Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Αλγόριθμοι Δρομολόγησης με Εκτίμηση της Χρονικής Διάρκειας των Εργασιών σε Πραγματικό Χρόνο

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΠΥΡΙΔΩΝ ΚΩΝΣΤΑΝΤΙΝΟΣ ΔΡΑΓΑΖΗΣ

Επιβλέπων : Δημήτριος Φωτάκης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Αλγόριθμοι Δρομολόγησης με Εκτίμηση της Χρονικής Διάρκειας των Εργασιών σε Πραγματικό Χρόνο

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΠΥΡΙΔΩΝ ΚΩΝΣΤΑΝΤΙΝΟΣ ΔΡΑΓΑΖΗΣ

Επιβλέπων : Δημήτριος Φωτάκης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Σεπτεμβρίου 2022.

Δημήτριος Φωτάκης Αναπληρωτής Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.	Ευριπίδης Μπάμπης Καθηγητής Sorbonne Université	Αριστείδης Παγουρτζής Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2022

Σπυρίδων Κωνσταντίνος Δραγάζης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η κάτωθι διπλωματική εργασία μελετά το πρόβλημα Χρονοδρομολόγησης Εργασιών Υπό Αβεβαιότητα σε μία μηχανή. Σκοπός των αλγορίθμων που αναπτύχθηκαν είναι η ελαχιστοποίηση μίας αντικειμενικής συνάρτησης η οποία είναι το άθροισμα των χρόνων αποπεράτωσης κάθε εργασίας. Πιο συγκεκριμένα, η κλάση των προβλημάτων Χρονοδρομολόγησης Εργασιών είναι από τα πλέον πιο μελετημένα προβλήματα στον τομέα των Αλγορίθμων και της Επιχειρησιακής Έρευνας απαριθμώντας ποικίλες εφαρμογές, με γνωστότερη τον προγραμματισμό των χρονοδρομολογητών στα λειτουργικά συστήματα των υπολογιστών. Οι δύο μεγάλες κατηγορίες προβλημάτων Χρονοδρομολόγησης διαφέρουν στην εκ των προτέρων γνώση του ακριβούς μεγέθους των εργασιών. Σημειώνουμε ότι και οι δύο κατηγορίες έχουν μελετηθεί εκτενώς στην βιβλιογραφία. Συνήθως, στην πλειοψηφία των αλγορίθμων που μελετούνται καθώς και στις πραγματικές εφαρμογές ο χρονοδρομολογητής δεν γνωρίζει το ακριβές μέγεθος κάθε εργασίας. Σε σημερινές εφαρμογές, όμως, έχουν αναπτυχθεί αλγόριθμοι συμπίεσης εργασιών ή βελτιστοποίησης του εκτελέσιμου κώδικα που φέρουν, καθιστώντας πλέον εφικτή την δυνατότητα να δώσουν πληροφορία για το ακριβές μέγεθος της κάθε εργασίας. Ακολουθώντας αυτό το μοντέλο μελετήθηκαν αρχικά οι κυριότεροι αλγόριθμοι χρονοδρομολόγησης για αγνώστου μεγέθους εργασίες. Κατόπιν, ορμώμενοι από ιδέες από τον τομέα των Αλγορίθμων καθοδηγούμενων από την Μηχανική Μάθηση συνδυάσαμε τους κλασικούς αλγορίθμους με νέους που μαθαίνουν το πραγματικό μέγεθος κάθε εργασίας κατά την διάρκεια της εκτέλεσης. Συνολικά, ο αλγόριθμος που κατασκευάστηκε είναι εύρωστος και συνεπής. Δηλαδή, διατηρεί καλή απόδοση όταν έχουμε μεγάλα σφάλματα καθώς και πλησιάζει τις επιδόσεις του βέλτιστου αλγορίθμου υπό την προϋπόθεση ότι τα σφάλματα είναι αρκούντως μικρά. Να σημειώσουμε ότι ο βέλτιστος αλγόριθμος έχει πρότερη γνώση του ακριβούς μεγέθους κάθε εργασίας.

Λέξεις κλειδιά

Χρονοδρομολόγηση Αγνώστου Μεγέθους Εργασιών, Αλγόριθμοι Καθοδηγούμενοι από Μηχανική Μάθηση, Ανάλυση Δυσκολότερου Στιγμιότυπου, Round Robin

Abstract

This diploma thesis revisits the problem of Non Clairvoyant Scheduling Under Explorable Uncertainty. The problem we are interested in is to schedule a set of jobs either on a single or on multiple machines. Our goal is to minimize the sum of completion times. In general, Scheduling is one of the most well-studied problems in Computer Science and Operations Research literature with a huge variety of real world applications. In the majority of the scheduling models, the assumption that all characteristics of a job are known in advance, is followed. More analytically, the two main categories in scheduling are divided based on the a priori or not knowledge of the exact characteristics of the jobs. Clairvoyant scheduling studies the case where we are, in advance, aware of the exact features of a job. On the contrary, we talk about non clairvoyant scheduling when there is absence of the characteristics of the jobs until their completion. In practice, in majority of real world applications the exact knowledge of jobs' features is not possible. In this work, we study the algorithmic aspects that lie in the field of Explorable Uncertainty, which belongs in the intersection of Clairvoyance and Non-Clairvoyance. We start by exploring the well-known *Round Robin* algorithm for the non-clairvoyant case. Afterwards, we move to a new direction that is Scheduling Under Uncertainty. In this framework, we initially consider an upper bound of the processing characteristics of the jobs and we can dynamically acquire the exact features by paying an extra cost. Finally, we focus on the area of Learning Augmented Algorithms where the goal is to design algorithms that use predictions from Machine Learning Models. In our approach, instead of predictions we decided to use the notion of testing and learn specific information about the jobs in parallel with their execution.

Key words

Clairvoyant Scheduling, Non-Clairvoyant Scheduling, Explorable Uncertainty, Learning Augmented Algorithms, Competitive Analysis, Round Robin Algorithm

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο της υποτροφίας του ιδρύματος Λεμός το οποίο ευχαριστώ ιδιαιτέρως για την ευκαιρία που μου έδωσε να δουλέψω σε ένα κορυφαίο πανεπιστήμιο όπως αυτό της Σορβόνης και να συνεργαστώ με σπουδαίους επιστήμονες στον τομέα της Θεωρητικής Πληροφορικής. Πιο συγκεκριμένα ευχαριστώ θερμά τους καθηγητές μου, τον κύριο Δημήτρη Φωτάκη και τον κύριο Ευριπίδη Μπάμπη για την πολύτιμη καθοδήγηση, τον χρόνο που αφιέρωσαν σε εμένα και τις πολύτιμες γνώσεις που μου μετέδωσαν. Επιπλέον, ιδιαίτερες ευχαριστίες οφείλω στον Κωνσταντίνο Δογέα για τις αμέτρητες ώρες που αφιέρωσε σε εμένα και τις πολύτιμες συμβουλές του τόσο σε ερευνητικό αλλά και σε προσωπικό επίπεδο. Σας ευχαριστώ με όλη μου την ψυχή γιατί με βοηθήσατε να ανακαλύψω τι πραγματικά μου ταιριάζει. Επιπλέον, θα ήθελα να ευχαριστήσω την οικογένεια μου για την αμέτρητη στήριξη που μου παρέχει στο να κάνω πράξη αυτό που πραγματικά αγαπώ. Τέλος, δεν θα μπορούσα να μην ευχαριστήσω τους φίλους μου που μαζί κάναμε τα φοιτητικά μας χρόνια να μοιάζουν αξέχαστα.

Σπυρίδων Κωνσταντίνος Δραγάζης,

Αθήνα, 20η Σεπτεμβρίου 2022

Περιεχόμενα

Εκτεταμένη Ελληνική Περίληψη

Το βασικό σκέλος της παρούσας διπλωματικής εργασίας έχει αποδοθεί στην αγγλική γλώσσα. Σε αυτό το κομμάτι, συνοψίζουμε το περιεχόμενό της, δίνοντας έμφαση στους βασικούς ορισμούς, τις μεθοδολογίες και τα θεωρήματα, αλλά χωρίς τις μαθηματικές αποδείξεις. Η δομή της ενότητας αυτής είναι σε ένα προς ένα αντιστοίχηση με το (αγγλικό) περιεχόμενο της διπλωματικής εργασίας. Να σημειωθεί ότι διπλά σε κάθε μετάφραση ενός αγγλικού όρου στα ελληνικά θα αναγράφεται μέσα σε παρένθεση ο αντίστοιχος όρος στα αγγλικά.

0.1 Χρονοδρομολόγηση εργασιών χωρίς πρότερη γνώση του χρόνου εκτέλεσης τους (Non-Clairvoyant Scheduling)

Στο πρώτο σκέλος της διπλωματικής εργασίας έγινε μία εκτενής μελέτη στην βιβλιογραφία της Χρονοδρομολόγησης Εργασιών όπου μελετήθηκαν οι διάφορες παραλλαγές των προβλημάτων, οι διαφορετικές αντικειμενικές συναρτήσεις προς βελτιστοποίηση και τα διάφορα είδη περιορισμών που μπορούν να προκύψουν σε μία εφικτή λύση. Περισσότερη έμφαση δόθηκε στον κλασσικό Αλγόριθμο Χρονοπρογραμματισμού εκ περιτροπής (Round Robin) στο πλαίσιο της Χρονοδρομολόγησης εργασιών χωρίς πρότερη γνώση του χρόνου αποπεράτωσής τους. Η αξία αυτού του αλγορίθμου αντλείται από την ευρεία χρησιμότητά του στους χρονοδρομολογητές εργασιών που κάθε λειτουργικό σύστημα υπολογιστών χρησιμοποιεί. Η αιτία για την χρήση αυτού του αλγορίθμου είναι το γεγονός ότι σχεδιάστηκε χωρίς να λαμβάνει υπόψη τα μεγέθη των εργασιών προς δρομολόγηση. Στην πράξη ούτε τα λειτουργικά συστήματα μπορούν να γνωρίζουν ακριβώς τον χρόνο που χρειάζεται για την αποπεράτωση της κάθε εργασίας. Πιο αναλυτικά, σε αυτό το μοντέλο δεν γνωρίζουμε το ακριβές μέγεθος κάθε εργασίας προς δρομολόγηση παρά μόνο την χρονική στιγμή της άφιξής της (release time). Ο σκοπός του αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής είναι να δίνει μία ίση και δίκαιη πρόσβαση στους πόρους της Κεντρικής Μονάδας Επεξεργασίας σε κάθε εργασία. Για την επίτευξη των παραπάνω, θεωρούμε ότι όλες οι εργασίες χωρίζονται σε ίσου μεγέθους μικρά κομμάτια τα οποία εκτελούνται κυκλικά. Κάθε εργασία που ολοκληρώνεται αφαιρείται από την ουρά προτεραιότητας. Από μία πιο μαθηματική σκοπιά η αναλλοίωτη του παραπάνω αλγορίθμου είναι ότι σε κάθε χρονική στιγμή όλες οι ενεργές εργασίες έχουν λάβει ίσο χρόνο επεξεργασίας. Για να θεωρηθεί ορθή μαθηματικά αυτή η υπόθεση θεωρούμε ότι χωρίζουμε τις εργασίες σε αρκούντως μικρά κομμάτια ώστε η διαφορά του χρόνου επεξεργασίας που έχουν λάβει σε κάθε χρονική στιγμή να τείνει στο μηδέν. Τέλος, σημαντική υπόθεση θεωρείται ότι δεν υπάρχει χρονικό κόστος για την παύση μίας εργασίας και την συνέχισή της αργότερα.

Ορμώμενοι από την κλασσική δουλειά των Phillips et al. [1993] προσπαθήσαμε να δώσουμε

μία πιο λεπτομερή απόδειξη στην ανάλυση της απόδοσης του αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής για μία ή για παράλληλες μηχανές και να χρησιμοποιήσουμε αυτές τις τεχνικές βελτιστοποίησης και σε άλλα προβλήματα. Η κλάση των αλγορίθμων που επιλύουν το συγκεκριμένο πρόβλημα ως μετρική απόδοσης έχουν το άθροισμα των χρόνων εκτέλεσης κάθε εργασίας (sum of completion times) αποσκοπώντας στην ελαχιστοποίηση της παραπάνω μετρικής. Η ανάλυση που ακολουθείται ονομάζεται ανάλυση ανταγωνισμού ή χειρότερης περίπτωσης (competitive analysis),συμβολίζεται ως $R_A(n, p)$ όπου n ο πληθάριθμος του συνόλου $\mathcal{J}$ των εργασιών και ο p ο πληθάριθμος του συνόλου των μηχανών όπου ανατίθενται οι εργασίες. Μαθηματικά ορίζεται ως εξής :

Definition 0.1.1.

$$R_A(p, n) = sup_{J:|J|=n} \frac{C_A(p, J)}{C_{OPT}(p, J)}$$

Ποιοτικά η παραπάνω μετρική συγκρίνει έναν αλγόριθμο, εν προκειμένω τον Αλγόριθμο Χρονοπρογραμματισμού εκ περιτροπής, με τον καλύτερο παντογνώστη αλγόριθμο. Στην περίπτωση που μελετάμε, ο καλύτερος παντογνώστης αλγόριθμος έχει γνώση για το μέγεθος των εργασιών πέρα από την χρονική τιμή άφιξης τους. Ο εν λόγω αλγόριθμος ονομάζεται Αλγόριθμος Συντομότερης Εργασίας (SPT-Shortest Processing Time) και δρομολογεί τις εργασίες κατά μη φθίνουσα σειρά μεγέθους. Η απόδειξη για το γεγονός ότι ένας άπληστος αλγόριθμος αποτελεί τον καλύτερο δυνατό αντλείται από ένα λήμμα ανταλλαγής. Αν αλλάξουμε την σειρά δύο οποιωνδήποτε εργασιών ενώ αυτές είναι προγραμματισμένες να δρομολογηθούν σε αύξουσα σειρά μεγέθους τότε θα σχηματίσουμε μία αντικειμενική συνάρτηση με μεγαλύτερη τιμή από την προηγούμενη. Αφότου ορίσαμε τον βέλτιστο αλγόριθμο, θα παραδώσουμε και τον ορισμό του Αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής.

Definition 0.1.2. Αλγόριθμος Χρονοπρογραμματισμού εκ περιτροπής ονομάζεται κάθε αλγόριθμο στον οποίο σε κάθε χρονική στιγμή όλες οι εργασίες έχουν λάβει ίσο χρόνο επεξεργασίας (processing time). Αν υπάρχουν πολλές χρονικές στιγμές άφιξης εργασιών, τότε ανάμεσα σε δύο διαδοχικές όλες οι εργασίες λαμβάνουν ίσο χρόνο επεξεργασίας.

Έπειτα από τον ορισμό του παραπάνω αλγορίθμου θα παραδώσουμε κάτω φράγματα (Lower Bounds) στην απόδοση κάθε στατικού και ντετερμινιστικού αλγορίθμου στην περίπτωση όπου έχουμε μία μηχανή και θα σκιαγραφήσουμε την τεχνική απόδειξης κάτω φραγμάτων σε προβλήματα Χρονοδρομολόγησης. Να σημειώσουμε ότι εξετάζουμε την στατική (static) εκδοχή του αλγορίθμου όπου γνωρίζουμε όλες τις εργασίες εξ αρχής καθώς και ότι επιτρέπεται να διακόψουμε κάποιες εργασίες χωρίς έξτρα κόστος (preemptive case). Η τεχνική λήψης κάτω φραγμάτων κατέχει ιδιαίτερη σημασία στον σχεδιασμό του αλγορίθμου προγραμματισμού εκ περιτροπής καθώς από εκεί αντλείται η ιδέα της "δίκαιης" μοιρασιάς των διαθέσιμων πόρων σε κάθε εργασία.

Theorem 0.1.3. *Κάθε στατικός(static), ντετερμινιστικός (deterministic) αλγόριθμος χρονοδρομολόγησης έχει ελάχιστο λόγο προσέγγισης (approximation ratio) προς τον Αλγόριθμο Συντομότερης Εργασίας ίσο με* $2(1 - \frac{1}{n+1})$.

Η απόδειξη τέτοιων αποτελεσμάτων σε προβλήματα Χρονοδρομολόγησης ποιοτικά είναι η εξής. Τρέχουμε όποιον αλγόριθμο επιθυμούμε έως την χρονική στιγμή $t = 1$. Κατόπιν, θεωρούμε

ότι το μέγεθος κάθε διεργασίας είναι ίσο με το μέγεθος του χρόνου επεξεργασίας που έλαβε η συγκεκριμένη εργασία προστιθέμενη με μία αμελητέα θετική ποσότητα που τείνει στο μηδέν. Αυτή η αναλλοίωτη μας εξασφαλίζει ότι καμία εργασία δεν έχει τελειώσει και καθιστά εύκολη την λήψη ενός κάτω φράγματος στην αντικειμενική συνάρτηση του αλγορίθμου και κατόπιν βελτιστοποίησης του.

Κατόπιν, θα παραθέσουμε το βασικό θεώρημα για την απόδοση του λόγου προσέγγισης του Αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής και θα σκιαγραφήσουμε την απόδειξη του. Πιο αναλυτικά, για την απόδειξη, οφείλουμε να κατασκευάσουμε την αντικειμενική συνάρτηση (objective function) για τους δύο αλγορίθμους και έπειτα να ελαχιστοποιήσουμε μαθηματικά τον λόγο τους.

