τους κάδους σε p ομάδες. Από κει και πέρα τα αντικείμενα πακετάρονται όπως στον *Best Fit* λαμβάνοντας υπόψη όμως τα $de\ facto$ κενά που έχουμε ορίσει. Ο αλγόριθμος αυτός μπορεί να τρέξει σε γραμμικό χρόνο για οποιοδήποτε προεπιλεγμένο X. Ένα μεγάλο μειονέκτημα είναι ότι οι κάδοι που κλείνουν οριστικά είναι αυτοί που έχουν κενό χώρο μικρότερο του x_1, γιατί από τις $de\ facto$ τιμές αυτός θεωρείται ίσος με το μηδέν. Δυστυχώς το κύριο πλεονέκτημα του αλγόριθμου αυτού φαίνεται να είναι απλά ότι τρέχει σε γραμμικό χρόνο, καθώς δεν συμπεριφέρεται καλύτερα στις χειρότερες περιπτώσεις από τον *First Fit* και επομένως έχει $R^\infty_{GXF} > 1.7$.

Ο πρώτος *online* αλγόριθμος με $R^\infty_A < T_\infty$ ήταν ο *Refined First Fit* (*RFF*) του *Yao*[13] [$Yao80$]. Αυτός ταξινομεί τα αντικείμενα, ανάλογα με το μέγεθός τους, σε ένα από τα διαστήματα $(0, \frac{1}{3}], (\frac{1}{3}, \frac{2}{5}], (\frac{2}{5}, \frac{1}{2}]$, και $(\frac{1}{2}, 1]$. Επίσης ταξινομεί και τους κάδους με ένα παρόμοιο σχέδιο. Κάθε αντικείμενο τοποθετείται σε μία ιδιαίτερη κατηγορία κάδων σύμφωνα με τον κανόνα *First Fit*, με την κατηγορία των κάδων να καθορίζεται από αυτήν των αντικειμένων. Τα αντικείμενα, που το μέγεθός τους ανήκει στο διάστημα $(\frac{1}{3}, \frac{2}{5}]$, αντιμετωπίζονται προηγουμένως στη λίστα. Συγκεκριμένα το ένα από τα έξι τέτοιου είδους αντικείμενα τοποθετείται κατευθείαν σε νέο κάδο με την ελπίδα ότι στη συνέχεια θα τοποθετηθεί στον ίδιο κάδο ένα με μέγεθος μεγαλύτερο του $\frac{1}{2}$. Αυτή η σκέψη θα επιτρέψει στον αλγόριθμο αυτό να αποφύγει μία άσχημη συμπεριφορά σε μη βολικές για αυτόν λίστες, κάτι που συμβαίνει στον *First Fit* (αλλά και στους *Harmonic* αλγόριθμους). Ο *Yao* έδειξε ότι $R^\infty_{RFF} = 1.666\ldots$

[13]Ουσιαστικά ο πρώτος που ξεπέρασε σε απόδοση και τον *First Fit* καθώς προηγήθηκε των *Harmonic* αλγορίθμων.

Στη συνέχεια θα δούμε κάποιες παραλλαγές των αλγορίθμων που έχουμε δει μέχρι τώρα. Η πρώτη είναι οι *Refined Harmonic* (RH_K) αλγόριθμοι [$LL85$] που είναι παράγωγα των *Harmonic* και *First Fit*. Οι RH_K χρησιμοποιούν την ιδέα του διαμερισμού του HK_{20} με την τροποποίηση ότι τα δύο διαστήματα $(\frac{1}{3}, \frac{1}{2}]$ και $(\frac{1}{2}, 1]$ αντικαθίστανται με τα $(\frac{1}{3}, y], (y, \frac{1}{2}], (\frac{1}{2}, 1-y]$ και $(1-y, 1]$, όπου $y = \frac{37}{96}$. Μοιάζουν στο πακετάρισμα με *Harmonic* αλγόριθμο, μόνο που εδώ προσπαθούμε να ταιριάξουμε στον ίδιο κάδο αντικείμενα με μέγεθος που ανήκει στο πρώτο διάστημα (από τα τέσσερα παραπάνω) με αυτά που ανήκουν στο τρίτο διάστημα. Και είναι αυτός ο σκοπός μας γιατί τέτοια αντικείμενα ταιριάζουν πάντα στον ίδιο κάδο χωρίς να χρειαστεί ν' ανοίξουμε έναν καινούργιο. Ο *Lee* έδειξε ότι $R^{\infty}_{RH_{20}} \leq \frac{373}{228} = 1.6359\ldots$ με μία μικρή βελτίωση αν αυξηθεί το K περαιτέρω.

Στη συνέχεια εμφανίστηκαν οι *Ramanan*, *Brown* και *Lee and Lee* [$RBL89$] με τους λεγόμενους *Modified Harmonic* (MH_K) αλγόριθμους στους οποίους προστέθηκε η πιθανότητα να πακετάρονται ακόμα μικρότερα αντικείμενα με μέγεθος στο $(\frac{1}{2}, \frac{1}{2} + y]$. Οι αλγόριθμοι αυτοί έχουν μία πιο πολύπλοκη δομή καθώς και διαφορετική τιμή του y(εδώ $y = \frac{265}{684}$). Λεπτομέρειες του ορισμού των MH_K δεν θα αναφέρουμε εδώ γιατί είναι αρκετά πολύπλοκες. Ο *Ramanan* έδειξε ότι $1.6156146 < R^{\infty}_{MH_{38}} \leq 1.\overline{615}$. Αργότερα έδειξε ότι υπάρχουν κι άλλες μεταβλητές (y) που θα μπορούσαν να ρίξουν τους δείκτες στο 1.59. Όμως δεν υπάρχει κάποια προσέγγιση να ρίξει τον δείκτη κάτω του $1.\overline{583}$ (όπως απέδειξε αργότερα).

Ο αλγόριθμος που έχει τα πρωτία, είναι ο *Harmonic*+1 του *Richey* [$Ric91$] ο οποίος χρησιμοποιεί διαφορετικές αρχές, αλλά ουσιαστικά επιτυγχάνει να πιάσει το όριο του *Ramanan*, τουλάχιστον στα τρία πρώτα δεκαδικά ψηφία. Παρ' όλο που τρέχει σε γραμμικό χρόνο, είναι αρκετά πολύπλοκος, με δομή που ανακατεύει πάνω από 70 διαστήματα και κανόνες που περιλαμβάνουν πίνακες ολόκληρους από μεταβλητές. Ο *Richey* έδειξε για αυτόν τον αλγόριθμο ότι: $1.5874 \leq R^{\infty}_{A} \leq 1.58872$.

Υπάρχει όμως ανοικτό το ερώτημα για το αν υπάρχουν *online* αλγόριθμοι που μπορούν να τα καταφέρουν καλύτερα από τον *Harmonic* + 1. Δεδομένης και της πολύπλοκης φύσης του τελευταίου αλλά και του γεγονότος ότι όλοι οι αλγόριθμοι που τα καταφέρνουν καλύτερα από τον *First Fit* στην *worst-case* συμπεριφορά τους, δείχνουν να είναι πολύ κατώτεροι στην κατά μέσο όρο απόδοσή τους, το ερώτημα αυτό φαίνεται να χρειάζεται απάντηση. Μέχρι που όμως φτάνουν οι δυνατότητες ενός *online* αλγόριθμου;

Αποδείξαμε σε προηγούμενο κεφάλαιο ότι δεν υπάρχει *online* αλγόριθμος με απόδοση κάτω του $\frac{4}{3}$. Δεν είναι όμως αυτό το γνήσιο κάτω όριο της κατηγορίας αυτής. Ο *Yao* κατάφερε να βρει ένα πιο στενό όριο για τους *online* αλγόριθμους. Απέδειξε ότι κανένας *online* αλγόριθμος δεν μπορεί να έχει $R_A^\infty < 1.5$. Τα επιχειρήματα που χρησιμοποίησε για να το αποδείξει αυτό απαιτούσαν μόνο τρία μεγέθη αντικειμένων: $\frac{1}{2}+$ ε, $\frac{1}{3}+$ ε και $\frac{1}{7}+$ ε. Παρατηρούμε ότι αυτά τα μεγέθη είναι ίσα με $\frac{1}{t_i}+$ ε, για $1 \leq i \leq 3$. Η t_i είναι η ακολουθία που χρησιμοποιήθηκε για τον ορισμό του T_∞ στο κεφάλαιο του *First Fit*. Δεν πέρασε πολύς καιρός και οι *Brown* και *Liang* γενίκευσαν αυτήν την προσέγγιση με τη χρήση αντικειμένων μεγέθους $\frac{1}{2}+$ ε, $\frac{1}{3}+$ ε, $\frac{1}{7}+$ ε, $\frac{1}{43}+$ ε και $\frac{1}{1807}+$ ε τα οποία παράγονται από το $\frac{1}{t_i}+$ ε για $1 \leq i \leq 5$. Απέδειξαν ότι κανένας *online* αλγόριθμος δεν μπορεί να έχει $R_A^\infty < 1.536346\ldots$ Εξαιτίας της ταχείας αύξησης των τιμών του t_i , προσπάθειες να βελτιώσουν αυτό το όριο χρησιμοποιόντας κι άλλα μεγέθη του $\frac{1}{t_i}+$ ε για $i > 5$, μάλλον δεν πετυχαίνουν ιδιαίτερη βελτίωση. Ωστόσο μία περισσότερο προσεχτική ανάλυση που χρησιμοποιεί τις τιμές για $i < 5$ μπορεί να κάνει τη διαφορά, όπως έδειξε ο *Vliet*. Βασισμένος σε μία προσέγγιση που προτάθηκε από τον *Galambos*, ο *Vliet* κατασκευάζει ένα γραμμικό πρόγραμμα του οποίου η λύση διευκρινίζει τον καλύτερο δυνατό τρόπο εκμετάλλευσης των μεγεθών των παραπάνω αντικειμένων και παράγει το χαμηλότερο όριο που είναι γνωστό ως τώρα, και παρατίθεται στο παρακάτω Θεώρημα [$Vli95$][$Vli96$]:

Θεώρημα 3.8.1 Για κάθε *online* αλγόριθμο A, $R_A^\infty \geq 1.540$.

Επομένως κάθε *online* αλγόριθμος δεν μπορεί να πιάσει απόδοση κάτω του 1.540. Για να επιτευχθούν καλύτερες αποδόσεις πρέπει να περάσουμε στην κατηγορία των *semi-online* και των $offline$ αλγορίθμων.

Βέβαια όταν έχουμε περιορισμό ως προς το μέγεθος των αντικειμένων της εισόδου του προβλήματος, επιτυγχάνονται καλύτερες αποδόσεις από το 1.540. Παράκάτω δίνεται ένας πίνακας που περιλαμβάνει τα χαμηλότερα γνωστά όρια που έχει αποδειχτεί ότι μπορεί να πετύχει ένας *online* αλγορίθμος για μεγέθη αντικειμένων έως $\frac{1}{m}$, όπου $1 \leq m \leq 5$. Αυτά συγκρίνονται με τις απόδοσεις των πιο αποτελεσματικών αλγοριθμων που είναι γνωστοί μέχρι τώρα, την οριακή τιμή του $R_{H_K}^\infty(a)$ (καθώς το $K \to \infty$), καθώς και με το $R_{FF}^\infty(a)$.

m	χαμηλότερο όριο	χαμηλότερο γνωστό όριο	$R^{\infty}_{H_K}(1/m)$	$R^{\infty}_{FF}(1/m)$
1	$1.540\ldots$	$1.588\ldots$	$1.691\ldots$	$1.700\ldots$
2	$1.389\ldots$	$1.423\ldots$	$1.423\ldots$	$1.500\ldots$
3	$1.291\ldots$	$1.302\ldots$	$1.302\ldots$	$1.333\ldots$
4	$1.229\ldots$	$1.234\ldots$	$1.234\ldots$	$1.250\ldots$
5	$1.188\ldots$	$1.191\ldots$	$1.191\ldots$	$1.200\ldots$

Φυσικό είναι όσο περιορίζονται τα αντικείμενα της εισόδου να βελτιώνονται δραματικά τα όρια και οι αποδόσεις. Κάτι όμως που αξίζει να σημειωθεί είναι ότι όσο μικρότερα γίνονται τα αντικείμενα, τόσο πιο κοντά στις βέλτιστες δυνατές τιμές φτάνουν οι αποδόσεις όλων των αλγορίθμων. Για παράδειγμα, για $\alpha = 1/5$, ακόμα και ο *First Fit* προσφέρει εγγυήσεις 1% από το καλύτερο δυνατό.

Κεφάλαιο 4

ΜΕΛΕΤΗ OFFLINE ΑΛΓΟΡΙΘΜΩΝ

4.1 ΟΙ SEMI-ONLINE ΑΛΓΟΡΙΘΜΟΙ

Οι *semi-online* αλγόριθμοι αποτελούν μία κατηγορία που συνδιάζει στοιχεία από τους *online* και τους *offline* αλγόριθμους. Όμως οι αλγόριθμοι που ανήκουν στην κατηγορία αυτή παρεκκλίνουν από τους ορισμούς των δύο βασικών κατηγοριών που έχουμε συναντήσει και αποτέλουν έτσι μία ανεξάρτητη οικόγενεια.

Ορισμός: Ένας αλγόριθμος είναι *semi-online* όταν για κάθε τρέχον αντικείμενο, μπορούμε να επανατοποθέτησουμε τα αντικείμενα των ανοικτών κάδων.

Δηλαδή μπορούμε, πριν τοποθετήσουμε το τρέχον αντικείμενο, να βγάλουμε ότι υπάρχει στους ανοικτούς κάδους και να το κατανείμουμε πάλι σε αυτούς με τρόπο που να μας βολεύει. Ουσιαστικά αν είχαμε έναν αλγόριθμο ο οποίος σε κάθε βήμα του εξετάζει όλους τους χρησιμοποιημένους κάδους[1] και ικανοποιεί τον παραπάνω ορισμό, αυτός δεν θα διέφερε σε τίποτα από έναν *offline* αλγόριθμο. Όμως πολλοί *semi-online* αλγόριθμοι διατηρούν σε κάθε τους βήμα, μόνο έναν ορισμένο αριθμό κάδων ανοικτό και έτσι δεν ανήκουν στην κατηγορία των *offline*.

Θα αναφέρουμε παρακάτω όρισμένους *semi-online* αλγόριθμους, χώρις να αναλύσουμε τους *packing rules* του καθενός λόγω της εξαιρετικής πολυπλοκότητας τους.

[1]Επομένως όλοι οι κάδοι είναι ανοικτοί σε κάθε του βήμα.

Οι $Galambos$ και $Woegigner$ $[GW93b]$ παρουσίασαν έναν τέτοιο αλγόριθμο[2], τον REP_3 ,που ποτέ δεν χρησιμοποιεί παραπάνω από τρείς ανοικτούς κάδους και για τον οποίο ισχύει $R^{\infty}_{REP_3} = T_{\infty}$.

Ο $Grove$ $[Gro94]$ από την πλευρά του δημιούργησε έναν αλγόριθμο με την ίδια συμπεριφορά χρησιμοποιώντας μία εναλλακτική έννοια την οποία αποκάλεσε $lookahead$. Για να κατανοήσουμε την έννοια αυτή θα υποθέσουμε ότι έχουμε μία νοητή αποθήκη στην οποία μπορούμε να τοποθετούμε τα αντικείμενα πριν τα ταιριάξουμε σε κάποιον κάδο. Η αποθήκη αυτή είναι σταθερού μεγέθους W, και έτσι ένα αντικείμενο a_i δεν χρειάζεται να πακεταριστεί, προτού εξετάσουμε όλα τα αντικείμενα μέχρι το a_j για τα οποία ισχύει, $\sum_{h=i}^{j} s(a_h) \leq W$ όπου $j > i$. Εφαρμόζοντας την μέθοδο $lookahead$ για αρκετά μεγάλες τιμές του W, ο αλγόριθμος του $Groves$ μπορεί επίσης να μας εγγυηθεί ότι πολύ καλές αποδόσεις.

Είπαμε παραπάνω ότι αν δεν επιβάλλεται κανένας περιορισμός στην επανατοποθέτηση, τότε θα μπορούσαμε να τα καταφέρουμε εξίσου καλά με τον καλύτερο $offline$ αλγόριθμο, επανατοποθετώντας τα πάντα κάθε φορά που θα φτάνει ένα αντικείμενο. Ωστόσο αυτό θα πολλαπλασίαζε τον χρόνο στον οποίο θα έτρεχε το πρόγραμμα, κατά ένα παράγοντα n και θα παρουσίαζε καθυστερήσεις γραμμικού χρόνου κάθε που θα φτάνει ένα αντικείμενο.

Πραγματοποιώντας κατάλληλα τις επανατοποθετήσεις, θα μπορούσαμε να μειώσουμε την καθυστέρηση ανά αντικείμενο σε $T(n)\frac{log(n)}{n}$, όπου $T(n)$ είναι ο χρόνος στον οποίο τρέχει ο αντίστοιχος $offline$ αλγόριθμος. Όμως σε εφαρμογές που επιθυμούμε να διατηρούν $online$ χαρακτηριστικά χρειαζόμαστε πιθανώς πιο δυνατούς περιορισμούς στο μέγεθος της επανατοποθέτησης. Για παράδειγμα να επιτρέπεται καθυστέρηση έως $O(logn)$ ανά αντικείμενο. Ακόμα όμως και κάτω από τέτοιους περιορισμούς, μπορούμε να πετύχουμε σοβαρές βελτιώσεις που να ρίχνουν την απόδοση κάτω του 1.540.

Οι $Gambosi$ et $al.$ $[GPT90]$ σχεδίασαν δύο αλγόριθμους για να σπάσουν αυτό το όριο. Ο πρώτος πέτυχε $R^{\infty}_{A} \leq 1.5$ με καθυστέρηση σταθερού χρόνου ανά αντικείμενο. Για να γινόταν εφικτή, μέχρι πριν τον αλγόριθμο αυτό μία τέτοια απόδοση, έπρεπε να μετακινηθούν $\Omega(n)$ αντικείμενα για κάθε νέο που έπρεπε να προσαρμοστεί. Όμως με την ενίσχυση κατάλληλων δομών δεδομένων οι $Gambosi$ et $al.$ μπόρεσαν να μεταχειριστούν μεγάλες συλλογές αντικειμένων, σαν μία ομάδα, και να τα μετακινήσουν όλα μαζί σε σταθερό χρόνο. Αυτή η

[2] Χαρακτηρίζεται και σαν $online$ $with$ $repacking$.

σκέψη έχει και ρεαλιστική διάσταση καθώς μπορούμε να φανταστούμε ότι τοποθετούμε τα αντικείμενα σε κουτιά και μετακινούμε ολόκληρα τα κουτιά από τον ένα κάδο στον άλλο. Ο αλγόριθμος αυτός βασίζεται σε μία απλή ταξινόμηση των αντικειμένων σε τέσσερις τύπους ανά το μέγεθος. Είναι τέτοια η δομή του, ώστε ποτέ να μην χρειάζονται παραπάνω από τρείς μετακινήσεις απλών αντικειμένων ή ομάδων, όταν ένα καινούργιο αντικείμενο πρέπει να τοποθετηθεί.

Ο δέυτερος αλγόριθμος των *Gambosi et al.* [$GPT90$] χωρίζει τα αντικείμενα σε έξι τύπους και έχει τον περιορισμό ότι το πολύ 7 ομάδες/αντικείμενα μπορούν να επανατοποθετηθούν κάθε φορά που φτάνει ένα νέο. Η απόδοσή του είναι ακόμα καλύτερη κάθως έχει $R_A^\infty \leq \frac{4}{3} = 1.\overline{3}$.

Πιο πρόσφατα οι *Ivkovic* και *Lloyd* [$IL93$] σχεδίασαν έναν αλγόριθμο με $R_A^\infty \leq \frac{5}{4} = 1.25$. Όμως εδώ μπορούν να γίνουν $O(logn)$ μετακινήσεις ομάδων/ αντικειμένων και οι κανόνες τοποθέτησης και επανατοποθέτησης είναι αρκετά πολύπλοκοι.Αυτή είναι και η καλύτερη επίδοση που γνωρίζουμε για τέτοιου είδους *semi-online* αλγόριθμους. Αν θέλουμε ακόμα καλύτερες επιδόσεις τότε πρέπει να ασχοληθούμε με καθαρά $offline$ αλγόριθμους οι οποίοι έχουν πρόσβαση σε όλα τα αντικείμενα της εισόδου από την αρχή του προβλήματος.

4.2 O FIRST FIT DECREASING

Ένας από τους πιο διάσημους $offline$ αλγόριθμους είναι ο $First\ Fit\ Decreasing$. Είδαμε σε προηγούμενα κεφάλαια πολλά παραδείγματα από λίστες, οι οποίες δυσκολεύουν πολύ κάποιους αλγόριθμους και τους οδηγούν σε αρκετά χαμηλές αποδόσεις. Αν παρατηρήσουμε τις λίστες αυτές, οι περισσότερες περιέχουν αντικείμενα που είναι παραταγμένα με αύξουσα σειρά. Με αφορμή αυτό το γεγονός, έγιναν μελέτες οι οποίες απέδειξαν ότι αλγόριθμοι, όπως ο FF και ο BF, συμπεριφέρονται με πολύ καλύτερο τρόπο, όταν δεκτούν σαν είσοδο λίστες με αντικείμενα ταξινομιμένα σε φθίνουσα σειρά.

