ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογιστών

Αλγόριθμοι χρονοδρομολόγησης καθοδηγούμενοι από δεδομένα για την ελαχιστοποίηση του χρόνου ολοκλήρωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

Βασιλικής Ν. Ευθυμίου

Επιβλέπων: Δημήτριος Φωτάκης
Καθηγητής ΗΜΜΥ ΕΜΠ

Αθήνα, Ιούλιος 2024

National Technical University of Athens

School of Electrical and Computer Engineering

Division of Computer Science
Computation and Reasoning Laboratory

Data-driven scheduling algorithms for total completion time minimization

Diploma Thesis

Vasiliki N. Efthymiou

Advisor: Dimitris Fotakis
Professor ECE NTUA

Athens, July 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογιστών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

Βασιλικής Ν. Ευθυμίου

Επιβλέπων: Δημήτρης Φωτάκης
Καθηγητής ΗΜΜΥ Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 10η Ιουλίου 2024.

.....	.....	.....
Δημήτρης Φωτάκης	Αριστείδης Παγουρτζής	Χαρίκλεια Ποδηματά
Καθηγητής ΗΜΜΥ ΕΜΠ	Καθηγητής ΗΜΜΥ ΕΜΠ	Αν. Καθηγήτρια MIT SLOAN

Αθήνα, Ιούλιος 2024.

.....

Βασιλική Ν. Ευθυμίου

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

Περίληψη

Στην συγκεκριμένη εργασία εξετάζουμε το πρόβλημα της ελαχιστοποίησης του χρόνου ολοκλήρωσης εργασιών με άμεσες αποφάσεις ("promptness") όπως αυτό ορίστηκε στο [23]. Πιο συγκεκριμένα, εξετάζουμε το σενάριο όπου οι εργασίες φτάνουν σταδιακά (online) και ο αλγόριθμος καλείται να λάβει αποφάσεις για την σειρά εκτέλεσης τους άμεσα, λαμβάνοντας υπόψη χρονικούς περιορισμούς. Εμπνευσμένοι από την βιβλιογραφία του σχεδιασμού αλγορίθμων καθοδηγούμενων από δεδομένα (data-driven algorithm design, [26]), επιγκεντρωνόμαστε στην επέκταση της συγκεκριμένης δουλειάς των Alon , Feldman και Fiat εξετάζοντας το πρόβλημα πέραν της ανάλυσης χειρότερης περίπτωσης. Συγκεκριμένα, προτείνουμε την μοντελοποίηση του προβλήματος ως ένα πρόβλημα μάθησης και παρέχουμε έναν αποδοτικό αλγόριθμο για την επίλυση αυτού. Υιοθετώντας μια προσέγγιση παρόμοια με την εργασία [27], τελικά αποδεικνύουμε ότι ο προτεινόμενος online αλγορίθμος παρέχει και ένα τρόπο ένα λύσουμε το offline πρόβλημα με σταθερό λόγο προσέγγισης ενώ ο ίδιος ο online αλγόριθμος πετυχαίνει σταθερό c-regret.

Λέξεις Κλειδιά

Αλγόριθμοι, χρονοδρομολόγηση, χρόνος ολοκλήρωσης, προσεγγιστικοί αλγόριθμοι, αλγόριθμοι μάθησης

Abstract

In this work, we consider the problem of online prompt scheduling with a focus on minimizing the total completion time (TCP). We are inspired by the idea of the paper [23], in which they introduced a simple scheduling algorithm $\mathcal{A}(p)$: "Pick the first available slot that the process fits in p" where p is a fixed pattern that leads to a partitioning of the time horizon into a sequence ('pattern') of slots, whose lengths are increasing powers of 2. Following the literature on data-driven algorithm design [26], we raise the question: "Can we learn from data such that we design a better performing pattern"? In order to answer this question, we combine 'Online Learning' with Approximation Algorithms techniques. In particular, we adopt an approach similar to [27]. First, we present a suitable Integer Program and prove that solving its continuous relaxation then applying a proper rounding scheme yields a constant approximation of the for the offline regime. We then leverage this approximation using a learning algorithm to devise a pattern that will work better in the future input sequences.
To measure performance we use a variation of the standard notion of Regret (the so-called c-regret). Our proposed algorithm is proven to be no c-regret.

Key Words

algorithms, scheduling, total completion time, online algorithms, competitive ratio, online learning, regret

Ευχαριστίες

Καταρχήν ευχαριστώ τα μέλη της εξεταστικής επιτροπής για το χρόνο που αφιέρωσαν να αξιολογήσουν αυτή την εργασία.

Στριμώχνω σε μια παράγραφο το ευχαριστώ στον επιβλέποντα κύριο Φωτάκη. Ευχαριστώ για τα υπέροχα "Διακριτά Μαθηματικά" από τα οποία ξεκίνησαν όλα. Ευχαριστώ για τις απαντήσεις στα δεκάδες(*100) email, ειδικά στην καραντίνα. Ευχαριστώ για όλα τα μαθήματα σας. Ευχαριστώ για όλες σας τις εξηγήσεις. Ευχαριστώ για την ευκαιρία που μου δώσατε να κάνω διπλωματική μαζί σας. Νιώθω ευγνώμων που μου δόθηκε η ευκαιρία να είμαι μέρος αυτού του οράματος (που τουλάχιστον νιώθω) ότι πρεσβεύει το Corelab: ένα ανοικτό περιβάλλον συνεργασίας για την παροχή ευκαιριών σε όλους όσους ενδιαφέρονται για την Θεωρητική Πληροφορική.

Επιπλέον, θα ήθελα να ευχαριστήσω ιδιαιτέρως τρεις ακόμα καθηγητές: Τον καθηγητή Άρη Παγουρτζή για τα πρώτα μου μαθήματα στην Πληροφορική και την πάντα φιλική του διάθεση στο Corelab. Τον καθηγητή Ιωάννη Εμίρη για την ευκαιρία που μου έδωσε να κάνω γνωρίσω την Υπολογιστική Γεωμετρία και τον καθηγητή Α. Παπαβασιλείου για την εισαγωγή στην επιχειρησιακή έρευνα, τις συζητήσεις για την βελτιστοποίηση καθώς και τις συμβουλές του για τις μεταπτυχιακές σπουδές.

Ευχαριστώ πολύ πραγματικά όλες και όλους (“τα παιδιά") από το Corelab για την όμορφη ατμόσφαιρα που δημιουργήσαμε. Για καθένα και καθεμία από εσάς εύχομαι απλώς να σας είχα γνωρίσει νωρίτερα. Ευχαριστώ για τις γνώσεις, τις ιδέες, τις συζητήσεις, την παρέα. Όλοι και όλες με τον τρόπο τους υπήρξαν πηγή έμπνευσης και δημιουργικότητας. Θα χαρώ πολύ να ξαναβρεθούμε μελλοντικά .

Ευχαριστώ τους ανθρώπους με τους οποίους κάναμε παρέα στα αμφίθεατρα τα φυσικά ή τα ηλεκτρονικά, όταν η πανδημία μας απέκλεισε την επαφή. Τους ευχαριστώ για όλες τις συζητήσεις μας ακαδημαικές ή μη, τις εργασίες που συνεργαστήκαμε, τα άγχη και τα αστεία που μοιραστήκαμε. Ειδικότερα ευχαριστώ (σε τυχαία μετάθεση): Ολίνα, Δήμητρα Λ., Στέφανε, Δαμιανέ, Νικόλα, Δημήτρη Μ, Δήμητρη Π., Αλέξανδρε Μ., Ευθύμη , Κωνσταντίνε, Μαρία, Μήτσο. Επίσης ευχαριστώ και όλους τους άλλους/άλλες που είχα την τύχη να γνωρίσω και να αλληπιδράσω, αλλά δεν σας ξέρω τόσο ώστε να μην ντραπώ να γράψω το όνομα σας. Σας κρατώ στην μνήμη μου και θα χαρώ να σας συναντήσω ξανά.

Ευχαριστώ τις “εξωσχολικές" μου φίλες Έλενα, Χριστίνα, Δήμητρα Καρ., Ιωάννα και τις φίλες μου από τον Καμπούρη : Ναταλία, Ιωάννα και Δήμητρα Καρκ και ξανά τις Ολίνα και

Δήμητρα Λ.

Ευχαριστώ τους καθηγητές μου από την ΣΗΜΜΥ και τη ΣΕΜΦΕ και ειδικά εκείνους που με κόπο προσπάθησαν να μας παρέχουν μια πλήρη διδακτική εμπειρία και με υπομονή απαντούσαν απορίες. Αποτελούν έμπνευση για μένα. Ένα ευχαριστώ στον μαθηματικό Γιώργο Φραγκουλόπουλο, που έκανε τα μαθηματικά το πιο ασφαλές μου χώρο στον Λύκειο και που πίστευε σε εμένα όταν εγώ δεν πίστευα καθόλου.

Ευχαριστώ τον Άλκη *που είναι "ο Άλκης" βασικά..* και που από τότε που τον γνώρισα, εμπλουτίζει τη κάθε μου μέρα με τόσους διαφορετικούς και όμορφους τρόπους.

Ευχαριστώ τους γονείς μου και τον αδερφό Χάρη μου για την αγάπη που μου έδειξαν και μου δείχνουν αυτά τα τελευταία χρόνια και που με στήριξαν και ήταν υπομονετικοί μαζί μου, όταν πραγματικά χρειάστηκε. Τέλος ευχαριστώ την Μαρίτα, *που μου μίλαγε για φως, όταν το μόνο που έβλεπα ήταν σκοτάδι.*

-Βάλια.