Theorem 0.1.4. *Ο Αλγόριθμος Χρονοπρογραμματισμού εκ περιτροπής έχει λόγο προσέγγισης (approximation ratio) προς τον Αλγόριθμο Συντομότερης Εργασίας ίσο με* $2(1 - \frac{1}{n+1})$.

Πλέον, θα εξετάσουμε την περίπτωση όπου δρομολογούμε τις διαθέσιμες εργασίες σε P μηχανές. Εδώ, πρέπει να σχολιάσουμε πως τα αποτελέσματα είναι παρόμοια με πριν καθώς αυτή η εκδοχή του προβλήματος μπορεί να παρομοιαστεί με την χρονοδρομολόγηση σε μία μηχανή με την διαφορά ότι η μηχανή εκτελεί με P φορές μεγαλύτερη ταχύτητα κάθε διεργασία. Το μοντέλο είναι παρόμοιο με προηγουμένως. Δηλαδή, όλες οι εργασίες έχουν δημιουργηθεί την χρονική στιγμή ίση με μηδέν και επιτρέπεται η διακοπή τους και η συνέχισή τους σε μεταγενέστερη χρονική στιγμή. Τα αποτελέσματα που προκύπτουν είναι τα εξής.

Theorem 0.1.5. *Για την στατική χρονοδρομολόγηση εργασιών σε p μηχανές, κάθε ντερμινιστικός και σε περιβάλλον με πλήρη γνώση αλγόριθμος A έχει λόγο προσέγγισης μεγαλύτερο ίσο από* $2 - \frac{2p}{n+p}$.

Για την απόδειξη ,όπως και πριν , θεωρούμε οποιοδήποτε αλγόριθμο A και τον εκτελούμε για μία χρονική μονάδα. Κατόπιν, θεωρούμε το μέγεθος κάθε διεργασίας ίσο με τον χρόνο που έχει λάβει συν μία αμελητέα θετική ποσότητα που τείνει στο μηδέν.

Theorem 0.1.6. *Για την στατική χρονοδρομολόγηση εργασιών σε p μηχανές, ο Αλγόριθμος Χρονοπρογραμματισμού εκ περιτροπής έχει λόγο προσέγγισης ίσο με* $2 - \frac{2p}{n+p}$.

Για την απόδειξη του παραπάνω θεωρήματος ήταν αρχικά απαραίτητος ο υπολογισμός του χρόνου εκτέλεσης κάθε διεργασίας. Αυτό μπορεί να επιτευχθεί αναδρομικά για κάθε εργασία και στην συνέχεια αθροίζοντας πάνω σε όλες τις εργασίες να σχηματίσουμε αναλυτικά την αντικειμενική συνάρτηση. Σχηματίζοντας τις αντικειμενικές συναρτήσεις γίνεται εφικτή η μαθηματική βελτιστοποίηση τους όπως πριν στην περίπτωση της μίας μηχανής.

Συνοψίζοντας, η δουλειά των Phillips et al. [1993] αποτελεί αφετηρία για πολλές εργασίες πάνω σε Χρονοδρομολόγηση εργασιών. Τα εργαλεία και οι ιδέες για την απόκτηση κάτω φραγμάτων και της βελτιστοποίησης των αντικειμενικών συναρτήσεων βρίσκουν εφαρμογή και σε άλλες εφαρμογές. Τέλος, κλείνοντας αυτό το κεφάλαιο θα σχολιάσουμε το χειρότερο δυνατό στιγμιότυπο για τον αλγόριθμο Χρονοπρογραμματισμού εκ Περιτροπής το οποίο αποτελεί η περίπτωση όπου όλες οι εργασίες έχουν ίσο μέγεθος. Ποιοτικά ο αλγόριθμος αυτός έχει σκοπό να ολοκληρώσει την επεξεργασία των εργασιών σε σειρά αυξανόμενου μεγέθους από ο καλύτερος

παντογνώστης αλγόριθμος (SPT) πληρώνοντας ένα χρονικό τίμημα για να το επιτύχει. Το χειρότερο δυνατό στιγμιότυπο αποτελεί την περίπτωση όπου οι μεταθέσεις που πράττει ο αλγόριθμος είναι περιττές και με όποια σειρά και να δρομολογήσει τις εργασίες χωρίς να τις διακόψει, αυτό θα αποτελεί την βέλτιστη στρατηγική.

0.2 Χρονοδρομολόγηση Εργασιών Υπό Αβεβαιότητα (Explorable Uncertainty in Scheduling)

Πριν αναλύσουμε την έννοια της Αβεβαιότητας σε προβλήματα Χρονοδρομολόγησης θα κάνουμε μία εισαγωγή στους Αλγόριθμους Ερωτημάτων (Query Algorithms). Πιο συγκεκριμένα, σε αυτή την κλάση αλγορίθμων τα δεδομένα εισόδου δεν είναι ακριβή και στην θέση τους δίνεται ένα διάστημα στο οποίο ανήκει η πραγματική τους τιμή, η οποία μπορεί να μαθευτεί μέσω ενός ερωτήματος (query) το οποίο έχει κάποιο κόστος. Σκοπός του αλγορίθμου είναι να βρει την λύση σε κάποιο δοθέν πρόβλημα ελαχιστοποιώντας παράλληλα το κόστος των ερωτημάτων που διενεργήθηκαν. Ένα παράδειγμα είναι το εξής. Ας υποθέσουμε ότι η είσοδος του προβλήματος είναι n το πλήθος διαστήματα πραγματικών αριθμών, σε καθένα από τα οποία ανήκει καθένας όρος από μία ακολουθία. Ο αλγόριθμος επιτρέπεται να κάνει χρήση ερωτημάτων, όπου το καθένα έχει κόστος 1 και λαμβάνει πίσω την ακριβή τιμή κάθε όρου της ακολουθίας που φυσικά ανήκε στο προηγούμενο διάστημα. Σκοπός του αλγορίθμου είναι να βρεθεί ο ελάχιστος αριθμός ερωτημάτων ώστε να τοποθετηθούν οι όροι της ακολουθίας σε αύξουσα διάταξη. Στην συνέχεια, ορμώμενοι από αυτήν την κλάση αλγορίθμων θα την επεκτείνουμε σε προβλήματα Χρονοδρομολόγησης. Πιο συγκεκριμένα, το μέγεθος κάθε εργασίας δεν θα είναι γνωστό εξαρχής αλλά ένα διάστημα στο οποίο ανήκει.

Στην συνέχεια της κάτωθι διπλωματικής εργασίας θα μελετήσουμε το πρόβλημα της Χρονοδρομολόγησης από την σκοπιά της Αβεβαιότητας. Πιο συγκεκριμένα, βασιζόμενοι στις δουλειές των Albers and Eckl [2020] και Albers and Eckl [2021] θα ορίσουμε το νέο μοντέλο του προβλήματος. Πλέον, κάθε εργασία χαρακτηρίζεται από τρία μεγέθη. Τον χρόνο άφιξης της r_i, το μέγεθος του τεστ της t_i και ένα άνω φράγμα στον χρόνο εκτέλεσης της u_i. Αν επιθυμούμε να εκτελέσουμε την εργασία χωρίς να δρομολογήσουμε το τεστ της νωρίτερα τότε αυτή θα τρέξει με μέγεθος u_i. Αν όμως εκτελέσουμε νωρίτερα το τεστ τότε κατά την αποπεράτωση του τεστ θα μάθουμε το πραγματικό μέγεθος της εργασίας p_i και η εργασία πλέον θα εκτελεστεί με αυτό το μέγεθος. Ποιοτικά, θα σημειώσουμε ότι το τεστ αντικατοπτρίζει έναν βελτιστοποιητή κώδικα (code optimizer) ή έναν συμπιεστή αρχείου. Για παράδειγμα, κατά την αποστολή ενός αρχείου έχουμε δύο επιλογές. Είτε να στείλουμε το αρχείο ασυμπίεστο είτε να δαπανήσουμε χρόνο για την συμπίεση του ώστε να απασχολούμε για λιγότερο χρόνο το κανάλι επικοινωνίας για την αποστολή του. Βασιζόμενοι στα παραπάνω καλούμαστε να επιλύσουμε το ίδιο πρόβλημα με πριν, δηλαδή να ελαχιστοποιήσουμε το άθροισμα των χρόνων εκτέλεσης κάθε εργασίας σε μία ή πολλές μηχανές όταν όλες είναι διαθέσιμες από την χρονική στιγμή $t = 0$ και όταν επιτρέπεται η διακοπή μίας εργασίας και η συνέχισή της αργότερα. Η συμβολή μας σε μία μηχανή είναι να δώσουμε μία πιο απλή και λεπτομερή απόδειξη από αυτή των Albers and Eckl [2020] λαμβάνοντας παράλληλα λόγο προσέγγισης $2(1 - \frac{1}{n+1})\phi$ αντί για 2ϕ, όπου ϕ είναι η μαθηματική σταθερά της χρυσής τομής. Γενικεύοντας για πολλές μηχανές λαμβάνουμε λόγο προσέγγισης ίσο με $2(1 - \frac{p}{n+p})\phi$. Τέλος, θα συμβολίσουμε με p_j^A το μέγεθος της εργασίας όπως καθορίζεται

στον αλγόριθμο $\mathcal{A}$ και ρ_i τον χρόνο στον βέλτιστο αλγόριθμο.

Σύμφωνα με τα παραπάνω, για την σχεδίαση ενός αλγορίθμου που λύνει αυτό το πρόβλημα αρχικά πρέπει να σχεδιαστεί ένα κριτήριο για την εκτέλεση ή μη του τεστ. Το πιο απλό κριτήριο είναι να θεωρήσουμε ένα κατώφλι α και να συγκρίνουμε τον λόγο $\frac{u_i}{t_i}$ με αυτό το κατώφλι. Με βάση αυτό το κριτήριο το μέγεθος της εργασίας όπως προκύπτει από τον αλγόριθμο απέχει κατά μία σταθερή πολλαπλασιαστική σταθερά από το μέγεθος στον βέλτιστο αλγόριθμο. Έτσι, προκύπτει το ακόλουθο λήμμα.

Lemma 0.2.1. *1.* $\forall j \in T : t_j \leq \rho_j, p_j \leq \rho_j$

2. $\forall j \in T : p_j^A \leq (1 + \frac{1}{\alpha})\rho_j$

3. $\forall j \in N : p_j^A \leq \alpha\rho_j$

Λύνοντας την εξίσωση $\alpha = 1 + \frac{1}{\alpha}$ λαμβάνουμε $\alpha = \phi \approx 1.618$. Ο αλγόριθμος που παραθέτουμε για την λύση του παραπάνω προβλήματος είναι ο εξής:

Algorithm 1: Golden Round Robin

```
for all J_i ∈ [m] do
    if u_i / t_i ≥ φ then
        Εκτέλεσε το τεστ για την εργασία i
    end
    else
        Μην εκτελέσεις το τεστ για την εργασία i
    end
end
Τρέξε τον Αλγόριθμο Χρονοπρογραμματισμού εκ περιτροπής.
```

Theorem 0.2.2. *Ο Αλγόριθμος Χρονοπρογραμματισμού εκ περιτροπής βασιζόμενος στην σταθερά της χρυσής τομής (Golden Round Robin) είναι* $2(1-\frac{1}{n+1})\phi$ *προσεγγιστικός για την ελαχιστοποίηση του αθροίσματος των χρόνων εκτέλεσης κάθε εργασίας.*

Η απόδειξη ποιοτικά βασίζεται στα εξής βήματα. Για την ίδια εργασία i, το μέγεθος της εργασίας p_i^A όπως καθορίζεται στον αλγόριθμο $\mathcal{A}$ και ρ_i ο χρόνος στον βέλτιστο αλγόριθμο είναι δυνατόν να διαφέρουν. Άρα η μερική διάταξη στα μεγέθη των εργασιών στον Αλγόριθμο Χρονοπρογραμματισμού εκ περιτροπής βασιζόμενος στην σταθερά της χρυσής τομής και τον βέλτιστο παντογνώστη αλγόριθμο είναι διαφορετική. Επιπλέον, οι εργασίες αποπερατώνονται σε αύξουσα σειρά στον Αλγόριθμο Χρονοπρογραμματισμού εκ περιτροπής βασιζόμενος στην σταθερά της χρυσής τομής. Αν αθροίσουμε στην αντικειμενική συνάρτηση με διαφορετική σειρά τις εργασίες όπως με την σειρά του βέλτιστου αλγορίθμου, θα δημιουργήσουμε μία μεγαλύτερη ή ίση αντικειμενική συνάρτηση. Χρησιμοποιώντας, τέλος, το παραπάνω Λήμμα αλλά και τον λόγο προσέγγισης του Αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής λαμβάνουμε το αποτέλεσμα. Η απόδειξη είναι παρόμοια για πολλές μηχανές.

Theorem 0.2.3. *Ο Αλγόριθμος Χρονοπρογραμματισμού εκ περιτροπής βασιζόμενος στην σταθερά της χρυσής τομής για την ελαχιστοποίηση του προβλήματος χρονοδρομόλογησης σε πολλαπλές μηχανές, όπου επιτρέπεται η διακοπή των εργασιών και η συνέχιση τους από το ίδιο σημείο αργότερα, ελαχιστοποιώντας το άθροισμα των χρόνων αποπεράτωσης κάθε εργασίας (P| pmtn,nonclair.| $\sum_i C_i$) υπό Αβεβαιότητα είναι* $(2 - \frac{2p}{n+p})\phi$ *προσεγγιστικός.*

Σε αυτό το σημείο παρατηρούμε πόσο χρήσιμη ήταν η απόδειξη του Αλγόριθμου Χρονοπρογραμματισμού εκ περιτροπής καθώς οι ίδιες τεχνικές δουλεύουν και στην περίπτωση της εκδοχής του προβλήματος υπό αβεβαιότητα. Να σημειώσουμε πως δεν έχει βρεθεί ο καλύτερος δυνατός αλγόριθμος για το πρόβλημα ούτε ακόμα στην περίπτωση που όλα τα τεστ t_i έχουν το ίδιο μέγεθος.

0.3 Εύρωστοι (Robust) και Συνεπείς (Consistent) Αλγόριθμοι

Τα αποτελέσματα που πήραμε στα προηγούμενα κεφάλαια ήταν σε ανάλυση χειρότερης περίπτωσης. Αυτό πολλές φορές στην πράξη μπορεί να μας οδηγήσει να απορρίψουμε αλγορίθμους που στην πράξη έχουν καλή απόδοση αλλά έχουν κακό λόγο προσέγγισης στην χειρότερη περίπτωση. Με αφορμή αυτό, τα τελευταία χρόνια οι ερευνητές προσπαθούν να συνδυάσουν αλγορίθμους για προβλήματα όπου η είσοδος γίνεται γνωστή σταδιακά και αλγορίθμους που εκμεταλλεύονται προβλέψεις από συστήματα μηχανικής μάθησης. Εν προκειμένω αντί για προβλέψεις έχουμε το τεστ για κάθε εργασία, το οποίο μόλις τρέξει μας φανερώνει το αληθινό μέγεθος κάθε εργασίας. Σκοπός μας είναι ο αλγόριθμος που θα κατασκευάσουμε να είναι εύρωστος, δηλαδή όταν τα σφάλματα από τις προβλέψεις είναι υψηλά να συμπεριφέρεται καλά και συνεπής, δηλαδή όταν τα σφάλματα είναι μικρά να πλησιάζει η απόδοσή του αυτή του καλύτερου αλγόριθμου.

Σε αυτό το κεφάλαιο, όπως και στα προηγούμενα, προσπαθούμε να λύσουμε το $1|pmtn| \sum_i C_i$. Το μοντέλο σε αυτή την περίπτωση είναι το εξής. Κάθε εργασία είναι διαθέσιμη από την χρονική στιγμή $t = 0$. Στην αρχή γνωρίζουμε μόνο το μέγεθος του τεστ για κάθε εργασία. Αν εκτελέσουμε το τεστ τότε μαθαίνουμε το πραγματικό μέγεθος της εργασίας p_i και την εκτελούμε με αυτό το μέγεθος. Επιπλέον, μπορούμε να τρέξουμε εργασίες στην μηχανή με διαφορετική ταχύτητα και επιτρέπεται να διακόψουμε μία εργασία. Το παραπάνω μοντέλο προσπαθήσαμε να αποτελεί μία απλή εκδοχή και γενίκευση των προηγούμενων κεφαλαίων.

Σε άλλα μοντέλα καθοδηγούμενα από την Μηχανική Μάθηση πάνω σε Χρονοδρομολόγηση δινόταν ταυτόχρονα με την έκδοση της εργασίας μία πρόβλεψη για το μέγεθος της. Αυτή η πρόβλεψη μπορούσε να ήταν είτε ακριβής είτε ανακριβής αλλά δεν είχε κάποιο έξτρα χρονικό κόστος. Σε αυτό το μοντέλο το τεστ αποτελεί την πρόβλεψη η οποία είναι ακριβής αλλά έχει χρονικό κόστος. Επιλέξαμε αυτό το μοντέλο γιατί θεωρούμε ότι ταιριάζει καλύτερο σε ένα βελτιστοποιητή κώδικα ή έναν συμπιεστή αρχείων καθώς και θεωρούμε ότι πρέπει να μελετηθεί και η υπόθεση όπου μεσολαβεί κάποιο διάστημα ανάμεσα στην έκδοση μίας εργασίας και της πρόβλεψης του μεγέθους της.