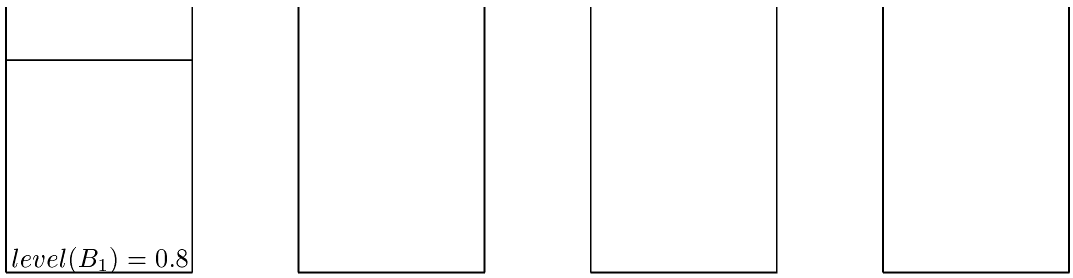
Βέβαια οι περισσότερες λίστες δεν είναι τόσο βολικές γιατί τα αντικείμενα τους έχουν διάταξη με τυχαία σειρά. Όμως ένας $offline$ αλγόριθμος έχει την πολυτέλεια να γνωρίζει πλήρως την λίστα εισόδου. Έτσι θα μπορούσε να ταξινομήσει πρώτα τα αντικείμενα κατα φθίνουσα σειρά και στη συνέχεια να αρχίσει να τα τοποθετεί. Αυτό ακριβώς κάνει και ο $First\ Fit\ Decreasing(FFD)$.

FFD Packing Rule

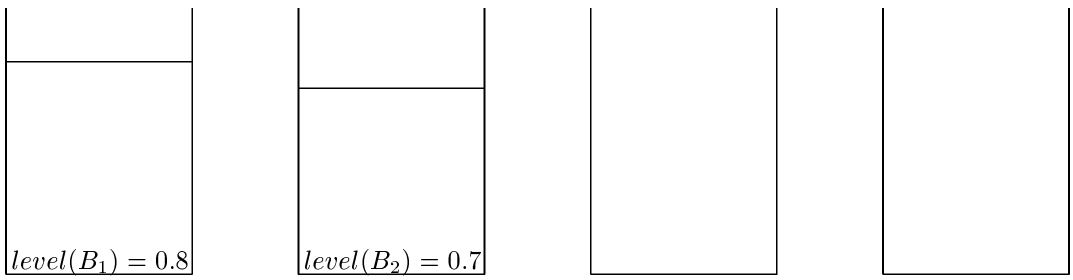
1. Ταξινομούνται τα αντικείμενα κατά φθίνουσα σειρά.

2. Τοποθετούνται τα αντικείμενα σύμφωνα με τον *packing rule* του *First Fit*.

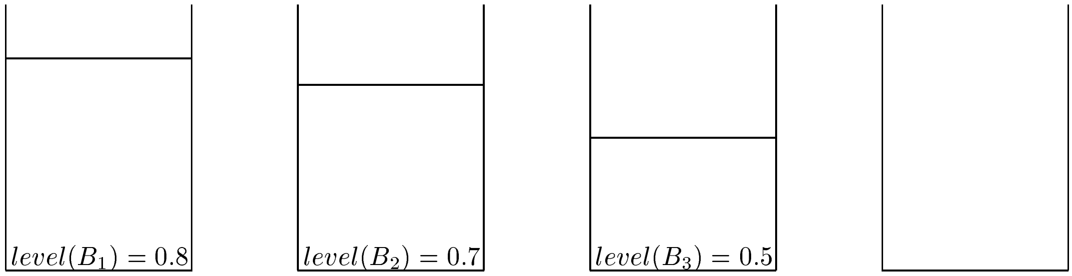
Παράδειγμα: Έστω ότι έχουμε σαν είσοδο τη λίστα $L = \{0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8\}$ με $OPT(L) = 3$ κάδους. Ο FFD θα ταξινομίσει αρχικά τα αντικείμενα κατά φθίνουσα σειρά. Έτσι δημιουργεί μία νέα λίστα, έστω L', της μορφής: $L' = \{0.8, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1\}$. Στη συνέχεια τοποθετεί τοποθετεί τα αντικείμενα της νέας λίστας L' σύμφωνα με τον $p.r.$ του FF.



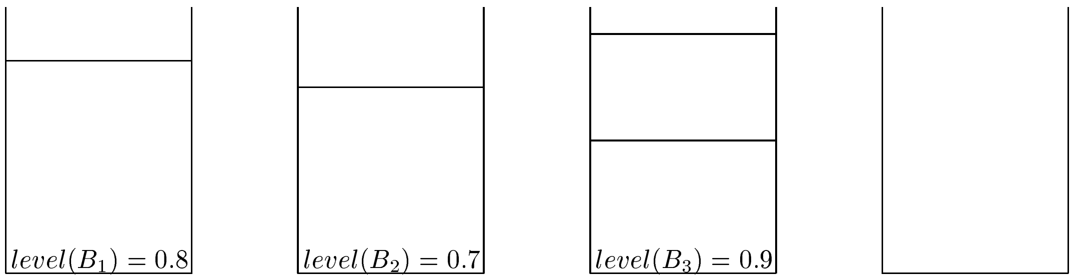
1ο ΒΗΜΑ Το $s(a'_1) = 0.8$ τοποθετείται στον πρώτο κάδο.



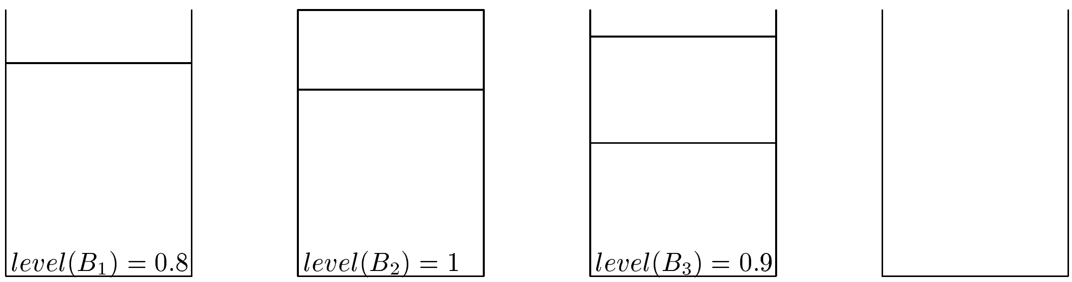
2ο ΒΗΜΑ Το $s(a'_2) = 0.7$ τοποθετείται στον δεύτερο κάδο B_2, καθώς δεν ταιριάζει στον B_1.



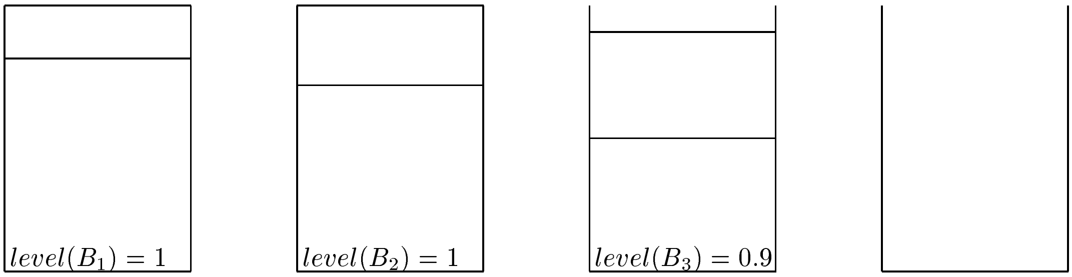
3ο ΒΗΜΑ Το $s(a'_3) = 0.5$ τοποθετείται στον B_3 κάδο, καθώς δεν ταιριάζει στον B_1, αλλά ούτε και στον B_2.



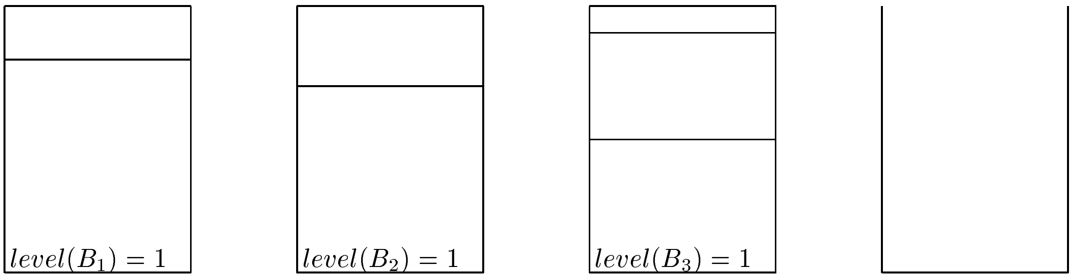
4ο ΒΗΜΑ Το $s(a'_4) = 0.4$ δεν ταιριάζει στον B_1, ούτε στον B_2. Ταιριάζει όμως στον B_3 και τοποθετείται εκεί.



5ο ΒΗΜΑ Το $s(a'_5) = 0.3$ δεν ταιριάζει στον B_1, αλλά ταιριάζει στον B_2 και τοποθετείται εκεί. Ο B_2 κάδος γεμίζει και δεν μπορούν να εισέλθουν σ' αυτόν αλλά αντικείμενα.



6ο ΒΗΜΑ Το $s(a'_6) = 0.2$ ταιριάζει στον B_1 κάδο και τοποθετείται εκεί. Ο B_1 γεμίζει επίσης.



7ο ΒΗΜΑ Το $s(a'_7) = 0.1$ δεν μπορεί να εισέλθει στους γεμάτους κάδους B_1 και B_2, αλλά μπορεί στον B_3 όπου και εισέρχεται. Ο B_3 γεμίζει επίσης.

Επομένως, για τη λίστα L ο FFD χρησιμοποιεί 3 κάδους που είναι και η βέλτιστη λύση.

Παραθέτουμε έδω ένα Λήμμα που μπορεί εύκολα να αποδειχτεί.

Λήμμα 4.2.1 Ο *First Fit Decreasing* χρησιμοποιεί το πολύ $\frac{3}{2}OPT(L) + 1$ κάδους.

Απόδειξη: Ας υποθέσουμε ότι η είσοδος του προβλήματος είναι η λίστα L. Ο FFD την ταξινομεί αρχικά κατά φθίνουσα σειρά. Έστω λοιπόν ότι προκύπτει η ταξινομημένη λίστα: $S = \{s_1, s_2, \ldots, s_n\}$. Μπορούμε να διακρίνουμε τα αντικείμενα της σε τέσσερις κατηγορίες A, B, C, D:

$A = \{s_i \, \epsilon \, S | \; \frac{2}{3} < s_i \leq 1\}$

$B = \{s_i \, \epsilon \, S | \; \frac{1}{2} < s_i \leq \frac{2}{3}\}$

$C = \{s_i \, \epsilon \, S | \; \frac{1}{3} < s_i \leq \frac{1}{2}\}$

$D = \{s_i \, \epsilon \, S | \; 0 < s_i \leq \frac{1}{3}\}$

Προφανώς ο FFD έχει τοποθετήσει πρώτα τα αντικείμενα της A κατηγορίας, στη συνέχεια της B, της C και τελευταία της D. Τα A αντικείμενα θα τοποθετηθούν το καθένα σε ένα κάδο. Το ίδιο και τα B, καθώς τα μεγέθη τους είναι αρκέτα μεγάλα για να ταιριάξουν δύο από αυτά στον ίδιο κάδο. Τα C αντικείμενα θα τοποθετηθούν σε ένα κάδο που περιέχει το πολύ ένα άλλο αντικείμενο της B ή της C κατηγορίας[3].

Η τοποθέτηση του FFD για τα A, B, C αντικείμενα είναι η βέλτιστη δυνατή. Πώς όμως αποδεικνύεται αυτό; Θα δούμε πως θα έπρατε μία *optimal* λύση. Καθώς τα A και B αντικείμενα απαιτούν ένα ξεχωριστό κάδο το καθένα, μπορούν κι εδώ να τοποθετηθούν στους πρώτους κάδους, όπως και στον FFD. Στη συνέχεια μπορεί να υπάρξουν κάδοι οι οποίοι περιέχουν και B και C αντικείμενα. Έστω ότι συναντάμε τον πρώτο κάδο αυτού του είδους. Εάν το C αντικείμενο δεν είναι το μεγαλύτερο δυνατό που ταιριάζει σ΄ αυτόν τον κάδο, η *optimal* λύση θα το αντάλλαζε[4] με αυτό. Με την ανταλλαγή όμως αυτή, το αντικείμενο που βρέθηκε στον κάδο που εξετάζουμε, είναι αυτό που θα τοποθετούσε και ο FFD. Η *optimal* λύση θα εργαστεί με τον ίδιο ακριβώς τρόπο και για τους υπόλοιπους κάδους αυτού του είδους[5] Επομένως, μέχρι στιγμής η *optimal* λύση συμπίπτει με αυτήν του FFD. Όμως οι υπόλοιποι κάδοι που απομένουν περιέχουν μόνο C αντικείμενα. Εδώ και οι δύο λύσεις θα τοποθετήσουν δύο C αντικείμενα ανά κάδο. Άρα η *optimal* λύση και αυτή του FFD συμπίπτουν πλήρως.

Αφού έχει τοποθετήσει ο FFD τα αντικείμενα των τριών πρωτων κατηγοριών,συνεχίζει με αυτά της D κατηγορίας. Αν η D έχει 0 στοιχεία, τότε ο

[3] Τα C αντικείμενα δεν ταιριάζουν σε κάδο που περιέχει A αντικείμενα.

[4] Θα τοποθετούσε στον κάδο που εξετάζουμε το μεγαλύτερο δυνατό, και θα έβαζε το C αντικείμενο που βρήκαμε στην αρχή, στον κάδο που υπήρχε το άλλο.

[5] Αυτοί που περιέχουν B και C αντικείμενα.

FFD ολοκληρώνει παραδίδοντας την *optimal* λύση. Στην αντίθετη περίπτωση υπάρχουν δύο πιθανές εκδοχές:

1η Εκδοχή: Να μην ανοικτεί κανένας καινούργιος κάδος και να τοποθετηθούν όλα τα D αντικείμενα στους προηγούμενους κάδους. Στην περίπτωση αυτή ο FFD επιστρέφει πάλι την *optimal* λύση.

2η Εκδοχή: Ανοίγει τουλάχιστον ένας καινούργιος κάδος. Σε αυτήν την περίπτωση, αν χρησιμοποιήθηκαν m κάδοι, οι πρώτοι $m-1$ έχουν κενό χώρο το πολύ $\frac{1}{3}$. Μόνο ο τελευταίος κάδος, B_m, μπορεί να έχει παραπάνω χώρο. Επομένως καταλήγουμε στις εξής σχέσεις:

$$(m-1)\frac{2}{3} < \sum_{i=1}^{n} s_i \leq OPT(L) \Rightarrow (m-1)\frac{2}{3} < OPT(L) \Rightarrow$$
$m < \frac{3}{2}OPT(L) + 1$

Όμως το γνήσιο όριο του FFD φαίνεται στο παρακάτω θεώρημα [$Joh73$]:

Θεώρημα 4.2.2 $R^{\infty}_{FFD} = \frac{11}{9} = 1.\overline{2}$

Συγκεκριμένα είχε αποδειχτεί ότι $FFD(L) \leq (11/9)OPT(L) + 4.$. Στη συνέχεια, ο *Baker* [$Bak83$] μείωσε τον πρόσθετο όρο της παραπάνω σχέσης από 4 σε 3, ενώ ο *Yue* [$Yue91$] πέτυχε τη μείωση του πρόσθετου όρου στο 1.

Όπως και με όλους τους προηγούμενους αλγόριθμους, οι δείκτες του FFD βελτιώνονται όσο το μέγιστο μέγεθος α των αντικειμένων μειώνεται. Μόνο που τώρα η εξάρτηση από το α είναι αρκετά πιο πολύπλοκη από όλες τις προηγούμενες περιπτώσεις [$Joh73$]:

Θεώρημα 4.2.3 Το $R^{\infty}_{FFD}(a)$ είναι ίσο με:

A. $\frac{11}{9}$, για $\frac{1}{2} < a \leq 1$

B. $\frac{71}{60}$, για $\frac{8}{29} < a \leq \frac{1}{2}$

C. $\frac{7}{6}$, για $\frac{1}{4} < a \leq \frac{8}{29}$

D. $\frac{23}{20}$, για $\frac{1}{5} < a \leq \frac{1}{4}$

Το παραπάνω θεώρημα δεν καλύπτει τις περιπτώσεις όπου $a \leq \frac{1}{5}$. Μία μεταγενέστερη, όμως, έρευνα έδειξε ότι για $m = \frac{1}{a} \geq 4$, ισχύει η εξής σχέση:

$R^{\infty}_{FFD}(\frac{1}{m}) = F_m \equiv 1 + \frac{1}{m+2} - \frac{2}{m(m+1)(m+2)}$

Όμως η σχέση αυτή ισχύει μόνο όταν ο m είναι άρτιος. Όταν ο m είναι περιττός, ο *Csirik* [$CS93$] έδειξε ότι:

$R^{\infty}_{FFD}(\frac{1}{m}) = G_m \equiv 1 + \frac{1}{m+2} - \frac{1}{m(m+1)(m+2)}$

Ο *Xu* [$Xu93$] βασίστηκε πάνω στις προηγούμενες σχέσεις και έδειξε ότι, όταν ο m είναι άρτιος, το $R^{\infty}_{FFD}(a)$ ισούται με το F_m σε όλο το διάστημα $(\frac{1}{m+1}, \frac{1}{m}]$. Όμως όταν ο m είναι περιττός, υπάρχει d_m[6] με $\frac{1}{m+1} < d_m < \frac{1}{m}$, τέτοιο ώστε το $R^{\infty}_{FFD}(a)$ να ισούται με το G_m για το διάστημα $(d_m, \frac{1}{m}]$. Το $R^{\infty}_{FFD}(a)$ ισούται με το F_m στο διάστημα $(\frac{1}{m+1}, d_m]$.

4.3 O BEST FIT DECREASING

BFD Packing Rule

1. Ταξινομούνται τα αντικείμενα κατά φθίνουσα σειρά.
2. Τοποθετούνται τα αντικείμενα σύμφωνα με τον *packing rule* του *Best Fit*.

Παρατηρούμε ότι ο *Best Fit Decreasing*(BFD) χρησιμοποιεί την ίδια μεθοδολογία με τον FFD, με τη μόνη διαφορά ότι εφαρμόζει τον *packing rule* του *Best Fit*.

Ο BFD συμπεριφέρεται σχεδόν με τον ίδιο τρόπο με τον FFD και για την λίστα L που χρησιμοποιούμε σαν παράδειγμα. Παραδίδει και αυτός τη βέλτιστη λύση.

Ο *Best Fit Decreasing* στηρίζεται επίσης στην ιδέα της φθίνουσας ταξινόμησης, επομένως αναμένεται να έχει εξίσου καλή απόδοση με τον FFD [$Joh73$]:

Θεώρημα 4.3.1 $R^{\infty}_{BFD} = \frac{11}{9} = 1.\overline{2}$

[6] Η ακριβής τιμή του d_m είναι $d_m = \frac{(m+1)^2}{(m^3+3m^2+m+1)}$.

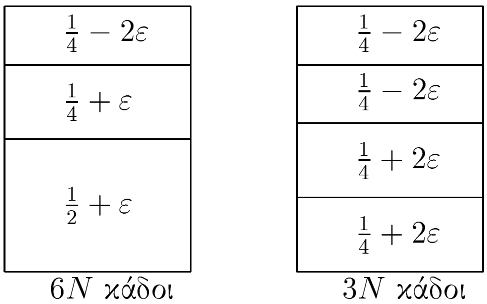
4.4 ΣΥΓΚΡΙΣΗ FFD ΜΕ BFD

Παρατηρούμε ότι οι BFD και FFD έχουν την ίδια χειρότερη επίδοση. Το ίδιο ακριβώς συνέβαινε με τους *First Fit* και *Best Fit*. Παρακάτω φαίνεται ένα παράδειγμα που οδηγεί τους BFD και FFD στην χαμηλότερη δυνατή απόδοση. Η λίστες που δυσκολεύουν τόσο πολύ τους δύο αυτούς αλγόριθμους είναι της μορφής:

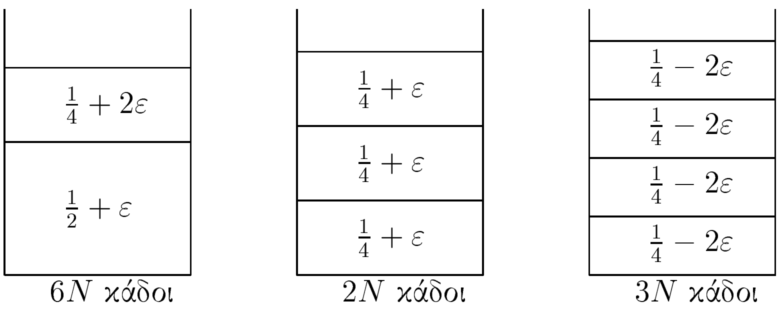
$$L = \{\underbrace{\frac{1}{2}+\varepsilon \dots \frac{1}{2}+\varepsilon}_{6N}, \underbrace{\frac{1}{4}+2\varepsilon \dots \frac{1}{4}+2\varepsilon}_{6N}, \underbrace{\frac{1}{4}+\varepsilon \dots \frac{1}{4}+\varepsilon}_{6N}, \underbrace{\frac{1}{4}-2\varepsilon \dots \frac{1}{4}-2\varepsilon}_{12N}\}$$

.