Contents

Εκτεταμένη Ελληνική Περίληψη

Στο κεφάλαιο αυτό παρουσιάζουμε συνοπτικά τα αποτελέσματα της παρούσας διπλωματικής εργασίας. Αρχικά παρουσιάζεται το θεμελιώδες υπόβαθρο για το πρόβλημα της ελαχιστοποίησης του χρόνου ολοκλήρωσης. Έπειτα, ανατρέχουμε στο πρόβλημα της ελαχιστοποίησης του χρόνου ολοκλήρωσης με άμεσες αποφάσεις ("promptness") όπως αυτό ορίστηκε στο [23]. Αφού παρουσιάζουμε τα κύρια αποτελέσματα τους, προτείνουμε την μοντελοποίηση του προβλήματος ως ένα πρόβλημα μάθησης και παρέχουμε έναν αποδοτικό αλγόριθμο για την επίλυση αυτού. Πιο συγκεκριμένα, προτείνουμε έναν αλγόριθμο που συνδυάζει τεχνικές προσεγγιστικών αλγορίθμων με τεχνικές μάθησης. Για την αξιολόγηση του αλγορίθμου μας χρησιμοποιούμε την μετρική του c-Regret και αποδεικνύουμε ότι συμπεριφέρεται βέλτιστα.

Προσεγγιστικοί Αλγόριθμοι

έστω ένα πρόβλημα που δεν ανήκει στην κλάση P (ή τουλάχιστον πιστεύουμε ότι δεν ανήκει). Παρόλο που δεν μπορούμε ποτέ να επιτύχουμε μια βέλτιστη λύση σε πολυωνυμικό χρόνο, έχει ακόμα νόημα να προσπαθούμε να υπολογίσουμε εφικτές λύσεις. Οι προσεγγιστικοί αλγόριθμοι είναι μια προσέγγιση για τον υπολογισμό τέτοιων λύσεων, ενώ παράλληλα παρέχουν αποδεδειγμένες εγγυήσεις για την επιτευχθείσα βελτιστοποίηση. Συγκεκριμένα:
Ένας αλγόριθμος A για ένα πρόβλημα ελαχιστοποίησης Π λέγεται ότι είναι a-προσεγγιστικός αν, για οποιοδήποτε είσοδο x του προβλήματος, η τιμή $f(A(x))$ της λύσης που επιστρέφει ο A βρίσκεται εντός ενός παράγοντα a φορές της βέλτιστη λύσης $OPT(x)$ για αυτό το instance x. Με άλλα λόγια:

$$a = \max_{x \in I} \frac{f(A(x))}{OPT(x)}$$

όπου I είναι το σύνολο όλων των instances.
Για μια πλήρη εισαγωγή στους προσεγγιστικούς αλγόριθμους, δείτε [22, 19].

Χρόνος Ολοκλήρωσης

Το πεδίο των αλγοριθμικών προβλημάτων χρονοδρομολόγησης περικλείει μια τόσο μεγάλη γκάμα προβλημάτων που θα ήταν αδύνατο να τα περιγράψουμε επαρκώς σε μια τέτοια εργασία. Για μια πληρέστερη παρουσίαση παραπέμπουμε τον αναγνώστη στα βιβλία [13, 17] ενώ για μια πιο σύντομη εισαγωγή στο [20].
Στην συγκεκριμένη εργασία θα επιγκεντρωθούμε στο πρόβλημα της "Ελαχιστοποίησης του χρόνου ολοκλήρωσης".

> Για μια διεργασία i με χρόνο επεξεργασίας p_i ορίζουμε ως χρόνο ολοκλήρωσης C_i την χρονική στιγμή $t \geq p_i$ την οποία το σύστημα ολοκλήρωσε την επεξεργασία της.

Η απλούστερη μορφή του προβλήματος ορίζεται ως εξής:

> Έστω σύστημα επεξεργασίας διεργασιών με μια μηχανή. Δοθέντος ενός συνόλου n εργασιών, με χρόνους επεξεργασίας $p_i, \quad i \in [n]$, πρέπει να βρούμε μια μετάθεση I του συνόλου $\{1, \ldots, n\}$ τέτοια ώστε να ελαχιστοποιήσουμε την ποσότητα :
>
> $$\sum_{i \in [n]} C_i$$

Εισαγωγή στην σχετική βιβλιογραφία

Στην εργασία αυτή θα επιγκεντρωθούμε στην ελαχιστοποίηση του χρόνου ολοκλήρωσης υπό τους παρακάτω περιορισμούς:

- Μη-διακοπτόμενη (non-preemptive) χρονδρομολόγηση: Ο περιορισμός αυτός σημαίνει ότι από την στιγμή που μια διεργασία θα επιλεγεί από το σύστημα θα ολοκληρώσει την επεξεργασία της χωρίς διακοπές

- Ισοδύναμες μηχανές (identical machines): Θεωρούμε ότι οι μηχανές που διαθέτει το σύστημα έχουν ισοδύναμη υπολογιστική ισχύ (με άλλα λόγια ο χρόνος επεξεργασίας της διεργασίας δεν εξαρτάται από την μηχανή που θα ανατεθεί)

Online Χρονοδρομολόγηση με Διακοπές

Στο online πλαίσιο όπου έχουμε release dates και μία μηχανή, τότε υπάρχει ένας βέλτιστος αλγόριθμος πολυωνυμικού χρόνου (για απόδειξη δείτε [3]):

> **Shortest Processing Time First (SRPT)**: Σε οποιοδήποτε σημείο χρόνου κατά το οποίο υπάρχει άφιξη δουλειάς ή ο επεξεργαστής του συστήματος είναι κενός, διατάξτε τις διαθέσιμες δουλειές σε φθίνουσα σειρά του λόγου $\frac{w_i}{p_i}$ (όπου p_i είναι ο υπόλοιπος χρόνος επεξεργασίας της δουλειάς i) και προγραμματίστε τη δουλειά με τον μεγαλύτερο λόγο.

Πολλαπλές Μηχανές (Υπολογιστική Δυσκολία):
Για την πλειοψηφία των προβλημάτων χρονοδρομολόγησης, η εκδοχή με διακοπές είναι πιο εύκολη από την αντίστοιχη χωρίς διακοπές. Ωστόσο, το πρόβλημα της ελαχιστοποίησης του συνολικού χρόνου ολοκλήρωσης με $m \geq 2$ μηχανές που δεν σχετίζονται μεταξύ τους όταν επιτρέπονται διακοπές, έχει αποδειχθεί ότι είναι strongly* NP-Hard [10].

Online Χρονοδρομολόγηση χωρίς Διακοπές

Στο online μη-διακοπτόμενο πλαίσιο (non-preemptive), οι Hoogeveen και Vestjens [6] έδειξαν ότι:

> Κανένας ντετερμινιστικός αλγόριθμος δεν μπορεί να έχει competitive ratio μικρότερο από 2 στο πρόβλημα της μιας μηχανής.

2-προσεγγιστικός αλγόριθμος για το online πρόβλημα της μιας μηχανής:
Το 1995 οι Phillips, Stein και Wein [5] παρουσίασαν τον πρώτο αλγόριθμο σταθερής προσέγγισης για το online πλαίσιο:

> Κάθε φορά που φτάνει μια νέα διαδικασία ή η μηχανή είναι αδρανής: Δημιουργήστε το schedule με διακοπές P για την τρέχουσα είσοδο I^t, υπολογίστε τους χρόνους ολοκλήρωσης C_j^P, και προγραμματίστε τη διαδικασία i με το μικρότερο C_i^P (που δεν έχει ήδη εξυπηρετηθεί) πρώτη.

Αυτός ο αλγόριθμος, αν και δεν χρησιμοποιεί διακοπές, επί της ουσίας υλοποιεί τον αλγόριθμο SRPT (Shortest Remaining Processing Time) στο online πλαίσιο. Έτσι στη βιβλιογραφία συχνά αναφέρεται "το SRPT ως ένας 2-προσεγγιστικός αλγόριθμος για το online πρόβλημα".

Competitive Analysis

Η Ανταγωνιστική Ανάλυση (Competitive Analysis) εκφράζει μια μετρική που εμπίπτει στο πλαίσιο της ανάλυσης χειρότερης περίπτωσης (worst-case analysis). Συγκεκριμένα, στο competitive analysis ένας online αλγόριθμος ALG συγκρίνεται με τον βέλτιστο offline αλγόριθμο OPT (που γνωρίζει ολόκληρη την ακολουθία αιτημάτων σ εκ των προτέρων και μπορεί να την εξυπηρετήσει με ελάχιστο κόστος). Δεδομένης μιας ακολουθίας εισόδου σ, ορίζουμε $A(\sigma)$ και $\mathrm{OPT}(\sigma)$ τα κόστη που προκύπτουν από τους αλγορίθμους ALG και OPT, αντίστοιχα.
Έστω τώρα ένα πρόβλημα ελαχιστοποίησης. Ένας αλγόριθμος ALG ονομάζεται a-competitive αν υπάρχει μια σταθερά b τέτοια ώστε $ALG(\sigma) \leq a \cdot \mathrm{OPT}(\sigma) + b$, για όλες τις ακολουθίες σ. Η σταθερά b πρέπει να είναι ανεξάρτητη από την είσοδο σ.
Ισοδύναμα ένας ντετερμινιστικός online αλγόριθμος ALG είναι a-ανταγωνιστικός όταν a είναι ο μέγιστος λόγος μεταξύ του κόστους του ALG και του OPT, δηλαδή:

$$a = \max_{\sigma} \frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)}$$

Στη βιβλιογραφία, η ποσότητα a ονομάζεται competitive ratio. Ο ίδιος ορισμός ισχύει και για προβλήματα μεγιστοποίησης, όπου a είναι απλώς:

$$a = \min_{\sigma} \frac{\text{OPT}(\sigma)}{\text{ALG}(\sigma)}$$

Ο ορισμός επεκτείνεται και σε randomized αλγόριθμους, όπου ο στόχος είναι να συγκριθεί το αναμενόμενο κόστος του ALG με το αναμενόμενο κόστος της βέλτιστης λύσης OPT. Λέμε ότι ο ALG είναι a-competitive αν και μόνο αν υπάρχουν a, b τέτοια ώστε:

$$\mathbb{E}[\text{ALG}(\sigma)] \leq a \cdot \mathbb{E}[\text{OPT}(\sigma)] + b, \quad \forall \sigma$$

ή ισοδύναμα αν:

$$a = \max_{\sigma} \frac{\mathbb{E}[\text{ALG}(\sigma)]}{\text{OPT}(\sigma)}$$

Για μια πληρέστερη παρουσίαση παραπέμπουμε τον αναγνώστη στα άρθρα [9, 30].

