ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική εργασία

ΑΛΓΟΡΙΘΜΟΙ ΧΡΟΝΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΤΥΧΑΙΩΝ ΕΡΓΑΣΙΩΝ ΣΕ ΔΥΟ ΠΑΡΑΛΛΗΛΕΣ ΜΗΧΑΝΕΣ ΜΕ ΑΠΕΡΙΟΡΙΣΤΗ ΧΩΡΗΤΙΚΟΤΗΤΑ

υπό

ΒΑΛΑΤΣΟΥ ΚΩΝ/ΝΟΥ

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Διπλώματος Μηχανολόγου Μηχανικού

2011

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής :

Πρώτος Εξεταστής (Επιβλέπων)	Δρ Γεώργιος Κοζανίδης Επίκουρος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Δευτερος Εξεταστής	Δρ Γεώργιος Λυμπερόπουλος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Τρίτος Εξεταστής	Δρ Δημήτριος Παντελής Επίκουρος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Πρώτα απ' όλα, θέλω να ευχαριστήσω τον επιβλέποντα της διπλωματικής εργασίας μου, Επίκουρο Καθηγητή κ. Γεώργιο Κοζανίδη, για την πολύτιμη βοήθεια και καθοδήγησή του κατά τη διάρκεια της δουλειάς μου. Η συμβολή του υπήρξε καθοριστική στην ολοκλήρωση της εργασίας αυτής. Επίσης, είμαι ευγνώμων στα υπόλοιπα μέλη της εξεταστικής επιτροπής της διπλωματικής εργασίας μου, Καθηγητή κ. Γεώργιο Λυμπερόπουλο και Επίκουρο Καθηγητή κ. Δημήτριο Παντελή για την προσεκτική ανάγνωση της εργασίας μου και για τις πολύτιμες υποδείξεις τους. Πάνω απ' όλα, είμαι ευγνώμων στους γονείς μου, Νικόλαο Βαλατσό και Μαγδαληνή Σανίδα για την ολόψυχη αγάπη και υποστήριξή τους όλα αυτά τα χρόνια. Αφιερώνω αυτήν την εργασία στην μητέρα και στον πατέρα μου.

Κώστας Βαλατσός

ΑΛΓΟΡΙΘΜΟΙ ΧΡΟΝΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΤΥΧΑΙΩΝ ΕΡΓΑΣΙΩΝ ΣΕ ΔΥΟ ΠΑΡΑΛΛΗΛΕΣ ΜΗΧΑΝΕΣ ΜΕ ΑΠΕΡΙΟΡΙΣΤΗ ΧΩΡΗΤΙΚΟΤΗΤΑ

ΚΩΝ/ΝΟΣ ΒΑΛΑΤΣΟΣ

Πανεπιστήμιο Θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών, 2011

Επιβλέπων Καθηγητής : Δρ. Γεώργιος Κοζανίδης, Επίκουρος Καθηγητής
Μεθόδων Βελτιστοποίησης Συστημάτων
Παραγωγής/Υπηρεσιών

Περίληψη

Ο προγραμματισμός εργασιών κατά παρτίδες έχει απασχολήσει την επιστημονική κοινότητα ιδιαίτερα τις τελευταίες δεκαετίες. Οι εφαρμογές του αφορούν πολλους τομείς : μηχανολογικές κατεργασίες, εξυπηρέτηση πελατών σε συστήματα αναμονής, τεχνολογία υπολογιστών, κ.α. Η περίπτωση που εξετάζουμε αφορά m μηχανές που επεξεργάζονται n εργασίες υπό μορφή παρτίδων. Στην παρούσα εργασία, αρχικά, κάνουμε μια περιγραφή του προβλήματος και των παραμέτρων του. Έπειτα, παρουσιάζουμε τους αλγορίθμους επίλυσης που χρησιμοποιήθηκαν. Στη συνέχεια, παρουσιάζονται τα πειράματα που διεξήχθησαν, προκειμένου να πάρουμε αποτελέσματα, από τα οποία προκύπτουν συμπεράσματα για το παρόν πρόβλημα. Επίσης, δίνονται προτάσεις για μελλοντική έρευνα.

Πίνακας Περιεχομένων

ΚΕΦΑΛΑΙΟ 1 : ΕΙΣΑΓΩΓΗ

Στην Επιστήμη των Υπολογιστών, λέμε οτι ένας αλγόριθμος είναι online όταν επεξεργάζεται τα δεδομένα εισόδου με την σειρά με την οποία αυτά γίνονται γνωστά, χωρις να τα έχει όλα διαθέσιμα εξ' αρχής. Αντίθετα, ένας offline αλγόριθμος γνωρίζει όλα τα δεδομένα εισόδου εξ' αρχής.
Επειδή ενας online αλγόριθμος δεν έχει πλήρη γνώση των δεδομένων εκ των προτέρων, μπορεί να πάρει αποφάσεις οι οποίες πιθανόν να μην είναι οι βέλτιστες, όπως θα αποδειχθεί τελικά.
Για αυτόν το λόγο, η μελέτη των online αλγορίθμων έχει εστιάσει στην αξιολόγηση της λήψης αποφάσεων που είναι δυνατόν να ληφθούν για το εκάστοτε πρόβλημα.
Η ανάλυση ανταγωνιστικότητας (competitive analysis) υλοποιεί αυτή την ιδέα, συγκρίνοντας την απόδοση του online και offline αλγορίθμου για το ίδιο πρόβλημα.

Στο πρόβλημα που εξετάζουμε στην παρούσα εργασία, ασχολούμαστε με τον χρονικό προγραμματισμό εργασιών (on- line scheduling) σε 2 παράλληλες μηχανές (m =2), με απεριόστη χωρητικότητα η καθεμία ($b = \infty$), όπου b είναι η χωρητικότητα της κάθε μηχανής και οι εργασίες καταφθάνουν με τρόπο τυχαίο. Δηλαδή, τα δεδομένα για κάθε εργασία δεν είναι διαθέσιμα μέχρις ότου απελευθερωθεί η εκάστοτε εργασία. Μας δίνονται n ανεξάρτητες εργασίες. Στόχος μας είναι να σχηματίσουμε παρτίδες εργασιών, τις οποίες θα τροφοδοτήσουμε κατάλληλα στις 2 παράλληλες μηχανές, με απώτερο σκοπό να ελαχιστοποιήσουμε τον χρόνο ολοκλήρωσης της επεξεργασίας όλων των εργασιών (makespan). Ο χρόνος επεξέργασίας της κάθε παρτίδας ορίζεται ως ο χρόνος επεξεργασίας της μεγαλύτερης σε διάρκεια εργασίας στην παρτίδα αυτή και όλες οι εργασίες στην παρτίδα έχουν τον ίδιο χρόνο έναρξης και τον ίδιο χρόνο ολοκλήρωσης. Κάθε εργασία J_j (1<j<n) έχει έναν χρόνο άφιξης r_j, ο οποίος δεν είναι γνωστός εκ των προτέρων και έναν χρόνο επεξέργασίας p_j, ο οποίος γνωστοποιείται κατά την στιγμή άφιξης της εργασίας, την χρονική στιγμή t. Δηλαδή, το πρόβλημα αφορά την ανάθεση όλων των εργασιών σε παρτίδες στις m μηχανές και τον προσδιορισμό των χρόνων έναρξης των παραγόμενων παρτίδων, κατά τέτοιο τρόπο ώστε ο χρόνος ολοκλήρωσης (makespan) $\max_{1<j<n} C_j$ να ελαχιστοποιείται, όπου C_j ο χρόνος ολοκλήρωσης της εργασίας J_j. Συμφωνα με τον τρόπο συμβολισμού που εισήγαγαν οι Graham et al. (1979), το παραπάνω πρόβλημα περιγράφεται ως εξής :
$P2|b=\infty, r_j, on-line|C_{\max}$. Αυτό το μοντέλο προγραμματισμού εμφανίζεται στην παραγωγική διαδικασία κατασκευής ημιαγωγών στην σύγχρονη βιομηχανία. Ο προγραμματισμός εργασιών κατα παρτίδες έχει μελετηθεί απο τους ερευνητές τις τελευταίες δύο δεκαετίες, ενδεικτικά [P.Brucker et al. (1998), B.Chen et al. (2001), T.C.E Cheng et al. (2004), C.Y Lee et al. (1992), J.J Yuan et al. (2004)].

Λαμβάνοντας ως κριτήριο τη δυναμικότητα της κάθε μηχανής, υπάρχουν δύο μοντέλα κατηγοριοποίησης. Ένα είναι το φραγμένο μοντέλο, στο οποίο το όριο b είναι πεπερασμένο ($b<\infty$). Μοντέλα αυτού του τύπου συναντώνται κατά την διαδικασία εισαγωγής σε κλιβάνους παρτίδων ολοκληρωμένων κυκλωμάτων, τα οποία στη συνέχεια εκτίθενται σε υψηλή θερμοκρασία. Κάθε κύκλωμα έχει έναν προκαθορισμένο χρόνο έκθεσης και ο κάθε κλίβανος έχει περιορισμένη χωρητικότητα.

Το δεύτερο ειναι το μη - φραγμένο μοντέλο, όπου $b=\infty$. Μοντέλα αυτού του τύπου συναντώνται σε καμίνους, όπως ασβεστοκαμίνους ή τσιμεντοκαμίνους, όπου πρώτες ύλες υφίστανται επεξεργασία (θερμαίνονται) και η κάμινος έχει αρκετά μεγάλη χωρητικότητα, έτσι ώστε να μην περιορίζει το μέγεθος των παρτίδων.

Όσον αφορά τα χαρακτηριστικά των πληροφοριών που είναι γνωστά πριν την ανάθεση των εργασιών, ο προγραμματισμός κατά παρτίδες διακρίνεται σε δύο κατηγορίες.

Η πρώτη είναι ο offline προγραμματισμός, κατά τον όποιο υπάρχει μία βασική θεώρηση ότι ο υπεύθυνος του προγραμματισμού έχει πλήρη γνώση των δεδομένων του προβλήματος, όπως ο συνολικός αριθμός εργασιών προς ανάθεση, οι χρόνοι άφιξης και επεξεργασίας, προτού οι αλγόριθμοι επίλυσης εφαρμοστούν.

Η δεύτερη είναι ο online προγραμματισμός, στον οποίο, σε αντίθεση με τον άνω, σε οποιαδήποτε χρονική στιγμή t, ο υπεύθυνος του προγραμματισμού γνωρίζει μόνο τις εργασίες που έχουν ήδη φτάσει και δεν γνωρίζει τα στοιχεία αυτών που πρόκειται να καταφθάσουν στο μέλλον. Σε αυτή την περίπτωση, οι εργασίες πρέπει να ανατεθούν με βάση τα υπάρχοντα στοιχεία.

Σε πολλά προβλήματα online προγραμματισμού, εξαιτίας αυτής της έλλειψης γνώσης πληροφοριών εκ των προτέρων, δεν είναι δυνατό να υπάρξουν αλγόριθμοι αυτής της κατηγορίας που θα δώσουν πάντοτε την βέλτιστη λύση. Γι'αυτό οι ερευνητές ασχολήθηκαν με προσεγγιστικούς online αλγορίθμους για αυτού του είδους τα προβλήματα. Η αποτελεσματικότητα ενός online αλγορίθμου δίνεται τυπικά από τον λόγο ανταγωνιστικότητας (competitive ratio) : όσο πλησιέστερα στο 1, τόσο πιο ακριβής. Λέμε ότι ενας αλγόριθμος έχει λόγο ανταγωνιστικότητας ρ (ή είναι ρ-competitive) εάν για οποιαδήποτε είσοδο, πάντα επιστρέφει μία εφικτή λύση με αντικειμενική τιμή το πολύ ρ - φορές μεγαλύτερη (για πρόβλημα ελαχιστοποίησης) από την βέλτιστη offline λύση του ίδιου προβλήματος.

Δηλαδή : $\rho = \sup_{\forall L}\{C_A(L)/C^*(L)\}$, όπου :

L : είναι η αλληλουχία των εργασιών

$C_A(L)$: ο χρόνος ολοκλήρωσης του online αλγορίθμου

$C^*(L)$: ο χρόνος ολοκλήρωσης του βέλτιστου offline αλγορίθμου

Παρατηρούμε ότι το ρ είναι το ανώτατο του λόγου $C_A(L)/C^*(L)$ για κάθε L και ακόμα ότι πάντα $\rho > 1$. Από την άλλη μεριά, ένα κατώτερο όριο για το πρόβλημα σημαίνει ότι δεν μπορεί να υπάρξει online αλγόριθμος με λόγο ρ μικρότερο από αυτή την τιμή. Συνεπώς, ένας online αλγόριθμος θα είναι βέλτιστος όταν ο λόγος του ρ ταυτίζεται με το κατώτερο όριο για το συγκεκριμένο πρόβλημα.

ΚΕΦΑΛΑΙΟ 2 : ΠΕΡΙΓΡΑΦΗ ΠΡΟΒΛΗΜΑΤΟΣ

Στην παρούσα εργασία, θα εξετάσουμε τους αλγορίθμους online προγραμματισμού τεσσάρων ερευνητών [Liu et al. (2011), Nong et al. (2008), Tian et al. (2009a), Tian et al. (2009b)], θα συγκρίνουμε τα αποτελέσματα των επιμέρους αλγορίθμων και θα καταλήξουμε στον βέλτιστο, εκείνον δηλαδη του οποίου η λύση είναι πλησιέστερα στη λύση του ολικά βέλτιστου offline αλγορίθμου.