Το μεγεθός τους είναι $30N$ αντικείμενα. Η τοποθέτηση που πραγματοποιεί η *optimal* λύση για τις λίστες L είναι η εξής:



Επομένως η *optimal* λύση χρησιμοποιεί $9N$ κάδους. Για τις λίστες L, οι FFD και BFD πραγματοποιούν την ίδια τοποθετήση η οποία φαίνεται παρακάτω:



Είναι προφανές ότι χρησιμοποιούν συνολικά $11N$ κάδους. Επομένως $\frac{FFD(L)}{OPT(L)} = \frac{BFD(L)}{OPT(L)} = \frac{11}{9}$. Το μεγαλύτερο αντικείμενο της εισόδου του παραδείγματος είναι το $\frac{1}{2} + \varepsilon$. Το ε παίρνει αρκετά χαμηλές τιμές και συνεπώς το $\frac{1}{2} + \varepsilon$ είναι λίγο μεγαλύτερο από το $\frac{1}{2}$. Επομένως, αν υπάρχει λίστα με αντικείμενα της τάξεως του $\frac{1}{2}$ που οδηγεί τους FFD και BFD στη χαμηλότερη δυνατή απόδοσή τους, τότε για όλες τις λίστες με όριο στο μέγεθος των αντικειμένων τους από το $\frac{1}{2}$ στο 1, οι FFD και BFD θα έχουν χειρότερη επίδοση ίση με $\frac{11}{9}$. Αυτό φαίνεται και στο θεώρημα 4.2.3.

Παρ' όλο που οι δύο αυτοί αλγόριθμοι παρουσιάζουν γενικά την ίδια απόδοση, για μεμονομένες λίστες η συμπεριφορά τους διαφοροποιείται. Για παράδειγμα, έρευνες $[Joh73][JDU74]$ έχουν δείξει ότι για όλες τις λίστες L με αντικείμενα μεγαλύτερα ή ίσα του $\frac{1}{6}$, $BFD(L) \leq FFD(L)$. Από την άλλη πλευρά, για λίστες που περιέχουν αντικείμενα μικρότερα του $\frac{1}{6}$, $BFD(L) = (10/9)FFD(L)$.

Κεφάλαιο 5

ΜΕΛΕΤΗ ΜΕΣΩΝ ΕΠΙΔΟΣΕΩΝ

Ένα απο τα μεγαλύτερα μειονεκτήματα της μελέτης των χειρότερων επιδόσεων είναι ότι δεν αντικατοπτρίζουν πλήρως, πολλές φορές, την συμπεριφορά ενός αλγόριθμου. Για παράδειγμα υπάρχουν κάποιοι αλγόριθμοι που ίσως να μην έχουν την καλύτερη *worst-case* συμπεριφορά, αλλά κατά μέσο όρο η απόδοση τους είναι αρκετά σταθερή και ικανοποιητική. Άλλωστε οι λίστες που οδηγούν έναν αλγόριθμο στην χαμηλότερη δυνατή απόδοση, είναι αρκετά πολύπλοκες και σπάνια τις συναντάει κανείς στην πραγματικότητα.

Η μελέτη μέσων επιδόσεων προσφέρει μία πιο σφαιρική εικόνα για κάθε αλγόριθμο, καθώς μελετάει την συμπεριφορά του κατα μέσο όρο, για διάφορες πιθανές εισόδους.

5.1 ΟΡΟΛΟΓΙΑ

Τα μεγέθη των αντικειμένων μίας λίστας ακολουθούν μία πιθανοτική κατανομή. Παρ΄ όλο που έχουν μελετηθεί πολλών ειδών κατανομές, τις περισσότερες φορές υποθέτουμε ότι έχουμε λίστες με ανεξάρτητα, πανομοιότυπα κατανεμημένα αντικείμενα. Έτσι συνήθως θα έχουμε σαν είσοδο λίστες L_n με τυχαία μεγέθη αντικειμένων.

Μία καινούργια μεταβλητή που θα χρησιμοποιήσουμε είναι η $s(L_n)$ που δηλώνει το άθροισμα των μεγεθών των αντικειμένων της λίστας L_n. Επίσης θα χρειαστούμε την ήδη γνωστή $R_A(L_n) \equiv \frac{A(L_n)}{OPT(L_n)}$ όπου A ένας τυχαίος αλγόριθμος.

Εδώ θα δοθεί και μία διαφορετική διάσταση στο πρόβλημα, καθώς θα μελετήσουμε, εκτός του αριθμού των κάδων, και το πόσο γεμάτοι είναι οι χρησιμοποιημένοι κάδοι. Η μεταβλητή $W_A(L_n) \equiv A(L_n) - s(L_n)$ δηλώνει το χώρο[1] που έχει μείνει συνολικά ανεκμετάλλευτος μετά την τοποθέτηση των αντικειμένων.

Τις περισσότερες φορές, είναι πολύ πιο εύκολο να βρούμε την κατά μέσο όρο τιμή του $W_A(L_n)$ παρά του $R_A(L_n)$. Όμως, καθώς για τις περισσότερες κατανομές ισχύει $\lim_{n\to\infty} E[\frac{OPT(L_n)}{s(L_n)}] = 1$, αν υπολογίσουμε τιμές σχετικές του $W_A(L_n)$, μπορούμε να καταλήξουμε σε ανάλογες εκτιμήσεις και αποτελέσματα για το $R_A(L_n)$. Αυτό φαίνεται κι από την παρακάτω σχέση:

$$E[W_A(L_n)] = \Theta(E[A(L_n) - OPT(L_n)])$$

Δηλαδή μία υψηλή κατά μέσο όρο τιμή του *wasted space*, υποδηλώνει, τις περισσότερες φορές, χαμηλές αποδόσεις για έναν συγκεκριμένο αλγόριθμο. Κάτι τέτοιο άλλωστε, είναι απόλυτα φυσιολογικό. Όσο χαμηλότερο είναι το *wasted space*, τόσο λιγότερο ανεκμετάλλευτο χώρο αφήνουμε σε κάθε κάδο και επομένως χρησιμοποιούμε και λιγότερους κάδους. Όταν ένας αλγόριθμος έχει χαμηλή απόδοση και χρησιμοποιεί έτσι αρκετούς κάδους για την τοποθέτηση όλων των αντικειμένων, ουσιαστικά δεν εκμεταλλεύεται πλήρως την χωρητικότητα των κάδων που χρησιμοποιεί. Γι' αυτό άλλωστε, μία βέλτιστη λύση έχει το χαμηλότερο δυνατό *wasted space*.

Θα ορίσουμε εδώ ένα δείκτη αντίστοιχο του απόλυτου πηλίκου προσέγγισης χειρότερης επίδοσης, ο οποίος όμως εκφράζει εκτιμήσεις μέσων επιδόσεων. Για κατανομή αντικείμενων έστω F, και έναν τυχαίο αλγόριθμο A, ορίζουμε:

$$\overline{R}_A^n(F) \equiv E[R_A(L_n)] = E[\frac{A(L_n)}{OPT(L_n)}]$$

Ο δείκτης αυτός ονομάζεται αναμενόμενο πηλίκο προσέγγισης (*expected ratio*). Όμοια ορίζουμε το *expected wasted space*:

$$\overline{W}_A^n(F) \equiv E[A(L_n) - s(L_n)]$$

Φυσικά οι λίστες L_n έχουν δημιουργηθεί σύμφωνα με την κατανομή F. Σ' αυτό το σημείο θα ορίσουμε το ασυμπτωτικά αναμενόμενο πηλίκο προσέγγισης (*asymptotic expected ratio*) ως εξής:

[1]Στην αγγλική ορολογία ονομάζεται *wasted space*.

$\overline{R}_A^{\infty}(F) \equiv lim_{n\to\infty}\overline{R}_A^{n}(F)$

Δεν υπάρχει αντίστοιχος δείκτης για το $wasted\ space$ καθώς το $lim_{n\to\infty}\overline{R}_A^{n}(F)$ συνήθως δεν συγκλίνει σε κάποια τιμή.

Κατά καιρούς έχουν υπολογιστεί αρκετά αποτελέσματα μέσης επίδοσης για αυθαίρετες κατανομές F. Αυτά όμως δεν αφορούν συνήθως συγκεκριμένους αλγόριθμους, αλλά πιο γενικές περιπτώσεις. Τα περισσότερα αποτελέσματα πάνω σε αναμενόμενα πηλίκα προσέγγισης αλγορίθμων, αφορούν ομοιόμορφες κατανομές. Αυτές μπορεί να είναι συνεχείς ή διακριτές. Οι συνεχείς δηλώνονται με τον συμβολισμό $U[a,b]$, όπου $0 \le a < b \le 1$. Τα μεγέθη των αντικειμένων επιλέγονται ομοιόμορφα εντός του διαστήματος $[a,b]$. Δηλαδή όλες οι τιμές εντός του διαστήματος $[a,b]$ έχουν την ίδια πιθανότητα να επιλεγούν, για να αποτελέσουν το μέγεθος ενός αντικειμένου. Όταν δεν επιβάλλεται κανένας περιορισμός στα μεγέθη, προφανώς θα έχουμε κατανομή $U[0,1]$. Οι διακριτές ομοιόμορφες κατανομές συμβολίζονται ως $U[j,k]$, όπου $1 \le j \le k$. Εδώ τα μεγέθη των αντικειμένων επιλέγονται ομοιόμορφα από το πεπερασμένο σύνολο $\{\frac{1}{k}, \frac{2}{k}, \ldots, \frac{j}{k}\}$. Παρατηρούμε ότι μία διακριτή κατανομή $U[jm,km]$, τείνει να γίνει ίδια με τη συνεχή κατανομή $U[0,\frac{j}{k}]$, καθώς το $m \to \infty$. Ωστόσο οι δύο αυτοί τύποι κατανομών οδηγούν σε αποτελέσματα που συχνά είναι πολύ διαφορετικά μεταξύ τους.

Παρακάτω θα μελετήσουμε τις μέσες επιδόσεις αρκετών από τους αλγόριθμους που έχουμε αναφέρει.

5.2 BOUNDED-SPACE ONLINE ΑΛΓΟΡΙΘΜΟΙ

Οι πρώτες έρευνες πάνω σε μέση επίδοση έγιναν για τον $Next\ Fit$. Οι $Coffman$, So, $Hofri$, και Yao $[CSH80]$ απέδειξαν ότι για $F = U[0,1]$, η κατανομή του $level$ του κλειστού κάδου με τον υψηλότερο δείκτη[2], συγκλίνει γεωμετρικά γρήγορα στην σταθερή κατανομή $V(x) = x^3$, $0 \le x \le 1$. Η μέση τιμή της κατανομής αυτής είναι $\frac{3}{4}$. Έτσι αποδείχτηκε ότι:

Θεώρημα 5.2.1 $\overline{R}_{NF}^{\infty}(U[0,1]) = \dfrac{4}{3}$

[2] Δηλαδή του κάδου που βρίσκεται πριν τον ανοικτό κάδο του NF.

Οι *Lee and Lee* [$LL87$] μελέτησαν τους *Harmonic* αλγόριθμους H_K, και απέδειξαν ότι για κάθε $K \geq 3$, $\overline{R}^{\infty}_{H_K}(U[0,1]) < 1.306$. Είναι φανερό ότι το ασυμπτοτικά αναμενόμενο πηλίκο προσέγγισης των H_K είναι αρκετά καλύτερο από αυτό του NF, ακόμα και για σχετικά μικρό K. Μάλιστα, λογικό είναι, καθώς το $K \to \infty$ να έχουμε ακόμα καλύτερες αποδόσεις:

Θεώρημα 5.2.2 $\lim_{K\to\infty} \overline{R}^{\infty}_{H_K}(U[0,1]) = \frac{\pi^2}{3} - 2 \simeq 1.2899$

Παρατηρουμε, ωστόσο, ότι η διαφορά που έχουν οι H_K και NF είναι πολυ μικρότερη από αυτήν της *worst-case* συμπεριφοράς τους[3]. Κάτι τέτοιο είναι λογικό, καθώς οι μέσες επιδόσεις όλων των αλγόριθμων δεν διαφέρουν τόσο πολύ όσο οι αντίστοιχες χειρότερες επιδόσεις τους.

Έχουν γίνει πειραματικές μελέτες για τους περισσότερους αλγόριθμους που έχουμε αναφέρει μέχρι αυτό το σημείο, όσον αφορά τις μέσες επιδόσεις τους, και έχουν παρασταθεί γραφικά. Σε αρκετές από αυτές παριστάνεται το $\overline{R}^{\infty}_{A}(U[0,b])$ ενός αλγόριθμου σαν συνάρτηση του b.

Θα δούμε σ' αυτό το σημείο έναν αλγόριθμο τον οποίο δεν έχουμε δει μέχρι τώρα και είναι ο *Smart Next Fit*(SNF). Η παρουσίαση καθώς και η μελέτη του επικεντρώθηκαν στην βελτίωση της μέσης επίδοσης του NF. Αυτός αναλύθηκε από τον *Ramanan* [$Ram89$] και υπακούει στον παρακάτω κανόνα:

SNF Packing Rule : Το επόμενο αντικείμενο της εισόδου τοποθετείται, εφόσον ταιριάζει, στον τρέχων ανοικτό κάδο B, όπως ακριβώς στον *Next Fit*. Εάν δεν ταιριάζει σ' αυτόν, τότε ανοίγει καινούργιος κάδος B' και τοποθετείται εκεί. Ο νέος τρέχων κάδος θα είναι εκείνος από τους B και B' που έχει περισσότερο διαθέσιμο χώρο[4].

Ο SNF είναι μία παραλλαγή που βρίσκεται μεταξύ του *Next Fit* και του 2-*Bounded Best Fit*. Είναι ανώτερος του NF για $\frac{1}{2} < b \leq 1$, ωστόσο η διαφορά τους τείνει στο 0 καθώς το $b \to \frac{1}{2}$.

Μία ενδιαφέρουσα παρατήρηση που προκύπτει από αρκετές γραφικές παραστάσεις είναι ότι, ενώ θα περίμενε κανείς οι καμπύλες όλων των αλγορίθμων να είναι γνησίως αύξουσες, αυτές δεν είναι. Για παράδειγμα η καμπύλη του NF

[3]Η διαφορά τους εδώ είναι $\frac{4}{3} - 1.2899 = 0.043$, ενώ στην *worst-case* ανάλυση ήταν $2 - 1.691 = 0.309$.

[4]Δηλαδή εκείνος από τους B και B' ο οποίος έχει χαμηλότερο *level*.

παρουσιάζει μέγιστο για $b \approx 0.841$. Αυτό σημαίνει ότι ο NF έχει την χαμηλότερη δυνατή απόδοση, όταν δέχεται σαν είσοδο λίστες με μέγιστο αντικείμενο 0.841 και όχι λίστες χωρίς περιορισμούς.

Όπως στην *worst-case* ανάλυση, έτσι και στην *average-case*, οι *online bounded-space* αλγόριθμοι έχουν κάποια όρια στην απόδοσή τους. Συγκεκριμένα, κανένας αλγόριθμος που ανήκει σ' αυτήν την κατηγορία, δεν έχει $\overline{R}_A^{\infty}(U[0,1]) = 1$. Επίσης οι $Coffman$ και $Shor$ $[CS93]$ απέδειξαν ότι το *expected wasted space* τους είναι κάθε φορά αναπόφευκτα μεγαλύτερο από μία συγκεκριμένη τιμή:

Θεώρημα 5.2.3 Άν ο Α είναι *online bounded-space* αλγόριθμος με K ανοικτούς κάδους τότε:

$$\overline{W}_A^n(U[0,1]) \geq \frac{n}{16(K+1)}$$

Όσο το Κ αυξάνει και ο αριθμός των αντικειμένων n της εισόδου μειώνεται, τόσο καλύτερες πιθανότητες υπάρχουν να έχουμε χαμηλότερο *wasted space*.

Ανάλογα αποτελέσματα έχουν βρεθεί για τους *bounded-space* αλγόριθμους, στην περίπτωση που έχουμε σαν είσοδο διακριτές κατανομές.

5.3 ΑΠΟΤΕΛΕΣΜΑΤΑ ΓΙΑ ΑΛΛΟΥΣ ONLINE ΑΛΓΟΡΙΘΜΟΥΣ

5.3.1 ΣΥΝΕΧΕΙΣ ΚΑΤΑΝΟΜΕΣ

Ένας από τους αλγόριθμους που αναφέραμε σε προηγούμενο κεφάλαιο, και ανήκει στην κατηγορία των αυθαίρετων *online*, είναι ο $Modified\ Harmonic$ MH_{38}. Οι $Ramanan$ και $Tsuga$ $[RT89f]$ έδειξαν ότι:

$$1.27684 \leq \overline{R}_{MH_{38}}^{\infty}(U[0,1]) \leq 1.27688$$

Η απόδοση αυτή είναι καλύτερη απο το $1.333\ldots$ του NF και από το 1.2899 των $Harmonic$ αλγόριθμων. Όμως δεν είναι τόσο καλή όσο το 1.227 του SNF. Ωστόσο, επινοήθηκαν $[RT89f]$ διάφορες παραλλαγές των $Modified$

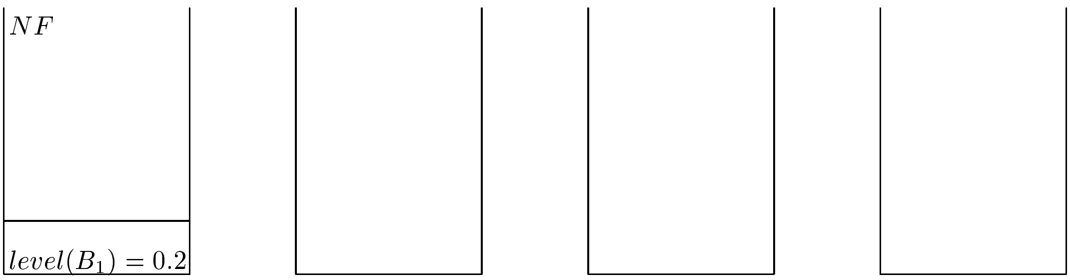
$Harmonic$ αλγόριθμων για τις οποίες ισχύει:

$$\overline{R}^{\infty}_{MH_K}(U[0,1]) < 1.2, \text{ όταν } K \geq 5 \text{ και } lim_{K\to\infty}\overline{R}^{\infty}_{MH_K}(U[0,1]) \approx 1.189$$

Ο $Hoffman$ $[Hof82]$ παρατήρησε ότι υπάρχουν $online$ αλγόριθμοι $\Theta(nlogn)$ χρόνου, των οποίων η αναμενόμενη συμπεριφορά, για $F = U[0,1]$, είναι ασυμπτοτικά βέλτιστη. Τέτοιοι είναι οι $Online\ Match$ αλγόριθμοι(OM_K για $K \geq 3$), τους οποίους συναντάμε για πρώτη φορά.

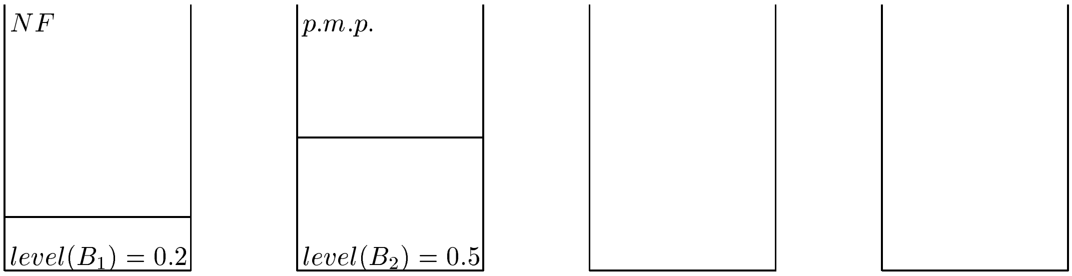
$\mathbf{OM_K}$ Packing Rule: Δύο αντικείμενα p και p' ονομάζονται συνταιριάζοντα για τον OM_K , αν ισχύει γι' αυτά η σχέση: $1 - \frac{1}{K} \leq s(p) + s(p') \leq 1$. Ο OM_K πραγματοποιεί δύο ξεχωριστές τοποθετήσεις. Η μία ονομάζεται $partial\ match\ packing$, ενώ η άλλη είναι μία $Next\ Fit$ τοποθέτηση που περιλαμβάνει αντικείμενα $\leq \frac{1}{2}$. Πιο συγκεκριμένα, αν για το τρέχων αντικείμενο p ισχύει $s(p) \geq \frac{1}{2}$, τότε αυτό τοποθετείται σ' έναν νέο κάδο και ο κάδος αυτός περιλαμβάνεται στο σύνολο των $partial\ match\ packing$ κάδων. Αν $s(p) < \frac{1}{2}$ και υπάρχει $partial\ match\ packing$ κάδος Β που περιέχει ένα[5] συνταιριάζων αντικείμενο p', τότε τοποθέτειται το p στον κάδο Β, ο οποίος κλείνει μόνιμα. Αν $s(p) < \frac{1}{2}$ και δεν υπάρχει τέτοιος κάδος, τότε το p μεταφέρεται στους NF κάδους και τοποθετείται εκεί σύμφωνα με τον κανόνα του $Next\ Fit$.