Χρονοδρομολόγηση με περιορισμούς αμεσότητας

Εξετάζουμε μια παραλλαγή του online προβλήματος χρονοδρομολόγησης με περιορισμούς αμεσότητας. Εδώ, η "αμεσότητα" (promptness) αναφέρεται σε μια συγκεκριμένη απαίτηση όπως αυτή ορίζεται στο [23]. Στην ουσία, ο περιορισμός αμεσότητας υπαγορεύει ότι ο αλγόριθμος χρονοπρογραμματισμού πρέπει να λάβει μια απόφαση χρονοδρομολόγησης για μια διεργασία ακριβώς κατά την στιγμή άφιξης της. Αυτό δεν σημαίνει ότι ο αλγόριθμος είναι υποχρεωμένος να εξυπηρετήσει την εργασία αμέσως μόλις φτάσει. Ωστόσο, πρέπει να καθορίσει τον ακριβή χρόνο καθυστέρησης που θα υποστεί η εργασία συνολικά, ακριβώς την στιγμή της άφιξης της.

Το κίνητρο για τους περιορισμούς αμεσότητας γίνεται εμφανές όταν εξετάζουμε πρακτικές εφαρμογές. Φανταστείτε το σενάριο ενός σταθμού φόρτισης ηλεκτρικών οχημάτων. Οι πελάτες που φτάνουν με άδειες μπαταρίες επιθυμούν φυσικά μια ακριβή εκτίμηση για το πότε το όχημά τους θα είναι πλήρως φορτισμένο και έτοιμο για τη συνέχεια του ταξιδιού τους. Οι περιορισμοί αμεσότητας επιτρέπουν τη σχεδίαση ενός συστήματος προγραμματισμού που παρέχει αυτή την ακριβή πληροφορία. Με

την ενημέρωση των πελατών για τον ακριβή χρόνο φόρτισης κατά την άφιξή τους, το σύστημα προάγει τη διαφάνεια και την προβλεψιμότητα, ενισχύοντας τη συνολική ικανοποίηση των πελατών.
Ομοίως, ο προγραμματισμός ραντεβού υγειονομικής περίθαλψης παραδείγματα την αξία των περιορισμών ακρίβειας. Οι ασθενείς συχνά έχουν πολυάσχολα προγράμματα και εκτιμούν το να γνωρίζουν ακριβώς πότε θα γίνει το ραντεβού τους [24]. Ένα σύστημα χρονοπρογραμματισμού που τηρεί τους περιορισμούς αμεσότητας εξαλείφει την αβεβαιότητα και ελαχιστοποιεί τους χρόνους αναμονής, οδηγώντας σε μια πιο θετική εμπειρία για τους ασθενείς.
Στο [23], παρουσιάζουν αρχικά έναν άμεσο online αλγόριθμο που χρησιμοποιεί ένα στατικό μοτίβο. Συνοψίζουμε εν συντομία πώς λειτουργεί:

- Κατασκευάζουμε προσεκτικά ένα μοτίβο p από δυνάμεις του δύο (η κατασκευή εξηγείται παρακάτω).
- Ο αλγόριθμος χρονοδρομολόγησης: Στον πρώτο γύρο, ξεκινάμε με ένα νέο αντίγραφο του μοτίβου p. Κάθε διεργασία θα χρονοδρομολογηθεί στην πρώτη διαθέσιμη θέση του p που χωράει. Εάν δεν υπάρχει διαθέσιμη θέση για τη διεργασία i, κατασκευάζουμε ένα νέο αντίγραφο του p κ.λπ.

Κατασκευή του μοτίβου Το βασικό μοτίβο κατασκευάζεται ως εξής:
Κατασκευάζουμε ένα (αναδρομικό) μοτίβο S_d από θέσεις των οποίων τα μήκη είναι αυξανόμενες δυνάμεις του 2 μέχρι το d έτσι ώστε $\forall k \leq d$ το πρόθεμα του μοτίβου μήκους $n_k = 2^{k+1} - 1$ να είναι το S_k, όπου το S_k ορίζεται ως εξής:

$$\begin{cases} S_0 = 1 \\ S_k = S_{k-1}||S_{k-1}||2^k, \quad k > 1 \end{cases} \tag{1}$$

Όσο συνεχίζουν να φτάνουν διεργασίες, μεγαλώνουμε αυτό το μοτίβο επ' άπειρον.

Αν γνωρίζουμε το μέγιστο μήκος $p_{\max}$ τότε μπορούμε να χρησιμοποιήσουμε έναν απλό αλγόριθμο:

Χρονοδρομολόγησε κάθε διεργασία i που έρχεται στο σύστημα, στην

πρώτη διαθέσιμη θέση που ταιριάζει (σε οποιοδήποτε μηχάνημα). Αν δεν υπάρχει τέτοια θέση, δημιουργήστε ένα νέο αντίγραφο του μοτίβου.

Διαφορετικά, απαιτείται μια πιο περίπλοκη έκδοση που ονομάζεται "Δυναμικός Αλγόριθμος Χρονοδρομολόγησης" αλλά η βασική ιδέα παραμένει η ίδια. Αυτό που αλλάζει είναι ότι επειδή δεν γνωρίζουμε το $p_{\max}$ ξεκινάμε με το μικρότερο δυνατό k και αυξάνουμε το k μόνο όταν είναι απαραίτητο: όταν φτάσει μια διεργασία με μήκος μεγαλύτερο από αυτό που μπορεί να εξυπηρετήσει το S_k. Ο "Δυναμικός Αλγόριθμος Χρονοδρομολόγησης" είναι ουσιαστικά ένα σύνολο κανόνων για την δυναμική ενημέρωση του μοτίβου ενώ διατηρεί την αναδρομική δομή που ορίστηκε προηγουμένως.
Είτε υποθέσουμε ότι γνωρίζουμε το $p_{\max}$ είτε όχι (άρα χρησιμοποιούμε την δυναμική εκδοχή):

> Ο αλγόριθμος χροδρομοδρομολόγησης με ακολουθίες των Eden et al. [23] είναι $O(\log p_{\max})$-competitive όπου $p_{\max}$ είναι το μέγιστο μήκος διεργασίας στην ακολουθία εισόδου.

Ο προτεινόμενος αλγόριθμος χρονοδρομολόγησης

Ας υποθέσουμε ότι χωρίζουμε την ακολουθία είσοδου σε T γύρους. Έστω N_t το σύνολο των διεργασιών που έφτασαν στον γύρο t. Εάν σε κάθε γύρο $t \in [T]$ χρησιμοποιούμε τον αλγόριθμο των Eden et al., τότε έχουμε έναν αλγόριθμο με competitive ratio $O(\log(p_{\max}))$. Τώρα, τίθεται το ερώτημα: Υπάρχει ένα μοτίβο (pattern) που επιτυγχάνει μικρότερο συνολικό χρόνο ολοκλήρωσης; Μπορούμε να αξιοποιήσουμε τις ακολουθίες είσοδου που παρατηρήθηκαν στο παρελθόν για να σχεδιάσουμε ένα μοτίβο πιο προσαρμοσμένο στα συγκεκριμένα δεδομένα που συναντά;

Συνάρτηση απωλειών

Κάθε pattern (μοτίβο) είναι ένα διάνυσμα ακεραίων $a \in \mathbb{N}^d$ για κάποιο $d \in \mathbb{N}$. Αυτός ο χώρος αναζήτησης είναι τεράστιος. Προκειμένου να βρούμε μια υπολογιστικά αποδοτική λύση υποθέτουμε ότι γνωρίζουμε το βέλτιστο μήκος ακολουθίας L και το

$p_{\max}$. Αν επιπλέον υποθέσουμε ότι αναζητούμε αύξουσες ακολουθίες και χωρίς βλάβη της γενικότητας ότι οι τιμές των συντεταγμένων είναι δυνάμεις του 2 τότε μπορούμε να γράψουμε μια αποδοτική αναπαράσταση του άγνωστου διανύσματος (pattern) ως :

$$a = \langle a_1, \ldots, a_l \rangle$$

όπου $l = \lceil p_{\max} \rceil$ Ακόμα και έτσι όμως, το να βρεθεί ένας κλειστός τύπος $F(a)$ ο οποίος να περιγράφει τον χρόνο ολοκλήρωσης μιας ακολουθίας εισόδου χρησιμοποιώντας το pattern a και τον greedy αλγόριθμο χρονοδρομολόγησης αποδεικνύεται πολύ απαιτητικό. Η ιδέα μας είναι να ορίσουμε μια συνάρτηση C^1 η οποία προσεγγίζει τον χρόνο ολοκλήρωσης αλλά έχει μια απλούστερη μορφή. Για να το πετύχουμε αυτό μπορούμε να σκεφτούμε ότι για μια διεργασία i που χρονοδρομολογήθηκε στην t-οστή ακολουθία ο χρόνος ολοκλήρωσης της διασπάται σε 3 όρους:

1. χρόνος ολοκληρωσης των t-1 ακολουθιών που προηγούνται
2. σχετικός χρόνος αναμονής $r(i)$ εντός της t-οστής ακολουθίας
3. χρόνος επεξεργασίας p_i

Εξ'αυτών είναι δύσκολο να προσδιορίσουμε τον χρόνο $r(i)$ καθώς αυτός εξαρτάται από όλες τις διεργασίες και (την σειρά αυτών) που προγραμματίστηκαν πριν την i στην ακολουθία υπ'αριθμόν t. Παρόλα αυτά επειδή παρατηρούμε ότι $0 \leq r(i) \leq L$ μπορούμε να θεωρήσουμε την συνάρτηση:

$$C^1(y) = \sum_{i \in N} \sum_{t \in T} y_{it}[(t-1)L] + \sum_{i \in N} \sum_{t \in T} y_{it} p_i$$

Η συνάρτηση αυτή αποτελεί προφανώς μια υποεκτίμηση του πραγματικού χρόνου ολοκλήρωσης. Ωστόσο η απλή μορφή της θα μας επιτρέψει να την μελετήσουμε σε βάθος και θα αποτελέσει το βασικό εργαλείο μας για την ανάλυση και υλοποίηση του αλγορίθμου μας. Στο τεχνικό μέρος της εργασίας μπορεί κανείς να βρει μια αναλυτική μελέτη των ιδιοτήτων της συνάρτησης υποεκτίμησης.