Όπως αναφέρθηκε και στην εισαγωγή, υπάρχει εκτεταμένη βιβλιογραφία στον online προγραμματισμό εργασιών κατά παρτίδες σε m παράλληλες μηχανές. Οι Zhang et al. (2003) θεώρησαν την ειδική περίπτωση όπου όλες οι εργασίες έχουν ομοιόμορφο χρόνο επεξεργασίας. Πρότειναν και για το φραγμένο μοντέλο ($b < \infty$) και για το μη ($b = \infty$), βέλτιστους online αλγορίθμους με λόγους ανταγωνιστικότητας $\rho = \frac{\sqrt{5}+1}{2}$ και $1+\beta_m$ αντίστοιχα, όπου β_m είναι η θετική λύση της εξίσωσης $(1+\beta_n)^{m+1} = \beta_m + 2$. Οι Nong et al. (2008) θεώρησαν το φραγμένο μοντέλο σε 2 παράλληλες μηχανές και πρότειναν έναν αλγόριθμο με ρ = $\sqrt{2}$. Οι Tian et al. (2009a) παρουσίασαν ενα κατώτερο όριο επίσης $\sqrt{2}$ και επιπλέον παρουσίασαν έναν online αλγόριθμο αρτιότερα δομημένο.

Εδώ, θα εστιάσουμε στην περίπτωση όπου m=2 και θα μελετήσουμε μια πιο γενική περίπτωση από αυτή των Zhang et al. (2003), στην οποία για δεδομένη είσοδο (εργασίες), κάθε εργασία έχει χρόνο επεξεργασίας p_j, ο οποίος είναι ίσος με α - φορές ($a > 1$) τον χρόνο της προηγούμενης. Δηλαδή, $p_{j+1} = ap_j$. Όταν το α=1, τότε έχουμε την περίπτωση ομοιόμορφου (uniform) χρόνου επεξεργασίας.

Στόχος μας είναι η ελαχίστοποιηση του χρόνου ολοκλήρωσης όλων των εργασιών. Όπως αναφέρθηκε και προηγούμενα, μελετάμε την μη- φραγμένη περίπτωση όπου όλες οι εργασίες μπορούν να επεξεργαστούν σε μία παρτίδα, δηλαδή $b = \infty$.

Οι εργασίες που υφίστανται επεξεργασία μαζί σχηματίζουν μία παρτίδα και όλες οι εργασίες της παρτίδας ξεκινούν και τελειώνουν στον ίδιο χρόνο. Ο συνολικός χρόνος επεξεργασίας της κάθε παρτίδας είναι ο χρόνος επεξέργασίας της μεγαλύτερης σε διάρκεια εργασίας της, συνεπώς, της τελευταίας.

Μας δίνονται 2 παράλληλες μηχανές 11, 12 και οι εργασίες που καταφθάνουν συμβολίζονται με $U = \{J_1, J_2, ..., J_n\}$, όπου κάθε εργασία J_j αποδεσμεύεται τον χρόνο r_j και τα r_j, n δεν είναι εκ των προτέρων γνωστά. Ο χρόνος p_j γίνεται γνωστός την στιγμή r_j. Χωρίς απώλεια της γενικότητας, υποθέτουμε ότι μόνο μία εργασία απελευθερώνεται κάθε χρονική στιγμή t, δηλαδή ότι $r_j < r_{j+1}$, για $1 < j < n-1$. Ακόμη, γίνεται αντιληπτό ότι οι χρόνοι επεξεργασίας των εργασιών είναι αύξουσα συνάρτηση του χρόνου.

Στο σημείο αυτό, θα αναφερθούμε στον ολικά βέλτιστο offline αλγόριθμο και στην διαδικασία λήψης αποτελεσμάτων από αυτόν. Αναφέρθηκε προηγουμένως ότι, εάν τα δεδομένα εισόδου είναι γνωστά εκ των προτέρων, το πρόβλημα γίνεται offline. Είναι ξεκάθαρο ότι το μη-φραγμένο πρόβλημα μπορεί να λυθεί τετριμμένα προγραμματίζοντας όλες τις εργασίες μίας παρτίδας στον χρόνο άφιξης της τελευταίας εργασίας. Στους κώδικες που παρατίθενται στο παράρτημα, τον ολικά βέλτιστο χρόνο ολοκλήρωσης του προβληματος (optimal makespan) τον υπολογίσαμε ίσο με $r_N + p_N$, δηλαδή τον χρόνο απελευθέρωσης της τελευταίας εργασίας + τον χρόνο επεξεργασίας της.

ΚΕΦΑΛΑΙΟ 3 : ΑΛΓΟΡΙΘΜΟΙ ΕΠΙΛΥΣΗΣ

Σε αυτήν την ενότητα, θα παρουσιάσουμε τους 4 αλγορίθμους των ερευνητών που μας απασχόλησαν.

3.1. Αλγόριθμος Liu et al. (2011)

Θεωρώντας την απόδειξη του κατώτερου ορίου $\frac{C_{ON}}{C_{OPT}} > \varphi$, αντιλαμβανόμαστε ότι ένας online αλγόριθμος προγραμματισμού δεν θα ξεκινήσει κάθε εργασία αμέσως μετά την έλευσή της, προκειμένου να έχει καλή ανταγωνιστικότητα. Βασιζόμενοι στον αλγόριθμο Modified-Sleepy των Tian et al. (2009a), οι Liu et al. (2011) παρουσιάζουν έναν νέο αλγόριθμο A_{α}^{x} (χωρητικότητας b=∞ και με $\frac{p_{j+1}}{p_j} = \alpha$). Συμβολίζουμε με U(t) το σύνολο των απρογραμμάτιστων εργασιών τον χρόνο t και με $J_{(t)}$ την μεγαλύτερη (ή τελευταία αφιχθήσα) εργασία στο U(t). Έστω $p_{(t)}, r_{(t)}$ η διάρκεια και η άφιξη της $J_{(t)}$ αντίστοιχα. Παρατηρούμε ότι η $J_{(t)}$ δεν ορίζεται εάν το U(t)=∅. Θεωρούμε τις 2 μηχανές την χρονική στιγμή t. Εάν μόνο μία μηχανή είναι απασχολημένη, ορίζουμε αυτήν την παρτίδα ως $B^{\cdot}(t)$ και τον χρόνο έναρξής της ως $S^{\cdot}(t)$, ενώ εάν και οι 2 μηχανές είναι ελεύθερες, τότε $S^{\cdot}(t)$ =0.
Οι Liu et al.(2011) έδωσαν ένα κατώτερο όριο του προβλήματος : $P2|online, r_j, B=\infty, nondecreasing|C_{\max}$. Απέδειξαν ότι δεν υπάρχει online αλγόριθμος για αυτό το πρόβλημα, με λόγο ανταγωνιστικότητας ρ μικρότερο του φ. Με $\varphi > 1$, όρισαν την θετική λύση της εξίσωσης $x^3 + (a-1)x^2 + (a^2 - a - 1)x - a^2 = 0$. Για τιμές του α (>1), οι οποίες επιλέγονται ως είσοδος, λαμβάνονται οι ρίζες της άνω τριτοβάθμιας. Από αυτές, επέλεξαν την θετική ρίζα για τις ανάγκες του προβλήματός μας. Απέδειξαν ακόμα ότι $1 < \varphi < 1 + \frac{1}{2\alpha}$. Προκειμένου να αποδείξουν κατά αντίφαση ότι ο αλγόριθμος A_{α}^{∞} έχει λόγο ανταγωνιστικότητας φ, υπέθεσαν ότι υπάρχει ένα μικρότερο υποσύνολο I, το οποίο αποτελείται από ένα ελάχιστο πλήθος εργασιών. Για αυτό το υποσύνολο, η διάρκεια του προγραμματισμού του A_{α}^{∞} συγκρινόμενη με αυτην του βέλτιστου offline αλγορίθμου είναι αυστηρά μεγαλύτερη του φ.

Με σ και π συμβόλισαν τα προγράμματα των αλγορίθμων A_{α}^{*} και offline αντίστοιχα, για το υποσύνολο I. Θεώρησαν $C_{\max}(\sigma)$ και $C_{\max}(\pi)$ τις αντικειμενικές τιμές τους αντίστοιχα. Ο λόγος τους είναι ο γνωστός μας λόγος ανταγωνιστικότητας ρ. Υπέθεσαν ακόμα, ότι υπάρχουν συνολικά n παρτίδες στο σ και ταξινομούνται κατα αύξουσα σειρά των χρόνων ολοκλήρωσης, δηλαδή $B_1,...,B_N$.

Με S_i και C_i συμβόλισαν τους χρόνους έναρξης και ολοκλήρωσης των B_i, αντίστοιχα. Με την υπόθεση των αύξοντων χρόνων επεξεργασίας και της χωρητικότητας $b=\infty$ των παρτίδων, έδειξαν ότι υπάρχει ακριβώς μια εργασία σε κάθε παρτίδα του $\sigma=\{B_1,...,B_N\}$.

Ακόμα, θεώρησαν ότι μία εργασία J_i λέγεται **κανονική** εάν είναι προγραμματισμένη τον χρόνο $t=\max\{\varphi S^{*}(S_i)+(\varphi-1)p_{(i)}, r_{(i)}\}$.

Διαφορετικά, εάν $S_i > t=\max\{\varphi S^{*}(S_i)+(\varphi-1)p_{(i)}, r_{(i)}\}$, τότε η J_i θα λέγεται **αργοπορημένη.**

Συμπερασματικά, η κύρια συνεισφορά των Liu et al. (2011) ήταν ένας βέλτιστος online αλγόριθμος με λόγο ανταγωνιστικότητας φ ($1<\varphi\le 1+\frac{1}{2\alpha}$), που είναι η θετική ρίζα της εξίσωσης $\varphi^3+(\alpha-1)\varphi^2+(\alpha^2-\alpha-1)\varphi-\alpha^2=0$. Τα παραπάνω γενικεύουν τα αποτελέσματα των Zhang et al. (2003).

Ο αλγόριθμος αυτός έχει ως εξής :

Βήμα 1: Κάθε στιγμή t, κατά την οποία τουλάχιστον μία μηχανή είναι ελεύθερη και τουλάχιστον μία εργασία είναι διαθέσιμη (Εάν όλες οι εργασίες έχουν ανατεθεί, ακύρωσε την ανάθεση), ανανέωσε το U(t).

Βήμα 2.1: Εάν το $t\ge\max\{\varphi S^{*}(t)+(\varphi-1)p_{(t)}, r_{(t)}\}$, ανάθεσε όλες τις εργασίες στο U(t) ως μία παρτίδα σε μία διαθέσιμη μηχανή ;

Βήμα 2.2: Διαφορετικά, περίμενε έως ότου μία νέα εργασία φτάσει ή μέχρι τον χρόνο $t'=\max\{\varphi S^{*}(t')+(\varphi-1)p_{(t')}, r_{(t')}\}$, όποιο από τα δύο συμβεί πρώτο.

Βήμα 3: Πήγαινε στο Βήμα 1.

Ο ψευδοκώδικας του αλγορίθμου (η ερμηνεία των συμβολισμών παρατίθεται στο Παράρτημα II) είναι ο ακόλουθος :

```
int U = 0;   int i = 0;   int total = 0;  int I1 = I2 = 0;
real t = 0;  real makespan = 0;  real S1 = S2 = 0;  real F1 = F2 = 0;  real t_fin;

while (total ≤ N)  { // while not all jobs have been scheduled

        if (U == 0 && i < N) {  // if no unscheduled jobs, go to next arrival
                i = i + 1;  t = r_i;  U = i;
                if (I1 == 1 && F1 ≤ t) {I1 = 0;  S1 = 0;  F1 = 0;}
                if (I2 == 1 && F2 ≤ t) {I2 = 0;  S2 = 0;  F2 = 0;}
        } // end if

        if  (I1 + I2 == 2) { // if both machines busy, go to first machine release
                if  (F1 < F2) { t = F1;  S1 = 0;  F1 = 0;  I1 = 0; }
                else  { t = F2;  S2 = 0;  F2 = 0;  I2 = 0; }
                while ((i + 1) ≤ N &&  r_{i+1} ≤ t) { i = i + 1; U = i; }
        } // end if

        S* = max (S1, S2)

        if  (t ≥ max (φS* + (φ-1)p_u, r_u) { //if condition true schedule jobs on 1st free machine
                if (I1 == 0) {
                        I1 =1; S1 = t;  F1 = S1 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F1; total = total + 1; }
                } // end if
                else if (I2 == 0) {
                        I2 = 1; S2 = t; F2 = S2 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F2; total = total + 1; }
                } // end else if
        } // end if
        else {  // if condition true

                if  (I1 == 1  &&  I2 == 1)  t_fin = min(F1, F2);
                else if (I1 == 1  &&  I2 == 0) t_fin = F1;
                else if (I1 == 0  &&  I2 == 1) t_fin = F2;
                else if (I1 == 0  &&  I2 == 0)  t_fin = oo; //oo = infinity (something very big)

                if ((i+1) ≤ N) { // if more jobs exist
                        t = min (t_fin, r_{i+1}, max (φS* + (φ-1)p_u, r_u));  // jump to next event
                        if (t == r_{i+1}) { i = i + 1; U = i; }
                        else if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end if
                else {  // if no more jobs exist
                        t = min (t_fin, max (φS* + (φ-1)p_u, r_u));  // jump to next event
                        if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end else
        } // end else
} // end while

print  makespan;
```