Αρχικά, πρέπει να ορίσουμε το σφάλμα στον αλγόριθμο. Στους κλασσικούς αλγορίθμους με προβλέψεις , έστω $\hat{y}_i$, το σφάλμα ορίζεται ως $\eta_i = |\hat{y}_i - y_i|$. Στην περίπτωση μας, όπου αντί για

προβλέψεις έχουμε τα τεστ επιλέξαμε να ορίζεται ως εξής :

$$\eta_j = |t_j + p_j - p_j|$$

$$\eta = \sum_j \eta_j = \sum_j t_j$$

Για την αποφυγή σύγχυσης του η με το n συμβολίζουμε $T := \sum_j t_j$. Παρακάτω, θα παρουσιάσουμε τον αλγόριθμο για το πρόβλημα και την απόδοσή του.

0.4 Algorithm

Algorithm 2: Συνδιάζοντας τον Αλγόριθμο Χρονοπρογραμματισμού εκ περιτροπής με Τεστ

$n \in \mathbb{N}, (t_i, p_i) \forall i \in [n]$, $\lambda \in [0,1]$ $T \leftarrow \sum_{i=1}^{n} t_i$ $t \leftarrow 0$ /* t → time */

$F \leftarrow \emptyset$ /* Το F είναι το σύνολο των εργασιών που εκτελέστηκαν */

while $t \leq \frac{T}{\lambda}$ **do**

  /* Σε $\frac{T}{\lambda}$ χρόνο όλα τα τεστ θα έχουν ολοκληρωθεί. */

  Τρέξε τον Αλγόριθμο Χρονοπρογραμματισμού εκ περιτροπής με $1 - \lambda$ μονάδες ταχύτητας. Εκτέλεσε τα Τεστ σύμφωνα με τον SPT αλγόριθμο σε λ μονάδες ταχύτητας. /* Η ταχύτητα είναι κανονικοποιημένη στην μέγιστη ταχύτητα της μηχανής */

  Για κάθε εργασία που ολοκληρώθηκε, πρόσθεσέ την στο F.

end

if $\ni job \in F^c$ **then**

  /* $\frac{T}{(1-\lambda)n} \leq 1$ */

  Τρέξε τις υπόλοιπες εργασίες p_i σε Αλγόριθμο Συντομότερου Εναπομείναντος Μεγέθους (SRPT).

end

Theorem 0.4.1. *Ο λόγος προσέγγισης του Συνδιασμού του Αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής με Τεστ είναι* $\frac{T(\frac{n}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{(n+1)}{2})}{\frac{n(n+1)}{2}} + 1$.

Ο λόγος που αρχικά εκφράζουμε παραμετρικά με το T την απόδοση του αλγορίθμου είναι ο εξής. Αν το σφάλμα ή αλλιώς τα τεστ είναι γραμμικά ή υπογραμμικά με το *n*, τότε η απόδοση του αλγορίθμου ασυμπτωτικά ισούται με αυτή του βέλτιστου αλγορίθμου. Με βάση αυτά μπορούμε να πούμε ότι ο αλγόριθμος είναι 1-συνεπής (1-consistent). Μένει λοιπόν να μελετήσουμε την απόδοση του στην περίπτωση όπου τα τεστ έχουν μεγάλο μέγεθος και καθυστερούν την αποπεράτωση των εργασιών. Αυτή η περίπτωση περιγράφεται στο παρακάτω λήμμα. Να τονίσουμε εδώ πως αν τα τεστ έχουν αυθαίρετα μεγάλο μέγεθος, ο αλγόριθμος Χρονοπρογραμματισμού εκ περιτροπής θα τερματίσει πρώτος και συνολικά ο αλγόριθμος θα έχει λόγο προσέγγισης ίσο με $\frac{2}{\lambda}$.

Lemma 0.4.2. *Ο Συνδιασμός του Αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής με Τεστ έχει στην χειρότερη περίπτωση λόγο προσέγγισης ίσο με* $\frac{2}{1-\lambda}$.

Το χειρότερο δυνατό στιγμιότυπο στο οποίο αντιστοιχεί ο παραπάνω λόγος προσέγγισης είναι όταν η διαφορά της ποσότητας των εργασιών που έχουν ήδη επεξεργαστεί μέχρι να ολοκληρωθούν τα τεστ με αυτή των τεστ είναι μία θετική αλλά πολύ μικρή ποσότητα. Σε αυτή την περίπτωση, κατά την ολοκλήρωση των τεστ ο αλγόριθμος SRPT θα τρέξει για αμελητέο χρόνο και δεν θα βελτιώσει την συνολική απόδοση. Έτσι, η απόδοση του αλγορίθμου θα είναι ίση με αυτή του Αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής πολλαπλασιασμένη με το ποσοστό της μείωσης της ταχύτητας ($\frac{1}{\lambda}$) με την οποία τρέχαμε τον συγκεκριμένο αλγόριθμο.

Η τιμή του λ δεν μπορεί θεωρητικά να εκτιμηθεί μέσω της ανάλυσης χειρότερης περίπτωσης. Ίσως με άλλα είδη αναλύσεως όπως η στοχαστική ανάλυση να μπορεί να εξαχθεί κάποιο συμπέρασμα για την τιμή του. Ωστόσο, αυτό απαιτεί μία υπόθεση για την κατανομή των μεγεθών των εργασιών. Για αυτό τον λόγο προτιμήσαμε να μην κάνουμε κάποια επιπλέον υπόθεση η οποία μπορεί να αποδειχθεί εσφαλμένη και προτιμήσαμε να σχεδιάσουμε πειράματα όπου μετράμε την απόδοση του παραπάνω αλγορίθμου και την συγκρίνουμε με αυτή του Αλγορίθμου Συντομότερης Εργασίας και του Αλγορίθμου Χρονοπρογραμματισμού εκ περιτροπής. Τα πειράματα σχεδιάστηκαν στην γλώσσα Python και τα διαγράμματα που ελήφθησαν βρίσκονται στο σκέλος 5 και ο κώδικας στο Α. Πειραματικά, λοιπόν, βρήκαμε ότι για $\lambda \geq \frac{1}{2}$ ο αλγόριθμος συμπεριφέρεται καλύτερα. Αυτό είναι αναμενόμενο κάπως καθώς ο σκοπός των τεστ είναι να βελτιώσουν την απόδοση του αλγορίθμου και σπάνια η τιμή κάθε τεστ θα ήταν τόσο κοντά σε αυτή του μεγέθους της αντίστοιχης εργασίας.

Κείμενο στα αγγλικά

Chapter 1

Introduction

One of the most crucial role in today's manufacturing and service industries is that of Scheduling. Scheduling is a decision making process where tasks must be assigned to resources while minimizing a certain objective function. Due to its tremendous applications since the past century, a collection of scheduling problems have been studied in detail by the Computer Science community. Each variation of the problem is inextricably tied with a different set of constraints. To explain in more detail, the a priori knowledge of the exact characteristics of the jobs belong to Non-Clairvoyant category. On the other side, their ignorance lie in the Clairvoyant regime. Moreover, the availability of the jobs to be scheduled define two major categories. Static scheduling where all jobs are available from the beginning and the dynamic case where the release times of the jobs are not necessarily zero. Another fundamental sector of scheduling algorithms is the absence or not of precedence constraints where the completion of some jobs must be completed before start executing some others. Every scheduling problem may satisfy a mixture of different constraints and in parallel minimize different objective functions. Not only distinct constraints but also non identical objective functions require totally different algorithms. To give a simple example of a scheduling problem is to allocate multiple programs that need to be executed in a single or multiple processors. Another example is to define the transmission order of files over a network, delivering post orders etc. A common objective function is the sum of the completion times of every job.

Scheduling has made its appearance since the ancient years. As Xenophon wrote in his famous novel "Kyrou Anabasis", Artaxerxes defeated his brother, Kyros, and maintained the crown of the Persian Kingdom. After Kyros' defeat and the death of some Greeks generals, the need to allocate soldiers to different groups has risen. The objective was take back the troops to Greece while minimizing the time needed and the possible losses. More recent examples of scheduling come from the fields of Operations Systems and Operations Research Theory. From the beginning of the 20^{th} century, graphic methods have been developed to determine the order and the timing of usage of a common resource in the industry. In 1956 Smith introduced the Shortest Processing Time (SPT) algorithm where it solves the single machine model while minimizing the sum of the completion times of all jobs. Until the 70's, various algorithms have been developed for simple models such as single machine scheduling with respect to minimizing the weighted sum of completions times. One of the most fundamental results was that of Graham et al. [1979]. To explain in more detail, given *P* parallel machines, a set of jobs $J = \{j_i \in \{1, 2, \ldots, n\}\}$ and a set of precedence constraints, that means if $j_i \prec j_j$ then job j_i must be completed before job j_j begins. The goal is to minimize the maximum of all completion times

that is the *makespan*. Even for the case of $P = 2$ Graham et al. proved that the problem is *NP-Hard*. The majority of scheduling problems, that have been studied since the 90's, have been studied in the *Clairvoyant* setting where the size of each job is considered to be known when the job becomes available. Although, in various applications, that come from Operating Systems, jobs' characteristics, for example the processing time, are considered to be unknown. In 1993, Phillips et al. [1993] proposed the famous *Round Robin* algorithm for minimizing the sum of completion times for both single and multiple machine in the *Non-Clairvoyant* environment. In their work, a job can be preempted and continue its execution later at no extra cost. Round Robin solves this problem in the static case, i.e. where all jobs are available at the same time. Since then, various extensions of this problem have been studied using extra constraints both in offline and online cases.

In parallel with the theory of Scheduling, the sector of *Query Algorithms* has also made huge improvements. In this type of problems, the value of the input is not specified in precise and it is permissible to apply a query to earn its exact value. In this category of algorithms the goal is to find the best solution for a problem while minimizing the number of applied queries. To give an example, in 1991 Kahan [1991] first considered a model that is motivated by air-traffic routing where a given neighbor of the position of each plane is known in advance but not its exact position, which can be acquired by applying a query at an extra cost. In another model, Olston and Widom [2000] studied the use of queries in distributed databases. More precisely, the values of a function are stored accurately in a central database while a certain interval, that contains the exact value, is stored in local databases. The goal is to compute the result of the application of some known operators (such as max, min, mean value) to the given function while minimizing the cost of the applied queries.

The query framework has also been used in scheduling problems, in a domain called *Scheduling Under Uncertainty*. Several works have appeared in the literature, such as Dürr et al., Albers and Eckl and Bampis et al., combining query models and classical scheduling. More specifically, each job i, that needs to be scheduled, is released with an *upper bound* u_i on its processing time and a *test size* t_i. If the job runs untested, its processing time is u_i. Otherwise, if we first test the job, its real processing time p_i is revealed, only after the completion of the test, and then the job can be executed using time p_i. The motivation behind this combination, is that the test represents a code optimizer or a file compressor. If the code optimizer is applied to a given job, its processing time may be decreased. In the file transmission setting, a file can be compressed before transmission and possibly reduce its size. In contrast with the *Query Algorithms*, in *Scheduling Under Uncertainty* the cost of each test-query contributes to the objective function of the scheduling problem that needs to be minimized instead of trying minimizing the number of applied queries. In Albers and Eckl [2020] and Dürr et al. [2018] they examined the problem of scheduling into a single machine while minimizing the sum of the completion times and in Bampis et al. [2021] they minimized the energy consumption when using different processing speeds.

Due to recent advances in Machine Learning and the high percentages of accuracy in various models, researchers tried to solve classic problems in algorithmic theory by using predictions from ML models. This approach introduce the new sector of Learning Augmented Algorithms. In practice researchers are trying to combine robust approximation algorithms that have worst-

case guarantees with advises from machine learning models. The use of Machine Learning Predictions is in a black-box manner without any assumption on the distribution of the error. On the one side, the aim is to use these predictions and when they are accurate, to obtain an algorithm that performs near optimal and on the other side when predictions go wrong, the algorithm must remain robust and do not acquire unbounded performance. A classic approach in this field is that of Lykouris and Vassilvtiskii [2018] where they considered the paging problem under machine learning predictions for the future demand or not of a certain page. In the sector of Scheduling, there are several works in Non-Clairvoyant Scheduling under predictions such as those of Purohit et al. [2018] and Im et al. [2021]. In a more detail, the exact processing size of each job remains unknown but each job comes with a prediction for its size. In most cases, the robust algorithm is *Round Robin* and it is modified in several manners in order to take the advantage of good predictions while remaining robust when predictions are inaccurate. The need of these type of algorithms in Scheduling comes from real applications. To give an example, in Operating Systems Theory a huge amount of tasks are processed in a regular basis and their size can be estimated from past observations with a good precision. The analysis of this type of Algorithms is parameterized according to an error in the predictions.

Prediction models in Learning Augmented Scheduling Algorithms consider each prediction to be completed immediately or being processed in a negligible amount of time. Although, in practice, several observations from Operating Systems Theory state that this is not the case. Prediction systems can process a job for a non negligible amount of time or they may run a short simulation of the job to derive a safe result. The difference, now, is that each prediction may take time and postpone the completion time of the rest of the jobs but it gives a precise result for the job's size. With previous experience from the *Scheduling Under Uncertainty* we consider a new model, where the prediction is similar to a test and when its execution is finished we know the exact processing size of the job. The difference with the models that come from *Scheduling Under Uncertainty* is that each job is not now having an upper bound, its processing time remains the same but it is unknown before the completion of test. The scheduler, now, has to design a mixture of processing tests in parallel with jobs.

1.1 Our contribution

In this project, in the first place, we explore the fields of Non-Clairvoyant Scheduling, Scheduling Under Certainty and Learning Augmented Algorithms and we try to combine the notion testing from the middle field to derive a robust and consistent algorithm. We analyze our algorithm according to the parameter T that is the sum of all tests and we use competitive analysis to derive the worst case scenario that is independent of this parameter. In case of the T to be sub-linear in terms of the number of jobs n then our algorithm performs near optimal. On the other side, if T is arbitrarily large the performance ratio of our algorithm is a constant factor multiplied with the performance of Round Robin. We introduce a parameter λ. According to its value, we regulate in which algorithm (Round Robin or Testing) we give more weight. Without any assumption on the distribution of the sizes of the Tests, we cannot draw a conclusion for the optimal value of λ, so we try to estimate this value experimentally. To sum up, we refer our main results:

- Rewriting Round Robin performance analysis (Phillips et al. [1993]) in a more analytical

way.

- Use the above analysis to solve the same problem under Uncertainty (Albers and Eckl [2020]) for both single and multiple machines.
- Combine the notion of testing with the classical Non-Clairvoyant setting to derive a Learning Augmented Algorithm.

1.2 Organization

We start our work by examining the classic paper of Phillips et al. [1993] in Non-Clairvoyant Scheduling. We focus on the famous *Round Robin* Algorithm and we give a more detailed proof for its competitive ratio that is $2(1 - \frac{1}{n+1})$. To prove that, we used a general optimization technique that is useful in variety of problems we met up. Then, we use this technique to write a complete proof for the multiple machines variation of the problem that achieves a competitive ratio of $2 - \frac{2p}{n+p}$.

Using the previous results we change setting and we study Scheduling Under Explorable Uncertainty based on Albers and Eckl [2020] and Albers and Eckl [2021]. We derive an alternative proof for Albers and Eckl [2020] Golden Round Robin Algorithm that achieves a ratio of $2(1 - \frac{1}{n+1})\phi$ instead of 2ϕ and using this technique we extend this algorithm to multiple processors.

Finally, we try to combine the field of Learning Augmented Algorithms with Explorable Uncertainty and we try to build a robust and consistent algorithm using testing for learning the exact workload of each job. All jobs are released from the beginning and they can be executed at different rates. As a job is released we only know its test size t_i. If we run a job untested, its processing time is p_i and it is unknown until its completion. On the other side, if we, firstly, test the job, we learn the exact processing time of the job at the completion of the test. We define as total error the sum of testing times T and we combine a Robust Algorithm as *Round Robin* with testing by running the two algorithms in parallel with different rates. More precisely, at the beginning, we run *Round Robin* at λ rate and all the tests in 1 - λ rate. When all the tests have been completed, we know the exact workload of every job and as some of the processing size has already been executed in *Round Robin* fashion, we execute the rest in a *Shortest Remaining Processing Time (SRPT)* way. The parameterized competitive ratio is $\frac{T(\frac{n}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{(n+1)}{2})}{\frac{n(n+1)}{2}} + 1$ while in worst case analysis the algorithm is $\frac{2}{1-\lambda}$ competitive. The worst case instance is when each test is negligibly larger than the processing time of its corresponding job and after the completion of the tests, *SRPT* algorithm run for a negligible amount of time. We did not find a theoretical way to derive the value of the rate λ. To overcome this obstacle, we design experiments whose results showed that for values of λ near 1 the algorithm behaves better.

Chapter 2

Non Clairvoyant Scheduling Theory

Scheduling is one of the most fundamental and well studied topics in Computer Science Literature. More specifically, Scheduling is a decision making process that aims at allocating resources to tasks over a given time period while minimizing one or more objectives. To provide with some examples, the resources may be CPU cores in a personal computer or in large cloud systems, runways or terminal gates at an airport, crews in large shipping companies. On the other hand, tasks may be processes in an Operation System, airports or passengers, shipping routes etc. Tasks are characterized by their release date, their deadline (if defined) and their priority level. It is easily understood that in real world applications not all tasks have the same priority. For example, in operating systems, processes assigned by the kernel of the OS have infinity priority versus to users' programs. As we mentioned before, the main goal of a Scheduling Algorithm is to provide a feasible assignment such that we minimize a given objective. It is true that, for a given problem under different objectives we need a completely different algorithm.