Algorithm 1 Αλγόριθμος Online Prompt Scheduling

Είσοδος: Περίοδος L, μέγιστος χρόνος επεξεργασίας $p_{\max}$, κανονικοποιητής $U : R^l \to R$

Έξοδος: Ένα πρόγραμμα διεργασιών $(y^{\sigma_1}, \ldots, y^{\sigma_T})$

1: Ορίστε $l \leftarrow \log p_{\max}$ (για τη διαμόρφωση περιορισμών C^1)
2: **for** $t \in [T]$ **do**
3: Υπολογίστε $(\tilde{y}, \tilde{\sigma}_t) = \arg\min_{y,\sigma} \left(\sum_{\tau=1}^{t-1} C_\tau^1(y_\tau^\sigma) + \frac{U(\sigma)}{\eta} \right)$
4: Στρογγυλοποιήστε $\tilde{\sigma}_t$ σε $a_t = \lceil \tilde{\sigma}_t \rceil$
5: Προγραμματίστε τις διεργασίες (greedily) στο N^t χρησιμοποιώντας το a_t
6: **end for**

Για να αναλύσουμε αυτόν τον αλγόριθμο αποτελεσματικά, παρατηρούμε ότι περιλαμβάνει τρία διακριτά στοιχεία: έναν αλγόριθμο μάθησης, έναν αλγόριθμο στρογγυλοποίησης και έναν αλγόριθμο χρονοδρομολόγησης.
Ο αλγόριθμος μάθησης που χρησιμοποιούμε στο βήμα 3 είναι επί της ουσίας ο αλγόριθμος Follow The Regularized Leader. (παραπέμπουμε τον αναγνώστη στη σχετική ενότητα της εργασίας).

Στρογγυλοποιημένη Λύση

Υπόθεση Υποθέτουμε ότι γνωρίζουμε την βέλτιστη περίοδο L (δηλαδή την περίοδο που θα χρησιμοποιούσε ο βέλτιστος (offline) αλγόριθμος χρησιμοποιώντας μια σταθερή ακολουθία για να προγραμματίσει όλη την είσοδο N).

Λήμμα

Έστω ότι $C^1(y^\sigma)$ είναι μία εκτίμηση (σύμφωνα με το $C^1()$) του συνολικού χρόνου ολοκλήρωσης χρησιμοποιώντας το pattern $\sigma \in R^d$ σε μια ακολουθία εισόδου N. Έστω ακόμη ότι $a = \lceil \sigma \rceil$ είναι το στρογγυλοποιημένο pattern. Τότε:

$$C^1(y^a) \leq 3C^1(y^\sigma), \quad \forall y^\sigma \in \{0,1\}^{|N|}, \quad a = \lceil \sigma \rceil$$

Θεώρημα 1

Εάν (σ, y^σ) είναι οποιαδήποτε εφικτή (κλασματική) λύση του προγράμματος C^1, τότε για το schedule y^a που προκύπτει από το μοτίβο $a = \lceil \sigma \rceil$ ισχύει:

$$F(y^a) \leq 3C^1(y^*) + 2LN \leq 3F(y^*) + 2LN$$

όπου το y^* είναι μια βέλτιστη λύση.

Μέτρηση Απόδοσης

Θέλουμε να συγκρίνουμε το $F_t(y_t^{a_t})$ με το πραγματικό βέλτιστο κόστος $F_t(y_t^{p^*})$ της καλύτερης σταθερής ακολουθίας p^*. Χρησιμοποιώντας το Θεώρημα 1 , προκύπτει ότι το πραγματικό κόστος $F_t(y_t^{a_t})$ είναι το πολύ $3C_t^1(y_t^{\sigma_t}) + 2LN$. Για την αξιολόγηση του αλγορίθμου μας Θα χρησιμοποιήσουμε μια παραλλαγή του Regret, συγκεκριμένα το c-Regret όπως ορίζεται σε σχετικές εργασίες ([29]). Για $c = 1$ ο ορισμός αυτός συμπτίπτει με τον κλασσικό ορισμό του Regret. Συγκεκριμένα στο πρόβλημα μας, ορίζουμε το c-regret ως:

$$R(T) = \sum_{t=1}^{T} F_t(y_t^{a_t}) - c \sum_{t=1}^{T} F_t(y_t^{p^*})$$

όπου το p^* είναι ο βέλτιστος σταθερός αντίπαλος, δηλαδή $p^* = \arg\min \sum_{t=1}^{T} F_t(y_t^{p^*})$. Ορίζουμε επίσης:

$$\tilde{R}(T) = \sum_{t=1}^{T} C_t^1(y_t^{a_t}) - \sum_{t=1}^{T} C_t^1(y_t^{\sigma^*})$$

όπου (σ^*, y^{σ^*}) είναι η λύση του κυρτού προγράμματος με αντικειμενική συνάρτηση $\min_{y,\sigma} \sum_{t=1}^{T} C_t^1(y^\sigma)$ και τους ίδιους περιορισμούς όπως πριν.

Λήμμα 2

Εάν $R(T)$ είναι η μετρική του c-Regret που ορίζεται παραπάνω, τότε για $c = 4$ ισχύει:

$$R(T) \leq 3\tilde{R}(T)$$

Συνάρτησης Απώλειας: Lipschitzness

Μπορούμε να γράψουμε:

$$C_t^1(y) = \sum_{i \in N} \sum_{\tau=1}^{n_t} [(\tau - 1)L + p_i] y_{i,\tau} = \langle A, y \rangle_F$$

όπου $\langle \cdot, \cdot \rangle_F$ είναι το εσωτερικό γινόμενο Frobenius μεταξύ των μητρών $A \in \mathbb{R}^{N^t \times T}$ και $y \in \mathbb{R}^{N^t \times T}$, τους οποίους ορίζουμε ως:

$$A = \begin{bmatrix} p_1 & L + p_1 & \dots & L(n_t - 1) + p_1 \\ p_2 & L + p_2 & \dots & L(n_t - 1) + p_2 \\ \vdots & \vdots & \ddots & \vdots \\ p_{N^t} & L + p_{N^t} & \dots & L(n_t - 1) + p_{N^t} \end{bmatrix} \quad \text{και} \quad y = \begin{bmatrix} y_{1,1} & y_{1,2} & \dots & y_{1,n_t-1} \\ y_{2,1} & y_{2,2} & \dots & y_{2,n_t-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N^t,1} & y_{N^t,2} & \dots & y_{N^t,n_t-1} \end{bmatrix}$$

Για να είμαστε ακριβείς, έστω $T^y(t)$ και $T^x(t)$ είναι ο αριθμός των αντιγράφων που χρειάζονται για να προγραμματίσουμε την είσοδο N^t χρησιμοποιώντας τις ακολουθίες x και y αντίστοιχα. Τώρα, ας ορίσουμε $n_t = \max\{T^y(t), T^x(t)\}$ και ας υποθέσουμε χωρίς περιορισμό γενικότητας ότι $T^x(t) < T^y(t)$. Μπορούμε να γράψουμε $C_t^1(x) = A^T \cdot y_x$ προσθέτοντας στη y_x επιπλέον στήλες για να την επεκτείνουμε στον ορίζοντα n_t με μηδενικές εγγραφές.

Η συνάρτηση $C_t^1(y^\sigma) = \langle A, y \rangle_F$ είναι $\|A^T\|_F$ Lipschitz ως προς οποιαδήποτε νόρμα, όπου $\|\cdot\|_F$ είναι η Frobenius νόρμα, δηλαδή: $\|A^T\|_F$ όπου το N^t είναι η είσοδος ακολουθίας στον γύρο t και το $n_t = n_t(\sigma)$ είναι ο αριθμός των αντιγράφων του προτύπου σ που απαιτούνται για να προγραμματίσουμε την είσοδο.

Το κεντρικό αποτέλεσμα της εργασίας διατυπώνεται παρακάτω:

Θεώρημα 2

Ο προτεινόμενος αλγόριθμος πετυχαίνει υπογραμμικό c-Regret για c=4 ως προς το T.

Chapter 1

Introduction

1.1 Contributions

This work tackles the challenge of online learning under promptness constraints. We introduce a novel model that captures the trade-off between learning accuracy and timely task completion. Our objective is to design an algorithm that exhibits optimal regret guarantees, a measure of performance loss compared to the best possible strategy in hindsight.
Our approach rests on two key pillars:

- Approximating Offline Completion Time: We establish a method to approximate the total completion time achievable in an offline setting where all information is available upfront, while adhering to the promptness constraints. This approximation is achieved by formulating an Integer Program (IP). We then delve into solving the Linear Programming (LP) relaxation of this IP, a continuous relaxation that provides a lower bound on the optimal solution. Finally, we propose a rounding scheme to convert the fractional LP solution into an integral solution suitable for the IP. This rounded solution is provably at most a factor of a times worse than the optimal IP solution (where a is a constant).