3.2. Αλγόριθμος Nong et al. (2008)

Εδώ, οι Nong et al. (2008) μελέτησαν την περίπτωση $P2|b=\infty, r_j, on-line|C_{\max}$. Με βάση τα αποτελέσματα των Zhang et al. (2003), δεν υπάρχει online αλγόριθμος με λόγο ανταγωνιστικότητας $< 1+\gamma_2$, όπου $\gamma_2 \approx 0.325$ είναι η λύση της εξίσωσης $(1+\gamma_2)^3 = \gamma_2 + 2$. Πρότειναν έναν αλγόριθμο που είναι καλύτερος από αυτόν των Zhang et al. (2001), ο οποίος βελτιώνει τον λόγο ρ από $\frac{3}{2}$ σε $\sqrt{2}$. Θεώρησαν μία απλή ακολουθία I, στην οποία κάθε εργασία έχει χρόνο επεξεργασίας ίσο με 1. Η πρώτη εργασία J_1 καταφθάνει τον χρόνο 0. Εάν η J_1 υφίσταται επεξεργασία την χρονική στιγμή $\sqrt{2}-1$ ή μετά, δεν έρχεται καμία άλλη εργασία στο I. Αυτό έχει ως συνέπεια ένα πρόγραμμα με διάρκεια (makespan) τουλάχιστον ίσο με $\sqrt{2}$, την στιγμή που η βέλτιστη διάρκεια είναι 1. Εάν ο αλγόριθμος $A_2(a)$ ξεκινήσει την J_1 πριν το χρόνο $\gamma_2 \approx 0.325$, έστω την στιγμή t_1, τότε η δεύτερη εργασία J_2 απελευθερώνεται σε χρόνο t_1+ε, όπου ε ένας μικρός θετικός αριθμός. Υπέθεσαν ότι ο $A_2(a)$ ξεκινά την εργασία J_2 την στιγμή t_2. Εάν $\frac{1+t_2}{1+t_1} \geq 1+\gamma_2$, καμία άλλη εργασία δεν φθάνει στο I. Τότε, ο $A_2(a)$ παράγει ένα πρόγραμμα με διάρκεια $1+t_2$ και η βέλτιστη διάρκεια είναι $1+t_1$+ε, που σημαίνει ότι, όταν $\varepsilon \to 0$, ο λόγος ρ είναι τουλάχιστον $\frac{1+t_2}{1+t_1+\varepsilon} \to \frac{1+t_2}{1+t_1} \geq 1+\gamma_2$.

Εάν $\frac{1+t_2}{1+t_1} < 1+\gamma_2$, τότε η τρίτη εργασία J_3 φτάνει τον χρόνο t_2+ε. Τότε ο $A_2(a)$ επιστρέφει ένα πρόγραμμα τουλάχιστον $2+t_1$ και η βέλτιστη διάρκεια είναι $1+t_2$+ε. Συνεπώς, όταν το $\varepsilon \to 0$, ο λόγος ρ είναι τουλάχιστον $\frac{2+t_1}{1+t_2+\varepsilon} \to \frac{2+t_1}{1+t_2} > \frac{2+t_1}{(1+\gamma_2)(1+t_1)} > \frac{2+\gamma_2}{(1+\gamma_2)^3} = 1+\gamma_2$, όπου η τελευταία ανισότητα προκύπτει από το γεγονός ότι η $\frac{2+t_1}{(1+\gamma_2)(1+t_1)}$ είναι φθίνουσα συνάρτηση και $t_1 < \gamma_2$ και η ισότητα από την $(1+\gamma_2)^3 = \gamma_2 + 2$. Από τα παραπάνω, γίνεται αντιληπτό ότι για να έχουμε καλύτερο λόγο ανταγωνιστικότητας, οι εργασίες σε μια ακολουθία θα πρέπει να περιμένουν για ένα εύλογο διάστημα, προτού δοθούν προς ανάθεση. Έστω J(t) η εργασία με τον μεγαλύτερο χρόνο επεξεργασίας ανάμεσα στις εργασίες που είναι διαθέσιμες αλλά δεν έχουν ανατεθεί τον χρόνο t. Εάν υπάρχουν 2 ή περισσότερες με το ίδιο p(t), τότε θεώρησαν ως J(t) αυτήν με τον μεγαλύτερο χρόνο r(t). Ακόμη, θεώρησαν ότι $\alpha = \sqrt{2}-1$, όπου α η λύση της εξίσωσης $\alpha^2 + 2\alpha = 1$.

Η βασική ιδέα του αλγορίθμου είναι να εφαρμόζει μια τακτική «delay and greedy» και ολοι οι χρόνοι έναρξης των παρτίδων που προκύπτουν από τον προγραμματισμό να είναι διαφορετικοί. Δεδομένης μιας ακολουθίας εργασιών, έστω σ το πρόγραμμα του $A_2(a)$ και π το πρόγραμμα του βέλτιστου offline αλγορίθμου (optimal-algorithm). Για ένα πρόγραμμα χ, έστω $C_{\max}(x)$ η διάρκειά του και s_j ο χρόνος έναρξης της εργασίας J_j. Για δύο εργασίες J_i και J_j σε 2 διαφορετικές παρτίδες στο σ, εάν $s_i > s_j$, τότε από την περιγραφή του αλγορίθμου $A_2(a)$, όλες οι εργασίες στην παρτίδα την οποία η J_i έχει ανατεθεί απελευθερώνονται μετά το χρόνο s_j, υπονοώντας ότι (1) $s_i > (1+a)s_j + ap_i$ άρα και $s_i - s_j > ap_i$ και (2) $C_{\max}(\pi) > s_j + p_i$. Θεώρησαν ως χρόνο έναρξης μιας παρτίδας τον s. Υποστηρίζουν ότι μια παρτίδα λέγεται κανονική, εάν ξεκινά την στιγμή $(1+a)r(s)+ap(s)$. Αντίστοιχα, μία εργασία λέγεται μη-κανονική αν s > $(1+a)r(s)+ap(s)$, που σημαίνει ότι και οι δύο μηχανές είναι απασχολημένες την χρονική στιγμή $(1+a)r(s)+ap(s)$.

Συνοπτικά : Ο συγκεκριμένος αλγόριθμος των Nong et al. (2008) αποδίδει καλύτερα από εκείνον των Zhang et al. (2001), ο οποίος έχει λόγο ανταγωνιστικότητας $\frac{3}{2}$. Θα ήταν ενδιαφέρον να διαπιστωθεί, εάν για το παρόν πρόβλημα υπάρχει online αλγόριθμος με κατώτερο όριο $1+\gamma_2 \approx 1.325$, που ταυτίζεται δηλαδή με εκείνο των Zhang et al. (2003), ή εάν το κατώτερο όριο μπορεί να βελτιωθεί περαιτέρω.

Ο αλγόριθμος $A_2(a)$ έχει ως εξής:

Την στιγμή t, εάν μια μηχανή είναι ελεύθερη και υπάρχουν εργασίες διαθέσιμες αλλά όχι ακόμα προγραμματισμένες και $t > (1+a)r(t)+ap(t)$, τότε ξεκίνα όλες τις διαθέσιμες εργασίες ως μία παρτίδα στην μηχανή με τον μικρότερο χρόνο ολοκλήρωσης έως τώρα ; αλλιώς μην κάνεις τίποτα και περίμενε.

Ο ψευδοκώδικας του αλγορίθμου είναι ο εξής:

```
int U = 0;   int i = 0;   int total = 0;  int I1 = I2 = 0;   real a = 2^0.5 - 1  (= ρίζα 2 μείον 1)
real t = 0;  real makespan = 0;  real S1 = S2 = 0;  real F1 = F2 = 0;  real t_fin;

while (total ≤ N)  { // while not all jobs have been scheduled

        if (U == 0 && i < N) {  // if no unscheduled jobs, go to next arrival
                i = i + 1;  t = r_i;  U = i;
                if (I1 == 1 && F1 ≤ t) {I1 = 0;  S1 = 0;  F1 = 0;}
                if (I2 == 1 && F2 ≤ t) {I2 = 0;  S2 = 0;  F2 = 0;}
        } // end if

        if  (I1 + I2 == 2) { // if both machines busy, go to first machine release
                if  (F1 < F2) { t = F1;  S1 = 0;  F1 = 0;  I1 = 0; }
                else  { t = F2;  S2 = 0;  F2 = 0;  I2 = 0; }
                while ((i + 1) ≤ N &&  r_{i+1} ≤ t) { i = i + 1; U = i; }
        } // end if

        if  (t ≥ (I + a)r_u + a p_u)  { //if condition true schedule jobs on 1st free machine
                if (I1 == 0) {
                        I1 =1; S1 = t;  F1 = S1 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F1; total = total + 1; }
                } // end if
                else if (I2 == 0) {
                        I2 = 1; S2 = t; F2 = S2 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F2; total = total + 1; }
                } // end else if
        } // end if
        else {  // if condition not true

                if  (I1 == I  &&  I2 == I)  t_fin = min(F1, F2);
                else if (I1 == 1  &&  I2 == 0) t_fin = F1;
                else if (I1 == 0  &&  I2 == 1) t_fin = F2;
                else if (I1 == 0  &&  I2 == 0)  t_fin = oo; //oo = infinity (something very big)

                if ((i+1) ≤ N) { // if more jobs exist
                        t = min (t_fin, r_{i+1}, (1 + a)r_u + a p_u);  // jump to next event
                        if (t == r_{i+1}) { i = i + 1; U = i; }
                        else if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end if
                else {  // if no more jobs exist
                        t = min (t_fin, (I + a)r_u + a p_u);  // jump to next event
                        if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end else
        } // end else
} // end while

print  makespan;
```

3.3. Αλγοριθμος Tian et al. (2009a)

Για online προγραμματισμό σε m=2 μηχανές με στόχο την ελαχιστοποίση της διάρκειας ολοκλήρωσης, οι Chen & Vestjens (1997) πρότειναν έναν online LPT-αλγόριθμο (largest processing time) με λόγο $\rho = \frac{3}{2}$ και απέδειξαν ότι ένας οποιοσδήποτε online αλγόριθμος έχει λόγο αναταγωνιστικότητας τουλάχιστον 1.3473, ενώ για m=2, το κατώτερο όριο είναι $\frac{5-\sqrt{5}}{2}$. Για m=2, οι Noga & Seiden (2001) πρότειναν έναν βέλτιστο online αλγόριθμο με $\rho = \frac{5-\sqrt{5}}{2}$. Στην offline εκδοχή του προβλήματος, για n εργασίες που ανατίθενται σε m παράλληλες μηχανές, οι Li et al. (2006) πρότειναν μια προσεγγιστική πολυωνυμική εκδοχή (PTAS-polynomial time approximation scheme). Στον online προγραμματισμό, οι Zhang et al. (2003) πρότειναν για το προβλημα του μη φραγμένου μεγέθους παρτίδας, έναν εφικτό αλγόριθμο με $\rho=1+\beta_m$, όπου β_m η θετική ρίζα της εξίσωσης : $(1+\beta_m)^{m+1} = \beta_m + 2$. Εδώ, για απεριόριστο μέγεθος παρτίδας, οι Tian et al. (2009a) έδειξαν ότι το κατώτερο όριο του λόγου ρ είναι $\sqrt{2}$, επαληθεύοντας ότι ο online αλγόριθμος που πρότειναν οι Nong et al. (2008) είναι ο καλυτερος δυνατός. Πρότειναν έναν νέο αλγόριθμο, τον Modified-Sleepy και απέδειξαν οτι έχει καλύτερη δομή από αυτόν του Nong. Απέδειξαν ακόμα, ότι, για ένα οποιοδήποτε πρόβλημα online προγραμματισμού σε 2 παράλληλες μηχανές με απεριόριστη χωρητικότητα, δεν υπαρχει online αλγόριθμος με λόγο ανταγωνιστικότητας ρ μικρότερο του $\sqrt{2}$.

Ο νέος αλγόριθμος

Επειδή ο αλγόριθμος είναι non-preemptive, όταν δύο μηχανές είναι απασχολημένες, θα πρέπει να περιμένει τουλάχιστον έως ότου μια μηχανή μείνει ελεύθερη. Επομένως, πρέπει να μπορεί να διαχειριστεί την περίπτωση, κατά την οποία και οι 2 μηχανές είναι ελεύθερες και την περίπτωση κατά την οποία μια μηχανή απασχολείται και η άλλη είναι ελεύθερη. Το μέγεθος παρτίδας είναι άπειρο ($b=\infty$). Κάθε χρονική στιγμή έναρξης των παρτίδων, ο αλγόριθμος θα επεξεργαστεί όλες τις απρογραμμάτιστες διαθέσιμες εργασίες ως μία παρτίδα. Άρα, κάθε χρονική στιγμή, υπάρχει το πολύ μία παρτίδα που ξεκινά να υφίσταται επεξεργασία. Έστω U(t) το σύνολο των απρογραμμάτιστων εργασιών τον χρόνο t και J(t) η μεγαλύτερη σε διάρκεια εργασία του συνόλου U που καταφθάνει τελευταία. Με p(t) και r(t) συμβολίζουμε τους χρόνους επεξεργασίας και άφιξης της J(t) αντίστοιχα.
Εάν τον χρόνο t, μόνο μία μηχανή εργάζεται σε μια παρτίδα, συμβολίζουμε αυτήν την παρτίδα με $B^*(t)$ και θεωρούμε ότι έχει χρονο έναρξης $S^*(t)$ και επεξεργασίας $p^*(t)$. Εάν την στιγμή t, και οι 2 μηχανές είναι ελεύθερες, τότε $S^*(t)=p^*(t)=0$.
Με $a=\sqrt{2}-1$ ορίζουμε την θέτικη ρίζα της εξίσωσης $a^2+2a-1=0$.