Παράδειγμα: Επειδή ο ορισμός των OM_K μοιάζει κάπως πολύπλοκος, θα δούμε εδώ ένα παράδειγμα για την καλύτερη κατανόηση τους. Θα μελέτησουμε την συμπεριφορά του OM_3 απέναντι στη λίστα $L = \{0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8\}$. Καθώς $K = 3$, για να είναι δύο αντικείμενα $s(p)$ και $s(p')$ συνταιριάζοντα, πρέπει να ικανοποιούν τη σχέση $1 - \frac{1}{3} \leq s(p) + s(p') \leq 1 \Rightarrow 0.\overline{6} \leq s(p) + s(p') \leq 1$.

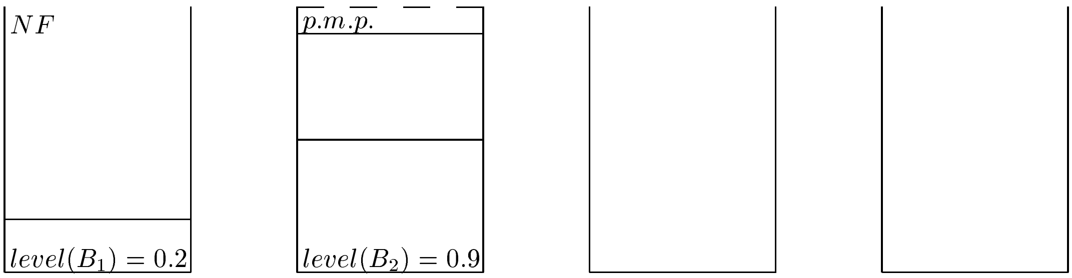


[5] Πρέπει να είναι μόνο ένα αντικείμενο μες στον κάδο.

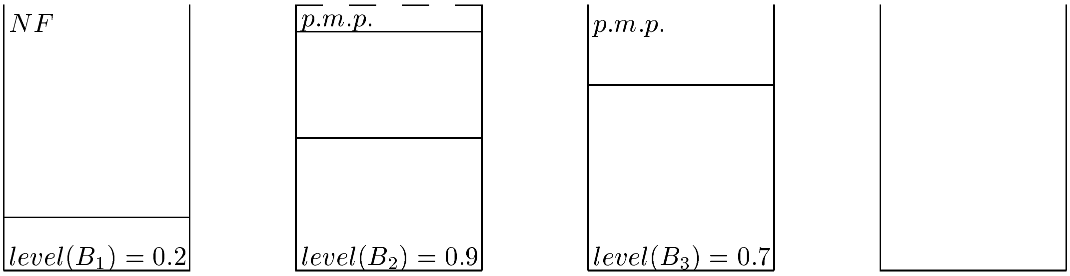
1ο ΒΗΜΑ Το $s(a_1) = 0.2$ είναι μικρότερο του $\frac{1}{2}$ και δεν υπάρχει $p.m.p.$ ($partial\ match\ packing$) κάδος με συνταιριάζων αντικείμενο. Επομένως ο κάδος B_1 στον οποίο τοποθετείται το a_1 γίνεται NF κάδος.



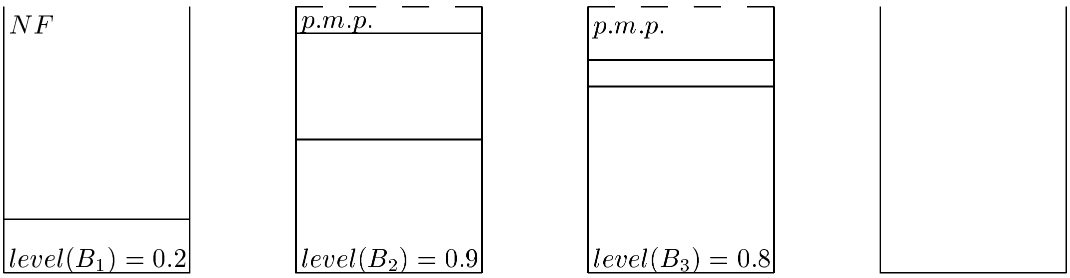
2ο ΒΗΜΑ Το $s(a_2) = 0.5$ είναι ίσο με το $\frac{1}{2}$. Έτσι τοποθετείται σε έναν καινούργιο κάδο, τον B_2, ο οποίος γίνεται $p.m.p.$ κάδος.



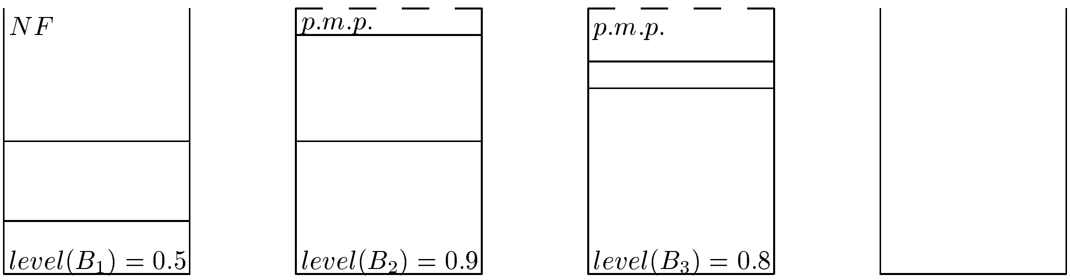
3ο ΒΗΜΑ Το $s(a_3) = 0.4$ είναι μικρότερο του $\frac{1}{2}$. Όμως $0.\overline{6} \leq s(a_3) + s(a_2) \leq 1$. Υπάρχει επομένως $p.m.p.$ κάδος με συνταιριάζων αντικείμενο για το $s(a_3)$ και είναι ο B_2. Έτσι το a_3 θα τοποθετηθεί στον B_2, ο οποίος θα κλείσει μόνιμα.



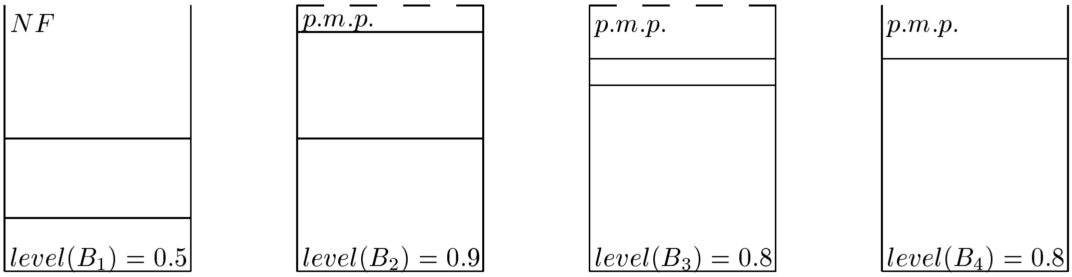
4ο ΒΗΜΑ Το $s(a_4) = 0.7$ είναι μεγαλύτερο του $\frac{1}{2}$. Επομένως θα ανοίξει ένας νέος κάδος (B_3), ο οποίος θα εισαχθεί στο σύνολο των $p.m.p.$ κάδων και στον οποίο θα τοποθετηθεί το a_4.



5ο ΒΗΜΑ Το $s(a_5) = 0.1$ είναι μικρότερο του $\frac{1}{2}$. Όμως $0.\overline{6} \leq s(a_5) + s(a_4) \leq 1$. Υπάρχει επομένως $p.m.p.$ κάδος με συνταιριάζων αντικείμενο για το a_5 και είναι ο B_3. Έτσι το a_5 θα τοποθετηθεί στον B_3, ο οποίος θα κλείσει μόνιμα.



6ο ΒΗΜΑ Το $s(a_6) = 0.3$ είναι μικρότερο του $\frac{1}{2}$. Όμως δεν υπάρχει ανοιχτός *p.m.p.* κάδος με συνταιριάζων αντικείμενο για το a_6. Έτσι αυτό θα τοποθετηθεί σε κάποιον NF κάδο σύμφωνα με τον ομόνυμο *packing rule*. Επομένως το a_6 εισέρχεται στον B_1 κάδο.



7ο ΒΗΜΑ Το $s(a_7) = 0.8$ είναι μεγαλύτερο του $\frac{1}{2}$. Επομένως θα ανοίξει ένας νέος κάδος (B_4), ο οποίος θα εισαχθεί στο σύνολο των *p.m.p.* κάδων και στον οποίο θα τοποθετηθεί το a_7.

Για τη συγκεκριμένη λίστα L, ο OM_3 χρησιμοποιεί 4 κάδους.

Οι αλγόριθμοι OM_K πληρώνουν το τίμημα της πολύ καλής κατά μεσό όρο απόδοσής τους, στις επιδόσεις τους απέναντι στις χειρότερες περιπτώσεις. Έχει αποδειχτεί ότι το ασυμπτωτικό τους πηλίκο προσέγγισης χειρότερης επίδοσης, είναι $R^{\infty}_{OM_K} = 2$. Για τις μέσες επιδόσεις τους αποδείχτηκε $[CL91]$ ότι:

Θεώρημα 5.3.1.1 $\overline{R}^{\infty}_{OM_K}(U[0,1]) = 1 + c\sqrt{\frac{K}{n}} + O(\frac{1}{K})$, όπου c μία σταθερά.

Από το παραπάνω θεώρημα φαίνεται ότι $lim_{K\to\infty}\overline{R}^{\infty}_{OM_K}(U[0,1]) = 1$.

Το 1984 οι *Bentley*, *Johnson*, *Leighton* και *McGeoch* $[BJL84]$ απέδειξαν το παρακάτω εντυπωσιακό αποτέλεσμα:

Θεώρημα 5.3.1.2 $\overline{R}^{\infty}_{FF}(U[0,1]) = 1$

Βέβαια το $\overline{R}_{FF}^{\infty}(U[0,1])$ πέφτει κάτω του 1.01 για n που ξεπερνάει το 250.000. Το ερώτημα είναι πόσο γρήγορα το $\overline{R}_{FF}^{\infty}(U[0,1])$ τείνει στο 1. Αυτό μπορεί να απαντηθεί αν στρέψουμε την προσοχή μας στο *expected wasted space* του FF. Η απόδειξη του παρακάρω θεωρήματος αναλύεται στα $[Sho86][CJS95]$:

Θεώρημα 5.3.1.3 $\overline{W}_{FF}^{n}(U[0,1]) = \Theta(n^{\frac{2}{3}})$

Εδώ ο *Best Fit* είναι πιο αποτελεσματικός $[Sho86][LS89]$:

Θεώρημα 5.3.1.4 $\overline{W}_{BF}^{n}(U[0,1]) = \Theta(\sqrt{n}log^{\frac{3}{4}}n)$

Οι αποδείξεις των παραπάνω δύο θεωρημάτων συνδέονται άμεσα με τη θεωρία του *stochastic planar matching*, η οποία έχει μία εκπληκτική ποικιλία εφαρμογών. Η θεμελίωση της θεωρίας αυτής έγινε από τους *Ajtai*, *Komlos* και *Tusnady* $[AKT84]$.

Να σημειώσουμε εδώ, ότι στην μελέτη μέσων επιδόσεων έχει σημασία αν ένας *online* αλγόριθμος γνωρίζει το μεγεθος n της λίστας εισόδου από την αρχή της εφαρμογής του. Αν το γνωρίζει, ονομάζεται *closed online*, ενώ αν δεν το γνωρίζει ονομάζεται *open online* αλγόριθμος.

Θεώρημα 5.3.1.5 Για κάθε *open online* αλγόριθμο A, ισχύει $[Sho86][Sho91]$:

$$\overline{W}_{A}^{n}(U[0,1]) = \Omega(\sqrt{nlogn})$$

Αυτό είναι και το χαμηλότερο δυνατό όριο για έναν *open online* αλγόριθμο. Για τους *closed online* αλγόριθμους ισχύει η σχέση:

$$\overline{W}_{A}^{n}(U[0,1]) = \Omega(\sqrt{n})$$

Όμως ουσιαστικά είναι τελείως ασήμαντη, καθώς το όριο αυτό, είναι φανερό ότι ισχύει και για έναν *optimal* αλγόριθμο. Κάποιο άλλο πιο στενό ορίο δεν έχει αποδεικτεί μέχρι στιγμής.

5.3.2 ΔΙΑΚΡΙΤΕΣ ΚΑΤΑΝΟΜΕΣ

Η βασική διαφορά των διακριτών ομογενών κατανομών από τις συνεχείς, είναι ότι για συγκεκριμένους συνδιασμούς των j και k μπορούμε να πετύχουμε $\overline{W}_{BF}^{n}(U[j,k]) = O(1)$. Δηλαδή το *expected waste* μπορεί να γίνει αρκετά χαμηλό και ανεξάρτητο του n, κάτι που είναι αδύνατο να συμβεί για συνεχείς ομογενείς κατανομές.

Ας δούμε όμως πρώτα πως συμπεριφέρονται οι FF και BF για διακριτές κατανομές ανάλογες των συνεχών $U[0,1]$. $[CCG91][CJS95]$:

Θεώρημα 5.3.2.1 Για $k > 1$,

A. $\overline{W}^n_{FF}(U\{k,k\}) = \Theta(\sqrt{nk})$

B. $\overline{W}^n_{FF}(U\{k,k\}) = O(\sqrt{n}logk)$

Επομένως για σταθερό k, οι BF και FF έχουν $\Theta(\sqrt{n})$ *expected waste*. Κάτι τέτοιο είναι αδύνατο για οποιονδήποτε *open online* αλγόριθμο υπό $U[0,1]$ κατανομή.

Θεώρημα 5.3.2.2 Για κάθε *open online* αλγόριθμο A ισχύει $[CCG91][CJS95]$:

$\overline{W}^n_{A}(U\{k,k\}) = \Omega(\sqrt{nlogk})$

Τα δύο παραπάνω θεωρήματα ισχύουν και για $j = k - 1$. Αυτό συμβαίνει γιατί η μόνη διαφορά μεταξύ των $U\{k,k\}$ και $U\{k-1,k\}$, είναι η παρουσία των αντικειμένων $\frac{k}{k} = 1$. Αυτά όμως γεμίζουν έναν κάδο το καθένα, χωρίς να προκαλούν επιπλέον *wasted space*.

Όμως για $j = k - 2$ τα όρια αλλάζουν. Για τον *Best Fit*, για παράδειγμα, ισχύει το εξής θεώρημα $[CJS93][KRS96]$:

Θεώρημα 5.3.2.3 Για $k > 2$, $\overline{W}^n_{BF}(U\{k-2,k\}) = O(1)$

Φαίνεται λοιπόν ο BF να έχει αρκετά καλές μέσες επιδόσεις για τις συγκεκριμένες διακριτές κατανομές. Μέχρι πρόσφατα, εκείνο που ήταν αποδεδειγμένο για τον *First Fit* ήταν:

$\overline{W}^n_{FF}(U\{k-2,k\}) = \Omega(n^{1/2})$

Όμως το 1998 οι $Albers$ και $Mitzenmacher$ [$MA98$] απέδειξαν ότι, για κατανομές $U\{k-2,k\}$, ο FF παρουσιάζει την ίδια σταθερότητα με τον BF. Θα αναφερθούμε πιο συγκεκριμένα για την προσπάθεια τους σε επόμενο κεφάλαιο.

Να σημειώσουμε εδώ ότι, παρ' όλο που οι BF και FF έχουν την ίδια χειρότερη επίδοση, πρόσφατες έρευνες έχουν δείξει ότι ο BF υπερτερεί του FF σε μέσες επιδόσεις.

Για κατανομές $U\{j,k\}$ με $j \le k-3$, η συμπεριφορά του BF δεν είναι τόσο καλή, καθώς αρχίζει να συμπεριφέρεται όπως στις συνεχείς κατανομές όπου το $\overline{W}^n_{BF}(U[0,b])$ έχει γραμμική συμπεριφορά για κάθε $b \in (0,1)$. Έχει γίνει η υπόθεση ότι για κάθε b υπάρχει ένας ακέραιος K_b, τέτοιος ώστε το $\overline{W}^n_{BF}(U\{j,k\})$ να είναι γραμμικό, όταν $k > K_b$ και $|b-\frac{j}{k}| < min\{\frac{b}{2}, \frac{1-b}{2}\}$. Η υπόθεση αυτή έχει επιβεβαιωθεί από πολλές έρευνες και προσομειώσεις. Τα αποτελέσματα μερικών απ' αυτές φαίνονται στο παρακάτω σχήμα:

j k	5	6	7	8	9	10	11	12	13	14
3	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ
4		Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ
5			Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ
6				Φ	Φ	Φ	Φ	Φ	Φ	Φ
7					Φ	Φ	Φ	Φ	Φ	Φ
8						Φ	Γ	Φ*	Φ*	Φ
9							Φ	Γ	Γ*	Φ*
10								Φ	Γ*	Γ*
11									Φ	Γ*
12										Φ

$\overline{W}^n_{BF}(U\{j,k\})$, για $j < k-1$.

Φ = Φραγμένο *wasted space*, Γ = Γραμμικό *wasted space*,
$*$ = δεν έχει αποδεικτεί.

Λογικό είναι, για αρκετά μικρές τιμές του j, το *expected waste* να είναι ιδιαίτερα ικανοποιητικό και ανεξάρτητο του n για πολλών ειδών κατανομές, ακόμα και για τον *First Fit* [$CCG91$]:

Θεώρημα 5.3.2.4 Για τους BF και FF έχουμε:

$A.\quad \overline{W}^n_{BF}(U\{j,k\}) = O(1), \quad$ όταν $k \geq \frac{j(j+3)}{2}$

$B.\quad \overline{W}^n_{FF}(U\{j,k\}) = O(1), \quad$ όταν $k \geq j^2$

Γενικά, τα όρια του *expected waste*, υπό διακριτές κατανομές, είναι εξαιρετικά καλά για έναν *open online* αλγόριθμο. Αυτό δηλώνει και το παρακάτω Θεώρημα $[CCG91]$:

Θεώρημα 5.3.2.5 Για κάθε διακριτή κατανομή $F = U\{j,k\}$ όπου $1 \leq j < k-1$, υπάρχει ένας *open online* αλγόριθμος A_F, τέτοιος ώστε $\overline{W}^n_{A_F}(F) = O(1)$.

Αν συγκρίνουμε το παραπάνω θεώρημα με το θεώρημα 5.3.1.5, βλέπουμε ότι για διακριτές κατανομές, έχουμε πολύ καλύτερες συμπεριφορές από ότι για αντίστοιχες συνεχείς.

5.4 OFFLINE ΑΛΓΟΡΙΘΜΟΙ

Γνωρίζοντας τις πολύ καλές, σε γενικές γραμμές, μέσες αποδόσεις των *online* αλγόριθμων, αναρωτιέται κανείς αν υπερτερούν αισθητά οι αντίστοιχοι $offline$.

Πάντως οι $First\ Fit\ Decreasing$ και $Best\ Fit\ Decreasing$, υπερτερούν αισθητά των $First\ Fit$ και $Best\ Fit$ για συνεχείς ομοιόμορφες κατανομές. Ένα πρώτο[6] αποτέλεσμα που το αποδεικνύει αυτο φαινεται παρακάτω $[Kno81][Lue82]$:

Θεώρημα 5.4.1 Αν $A \in \{FFD, BFD\}$, τότε:

$\overline{W}^n_A(U[0,1]) = \Theta(\sqrt{n})$

Παρατηρούμε μάλιστα ότι οι BFD και FFD ξεπερνούν τον καλύτερο δυνατό *open online* αλγόριθμο, κατά ένα παράγοντα $\sqrt{logn}$.

Για $U[0,b]$ κατανομές, με $0 < b < 1$, το πλεονέκτημα των BFD και FFD αυξάνεται θεαματικά έναντι των *online* ισοδύναμων τους. Ωστόσο, αυτό έχει

[6] Πρώτο όχι μόνο στην περιγραφή μας, αλλά και ιστορικά.