To give some examples of Scheduling problems, we will start with Graham [1966] where it is described one of the most famous and fundamentals problems in this area. We have m identical machines, n independent and nonpreemptable jobs and the goal is to minimize the *makespan*(C_{max}), that is the completion time of the last job that leaves the machine. There are two main variations of this problem, the static and the dynamic case. In the static case all jobs have release time equal to zero and in the dynamic case there are release times greater than zero for which the scheduler is unaware of until their arrival. Even for the case $m = 2$, both in static and online case, the problem is **NP-Hard** because it can be reduced to the problem of **Partition**. For the online case, Graham proposed an heuristic, the *List Algorithm* which is $2 - \frac{1}{m}$ competitive for $m = 2, 3$. Coffman and Graham [1972] proved that there exists a $2 - \epsilon$ competitive algorithm for all m that approaches 2 as m goes to infinity. We notice that in the above problem all job sizes are known in advance. Although, the same problem can be considered with an almost limitless number of variants. For example, there can be precedence constraints between jobs or the jobs can run at different speeds at different machines or a job can consit of one or multiple task or a job can be available at a release time or it can be preemptable or its size may be known only after its completion time. In Błażewicz et al. [1988], Conway et al. [2003], Graham et al. [1979] they compact the most famous results in Scheduling Problems until 70's.

2.0.1 Notation for Deterministic Models

In this section we are going to mention the variety of different criteria that have been developed in theoretical studies of scheduling. These criteria reflect the variety of different circumstances in which scheduling problems arise and the different costs and values that are relevant in each case.

First, we will start with some notation. We denote the number of available individual machines with *1,2,...,m* and the number of jobs to be scheduled with *1,2,...,n*. The case where $m < n$ is a trivial one where we can solve the problem optimal by assigning every job to a single machine, so we consider $m > n$. The relevant attributes of job *i* that are given as part of the problem description are denoted by the following variables.

1. r_i is the *release time*, that is the time the job is available to the machine.
2. d_i is the *due-date*, that is the time by which the processing of the last operation should have been completed.
3. c_i is the *completion time*, that is the time by which the job is finished.
4. $p_{i,j}$ is the processing time of the job i to the machine j. If all machines are identical then we use p_i.
5. wt_i is the *waiting time* or *flow time*, that is the amount of time which elapses between its release and completion so $wt_i = c_i - r_i$.
6. w_i is the weight of job i, that is a priority factor, denoting the relevant importance of job i versus to the other jobs in the systems.

According to Graham notation, a scheduling problem is described by at triplet $\alpha|\beta|\gamma$. The field α describes the machine environment and takes just one entry. The field β provides details of processing characteristics and constraints. It may contain no entry, single or multiple entries. The γ field describes the objectives to be minimized.

Now, we will look in more details the contents of each field. The possible environments specified in the α field are :

- **Single machine** (1) The case of a single machine is the simplest of all possible machine environments and is a special case of all other more complicated machine environments.
- **Identical parallel machines** (P_m) There are *m* identical machines in parallel If a job j cannot be processed on just any machine, but only on any belonging to a specific subset M_j, then the entry M_j appears in the β field.
- **Parallel Machines with different speeds** (Q_m) The speed of machine *i* is denoted by u_i and the time that job *j* spends on machine *i* is $\frac{p_j}{u_i}$.
- **Unrelated machines in parallel** (R_m) There are *m* different machines in parallel. Machine *i* can process job *j* at speed u_{ij}.

- **Flow shop** (F_m) There are *m* machines in series, each job has to be on each one of the *m* machines and all jobs have to follow the same route.
- **Flexible flow shop** (FFc) A flexible flow shop is a generalization of the flow shop and the parallel machine environments. Instead of *m* machines in series there are *c* work centers where in each work center there is a number of identical machines in parallel. Each job has its own route to follow through the shop.
- **Open shop** (O_m) There are m machines. each job has to be processed again on each one of the *m* machines. However, some of these processing times may be zero.

The processing restrictions and constraints specified in the β field may include multiple entries. Possible entries in the β field are:

- **Release dates** (r_j) If this symbol appears in the β field, then job *j* cannot start its processing before its release date r_j. If r_j does not appear in the β field, the processing of job j may start at any time. In contrast to release dates, due dates are not specified in this field. The type of objective function gives sufficient indication whether or not there are due dates.
- **Preemptions** (pmtn) The scheduler is allowed to interrupt the processing of a job at any point and put a different job on the machine instead. The amount of processing a preempted job already has received is not lost.
- **Precedence constraints** (prec) Precedence constraints may appear in a single machine or in a parallel machines environment, requiring that one or more jobs may have to be completed before another job is allowed to start its processing.
- **Sequence dependent setup times** (s_{jk}) The s_{jk} represents the sequence dependent setup time that is incurred between the processing of jobs j and k.
- **Job families** (fmls) The n jobs belong in this case to F different job families. Jobs from the same family may have different processing times, but they can be processed on a machine one after another without requiring any setup in between. However, if the machine switches over from one family to another, say from family g to family h, then a setup is required.
- **Batch processing** (batch(b)) A machine may be able to process a number of jobs, say b, simultaneously; that is, it can process a batch of up to b jobs at the same time.
- **Breakdowns** (brkdwn) Machine breakdowns imply that a machine may not be continuously available.
- **Machine eligibility restrictions** (M_j) M_j denotes the set of machines that can process job j.
- **Permutation** (prmu) A constraint that may appear in the flow shop environment is that the queues in front of each machine operate according to the First In First Out (FIFO) discipline.

- **Blocking** (block) Blocking is a phenomenon that may occur in flow shops. If a flow shop has a limited buffer in between two successive machines, then it may happen that when the buffer is full the upstream machine is not allowed to release a completed job.
- **No-wait** (nwt) The no-wait requirement is another phenomenon that may occur in flow shops. Jobs are not allowed to wait between two successive ma- chines.
- **Recirculation** (rcrc) Recirculation may occur in a job shop or flexible job shop when a job may visit a machine or work center more than once.

Any other entry that may appear in the β field is self-explanatory. In the γ field we write the objective function to be minimized. The objective to be minimized is always a function of the completion times of the jobs, which, of course, depend on the schedule. We quote some of the most common objective functions.

- **Makespan** (C_{mac}) The makespan is equivalent to the completion time of the last job to leave the system.
- **Sum of completion times** ($\sum_i C_i$) With this metric we sum with equal weight the completion time of every job.
- **Total weighted completion time** Since some jobs may have higher priority to be completed, we give different weights to the completion of the jobs. ($\sum_i w_i C_i$)

2.0.2 Non Clairvoyant Scheduling

Until, 1993 and the famous paper of Motwani et al. Phillips et al. [1993], most researchers studied the clairvoyant setting of scheduling where characteristics of a job such as its size is considered to be known. Since the last assumptions were considered to be false for applications that come of the operating systems, the Non-clairvoyant Scheduling setting has made its appearance. To give an example, imagine running the training of a machine learning model in your computer. No one knows exactly the processing time it needs to be completed. If OS let this process run alone and then serve the others then the starvation phenomenon in OS will arise. In Non-clairvoyant Scheduling the characteristic of each job are considered to be unknown and the aim is to construct algorithms that perform well in the competitive analysis performance measure. The only works on Non-Clairvoyant scheduling until Phillips et al. [1993] were Feldmann et al. [1991] and Shmoys et al. [1991] whose aim was to minimize the makespan. The simplest model that arises in this new setting and matches with the demands of OS developers is $1|prmt, non-clair|\sum_i C_i$. To give a concrete example for the above, the simplest computational model for a computer is one processor where we can stop the processing of a job, save the internal state of the processor in the memory and continue its execution later. The motivation behind the objective is that at a first try we should consider two simple objectives, the one we talk about and the makespan. The problem with the makepsan is that it measures better the performance for the hole batch of jobs and may not differ if we change the relative order of two of them. Considering, now, simple scheduling polices such as FIFO (First In First Out) it easily understood that they can go

arbitrarily bad. Imagine the job with the largest size to be scheduled first and all the other jobs to be equal and have less size. The first thing we have to study is the Lower Bound that each deterministic and static algorithm can achieve and the second to design a fair scheduling policy. Due to intuition reasons we believe that is better to approximate these sections in the opposite way. Intuitively, as we do not know in advance the exact job size we should follow a fair policy and give each job equal processing time. More precisely, we divide each job in arbitrarily small sizes and run each piece in a cyclic way. We will see that this pure and fair policy is the best we can do.

In this section we will discuss both the basic theory and the notation in Non-Clairvoyant Scheduling and we will also present some of the field's core concepts. The analysis in this chapter follows Phillips et al. [1993] which is a classic approach for this problem.

2.1 Competitive analysis as a performance measure

Comparing different algorithms is hard. For almost any pair of algorithms and any measure of algorithmic performance, each algorithm will perform better than the other on some inputs. Especially, in the sector of Online Algorithms the performance of a certain algorithm can have large deviations under different metrics. A famous example is the Frequency Algorithm for the List Accessing Problem. If we analyze the performance of this algorithm according to Competitive Analysis (Worst Case Instance), it can go arbitrarily bad. In other words, its performance is unbounded in comparison with the best offline algorithm. On the other hand, according to average analysis where it is assumed that every element in the list has equal probability with the others to be selected, the Frequency Algorithm minimizes this metric. As we have seen that these probabilistic assumptions usually fail, that is each job that arrives has size that follows some distribution and also we have noticed that there exists correlation between consecutive jobs, all of these lead us to follow competitive analysis. More formally, we follow the definition from Phillips et al. [1993] for the performance ratio of an algorithm A as

$$R_A(n) = sup_{J:|J|=n} \frac{\hat{W}_A(J)}{\hat{W}_{OPT}(J)}$$

where $\hat{W}_A(J)$ is the objective function we use to measure the performance of algorithm A and $\hat{W}_{OPT}(J)$ is the objective function of the best offline algorithm for a certain problem. We will use this analysis to give a more detailed proof for the competitive ratio of Round Robin Algorithm. We follow this analysis because it is robust and works for every instance.

2.1.1 Round Robin, Algorithm and Analysis for $1 \mid r_i = 0 \mid \sum C_i$

Based on Phillips et al. [1993] analysis we will provide a complete analysis of Round Robin Algorithm and prove the approximation ratio it achieves. First of all, we denote that we follow the notation of Phillips et al. [1993].To describe the problem we need a set $J = \{j_1, j_2, \ldots, j_n\}$ of n jobs with their processing times $p_1, p_2, \ldots, p_n$ respectively. Here, we have to say that Round

Robin (RR) is not aware of the exact processing times and only the Optimal Offline Algorithm, which we denote OPT, knows them. As an objective to measure the performance of RR algorithm we use the sum of the completion times of every job $C = \sum_{i=1}^{n} c_i$, where the completion time c_i equals the time when a job finished its processing. Finally, to find the approximation ratio of our algorithm we use the notion of the competitive ratio which is a worst case analysis and we compare an online algorithm with the best optimal offline algorithm. More formally, with $R_A(n)$ we denote the performance ratio of the algorithm and with ALG the objective value of the algorithm :

$$R_A(n) = sup_{J:|J|=n} \frac{ALG}{OPT}$$

Now, we will follow the definition of Round Robin Algorithm (RR) that Phillips et al. [1993] gives.

Definition 2.1.1 (Round Robin as defined in Phillips et al. [1993])**.** The Round Robin (RR) algorithm is a non-clairvoyant scheduling algorithm which at all times ensures that each uncompleted job has received an equal amount of processing time.

The above definition is the mathematical definition of Round Robin. In practice, we divide all jobs in arbitrarily small pieces and we use a priority queue where we cycle through and run equal each job. It is widely accepted that if all pieces are small enough the above definition can be implemented in practice. A question that arises here is; is there a trade-off between the number of preemptions and the length of small pieces. Actually, there is but there exist other algorithms that minimize the number of preemptions, the Geometric Algorithms. In this project, we are free to use as many preemptions we want. Before starting analyzing the performance of the algorithm we should first give some preliminaries definitions.

Definition 2.1.2. $D_{i,j}^{A}$ is the time allocated by algorithm A to job J_i before job J_j completes. In other words it is the amount of time by which job J_i delays job J_j.

The above definition is very useful in understanding the objective of Round Robin. As all jobs receive equal time, at the time instance equal to smallest current job all active jobs have received equal amount of time. As a result, from all active jobs the smallest one is finishing first each time. So, the $D_{i,j}^{RR}$ for $p_i \leq p_j$ equals to p_i and $D_{j,i}^{RR} = p_i$. Using this argument inductively we see that the smallest job delays all the n-1 bigger ones, the second smallest job all the n-2 bigger etc.

Definition 2.1.3. The pairwise delay $P_{i,j}^{A} = D_{i,j}^{A} + D_{j,i}^{A}$ is the amount by which two jobs J_i and J_j delay each other.

Remark. $\sum_{i=1}^{n} C_i = \sum_{i=1}^{n} p_i + \sum_{1 \leq i < j \leq n} P_{i,j}^{A}$

Using the above remark we can derive that

$$min\{P_{i,j}^{A}\} = min\{p_i, p_j\}$$

As a result OPT's objective equals

$$OPT = \sum_{i=1}^{n} p_i + \sum_{i=1}^{n} (n-i)p_i$$

$$OPT = \sum_{i=1}^{n}(n-i+1)p_i$$

On the other side, for Round Robin is true that

$$P_{i,j}^{A} = 2 \times min\{p_i, p_j\} \implies$$

$$RR = (\sum_{i=1}^{n} p_i)_{RR} = \sum_{i=1}^{n} p_i + 2\sum_{i=1}^{n}(n-i)p_i$$

Combining the above results our aim is to maximize the ratio $\frac{RR}{OPT}$

$$\frac{RR}{OPT} = \frac{\sum_{i=1}^{n} p_i + 2\sum_{i=1}^{n}(n-i)p_i}{\sum_{i=1}^{n} p_i + \sum_{i=1}^{n}(n-i)p_i}$$

Now, we are going to prove a useful lemma that allows us to show that the worst case instance is when all jobs have equal size. Intuitively, the worst case is when this fair policy does not worth the case that is all jobs are equal.

Lemma 2.1.4. *The solution of the below Linear Program (P) is* $p_i = p_j$ *,*$\forall i \neq j$.

$$\begin{aligned} \textit{max} \quad & \sum_{i=1}^{n}(n-i)p_i \\ \textit{subject to} \quad & \sum_{i=1}^{n} p_i \\ & p_i \leq p_j, \qquad i \leq j \\ & p_j \geq 1, \qquad j = 1, \ldots, n \end{aligned}$$

Proof. To prove by contradiction, we assume that $p_i < p_j$ for $i < j$. Then, we substitute $p_i \leftarrow p_i + \alpha$ and $p_i \leftarrow p_j - \alpha$ for sufficiently small $\alpha > 0$. There exists such an α because of the density property in the set of rational numbers. We see that all constraints are satisfied and we examine the new value of the objective function, lets say OPT'. We see that

$$OPT' = OPT + \alpha(j-i) > OPT$$

As a result, we derive that $p_i = p_j$,$\forall i \neq j$. So, if all p_i are equal , the above LP program is both feasible and bounded. The reason is that all constraints have been satisfied and the objective function is bounded by $p_i \sum_{i=1}^{n}(n-i)$ The maximum objective value is

$$\frac{(n-1)\sum_{i=1}^{n} p_i}{2}$$

□

Using the above lemma we can derive the approximation ratio of Round Robin. In fact, the question is; For a fixed sum of processing times what are the worst case values for p_i that maximize the objective of Round Robin?

Theorem 2.1.5. *The Round Robin (RR) algorithm is* $2(1-\frac{1}{n+1})$ *competitive.*

Proof. By using Lemma 6.1 and the fact that

$$\frac{RR}{OPT} = \frac{\sum_{i=1}^{n} p_i + 2\sum_{i=1}^{n}(n-i)p_i}{\sum_{i=1}^{n} p_i + \sum_{i=1}^{n}(n-i)p_i}$$

$$\frac{RR}{OPT} = 1 + \frac{\sum_{i=1}^{n}(n-i)p_i}{\sum_{i=1}^{n} p_i + \sum_{i=1}^{n}(n-i)p_i}$$

$$\frac{RR}{OPT} = 2 - \frac{\sum_{i=1}^{n} p_i}{\sum_{i=1}^{n} p_i + \sum_{i=1}^{n}(n-i)p_i}$$

In order to maximize $\frac{RR}{OPT}$ we need to minimize $\frac{\sum_{i=1}^{n} p_i}{\sum_{i=1}^{n} p_i + \sum_{i=1}^{n}(n-i)p_i}$ so we need to maximize $\sum_{i=1}^{n}(n-i)p_i$. By using Lemma 6.1 we see that

$$sup\{\frac{RR}{OPT}\} = 2 - \frac{\sum_{i=1}^{n} p_i}{(1+\frac{n-1}{2})\sum_{i=1}^{n} p_i}$$

$$sup\{\frac{RR}{OPT}\} = 2 - \frac{2}{n+1} = 2(1-\frac{1}{n+1})$$

□

2.1.2 Round Robin extension for multiple processors

In this section we are going to examine the extension of Round Robin Algorithm for multiple processors and derive a more detailed proof for its competitive ratio. We consider no communication costs for transferring one job from a certain processor to another. The invariant for Round Robin remains the fact that for every time t all active jobs have received the same amount of processing time. In practice, this setting is similar to before with one processor that is p times faster. The reason is; we do not examine the allocation of jobs to machines because we use only the invariant that there exists an allocation such that all jobs receive equal processing time and that migrations of jobs from one machine to another is free of cost. To start with, we have to define the performance ratio for a scheduling problem with *p* processors.