Γίνεται αντιληπτό, ότι, για κάθε χρονική στιγμή που πρέπει να αποφανθούμε, η δράση του αλγορίθμου Modified-Sleepy εξαρτάται όχι μόνο απο την πληροφορία της τελευταίας εργασίας του U(t), αλλά και από την τρέχουσα παρτίδα $B^*(t)$, σε αντίθεση με τον αλγόριθμο των Nong et al. (2008), όπου αξιοποιείται μόνο η τελευταία αφιχθείσα εργασία στο U(t).

Συνεπώς, παρότι και οι δύο αλγόριθμοι εφαρμοζουν μια τακτική αργοπορίας, η καθυστέρηση που προκύπτει απο τον Modified-Sleepy είναι μικρότερη από την αντιστοιχη των Nong et al. (2008) στις περισσότερες περιπτώσεις. Άρα, η στρατηγική του αλγορίθμου Modified-Sleepy φαίνεται πιο λογική, άρα έχει και αρτιότερη δομή.

Με M_1, M_2 οι Tian et al. (2009a) συμβόλισαν τις 2 παράλληλες μηχανές και με σ και π τα προγράμματα των Modified-Sleepy και βέλτιστου offline αλγορίθμου, αντίστοιχα. Τα εξαγόμενα των δυο αλγορίθμων είναι τα $C_{\max}(\sigma)$ και $C_{\max}(\pi)$, αντίστοιχα.Υπέθεσαν, ακόμα, ότι υπάρχουν n παρτίδες στο σ, οι οποίες συμβολίζονται ως $B_1, B_2, ..., B_n$.

Για κάθε i, η τελευταία μεγαλύτερης διάρκειας εργασία στην B_i γράφεται ως J_t και με p_i και r_i, οι χρόνοι επεξεργασίας και άφιξης αντίστοιχα. Με S_i και C_i συμβολίζονται οι χρόνοι έναρξης και ολοκλήρωσης της παρτίδας B_i, αντίστοιχα. Υπέθεσαν ότι $C_1 \le C_2 \le .... \le C_n$. Παρατήρησαν ότι $S_i < S_j$, σημαίνει ότι $r_j \ge S_i$. Το εξαγόμενο $C_{\max}(\sigma)$ αναφέρεται στην παρτίδα B_t, δηλαδή $C_{\max}(\sigma) = C_n$. Χωρίς απώλεια της γενικότητας, θεώρησαν την B_n να προγραμματίζεται στην μηχανή M_1. Στο πρόγραμμα σ, εάν υπάρχει κάποια παρτίδα με χρόνο έναρξης μεγαλύτερο του S_n, ακυρώνουν αυτή την παρτίδα, αφού το αποτέλεσμα του αλγορίθμου έχει επιτευχθεί με την παρτίδα B_n. Αυτό δεν θα επηρέαζε το $C_{\max}(\sigma)$, αλλά ίσως μείωνε το $C_{\max}(\pi)$. Ο λόγος ανταγωνιστικότητας δεν θα μειωνόταν. Θεώρησαν ότι δεν καταφθάνουν άλλες εργασίες μετά τον χρόνο S_n.

Ακόμα, υποστήριξαν ότι μία εργασία J_i λέγεται <u>κανονική</u> στο σ εάν ο χρόνος έναρξής της ισούται με : $S_i = \max\{S^*(S_i) + ap^*(S_i), ap_i, r_i\}$, ενώ εάν η J_i είναι <u>μη- κανονική</u> στο σ, τότε για κάθε J_j με $S_j \le S_i$, $S_i \ge S_j + ap_j$.

<u>Ο αλγόριθμος Modified-Sleepy έχει ως εξής</u> :

Την χρονική στιγμή t, εάν μια μηχανή είναι ελεύθερη, $U(t) \neq \varnothing$ και $t > \max\{ap(t), S^*(t) + ap^*(t)\}$, τότε ξεκίνα το U(t) ως μία παρτίδα στην μηχανή ;
αλλιώς μην κάνεις τίποτα και περίμενε.

Ο ψευδοκώδικας του αλγορίθμου Modified – Sleepy είναι ο εξής :

```
int U = 0;   int i = 0;   int total = 0;  int I1 = I2 = 0;   real a = 0.414213562;  real t = 0;
real makespan = 0;  real S1 = S2 = 0;  real F1 = F2 = 0;  real t_fin; real S* = 0;  real p* = 0;

while (total ≤ N)  { // while not all jobs have been scheduled

        if (U == 0 && i < N) {  // if no unscheduled jobs, go to next arrival
                i = i + 1;  t = r_i;  U = i;
                if (I1 == 1 && F1 ≤ t) {I1 = 0;  S1 = 0;  F1 = 0;}
                if (I2 == 1 && F2 ≤ t) {I2 = 0;  S2 = 0;  F2 = 0;}
        } // end if

        if  (I1 + I2 == 2) { // if both machines busy, go to first machine release
                if  (F1 < F2) { t = F1;  S1 = 0;  F1 = 0;  I1 = 0; }
                else  { t = F2;  S2 = 0;  F2 = 0;  I2 = 0; }
                while ((i + 1) ≤ N &&  r_{i+1} ≤ t) {
                        i = i + 1; U = i;
                }
        } // end if

        if (I1+I2 == 0)  { S* = 0; p* = 0; }
        else if (I1 == 1) { S* = S1; p* = F1 – S1; }
        else if (I2 == 1)  { S* = S2; p* = F2 – S2; }

        if  (t ≥ max[a p_u, S* + a p*] )  { //if condition true schedule jobs on 1st free machine
                if (I1 == 0) {
                        I1 =1; S1 = t;  F1 = S1 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F1; total = total + 1; }
                } // end if
                else if (I2 == 0) {
                        I2 = 1; S2 = t; F2 = S2 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F2; total = total + 1; }
                } // end else if
        } // end if

        else {  // if condition not true
                if  (I1 == 1  &&  I2 == 1)  t_fin = min(F1, F2);
                else if (I1 == 1  &&  I2 == 0) t_fin = F1;
                else if (I1 == 0  &&  I2 == 1) t_fin = F2;
                else if (I1 == 0  &&  I2 == 0)  t_fin = oo; //oo = infinity (something very big)

                if ((i+1) ≤ N) {  // if more jobs exist
                        t = min (t_fin, r_{i+1}, max[a p_u, S* + a p*]);  // jump to next event
                        if (t == r_{i+1}) {
                                i = i + 1; U = i;
                        }
                        else if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end if
```

```
                else {  // if no more jobs exist
                        t = min (t_fin, max[a p_u, S* + a p*]);  // jump to next event
                        if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end else
        } // end else
} // end while
print   makespan;
```

3.4. Αλγόριθμος Tian et al. (2009b)

Για το πρόβλημα $Pm \mid p-batch, b=\infty, online \mid C_{\max}$, οι Tian et al. (2009b) έδωσαν ένα κατώτερο όριο $1+a_m$ για τον λόγο ανταγωνιστικότητας ρ και παρουσίασαν έναν νέο online αλγόριθμο Η(a_m), όπου a_n είναι η θετική λύση της εξίσωσης $a_m^2 + ma_m - 1 = 0$. Συμβόλισαν με r_j και p_j τους χρόνους άφιξης και επεξεργασίας της εργασίας J_j, αντίστοιχα. Για μία ακολουθία εργασιών I στο online πρόγραμμα του Η(a_m), S_j ο χρόνος έναρξης της εργασίας J_j και π το πρόγραμμα του βέλτιστου offline αλγορίθμου, με S^*_j τον χρόνο έναρξης της εργασίας J_j στο π. Τέλος, με $C_{\max}(\sigma)$ και $C_{\max}(\pi)$ συμβόλισαν τις διάρκειες των προγραμμάτων σ και π, αντίστοιχα.
Ακόμα, απέδειξαν για το πρόβλημα $Pm \mid p-batch, b=\infty, online \mid C_{\max}$, ότι δεν υπάρχει online αλγόριθμος με λόγο ανταγωνιστικότητας μικρότερο του $1+a_m$.
Τέλος, για το συγκεκριμένο πρόβλημα, απέδειξαν ότι ο αλγόριθμος Η(a_m) είναι ο καλύτερος δυνατός online αλγόριθμος με λόγο ανταγωνιστικότητας $1+a_m$.

Ένας νέος online αλγόριθμος

Σε προβλήματα online προγραμματισμού, μία εργασία J_j είναι διαθέσιμη σε χρόνο t, εάν έχει φτάσει αλλά δεν έχει προγραμματιστεί τον χρόνο t. Έστω U(t) το σύνολο των απρογραμμάτιστων εργασιών τον χρόνο t και J(t) η εργασία με την μεγαλύτερη διάρκεια (επεξεργασίας) στο U(t). Με p(t) και r(t) δηλώνονται οι χρόνοι επεξεργασίας και άφιξης της J(t) αντίστοιχα. Έστω $B^*(t)$ η παρτίδα με τον μεγαλύτερο χρόνο έναρξης πριν τον χρόνο t στο σ και $S^*(t)$ και $p^*(t)$ οι χρόνοι έναρξης και επεξεργασίας της $B^*(t)$, αντίστοιχα. Εάν η $B^*(t)$ δεν υπάρχει, τότε $S^*(t)=p^*(t)=0$. Επιπλέον, έστω $p_{\max}(t)$ ο μέγιστος χρόνος επεξεργασίας των εργασιών με χρόνο άφιξης το πολύ t.

Ο αλγόριθμος Η(a_m) έχει ως εξής :

Την χρονική στιγμή t, εάν μία μηχανή είναι ελεύθερη, $U(t) \neq \varnothing$ και $t \geq \eta(t) = S^*(t) + a_m p_{\max}(t)$, τότε ξεκίνα την U(t) ως μία παρτίδα στην μηχανή τον χρόνο t ; αλλιώς μην κάνεις τίποτα και περίμενε.

Ο ψευδοκώδικας του αλγορίθμου Η(a_m) είναι ο ακόλουθος :

```
int U = 0;  int i = 0;  int total = 0;  int I1 = I2 = 0;  real a = 0.414213562;
real t = 0;  real makespan = 0;  real S1 = S2 = 0;  real F1 = F2 = 0;  real t_fin; real S* = 0;

while (total ≤ N)  { // while not all jobs have been scheduled

        if (U == 0 && i < N) {  // if no unscheduled jobs, go to next arrival
                i = i + 1;  t = r_i;  U = i;
                if (I1 == 1 && F1 ≤ t) {I1 = 0;  S1 = 0;  F1 = 0;}
                if (I2 == 1 && F2 ≤ t) {I2 = 0;  S2 = 0;  F2 = 0;}
        } // end if
        if  (I1 + I2 == 2) { // if both machines busy, go to first machine release
                if  (F1 < F2) { t = F1;  S1 = 0;  F1 = 0;  I1 = 0; }
                else  { t = F2;  S2 = 0;  F2 = 0;  I2 = 0; }
                while ((i + 1) ≤ N &&  r_{i+1} ≤ t) {
                        i = i + 1; U = i;
                }
        } // end if

        if (I1+I2 == 0)  S* = 0;
        else if (I1 == 1)  S* = S1;
        else if (I2 == 1)  S* = S2;

        if  (t ≥ S* + a p_u)  { //if condition true schedule jobs on 1st free machine
                if (I1 == 0) {
                        I1 = I; S1 = t;  F1 = S1 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F1; total = total + 1; }
                } // end if
                else if (I2 == 0) {
                        I2 = 1; S2 = t; F2 = S2 + p_u;  total = U;  U = 0;
                        if  (total == N) { makespan = F2; total = total + 1; }
                } // end else if
        } // end if
        else {  // if condition not true
                if  (I1 == 1  &&  I2 == 1)  t_fin = min(F1, F2);
                else if (I1 == 1  &&  I2 == 0) t_fin = F1;
                else if (I1 == 0  &&  I2 == 1) t_fin = F2;
                else if (I1 == 0  &&  I2 == 0) t_fin = oo; //oo = infinity (something very big)

                if ((i+1) ≤ N) { // if more jobs exist
                        t = min (t_fin, r_{i+1}, S* + a p_u);  // jump to next event
                        if (t == r_{i+1}) {
                                i = i + 1; U = i;
                        }
                        else if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end if
                else {  // if no more jobs exist
                        t = min (t_fin, S* + a p_u);  // jump to next event
                        if (t == t_fin  && t_fin == F1) { I1 = 0;  S1 = 0;  F1 = 0; }
                        else if (t == t_fin  && t_fin == F2) { I2 = 0;  S2 = 0;  F2 = 0; }
                } // end else
        } // end else
} // end while
print  makespan;
```

ΚΕΦΑΛΑΙΟ 4 : ΑΡΙΘΜΗΤΙΚΑ ΠΕΙΡΑΜΑΤΑ

4.1. Πειραματική Διαδικασία.