αποδεικτεί μόνο για τον FFD $[BJL84]$:

Θεώρημα 5.4.2 Για κατανομές τύπου $U[0,b]$ ισχύει:

$\overline{W}^n_{FFD}(U[0,b]) = \Theta(1)$,αν $b \leq \frac{1}{2}$

$\overline{W}^n_{FFD}(U[0,b]) = \Theta(n^{1/3})$,αν $\frac{1}{2} < b < 1$

Οι $Floyd$ και $Karp$ $[FK91]$, κάνοντας κάποιες τροποποιήσεις στον FFD, απέδειξαν ότι υπάρχει $offline$ αλγόριθμος, ο οποίος ,για κατανομές $U[0,b]$, επιτυγχάνει να έχει πολύ χαμηλό *wasted space*.

Θεώρημα 5.4.3 Υπάρχει $offline$ αλγόριθμος A, $O(nlogn)$ χρόνου, ώστε:

$\overline{W}^n_{A}(U[0,b]) = O(1)$,αν $\frac{1}{2} < b < 1$

Για διακριτές κατανομές, τα πλεονεκτήματα των FFD και BFD δεν είναι τόσο ξεκάθαρα, καθώς οι αποδόσεις των $online$ αλγόριθμων είναι αρκετά ικανοποιητικές. Παρ' όλα αυτά, το γεγονός ότι ανήκουν στην κατηγορία των $offline$ αλγόριθμων τους κάνει να υπερτερούν, έστω και σε χαμηλότερο βαθμό, από τους αντίστοιχους $online$.

Κεφάλαιο 6

APPROXIMATION SCHEMES

Στα προηγούμενα κεφάλαια μελετήσαμε ένα πλήθος αλγορίθμων. Κατά την μελέτη αυτή έγινε κατανοητό ότι είναι δύσκολο πολλές φορές, να πετύχουμε καλές αποδόσεις σε εξίσου καλούς χρόνους. Για παράδειγμα, κάποιοι *bounded-space* αλγόριθμοι, αν και μπορούν να εφαρμοστούν σε γραμμικούς χρόνους, δεν έχουν τις καλύτερες δυνατές επιδόσεις. Κάτι τέτοιο άλλωστε είναι απόλυτα λόγικο, καθώς ένας αλγόριθμος που αποδίδει αρκετά καλά, το πιο πιθανό είναι να έχει έναν πολύπλοκο μηχανισμό λειτουργίας και να εξετάζει εξαντλητικά όλους τους χρησιμοποιήμενους κάδους, χωρίς να κλείνει κανέναν από αυτούς μόνιμα.

Συνεπώς, η απόδοση και ο χρόνος εφαρμογής είναι δύο μεγέθη, όπου μία καλή τιμή του ενός ισοδυναμεί με μία όχι τόσο καλή τιμή του άλλου. Παρακάτω θα μελετήσουμε κάποιους προσεγγίστικους αλγόριθμους, στους οποίους φαίνεται, με ακριβείς σχέσεις, το χρονικό κόστος που πληρώνουμε για να προσσεγγίσουμε τη βέλτιστη λύση.

6.1 ΟΡΙΣΜΟΙ

Approximation Scheme Έστω Π ένα NP-$Hard$ πρόβλημα με αντικειμενική συνάρτηση f_Π. Λέμε ότι ένας αλγόριθμος A είναι *approximation scheme* (προσεγγιστικό σχήμα) για το Π, αν σε είσοδο (I, ε), όπου I είναι ένα στιγμιότυπο του Π και $\varepsilon > 0$ είναι η παράμετρος σφάλματος, αποδίδει λύση s ώστε:

$f_\Pi(I, s) \leq (1+\varepsilon)OPT$, αν το Π είναι πρόβλημα ελαχιστοποίησης.

$f_\Pi(I,s) \geq (1-\varepsilon)OPT$, αν το Π είναι πρόβλημα μεγιστοποίησης,

OPT είναι το κόστος της βέλτιστης λύσης.

Polynomial Time Approximation Scheme (PTAS) Αν για $\varepsilon > 0$ έχουμε $(1+\varepsilon) - approximation$[1] αλγόριθμους A_ε για το πρόβλημα Π, των οποίων ο χρόνος εφαρμογής είναι πολυωνυμικός για καθορισμένο ε, τότε αυτοί ονομάζονται *Polynomial Time Approximation Scheme* για το Π.

Fully Polynomial Time Approximation Scheme (FPTAS) Ένας αλγόριθμος $PTAS$ ονομάζεται *Fully Polynomial Time Approximation Scheme*, αν ο χρόνος εφαρμογής του είναι πολυωνυμικός τόσο στο μεγέθος της εισόδου, όσο και στον παράγοντα $\frac{1}{\varepsilon}$.

Επομένως οι $FPTAS$ αλγόριθμοι είναι υποκατηγορία των $PTAS$. Ένας $PTAS$ αλγόριθμος, για παράδειγμα, μπορεί να τρέχει σε $O(n^{\frac{1}{\varepsilon}})$, ενώ ένας $FPTAS$ σε χρόνο $O(n^2\frac{1}{\varepsilon})$.

6.2 ΕΝΑ ΑΣΥΜΠΤΩΤΙΚΟ PTAS

Ας ερθουμε τώρα στο πρόβλημα του *Bin Packing*. Το παρακάτω θεώρημα αποδεικνύει ουσιαστικά ότι δεν υπάρχει $PTAS$ για το πρόβλημά μας.

Θεώρημα 6.2.1 Για $\varepsilon > 0$, δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου, με απόδοση κάτω του $\frac{3}{2} - \varepsilon$ για το πρόβλημα του *Bin Packing*, υποθέτοντας ότι $P \neq NP$.

Απόδειξη Θα χρησιμοποιήσουμε το NP-πλήρες πρόβλημα $PARTITION$: Ας υποθέσουμε ότι έχουμε ένα σύνολο $A = \{a_1, a_2, \ldots, a_n\}$ από θετικούς αριθμούς. Υπάρχουν δύο συμπληρωματικά υποσύνολα A_1, A_1' τέτοια ώστε $\sum_{a_i \epsilon A_1} a_i = \sum_{a_j \epsilon A_1'} a_j$;

Έστω ότι $M = \sum_{i=1}^{n} a_i$. Υποθέτουμε ότι έχουμε ένα *Bin Packing* πρόβλημα με είσοδο τα αντικείμενα $\frac{2a_1}{M}, \ldots, \frac{2a_n}{M}$. Παρατηρούμε ότι η απάντηση στο $PARTITION$ πρόβλημα είναι "ναι", όταν και μόνο όταν το αντίστοιχο στιγμιότυπο του *Bin Packing* μπορεί να λυθεί με 2 κάδους.

[1]Ένας αλγόριθμος A είναι $(1+\varepsilon) - approximation$, όταν ισχύει $R_A = 1 + \varepsilon$.

Επομένως, αν υπάρχει αλγόριθμος πολυωνυμικού χρόνου που να επιστρέφει λύση κάτω των τρίων κάδων, δηλαδή κάτω του $\frac{3}{2}$ της *optimal* λύσης, που στο συγκεκριμένο παράδειγμα είναι 2, τότε και η απάντηση στο $PARTITION$ θα είναι "ναι". Δηλαδή θα υπάρχει αλγόριθμος πολυωνιμικού χρόνου για το $PARTITION$. Αυτό όμως συμβαίνει μόνο αν $P = NP$.

Βέβαια η παραπάνω απόδειξη στηρίζεται στο γεγονός ότι η βέλτιστη λύση είναι μόλις 2 κάδοι, παρ' όλο που ο αριθμός των αντικειμένων δεν περιορίζεται και θα μπορούσε, σε κάποιες περιπτώσεις, να είναι ιδιαίτερα μεγάλος. Τί συμβαίνει όμως σε παραδείγματα, όπου η βέλτιστη λύση αυξάνει ως προς n; Παρακάτω θα παρουσιάσουμε έναν αλγόριθμο που δεν είναι $PTAS$, σύμφωνα με τον αυστηρό ορισμό του τελευταίου, αλλά που ασυμπτοτικά τείνει να γίνει.

Θεώρημα 6.2.2 Για κάθε ε, όπου $0 < \varepsilon \leq \frac{1}{2}$, υπάρχει αλγόριθμος A_ε, που τρέχει σε πολυωνιμικό χρόνο ως προς n και χρησιμοποιεί το πολύ $(1 + 2\varepsilon)OPT + 1$ κάδους.

Η οικογένεια των αλγορίθμων A_ε αποτελούν ένα ασυμπτοτικό προσεγγιστικό σχήμα πολυωνυμικού χρόνου (*asymptotic polynomial time approximation scheme*), καθώς για κάθε $\varepsilon > 0$, υπάρχει $N > 0$ και ένας αλγόριθμος πολυωνυμικού χρόνου, έστω Β, που ανήκει σ' αυτήν την οικογένεια, ο οποίος παρουσιάζει απόδοση $1 + \varepsilon$ για όλα τα παραδείγματα για τα οποία $OPT \geq N$. Παρ' όλα αυτά, οι αλγόριθμοι A_ε δεν προσφέρουν την πιο πρακτική δυνατή λύση, καθώς ο χρόνος στον οποίο τρέχουν είναι ιδιαίτερα υψηλός. Θα αποδείξουμε το παραπάνω θεώρημα σε τρία βήματα.

Λήμμα 6.2.3 Ας υποθέσουμε ότι το $\varepsilon > 0$ είναι σταθερό και έστω Κ ένας επίσης σταθερός μη αρνητικός ακέραιος. Έστω ότι τα αντικείμενα έχουν μέγεθος τουλάχιστον ε και ο αριθμός των πιθανών μεγεθών που μπορούν να πάρουν είναι Κ. Τότε υπάρχει αλγόριθμος πολυωνυμικού χρόνου που προσφέρει τη βέλτιστη λύση για το συγκεκριμένο πρόβλημα.

Απόδειξη: Ο αριθμός των αντικειμένων που μπορούν να ταιριάξουν σε ένα κάδο είναι το πολύ $\lfloor \frac{1}{\varepsilon} \rfloor$. Έστω ότι $\lfloor \frac{1}{\varepsilon} \rfloor = M$. Ο αριθμός των διαφορετικών μορφών[2] που μπορεί να πάρει ένας κάδος είναι το πολύ $R = \binom{M+K}{M}$, δηλαδή ένας επαναληπτικός συνδυασμός του Κ ανά Μ. Το R είναι μία σταθερά αρκετά μεγάλου μεγέθους. Αν ο συνολικός αριθμός κάδων που χρησιμοποιούνται είναι

[2]Δύο κάδοι έχουν την ίδια μορφή όταν περιέχουν ακριβώς τα ίδια αντικείμενα.

το πολύ n, τότε ο αριθμός των διαφορετικών πιθανών τοποθετήσεων, που μπορούν να αποτελέσουν την έξοδο του προβλήματος, είναι το πολύ[3], $P = \binom{n+R}{R}$, που είναι πολυωνυμικό ως προς n. Απαριθμώντας τις και διαλέγοντας την καλύτερη τοποθέτηση, παίρνουμε τη βέλτιστη λύση.

Λήμμα 6.2.4 Έστω $\varepsilon > 0$ σταθερό. Έστω επίσης ότι τα αντικείμενα της εισόδου έχουν μέγεθος τουλάχιστον ε. Υπάρχει προσεγγιστικός αλγόριθμος πολυωνυμικού χρόνου που λύνει το συγκεκριμένο πρόβλημα με απόδοση όχι πέρα του $(1+\varepsilon)$.

Απόδειξη: Ας συμβολίσουμε με I το συγκεκριμένο στιγμιότυπο με τον παραπάνω περιορισμό. Ταξινομούμε τα αντικείμενα κατά αύξουσα σειρά μεγέθους και τα χωρίζουμε σε $K = \lceil \frac{1}{\varepsilon^2} \rceil$ ομάδες, κάθε μία από τις οποίες έχει το πολύ $Q = \lfloor n\varepsilon^2 \rfloor$ αντικείμενα. Δύο διαφορετικές ομάδες μπορεί να περιέχουν αντικείμενα του ίδιου μεγέθους.

Δημιουργούμε ένα καινούργιο στιγμιότυπο, έστω J, κάνοντας το μεγέθος κάθε αντικειμένου ίσο με το μέγεθος του μεγαλύτερου αντικειμένου της ομάδας του. Το νέο αυτό στιγμιότυπο περιέχει K το πολύ διαφορετικά μεγέθη. Επομένως από το Λήμμα 6.2.3 μπορούμε να βρούμε μία βέλτιστη λύση για το J. Η λύση όμως αυτή είναι εφικτή και για τα μεγέθη του στιγμιότυπου I. Θα δείξουμε παρακάτω ότι $OPT(J) \leq (1+\varepsilon)OPT(I)$.

Για να το πετύχουμε αυτό θα δημιουργήσουμε ένα νέο στιγμιότυπο, έστω J', κάνοντας το μέγεθος κάθε αντικειμένου του I ίσο με το με το μέγεθος του μικρότερου αντικειμένου της ομάδας του. Είναι προφανές ότι $OPT(J') \leq OPT(I)$.

Αν αφαιρούσαμε τα Q πρώτα αντικείμενα του J' και τα Q τελευταία του J, τότε για τις παραγόμενες λίστες που θα προέκυπταν θα ίσχυε $OPT(J) \leq OPT(J')$. Αυτό θα συνέβαινε γιατί κάθε αντικείμενο του J' θα ήταν λίγο μεγαλύτερο από το αντίστοιχο στην κατάταξη αντικείμενο του J. Αν τώρα προσθέσουμε και τα αντικείμενα που αφαιρέσαμε, τότε θα στοίχιζε στον J, στην χειρότερη περίπτωση Q παραπάνω κάδους. Επομένως η βέλτιστη λύση του J δεν ξεπερνάει ποτέ τους $OPT(J') + Q$ κάδους. Δηλάδη:

$OPT(J) \leq OPT(J') + Q \leq OPT(I) + Q$

[3]Επαναληπτικός συνδυασμός του n ανά R.

Όμως, καθώς κάθε αντικείμενο του I έχει μέγεθος τουλάχιστον ε, ισχύει $OPT(I) \geq n\varepsilon$. Επομένως:

$n\varepsilon^2 \leq \varepsilon OPT(I) \Rightarrow Q = \lfloor n\varepsilon^2 \rfloor \leq \varepsilon OPT(I)$

Από τις δύο τελευταίες ανισότητες καταλήγουμε στο εξής:

$OPT(J) \leq (1+\varepsilon)OPT(I)$

Απόδειξη Θεωρήματος 6.2.2: Ας υποθέσουμε ότι έχουμε μία τυχαία είσοδο L. Αν αφαιρέσουμε απο αυτήν τα αντικείμενα με μέγεθος μικρότερο του ε, τότε προκύπτει η L'. Όπως αποδείξαμε στο Λήμμα 5.2.4, υπάρχει λύση για την L', όπου χρησιμοποιούνται το πολύ $(1+\varepsilon)OPT(L')$ κάδοι. Αφού την εφαρμόσουμε, τοποθετούμε τα τα μικρά αντικείμενα ($< \varepsilon$) σύμφωνα με τον *packing rule* του *First Fit*, στους κάδους που είναι ήδη ανοικτοί από την τοποθέτηση των αντικειμένων του L'. Αν κάποιο από αυτά δεν ταιριάζει στους ήδη ανοικτούς κάδους, θα ανοίξουν καινούργιοι κάδοι.

Στην περίπτωση που δεν ανοίξει καινούργιος κάδος, απαιτούνται το πολύ $(1+\varepsilon)OPT(L') \leq (1+\varepsilon)OPT(L)$ κάδοι. Αν υπάρξει κάποιος πρόσθετος κάδος, τότε έστω ότι οι συνολικοι κάδοι είναι Μ. Είναι προφανές ότι οι $M-1$ πρώτοι κάδοι έχουν *level* τουλάχιστον $1-\varepsilon$. Επομένως το άθροισμα των αντικειμένων της εισόδου L είναι τουλάχιστον $(M-1)(1-\varepsilon)$. Άρα:

$(M-1)(1-\varepsilon) \leq \sum_{i=1}^{n} s(a_i) \leq OPT(L) \Rightarrow M \leq \dfrac{OPT(L)}{1-\varepsilon} + 1 \leq (1+2\varepsilon)OPT(L) + 1$

όπου για να καταλήξουμε στο τελευταίο αποτέλεσμα χρησιμοποιήσαμε τη σχέση $\varepsilon \leq \frac{1}{2}$. Επομένως, για κάθε ε τέτοιο ώστε $0 < \varepsilon \leq \frac{1}{2}$, υπάρχει αλγόριθμος πολυωνυμικού χρόνου με απόδοση το πολύ $(1+2\varepsilon)OPT(L)+1$ κάδους.

Ο *packing rule* του αλγόριθμου(A_ε), στον οποίο καταλήξαμε, είναι ο εξής:

$\mathbf{A}_\varepsilon$ packing rule :

1. Παράβλεψε τα αντικείμενα μεγέθους $< \varepsilon$.
2. Στρογγυλοποίησε τα μεγέθη των υπολοίπων για να υπάρχει σταθερός αριθμός μεγεθών (Λήμμα 5.2.4).

3. Βρες την *optimal* τοποθέτηση (Λήμμα 5.2.3).
4. Χρησιμοποίησέ την για τα αντικείμενα με τα κανονικά μεγέθη.
5. Τοποθέτησε τα αντικείμενα μεγέθους $< \varepsilon$ σύμφωνα με τον $First\ Fit$.

Κεφάλαιο 7

ΔΥΟ ΠΡΟΣΦΑΤΟΙ ΑΛΓΟΡΙΘΜΟΙ

7.1 ΑΛΥΣΙΔΕΣ MARKOV

7.1.1 ΟΡΙΣΜΟΣ

Ένα σύνολο από τυχαίες μεταβλητές $\{Q_n\}$ αποτελούν μία αλυσίδα *Markov* όταν η πιθανότητα η επόμενη τιμή (κατάσταση) να είναι ίση με x_{n+1} εξαρτάται μονάχα από την παρούσα τιμή (κατάσταση) x_n και όχι από οποιαδήποτε άλλη τιμή του παρελθόντος.

Η ιδιότητα αυτή είναι γνωστή σαν έλλειψη μνήμης και ισχύει για οποιαδήποτε χρονική στιγμή. Η συνθήκη αυτή επιβάλλει ότι το χρονικό διάστημα μεταξύ διαδοχικών αλλαγών κατάστασης ακολουθεί μία κατανομή πιθανότητας που εξασφαλίζει την έλλειψη μνήμης. Η κατανομή αυτή πρέπει να είναι η γεωμετρική κατανομή για διακριτό χρόνο ή η εκθετική κατανομή για συνεχή χρόνο.

7.1.2 ΠΡΟΣΕΓΓΙΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Τα τελευταία χρόνια ξεκίνησε ένας νέος τρόπος προσσέγγισης του *Bin Packing*. Αυτός δεν είναι άλλος από τις αλυσίδες *Markov*. Πολλές από τις πιο πρόσφατες μελέτες, μερικές από τις οποίες θα μελετήσουμε στα επόμενα κεφάλαια, έχουν στηριχθεί καθαρά στις αλυσίδες *Markov* με εξαιρετικά επιτυχή αποτελέσματα. Ας δούμε όμως με ποιό τρόπο χρησιμοποιούνται στα πλαίσια του προβλήματος που μελετάμε.

Οι αλυσίδες $Markov$ επιλέχτηκαν για την παράσταση των στιγμιοτύπων του προβλήματος κατά την εφαρμογή ενός αλγόριθμου. Η χρήση τους γίνεται για διακριτές κατανομές. Όμως, όπως έχουμε δει σε προηγούμενο κεφάλαιο, μία διακριτή κατανομή $U\{j,k\}$ προσεγγίζει την συνεχή $U[0,a]$, αν $j = ak$ και $k \to \infty$. Επομένως η χρήση τους μπορεί να επεκταθεί και για συνεχείς κατανομές.

Όταν έχουμε σαν είσοδο μία διακριτή κατανομή $U\{j,k\}$ είναι ακριβώς το ίδιο με το να έχουμε αντικείμενα που παίρνουν τιμές από το συνολο $\{1,2,\ldots,j\}$ και τα οποία πρέπει να τοποθετήσουμε σε κάδους χωρητικότητας k.

Κατά την εκτέλεση ενός αλγόριθμου, η διαδικασία χωρίζεται σε χρονικά στιγμιότυπα. Η μετάβαση από το ένα στιγμιότυπο στο άλλο γίνεται με την τοποθέτηση του επόμενου κατά σειρά αντικειμένου. Κάθε στιγμιότυπο απεικονίζεται ως εξής:

$$s(t) = (s_1(t), s_2(t), \ldots, s_{k-1}(t))$$

Η μεταβλητή $s_i(t)$ συμβολίζει τον αριθμό των κάδων που έχουν αχρησιμοποίητο χώρο i. Εκείνους δηλαδή τους κάδους, έστω B_j, για τους οποίους ισχύει: $1 - level(B_j) = i$. Ο αχρησιμοποίητος αυτός χώρος[1] μπορεί να είναι από 1 έως $k-1$.