Definition 2.1.6. Let $C_A(p, J)$ denote the total completion time of an instance J when scheduled by algorithm *A* on p processors. The performance ratio of a non-clairvoyant algorithm for static scheduling is

$$R_A(p, n) = sup_{J:|J|=n}\frac{C_A(p, J)}{C_{OPT}(p, J)}$$

Firstly, we are going to examine the best offline algorithm for the problem. According to Conway et al. [2003], McNaughton [1959] and as Phillips et al. [1993] mentioned the optimal offline algorithm obeys the SRPT (Shortest Remaining Processing Time) rule. Therefore, let assume that $n > p$ and that $p_1 \geq p_2 \geq \cdots \geq p_n$, we sort the jobs in groups of cardinality equal to p and we assign one job to one processor. Notice that a group of higher index cannot finish before a group of a lower index. We can consider these groups as a single job in the single processor setting where the order of the groups follow the order of the jobs. That is, the smallest group will finish first, the largest at the end etc.

Definition 2.1.7. Let J be any instance of size n with the execution times of the jobs being such that $p_1 \geq p_2 \geq \cdots \geq p_n$. For $2 \leq k \leq \lceil \frac{n}{p} \rceil$, define the group G(k) of jobs to be

$$G(k) = \{J_i | (k-1)p < i < kp\}$$

All groups, except possibly the last one, consist of p consecutive jobs.

According to the SPT rule and the above definition we derive that

$$C_{OPT}(p, J) = \sum_{k=1}^{\lceil \frac{n}{p} \rceil} \sum_{J_i \in G(k)} kp_i$$

It remains to derive the completion time of each job in the Round Robin algorithm to compose $C_{RR}(p, J)$. To do this, we can construct a recursive relation that correlates two consecutive in size jobs.

Lemma 2.1.8. *For the Round Robin algorithm,* $C_i = C_{i+1} + (\frac{i}{p})(p_i - p_{i+1})$ *for* $i \geq p$ *while* $C_i = C_{i+1} + (p_i - p_{i+1})$ *for* $i < p$.

Proof. For $i \geq p$, at the completion time of J_{i+1} the rest active and remaining i jobs have received exactly p_{i+1} processing time. When J_i completes, an amount of $i(p_i - p_{i+1})$ has been done in p processors, so it has been elapsed $(\frac{i}{p})(p_i - p_{i+1})$ time units. It remains to study the case where $i < p$. In this case, every job is running alone on a single processor. When job p+1 finished every job such that $i < p$ has received p_{p+1} amount of time so it remains to run for $p_i - p_{p+1}$ to be finished. □

We notice that when J_{p+1} finishes we have already executed $\sum_{j=p+1}^{n} p_j$ amounts of workload in p processors so in $\frac{1}{p}\sum_{j=p+1}^{n} p_j$ time and the rest jobs with indices $i < p$ have received p_{p+1} amount of time each in p processors, in total $p\frac{1}{p}p_{p+1}$ time units. As a result, $C_{p+1} = \frac{1}{p}\sum_{j=p+1}^{n} p_j + x_{p+1}$. Using the above relation we derive that for $i < p$

$$C_i = p_i + \frac{1}{p}\sum_{j=p+1}^{n} p_j$$

For $i > p$ with similar arguments we try to compute C_{i+1}. Smaller jobs than J_{i+1} have run for $\frac{1}{p}\sum_{j=i+1}^{n} p_j$ time units, while larger jobs for $\frac{i}{p}p_{i+1}$ time, so $C_i = \frac{i}{p}p_i + \frac{1}{p}\sum_{j=i+1}^{n} p_j$. Using the above results, we are trying now to compute the objective function for RR. For RR, by summing up we have,

$$C_{RR} = \sum_{i=1}^{p} p_i + \sum_{i=p+1}^{n} \frac{2i-1}{p} p_i$$

Now we are going to define some new notation. Let a_i be the coefficient of p_i in RR objective and k_i in OPT. We see that for $i \leq p$ $a_i = k_i = 1$ and for $i > p$ $a_i \geq k_i$. As a result, we can write the ratio of the two objectives as :

$$\frac{C_{RR}}{C_{OPT}} = 1 + \frac{\sum_{i=p+1}^{n}(a_i - k_i)p_i}{C_{OPT}}$$

We see that our goal is to solve this Linear Program :

$$\begin{aligned}
\text{max} \quad & C_{OPT} = \sum_{k=1}^{\lceil \frac{n}{p} \rceil} \sum_{J_i \in G(k)} k p_i \\
\text{subject to} \quad & \sum_{i=p+1}^{n} (a_i - k_i) p_i = C \\
& p_i \geq p_j, && i \leq j \\
& a_i = k_j, && i \leq p \\
& a_i \geq k_j, && i \geq p \\
& p_j \geq 1, && j = 1, \ldots, n
\end{aligned}$$

Let consider jobs in the same group $G(k)$: J_i and J_j for $i > j$. Let assume $p_i > p_j$, then there exists an arbitrarily small and positive α such that if we replace p_j with $p_j + \frac{\alpha}{a_j - k}$ and p_i with $p_i - \frac{\alpha}{a_j - k}$ the constraints of the maximization problem are satisfied and the new objective $C'_{OPT} = C_{OPT} + k\alpha(\frac{1}{a_j - k} - \frac{1}{a_i - k})$ where $k\alpha(\frac{1}{a_j - k} - \frac{1}{a_i - k}) < 0$ because $i < j$ so $a_i < a_j$. We conclude that in the same group G_k all jobs that belong to this have the same size let this size be named y_k. Our new objectives become,

$$C_{OPT} = \sum_{k=1}^{\frac{n}{p}} p k y_k$$

$$C_{RR} = \sum_{k=1}^{\frac{n}{p}} p(2k-1) y_k$$

In the same spirit as before we write the ratio of the two objectives as

$$\frac{C_{RR}}{C_{OPT}} = 2 - \frac{\sum_{k=1}^{\frac{n}{p}} y_k}{\sum_{k=1}^{\frac{n}{p}} k y_k}$$

In order to maximize this ratio, we should minimize $\frac{\sum_{k=1}^{\frac{n}{p}} y_k}{\sum_{k=1}^{\frac{n}{p}} k y_k}$. So, we should maximize $\sum_{k=1}^{\frac{n}{p}} k y_k$ subject to a fixed $\sum_{k=1}^{\frac{n}{p}} y_k$ with $y_i \geq y_j$ for $i \leq j$. This the same linear program as in the single processor case where maximum is achieved when all y_i are equal. Finally, the competitive ratio is $2 - \frac{2p}{n+p}$.

2.1.3 Lower Bound for one and multiple processors

It remains to show that the competitive ratios we derived before match with some lowers bounds in order to be tight. We will start with the case where $p = 1$ (single machine).

Theorem 2.1.9. *For the static scheduling problem, every deterministic, non-clairvoyant algorithm has* $R_A(n) \geq 2 - \frac{2}{n+1}$.

Proof. Fix any deterministic algorithm A for 1 time unit and assume that no job finishes. We define e_i to be equal with the amount of time A spent executing job J_i. The adversary chooses as an instance J each J_i to have size $p_i = e_i + \epsilon$ for arbitrarily small and positive ϵ. Therefore, each job runs for a negligible amount of time to finish, so $\sum_i C_i = n$. From the other side, $C_{OPT} = \sum_{i=1}^{n}(n-i+1)p_i$, assumed that p_i are sorting in non-decreasing order ($p_1 \leq p_2 \leq \cdots \leq p_n$) and $\sum_{p_i} = 1$. Using the same argument we used in maximization step in Round Robin competitive ratio we find that $C_{OPT}(J) \leq \frac{n+1}{2}$, so $R_A(J) \geq \frac{n}{\frac{n+1}{2}} = 2 - \frac{2}{n+1}$. □

In fact, this lower bound can lead us to derive the Round Robin Algorithm as we find that all e_i are equal so an algorithm that matches this lower bound must give equal processing time to every job. We use same arguments to derive a lower bound for the case of multiple processors.

Theorem 2.1.10. *For the static scheduling on p processors, any deterministic, non-clairvoyant algorithm A has* $R_A(p,n) \geq 2 - \frac{2p}{n+p}$.

Proof. Same as before (single processor), fix any deterministic non-clairvoyant algorithm A and let it run for one unit of time assuming no job has finished yet. Let e_i the amount of processing that job J_i has received. The adversary chooses $p_i = e_i + \epsilon$, for arbitrarily small and positive ϵ. Ar a result, sum of completion times for RR is equal to n, since all jobs finish immediately after time 1. On the other side, the optimal clairvoyant algorithm is SPT :

$$C_{OPT}(p,J) = \sum_{k=1}^{\lceil \frac{n}{p} \rceil} \sum_{J_i \in G(k)} kp_i = \sum_{k=1}^{\lceil \frac{n}{p} \rceil} \sum_{J_i \in G(k)} ke_i + \hat{\epsilon}$$

where $\hat{\epsilon} \to 0$ s.t $e_i \geq 0$, $e_1 \leq e_2 \leq \cdots \leq e_n$ and $\sum_i e_i = p$. Here, we use the same argument as in maximization step when deriving the competitive ratio of RR for multiple machines. We conclude that C_{OPT} is maximized when each $e_i = \frac{p}{n}$ giving that $R_A(n,p) = 2 - \frac{2p}{n+p}$. □

In Phillips et al. [1993] they proved that even randomization cannot provide us a remarkably lower bound. On the other hand, the approximation ratio of the deterministic and the randomized algorithm is asymptotically the same using Yao [1977] results. Since this work, a lot of progress has become in scheduling problems where job sizes are considered to be unknown. Here are some examples; Fox et al. [2013], Lindermayr and Megow [2022], Garg et al. [2019] who studied non-clairvoyant scheduling under predictions; the online case of the problem; the same problem under precedence constraints. These modern approaches gave us the motivation to examine this setting using Uncertainty and Learning Augmented Algorithms which are the following chapters.

2.2 Online-Dynamic Non-Clairvoyant Scheduling

The previous results were applied to the static case of the problem where it assumed that all jobs have release time equal to zero. Now, the jobs are considered to have arbitrarily release times and the algorithm is not aware of their existence until their release. For the same objective function as before, that is $\sum C_i$ according to Moseley and Vardi [2022] Round Robin is a 4-approximation algorithm. More precisely, in Moseley and Vardi [2022] they minimized the l_p

norm of completion times vector, using potential functions techniques, and they proved that is equal to $2\sqrt[p]{2(2)^{p-1}}$. In Phillips et al. [1993] they consider a different objective function that is usually consider instead of the previous one in the dynamic aspect of the problem, the sum of Flow Times $\sum F_i$. We remind that for a job $\mathcal{J}_i$ the Flow time is $F_i = C_i - r_i$. We will show that $1|prmt, non-clair|\sum_i F_i$ is unbounded. To prove that we cite an example from Phillips et al. [1993]. Intuitively, the worst case scenario in the static case is when all jobs are equal, so we are going to try to derive an example where between any two consecutive release or completion times, all jobs remaining in the machine have equal remaining size.

Theorem 2.2.1. *For the dynamic scheduling problem of size n, Round Robin has performance ratio* $\Omega(\frac{n}{\log n})$.

Proof. To prove the above theorem, as we stated before, our aim is to construct an instance where Round Robin's objective is unbounded compared with the Optimal Objective. To start with the Optimal Algorithm. Using an Exchange Argument it is easily derived that Optimal Algorithm schedules the job with the shortest remaining processing time. A formal proof can be found there McNaughton [1959]. Now, we are ready to construct an instance. Our intuition is the static case where the worst case is when all jobs are equal. So, we are constructing an instance where between two consecutive release or completion times all active jobs have equal remaining processing time. More formally:

$$\text{Let } H_k \text{ denote the k-th Harmonic number, } H_k = \sum_{i=1}^{k} \frac{1}{i}$$

Consider the set of n jobs with the following release and execution times.

- At time $t = 0$, two jobs of length 1 are released.
- For $1 \leq k \leq n-2$, at time H_k, a job of length $\frac{1}{k+1}$ is released.

Let, firstly, consider the Optimal Algorithm that is Shortest Remaining Processing Time. During $0 \leq t \leq 1$ it will schedule one of the jobs of length 1. Between two consecutive release times the elapsed time is $H_{k+1} - H_k = \frac{1}{k+1}$, so for $t \geq 1$ it schedules the new arrived jobs of length $\frac{1}{k+1}$,that finish immediately before the next release. At time $H_{n-2} + \frac{1}{n-1}$ it schedules the last job of length 1. Now, to simplify the calculation we use the following useful identity :

$$\sum_{k=1}^{n} H_k = (n+1)H_n - n$$

So, by summing over release times and completion times we find :

- $\sum_{i=1}^{n} r_i = \sum_{k=1}^{n-2} H_k = (n-1)H_{n-2} - (n-2)$
- $\sum_{i=1}^{n} c_i = \sum_{k=1}^{n-1} H_k + (1 + H_{n-1}) = (n-1)H_{n-1} - (n-2)$
- $\sum_{i=1}^{n} F_i = 2H_{n-1} + 1$

Round Robin, on the other hand, does not complete any job until time $H_{n-1}+1$. To derive that, observe that when job of size $\frac{1}{k}$ enters the system, the k jobs that have already released require $\frac{1}{k}$ amount of time. It is an easy induction to derive this result. So, making the same calculations we find that :

- $\sum_{i=1}^{n} r_i = \sum_{k=1}^{n-2} H_k = (n-1)H_{n-2} - (n-2)$
- $\sum_{i=1}^{n} c_i = n + nH_{n-1}$
- $\sum_{i=1}^{n} F_i = 2n + H_{n-1} - 1$

Using the fact that $H(n) = \mathcal{O}(\log n)$ we derive that :

$$\sup\{\frac{RR}{SRPT}\} = \Omega(\frac{n}{\log n})$$

□

Chapter 3

Explorable Uncertainty in Scheduling

3.1 Introduction to the notion of Uncertainty

3.1.1 Motivation

Recent advances in distributing database systems came into surface mathematical models for problems where input values are not given in precise but lie in an interval where their exact value can be acquired at an extra cost. One of the first works in the Uncertainty field was that of Feder et al. [2000] and Olston and Widom [2000]. The initial motivation behind the problem is the following; imagine we have a distributed system, i.e. a central and some local databases, where local cached copies of databases files are used to support quick processing of queries at client sites. Each file, has a corresponding master copy in the central database. The updates over these files are made at a very high rate and as a result it is likely to arise differences between master files and their copies. Let assume that these files represent data that is a value of a real number. The clients store for each cached value an interval that is guaranteed to contain the master value. Moreover, it is possible for the users to apply a query to the central database to learn the exact value of data at a extra cost. The aim is to compute a function over the data under a certain amount of precision while minimizing the cost of the queries.

Apart from distributed databases , a variety of problems has been examined under the notion of Uncertainty. To give some examples, in Erlebach et al. [2008] and in Erlebach et al. [2015] they consider the Minimum Spanning Tree Under Uncertainty and other geometric problems on the Euclidean Plane where each point is not known exactly but in a region. Kahan [1991] considered an application from Air-traffic where the exact position of airplanes is not known exactly but it can be acquired in an extra cost. Actually, the work of Kahan [1991] was one of the first and most fundamental in the field. Moreover, in more recent works such as those of Chaplick et al. [2021] and Halldórsson and de Lima [2021] they examined the problem of shortest path under uncertainty. Except of problems that can be solved in polynomial time in their classic setting, *NP-Hard* problems, such as Knapsack and Travelling Salesman Problem, has been examined under the notion of Uncertainty. In the first category , that is problems with a known exact solution in polynomial time, the goal is to find a solution while minimizing queries. On the other hand, for *NP-Hard* problems the goal is to find an approximation algorithm whose ratio depend on the number of queries.

Now, moving to a more familiar area we will talk about applications of solving problems under Uncertainty in the sector of Scheduling. In the very first works in Scheduling under Un-

certainty, they provided a different motivation-applications in contrast with the previous work. To give an example, imagine a computer processor environment, then the test-query could be a code optimizer that takes unit time to process the program and possibly reduces its running time and returns its exact workload at the end. Another example is file transmission over a network. We can spend time before sending a file to compress it and reduce its size or send it uncompressed. The compression plays the role of the query because after that the exact size of the file is known. Finally, in the medicine environment, the test can represent a diagnosis. Before, subscribing a therapy to a patient, we can spend more time to extra clinical examinations to possibly reduce the period of the therapy with a more targeted one. This type of problems are a specific version to the previous one. Their difference is that , before, algorithm's output should be the value of a function under certain precision while minimizing the cost of queries. Now, the cost of the queries is incorporated to the objective value that should be minimized. The size of each task that needs to be scheduled, lie in an uncertainty interval and its size can only be reduced. The goal is to find a trade-off between the extra cost spent at tests-queries and their contribution in minimizing the objective function. Two relevant works that consider these types of problems are these of Dürr et al. [2018], Dürr et al. [2020] where they considered uniform testing times and they provided a 2-approximation algorithm for a single machine environment for minimizing sum of completion times.

In this section, we studied the problem of single machine scheduling under Uncertainty while minimizing the sum of the completion times of the jobs. Before the start of our analysis we will make a short parenthesis to give the derived results of the relevant work in the field such as Dürr et al. [2018] results. Moreover, we are going to introduce some notation needed for the following results.We denote that uniform instances are considered instances with uniform upper bound and extreme uniform instances where all processing times are considered to be equal to zero or to their upper bounds.