Σε αυτήν την ενότητα, θα παρουσιάσουμε τα αριθμητικά αποτελέσματα των online αλγορίθμων που έως τώρα παρουσιάστηκαν θεωρητικά.

Οι αλγόριθμοι των τεσσάρων ερευνητών που μας απασχόλησαν, προγραμματίστηκαν σε γλώσσα Compaq Visual Fortran 90/95, σε λειτουργικό σύστημα Microsoft Windows XP/2000 και παρατίθενται στο Παράρτημα ΙΙ. Πέραν των online αλγορίθμων και των συμπερασμάτων που προέκυψαν από την σύγκριση των αποτελεσμάτων τους, δόθηκε και το αποτέλεσμα του βέλτιστου offline αλγορίθμου (**Optimal algorithm**) για το πρόβλημα και έγινε σύγκριση με τα αντίστοιχα αποτελέσματα των τεσσάρων προαναφερθέντων αλγορίθμων. Η τιμή που πήραμε ως αποτέλεσμα μετά την εκτέλεση των προγραμμάτων, ήταν ο συνολικός χρόνος ολοκλήρωσης των εργασιών (**makespan**), δηλαδή η διάρκεια της προσομοίωσης της παραγωγικής διαδικασίας που μελετήσαμε. Ακόμα, ως επιπλέον κριτήριο σύγκρισης της ποιότητας των αποτελεσμάτων των αλγορίθμων, υπολογίσαμε τον λόγο ανταγωνιστικότητας ρ των αλγορίθμων, που, όπως αναφέρθηκε προηγουμένως, αποτελεί μέτρο σύγκρισης των online αλγορίθμων.

Συνολικά, δημιουργήθηκαν 18 διαφορετικοί συνδυασμοί δεδομένων που προέκυψαν από κατάλληλη επιλογή των παραμέτρων εισόδου του προβλήματος και συγκεκριμένα : της τιμής **a**, που αποτελεί θετικό πραγματικό αριθμό μεγαλύτερο/ίσο της μονάδας (συνεπώς και του **φ**, διότι φ=f(a)) και του πλήθους των εργασιών **N** που καταφθάνουν στις 2 παράλληλες μηχανές. Κάθε πρόγραμμα εκτελέστηκε 10 φορές, με τυχαία επιλογή των χρόνων άφιξης και επεξεργασίας των εργασιών r(t) και p(t) αντίστοιχα, λόγω της γεννήτριας τυχαίων αριθμών που χρησιμοποιήθηκε στο πρόγραμμα Fortran, προκειμένου να υπάρχει τυχαιότητα στα αποτελέσματα και να έχουμε πληρέστερη εικόνα του προβλήματος.

Σε αυτό το σημείο, θα αναφερθούμε στον τρόπο με τον οποίο εισάγαμε στο πρόβλημά μας τα δεδομένα του προβλήματος, δηλαδή τους χρόνους άφιξης r_j, επεξεργασίας p_j, του πλήθους των εργασιών N και των ενδιάμεσων αφίξεων U_j εργασιών, για τους οποίους γίνεται λόγος στην βιβλιογραφία. Για την δημιουργία τυχαίων αριθμών, θεωρήσαμε στο πρόβλημά μας γεννήτρια τυχαίων αριθμών προκειμένου να πάρουμε αρχικές τιμές για τις μεταβλητές r_j, p_j, U_j. Συγκεκριμένα, πήραμε γεννήτρια ομοιόμορφης κατανομής στο διάστημα (0,1), Uniform(0,1). Χωρίς απώλεια της γενικότητας, δεχτήκαμε ότι οι εργασίες αρχίζουν να αποδεσμεύονται την χρονική στιγμή t=0. Οι χρόνοι μεταξύ των αφίξεων διαδοχικών εργασιών θεωρήθηκε ότι ακολουθούν εκθετική κατανομή με μέση τιμή μ=1. Οι ενδιάμεσες αφίξεις υπολογίζονται ως εξής, χρησιμοποιώντας αντίστροφο μετασχηματισμό : $X = F^{-1}(U) = -(\frac{1}{\mu})\ln(1-U)$. Δηλαδή, έτσι υπολογίσαμε το πλήθος

των γεγονότων τα οποία λαμβάνουν χώρα σε ένα ορισμένο χρονικό διάστημα, π.χ (0,t), όταν δύο διαδοχικά γεγονότα (αφίξεις) απέχουν μεταξύ τους τυχαίο χρόνο με εκθέτικη κατάνομη και μέση τιμή μ. Οι χρόνοι άφιξης δόθηκαν από την σχέση : $r_{j+1} = r_j + x_{j+1}$, όπου $1 \le j \le N-1$. Στην παράμετρο N, δόθηκε ως είσοδος ένα πλήθος τιμών, για να πάρουμε μία ολοκληρωμένη εικόνα των αποτελεσμάτων των αλγορίθμων.

Τα αποτελέσματα από τους 18 συνδυασμούς δεδομένων, που αναφέρθηκαν παραπάνω, θα δοθούν σε έναν συγκεντρωτικό πίνακα, στον οποίο ακόλουθούμε τον εξής συμβολισμό : Στις δύο πρώτες στήλες, έχουμε τους αριθμούς N και a, αντίστοιχα. Κάθε γραμμή του πίνακα αντιπροσωπεύει και έναν διαφορετικό συνδυασμό δεδομένων, από τους 18 συνολικά που υπάρχουν. Οι στήλες 3-6, αναφέρονται στους online αλγορίθμους των τεσσάρων ερευνητών και περιέχουν τέσσερις αριθμούς. Ο πρώτος αριθμός της παρένθεσης δείχνει πόσες φορές ο συγκεκριμένος αλγόριθμος ήταν ο καλύτερος από τους υπολοίπους, για το συγκεκριμένο σετ δεδομένων. Ο δεύτερος αριθμός δείχνει πόσες φορές ο αλγόριθμος ήταν δεύτερος καλύτερος, ο τρίτος αριθμός πόσες φορές ήταν τρίτος καλύτερος και ο τέταρτος πόσες φορές ο αλγόριθμος ήταν ο χειρότερος από τους τέσσερις, για το συγκεκριμένο σετ δεδομένων. Εάν δύο ή και περισσότεροι αλγόριθμοι ανήκουν ταυτόχρονα στην ίδια θέση κατάταξης, τότε αναγράφονται και οι δύο στην ίδια θέση.

Οι στήλες 7-10, αναφέρονται στον λόγο ανταγωνιστικότητας των αλγορίθμων και συγκεκριμένα οι αριθμοί σε παρένθεση υποδηλώνουν τα εξής : ο πρώτος αναφέρεται στην ελάχιστη τιμή του λόγου ρ, για τα 10 πειράματα που έγιναν, ο δεύτερος αριθμός αναφέρεται στον μέσο όρο του λόγου ρ και ο τρίτος αναφέρεται στην μέγιστη τιμή του ρ για τα 10 πειράματα που διεξήχθησαν.

Ακολουθεί ο συγκεντρωτικός πίνακας των αποτελεσμάτων :

4.2. Πίνακας Αποτελεσμάτων

N	a	Liu	Nong	Tian-Fu	Tian-Cheng	CR-Liu (min,avg,max)	CR-Nong (min,avg,max)	CR-Tian_Fu (min,avg,max)	CR-Tian_Cheng (min,avg,max)
10	1.001	(7, 3, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1, 1.04)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
20	1.001	(5, 5, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1.01, 1.02)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
50	1.001	(9, 1, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1, 1.01)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
100	1.001	(5, 5, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1, 1)	(1.41, 1.41, 1.41)	(1, 1, 1)	(1, 1, 1)
500	1.001	(5, 5, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1, 1)	(1.41, 1.41, 1.41)	(1, 1, 1)	(1, 1, 1)
1000	1.001	(6, 4, 0, 0)	(0, 0, 0, 10)	(9, 1, 0, 0)	(9, 1, 0, 0)	(1, 1, 1)	(1.41, 1.41, 1.41)	(1, 1, 1)	(1, 1, 1)
10	1.005	(8, 2, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1, 1.03)	(1.41, 1.41, 1.41)	(1, 1, 1)	(1, 1, 1)
20	1.005	(6, 4, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1.01, 1.03)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
50	1.005	(7, 3, 0, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(10, 0, 0, 0)	(1, 1, 1.02)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
100	1.005	(6, 3, 1, 0)	(0, 0, 0, 10)	(8, 2, 0, 0)	(8, 0, 2, 0)	(1, 1, 1.01)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
500	1.005	(3, 5, 2, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(8, 2, 0, 0)	(1, 1.01, 1.02)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
1000	1.005	(6, 4, 0, 0)	(0, 0, 0, 10)	(5, 4, 1, 0)	(3, 2, 5, 0)	(1, 1.02, 1.06)	(1.41, 1.41, 1.41)	(1, 1.02, 1.04)	(1, 1.02, 1.04)
10	1.01	(6, 2, 2, 0)	(0, 0, 0, 10)	(10, 0, 0, 0)	(8, 2, 0, 0)	(1, 1.02, 1.08)	(1.41, 1.41, 1.41)	(1, 1, 1.02)	(1, 1.02, 1.05)
20	1.01	(7, 3, 0, 0)	(0, 0, 0, 10)	(9, 1, 0, 0)	(9, 0, 1, 0)	(1, 1.01, 1.04)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)

Πίνακας 1

N	a	Liu	Nong	Tian-Fu	Tian-Cheng	CR-Liu (min,avg,max)	CR-Nong (min,avg,max)	CR-Tian_Fu (min,avg,max)	CR-Tian_Cheng (min,avg,max)
50	1.01	(5, 5, 0, 0)	(0, 0, 0, 10)	(7, 3, 0, 0)	(7, 3, 0, 0)	(1, 1.01, 1.02)	(1.41, 1.41, 1.41)	(1, 1.01, 1.02)	(1, 1.01, 1.02)
100	1.01	(5, 3, 2, 0)	(0, 0, 0, 10)	(8, 2, 0, 0)	(6, 3, 1, 0)	(1, 1.01, 1.03)	(1.41, 1.41, 1.41)	(1, 1, 1.01)	(1, 1, 1.01)
500	1.01	(0, 1, 9, 0)	(0, 0, 0, 10)	(4, 5, 1, 0)	(6, 4, 0, 0)	(1, 1.08, 1.17)	(1.41, 1.41, 1.41)	(1, 1.03, 1.08)	(1, 1.03, 1.07)
1000	1.01	(10, 0, 0, 0)	(0, 0, 0, 10)	(0, 10, 0, 0)	(0, 10, 0, 0)	(1, 1.23, 1.26)	(1.41, 1.41, 1.41)	(1, 1.32, 1.35)	(1, 1.32, 1.35)

Πίνακας 1 (Συνέχεια)

ΚΕΦΑΛΑΙΟ 5 : ΣΥΜΠΕΡΑΣΜΑΤΑ - ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ

5.1 Συμπεράσματα

Από τα αριθμητικά αποτελέσματα που πήραμε από την εκτέλεση των πειραμάτων, τα οποία δόθηκαν στην προηγούμενη ενότητα, βγάζουμε τα εξής συμπεράσματα :

- Ο αλγόριθμος των Nong et al. (2008) είχε την χειρότερη απόδοση από τους τέσσερις, προσεγγίζοντας την βέλτιστη λύση του offline αλγορίθμου με απόκλιση της τάξης του 41% για όλα τα σετ μετρήσεων (δηλαδή, ο λόγος ρ ήταν ίσος με 1.41). Η αιτία αυτής της απόκλισης, όπως αναφέρθηκε, είναι ότι ο αλγόριθμος χρησιμοποιεί πληροφορία, προκειμένου να καθορίσει την δράση του, μόνο από την τελευταία, μεγαλύτερη σε διάρκεια εργασία J(t) στο σύνολο U(t), αγνοώντας την τρέχουσα παρτίδα $B^*(t)$ τον χρόνο t, πράγμα που δεν συμβαίνει με τους άλλους αλγορίθμους. Άρα η στρατηγική των υπολοίπων αλγορίθμων φαντάζει πιο ορθολογική, παρά το ότι όλοι χρησιμοποιούν στρατηγική καθυστέρησης.

- Τα αποτελέσματα των αλγορίθμων των Liu et al. (2011), Tian et al. (2009a), Tian et al. (2009b), κρίνονται αρκετά ικανοποιητικά στην πλειονότητα των περιπτώσεων, προσεγγίζοντας με αρκετή ακρίβεια τον offline αλγόριθμο. Ακόμα, στις περισσότερες περιπτώσεις, οι αλγόριθμοι των Tian et al. (2009a) και Tian et al. (2009b) δίνουν πανομοιότυπα αποτελέσματα, με τον αλγόριθμο των Liu et al. (2011) να υστερεί σε αυτές τις περιπτώσεις των δύο ανωτέρω. Η δε πλήρης ταύτιση των λόγων ανταγωνιστικότητας τους παρατηρείται για μικρές τιμες του N (10, 20, 50), καθώς και για μικρές τιμές του α (= 1.001).