Θα δούμε εδώ πως μεταβάλλονται τα στιγμιότυπα της διαδικασίας κατά την εφαρμογή του $Best\ Fit$. Η διαδικασία ξεκινάει με το στιγμιότυπο[2] $s(0) = (0,0,\ldots,0)$ το οποίο δηλώνει ότι δεν υπάρχουν ανοικτοί κάδοι. Έστω ότι φτάνουμε στην χρονική στιγμή t και το επόμενο αντικείμενο που πρέπει να τοποθετηθεί, έχει μέγεθος d. Ας υποθέσουμε εδώ ότι ο i είναι ο χαμηλότερος δείκτης για τον οποίο ισχύει $i \geq d$. Είναι προφανές από τον ορισμό του, ότι ο BF θα τοποθετήσει το αντικείμενο αυτό σ′ έναν κάδο με διαθέσιμο χώρο ίσο με i. Επομένως στο επόμενο στιγμιότυπο θα έχουμε $s_i(t+1) = s_i(t) - 1$. Αν $i > d$ τότε $s_{i-d}(t+1) = s_{i-d}(t) + 1$, αλλιώς αν $i = d$ κλείνει ο συγκεκριμένος κάδος και δεν αλλάζει κανένας περαιτέρω όρος. Σε περίπτωση που δεν υπάρχει $i \geq d$, δηλαδή δεν υπάρχει ανοικτός κάδος στον οποίο να ταιριάζει το αντικείμενο d, τότε ανοίγει ένας καινούργοις κάδος και έχουμε $s_{k-d}(t+1) = s_{k-d}(t) + 1$. Όλοι οι υπόλοιποι όροι παραμένουν σταθεροί. Αυτή η διαδικασία συνεχίζεται μέχρι την τοποθέτηση και του τελευταίου αντικειμένου.

[1] Έχουμε πει ότι ανάγουμε το πρόβλημά μας σ′ εκείνο με αντικείμενα μεγέθους $1,2,\ldots,j$ και κάδους χωρητικότητας k.

[2] Πριν τοποθετηθεί το πρώτο αντικείμενο.

Οι αλυσίδες $Markov$ αποδείχτηκαν εξαιρετικά χρήσιμες για αλγόριθμους των οποίων η μεθοδολογία στηρίζεται στο *level* των κάδων. Δηλαδή στο πόσο γεμάτος είναι κάθε κάδος σε κάθε στιγμιότυπο του προβλήματος. Ένας τέτοιος αλγόριθμος είναι και ο *Best Fit*, για τον οποίο έχουν αποδεικτεί πολλά νέα θεωρήματα μέσω αυτής της μεθοδολογίας.

Στη συνέχεια θα παρουσιάσουμε δύο νέους αλγόριθμους, οι οποίοι είναι αρκετά πιο πρόσφατοι από τους προηγούμενους. Ο πρώτος είναι ο *Random Fit* και ο δεύτερος ο $Sum\ Of\ Squares$. Παρουσιάστηκαν το 1998 και το 1999 αντίστοιχα.

7.2 RANDOM FIT

Ο *Random Fit* είναι ένας αλγόριθμος, ο οποίος επινοήθηκε για έναν πολύ συγκεκριμένο σκοπό. Αναφερθήκαμε πριν στις αλυσίδες $Markov$, έναν πολύ πρόσφατο τρόπο προσέγγισης του προβλήματος που έχει στεφθεί με φοβερή επιτυχία, καθώς οδήγησε στην απόδειξη πολλών νέων θεωρημάτων. Τα περισσότερα από τα θεωρήματα αυτά αφορούν τις μέσες επιδόσεις διάφορων αλγορίθμων υπό διακριτές αλλά και συνεχείς κατανομές.

Όμως, είπαμε ότι ο συγκεκριμένος τρόπος μεθοδολογίας είναι βολικός για αλγόριθμους με ορισμούς που βασίζονται στο *level* κάθε κάδου. Ο $First\ Fit$ αντίθετα δεν βασίζεται σε κάτι τέτοιο. Σημασία σ' αυτόν έχει η διάταξη των κάδων και όχι το *level* τους. Σε μία αναπαράσταση του προβλήματος με αλυσίδες $Markov$ δεν θα ήταν δυνατόν να βρούμε τον κάδο στον οποίο θα τοποθετηθεί το επόμενο αντικείμενο, κάθως δεν προσδιορίζεται από πουθενά ποιά είναι η διάταξη των κάδων.

Το 1996 η *Kenyon* [$KRS96$], χρησιμοποιόντας τις αλυσίδες $Markov$, έδειξε ότι ο *Best Fit* είναι σταθερός υπό διακριτές κατανομές της μορφής $U\{k-2,k\}$. Να σημειώσουμε εδώ ότι ένας αλγόριθμος είναι σταθερός υπό μία κατανομή, όταν το *expected waste* του δεν ξεπερνάει κάποιο όριο ακόμα κι όταν το n^3 τείνει στο άπειρο. Το 1998 ο $Mitzenmacher$ [$MA98$] κατόρθωσε να αποδείξει ότι το ίδιο ισχύει και για τον *First Fit*. Είχε σαν δεδομένο ότι αυτό ίσχυε για τον BF και καταφερε, μέσω του αλγόριθμου του *Random Fit*, να αποδείξει το ίδιο και για τον FF. Επομένως ο *Random Fit* αποτέλεσε το συνδετικό κρίκο για την απόδειξη ενός θεωρήματος που αφορά τον FF, μέσω

[3] Όπου n ο αριθμός των αντικειμένων της εισόδου.

των αλυσίδων $Markov$. Ας δούμε όμως ποιός είναι ο ορισμός του $Random$ $Fit(RF)$.

RF Packing Rule : Πριν την τοποθέτηση του επόμενου αντικειμένου συμβαίνουν τα εξής:

1. Οι ανοικτοί κάδοι αντιμετατίθονται εντελώς τυχαία και σχηματίζουν μία καινούργια διάταξη.

2. Τοποθετείται το επόμενο αντικείμενο σύμφωνα με τον *packing rule* του *First Fit*.

7.2.1 ΜΕΛΕΤΗ ΧΕΙΡΟΤΕΡΗΣ ΕΠΙΔΟΣΗΣ

Θεώρημα 7.2.1.1

Α. Για κάθε λίστα L, $RF(L) \leq 2OPT(L) - 1$.

Β. Υπάρχουν λίστες L, για τις οποίες ο $OPT(L)$ παίρνει αυθαίρετα μεγάλες τιμές, τέτοιες ώστε να υπάρχει μεγάλη πιθανότητα να ισχύει: $RF(L) = 2OPT(L) - 1$

Απόδειξη: Α. Σε κάθε χρονική στιγμή, οι κάδοι που χρησιμοποιεί ο RF αποτελούνται από το πολύ έναν κάδο, ο οποίος έχει αχρησιμοποίητο χώρο τουλάχιστον $\frac{1}{2}$. Επομένως ο αριθμός των κάδων που χρησιμοποιεί η *optimal* λύση είναι τουλάχιστον $\lfloor \frac{1}{2}RF(L) + 1 \rfloor$.

Β. Για κάθε $n \geq 2$, ας υποθέσουμε οτι έχουμε σαν λίστα εισόδου του προβλήματος, n αντικείμενα μεγέθους $\frac{1}{2}$, στα οποία παρεμβάλλονται n^2 αντικείμενα μεγέθους $\frac{1}{2n^3}$. Επομένως η λίστα αυτή είναι της μορφής:

$$L_n = \{\frac{1}{2}, \frac{1}{2n^3}, \ldots, \frac{1}{2n^3}, \frac{1}{2}, \frac{1}{2n^3}, \ldots, \frac{1}{2n^3}, \frac{1}{2}\}$$

Παρατηρούμε ότι το συνολικό άθροισμα των μικρών αντικειμένων, δηλαδή των $\frac{1}{2n^3}$, είναι:

$$\frac{1}{2n^3}n^2(n-1) = \frac{1}{2} - \frac{1}{2n} < \frac{1}{2}$$

Έτσι για την *optimal* λύση ισχύει: $OPT(L_n) = \lfloor \frac{n}{2} \rfloor + 1$. Θα δείξουμε ότι με μεγάλη πιθανότητα ο RF χρησιμοποιεί n κάδους για τη συγκεκριμένη λίστα.

Συγκεκριμένα, πριν την τοποθέτηση ενός μεγάλου αντικειμένου, η πιθανότητα ένας κάδος να περιέχει μεγάλο αντικείμενο και να μην περιέχει μικρό είναι φραγμένη από τον παράγοντα $(1-\frac{1}{n})^{n^2} \leq e^{-n}$. Συνεπώς, η πιθανότητα για κάθε μία από τις n τοποθετήσεις μεγάλων αντικειμένων, κάποιος ανοικτός κάδος που περιέχει ένα μεγάλο αντικείμενο να μην περιέχει ένα αντίστοιχο μικρό, είναι φραγμένη από το n^2e^{-n}. Καταλήγουμε ότι με πιθανότητα τουλάχιστον $1 - \frac{n^2}{e^n}$, ο RF χρησιμοποιεί n κάδους για τη λίστα L_n.

7.2.2 RANDOM-FIT(d)

Βλέπουμε ότι ο RF δεν πιάνει τις αποδόσεις των BF και FF. Έχει επιδόσεις παρόμοιες με αυτές του NF και του WF.

Θα αναφέρουμε εδώ μία παραλλαγή του RF, η οποία μελετήθηκε πιο πρόσφατα και ονομάζεται *Random-Fit(d)*[4]. Ο $RF(d)$, σε αντίθεση με τον RF, ξεπερνάει τις αποδόσεις των BF και FF, καθώς είναι αρκετά πιο αποτελεσματικός. Ας δούμε όμως τον ορισμό του:

RF(d) Packing Rule : Για κάθε τρέχων αντικείμενο γίνεται τυχαία αντιμετάθεση των κάδων και σχηματίζεται έτσι μία νέα διάταξη. Στη συνέχεια για τους πρώτους d κάδους στους οποίους ταιριάζει το αντικείμενο, εφαρμόζεται ο *packing rule* του *Best Fit*. Αν οι κάδοι στους οποίους ταιριάζει είναι λιγότεροι, εφαρμόζεται ο BF μονάχα σ' αυτούς. Αν δεν ταιριάζει σε κανέναν ανοικτό κάδο, τότε ανοίγει ένας καινούργιος και τοποθετείται εκεί.

Θεώρημα 7.2.2.1 $[MA98]$ Για κάθε $d \geq 2$, $R^{\infty}_{RF(d)} \leq \frac{17}{10}$.

7.2.3 ΜΕΛΕΤΗ ΜΕΣΗΣ ΕΠΙΔΟΣΗΣ

Είπαμε ότι ο *Random Fit* είναι ένας αλγόριθμος ο οποίος είναι αρκετά πρόσφατος και παρουσιάστηκε στα πλαίσια μίας συγκεκριμένης απόδειξης. Γι' αυτό ακριβώς το λόγο δεν είναι γνωστά πολλά στοιχεία για τις μέσες επιδόσεις του.

Θεώρημα 7.2.3.1 $[MA98]$ Ο RF είναι σταθερός υπό διακριτές κατανομές του τύπου $U\{k-2, k\}$, για κάθε $k \geq 3$.

---

[4] Ορίζεται για $d \geq 2$.

Το θεώρημα αυτό χρησιμοποιήθηκε για την απόδειξη της σταθερότητας του $First\ Fit$ για τις ίδιες κατανομές και η απόδειξη του ιδιαίτερα μεγάλη και πολύπλοκη.

Το ίδιο ισχύει και για τον $RF(d)$:

Θεώρημα 7.2.3.2 $[MA98]$ Ο $RF(d)$ για $d \geq 2$ είναι σταθερός υπό διακριτές κατανομές του τύπου $U\{k-2, k\}$, για κάθε $k \geq 3$.

Παρ΄ όλα αυτά, οι δύο αυτοί αλγόριθμοι δεν είναι σταθεροι κάτω από τις κατανομές $U\{k-1, k\}$, $U\{k, k\}$:

Θεώρημα 7.2.3.3 $[MA98]$ Οι RF και $RF(d)$, για $d \geq 2$, έχουν *expected waste* $o(n)$ υπό διακριτές κατανομές του τύπου $U\{k-1, k\}$ και $U\{k, k\}$, για κάθε $k \geq 3$.

Ο $Mitzenmacher$ στηρίχθηκε πάνω στα παραπάνω θεωρήματα που αφορούν τον RF και με πιθανοτικές μελέτες σχετικές με τα χρονικά βήματα εντός των οποίων μπορουμε να φτάσουμε σε συγκεκριμένες καταστάσεις $Markov$, απέδειξε ότι ο FF είναι σταθερός για κατανομές $U\{k-2, k\}$. Αναλυτικά η μελέτη του βρίσκεται στο $[MA98]$.

7.3 SUM-OF-SQUARES

Ο $Csirik$ το 1999 $[CJK99]$ και πιο ολοκληρωμένα το 2000 $[CJK00]$, παρουσίασε, μέσα από δύο εργασίες του, έναν εντελώς καινούργιο αλγόριθμο, τον *sum-of-squares*(SS). Είναι ένας *online unbounded-space* αλγόριθμος, του οποίου η συμπεριφορά μελετήθηκε κυρίως υπό διακριτές κατανομές. Παρατηρούμε ότι το ίδιο συνέβαινε και για τον *Random Fit*. Αυτό δεν είναι καθόλου τυχαίο, καθώς η μέση επίδοση υπό διακριτές κατανομές μελετάται σε ιδιαίτερα μεγάλο βαθμό τα τελευταία χρόνια. Κι αυτό συμβαίνει γιατί δεν υπήρχε μεγάλο ιστορικό στο συγκεκριμένο πεδίο, ενώ έχει πολλές εφαρμογές στη σύγχρονη πραγματικότητα.

Έχουμε πει ότι όταν αντιμετωπίζουμε διακριτές κατανομές της μορφής $U\{j, k\}$, μπορούμε, για περισσότερη ευκολία, να τις προσομιώσουμε με την περίπτωση όπου έχουμε κάδους μεγέθους k και αντικείμενα με διακριτά μεγέθη μεταξύ των τιμών $\{1, 2, \ldots, j\}$.

Η δομή του SS σχετίζεται με την προσέγγιση του προβλήματος με τις αλυσίδες $Markov$. Αν για κάποια χρονική στιγμή t,ο αριθμός των κάδων με αχρησιμοποίητο χώρο g συμβολίζεται $s_g(t)$, τότε ο SS λειτουργεί ως εξής:

SS Packing Rule : Σε μία συγκεκριμένη χρονική στιγμή του προβλήματος οριζουμε τη μεταβλητή $ss(t) = \sum_{g=1}^{k-1} s_g^2(t)$. Αυτή εκφράζει το άθροισμα των τετραγώνων των όρων μίας κατάστασης $Markov$ για τη δεδομένη χρονική στιγμή. Το τρέχων αντικείμενο θα τοποθετηθεί σ' εκείνον τον κάδο, ώστε το $ss(t+1)$, δηλαδή το άθροισμα των τετραγώνων του πλήθους των αχρησιμοποίητων χώρων για το στιγμιότυπο της επόμενης χρονικής στιγμής, να είναι το ελάχιστο δυνατό.

Κάθε φορά που φτάνει ένα αντικείμενο μεγέθους l, υπάρχουν τρεις πιθανές εκδοχές:

1. Να ανοίξει ένας καινούργιος κάδος. Εδώ η ss αυξάνει κατά $1 + 2s_{k-l}(t)$.

2. Να γεμίσει πλήρως ένας χρησιμοποιημένος κάδος. Εδώ η ss μειώνεται κατά $2s_l(t) - 1$.

3. Να τοποθετηθεί το αντικείμενο σε έναν κάδο με αχρησιμοποίητο χώρο g, όπου $g > l$. Εδώ η ss αυξάνει[5] κατά $2(s_{g-l}(t) - s_g(t) + 1)$. Σε αυτήν την περίπτωση αναζητήται το κενό εκείνο g, το οποίο μεγιστοποιεί τον όρο $s_g(t) - s_{g-l}(t)$.

Τα παραπάνω αποτελέσματα προκύπτουν με απλές πράξεις. Για παράδειγμα, έστω ότι έχουμε την δεύτερη εκδοχή. Πριν την τοποθέτηση του αντικειμένου έχουμε:

$$ss(t) = s_1^2(t) + s_2^2(t) + \ldots + s_l^2(t) + \ldots + s_{k-1}^2(t)$$

Μετά την τοποθέτηση του, έχουμε την εξής αλλαγή: $s_l(t+1) = s_l(t) - 1$. Επομένως ισχύει:

$$ss(t+1) = s_1^2(t+1) + s_2^2(t+1) + \ldots + s_l^2(t+1) + \ldots + s_{k-1}^2(t+1) = s_1^2(t) + s_2^2(t) + \ldots + (s_l(t) - 1)^2 + \ldots + s_{k-1}^2(t) = ss(t) - (2s_l(t) - 1)$$

Κάθε φορά που φτάνει ένα αντικείμενο, ο αλγόριθμος πραγματοποιεί μία εξαντλητική έρευνα για την αποτίμηση της μεταβολής του $ss(t)$ και την εύρεση

[5]Εδώ η ss αυξάνει ή μειώνεται ανάλογα με το αν ο όρος $2(s_{g-l}(t) - s_g(t) + 1)$ είναι θετικός ή αρνητικός αντίστοιχα.

του κατάλληλου κάδου. Η έρευνα αυτή γίνεται σε χρόνο $O(k)$. Επομένως ο συνολικός χρόνος για την τοποθέτηση n αντικειμένων είναι $O(nk)$.

Είναι προφανές ότι ο SS προσπαθεί να πετύχει, για κάθε αντικείμενο που φτάνει, τη μικρότερη δυνατή αύξηση της μεταβλητής $ss(t)$ ή ακόμα και τη μείωση της, εφ' όσον αυτό είναι εφικτό. Είδαμε παραπάνω ότι μείωση επιτυγχάνεται σίγουρα στην περίπτωση που το τρέχων αντικείμενο ταιριάζει απόλυτα σε κάποιον κάδο. Στην περίπτωση που δεν ταιριάζει απόλυτα κάπου, είναι αβέβαιο αν θα αυξηθεί ή θα μειωθεί η $ss(t)$. Επομένως τις περισσότερες φορές ο SS επιλέγει να τοποθετήσει ένα αντικείμενο σε έναν κάδο στον οποίο ταιριάζει απόλυτα, εφ' όσον φυσικά αυτός υπάρχει.

Ο *Csirik* $[CJK99]$ ύστερα από εκτενή πειραματική μελέτη, παρουσίασε το παρακάτω αποτέλεσμα:

Λήμμα 7.3.1 Για διακριτές κατανομές της μορφής $U\{j,k\}$ με $k = 100$ και $1 \leq j \leq 98$ ισχύει: $\overline{W}^n_{SS} = O(1)$.

Δηλαδή για τις συγκεκριμένες κατανομές ο SS είναι σταθερός, καθώς το *wasted space* είναι επίσης σχεδόν σταθερό ανεξαρτήτως της αύξησης του n. Επίσης, ο *Csirik* έδειξε ότι για $j = 99$, $\overline{W}^n_{SS} = O(\sqrt{n})$. Είναι αξιοσημείωτη η διαφορά του *wasted space* κατά τη μεταβολή του j από 98 σε 99. Πάντως για τις ίδιες κατανομές, και για $j = 97, 98, 99$, και το *wasted space* του *Best Fit* μεταβάλλεται μεταξύ των $\Theta(n)$, $O(1)$ και $\Theta(\sqrt{n})$.

Στη συνέχεια, ύστερα από θεωρητικές μελέτες, αποδείχτηκε $[CJK00]$ για όλες τις διακριτές κατανομές το εξής:

Θεώρημα 7.3.2 Για όλες τις ομοιόμορφες διακριτές κατανομές της μορφής $U\{j,k\}$ ισχύει:

$$\overline{W}^n_{SS}(U\{j,k\}) = O(1), \text{ για } j < k-1$$

Το παραπάνω θεώρημα φανερώνει τις ιδιαίτερα καλές επιδόσεις του SS υπό διακριτές κατανομές. Άλλωστε ο αλγόριθμος αυτός παρουσιάστηκε για να αντιμετωπίσει αποτελεσματικά το συγκεκριμένο είδος κατανομών.

Για την χειρότερη επίδοση του SS δεν υπήρξε ιδιαίτερα αναλυτική μελέτη. Εκείνο που απέδειξε ο *Csirik* $[CJK00]$, το οποίο όμως δεν μας λέει και πολλά πράγματα, είναι ότι για όλες τις λίστες L: $SS(L) \leq 3OPT(L)$.