Dürr et al. [2018] Results		
competitive ratio	lower bound	upper bound
deterministic algorithms	1.8546	2
randomized algorithms	1.6257	1.7453
det. alg. on uniform instances	1.8546	1.9338
det. alg. on extreme uniform instances	1.8546	1.8668
det. alg. on extreme uniform inst. with $\bar{p} \approx 1.9896$	1.8546	1.8552

3.2 Problem Definition

Scheduling Under Explorable Uncertainty introduces a new model in Scheduling different than that we have seen in Non-Clairvoyant Scheduling. More precisely, every job is described by 3 values. To explain in more detail, the set of jobs is defined as $J = \{i : i \in \{1, 2, \ldots, n\}\}$ and the fist value of each job is its release time r_i and the other two is its test size t_i and an upper bound u_i on its processing time (p_i). If we run the job untested, it has size of u_i units, else we

first run its test with size t_i and when it has finished we learn its exact workload p_i such that $0 \leq p_i \leq u_i$. Our goal is to minimize the sum of completion times $\sum_i C_i$. Now, not only the scheduler has to decide the order of allocation of the jobs to machines but also to find a criterion of whether to do the test or not.

3.3 Explorable Uncertainty in Scheduling with Non-Uniform Testing Times

In this class of problems we follow the work of Albers and Eckl [2020] where it is assumed that testing times can take any real value. In Albers and Eckl [2020], they extend the formulation of the problem that Dürr et al. [2018] proposed where Uniform Testing Times is considered, that is to say that every test has equal size (usually all jobs' sizes are normalized to test size so $t_i = 1$, $\forall i$), to a more general model where tests lengths can take arbitrary values. Before describing any algorithm we have to find a criterion for applying a test to a job or not. A useful invariable, that the criterion must satisfy, is to achieve a constant multiplicative factor in the algorithmic size of each job. We refer that the algorithmic size is the length of a job for an algorithm $\mathcal{A}$ based on the selected criterion for testing. The main idea about that is to determine a threshold and compare the ratio $\frac{u_i}{t_i}$ with this threshold. It is not known if the optimal function of u_i, t_i for minimizing the algorithmic size of job J_i is this ratio and it remains an open question for future works. According to this remark, we can bound the algorithmic running time with the optimal one. More precisely, we present a proof from Albers and Eckl [2020] that shows that we get an algorithmic running time ϕ times larger than the optimal, where ϕ is the golden ratio. The mathematical constant ϕ is obtained by solving for α the equation $1 + \frac{1}{\alpha} = \alpha$. The left side stands for the multiplicative factor of the tested jobs to optimal ones and the right one for the untested. The size of the jobs in the optimal algorithm is the minimum between the upper bound and the test plus the real processing time. In the following lemma we will proof analytically how the above equation was derived. In practice, the left side represents the ratio between the optimal job size and the algorithmic job size when the algorithm tests this job. Now, we will give some notation that is necessary for the proof. Let T be the set of the tested jobs such that the jobs J_i where $\frac{u_i}{t_i} \geq \alpha$, α is our threshold and N the set of not tested jobs ($\frac{u_i}{t_i} < \alpha$). Last, let ρ_i be the optimal runtime for a single job J_i and p_i^A its algorithmic runtime.

Lemma 3.3.1. *1. For every tested job it is true :* $\forall j \in T : t_j \leq \rho_j, p_j \leq \rho_j$

2. For every tested job it is true : $\forall j \in T : p_j^A \leq (1 + \frac{1}{\alpha})\rho_j$

3. For every untested job it is true : $\forall j \in N : p_j^A \leq \alpha\rho_j$

Proof. 1. If $\rho_j = t_j + p_j \implies \rho_j \geq t_j$, else if $\rho_j = u_j \geq \alpha t_j > t_j$

2. If $\rho_j = t_j + p_j \implies \rho_j = p_j^A$ else $\rho_j = u_j$ and $p_j^A = t_j + p_j \leq \frac{u_j}{\alpha} + u_j = (1 + \frac{1}{\alpha})\rho_j$

3. If $\rho_j = u_j \implies \rho_j = p_j^A$ else $\rho_j = t_j + p_j \geq \frac{1}{\alpha}u_j \implies p_j^A \leq \alpha\rho_j$

□

In order to find the value of α we solve the second degree equation $\alpha = 1 + \frac{1}{\alpha}$ where we derived 2 solutions with the positive one to be equal with $\alpha = \frac{1+\sqrt{5}}{2} = \phi$, where $\phi \approx 1.6180$ is the golden ratio. According to the above lemma we derive that $p_j^A \leq \phi\rho_j$. Using the above result, as we are going to see in the following proofs we can use any known scheduling algorithm where jobs are completed in a non-decreasing order and multiply their ratio by ϕ to obtain an algorithm that solves $1|$ pmtn,non-clair.$|\sum_i C_i$ under Uncertainty.

3.3.1 Golden Round Robin

In this section we are going to examine static single machine scheduling where preemption is allowed under Uncertainty.Staying in the previous model every job comes in with two values: its test size t_i and its upper bound u_i. If we run the test for the job J_i then $p_i^A = t_i + p_i$ else $p_i^A = u_i$. We want to minimize $\sum_i C_i$. So, in other words, the problem is $1|$ pmtn,non-clair.$|\sum_i C_i$ under Uncertainty. The first algorithm we are going to examine is Golden Round Robin from Albers and Eckl [2020]. The intuition behind this algorithm is to use the above criterion for testing and get an approximation ratio equal to that of Round Robin Multiplied by ϕ. In Albers and Eckl [2020] they used a analysis different from that of Phillips et al. [1993] for the classic *Round Robin* algorithm and they derived a competitive ratio equal to 2ϕ. Our contribution is using similar arguments with Phillips et al. [1993] and deriving a competitive ratio equal to $2(1-\frac{1}{n})\phi$, where *n* is the number of jobs to be scheduled. One of the pros of this analysis is that it can be extended to multiple parallel machines setting.

Algorithm 3: Golden Round Robin

```
for all J_i ∈ [m] do
    if u_i / t_i ≥ φ then
        test job i
    end
    else
        run job i untested
    end
end
Apply Round Robin Algorithm /* At every time t all active jobs have
    received equal amount of processing time                    */
```

Golden Round Robin (Algorithm 3) in Albers and Eckl [2020] prove that this algorithm has 2ϕ approximation ratio, for which they prove that is asymptotically tight. We are going to derive a more detailed proof and show that the approximation ratio is exactly $2(1-\frac{1}{n})\phi$ and show that matches with the tight example they propose. Before we give the proof, we prove a trivial but useful lemma.

Lemma 3.3.2. *Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ a non-decreasing sequence and $\beta_1, \beta_2, \ldots, \beta_n$ a sequence of length n. Then $S = \sum_{i=1}^{n} \alpha_i\beta_i$ is minimized when $\beta_1, \beta_2, \ldots, \beta_n$ is in non-increasing order.*

Proof. By hypothesis it is true that $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_n$, $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$ and $S = \sum_{i=1}^{n} \alpha_i \beta_i$. If we change β_i with β_j with $i < j$ then a new sum S' results. If we consider the difference $S' - S = \alpha_i(\beta_j - \beta_i) + \alpha_j(\beta_i - \beta_j) = (\beta_j - \beta_i)(\alpha_i - \alpha_j) > 0$. □

By using the above lemma, we can claim that we can combine immediately the testing criterion with the Round Robin proof from Phillips et al. [1993] and get a $2(1 - \frac{1}{n})\phi$ competitive algorithm.

Theorem 3.3.3. *Golden Round Robin is* $2(1-\frac{1}{n})\phi$ *competitive for minimizing sum of completion times objective.*

Proof. First, we assume that $\rho_1 \leq \rho_2 \leq \cdots \leq \rho_n$ and let σ the permutation of the set $\{1, 2, \ldots, n\}$ such that job i (J_i) has optimal running time ρ_i and algorithmic running time $p^A_{\sigma(i)}$. Let ,also , $p^A_1 \leq p^A_2 \leq \cdots \leq p^A_n$. We notice that p^A_i does not correspond to ρ_i but to $\rho_{\sigma^{-1}(i)}$. So, OPT objective value is :

$$OPT = \sum_{i=1}^{n} (n - i + 1)\rho_i$$

ALG 1 objective value is :

$$GRR = 2\sum_{i=1}^{n} (n - i)p^A_i + \sum_{i=1}^{n} p^A_i$$

We notice that in Golden Round Robin as also in Round Robin the jobs are finishing in increasing order of sizes. This order is $\sigma\{1, 2, \ldots, n\}$. If we change that order to $\{1, 2, \ldots, n\}$ according to 3.3.2 the new objective is bigger than the previous one. That is to say :

$$GRR \leq 2 \sum_{\sigma\{1,2,\ldots,n\}} (n - i)p^A_i + \sum_{\sigma\{1,2,\ldots,n\}} p^A_i = \phi(2 \sum_{\{1,2,\ldots,n\}} (n - i)\rho_i + \sum_{\{1,2,\ldots,n\}} \rho_i)$$

From the above result we derive:

$$sup\{\frac{GRR}{OPT}\} \leq \phi \; sup\{\frac{RR}{OPT}\} = 2(1 - \frac{1}{n})\phi$$

□

Tightness example : Now, we present a tight example whose instance is the same as in Albers and Eckl [2020]. The difference is that our example is not asymptotically tight (the objective of two values are equal if we let *n* to tend to infinity). Let consider *n* jobs with $u_j = p_j = 1$ and $t_j = \frac{1}{\phi}$ for all jobs. Since, $\frac{u_j}{t_j} \geq \phi$ the algorithm tests every job and therefore it runs a round robin scheme on n jobs with runtime $p^A_j = 1 + \frac{1}{\phi} = \phi$. As all algorithmic job sizes are equal all jobs are finishing the same time instance and we receive that $GRR = n^2\phi$. $OPT = \frac{n^2}{2} + \frac{n}{2}$ as it does not test any job. The final ratio is :

$$\frac{GRR}{OPT} = \frac{n^2\phi}{\frac{n^2}{2} + \frac{n}{2}} = 2(1 - \frac{1}{n})\phi$$

3.3.2 Golden Round Robin for multiple machines

In this section we extend the previous model for multiple machines. In this section, our goal is to minimize P| pmtn,non-clair.| $\sum_i C_i$. In the same spirit as before, we propose the Golden Round Robin Algorithm which is an adaptation of *Golden Round Robin* for multiple machines, that is in every time instance all jobs have received equal amount of processing time. We will use the same idea as before because, as we referred to the previous chapter, the extension of Round Robin in multiple machines is similar with a single machine that runs at a higher rate. As a result it remains an algorithm that all jobs are finished in a non-decreasing order and we can use again the same argument as in the previous proof, that is reordering the completion times of each job in the objective.

Theorem 3.3.4. *Golden Round Robin for minimizing P| pmtn,non-clair.| $\sum_i C_i$ is $(2 - \frac{2p}{n+p})\phi$ competitive.*

Proof. The proof is the same as before. We have to show that OPT objective remains in the same form as Round Robin and ,also, we have to show that if we change the order of jobs' sizes in GRR objective we form a bigger objective value. As before, we assume that $\rho_1 \leq \rho_2 \leq \cdots \leq \rho_n$ and let σ the permutation of the set $\{1, 2, \ldots, n\}$ such that job i (J_i) has optimal running time ρ_i and algorithmic running time $p^A_{\sigma(i)}$. Let ,also , $p^A_1 \leq p^A_2 \leq \cdots \leq p^A_n$. We notice that p^A_i does not correspond to ρ_i but to $\rho_{\sigma^{-1}(i)}$.

$$C_{OPT}(p, J) = \sum_{k=1}^{\lceil \frac{n}{p} \rceil} \sum_{J_i \in G(k)} k\rho_i$$

$$C_{GRR} = \sum_{\sigma\{1,2,\ldots,n\}} p^A_i + \sum_{\sigma\{1,2,\ldots,n\}} \frac{2i-1}{p} p^A_i$$

We see that GRR's objective function coefficients are in non-decreasing order so if we sum in the same order as OPT, that is $\{1, 2, \ldots, n\}$, the objective is going to acquire a higher value.

$$C_{GRR} \leq \sum_{i=1}^{n} p^A_i + \sum_{i=1}^{n} \frac{2i-1}{p} p^A_i \leq \phi(\sum_{i=1}^{n} \rho_i + \sum_{i=1}^{n} \frac{2i-1}{p} \rho_i)$$

To sum up:

$$sup\{\frac{GRR}{OPT}\} \leq \phi(sup\{\frac{RR}{OPT}\}) = (2 - \frac{2p}{n+p})\phi$$

□

Tightness example We consider the same instance as before , that is $u_j = p_j = 1$ and $t_j = \frac{1}{\phi}$ for all jobs. As a result, Golden Round Robin will test every job and its objective will be the same as Round Robin with n equal to $1 + \frac{1}{\phi}$ jobs. On the other way, OPT does not test any

job and run in SPT with n equal jobs with size equal to 1. The reason is that the $sup\{\frac{RR}{OPT}\}$ is obtained when all jobs are equal. More precisely:

$$\frac{C_{GRR}}{C_{OPT}} = 2\phi - \frac{\sum_{k=1}^{n/p}(1+\frac{1}{\phi})}{\sum_{k=1}^{n/p} k}$$

$$\frac{C_{GRR}}{C_{OPT}} = 2\phi - \phi\frac{\frac{n}{p}}{\frac{1}{2}\frac{n}{p}(\frac{n}{p}+1)}$$

$$\frac{C_{GRR}}{C_{OPT}} = (2 - \frac{2p}{n+p})\phi$$

3.4 Minimizing the Makespan Under Uncertainty

Minimizing the Makespan (C_{max}) in a single machine without Uncertainty is a trivial problem. We can use any order of the jobs to obtain the optimal objective value. Although, under Uncertainty we can not solve the problem optimal using worst case analysis. The reason is that by using any deterministic criterion for testing an adversary $\mathcal{A}$ can force us to take always the wrong decision. We can use the threshold criterion (lemma 3.3.1) that we used before to obtain a ϕ approximation algorithm. In more detail, again we order the jobs in any order but every job can be ϕ times larger than in the optimal algorithm.

For multiple machines, in a non-preemptive environment Albers and Eckl [2021] proved a 3.1016 approximation ratio, while in preemptive setting a 2 approximation factor. In Albers and Eckl [2021] they proposed the *SBS* algorithm and instead of ϕ they compared the ratio $\frac{u_j}{t_j}$ with a more complex threshold that is a function of the number of the parallel machines *m*. When, testing times are all equal (uniform case) they proved a better result using a different threshold. Finally, we denote that the above results are true then all jobs are released from the beginning. The fully online version of the problem has not been examined yet.

3.5 Testing Models for Non-Clairvoyant Scheduling

Based on the previous model for Scheduling Under Uncertainty we consider a new model where a job J_i is released with only known the testing time t_i. If we test a job, at the end of the test, we learn the exact job size else we learn job size only when it has been completed. Before analyzing this model we consider an instance in order to estimate if we can achieve improvements in this direction. At first, we consider an instance where k of n jobs' tests are considered to have zero length and the rest infinity. We should examine if there exist an improvement to Round Robin approximation ratio.

3.5.1 Semi Clairvoyance - Round Robin Extension

In this section we are going to study a similar problem as before where we have $1|r_i = 0|\sum C_i$ and a set of n jobs $J = \{j_1, j_2, \ldots, j_n\}$ with the difference that we are now knowing the exact

processing time for k of the n jobs but not for the other $n-k$. The question is that if we can do something better than Round Robin. An answer is combining RR with SPT.

Suppose we have a set of 3 three jobs , let say $J = \{j_1, j_2, j_3\}$ with processing times 1,2,3 respectively. We know that job j_2 has a processing time of 2 but we do not the processing times for jobs 1 and 3.
The objective value for Round Robin schedule is

$$3 + 5 + 6 = 14$$

If we think about a different schedule which applies Round Robin until every job has received a processing time of at least 2, then schedules the known job j_2 and finally uses again RR for the rest.The objective value of this algorithm is

$$2 + 5 + 6 = 13$$

Let assume a set K with $K \subseteq J$, to be consisted of the jobs with the known processing times. Before starting the analysis we are going to repeat the following remarks from Phillips et al.

Algorithm 4: Round Robin in Semi-Clairvoyant Setting

$\kappa, n \in \mathbb{N}$,$J = \{i : j_i \in Jobs\}$,$K = \{i : j_i \in$ Known Jobs$\}$, $p_i \quad \forall i \in K$
  $arr[i] \leftarrow i \quad \forall i \in J$ `/* processing time that each job has received */`
$t \leftarrow 0$ `/* t = time */`
$F \leftarrow \emptyset$ `/* F is the set of finished jobs */`
Sort J **Sort** K **while** $J \setminus F \neq \emptyset$ **do**
  while $arr[i] \leq min_{j \in K}\{p_j\}$ **do**
    while *no job has finished* **do**
      Run Round Robin for all jobs $j \in J \setminus K$
    end
    Add finished jobs to F
  end
  Run the whole $min_{j \in K}\{p_j\}$ Update arr[i] Add $min_{j \in K}\{p_j\}$ to F Remove $min_{j \in K}\{p_j\}$ from K
end

[1993] and Albers and Eckl [2020].