- Για μεγαλύτερες τιμές των παραμέτρων α (= 1.01) και N (= 500, 1000) αρχίζει να παρατηρείται απόκλιση των τριών παραπάνω αλγορίθμων από την βέλτιστη λύση, με τους λόγους ανταγωνιστικότητας να κυμαίνονται στο εύρος τιμών 1.05 έως 1.1 για τους Tian et al. (2009a), Tian et al. (2009b) και για τους Liu et al. (2011) στο εύρος 1.1 έως 1.2. Για τη μέγιστη τιμή του α στο πείραμα (=1.01), οι αποκλίσεις άγγιξαν τις τιμές 1.25 για τους Liu et al. (2011) και 1.35 για τους αλγορίθμους των Tian et al. (2009a), Tian et al. (2009b), όσον αφορά τον λόγο ρ.

- Φαίνεται ξεκάθαρα από τα παραπάνω, ότι, όσο αυξάνεται η τιμή του α, τόσο αλλοιώνεται η προσέγγιση της βέλτιστης λύσης. Η τιμή του α, όπως αναφέρθηκε, αποτελεί δεδομένο εισόδου, που δίνεται από τον χρήστη και λαμβάνει τιμές >1. Κάθε τιμή του α αντιστοιχεί σε μία τιμή φ του

προβλήματος, που αποτελεί λύση της τριτοβάθμιας εξίσωσης $x^3+(a-1)x^2+(a^2-a-1)x-a^2=0$. Τιμές του α μεγαλύτερες του 1.1, οδηγούν σε σημαντικες αποκλίσεις, γι' αυτό και προτιμήθηκαν τιμές του βήματος (για το α) της τάξης του $10^{-2}\sim10^{-3}$.

5.2 Μελλοντική Έρευνα

Σε αυτήν την διπλωματική εργασία, μελετήσαμε το πρόβλημα του προγραμματισμού εργασιών, που καταφθάνουν με τυχαίο τρόπο σε δύο παράλληλες μηχανές απεριόριστης χωρητικότητας, χρησιμοποιώντας ως εργαλεία τους online αλγορίθμους που πρότειναν σε δημοσιεύσεις τους οι ακόλουθοι ερευνητές : Liu et al. (2011), Nong et al. (2008), Tian et al. (2009a) και Tian et al. (2009b). Έγινε σύγκριση των αλγορίθμων και δόθηκαν τα αποτελέσματά τους, καθώς και τα συμπεράσματα που προέκυψαν από αυτά.
Ορισμένα ερωτήματα που έμειναν αναπάντητα και θα μπορούσαν να αποτελέσουν αντικείμενο περαιτέρω έρευνας είναι τα εξής :

- Θεωρήσαμε μόνο την περίπτωση, κατά την οποία το μέγεθος της παρτίδας είναι απεριόριστο, $b=\infty$. Ως φυσική επέκταση αυτής της εργασίας, έχει ενδιαφέρον η εξέταση της φραγμένης περίπτωσης, $b<\infty$, η οποία απαιτεί νέες μεθόδους ανάλυσης. Για μελλοντικη έρευνα των online αλγορίθμων προγραμματισμού εργασιών, αποτελεί πρόκληση η ανάλυση της μέσης περίπτωσης της απόδοσής τους (average case performance).

- Θεωρήσαμε την περίπτωση, όπου, έχουμε μόνο δύο παράλληλες μηχανές επεξεργασίας, ενώ η γενικότερη περίπτωση για m>2 μηχανές δεν εξετάστηκε. Σύμφωνα με τους Zhang et al. (2003), ο λόγος ανταγωνιστικότητας ρ εξαρτάται από τον αριθμό των μηχανών και είναι $1+\beta_m$, όπου το β_m είναι η ρίζα της εξίσωσης $(1+\beta_m)^{m+1}=2+\beta_m$. Έχει ενδιαφέρον η παρατήρηση ότι ο λόγος ρ μειώνεται όσο ο αριθμός των μηχανών αυξάνεται και τείνει στο 1, καθώς το $m\to\infty$. Προτείνεται η εξέταση του παραπάνω ισχυρισμού των Zhang et al. (2003).

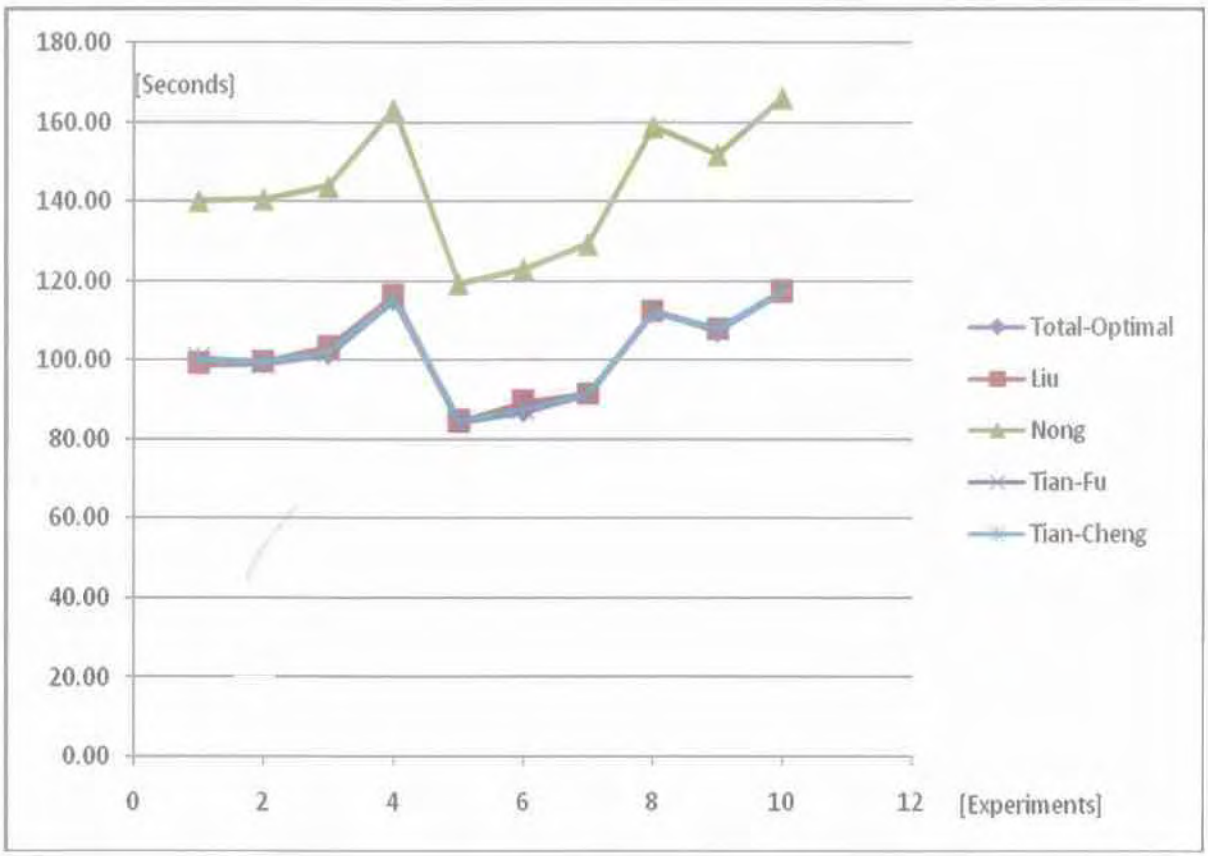
- Μέσα στους τέσσερις online αλγορίθμους που εξετάσαμε, δεν συμπεριλάβαμε την περίπτωση των πυκνών χρονικών αλγορίθμων [DA- algorithms, (dense-algorithms)]. Σύμφωνα με τους Tian et al. (2009b), ένας online αλγόριθμος λέγεται πυκνός, εάν πάντοτε αμέσως αναθέτει τις μέχρι νεοτέρας διαθέσιμες εργασίες σε μία από τις διαθέσιμες (ελεύθερες) μηχανες, εφόσον υπάρχουν το λιγότερο δύο διαθέσιμες μηχανές. Ακόμα, πρότεινε κατώτερο όριο του λόγου ρ για online πυκνούς αλγορίθμους, ίσο με 3/2. Μένει να επιβεβαιωθεί το παραπάνω, ή να βρέθει νέο κατώτερο όριο.

- Μπορούν κατα τον ίδιο τρόπο να εξεταστούν offline αλγόριθμοι, πέραν των online που εξετάστηκαν σε αυτήν την εργασία.

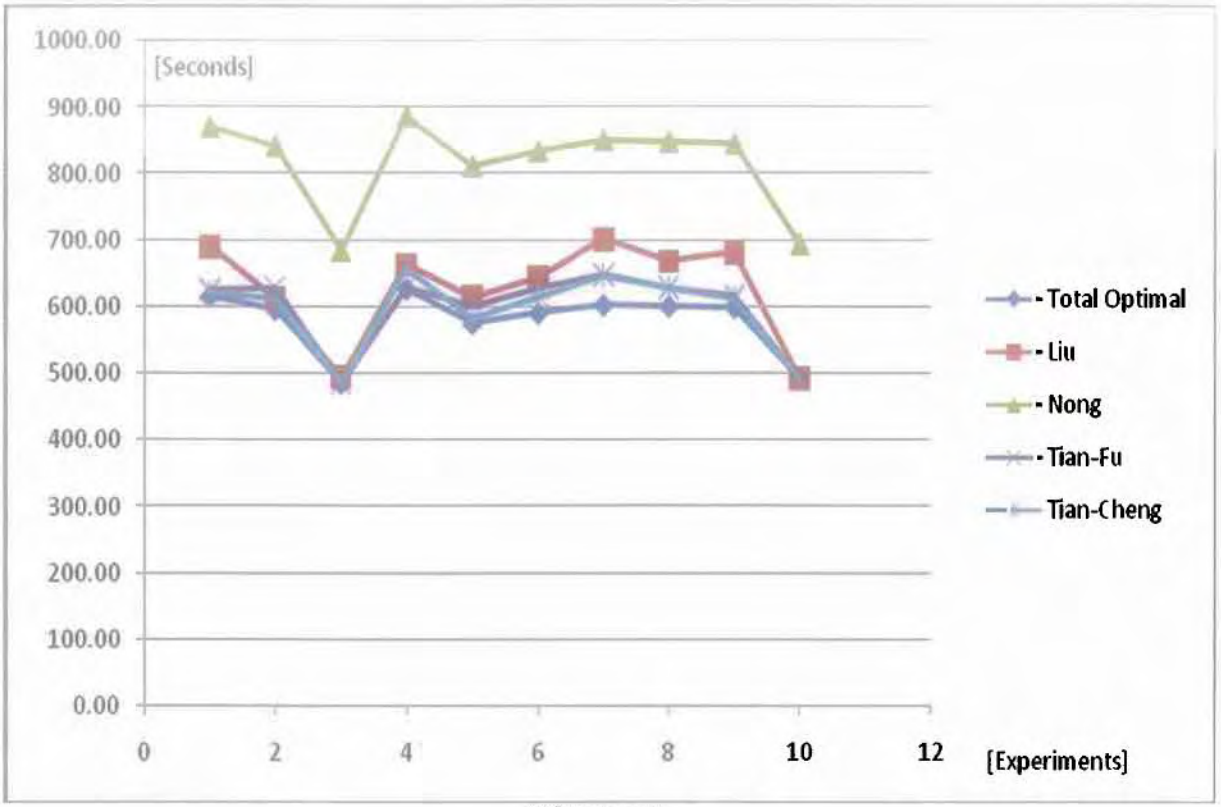
ΠΑΡΑΡΤΗΜΑ Ι : Διαγράμματα Σύγκρισης Αλγορίθμων