- Learning Algorithm with Regret Guarantees: By leveraging the concept of "follow the regularized leader," we construct a no-regret learning algorithm. Here, regret is measured using a slightly stricter notion than the standard c-regret. This algorithm operates under a regime of underestimation losses. The final step involves bounding the true performance of our learning algorithm by the regret incurred on the approximated losses (derived from the relaxation of the IP). This connection allows us to translate the regret guarantee on the approximated losses into a guarantee on the actual completion time achieved by the learning algorithm.

In essence, this work similar to [27] proposes a novel approach to online learning for scheduling with promptness constraints. By approximating the optimal offline completion time and employing a carefully crafted learning algorithm, we ensure that the learning process progresses efficiently while respecting the time constraints imposed on task execution.

1.2 Organization

The structure of this work is as follows:

- Chapter 2: This chapter provides a concise introduction to scheduling literature relevant to minimizing total completion time. It also highlights key findings from existing research in this domain.
- Chapter 3: In this chapter, we shift our focus towards learning with promptness constraints. We delve into the work of Feldman et al. (provide a brief overview of their work or its significance).
- Chapter 4: This chapter marks the beginning of the original research presented in this work. We introduce the central research question we aim to address. Subsequently, we present the proposed algorithm, its technical underpinnings, and the corresponding performance guarantees.

Chapter 2

Introduction to online scheduling and the minimization of total completion time

In this section, we discuss scheduling, the combinatorial problem of interest.

2.1 General

In the field of operations research and computer science, scheduling problems involve assigning a sequence of jobs to resources, such as machines or workers, in an optimal way. Each job is characterized by a specific processing time, which represents the duration required to complete the job. The goal of scheduling is often to optimize a particular objective, such as minimizing the total completion time, reducing job tardiness, or maximizing resource utilization. Scheduling problems can vary greatly in complexity, depending on factors like the number of available resources, whether jobs can be preempted or must be executed without interruption, and whether resources are identical or have different capabilities. These problems are fundamental in many real-world applications, including manufacturing, project management, and computer systems, where efficient scheduling can lead to signif-

icant cost savings and productivity improvements.
Due to the vast diversity of scheduling problems, it is impractical to cover all variants comprehensively in this thesis.In this literature review chapter, we focus on online scheduling with the objective of minimizing total completion time. The following sections delve into various aspects of this topic, including the theoretical foundations, different algorithms proposed in the literature, and their comparative performance.
Online scheduling problems have been studied since as early as 1960 [2]. Since then, a rich body of literature has developed, exploring various objectives (e.g. minimization of makespan, total completion time or flow time), machine models, and input assumptions. However, research in this area became more in-depth with the introduction of the competitive analysis framework. For a comprehensive review of the results, see [18].

2.1.1 Approximation Algorithms

Consider any problem that does not lie in $\mathcal{P}$ (or at least we believe that it does not). Although we can never attain an optimal solution in polynomial time, it still makes sense to try to compute feasible solutions. Approximation algorithms are an approach to compute such solutions while also providing provable guarantees about the achieved optimality. In particular:

> An algorithm $\mathcal{A}$ for a minimization problem P is said to be a-approximate if, for any instance x of the problem, the value $f(\mathcal{A}(x))$ of the solution returned by $\mathcal{A}$ is guaranteed to be within a factor of a times the optimal solution $OPT(x)$ achievable for that instance. In other words:
>
> $$a = \arg\max_{x \in I} \frac{f(\mathcal{A}(x))}{\mathrm{OPT}(x)}$$
>
> where I is the set of all instances.

For a complete introduction to approximation algorithms, see [22, 19].

2.1.2 Online Algorithms

The term “online algorithm" refers to algorithms that receive input sequentially and must make irrevocable decisions based solely on the available data at each step. In contrast, an “offline algorithm" is given the entire input stream from the outset. Clearly, an online algorithm is always at a disadvantage compared to its corresponding offline algorithm due to incomplete information. As a result, online algorithms can only approximate the optimal solution. To assess their performance, the most commonly used measure is the competitive ratio, which comes from worst-case analysis.

A natural example of an online problem comes from operating systems. The problem of *paging* involves managing a disk with N pages and a faster memory (cache) with k pages. The goal is to use the fast memory so that each new request can be found in the fast memory. If a requested page is not in the fast memory, we incur the cost of fetching it from the disk. This event is known as a "cache miss" or "page fault." Requests (pages) arrive sequentially over time, and the objective is to minimize the number of page faults while only being able to change one page in the fast memory at a time.

2.1.3 Competitive Analysis

Competitive Analysis is a strong worst-case performance measure where an online algorithm ALG is compared to an optimal offline algorithm OPT that knows the entire request sequence σ in advance and can serve it with minimum cost. Given a sequence σ, let A(σ) and OPT(σ) denote the costs incurred by ALG and OPT, respectively.

Consider a minimization problem. Algorithm ALG is called a-competitive if there exists a constant b such that $ALG(\sigma) \leq a \cdot OPT(\sigma) + b$, for all sequences σ. The constant b must be independent of the input σ.

An equivalent definition says that a deterministic online algorithm ALG is a-competitive when a is the maximum ratio between the cost of ALG and OPT,

i.e.:

$$a = \max_{\sigma} \frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)}$$

In the literature, this quantity a is called the competitive ratio. The same definition holds for maximization problems as well, where a is simply:

$$a = \min_{\sigma} \frac{\text{OPT}(\sigma)}{\text{ALG}(\sigma)}$$

The definition also extends to randomized algorithms, where the goal is to compare the expected cost of ALG with the expected cost of OPT. We say that ALG is a-competitive if and only if $\exists\, a, b$ such that $\mathbb{E}[\text{ALG}(\sigma)] \leq a \cdot \mathbb{E}[\text{OPT}(\sigma)] + b, \quad \forall \sigma$ or equivalently if:

$$a = \max_{\sigma} \frac{\mathbb{E}[\text{ALG}(\sigma)]}{\text{OPT}(\sigma)}$$

For a throughout review of online algorithms see [9, 30].

2.1.4 The objective of "Total Completion Time" (TCP)

For n processes the (unweighted) total completion time (TCP) is defined as:

$$\text{TCP} = \sum_{i=1}^{n} C_i$$

where C_i is the total completion time of process i; the absolute time step at which process i finishes execution within the schedule.

The most general version of the scheduling problem we consider optimizes the "Weighted Total Completion Time" (WTCP). Here, each job i has an associated processing time p_i and a weight w_i. The weight reflects the relative importance of job i. The objective function aims to minimize the following:

$$W\text{TCP} = \sum_{i=1}^{n} w_i C_i$$

2.1.5 The “Map" of Total Completion Time problems

Scheduling problems that appear in literature come in many different forms [17]. Here we give a basic decompositionof scheduling problems with the objective to minimize total completion time, based on the following key features:

Online/Offline input sequence: In the offline scenario, the entire sequence of tasks (or processes) is known upfront before the scheduling algorithm begins. This allows the algorithm to consider all tasks simultaneously and potentially find an optimal solution. In contrast, in the online scenario, the tasks or processes are revealed sequentially. The algorithm must make decisions about scheduling, with limited knowledge about future tasks.

Processes with/without release dates: Processes may all arrive at the system at the same timestep or have varying arrival times. The latter devises a different problem where the scheduling algorithm should respect the fact that it cannot schedule a process before its release date. If we are in the online case, then essentially there are release dates but if we are in the offline case we may or may not have release dates.

Preemptive/Non-Preemptive Scheduling: Preemptive scheduling concerns scheduling algorithms where interruptions are allowed. A process can be stopped and be resumed later, while non-preemptive assumes uninterrupted processing.

Single/Multiple Machines: In the setting of multiple machines the literature refers to :

- “parallel machines": each job can be processed on any of the available machines, but it must be processed by only one machine at a time. Similarly, each machine processes only one job at a time.
- “identical machines": the machines have identical processing powers
- “unrelated machines": If we have m machines and n processes, there is no relation between the processing times p_{ij} i.e they are predefined, independent of the scheduling assignment.

Throughout this text, when we mention multiple machines, we are considering parallel identical machines unless specified otherwise.

In scheduling literature, it extremely common to refer to the respective problems using the notation introduced by Graham, Lawler, Lenstra, and Rinnooy Kan [4]. The problems that we consider are special variants of the single machine scheduling problem $(1, P, R)|r_j|, \text{prec}, (, \text{pmtn})|\sum_j w_j C_j$. The entry "1" in the first field indicates that the scheduling environment provides only one machine,"P" indicates parallel machines and R indicates "unrelated" machines. The second field can be empty or contain some of the job characteristics r_j , prec, or pmtn, indicating whether there are nontrivial release dates or precedence constraints and whether preemption is allowed. For example, $1|r_j|\sum_j w_j C_j$ refers to the **single-machine**, **with release dates** to minimize weighted total completion time scheduling where preemption is not-allowed and there are no precedence constraints.