Ο SS, αν και διακρίνεται για τις επιδόσεις του απέναντι σε διακριτές κατανομές, έχει το μεγάλο μειονέκτημα ότι είναι ιδιαίτερα χρονοβόρος. Να θυμίσουμε ότι η χρονική πολυπλοκότητα του είναι της τάξης του $O(nk)$. Για αυτό το λόγο ερευνήθηκαν, μέχρι πρόσφατα $[CJK02][BBJ03]$, αρκετές παραλλαγές του, οι οποίες τρέχουν σε καλύτερους χρόνους. Εκείνοι που τις παρουσίασαν υποστήριξαν ότι οι παραλλαγές αυτές έχουν ασυμπτοτικά *expected waste* της ίδιας τάξης με τον SS, με τη διαφορά ότι οι σταθερές πολλαπλασιασμού είναι αρκετά μεγαλύτερες[6]. Για παράδειγμα, μελέτησαν την κατανομή $U\{400, 1000\}$. Για $n = 100.000$ αντικείμενα, ο αριθμός των κάδων που χρησιμοποιεί μία παραλλαγή είναι 400% παραπάνω από την βέλτιστη λύση, ενώ ο *Best Fit* χρησιμοποιεί μόλις 0.3% παραπάνω και ο SS 0.25%. Η κατάσταση βελτιώνεται για την παραλλαγή του SS, όταν έχουμε πολύ μεγαλύτερες τιμές του n. Για $n = 10^7$, η παραλλαγή χρησιμοποιεί 9.8% περισσότερους κάδους, ενώ ο SS 0.0025%. Είναι φανερό επομένως ότι οι παραλλαγές αυτές είναι κυρίως θεωρητικού ενδιαφέροντος[7] και δεν μπορουν να ανταγωνιστούν τους BF και SS παρά μόνο για πολύ μεγάλες τιμές του n.

Ας δούμε όμως πιο αναλυτικά την δομή και τη συμπεριφορά τριών συγκεκριμένων παραλλαγών.

7.3.1 ΠΑΡΑΛΛΑΓΕΣ

SrS ΑΛΓΟΡΙΘΜΟΙ

Εδώ, αντί να παίρνουμε το άθροισμα των τετραγώνων των μετρητών μίας κατάστασης *Markov*, παίρνουμε το αθροισμα της r-οστής τους δύναμης, όπου $r \geq 1$. Επομένως η μεταβλητή που θέλουμε να ελαχιστοποιήσουμε εδώ είναι η:

$$ss_r(t) = s_1^r(t) + s_2^r(t) + \ldots + s_{k-1}^r(t)$$

Για όλα τα πεπερασμένα r, οι αλγόριθμοι SrS αποδίδουν σε γενικές γραμμές το ίδιο με τον SS. Για $r = 1$, ο $S1S$ είναι ένας *Any Fit* αλγόριθμος, δηλαδή δεν ανοίγει ποτέ έναν καινούργιο κάδο αν το τρέχων αντικείμενο ταιριάζει σε κάποιον από τους χρησιμοποιημένους. Αυτό αποδεικνύεται πολύ εύκολα. Στον $S1S$ επιθυμούμε σε κάθε βήμα να ελαχιστοποιήσουμε την μεταβλητή $ss_1(t)$ η

[6] Αν W_{SS}^n το *expected waste* του SS και W_p^n το αντίστοιχο της παραλλαγής του, τότε είναι και τα δύο της τάξης του $O(1)$, με τη διαφορά ότι αν $W_{SS}^n \leq c$ και $W_p^n \leq c'$, τότε $c' \gg c$.

[7] Κάτι που δήλωσαν και οι ίδιοι οι εμπνευστές τους.

οποία είναι ίση απλά με το άθροισμα των μετρητών μίας κατάστασης $Markov$. Αν το τρέχων αντικείμενο για τη χρονική στιγμή t, έχει μέγεθος l τότε, όπως έχουμε πει, υπάρχουν τρεις πιθανές εκδοχές για την τοποθέτηση του. Η πρώτη εκδοχή είναι να ταιριάξει απόλυτα σε έναν ανοικτό κάδο. Στην περίπτωση αυτή η μοναδική μεταβολή που γίνεται στην επόμενη κατάσταση $Markov$ είναι: $s_l(t+1) = s_l(t) - 1$. Επομένως η $ss_1(t)$ θα μειωθεί κατά 1. Η δεύτερη εκδοχή είναι είναι να τοποθετηθεί σε έναν κάδο στον οποίο δεν ταιριάζει απόλυτα. Δηλαδή σε κάποιον κάδο με κενό $g > l$. Στην περίπτωση αυτή συμβαίνουν τα εξής: $s_g(t+1) = s_g(t) - 1$ και $s_{g-l}(t+1) = s_{g-l}(t) + 1$. Άρα η μεταβολή του αθροίσματος των όρων της κατάστασης $Markov$ ($ss_1(t)$) είναι μηδενική. Η τρίτη εκδοχή είναι να ανοίξει ένας νέος κάδος και να τοποθετηθεί εκεί το αντικείμενο. Έδω θα έχουμε: $s_{k-l}(t+1) = s_{k-l}(t) + 1$, επομένως η $ss_1(t)$ αυξάνει κατά 1. Το συμπέρασμα είναι ότι στην περίπτωση που δεν τοποθετηθεί το τρέχων αντικείμενο σε νέο κάδο αλλά σε κάποιον ήδη ανοικτό, η $ss_1(t)$ θα μειωθεί ή θα παραμείνει σταθερή. Αντίθετα αν ανοίξει καινούργιος κάδος η $ss_1(t)$ θα αυξηθεί. Επομένως είναι αδύνατο από τον ορισμό του $S1S$ να ανοίξει νέος κάδος αν το αντικείμενο μπορεί να τοποθετηθεί σε κάποιον ήδη ανοικτό. Αρα ο $S1S$ ανήκει στην κατηγορία των $Any\ Fit$ αλγόριθμων. Να σημειώσουμε εδώ ότι ο SS δεν είναι $Any\ Fit$ αλγόριθμος.

Ο αλγόριθμος $S\infty S$ μειώνει πάντα τον υψηλότερο δείκτη της κατάστασης $Markov$. Θα δούμε γιατί συμβαίνει αυτό. Ας υποθέσουμε ότι τη χρονική στιγμή t, το κενό με τον υψηλότερο δείκτη είναι το w. Τότε η μεταβλητή $ss_\infty(t)$ έχει ως εξής:

$$ss_\infty(t) = s_1^\infty(t)+s_2^\infty(t)+\ldots+s_w^\infty(t)+\ldots+s_{k-1}^\infty(t) \Rightarrow ss_\infty(t) = s_w^\infty(t)[(\tfrac{s_1(t)}{s_w(t)})^\infty + (\tfrac{s_2(t)}{s_w(t)})^\infty + \ldots + 1 + \ldots + (\tfrac{s_{k-1}(t)}{s_w(t)})^\infty] \Rightarrow ss_\infty(t) = s_w^\infty(t)$$

Επομένως για να μειωθεί η $ss_\infty(t)$, αρκεί να μειωθεί ο υψηλότερος δείκτης της κατάστασης $Markov$.

Ο $S\infty S$ έχει το πλεονέκτημα ότι για τις διακριτές κατανομές $U\{j,k\}$, το *expected waste* του είναι το βέλτιστο. Βέβαια, για άλλου είδους κατανομές είναι γραμμικο ως προς n, οπότε δεν είναι και τόσο αποδοτικός.

Ο SS στηρίζεται στο ότι προσπαθεί να μειώσει την τιμή της μεταβλητής $ss(t)$ σε κάθε χρονική στιγμή t. Είναι έτσι ο ορισμός του που, για να πετύχει το σκοπό του, προσπαθεί προσεγγιστικά να διατηρήσει ίδιο αριθμό κάδων για κάθε κενό χώρο g, όπου $1 \leq g < k$. Όσο καλύτερα το καταφέρνει αυτό για κάποια τυχαία είσοδο, τόσο καλύτερες επιδόσεις πετυχαίνει. Όμως έχει παρατηρηθεί σε πολλές περιπτώσεις, ότι η μεταβλητή $ss(t)$ μειώνεται όταν μειώνονται οι υψηλότεροι δείκτες κάδων με αχρησιμοποίητο χώρο g. Σ' αυτή την ιδέα στηρίζονται και οι παραλλαγές που θα αναφέρουμε παρακάτω. Συγκεκριμένα, αντι να εξετάζονται όλοι οι αχρησιμοποίητοι χώροι, εξετάζονται μόνο οι m από αυτούς με τους μεγαλύτερους μετρητές.

ΠΑΡΑΜΕΤΡΙΚΟΣ SSmax ΑΛΓΟΡΙΘΜΟΣ

Ο $SSmax(m)$ υπολογίζει την τιμή της μεταβλητής $ss(t)$ μόνο για εκείνες κατηγορίες κάδων που έχουν τους m μεγαλύτερους μετρητές στην κατάσταση *Markov*. Όμως ο αλγόριθμος αυτός λαμβάνει υπόψη στον υπολογισμό του $ss(t)$ και τις περιπτώσεις όπου ανοίγει ένας καινούργιος κάδος ή το τρέχων αντικείμενο ταιριάζει απόλυτα σε έναν κάδο. Φυσικά η τοποθέτηση γίνεται σε εκείνον τον κάδο όπου ελαχιστοποιείται η τροποποιημένη μεταβλητή $ss(t)$. Είναι φανερό ότι αν $m = k$ τότε έχουμε τον γνήσιο SS αλγόριθμο. Ο χρόνος του είναι $O(mlogk)$. Για μικρές[8] τιμές του m, οι επιδόσεις του $SSmax(m)$ είναι ποιοτικά ίδιες με αυτές του $S\infty S$. Ωστόσο, όταν το k παίρνει τιμές γύρω από το $2\sqrt{k}$, παρουσιάζει εμφανείς μεταπτώσεις στην απόδοση του. Ύστερα από πολλούς πειραματισμούς, διαπιστώθηκε ότι ο $SSmax(2\sqrt{k})$ πλησιάζει τις λύσεις του SS κατά έναν παράγοντα που ποτε δεν ξεπερνάει το 1.4. Μάλιστα σε πολλές πειπτώσεις[9] χρησιμοποιεί μόλις 1.02 παραπάνω κάδους από αυτους του SS. Επομένως ο $SSmax(m)$ αγγίζει το *maximum* των δυνατοτήτων του για $m = 2\sqrt{k}$.

Segregated Sum Of Squares ΑΛΓΟΡΙΘΜΟΙ

Οι *Segregated Sum Of Squares* (SSS) αλγόριθμοι χωρίζουν το άνυσμα των καταστάσεων *Markov* σε t ίσα κατά προσέγγιση τμήματα, τα οποία αριθμούνται από 0 έως $t-1$. Η οικογένεια των αλγόριθμων αυτών απαριθμεί πολλά μέλη, τα οποία διαφοροποιούνται μεταξύ τους από τον τρόπο που γίνεται η επιλογή του t, καθώς και άλλους παράγοντες που έχουν σχέση με τη δομή τους. Ένα βοηθητικό άνυσμα $L[0 \ldots t-1]$, κρατάει τους κενούς χώρους με τους μεγαλύτερους μετρητές για κάθε τμήμα. Η τιμή του $ss(t)$ υπολογίζεται απο τις τιμές των κενών g που κρατάει το άνυσμα L, καθώς και κάποιες $O(t)$ πρόσθετες τιμές. Επομένως ο κατάλληλος κάδος κάθε φορά, αποφασίζεται σε χρόνο $\Theta(t)$. Τα πιο γνωστά μέλη της οικογένειας SSS είναι τα εξής:

[8] Για παράδειγμα όταν $m = logk$.

[9] Όταν λέμε περιπτώσεις εννοούμε διαφορετικές τιμές του n.

SSS(sqrt) Ο αριθμός των διαστημάτων για τον αλγόριθμο αυτό, είναι $t = \lceil\sqrt{k}\rceil$. Κάθε διάστημα έχει μέγεθος $\lceil\frac{k-1}{\sqrt{k}}\rceil$. Ο αλγόριθμος λειτουργεί ως εξής: Ας υποθέσουμε ότι το επόμενο αντικείμενο που πρέπει να τοποθετηθεί έχει μέγεθος s. Αρχικά εξετάζονται τα κενά που βρίσκονται στο τμήμα $m = \lceil\frac{s}{\sqrt{k}}\rceil - 1$. Το τμήμα αυτό περιέχει κενά στα οποία ταιριάζει καλύτερα το τρέχων αντικείμενο[10]. Υπολογίζουμε τις μεταβολές της μεταβλητής $ss(t)$ για όλες τις πιθανές τοποθετήσεις του αντικειμένου στους κάδους με κενά g, που ανήκουν στο τμήμα[11] m. Στη συνέχεια δοκιμάζουμε για όλα τα τμήματα $m' > m$, να τοποθετήσουμε το αντικείμενο στους κάδους με τους μεγαλύτερους μετρητές[12] για κάθε τμήμα και υπολογίζουμε τις αλλαγές της $ss(t)$. Τα τμήματα m' είναι επίσης $\Theta(\sqrt{k})$. Το αντικείμενο s τοποθετείται τελικά στον κάδο που ελαχιστοποιεί την τιμή της $ss(t)$. Η όλη διαδικασία για την εύρεση του κατάλληλου κάδου, είναι φανερό ότι απαιτεί $\Theta(\sqrt{k})$ χρόνο. Στη συνέχεια μεταβάλλουμε κατάλληλα την κατάσταση $Markov$ με τον τρόπο που αναφέραμε στο προηγούμενο κεφάλαιο. Η διαδικασία αυτή απαιτεί εξίσου $\Theta(\sqrt{k})$ χρόνο. Επομένως ο συνολικός χρόνος τοποθέτησης ενός αντικειμένου είναι $\Theta(\sqrt{k})$.

SSS(log) Ο αλγόριθμος αυτός διαχωρίζει την κατάσταση $Markov$ σε $t = \lceil logk\rceil$ τμήματα. Ένα χαρακτηριστικό του $SSS(log)$ είναι ότι για κάθε τμήμα, διατηρεί ένα σωρό[13] στον οποίο τοποθετούνται οι μετρητές του τμήματος. Αν $m = \lceil\frac{slogk}{k}\rceil - 1$, υπολογίζουμε τις μεταβολές της μεταβλητής $ss(t)$ για τα παρακάτω κενά:

1. Το κενό με το μεγαλύτερο μετρητή του τμήματος m, δηλαδή το κενό $L(m)$, και τα $logk$ κενά που ακολουθουν το κενό $L(m)$ στην κατάσταση $Markov$. Δηλαδή τα κενά $L(m)$, $L(m) + 1$, …, $L(m) + logk$

2. Για κάθε τμήμα $m' > m$, τα κενά με το μεγαλύτερο μετρητή, $L(m')$.

Επομένως υπολογίζουμε τις μεταβολές της $ss(t)$ μόνο για $\Theta(logk)$ κενά και τοποθετούμε φυσικά το αντικείμενο μας στον κάδο για τον οποίο ελαχιστοποιήται η $ss(t)$. Εφ΄ όσον χρησιμοποιήται ένας σωρός για κάθε τμήμα, ο μεγαλύτερος μετρητής ενός τμήματος μπορεί να βρεθεί απλά εξετάζοντας τη ρίζα του αντίστοιχου σωρού. Όταν τοποθετηθεί το τρέχων αντικείμενο σε κάποιον κάδο με κενό έστω g, τότε ενημερώνουμε τους δύο σωρούς που περιέχουν τους μετρητές

[10] Δηλαδή τα μικρότερα δυνατά κενά που είναι όμως μεγαλύτερα του μεγέθους s.

[11] Υπάρχουν $\Theta(\sqrt{k})$ στοιχεία σε κάθε τμήμα.

[12] Να θυμίσουμε ότι μετρητές ονομάζουμε τα $s_i(t)$ της κατάστασης $Markov$.

[13] Να θυμίσουμε ότι σωρός είναι ένα πλήρες δυαδικό δέντρο στο οποίο η τιμή της ρίζας είναι μεγαλύτερη ή ίση από τις τιμές των υπόλοιπων κόμβων και αυτό ισχύει αναδρομικά για την ρίζα κάθε υποδέντρου.

για τα κενά g και $g-s$. Ο μετρητής g μειώνεται κατά 1, ενώ ο αντίστοιχος $g-s$ αυξάνεται κατά 1. Αν κατά τη διαδικασία αυτή πάρει τα πρωτία κάποιος άλλος μετρητής για κάποιο τμήμα, γίνεται μεταταβάλεται η μορφή του σωρού με τη μέθοδο *Heapsort*, ώστε ο νέος μετρητής να βρεθεί στη ρίζα του. Η όλη διαδικασία απαιτεί $\Theta(logk)$ χρόνο.

SSS(loglog) Τα τμήματα εδώ είναι $t = \lceil loglogk \rceil$. Ο $SSS(loglog)$ διατηρεί ένα σωρό για κάθε τμήμα ο οποίος όμως εδώ περιέχει το πολύ $logk$ μη μηδενικούς μετρητές του τμήματος. Εφ' όσον δεν περιέχονται όλοι οι μετρητές στο σωρό, είναι πιθανό ο υψηλότερος μετρητής να μην βρίσκεται στη ρίζα του σωρού. Όταν ένας μετρητής μεταβάλλεται, εισάγεται στο σωρό[14] και αν ο αριθμός των μετρητών ξεπεράσει τους $logk$, εξάγεται ο τελευταίος από αυτούς. Ας υποθέσουμε ότι το αντικείμενο που πρέπει να τοποθετηθεί έχει μέγεθος s και ότι $m = \lceil \frac{sloglogk}{k} \rceil - 1$. Θα συμβολίσουμε με $H(m)$ την μεγαλύτερη τιμή[15] του σωρού του τμήματος m, η οποία προφανώς βρίσκεται στη ρίζα του. Ο αλγόριθμος υπολογίζει τις μεταβολές της $ss(t)$ κατά την πιθανή τοποθέτηση του τρέχοντος αντικειμένου στα παρακάτω κενά:

1. Το κενό με τον μεγαλύτερο μετρητή του σωρού του τμήματος m, και τα $loglogk$ κενά που ακολουθούν στην κατάσταση *Markov*. Αυτά είναι τα κενά $H(m)$, $H(m)+1$, ..., $H(m) + loglogk$.

2. Τα κενά $H(m')$, για κάθε τμήμα $m' > m$.

Ύστερα από εκτενή πειραματική μελέτη της απόδοσης των SSS αλγορίθμων για διακριτές $U\{j, k\}$ κατανομές με $k = 100$ και j από 1 έως 99, οι επινοητές τους κατέληξαν σε κάποια ενδιαφέροντα αποτελέσματα. Όλοι οι SSS αλγόριθμοι έχουν σταθερό *wasted space* για $j \leq 98$ και για n μεταξύ των τιμών $\{10^5, 10^6, 10^7, 10^8, 10^9\}$. Για $j = 99$ και για τις ίδιες τιμές του n, το *wasted space* τους έχει ακριβώς την ίδια συμπεριφορά με αυτήν του SS. Γενικά όλοι οι αλγόριθμοι της οικογένειας των SSS εκτός του $SSS(loglog)$ υπολείπονται του *wasted space* του SS κατά ποσοστό που δεν ξεπερνά το 0.80%. Αυτό δείχνει ότι οι SSS, αν και τρέχουν σε χαμηλότερους χρόνους, έχουν αποδόσεις αρκετά κοντά σ' αυτήν του SS, για διακριτές κατανομές. Όμως η συμπεριφορά τους δεν είναι εξίσου καλή για συνεχείς κατανομές. Για αρκετές από τις κατανομές αυτού του είδους για τις οποίες ο SS παρουσιάζει *sublinear waste*, οι SSS παρουσιάζουν γραμμικό *waste*.

[14] Εφ' όσον δεν υπήρχε ήδη σ' αυτόν.

[15] Όπως είπαμε, αυτό δεν σημαίνει ότι είναι και η μεγαλύτερη τιμή του τμήματος.

Γενικά, το συμπέρασμα που βγαίνει είναι ότι *sum of squares* αλγόριθμοι παρουσιάζουν ικανοποιήτικες αποδόσεις κυρίως για διακριτές κατανομές. Μάλιστα είναι πολύ σημαντικό ότι υπάρχουν αρκετές παραλλαγές τους με εξίσου καλές αποδόσεις, για τις οποίες δεν χρειάζεται να γίνεται υπολογισμός του συνόλου των μεταβολών της $ss(t)$ και γι' αυτό το λόγο τρέχουν σε αρκετά ικανοποιητικούς χρόνους.

Κεφάλαιο 8

ΣΥΜΠΕΡΑΣΜΑΤΑ

Το *Bin Packing* είναι ένα από τα δυσκολότερα πρόβληματα της Επιστήμης της Πληροφορικής με πολλές εφαρμογές στη συγχρονη πραγματικότητα. Χαρακτηριστικές εφαρμογές του είναι η κατανομή μνήμης σε υπολογιστικά συστήματα και η βιομηχανική κατανομή χώρου. Η μελέτη του μπορεί να προσφέρει της υπηρεσίες της σε πολλές άλλες σημερινές εφαρμογές. Για παράδειγμα, στην βέλτιστη κατανομή του τηλεοπτικού χρόνου για την προβολή διαφημιστικών μυνημάτων. Εδώ αντικείμενα είναι τα διαφημιστικά μυνήματα και κάδοι τα τηλεοπτικά διαλείμματα για την προβολή τους. Χρήση του γίνεται και στις τηλεπικοινωνίες για την κατανομή διάφορων εύρων συχνοτήτων (κάδοι) στις συχνότητες πιθανών χρηστών (αντικείμενα).