Διαγράμματα 1- 3 : Διάρκεια Πειραμάτων – Αριθμός Πειραμάτων

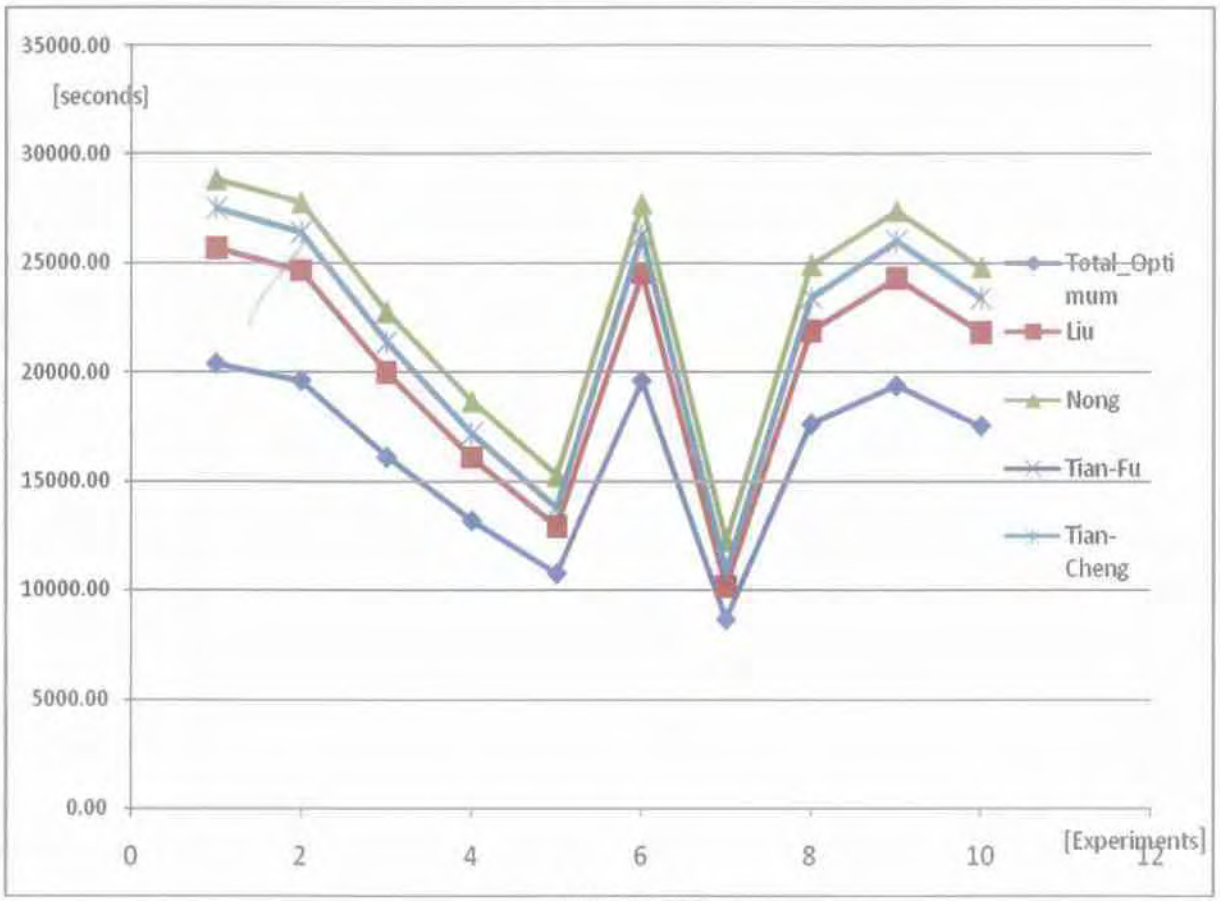
- Διάγραμμα 1: α=1.01, N=100
- Διάγραμμα 2 : α=1.01, N=500
- Διάγραμμα 3 : α=1.01, N=1000



Διάγραμμα 1



Διάγραμμα 2



Διάγραμμα 3

ΠΑΡΑΡΤΗΜΑ ΙΙ : ΚΩΔΙΚΕΣ ΕΠΙΑΥΣΗΣ

Οι κώδικες επίλυσης των τεσσάρων αλγορίθμων παρατίθενται κατά σειρά. Οι τιμές των παραμέτρων α και φ, διαβάζονται από αρχείο (αφού έχουν υπολογιστεί πριν), ενώ ο αριθμός των εργασιών N εισάγεται από τον χρήστη. Πρώτα γίνεται μία επεξήγηση των συμβολισμών που χρησιμοποιούνται στα προγράμματα.

U : το σύνολο των απρογραμμάτιστων εργασιών.

N : το πλήθος των εργασιών που καταφθάνουν.

total : μεταβλητή που χρησιμοποιείται για να απεικονίσουμε το σύνολο των εργασιών, κάθε δεδομένη χρονική στιγμή.

I1, I2 : ακέραιες μεταβλητές, δυαδικού χαρακτήρα, που δηλώνουν αν οι δύο μηχανές είναι απασχολημένες ή όχι.

p(i) : οι χρόνοι επεξεργασίας των εργασιων που καταφθάνουν.

r(i) : οι χρόνοι άφιξης των εργασιών που καταφθάνουν.

X(i) : οι ενδιάμεσες αφίξεις των εργασιών.

W(i) : ενδιάμεση μεταλητή, που χρησιμοποιείται για τον προσδιορισμό των X(i).

makespan : η διάρκεια ολοκλήρωσης των εργασιών του προβλήματος.

S1, S2 : οι χρόνοι έναρξης των δύο μηχανών, αντίστοιχα.

S(i) : οι χρόνοι έναρξης των παρτίδων.

F1, F2 : οι χρόνοι λήξης (της επεξεργασίας) των δύο μηχανών.

t_fin : ο χρόνος λήξης της επεξεργασίας (της παρτίδας).

S_opt : ο χρόνος έναρξης (επεξεργασίας) της J(t), δεδομένου ότι μόνο μία μηχανή είναι απασχολημένη.

S_optimum : ο χρόνος έναρξης (επεξεργασίας) της J(t), δεδομένου ότι μόνο μία μηχανή είναι απασχολημένη.

p_opt : ο χρόνος επεξεργασίας της J(t), δεδομένου ότι μόνο μία μηχανή είναι απασχολημένη.

i : μετρητής.

t : μετρητής χρόνου.

m : μέση τιμή του χρόνου μεταξύ των αφίξεων διαδοχικών εργασιών.

alpha : παράμετρος, η οποία λαμβάνει την τιμή $\sqrt{2}-1$ από την θεωρία.

7.1. Αλγόριθμος Liu et al. (2011)

```
program Algorithm_Liu
implicit none
integer::i,N,l1,l2,total
double precision,allocatable::p(:),r(:),S(:),W(:),X(:)
integer::U  ! synolo aprogrammatistwn ergasiwn ton xrono t
double precision::t,t_fin,makespan !time
double precision,parameter::m=1  ! mesos oros
double precision::a,phi,S1,S2,F1,F2,S_optimum

 open(10,file='data.txt')

        print*,'o ari8mos phi tou provlhmatos einai'
        read(10,*) phi  ;  print *, phi

        print*,'o ari8mos a tou provlhmatos einai'
        read(10,*)  a  ;  print * , a

        print*,'Dwse to plh8os twn ergasiwn pou kataf8anoun'
        read *, N

        allocate( p(N),r(N),S(N),W(N),X(N) )


        !gennhtria tyxaiwn ari8mwn

  !call random_seed   ()
        call random_number (W)
        call random_number (p)
        call random_number (r)

 ! ypologismos twn p(t)
 print*,'p(1)=',p(1)
        do i=1,N-1
   p(i+1)=a*p(i)
  print*,'p(',i+1,')=',p(i+1)

        enddo

 ! ypologismos twn interarrivals X(t)
 print*
 print*,'W=',W
```

```
print*

 do i=1,N
       X(i)=(-1./m)*log(1-W(i))
       print*,'X(',i,')=',X(i)

      enddo

      !ypologismos twn r(t)
      print*
      r(1)=x(1)  ; print*,'r(          1 )=',r(1)

       do i=1,N-1
              r(i+1)=r(i)+x(i+1)
              print*,'r(',i+1,')=',r(i+1)
       enddo


! enar3h alogorithmou

 !arxikopoihsh timwn
      U=0; i=0; total=0; 11=0; 12=0; S1=0; S2=0; F1=0; F2=0 ; makespan=0 ; t=0

do while (total <= N)    !while not all jobs have been scheduled

              if( U==0.and. i<N ) then   !if no unscheduled jobs,go to next arrival
                     i=i+1;t=r(i);U=i
                      if(I1==1.and.F1<=t) then
                            I1=0;S1=0;F1=0
                             if(I2==1.and.F2<=t) then
                              I2=0;S2=0;F2=0
                             endif
                      endif
              endif


       if(I1+I2==2) then    !if both machines busy,go to first machine release
              if(F1<F2) then
                      t=F1 ; S1=0 ; F1=0 ; 11=0
                     else
                      t=F2 ; S2=0 ; F2=0 ; 12=0
              endif
                    do while ((i+1<=N) .and. (r(i+1)<=t))
```

```
                    i=i+1 ; U=i
                  enddo
        endif


            S_optimum = max (S1,S2)

      if( t >= max(phi * S_optimum + (phi - 1)*p(U) , r(U) ))  then    ! if condition
true ,schedule jobs on Ist free machine
            if (I1==0) then
                  I1=I ; S1=t ; F1=S1+p(U) ; total=U ; U=0
                        if(total==N)  then
                          makespan=F1 ; total=total+1
                        endif
            else if ( I2==0 )  then
                  I2=1 ; S2=t ; F2=S2+p(U) ; total=U ; U=0
                        if(total==N) then
                          makespan=F2 ; total=total+1
                        endif
            endif



      else   !if ( t < max(phi * S_optimum + (phi - 1)*p(U) , r(U) ))  then    { if
condition true }

            if (I1==I .and. I2==I) then
                  t_fin = min(F1,F2)
             else if(I1==1 .and. I2==0) then
                  t_fin=F1
             else if(I1==0 .and. I2==I) then
                  t_fin=F2
             else if(I1==0 .and. I2==0) then
                  t_fin=I0E+020
            endif

            if(i+1<=N) then   !if more jobs exist
                  t=min (t_fin,r(i+I),max(phi*S_optimum + (phi-1)*p(U),r(U)))
!jump to next event
                   if(t == r(i+1)) then
                        i=i+1 ; U=i
                       else if(t==t_fin .and. t_fin==F1) then
                        I1=0 ; S1=0 ; F1=0
```

```
                    else if(t==t_fin .and. t_fin==F2) then
                     I2=0 ; S2=0 ; F2=0
                  endif
            else  ! if no more jobs exist
                  t=min(t_fin,max(phi*S_optimum  +  (phi-1)*p(U),r(U)))     !
jump to next event
                  if(t==t_fin .and. t_fin==F1) then
                        I1=0 ; S1=0 ; F1=0
                   else if(t==t_fin .and. t_fin==F2) then
                        I2=0 ; S2=0 ; F2=0
                  endif
            endif
      endif
 enddo

 print*,'makespan=',makespan

 close(10)

end program
```

7.2. Αλγόριθμος Nong et al. (2008)

```
program Nong_Algorithm
implicit none
integer::i,N,I1,I2,total
double precision,allocatable::p(:),r(:),S(:),W(:),X(:)
integer::U  ! synolo aprogrammatistwn ergasiwn ton xrono t
double precision::t,t_fin,makespan !time
double precision,parameter::m=1  ! mesos oros
double precision::S1,S2,F1,F2,a,alpha

 open(10,file='data.txt')

       print*,'Dwse to plh8os twn ergasiwn pou kataf8anoun'
       read *, N

       print*,'o ari8mos a tou provlhmatos einai'
       read(10,*)  a  ;  print * , a

        alpha = sqrt(2.0) - 1.0


       allocate( p(N),r(N),S(N),W(N),X(N) )


       !gennhtria tyxaiwn ari8mwn

 !call random_seed   ()
       call random_number (W)
       call random_number (p)
       call random_number (r)

! ypologismos twn p(t)
print*,'p(1)=',p(1)
       do i=1,N-1
  p(i+1)=a*p(i)
 print*,'p(',i+1,')=',p(i+1)

       enddo

! ypologismos twn interarrivals X(t)
print*
print*,'W=',W
```

```
print*

 do i=1,N
       X(i)=(-1./m)*log(1-W(i))
       print*,'X(',i,')=',X(i)

      enddo

      !ypologismos twn r(t)
      print*
      r(1)=x(1)  ; print*,'r(          1 )=',r(1)

       do i=1,N-1
              r(i+1)=r(i)+x(i+1)
              print*,'r(',i+1,')=',r(i+1)
       enddo


! enar3h alogorithmou

 !arxikopoihsh timwn
      U=0; i=0; total=0; I1=0; I2=0; S1=0; S2=0; F1=0; F2=0 ; makespan=0 ; t=0

do while (total <= N)    !while not all jobs have been scheduled

              if( U==0.and. i<N ) then   !if no unscheduled jobs,go to next arrival
                    i=i+1;t=r(i);U=i
                     if(I1==1.and.F1<=t) then
                           I1=0;S1=0;F1=0
                            if(I2==1.and.F2<=t) then
                             I2=0;S2=0;F2=0
                            endif
                     endif
              endif


       if(I1+I2==2) then    !if both machines busy,go to first machine release
              if(F1<F2) then
                     t=F1 ; S1=0 ; F1=0 ; I1=0
                     else
                     t=F2 ; S2=0 ; F2=0 ; I2=0
              endif
                    do while ((i+1<=N) .and. (r(i+1)<=t))
```

```
                    i=i+1 ; U=i
                  enddo
        endif


      if( t >= (1+alpha)*r(U) + alpha*p(U) )  then    ! if condition true ,schedule
jobs on Ist free machine
            if (II==0) then
                  I1=1 ; S1=t ; F1=S1+p(U) ; total=U ; U=0
                        if(total==N)  then
                          makespan=F1 ; total=total+1
                        endif
            else if ( I2==0 )  then
                  I2=1 ; S2=t ; F2=S2+p(U) ; total=U ; U=0
                        if(total==N) then
                          makespan=F2 ; total=total+1
                        endif
            endif



      else  !if ( t < (1+alpha)*r(U) + alpha*p(U) )  then   { if condition true }

            if (II==1 .and. I2==1) then
                  t_fin = min(F1,F2)
             else if(I1==1 .and. I2==0) then
                  t_fin=F1
             else if(I1==0 .and. I2==I) then
                  t_fin=F2
             else if(I1==0 .and. I2==0) then
                  t_fin=I0E+020
            endif

            if(i+1<=N) then   !if more jobs exist
                  t=min  (t_fin,r(i+1),(1+alpha)*r(U)  +  alpha*p(U))  !jump  to
next event
                   if(t == r(i+1)) then
                        i=i+1 ; U=i
                        else if(t==t_fin .and. t_fin==F1) then
                        II=0 ; S1=0 ; F1=0
                        else if(t==t_fin .and. t_fin==F2) then
                        I2=0 ; S2=0 ; F2=0
```

```
                endif
            else  ! if no more jobs exist
                t=min(t_fin,(1+alpha)*r(U) + alpha*p(U))   ! jump to next
event
                if(t==t_fin .and. t_fin==F1) then
                    I1=0 ; S1=0 ; F1=0
                 else if(t==t_fin .and. t_fin==F2) then
                    I2=0 ; S2=0 ; F2=0
                endif
            endif
        endif
 enddo

 print*,'makespan=',makespan


 close(10)


end program
```