Remark. $\sum_{i=1}^{n} C_i = \sum_{i=1}^{n} p_i + \sum_{1 \leq i < j \leq n} P_{i,j}^A$

Remark. $C_i = \sum_{k \in [n]} D_{i,k}^A$

To calculate the amount of time we may save, we should examine the quantity $P_{i,j}^A = D_{i,j}^A + D_{j,i}^A$ in both Round Robin Algorithm and in Algorithm 4.

Round Robin:

$$D_{i,j}^{A} = \begin{cases} p_i & \text{if } p_i \leq p_j, \\ p_j & \text{if } p_j \leq p_i \end{cases}$$

$$D_{j,i}^{A} = \begin{cases} p_i & \text{if } p_i \leq p_j, \\ p_j & \text{if } p_j \leq p_i \end{cases}$$

So, we see that $P_{i,j}^{RR} = 2 \times min\{p_i, p_j\}$ as Phillips et al. [1993].

Round Robin-Semi Clairvoyant: Let assume a job k that belongs in the set of known jobs:

$$D_{i,k}^{A} = \begin{cases} p_i & \text{if } p_i \leq p_k, \\ p_k & \text{if } p_k \leq p_i \end{cases}$$

$$D_{k,i}^{A} = \begin{cases} 0 & \text{if } p_i \leq p_k, \\ p_k & \text{if } p_k \leq p_i \end{cases}$$

For the rest jobs that belong to $J \setminus K$, that is the unknown ones, $P_{i,j}^{RR} = 2 \times min\{p_i, p_j\}$ is the same as in the classic Round Robin. As we see, in this setting, the new completion time of an unknown job i with $p_i \leq p_k$, C_k' is (we assume $p_1 \leq p_2 \leq \cdots \leq p_n$):

$$C_i' = C_i^{RR} - \sum_{\{k \in K : p_i \leq p_k\}} p_i$$

For the known jobs, their completion times remain the same as RR. As a consequence the worst case scenario is that $\{k \in K : p_i \leq p_k\} = \emptyset$. Then it means that we know the k smallest jobs and we do not achieve a better competitive ratio. On the other hand, the best case scenario is when we know the k biggest jobs and the new objective is :

$$RR' = RR - \sum_{i=2}^{k} \sum_{j=1}^{i-1} p_j$$

$$RR' = RR - \sum_{j=1}^{k-1} (k-j) p_j$$

Where it is true that (Phillips et al. [1993]):

$$RR = \sum_{i=1}^{n} p_i + 2 \sum_{i=1}^{n} (n-i) p_i$$

$$OPT = \sum_{i=1}^{n} (n-i+1) p_i$$

We seek for $sup\{\frac{RR'}{OPT}\}$

$$\frac{RR'}{OPT} = \frac{\sum_{i=1}^{n}(n-i+1)p_i}{\sum_{i=1}^{n}(n-i+1)p_i} + \frac{\sum_{i=1}^{n}(n-i)p_i - \sum_{i=1}^{k-1}(k-i)p_i}{\sum_{i=1}^{n}(n-i+1)p_i}$$

$$\frac{RR'}{OPT} = 1 + \frac{\sum_{i=1}^{k-1}(n-k)p_i + \sum_{i=k}^{n}(n-i)p_i}{\sum_{i=1}^{n}(n-i+1)p_i}$$

Where we can write the denominator of the fraction as

$$\sum_{i=1}^{n}(n-i+1)p_i = \sum_{i=1}^{k-1}(n-k)p_i + \sum_{i=k}^{n}(n-i)p_i + \sum_{i=1}^{k-1}(k-i+1)p_i + \sum_{i=k}^{n}p_i$$

If we want to maximize $\frac{RR'}{OPT}$ we should maximize $\frac{\sum_{i=1}^{k-1}(n-k)p_i+\sum_{i=k}^{n}(n-i)p_i}{\sum_{i=1}^{n}(n-i+1)p_i}$ and if we sumbolize $\sum_{i=1}^{k-1}(n-k)p_i + \sum_{i=k}^{n}(n-i)p_i = C$ we have to maximize

$$\frac{C}{C + \sum_{i=1}^{k-1}(k-i+1)p_i + \sum_{i=k}^{n}p_i}$$

So we need to minimize

$$\sum_{i=1}^{k-1}(k-i+1)p_i + \sum_{i=k}^{n}p_i$$

which takes its minimum value for $p_i = p_{min}\ \forall i \in$ [n]. Now, by deleting the term p_{min} from both nominator and denominator or assuming that $p_{min} = 1$ (every job is normalized to p_{min}), we derive that:

$$\frac{RR'}{OPT} = 1 + \frac{(n-k) + \frac{(n-k+1)(n-k)}{2}}{\frac{n(n+1)}{2}}$$

Where :

$$(n-k) + \frac{(n-k+1)(n-k)}{2} = (n-k) + \frac{1}{2}(n^2 - 2kn + k^2 + n - k) = \frac{3}{2}(n-k) + \frac{1}{2}(n-k)^2$$

Therefore:

$$\frac{RR'}{OPT} = 1 + \frac{3(n-k) + (n-k)^2}{n^2+n} \xrightarrow[n\to\infty]{} 2 + (\frac{k}{n})^2 - 2\frac{k}{n}$$

Where for $k = 0$ we have the same approximation ratio, asymptotically, as Round Robin and for $k = n$ we are optimal because in fact we do SPT. To conclude, in a worst case analysis we do not achieve any improvements. Maybe in different type of analyses such Average or Smooth Analysis we may get an improvement but in this project we examine the competitive ratios of the algorithms under the metric of competitive analysis. As a result, we try in the next section to derive a parameterized on the sum of the sizes of all tests competitive ratio using ideas from the modern field of Learning Augmented Algorithms.

Chapter 4

Deriving Robust and Consistent Algorithms

4.1 Introduction

In this section we are going to combine a robust and a consistent algorithm in order to solve scheduling on a single machine. More precisely, we are trying to solve $1|pmtn|\sum_i C_i$ by using tests from which we learn the exact processing time of each job. The motivation of testing is that it is similar to a code optimizer or a file compression script that can possibly shorten the processing time of a job. Due to this, we want to combine a robust Algorithm like Round Robin (RR) , that has an approximation ratio of $2(1-\frac{1}{n+1})$ where n is the number of jobs and a consistent algorithm that we are going to describe in detail. When testing has competitive performance , that is error is small, the consistent algorithm performs much faster than the Round Robin Algorithm.

4.2 Motivation

As we saw in the previous chapters, in the majority of online algorithms we deal with Uncertainty. To give an example from Scheduling we may not know the release times or jobs' sizes. What we do to overcome this obstacle is to design algorithms that perform well in all possible instances of the problem and probably guarantee an nearly optimal solution in comparison with the best offline algorithm. From another point of view, in machine learning they use a huge amount of data samples in order to learn a distribution that generates these data and then predict the future while guaranteeing a small generalization error. Following the classic work of Lykouris and Vassilvtiskii [2018] in the field of Learning Augmented Algorithms we studied how to use ML predictions as an oracle and combine them with a robust online algorithm in order to obtain an algorithm that performs near optimal when predictions are accurate while staying robust when predictions are arbitrarily bad. We are concerned about robustness because it is shown in practice that in many real world application some time ML models do not have small generalization error.

In the new setting of Learning Augmented algorithms some famous algorithmic problems have been studied such as online caching (Jiang et al. [2020] , Lykouris and Vassilvtiskii [2018], Rohatgi [2020]), ski rental (Purohit et al. [2018]), scheduling (Im et al. [2021]), load balancing (Lattanzi et al. [2020]), secretary problem (Antoniadis et al. [2020b]), metrical task systems (Antoniadis et al. [2020a]), set cover (Bamas et al. [2020b]), flow and matching (Lavastida et al. [2020]), bin packing (Angelopoulos et al. [2021]) etc. In every problem, an important metric in the analysis is the error. For example, in non-clairvoyant scheduling the ML model observes

some features of a job and predicts its size so an naive measure for the error is the absolute value of the difference between the prediction and the real measure. Although, the choice of the measure , as Im et al. [2021] analyses, plays a very important role for the design and the analysis of the algorithm.

In the majority of the works on Learning Augmented Algorithms they consider an ML model as a black-box oracle where no assumption is made for the distribution of the error. More specifically, in Scheduling problems, the prediction is defined not to have extra time cost and it can be used immediately as a job arrives. Although, in practice, time needed for the prediction may not be negligible in relation to job size or the prediction may be a test script , like a code optimizer or a file compressor, that can gives away the exact size of the job in contrast with the ML predictions which may be inaccurate. To summarize, we study the trade-off between exact predictions that have time cost and postpone the completion of the jobs and ML predictions that are free in terms of time but may be inaccurate.

4.3 Model-Notation

As we have said before, the problem we want to solve is $1|pmtn|\sum_i C_i$. Every job j comes with its testing time t_j. Initially, we work in the non-clairvoyant setting and we do not know the exact processing time of every job. We can decide if we are going to run the test t_j and after its completion we learn the exact processing time of the job that is notated as p_j. Moreover, we assume that for every job j is true that $p_j \neq 0$ and we normalize all jobs to the smallest one so that $p_{min} = 1$. Finally, the partial (η_j) and total error η as defined in Purohit et al. [2018] in our case is

$$\eta_j = |t_j + p_j - p_j|$$

$$\eta = \sum_j \eta_j = \sum_j t_j$$

To avoid confusion of η with n we symbolise $T := \sum_j t_j$.

4.4 Algorithm

In this algorithm we run in parallel the tests and the jobs in a Round Robin fashion. This is equivalent to splitting the machine into two machines with adjacent velocities, that is to say the two new velocities sum up to one of the initial machine. We state that we run all the tests since there is not any criterion to select or not the execution of a test. For example, if we do not test any job, the adversary can choose this job to be the biggest one. When Testing is finished, we continue in executing in a SRPT fashion the remaining active jobs.

4.5 Analysis

Before starting analyzing the algorithm, it is easily understood that if the tests are small compared with the jobs, SRPT will prevail to Round Robin and the algorithm will perform near optimum.

Algorithm 5: Combining Round Robin with Testing

$n \in \mathbb{N}, (t_i, p_i) \forall i \in [n]$, $\lambda \in [0,1]$ $T \leftarrow \sum_{i=1}^{n} t_i$ $t \leftarrow 0$ `/* t → time */`

$F \leftarrow \emptyset$ `/* F is the set of finished jobs */`

while $t \leq \frac{T}{\lambda}$ **do**

  `/* In` $\frac{T}{\lambda}$ `time all tests have finished. */`

  Run Round Robin at $1-\lambda$ rate. Run the Tests in SPT at λ rate. For every finished job add it to F.

end

if $\ni$ *job* $\in F^c$ **then**

  `/*` $\frac{T}{(1-\lambda)n} \leq 1$ `*/`

  Run the remaining p_i in SRPT.

end

On the other hand, if the tests are arbitrarily large, Round Robin will execute the jobs later up to a multiplicative factor inversely proportional with the velocity we use in Round Robin machine. Intuitively, the worst case is that the adversary choose the tests almost equal to the jobs' sizes so as to not get a significant advantage while running SRPT. Now, we will continue with a more mathematical analysis of the algorithm.

Without loss of generality we assume that

$$1 = p_1 \leq p_2 \leq ... \leq p_n$$

Before we continue to evaluation of the new objectives, we first find how long the execution of the jobs is delayed due to the tests. In the following lemmas, we accurately compute jobs lengths when tests have all finished.

Lemma 4.5.1. *When Round Robin algorithm runs at* $0 \leq 1-\lambda \leq 1$ *rate, it has an approximation ratio of* $\frac{2-\frac{2}{n+1}}{1-\lambda}$ *for the objective of* $\sum_j C_j$.

Proof. Round Robin is run at rate $1-\lambda$ so the new completion times are $C'_j = \frac{C_j}{1-\lambda} \implies \sum_j C'_j = \frac{1}{1-\lambda} \sum_j C_j \leq \frac{2-\frac{2}{n+1}}{1-\lambda} OPT$. □

Lemma 4.5.2. *If no job* j *,in Algorithm 1, has finished until time* $t = \frac{T}{\lambda}$ *then its remaining processing time is* $p_i - \frac{T}{\lambda}\frac{1-\lambda}{n}$.

Proof. Tests finish at time $t = \frac{T}{\lambda}$. Round Robin, until that time, has processed a total load of $\frac{T}{\lambda} \times (1-\lambda)$ so each job has been eliminated by $\frac{T}{\lambda}\frac{1-\lambda}{n}$ load units because each job has received equal processing time. □

Lemma 4.5.3. *If no job has finished in Round Robin schedule until all tests have finished then* $\frac{T}{\lambda} \times (1-\lambda) \leq \sum_i p_i$.

Proof. As we showed before Round Robin has processed a total load of $\frac{T}{\lambda} \times (1-\lambda)$. As a result the following results hold:

1. $\frac{T}{n}\frac{1-\lambda}{\lambda} \leq p_i \ ,\forall i \in [n]$
2. $\frac{T}{\lambda} \times (1-\lambda) \leq \sum_{i=1}^{n} p_i \leq OPT$

□

Using the above results, we are ready now to calculate the objective of the sum of completion times parametrically in term of the sum of the tests T.

Theorem 4.5.4. *Algorithm 5 has approximation sum of completion times equal to* $T\{\frac{n}{\lambda} - \frac{1-\lambda}{\lambda}\frac{(n+1)}{2}\} + OPT$ *when no job has finished until* $t = \frac{T}{\lambda}$.

Proof. Sum of C_j for Algorithm 1 is :

$$ALG = n\frac{T}{\lambda} + \sum_{j=1}^{n}(n-i+1)(p_i - \frac{1-\lambda}{\lambda}\frac{T}{n})$$

$$= n\frac{T}{\lambda} - (\frac{1-\lambda}{\lambda}\frac{T}{n})\sum_{j=1}^{n}(n-i+1) + \sum_{j=1}^{n}(n-i+1)p_i$$

$$= n\frac{T}{\lambda} - (\frac{1-\lambda}{\lambda}\frac{T}{n})\frac{n(n+1)}{2} + OPT$$

$$= n\frac{T}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{T(n+1)}{2} + OPT$$

$$= T\{\frac{n}{\lambda} - \frac{1-\lambda}{\lambda}\frac{(n+1)}{2}\} + OPT$$

□

Lemma 4.5.5. $OPT \geq \frac{n(n+1)}{2}$

Proof. The proof is trivial by using the assumption that $p_{min} = 1$ and $p_i \geq p_{min} \ ,\forall i \in [n]$. The equality holds when all jobs are equal. □

Using the above theorem and the lemma we derive the approximation ratio of the algorithm with the term T inside.

Theorem 4.5.6. *The approximation ratio of Algorithm 5 is* $\frac{T(\frac{n}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{(n+1)}{2})}{\frac{n(n+1)}{2}} + 1$.

Proof. As we have showed in 4.5.4 and in 4.5.5:

$$ALG = T(\frac{n}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{(n+1)}{2}) + OPT \implies$$

$$\frac{ALG}{OPT} = \frac{T(\frac{n}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{(n+1)}{2})}{OPT} + 1$$

$$\leq \frac{T(\frac{n}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{(n+1)}{2})}{\frac{n(n+1)}{2}} + 1$$

□

From 4.5.6 we can easily derive that when $T = \mathcal{O}(1)$ or $T = \mathcal{O}(\sqrt{n})$ by taking a limit of $n \to \infty$ that $\frac{ALG}{OPT} \to 1$.
When no job has finished ,as we derived before, it is true for every job i that $\frac{1-\lambda}{\lambda}\frac{T}{n} \leq p_i$. For $p_i = p_{min} = 1$ we derive that $T \leq n\frac{\lambda}{1-\lambda}$. If we use this bound to 4.5.6 the result is :

$$\frac{ALG}{OPT} \leq n\frac{\lambda}{1-\lambda}\{\frac{(\frac{n}{\lambda} - (\frac{1-\lambda}{\lambda})\frac{(n+1)}{2})}{\frac{n(n+1)}{2}}\} + 1$$

$$\frac{ALG}{OPT} \leq \frac{\frac{n^2}{1-\lambda} - \frac{n(n+1)}{2}}{\frac{n(n+1)}{2}} + 1$$

We see that this fraction is a monotonic function in n so by taking limits to $n \to \infty$ we derive that :

$$\frac{ALG}{OPT} \leq \frac{2}{1-\lambda}$$

Now, we will use again the same lemma as in Round Robin proof in 2 to show that in worst case the adversary permits the SRPT to run for negligibly small time.