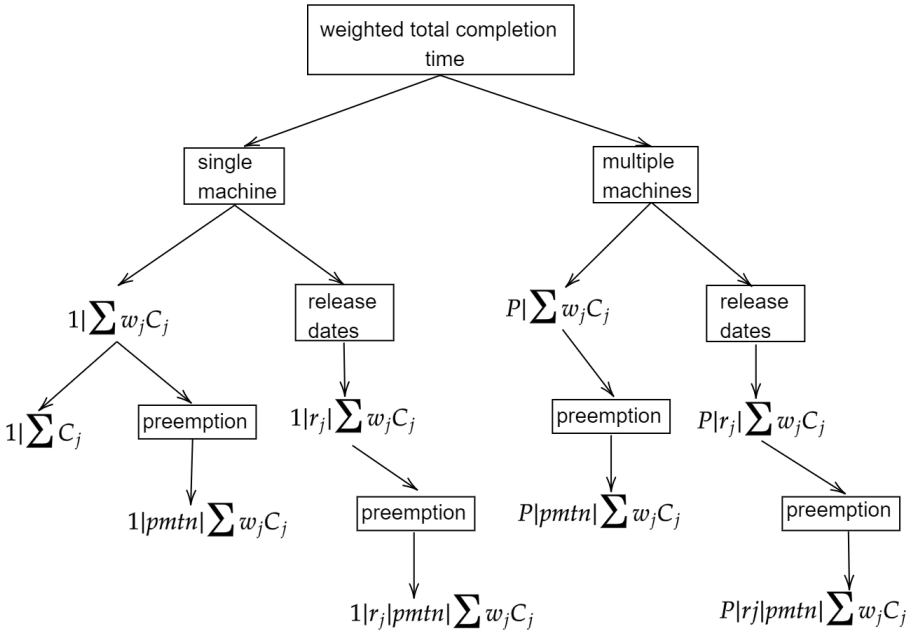


Figure 2.1: A primal classification of total completion time problems

2.2 Warmup: Scheduling in the offline setting:

In the offline single-machine setting where preemption is not allowed and weights are uniform for all processes (w.l.o.g assume unit weights), the optimal algorithm for minimizing the unweighted total completion time is:

Shortest Process Time First (SPT): serve the shortest process first.

Intuitively, using the formula for total completion time, we can see that the i-th process in our schedule contributes $(n - i + 1)p_i$ to the total completion time:
Let s_i be the time step at which process i will start being executed by the system and assume $I = 1, 2, .., n$ is the order we serve (schedule) the processes:

$$\sum_{i=1}^{n} C_i = \sum_{i=1}^{n}(p_i + s_i) = p_1 + (p_1 + p_2) + ... + (p_1 + .. + p_n)$$
$$= np_1 + (n-1)p_2 + .. + p_n = \sum_{i=1}^{n}(n - i + 1)p_i$$

Therefore, the optimal solution should minimize the contribution of larger processing times by scheduling them as late as possible.

In the offline single-machine setting with weights, the optimal algorithm for minimizing the weighted total completion time is

Shortest Process Time First (WSPT): serve the process with the minimum $\frac{p_i}{w_i}$ first.

This rule is due to Smith in 1965 [1]. Smith's ratio rule produces an optimal schedule for the non-preemptive problem, but as it has been proven, preemption introduces no other benefit; therefore, it is also optimal in the non-preemptive case. However, introducing release dates makes the problem NP-Hard :

Minizing total completion time in the single-machine case with release dates $(1 \mid r_j \mid \sum_j C_j)$, is NP-Hard.

The proof uses a reduction from 3-PARTITION. In addition, there exists a PTAS for this problem.

Multiple Machines (Hardness results):
In the offline setting with arbitrary weights and a fixed number $m \geq 2$ of identical machines, it was shown in 1974 that the problem is NP-Hard using a reduction from knapsack [6]. In the special case, where $w_i = 1,\ \forall i \in [n]$ and we have m machines (specified in the input) the problem, $(Pm|\sum C_j)$, is solvable in polynomial time by using SPT to choose a process and assigning it to the first available machine [14]. The latter is known as **"list scheduling"**.

Summary for offline TCP problems		
problem	**complexity**	**algorithm**
$1\|\sum_{i=1}^{n} C_j$	P	SPT
$P\|\sum_{i=1}^{n} C_j$	P	SPT
$1\|\sum_{i=1}^{n} w_j C_j$	P	WSPT
$1\|pmtn\|\sum_{i=1}^{n} w_j C_j$	P	WSPT
$P\|\sum_{i=1}^{n} w_j C_j$	NP-Hard	
$1\|r_j\|\sum_{i=1}^{n} C_j$	NP-hard	

2.3 Online Scheduling to minimize total completion time

2.3.1 Online Scheduling with preemption

In the online regime where we have release dates and a single machine then there is an optimal polynomial-time algorithm (for proof see [3]):

> **Shortest Processing Time First (SRPT)**: At any point of time at which there is a job release or the system processor is empty, order the available jobs in decreasing order of $\frac{w_i}{p_i}$ (where p_i is the remaining processing time of job i) and schedule the job with the largest ratio.

Multiple Machines (Hardness results):
For the majority of scheduling problems, the preemptive version is not harder to solve than its non-preemptive counterpart. However, the problem of minimizing total completion time with $m \geq 2$ unrelated identical machines when preemption is allowed is proved to be strongly* NP-hard [10].
**Strongly NP-hard:* An optimization problem is called strongly NP-hard if it is NP-hard even for instances in which all numbers are polynomially bounded in the input size.

2.3.2 Online Scheduling without preemption

In the online non-preemptive setting, Hoogeveen and Vestjens [6] showed that:

> No deterministic algorithm can have a competitive ratio smaller than 2 in the single-machine problem.

2-approximation algorithm for the online single-machine setting:
In 1995 Phillips, Stein, and Wein [5] provided the first constant approximation algorithm for the online setting:

> Every time a new process arrives or the machine is idle: Form the preemptive schedule P on the current input I^t, compute the completion times C_j^P, and schedule the process i with the smallest C_i^P that has not been served) first.

This algorithm, although not employing preemption, effectively implements SRPT (Shortest Remaining Processing Time) in the online setting. Due to this characteristic, the literature often refers to SRPT as a 2-approximation algorithm in the online setting.

Proof)
Assume C_j^N is the total completion time of job j in the non-preemptive schedule. Now

$$C_j^N = \sum_{k:C_k^P \leq C_j^P} p_k + p_j$$

where $w_j = \sum_{k:C_k^P \leq C_j^P} p_k$ is the waiting time of process j.
Naturally:

$$p_j \leq C_j^P$$

In addition, if S is the set of processes that finished before j in the optimal preemptive schedule P, $S = \{k : C_k^P \leq C_j^P\}$ then those are going to be served before j also in N (by construction of the algorithm) , therefore :

$$C_j^P \geq \sum_{k \in S} p_k$$

Combining all the results above we get:

$$C_j^N \leq 2C_j^P \quad \square$$

Lower bound on any online algorithm

When we do not restrict to deterministic algorithms it was shown, [8], that:

> Any online algorithm has competitive ratio a where
>
> $$a = \frac{e}{e-1} \approx 1.58$$

In [7] a randomized algorithm with the expected competitive ratio $e/(e-1)$ was given. Their approach is based on what we call a-point scheduling (e.g [11]). Given a preemptive schedule P and a parameter $a \in [0, 1]$, we define $C_j^P(a)$ to be the time at which an a-fraction of job j is completed. An a-schedule is a non-preemptive schedule obtained by list scheduling jobs in order of increasing $C_j^P(a)$, possibly introducing idle time to account for release dates.

Multiple machines

For the online total completion time minimization problem without preemption, Arjen P. A. Vestjens provided the following result in his phd thesis [8]

> Any deterministic online algorithm for minimizing total completion time on identical machines has a performance bound of at least 1.309. This lower bound can be improved for specific values of m.

Table 2.1: Lower bounds scaling with m for minimizing $\sum C_j$

m	1	2	3	4	5	100	∞
LB	2	1.520	1.414	1.373	1.364	1.312	1.309

In [5] they showed that for identical machines SRPT is a 3-approximation while in [7] they provided a 2.85-competitive algorithm that runs in $O(n \log n)$.

Table 2.2: The one star * denotes randomized while the double stars ** denote deterministic algorithms. See [8] for more insights.

Approximation and hardness results for online TCP problems			
problem	**complexity**	**best-known**	**lower bounds**
$1\|r_j\|\sum_{i=1}^{n} C_j$	NP-hard	$1+\epsilon^*$ 2^{**}	1.5280^* 2^{**}
$P\|r_j\|\sum_{i=1}^{n} C_j$	NP-Hard		1.309^{**}
$1\|r_j\|\text{pmtn}\|\sum_{i=1}^{n} C_j$	P	SRPT	
$P\|r_j\|\text{pmtn}\|\sum_{i=1}^{n} C_j$	NP-hard	$1+\epsilon^*$ 2^{**}	1.047^{**}

2.3.3 Weighted Total Completion Time

The more general version of the problem should be treated as a problem independently since results can vary.

Table 2.3: The one star * denotes deterministic while the double stars ** denote randomized algorithms. See [16] for the whole summary.

Approximation and hardness results for single machine			
problem	**complexity**	**best-known**	**lower bounds**
$1\|r_j\|\sum_{i=1}^{n} C_j$	NP-hard	$1+\epsilon^*$ 2^{**}	1.5280^* 2^{**}
$1\|r_j\|\sum_{i=1}^{n} w_jC_j$	P	$1+\epsilon^*$ 2^{**}	1.5280^* 2^{**}
$1\|\text{pmtn}\|\sum_{i=1}^{n} w_jC_j$	NP-hard	$1+\epsilon^*$ 2^{**}	1.038^* 1.073^{**}

In [12] they provide a 4/3 algorithm for the minimization of the weighted total completion time where preemption is allowed ,$1|\text{pmtn}|\sum_{i=1}^{n} w_jC_j$, using a-points.

As for the multiple machines, $P|\sum w_jC_j$ is NP-complete. However, the weighted version of Smith's ratio (WSPT) is a $(1+\sqrt{2})/2$ approximation.