7.3. Αλγόριθμος Tian et al. (2009a)

```
 program Algorithm_Tian_Fu
 implicit none
integer::i,N,l1,l2,total
double precision,allocatable::p(:),r(:),S(:),W(:),X(:)
integer::U  ! synolo aprogrammatistwn ergasiwn ton xrono t
double precision::t,t_fin,makespan !time
double precision,parameter::m=1  ! mesos oros
double precision::S1,S2,F1,F2,a,alpha,S_opt,P_opt

 open(10,file= data.txt')

        print*,'Dwse to plh8os twn ergasiwn pou kataf8anoun'
        read *, N

        print*,'o ari8mos a tou provlhmatos einai'
        read(10,*)  a  ;  print * , a

         alpha = sqrt(2.0) - 1.0


        allocate( p(N),r(N),S(N),W(N),X(N) )


        !gennhtria tyxaiwn ari8mwn

  !call random_seed   ()
        call random_number (W)
        call random_number (p)
        call random_number (r)

 ! ypologismos twn p(t)
 print*,'p(1)=',p(1)
        do i=1,N-1
   p(i+1)=a*p(i)
  print*,'p(',i+1,')=',p(i+1)

        enddo

 ! ypologismos twn interarrivals X(t)
 print*
 print*,'W=',W
```

```
print*

 do i=1,N
       X(i)=(-1./m)*log(1-W(i))
       print*,'X(',i,')=',X(i)

      enddo

      !ypologismos twn r(t)
      print*
      r(1)=x(1)  ; print*,'r(          1 )=',r(1)

       do i=1,N-1
              r(i+1)=r(i)+x(i+1)
              print*,'r(',i+1,')=',r(i+1)
       enddo


 ! enar3h alogorithmou

  !arxikopoihsh timwn
      U=0; i=0; total=0; I1=0; I2=0; S1=0; S2=0; F1=0; F2=0 ; makespan=0 ; t=0 ;
S_opt=0 ; P_opt=0

 do while (total <= N)    !while not all jobs have been scheduled

                 if( U==0.and. i<N ) then   !if no unscheduled jobs,go to next arrival
                        i=i+1;t=r(i);U=i
                         if(I1==1.and.F1<=t) then
                               I1=0;S1=0;F1=0
                                if(I2==1.and.F2<=t) then
                                 I2=0;S2=0;F2=0
                                endif
                         endif
                 endif


        if(I1+I2==2) then    !if both machines busy,go to first machine release
                 if(F1<F2) then
                         t=F1 ; S1=0 ; F1=0 ; 11=0
                        else
                         t=F2 ; S2=0 : F2=0 ; 12=0
                 endif
```

```
                    do while ((i+1<=N) .and. (r(i+1)<=t))
                     i=i+1 ; U=i
                    enddo
        endif


        if(I1+I2==0) then
          S_opt=0 ; P_opt=0
      else if(I1==1) then
          S_opt=S1 ; P_opt=F1-S1
        else if(I2==1) then
              S_opt=S2 ; P_opt=F2-S2
        endif



       if( t >= max( alpha*p(U),S_opt + alpha*P_opt ) )  then    ! if condition true
,schedule jobs on 1st free machine
              if (I1==0) then
                    I1=1 ; S1=t ; F1=S1+p(U) ; total=U ; U=0
                          if(total==N)  then
                            makespan=F1 ; total=total+1
                          endif
              else if ( I2==0 )  then
                    I2=1 ; S2=t ; F2=S2+p(U) ; total=U ; U=0
                          if(total==N) then
                            makespan=F2 ; total=total+1
                          endif
              endif



       else  !if ( t < max( alpha*p(U),S_opt + alpha*P_opt )  then   { if condition not
true }

              if (I1==1 .and. I2==1) then
                    t_fin = min(F1,F2)
               else if(I1==1 .and. I2==0) then
                    t_fin=F1
               else if(I1==0 .and. I2==1) then
                    t_fin=F2
               else if(I1==0 .and. I2==0) then
                    t_fin=10E+020
```

```
                endif

                if(i+1<=N) then   !if more jobs exist
                        t=min (t_fin , r(i+1),max( alpha*p(U) , S_opt + alpha*P_opt
)) !jump to next event
                        if(t == r(i+1)) then
                                i=i+1 ; U=i
                                else if(t==t_fin .and. t_fin==F1) then
                                I1=0 ; S1=0 ; F1=0
                                else if(t==t_fin .and. t_fin==F2) then
                                I2=0 ; S2=0 ; F2=0
                        endif
                else  ! if no more jobs exist
                        t=min( t_fin , max( alpha*p(U) , S_opt + alpha*P_opt ) )  !
jump to next event
                        if(t==t_fin .and. t_fin==F1) then
                                I1=0 ; S1=0 ; F1=0
                         else if(t==t_fin .and. t_fin==F2) then
                                I2=0 ; S2=0 ; F2=0
                        endif
                endif
        endif
 enddo

 print*,'makespan=',makespan


 close(10)


end program
```

7.4. Αλγόριθμος Tian_et al. (2009b)

```
program Algorithm_Tian_Cheng
implicit none
integer::i,N,11,I2,total
double precision,allocatable::p(:),r(:),S(:),W(:),X(:)
integer::U  ! synolo aprogrammatistwn ergasiwn ton xrono t
double precision::t,t_fin,makespan !time
double precision,parameter::m=1  ! mesos oros
double precision::S1,S2,F1,F2,a,alpha,S_opt

 open(10,file='data.txt')

        print*,'Dwse to plh8os twn ergasiwn pou kataf8anoun'
        read *, N

        print*,'o ari8mos a tou provlhmatos einai'
        read(10,*)  a  ;  print * , a

         alpha = sqrt(2.0) - 1.0


        allocate( p(N),r(N),S(N),W(N),X(N) )


        !gennhtria tyxaiwn ari8mwn

  !call random_seed   ()
        call random_number (W)
        call random_number (p)
        call random_number (r)

 ! ypologismos twn p(t)
 print*,'p(1)=',p(1)
        do i=1,N-1
   p(i+1)=a*p(i)
  print*,'p(',i+1,')=',p(i+1)

        enddo

 ! ypologismos twn interarrivals X(t)
 print*
 print*,'W=',W
```

```
print*

do i=1,N
      X(i)=(-1./m)*log(1-W(i))
      print*,'X(',i,')=',X(i)

      enddo

      !ypologismos twn r(t)
      print*
      r(1)=x(1)  ; print*,'r(          1 )=',r(1)

       do i=1,N-1
            r(i+1)=r(i)+x(i+1)
            print*,'r(',i+1,')=',r(i+1)
       enddo


 ! enar3h alogorithmou

  !arxikopoihsh timwn
      U=0; i=0; total=0; I1=0; I2=0; S1=0; S2=0; F1=0; F2=0 ; makespan=0 ; t=0 ;
t_fin=0; S_opt=0
 do while (total <= N)    !while not all jobs have been scheduled

            if( U==0.and. i<N ) then   !if no unscheduled jobs,go to next arrival
                  i=i+1;t=r(i);U=i
                  if(I1==1.and.F1<=t) then
                        I1=0;S1=0;F1=0
                        if(I2==1.and.F2<=t) then
                         I2=0;S2=0;F2=0
                        endif
                  endif
            endif


       if(I1+I2==2) then    !if both machines busy,go to first machine release
            if(F1<F2) then
                  t=F1 ; S1=0 ; F1=0 ; I1=0
                  else
                  t=F2 ; S2=0 ; F2=0 ; I2=0
            endif
                  do while ((i+1<=N) .and. (r(i+1)<=t))
```

```
                    i=i+1 ; U=i
                  enddo
        endif


        if(I1+I2==0) then
          S_opt=0
      else if(I1==1) then
          S_opt=S1
        else if(I2==1) then
            S_opt=S2
        endif



      if( t >= S_opt + alpha*p(U) )  then    ! if condition true ,schedule jobs on 1st
free machine
            if (I1==0) then
                I1=I ; S1=t ; F1=S1+p(U) ; total=U ; U=0
                  if(total==N)  then
                    makespan=F1 ; total=total+1
                  endif
            else if ( I2==0 )  then
                I2=I ; S2=t ; F2=S2+p(U) ; total=U ; U=0
                  if(total==N) then
                    makespan=F2 ; total=total+1
                  endif
            endif



      else  !if ( t < max( alpha*p(U),S_opt + alpha*P_opt )  then   { if condition not
true }

            if (I1==1 .and. I2==1) then
                t_fin = min(F1,F2)
             else if(I1==1 .and. I2==0) then
                t_fin=F1
             else if(I1==0 .and. I2==I) then
                t_fin=F2
             else if(I1==0 .and. I2==0) then
                t_fin=I0E+020
            endif
```

```
                if(i+1<=N) then   !if more jobs exist
                        t=min (t_fin , r(i+1), S_opt + alpha*p(U) ) !jump to next event
                        if (t == r(i+1)) then
                                i=i+1 ; U=i
                                else if(t==t_fin .and. t_fin==F1) then
                                l1=0 ; S1=0 ; F1=0
                                else if(t==t_fin .and. t_fin==F2) then
                                l2=0 ; S2=0 ; F2=0
                        endif
                else  ! if no more jobs exist
                        t=min( t_fin ,  S_opt + alpha*p(U) )  ! jump to next event
                        if(t==t_fin .and. t_fin==F1) then
                                l1=0 ; S1=0 ; F1=0
                         else if(t==t_fin .and. t_fin==F2) then
                                l2=0 ; S2=0 ; F2=0
                        endif
                endif
        endif
 enddo

 print*,'makespan=',makespan


 close(10)


end program
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M.Y., Pots, C.N., Tautenhahn, T. and Van de Velde, S. (1998) Scheduling a batching machine. Journal of Scheduling, 1, 31-54.

[2] B. Chen, A.P.A. Vestjens, Scheduling on identical machines : How good is LPT in an on-line setting ? Operations Research Letters 21 (1997) 165-169.

[3] B. Chen, X. Deng, W. Zang, On-line scheduling a batch processing machine to minimize total weighted job completion time, Lecture notes in Computer Science 2223 (2001) 380-389.

[4] T.C.E. Cheng et al., C.T. Ng, J.J. Yuan, Z.H. Liu, Single machine parallel batch scheduling subject to precedence constraints, Naval Research Logistics 51 (2004) 949-958.

[5] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, Optimization and approximation in deterministic sequencing and scheduling :A survey, Annals Discrete Mathematics 5 (1979) 287-326.

[6] C.Y. Lee, R. Uzsoy, L.A. Martin Vega, Efficient algorithms for scheduling semiconductor burn- in operations, Operations Research 40 (1992) 764-775.

[7] S.G. Li, G.J. Li, X.Q. Qi, Minimizing total weighted completion time on identical parallel batch machines, International Journal of Foundations of Computer Science 176 (2006) 1441-1454.

[8] Ming Liu, Chenbin Chu, Yinfeng Xu, Feifeng Zheng (2011), "An optimal semi-online algorithm for scheduling on two parallel batch processing machines", unpublished manuscript.

[9] J. Noga, S.S. Seiden, An optimal online algorithm for scheduling two machines with release times, Theoretical Computer Science 268 (2001) 133 - 143.

[10] Q.Q Nong, T.C.E Cheng, C.T. Ng, " An improved on- line algorithm for scheduling on two unrestrictive parallel batch processing machines", Operations Research Letters, 36, (2008) 584-588.

[11] Ji Tian, Ruyan Fu, Jinjiang Yuan (2009a), "A best online algorithm, for scheduling on two parallel batch machines", Theoretical Computer Science, 410 (2009) 2291-2294.

[12] Ji Tian, T.C.E Cheng, C.T Ng, Jinjiang Yuan (2009b), "Online scheduling on unbounded parallel batch machines to minimize the makespan", Information Processing Letters, 109 (2009) 1211 - 1215.

[13] J.J. Yuan, Z.H. Liu, C.T. Ng, T.C.E. Cheng, The unbounded single machine parallel batch scheduling problem with family jobs and release dates to minimize makespan, Theoretical Computer Science 320 (2004) 199-212.

[14] G. Zhang, X. Cai, C.K. Wong, On-line algorithms for minimizing makespan on batch processing machines, Naval Research Logistics 48 (2001) 241-258.

[15] Guochuan Zhang, Xiaoqiang Cai, C.K Wong (2003) "Optimal on – line algorithms for scheduling on parallel batch processing machines", IIE Transactions, 35, 175-181.