Lemma 4.5.7. *The solution of the below Linear Program (P) is .*

$$\begin{aligned}
&\textit{max} && \sum_{i=1}^{n}(n-i)p_i \\
&\textit{subject to} && \sum_{i=1}^{n} p_i \\
& && p_i \leq p_j, \qquad i \leq j \\
& && p_j \geq 1, \qquad j = 1, \ldots, n
\end{aligned}$$

Proof. To prove by contradiction, we assume that $p_i < p_j$ for $i < j$. Then, we substitute $p_i \leftarrow p_i + \alpha$ and $p_i \leftarrow p_j - \alpha$ for sufficiently small $\alpha > 0$. There exists such an α because of the density property in the set of rational numbers. We see that all constraints are satisfied and we examine the new value of the objective function, lets say OPT'. We see that

$$OPT' = OPT + \alpha(j - i) > OPT$$

As a result, we derive that $p_i = p_j$,$\forall i \neq j$. So, if all p_i are equal , it is trivial that the above LP program is both feasible and bounded. The maximum objective value is

$$\frac{(n-1)\sum_{i=1}^{n} p_i}{2}$$

□

Lemma 4.5.8. *Algorithm 5 is at most $\frac{2}{1-\lambda}$ competitive.*

Proof. From 4.5.4 we know that :

$$ALG = T\frac{n}{\lambda} - T\frac{1-\lambda}{\lambda}\frac{n+1}{2} + OPT \implies$$

$$\frac{ALG}{OPT} = \frac{T\frac{n}{\lambda}}{OPT} - \frac{T\frac{1-\lambda}{\lambda}\frac{n+1}{2}}{OPT} + 1$$

So we need to maximize :

$$\frac{T\frac{n}{\lambda}}{OPT} - \frac{T\frac{1-\lambda}{\lambda}\frac{n+1}{2}}{OPT} + 1$$

Since no job has finished until time $t = \frac{T}{\lambda}$, it is true that every p_i has a load bigger than the amount that Round Robin has processed for this specific job until there.

$$p_i \geq \frac{1-\lambda}{\lambda}\frac{T}{n} \forall i$$

so we derive that (for $i = 1$):

$$OPT \geq \frac{n(n+1)}{2}\frac{1-\lambda}{\lambda}\frac{T}{n} = T\frac{n+1}{2}\frac{1-\lambda}{\lambda} \implies$$

$$T \leq \frac{\lambda}{1-\lambda}\frac{2}{n+1}OPT \implies$$

$$T\frac{\frac{n}{\lambda}}{OPT} \leq \frac{2}{1-\lambda}\frac{n}{n+1} \leq \frac{2}{1-\lambda}$$

The term $\frac{T\frac{1-\lambda}{\lambda}\frac{n+1}{2}}{OPT}$ need to be minimized. We know that $\sum_{i=1}^{n} p_i \geq T\frac{1-\lambda}{\lambda}$ because Round Robin has run less load than the total one.

$$\frac{T\frac{1-\lambda}{\lambda}\frac{n+1}{2}}{OPT} \geq \frac{\frac{n+1}{2}\sum_{i=1}^{n} p_i}{OPT}$$

where we have to maximize OPT subject to a fixed $\sum_{i=1}^{n} p_i$, where we have proved in 4.5.7 (or Phillips et al. [1993] analysis)that this happens for $OPT = \frac{n+1}{2}\sum_{i=1}^{n} p_i$ and so $\frac{T\frac{1-\lambda}{\lambda}\frac{n+1}{2}}{OPT} \geq 1 \implies -\frac{T\frac{1-\lambda}{\lambda}\frac{n+1}{2}}{OPT} \leq -1$ Finally, from the above results we see that

$$\frac{ALG}{OPT} \leq \frac{2}{1-\lambda}$$

□

We need to show an instance of an algorithm that achieves this ratio. The above instance achieves this ratio : $p_i = p_j, \forall i \neq j$ and $p_i = \frac{1-\lambda}{\lambda}\frac{T}{n} + \epsilon$ for sufficiently small ϵ.

To conclude, we found the approximation expressed with the parameter T and derived that the worst case is when tests are completed almost near with Round Robin. As a result we cannot derive theoretically the optimum value for λ. In fact, maybe by using a different type of analysis we maybe can derive a way to estimate λ, which is now depending from T. We ended up designing code experiments to draw an inference for the optimal value of λ.

Chapter 5

Experimental Results

5.1 Introduction

As we saw in the previous chapter, we did not find a way to estimate the parameter λ for algorithm 5. In fact, λ is a function of the other parameter T. A common way to solve this problem is to design experiments where we constructed instances and compare the optimal offline algorithm *SPT*, the robust algorithm *Round Robin* and the robust and consistent algorithm 5 we designed in chapter 4. The aim of the experiments is to analyze the performance of these algorithms in a variety of instances and not only in the worst case instance. In practise, this is a good technique to compare algorithms as worst case instances do not come up frequently. Moreover, this is a convenient way to represent the performance of an algorithm in industry in different audiences that may not consist necessarily of computer scientists. Finally, we concluded that in order to estimate theoretically λ we should add extra assumptions to the problem. By adding extra assumptions we reduce the applicability of the problem and make it more specified. In the rest of this section, we present the hypothesises tests, the graphical representation of the results and we add comments on them.

5.2 Design of the experiments

5.2.1 Hypothesis tests

As we described in the introduction before, the tested parameter in these experiments is the value of λ which is a normalized parameter that denotes the processing rate of SPT in compare with the *Robust and Consistent* algorithm 5. This parameter can be imagined as a weight in the processing rate. Imagine that you are practitioners or software engineers and you have to decide in which algorithm you should give more importance. If you believe that the tests-code optimizers you have developed are fair enough then ,of course, you should apply testing at a higher rate than *Round Robin*. As there is little data available about the performance of code optimizers we made the assumption that tests' sizes cannot be arbitrarily large or tent to infinity because this is not the case of their usage. As a consequence, we made the hypothesis that their size lie in the same interval as job sizes. The second hypothesis we used is that their sizes are distributed uniformly in this interval. The uniform distribution is the simplest distribution when having no data. On the other hand, we believe that is not fair to use ,for example, a mix of Gaussian distributions as without any data available the estimation of their parameters (mean value and deviation) would

be completely random. To summarize, we tried to take into account the simplest hypothesises available.
On the other hand, for the estimation of the parameter we tried a wide spectrum of values of λ in the range of the open interval $(0, 1)$. As values of tests came from a distribution, we have run the experiment for a certain value of λ for many times and then we took the mean value of the results. In this way, we filter the dependence of the derived objectives to certain instances obtained from sampling the uniform distribution. More specifically, we tried 7 candidate values of λ , the multiples of 0.125, due to the fact that every experiment takes a huge amount of time to be executed when using a CPU and there is no tremendous difference in the derived results.

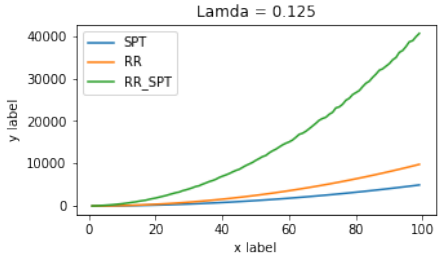
5.2.2 Technical details about the code section

The code for experimental results is written in *Python* language for simplicity in representing the graphical results. We denote that there is no different by using a more "fast" language ,for example *C++*, as we computed algebraically the objective function for every given instance. There are many reasons to select this choice. For example, computing the objective algebraically does not depend on implementation choices for Round Robin such as the priority queue and the processing quantum that every job is divided in. We implemented all the objective function on our own and we tested them on a class of instances before we used them in the experiments. The code was written in the *Google Colab* environment so as it was easier for cross checks and sharing.
For implementing the uniform distribution we used the *numpy random* library where it is implemented a pseudo-random numbers generator. Thanks to its Cryptographic implementation no one can distinguish the difference between numbers come from *numpy random* function and from a real uniform distribution. For completeness we run the experiments many times and we took the mean value in order to derive an independent of certain samples result. We set the *seed* of this function equal to 17 (it is not false to select any other number) for debugging. We denote that the hardest computational part of these experiments is the sampling as the other algorithms are implemented in linear time.

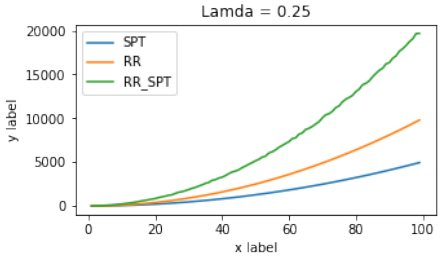
5.3 Results

In the following figures we present the performance of the above three algorithms in comparison with the numbers of the jobs in the instance. It is clear that for higher values of λ the algorithm 5 performs near optimal. The reason is that by using worst case analysis we may derive a competitive ratio that is only achieved by a single instance that rarely happens in practise. A famous example of the above argument is Simplex Algorithm for solving Linear Programs where in worst case Simplex has exponential complexity but in practise it runs in polynomial time. We are going to analyze in more detail every graph. We denote that *y-axis* represents the value of the objective function, that is sum of completion times of every job and *x-axis* the cardinality of the set that contains the jobs to be scheduled.

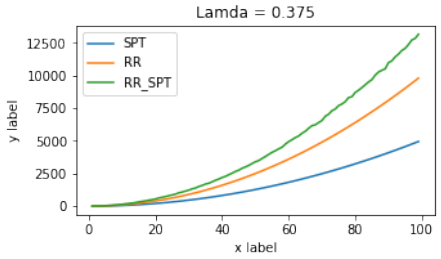
- $\lambda = 0.125$: For this value of λ it is clear that we give more importance to *Round Robin* Algorithm. As a result, it is more possible that by the completion of the tests the majority



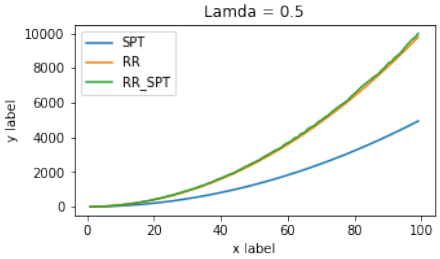
(a) $\lambda = 0.125$



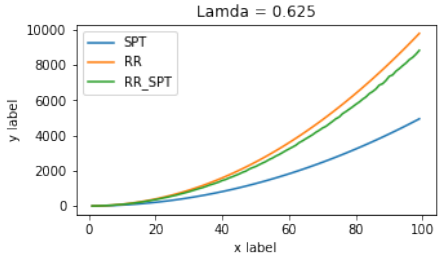
(b) $\lambda = 0.250$



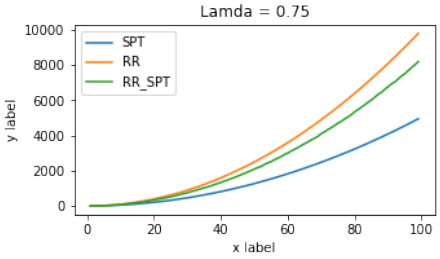
(c) $\lambda = 0.375$



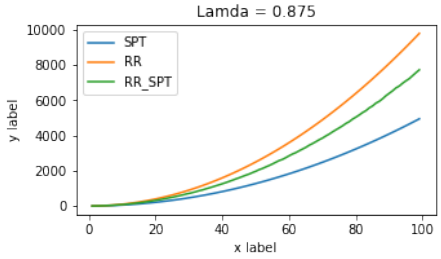
(d) $\lambda = 0.500$



(e) $\lambda = 0.625$



(f) $\lambda = 0.750$



(g) $\lambda = 0.875$

Figure 5.1: Experimental analysis of algorithm 5 within the different values of λ.

of the jobs has been completed and the tests had a negative contribution by postponing the completion of the jobs.

- $\lambda = 0.250$ and $\lambda = 0.375$: We see that by increasing the value of λ the performance of algorithm 5 tends to reach that of *Round Robin*.
- $\lambda = 0.500$: We see that when we give equal processing weight to both algorithms they have almost the same performance. Running in *SPT* fashion , after the completion of the tests, can counterbalance the loss from executing the tests in parallel at the beginning.
- $\lambda > 0.500$: For values of λ higher than a half we see that algorithm 5 performs better than *Round Robin* because in practise tests tend to have a reasonable size and after their completion the performance of the algorithm earns a lot by running in SPT fashion.

Bibliography

Susanne Albers and Alexander Eckl. Explorable uncertainty in scheduling with non-uniform testing times. *arXiv preprint arXiv:2009.13316*, 2020.

Susanne Albers and Alexander Eckl. Scheduling with testing on multiple identical parallel machines. In *Workshop on Algorithms and Data Structures*, pages 29–42. Springer, 2021.

Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ml predictions for online algorithms. In *International Conference on Machine Learning*, pages 303–313. PMLR, 2020.

Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. Online bin packing with predictions. *arXiv preprint arXiv:2102.03311*, 2021.

Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, pages 345–355. PMLR, 2020a.

Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. *Advances in Neural Information Processing Systems*, 33:7933–7944, 2020b.

Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. *Advances in Neural Information Processing Systems*, 33:15350–15359, 2020a.

Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. *Advances in Neural Information Processing Systems*, 33:20083–20094, 2020b.

Evripidis Bampis, Konstantinos Dogeas, Alexander Kononov, Giorgio Lucarelli, and Fanny Pascual. Speed scaling with explorable uncertainty. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 83–93, 2021.

Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1):1–39, 2007.

Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

Joan Boyar, Kim S. Larsen, and Abyayananda Maiti. A comparison of performance measures via online search. *Theoretical Computer Science*, 532:2–13, 2014. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2013.07.022. URL https://www.sciencedirect.com/science/article/pii/S0304397513005380. Frontiers in Algorithmics.

J. Błażewicz, G. Finke, R. Haupt, and G. Schmidt. New trends in machine scheduling. *European Journal of Operational Research*, 37(3):303–317, 1988. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(88)90192-0. URL https://www.sciencedirect.com/science/article/pii/0377221788901920.

Steven Chaplick, Magnús M Halldórsson, Murilo S de Lima, and Tigran Tonoyan. Query minimization under stochastic uncertainty. *Theoretical Computer Science*, 895:75–95, 2021.

Edward G Coffman and Ronald L Graham. Optimal scheduling for two-processor systems. *Acta informatica*, 1(3):200–213, 1972.

R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Dover Books on Computer Science Series. Dover, 2003. ISBN 9780486428178. URL https://books.google.gr/books?id=Yr5_kQDa_ssC.

Fotakis Dimitris, Jannik Matuschke, and Papadigenopoulos Orestis. Malleable scheduling beyond identical machines. *Journal Of Scheduling*, 2022.

Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. Scheduling with explorable uncertainty. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82(12):3630–3675, 2020.

Thomas Erlebach, Michael Hoffmann, Danny Krizanc, Matús Mihal'Ák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. *arXiv preprint arXiv:0802.2855*, 2008.

Thomas Erlebach, Michael Hoffmann, et al. Query-competitive algorithms for computing with uncertainty. *Bulletin of EATCS*, 2(116), 2015.

Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 602–607, 2000.

S. Feldmann, Jiri Sgall, and S.-H Teng. Dynamic scheduling on parallel machines. pages 111–120, 11 1991. doi: 10.1109/SFCS.1991.185355.

Dimitris Fotakis, Evangelia Gergatsouli, Themis Gouleakis, and Nikolas Patris. Learning augmented online facility location. *arXiv preprint arXiv:2107.08277*, 2021.

Dimitris Fotakis, Jannik Matuschke, and Orestis Papadigenopoulos. Malleable scheduling beyond identical machines. *Journal of Scheduling*, pages 1–18, 2022.

Kyle Fox, Sungjin Im, Janardhan Kulkarni, and Benjamin Moseley. Online non-clairvoyant scheduling to simultaneously minimize all convex functions. pages 142–157, 2013.

Naveen Garg, Anupam Gupta, Amit Kumar, and Sahil Singla. Non-clairvoyant precedence constrained scheduling. *arXiv preprint arXiv:1905.02133*, 2019.

Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *International Conference on Machine Learning*, pages 2319–2327. PMLR, 2019.

R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966. doi: 10.1002/j.1538-7305.1966.tb01709.x.

R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. 5:287–326, 1979. ISSN 0167-5060. doi: https://doi.org/10.1016/S0167-5060(08)70356-X. URL `https://www.sciencedirect.com/science/article/pii/S016750600870356X`.

Magnús M Halldórsson and Murilo Santos de Lima. Query-competitive sorting with uncertainty. *Theoretical Computer Science*, 867:50–67, 2021.

Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 285–294, 2021.

Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. *arXiv preprint arXiv:2006.09509*, 2020.

Simon Kahan. A model for data in motion. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of computing*, pages 265–277, 1991.

Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.

Thomas Lavastida, Benjamin Moseley, R Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. *arXiv preprint arXiv:2011.11743*, 2020.

Alexander Lindermayr and Nicole Megow. Non-clairvoyant scheduling with predictions revisited. *arXiv preprint arXiv:2202.10199*, 2022.

Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.

Robert McNaughton. Scheduling with deadlines and loss functions. *Management science*, 6(1): 1–12, 1959.

Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. *Advances in Neural Information Processing Systems*, 31, 2018.

Benjamin Moseley and Shai Vardi. The efficiency-fairness balance of round robin scheduling. *Operations Research Letters*, 50(1):20–27, 2022.

Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. Technical report, Stanford, 2000.

S Phillips, R Motwani, and E Torng. Non-clairvoyant scheduling. *Proc. 4th Annu. ACM-SIAM SODA*, pages 422–431, 1993.

Michael L Pinedo. *Scheduling*, volume 29. Springer, 2012.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1845. SIAM, 2020.

Tim Roughgarden. *Beyond the worst-case analysis of algorithms*. Cambridge University Press, 2021.

Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

David Shmoys, Joel Wein, and David Williamson. Scheduling parallel machines on-line. *Siam Journal on Computing - SIAMCOMP*, 24:131–140, 11 1991. doi: 10.1109/SFCS.1991.185361.

Daniel Sleator and Robert Tarjan. Amortized efficiency of list update rules. pages 488–492, 01 1984. doi: 10.1145/800057.808718.

Arjen PA Vestjens. *On-line machine scheduling*. Citeseer, 1997.

Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227. IEEE Computer Society, 1977.

Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th annual foundations of computer science*, pages 374–382. IEEE, 1995.

Appendix A

Code for Experimental Results

The code of the experimental results can be found in `https://github.com/SpyrosDragazis/SchedulingUnderExplorableUncertainty.git`.