Chapter 3

Online prompt scheduling without preemption

3.1 Problem description and the notion of promptness

We consider a variation of online scheduling with promptness constraints. Here, "promptness" refers to a specific requirement as defined in [23]. In essence, the promptness constraint dictates that the scheduling algorithm must make a scheduling decision for a job exactly at its release time. This means the algorithm is not obligated to serve the job immediately upon arrival. However, it must determine the exact amount of delay the job will experience at the time of its release.
The motivation for promptness constraints becomes readily apparent when we explore practical applications. Imagine the scenario of an electric vehicle charging station. Customers arriving with depleted batteries naturally desire an accurate estimate of when their vehicle will be fully charged and ready for their onward journey. Promptness constraints enable the design of a scheduling system that delivers this very information. By informing customers of the precise charging time upon arrival, the system fosters transparency and predictability, enhancing overall

customer satisfaction.
Similarly, healthcare appointment scheduling exemplifies the value of promptness constraints. Patients often navigate busy schedules and appreciate knowing exactly when their appointment will take place [24]. A scheduling system that adheres to promptness constraints eliminates uncertainty and minimizes waiting times, leading to a more positive patient experience.
In essence, promptness constraints represent more than just a technical hurdle in online scheduling algorithms. They reflect a fundamental understanding of user needs and the desire for predictability in dynamic environments. By incorporating promptness constraints, scheduling algorithms can evolve beyond raw efficiency metrics and contribute to a more user-centric experience in real-world applications.

3.1.1 Static Algorithms

In [23], they first present a prompt online algorithm that uses a static pattern. We briefly summarize how it works:

- Carefully construct a pattern p from powers of two 2 (the construction is explained below).
- The scheduling algorithm: In the first round, start with a fresh copy of the pattern p. Schedule each process at the first slot of p that it fits. If there is no available slot for process i, use a fresh copy of p e.t.c

The first static algorithm that they consider is :

Algorithm A that uses pattern P made off a sequence of lengths that are increasing powers of 2 so as there are exactly 2^{d-i} slots of length 2^i:

$$P = \langle 1, \dots 1, 2 \dots 2, \dots, 2^{d-1}, 2^{d-1}, 2^d \rangle$$

Secondly, they consider a pattern with the same slots but with a different arrangement:

Algorithm B

Constructing the pattern The basic pattern is made as such:
We build a (recursive) pattern S_d of slots whose lengths are increasing powers of 2 up to d such as $\forall k \leq d$ the prefix of the pattern of length $n_k = 2^{k+1} - 1$ is S_k where S_k is defined as follows:

$$\begin{cases} S_0 = 1 \\ S_k = S_{k-1}||S_{k-1}||2^k, \quad k > 1 \end{cases} \tag{3.1}$$

As long as processes keep arriving, we grow this pattern infinitely.

Proposition: In S_d each power takes 2^d time in total: 2^{d-i} copies of slots of length 2^i, $\forall i \in \{0, .., d\}$.

Proof)
The proof is by induction on the maximum exponent in the pattern, namely d.

- **Base:** For $d = 1$ we have $S_1 = S_0||S_0||2^1 =< 1, 1, 2 >$. In S_1 we have $2^{1-0} = 2$ slots of length $2^0 = 1$ and $2^{1-1} = 1$ slot of length $2^1 = 2$, so it holds.
- **Inductive hypothesis:** Assume it holds for S_{d-1}
- **Inductive step:** Since $S_d = S_{d-1}||S_{d-1}||2^d$ and by applying the hypothesis on each copy of S_{d-1} we get that for every $i < d$ there are $2 \times 2^{d-1-i} = 2^{d-1}$ copies of slots of length 2^i.

3.1.2 Lower Bounds on the static algorithms

Consider an input sequence with $2^{d/2}$ processes with lengths equal to 2 and one last process with length 2^d. SPT (the optimal algorithm) will schedule first the

short processes and finally the large one and will pay:

$$\begin{aligned}\text{COST(OPT)} &= 2 + (2+2) + \cdots + \underbrace{(2 + \cdots + 2)}_{2^{d/2} \text{ times}} + (2^{d/2} \cdot 2 + 2^d) \\ &= \sum_{i=1}^{d/2} 2i + (2^{d/2} \cdot 2 + 2^d) = \Theta(2^d)\end{aligned}$$

Meanwhile, the static algorithm A 3.1.1 will schedule (by construction) the length 2 processes after the 2^d unit lengths and will pay:

$$\text{COST(ALG)} \geq 2^{d/2} \cdot 2^d + (2^d + 2^{d/2} \cdot 2 + 2^d) \geq 2^{d/2} \cdot 2^d$$

Now $P_{\max} = 2^d$ and therefore the competitive ratio for Algorithm A is $\Omega(\sqrt{P_{\max}})$.

In addition, in [23] they prove a lower bound for the static Algorithm B that schedules using as a basic recursive pattern :
Assume Algorithm B uses S_n as a basic pattern for some fixed n.
Now for some fixed k with $k < n$ they consider an input sequence that consists of three parts:

- Large jobs: $(n-k)2^{n-k}$ jobs, each with size 2^k.
- Medium jobs: 2^{n-i} jobs with processing time 2^i for all values of i from 0 to k-1.
- Small jobs: 2^n jobs with processing time 1.

All the jobs arrive at time t=0 but one after another (we “see" them sequentially).

We know that the first copy of S_n will end at $e(S_n(0)) = \sum_{i=0}^{n} 2^{n-i}2^i = (n+1)2^n$. Algorithm B will fit all the *large* and *medium jobs* in $S_n(0)$ and then it can will create a new copy of S_n to fit all the unit jobs.
To see why: There are $\sum_{i=k}^{n} 2^{n-i}2^i = (n-k)2^k$ slots with length $\geq 2^k$ to fit the $(n-k)2^{n-k}$ jobs with length 2^k. There are also exactly 2^{n-i} slots of length 2^i for

every $0 \leq i \leq k-1$ to fit the medium jobs.
Finally, it will create a second copy of S_n, $S_n(e(S_n(0))$ to fit all the unit jobs. These 2^n jobs all have waited for at least $e(S_n(0))$ and therefore:

$$\text{COST(B)} \geq \sum_{i=1}^{2^n} w_i \geq 2^n \cdot n2^n = n2^{2n}$$

Now SPT will pay $C_1 + C_2 + C_3$ where C_1, C_2 and C_3 are the total completion times of the small, medium and large jobs respectively. Unit length jobs have total completion time:

$$C_1 = \sum_{i=1}^{2^n} i = 2^n(2^n+1)/2 \leq 2^{2n+2}$$

Now a medium jobs has been waiting for the unit jobs plus the shorter medium jobs before it plus the same size jobs before it. So the $j-th$ job of size 2^i has total completion time

$$\begin{aligned} C_{ji} &= 2^n + \sum_{m=0}^{i-1} 2^{n-m}2^m + (j-1)2^i + 2^i \\ &= 2^n + i2^n + j2^i \end{aligned}$$

So:

$$\begin{aligned} C_2 &= \sum_{i=0}^{k-1}\sum_{j=1}^{2^{n-i}} C_{ji} = \sum_{i=0}^{k-1}\sum_{j=1}^{2^{n-i}} 2^n + i2^n + j2^i \\ &= \sum_{i=0}^{k-1}(i+1)2^{2n-i} + 2^i\sum_{j=1}^{2^{n-i}} j = \sum_{i=0}^{k-1}(i+1)2^{2n-i} + \frac{2^{n-i}(2^{n-i}+1)}{2} \\ &\leq \sum_{i=0}^{k-1}(i+2)2^{2n-i} = 2^{2n}\cdot 2^{1-k}(-k+3\cdot 2^k-3) \leq 6\cdot 2^{2n}, \quad \forall k > 0 \end{aligned}$$

Large jobs, have been waiting for short and medium jobs to finish and for the

previous jobs of the same size.

$$C_3 = 2^n + \sum_{i=1}^{(n-k)2^{n-k}} k2^n + (i-1)2^k + 2^k = (n-k)k2^{2n-k} + \sum_{i=1}^{(n-k)2^{n-k}} i2^k$$
$$= (n-k)k2^{2n-k} + \frac{(n-k)2^{n-k}((n-k)2^{n-k}+1)}{2}2^k$$
$$= (n-k)k2^{2n-k} + \frac{(n-k)^2 2^{2n-k} + (n-k)2^{n-k}}{2} \leq (n-k)k2^{2n-k} + (n-k)^2 2^{2n-k}$$
$$= n(n-k)2^{2n-k}$$

So by summing all over the 2^n unit jobs we get:

$$\text{COST(OPT)} = C_1 + C_2 + C_3 \leq 2^{2n+2} + 6 \cdot 2^{2n} + n(n-k)2^{2n-k} = 2^{2n}[10 + \frac{n(n-k)}{2^k}]$$

3.1.3 Dynamic Menu Scheduling algorithm description

We introduce the following notation (as in [23]):

- An interval sequence $S_k(t)$ is a sequence of intervals whose lengths follow the recursive structure S_k specified in 3.1 that starts at timestep t.
  Example:

$$S_2(2) = \langle \underbrace{[2,3]}_{1}, \underbrace{[3,4]}_{1}, \underbrace{[4,6]}_{2}, \underbrace{[6,7]}_{1}, \underbrace{[7,8]}_{1}, \underbrace{[8,10]}_{2}, \underbrace{[10,14]}_{4} \rangle$$

- A^j is a vector of length l_j made off disjoint, ordered by increasing order of starting time interval sequences $A^j_{l_i} = S_{k_i}(t_i)$, i.e:

$$A^j = \langle A^j_{l_1}, \ldots, A^j_{l_j} \rangle$$

  where A^j_i is fixed for every $i < l_j$ while $A^j_{l_j}$ may be subject to change.
  State A^j represents every's machine's division (uniform for all of them).
  We refer to $A^j_{l_j}$ as the *tentative slot* of state A^j.