Γιάυτόν ακριβώς τον λόγο έχει γίνει σημαντική έρευνα πάνω στο αντικείμενό του για την εύρεση αποδοτικών λύσεων. Παρ΄ όλα αυτά, αν και έχουν αποδειχτεί θεωρήματα τα οποία δηλώνουν τα χαμηλότερα δυνατά όρια για αρκετές κατηγορίες αλγορίθμων, σε πολλές από τις κατηγορίες αυτές δεν έχουν βρεθεί συγκεκριμένοι αλγόριθμοι που να πιάνουν τις βέλτιστες αποδόσεις της κατηγορίας τους. Συνήθως παρουσιάζονται αλγόριθμοι οι οποίοι ασυμπτοτικά πιάνουν τα όρια αυτά. Αυτό δείχνει ότι υπάρχει ελεύθερο πεδίο για περαιτέρω έρευνα πάνω στο πρόβλημα αυτό. Ανά τακτά χρονικά διαστήματα, παρουσιάζονται νέοι αλγόριθμοι με αρκετά πολύπλοκους ορισμούς και δημοσιεύονται ποικίλα πολυσέλιδα κείμενα που ασχολούνται με τη μελέτη της συμπεριφοράς τους.

Στο σημείο αυτό θα παραθέσουμε έναν πίνακα με τα κυριότερα χαρακτηριστικά των σημαντικότερων αλγόριθμων που έχουμε αναφέρει:

όνομα	κατηγορία	$w.$-$c.$	Μέση απόδοση	Χρόνος
$NextFit$	*online* 1-*b.s.*	2	$ER^{\infty}(U[0,1]) = \frac{4}{3}$	$O(n)$
First Fit	*online*	1.7	$ER^{\infty}(U[0,1]) = 1$	$O(nlogn)$
Best Fit	*online*	1.7	$EW(U[0,1]) = \Theta(n^{\frac{1}{2}} log^{\frac{3}{4}} n)$	$O(nlogn)$
Next-K Fit	*online K-b.s.*	$1.7 + \frac{3}{10(K-1)}$	$NF_2 : ER^{\infty}(U[0,1]) \approx 1.24$	$O(nlogK)$
K-Bounded Best Fit	*online K-b.s.*	1.7	$BBF_2 : ER^{\infty}(U[0,1]) \approx 1.18$	$O(nlogK)$
$Harmonic_K$	*online K-b.s.*	$1.69103,\ K \to \infty$	$ER^{\infty}(U[0,1]) = 1.2899$	$O(nlogK)$
First Fit Decreasing	$offline$	$1.222\ldots$	$EW(U[0,1]) = \Theta(n^{\frac{1}{2}})$	$O(n^2 logn)$
Best Fit Decreasing	$offline$	$1.222\ldots$	$EW(U[0,1]) = \Theta(n^{\frac{1}{2}})$	$O(n^2 logn)$
Smart Next Fit	*online* 1-*b.s.*	2	$ER^{\infty}(U[0,1]) \approx 1.24$	$O(nlogK)$
Online $Match_K$	*online K-b.s.*	2	$ER^{\infty}(U[0,1]) = 1,\ K \to \infty$	$\Theta(nlogn)$
Random Fit	*online*	2	$EW(U\{k-2,k\}) = O(1)$	$O(nlogn)$
Random Fit(d)	*online d-b.s.*	1.7	$EW(U\{k-2,k\}) = O(1)$	$O(nlogd)$
Sum of Squares	*online*	≤ 3	$EW(U\{j,k\}) = O(1),\ j < k-1$	$O(nk)$

Βέβαια στη μελέτη μας αναφέραμε αρκετούς αλγόριθμους που δεν βρίσκονται στον παραπάνω πίνακα. Για κάποιους απο αυτούς αποδείχτηκε ότι δεν είναι αποδοτικοί και δεν έγιναν ιδιαίτερες μελέτες όσον αφορά τη συμπεριφορά τους. Για άλλους δεν έχουν γίνει πρόσφατες μελέτες πάνω στη μέση απόδοση τους και ότι στοιχεία υπάρχουν είναι καθαρά πειραματικά. Στην τελευταία σελίδα υπάρχει ένας πίνακας με όλους τους αλγόριθμους που έχουμε αναφέρει, το χρόνο που παρουσιάστηκαν, το όνομα εκείνου που τους επινόησε και το αντίστοιχο *paper*. Να πούμε ότι οι *Next Fit* και *Best Fit* χρονολογούνται πριν το 1973, αλλά τότε έγινε η πρώτη ουσιαστική μελέτη των αποδόσεων τους.

Τα τελευταία χρόνια έχει δοθεί ιδιαίτερη βάση στην ανάλυση μέσων επιδόσεων, η οποία άλλωστε είναι αρκετά πιο πολύπλοκη από την αντίστοιχη των χειρότερων επιδόσεων. Και είναι πιο πολύπλοκη επειδή το αντικείμενο της σχετίζεται με πιθανοτικά μεγέθη, των οποίων η μελέτη χαρακτηρίζεται από υψηλό βαθμό δυσκολίας. Γι' αυτό το λόγο άλλωστε, δεν έχει γίνει ακόμα η ανάλογη έρευνα πάνω στις μέσες επιδόσεις αλγορίθμων που χρονολογούνται αρκετές δεκαετίες πριν. Βλέπουμε τα τελευταία χρόνια να αποδεικνύονται αρκετά θεωρήματα για κλασσικούς αλγόριθμους όπως ο *First Fit* και ο *Best Fit*. Βέβαια κάτι τέτοιο είναι λογικό, καθώς όλα τα θεωρήματα μέσων επιδόσεων περιλαμβάνουν πολυσέλιδες και ιδιαίτερης πολυπλοκότητας αποδείξεις. Κι αν συνυπολογίσει κανείς το πλήθος των κατανομών που πρέπει να ερευνηθούν, γίνεται κατανοητό πόσο μεγάλο είναι το εύρος της μελέτης αυτής.

Την τελευταία δεκαετία γίνεται προσέγγιση με καινούργιες μεθόδους όπως οι αλυσίδες *Markov* και μάλιστα με ιδιαίτερη επιτυχία. Μέσω αυτών των μεθόδων, αποδεικνύονται συνεχώς νέα θεωρήματα τα οποία βάζουν τις βάσεις για την μετέπειτα έρευνα του προβλήματος.

Το *Bin Packing* είναι ένα εξαιρετικά ενδιαφέρον πρόβλημα που αξίζει κανείς να το μελετήσει, καθώς θα τον βοηθήσει να εμπλουτίσει τις γνώσεις του πάνω στη μελέτη προσεγγιστικών αλγορίθμων και τη θεωρία της πολυπλοκότητας γενικότερα.

ABF_K	$Csirik$	1991	$[CJ91]$
AFB_K	$Csirik$	1991	$[CJ91]$
$Almost\ Worst\ Fit$	$Johnson$	1973	$[Joh73]$
$Best\ Fit$	$Johnson$	1973	$[Joh73]$
$Best\ Fit\ Decreasing$	$Johnson$	1973	$[Joh73]$
$First\ Fit$	$Johnson$	1973	$[Joh73]$
$First\ Fit\ Decreasing$	$Johnson$	1973	$[Joh73]$
$Group\text{-}X\ Fit$	$Johnson$	1973	$[Joh73]$
$Harmonic + 1$	$Richey$	1991	$[Ric91]$
$Harmonic_K$	Lee	1985	$[LL85]$
$K\text{-}Bounded\ Best\ Fit$	$Csirik$	1991	$[CJ91]$
$Modified\ Harmonic_K$	$Ramanan$	1989	$[RBL89]$
$Next\ Fit$	$Johnson$	1973	$[Joh73]$
$Next\text{-}K\ Fit$	$Johnson$	1973	$[Joh73]$
$Online\ Match_K$	$Hoffman$	1982	$[Hof82]$
$Random\ Fit$	$Mitzenmacher$	1998	$[MA98]$
$Random\ Fit(d)$	$Mitzenmacher$	1998	$[MA98]$
REP_3	$Galambos$	1993	$[GW93b]$
$Refined\ First\ Fit$	Yao	1980	$[Yao80]$
$Refined\ Harmonic_K$	Lee	1985	$[LL85]$
$Simplified\ Harmonic_K$	$Woegigner$	1993	$[Woe93]$
$Smart\ Next\ Fit$	$Ramanan$	1989	$[Ram89]$
$Sum\ Of\ Squares$	$Csirik$	1999	$[CJK99]$
$Worst\ Fit$	$Johnson$	1973	$[Joh73]$

Βιβλιογραφία

[AKT84] M. Ajtai, J. Komlos, G. Tusnady. On optimal matchings. Combinatorica, 4: 259-264, 1984.

[Bak83] B. S. Baker. 'A new proof for the first fit decreasing algorithm', Technical Report, Bell Laboratories, Murray Hill, 1983.

[BBJ03] M. A. Bender, B. Bradley, G. Jagannathan. The Robustness of the Sum-of-Squares Algorithm for Bin Packing, 2003

[BJL84] J. L. Bentley, D. S. Johnson, F. T. Leighton, C. C. McGeoch, L. A. McGeoch. Some unexpected expected behavior results for bin packing. In Proceedings of the 16th Annual ACM Symposium on Theory of Computing, p. 279-288, 1984.

[CCG91] E. G. Coffman, C. Courcoubetis, M. R. Garey, D. S. Johnson, L. A. McGeorg, P. W. Shor, R. R. Weber, and M. Yannakakis. Fundamental discrepancies between average-case analyses under discrete and continuous distributions: A bin packing case study. In Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, p. 230-240. ACM Press, 1991.

[CCG00] E. G. Coffman, C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, and M. Yannakakis. Bin packing with discrete item sizes, part I: Perfect packing theorems and average case behavior of optimal packings. SIAM Journal on Discrete Math., 13:384-402, 2000.

[CGJ96] E.G. Coffman Jr., M.R. Garey and D.S. Johnson. Approximation Algorithms For Bin Packing: A Survey. In Approximation Algorithms For NP – Hard Problems, D.S. Hochbaum, ed., PWS Publishing Company, Boston, 1996.

[CI89] J. Csirik, B. Imreh. On the worst-case performance of the NkF bin-packing heuristic. Acta Cybernetica, 9: 89-105, 1989.

[$CJ91$] *J. Csirik, D. S. Johnson. Bounded Space On-Line Bin Packing: Best is Better than First. In Proceedings, Second Annual ACM-SIAM Symposium on Discrete Algorithms, p.* 309-319, *Philadelfia*, 1991. *Society for Industrial and Applied Mathematics.*

[$CJ01$] *J. Csirik, D. S. Johnson. Bounded Space On-Line Bin Packing: Best is Better than First, final draft of paper that appeared in ALGORITHMICA*, 31-2-2001.

[$CJK99$] *J. Csirik, D. S. Johnson, C. Kenyon, P. W. Shor, R. R. Weber. A self organizing bin packing heuristic. In Proceedings of ALENEX workshop, p.* 246-265, 1999.

[$CJK00$] *J. Csirik, D. S. Johnson, C. Kenyon, J. Orlin, P. W. Shor, R. R. Weber. On the sum-of-squares algorithm for bin packing. In Proceedings of the* 32*nd Annual ACM. Symp. on the theory of Computing, p.* 208-217, 2000.

[$CJK02$] *J. Csirik, D. S. Johnson, C. Kenyon, J. Orlin, P. W. Shor, R. R. Weber. On the sum-of-squares algorithm for bin packing. Unpublished Manuscript*, 2002.

[$CJS93$] *E. G. Coffman, D. S. Johnson, P. W. Shor, R. R. Weber. Markov chains, computer proofs and best fit bin packing. In Proceedings of the* 25*th ACM Symposium on the Theory of Computing, p.* 412-421, *New York*, 1993. *ACM Press.*

[$CJS95$] *E. G. Coffman, D. S. Johnson, P. W. Shor, R. R. Weber. Bin packing with discrete item sizes, part II: Average-case behavior of first fit*, 1995. *manuscript.*

[$CL91$] *E. G. Coffman, Jr. and G. S. Lueker. Probabilistic Analysis of Packing and Partitioning Algorithms. Wiley, New York*, 1991.

[$CS93$] *E. G. Coffman, P. W. Shor. Packing in two dimensions: Asymptotic average-case analysis of algorithms. Algorithmica*, 9: 253-277, 1993.

[$CSH80$] *E. G. Coffman, Jr. K. So, M. Hofri, A. C. Yao. A stochastic model of bin-packing. Inf. and Cont.*, 44: 105-115, 1980.

[$FK91$] *S. Floyd, R. M. Karp. FFD bin packing for item sizes with distributions on* $[0, 1/2]$. *Algorithmica*, 6: 222-240, 1991.

[GGJ76] M. R. Garey, R. L. Graham, D. S. Johnson, A. C. Yao. Resource constrained scheduling as generalized bin packing. J. Comb. Th. Ser. A, 21: 257-298, 1976.

[GPT90] G. Gambosi, A. Postiglione, M. Talamo. New algorithms for on-line bin packing. In R. Petreschi, G. Ausiello, D. P. Bovet, editor, Algorithms and Complexity, Proceedings of the First Italian Conference, p. 44-59, Singapore, 1990. World Scientific.

[Gro94] E. F. Grove. Online bin packing with lookahead. Unpublished manuscript, July 1994.

[GW93b] G. Galambos, G. J. Woegigner. Repacking helps in bounded space on-line bin-packing. Computing, 49: 329-338, 1993.

[Hof82] U. Hoffman. A class of simple stochastic online bin packing algorithms. Computing, 29: 227-239, 1982.

[IL93] Z. Ivkovic, E. Lloyd. Fully dynamic algorithms for bin packing: Being myopic helps. In Proceedings of the First European Symposium on Algorithms, No 726 in Lecture notes in Computer Science, p. 224-235, New York, 1993. Springer-Verlag. Journal version to appear in SIAM J. Comput.

[JDU74] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J. Comput., 3: 299-325, 1974.

[Joh72] D. S. Johnson. 'Near-optimal bin packing algorithms', Technical Report MAC TR – 109, Project MAC, MIT, Cambridge, 1972.

[Joh73] D. S. Johnson. Near-Optimal Bin Packing Algorithms. PhD thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge 1973.

[Joh74] D. S. Johnson. Fast algorithms for bin packing. Journal of Computer and System Sciences, 8: 272-314, 1974.

[Joh90] D. S. Johnson. Catalog Of Complexity Classes. In Algorithms and Complexity, Volume A, J. V. Leeuwen, Utrecht University, The Netherlands, 1990.

[$KL00$] $Prof.$ $Dr.$ $R.$ $Klein$, $Dr.$ $E.$ $Langetepe$, $Dipl.$ $Inform.$ $T.$ $Kamphans.$ $Online$-$Algorithmen.$ $Proseminar$ im $Wintersemester$ 00/01, $Vortrag$ Bin $Packing$ von $Thilo$ $Geertzen$ 25. $Oktober$ 2000.

[$Kno81$] $W.$ $Knodel.$ A bin $packing$ $algorithm$ $with$ $complexity$ $O(nlogn)$ and $performance$ 1 in the $stochastic$ $limit.$ In $J.$ $Gruska$ and $M.$ $Chytil$, $editors$, $Proceedings$ 10th $Symposium$ on $Mathematical$ $Foundations$ of $Computer$ $Science$, $Lecture$ $Notes$ in $Computer$ $Science$, 118, $p.$ 369-378, $Berlin$, 1981. $Springer$-$Verlag.$

[$KRS96$] $C.$ $Kenyon$, $Y.$ $Rabani$, $A.$ $Sinclair.$ $Biased$ $random$ $walks$, $Lyapunov$ $functions$ and $stochastic$ $analysis$ of $best$ fit bin $packing.$ In $Proceedings$ of the 7th $Annual$ ACM-$SIAM$ $Symposium$ on $Discrete$ $Algorithms$, $p.$ 351-358, $Philadelfia$, 1996. $Society$ for $Industrial$ and $Applied$ $Mathematics.$

[$LL85$] $C.$ $C.$ Lee, $D.$ $T.$ $Lee.$ A $simple$ on-$line$ $packing$ $algorithm.$ $J.$ ACM, 32: 562-572, 1985.

[$LL87$] $C.$ $C.$ Lee, $D.$ $T.$ $Lee.$ $Robust$ on-$line$ bin $packing$ $algorithms.$ $Technical$ $report$, $Department$ of $Electrical$ $Engineering$ and $Computer$ $Science$, $Northwestern$ $University$, $Evaston$, IL, 1987.

[$LS89$] $F.$ $T.$ $Leighton$, $P.$ $Shor.$ $Tight$ $bounds$ for $minimax$ $grid$ $matching$ $with$ $applications$ to the $average$ $case$ $analysis$ of $algorithms.$ $Combinatorica$, 9: 161-187, 1989.

[$Lue82$] $G.$ $S.$ $Lueker.$ An $average$-$case$ $analysis$ of bin $packing$ $with$ $uniformly$ $distributed$ $item$ $sizes.$ $Technical$ $Report$ 181, $University$ of $California$ at $Irvine$, $Department$ of $Information$ and $Computer$ $Science$, 1982.

[$LZ02$] $J.$ $V.$ $Leeuwen$, $C.$ $Zonnenberg$, $Lecture$ of $Algorithmic$ $Modeling$ and $Complexity$, 2002.

[$MA98$] $M.$ $Mitzenamacher$, $S.$ $Albers.$ $Average$ $case$ $Analyses$ of $First$ Fit and $Random$ Fit bin $packing.$ In $Proceedings$ of the 9th $Annual$ ACM $Symposium$ on $Discrete$ $Algorithms$, $p.$ $290 - 299$, 1998.

[$Mao93a$] $W.$ $Mao.$ $Best$-k-fit bin $packing.$ $Computing$, 50: 265-270, 1993.

[$Mao93b$] $W.$ $Mao.$ $Tight$ $worst$-$case$ $performance$ $bounds$ for $next$-k-fit bin $packing.$ $SIAM$ $J.$ $Comput.$, 22: 46-56, 1993.

[May01] P. Maymounkov. Divergence-proving Techniques for Best Fit Bin Packing and Random Fit. Senior Thesis to Mathematics in partial fulfillment of the honors requirements for the degree of Bachelor of Arts, Harvard College Cambridge, Massachusetts. April 2, 2001.

[Pap] C.H. Papadimitriou, University of California- San Diego. Computational Complexity. Addison-Wesley Publishing Company.

[Ram89] P. Ramanan. Average-case analysis of the smart next fit algorithm. Inf. Proc. Lett., 31: 221-225, June 12 1989.

[RBL89] P. Ramanan, D. J. Brown, C. C. Lee, D. T. Lee. On-line bin packing in linear time. J. Algorithms, 10: 305-326, 1989.

[RT89] P. Ramanan, K. Tsuga. Average-case analysis of the modified harmonic algorithm. Algorithmica, 4: 519-533, 1989.

[Sho86] P. W. Shor. The average-case analysis of some on-line algorithms for bin packing. Combinatorica, 6(2): 179-200, 1986.

[Sho91] P. W. Shor. How to pack better than best-fit: Tight bounds for average-case on-line bin packing. In Proceedings, 32nd Annual Symposium on Foundations of Computer Science, p. 752-759, New York, 1991. IEEE Computer Society Press.

[Ull71] J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, October 1971.

[Vaz01] V. V. Vazirani. Approximation Algorithms, Springer-Verlag, Berlin, 2001 p. $74 - 78$ and $343 - 348$.

[Vli95] A. van Vliet. Lower and Upper Bounds for On-Line Bin Packing and Scheduling Heuristic. PhD thesis, Erasmus University, Rotterdam, Netherlands, 1995.

[Vli96] A. van Vliet. On the asymptotic worst case behavior of harmonic fit. J. Algorithms, 20: 113-136, 1996.

[Wil99] D. P. Williamson. A PTAS for Bin Packing. In Approximation Algorithms, IBM Research Report, RC 21409, 17-2-99.

[Woe93] G. Woegigner. Improved space for bounded-space, on-line bin-packing. SIAM J. Disc. Math., 6: 575-581, 1993.

[$Xu93$] $K.\ Xu.\ A\ Bin$-$Packing\ Problem\ with\ Item\ Sizes\ in\ the\ Interval$ $(0,\ a]\ for\ a \leq \frac{1}{2}$. $PhD\ thesis,\ Chinese\ Academy\ of\ Sciences,$ $Institute\ of\ Applied\ Mathematics,\ Beijing,\ China$, 1993.

[$Yao80$] $A.\ C.\ Yao.\ New\ algorithms\ for\ bin\ packing.\ J.\ Assoc.\ Comput.$ $Mach.$, 27: 207-227, 1980.

[$Yue91$] $M.\ Yue.\ A\ simple\ proof\ of\ the\ inequality\ FFD \leq \frac{11}{9}OPT(L) + 1 \forall L,\ for\ the\ FFD\ bin$-$packing\ algorithm.\ Acta\ Math.\ App.\ Sinica$, 7: 321-331, 1991.

[$Zha94$] $G.\ Zhang.\ Tight\ worst$-$case\ performance\ bound\ for\ AFB_k.$ $Technical\ Report$ 015, $Institute\ of\ Applied\ Mathematics,\ Academia$ $Sinica,\ Beijing,\ China,\ May$ 1994.