- For every interval sequence A_i we denote with $b(A_i)$ and with $e(A_i)$ its start and end respectively.
- X^j is a vector that keeps track of the intervals that the first j processes occupied.
- I(j) is the interval that process j chooses to be scheduled
- r_j is the release time of job j (basically its arrival time) and c_j is the completion time (that is dependent on the scheduling algorithm)

If we know the maximum length $p_{\max}$ then we can use a simple algorithm:

> *Schedule every process i that comes to the system in the first slot that fits (on any machine). If no such one exists, create a new copy of the pattern.*

Otherwise, a more sophisticated version is needed:

> *When the $j-th$ process comes to the system : State vector A^{j-1} is presented to process j. Now process j will choose an available interval $I(j)$ that is either in $A^{j-1}_{l_{j-1}}$ or in $A^j_{l_j}$ (further explained below).*

We assume that every process will choose the first available slot (that respects the rules). Assume process j has length 2^k.

Obviously, if $r_j \geq e(A^{j-1}_{l_{j-1}})$, then we fix $A^{j-1}_{l_{j-1}}$ and introduce a new tentative slot $A^j_{l_j} = S_k(r_j)$.

Else if $r_j < e(A^{j-1}_{l_{j-1}})$:
- If the tentative slot $A^{j-1}_{l_{j-1}} = S_d(t_{l_{j-1}})$ is occupied such as we cannot break it to a $S_q(t_{l_{j-1}}), q < d$ without leaving some processes without slots, then we introduce $A^j_{l_j} = S_k(e(A_{l_{j-1}}))$ (so we extend the current state by the smallest sequence that can serve j).
- Else, find the smallest prefix $S_q(t_{l_{j-1}})$ of $A_{l_{j-1}}$ such that if we replace $A^{j-1}_{l_{j-1}}$ with

Introducing a new tentative slot

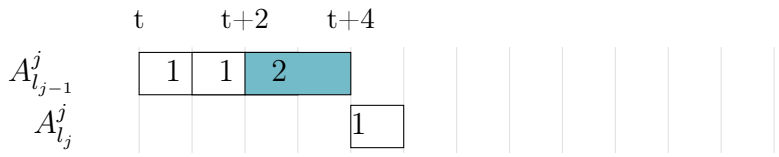


Figure 3.1: $r_j = 2$, $p_j = 1$, *blue* is occupied

Expanding the current tentative slot

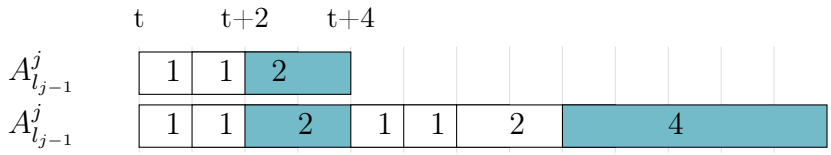


Figure 3.2: $r_j = 2$, $p_j = 4$, *blue* is occupied

S_q we still serve all the processes in the current $A_{l_{j-1}}$ plus process j.

These 4 cases are summarized in table 2 of [23].

	ℓ_j	$A^j_{\ell_j}$	$c_j \leq e\left(A^{j-1}_{\ell_{j-1}}\right)$	$r_j \geq e\left(A^{j-1}_{\ell_{j-1}}\right)$	$A^{j-1}_{\ell_{j-1}} = S_d(t)$ $k \leq d$
1	ℓ_{j-1}	$A^{j-1}_{\ell_{j-1}}$	True	-	-
2	$\ell_{j-1}+1$	$S_k(r_j)$	False	True	-
3	$\ell_{j-1}+1$	$S_k\left(e\left(A^{j-1}_{\ell_{j-1}}\right)\right)$	False	False	True
4	ℓ_{j-1}	$S_k\left(b\left(A^{j-1}_{\ell_{j-1}}\right)\right)$	False	False	False

Example on how to read the table:
In row 1 there is an interval in $A^{j-1}_{l_{j-1}}$ so that the completion time of process j is before the end of $A^{j-1}_{l_{j-1}}$ and therefore we do not increase the length of the current state ($l_j = l_{j-1}$) neither introduce a new tentative slot ($A^j_{l_j} = A^j_{l_{j-1}}$).
In all the other cases, there is not a slot in $A^{j-1}_{l_{j-1}}$ to fit process j without any modification ($\{c_j \leq e(A^{j-1}_{l_{j-1}})\}$ =False).
In row 2, process j arrives after the end of $A^{j-1}_{l_{j-1}}$ so we introduce a new tentative

slot $S_k(r_j)$. In row 3 and 4 the process does arrive before the end of $A^{j-1}_{l_{j-1}}$ ($\{r_j \geq e(A^{j-1}_{l_{j-1}})\}$ =False) but because in row 3 there is no way to extend $A^{j-1}_{l_{j-1}}$ so as to respect the structure we need to introduce a new tentative slot while in row 4 there is a way to expand $A^{j-1}_{l_{j-1}}$ (if $A^{j-1}_{l_{j-1}}$ was $S_k(t)$ we can find a $d \geq k$ so as $S_k(t)$ is a prefix of $S_d(t)$).

Theorem: The dynamic scheduling algorithm is $O(\log p_{\max})$ where $p_{\max}$ is the maximum length of any process in the input.

Note: In their analysis they compare their algorithm with $SRPT$ which is a 2-approximation in the single machine setting and a 3-approximation in the identical machines ([5]).

What is interesting here is that when $P_{\max}$ is known this dynamic algorithm with the menu is equivalent to considering a static algorithm in the simple setting (where agents have no choice) that uses S_d where $d = \lceil \log P_{\max} \rceil$ as a base-pattern.

3.1.4 Lower Bound on any prompt algorithm

Having demonstrated that the two static algorithms suffer a $\Omega(\sqrt{P_{\max}})$ competitive ratio, we now turn our attention to the dynamic algorithm, which achieves $O(\log P_{\max})$. Naturally, we wonder whether this performance is optimal. Indeed, Fiat et al. proved the following result:

Any prompt online scheduling algorithm for processes with unit weights and arbitrary lengths must have a competitive ratio of $\Omega(\log P_{\max})$, even if randomization is allowed.

3.1.5 Other settings for prompt scheduling

In the same work, they provide algorithms for arbitrary weights and uniform processing times as well as the most general case with both arbitrary weights and

processing times. A summary of results can be found in the following table:

Processing Time	Job Weight	Menu Entries	Upper Bound (Deterministic)	Lower Bound (Randomized)
$p_j \in \mathbb{Z}^+$	$\mathrm{w}_j = 1$	intervals (various lengths) no prices	$O(\log P_{\max})$	$\Omega(\log P_{\max})$
$p_j = 1$	$w_j \in \mathbb{Z}^+$	unit length intervals with prices	$O(\log W_{\max}(\log\log W_{\max} + \log n)$	$\Omega(W_{\max})$
$p_j \in \mathbb{Z}^+$	$w_j \in \mathbb{Z}^+$	intervals (various lengths) with prices	$O((\log n + \log P_{\max}) \log B_{\max})$	$\Omega(\max(\log P_{\max}, \log B_{max}))$

where $p_{\max}$ and $W_{\max}$ is the maximum length and weight in the input sequence respectively (not known apriori) and $B_{\max}$ is an apriori upper-bound on $W_{\max}$.

Chapter 4

Learning how to schedule with promptness

In this chapter we focus again on online non-preemptive prompt scheduling to minimize total completion time. The algorithm from section 3.1.3, that uses a fixed pattern is an $O(\log(p_{\max}))$ competitive algorithm. In this work, we challenge the notion of a single, universally optimal pattern and explore the potential for identifying alternative patterns that lead to a smaller total completion time for the set of jobs.

4.1 Background on Online learning

"Online learning" is an umbrella term for problems where data points arrive over time while an agent has to make decisions based solely on past observations. Online learning algorithms are a type of online algorithm, but their performance is assessed using a different metric called "Regret" (to be defined shortly after). Typically, we study online learning under the online convex optimization setting (OCO). Formally:

Setting:

- Learner: Makes decisions represented by actions denoted as x_t at each round

t. These actions belong to a convex and non-empty subset $\mathcal{X}$ in d-dimensional Euclidean space ($\mathbb{R}^d$).

- Environment: Chooses a convex, non-negative cost function f_t that maps elements of X to real numbers ($\mathbb{R}$) at each round t.

Interaction: At each round t:

- The learner chooses an action x_t from the set $\mathcal{X}$.
- The learner incurs a loss $f_t(x_t)$
- The environment reveals f_t to the learner.

Terminology:

- Full Feedback (or Full-Information): This framework is categorized as "full feedback" because the learner observes the complete loss function f_t, not just the specific loss associated with its chosen action.
- Oblivious vs. Non-Oblivious Environment:
  - Oblivious Environment: The environment's choice of the loss function f_t is independent of the learner's action x_t. In simpler terms, the environment's loss function selection doesn't "react" to the learner's choices.
  - Non-Oblivious Environment: The environment's choice of the loss function f_t can depend on the learner's action x_t. Here, the environment can potentially adapt the loss function based on the learner's behavior.

At this point, it is vital to define how optimal is defined in an online setting.
The sequential nature of the online setting allows for the possibility of generating inputs that specifically challenge the decision-maker's performance. To establish, a meaningful performance metric, the online learning community adopted a concept from game-theory, typically referred in the literature as regret. Let X be a set of feasible choices, $f_t : X \to R$ be a convex loss function and T be discrete a time

horizon at which inputs will occur. The **"Regret"** of an algorithm $\mathcal{A}$, is defined as the difference between the sum of the losses $f_t(x_t)$ at each timestep, where x_t is $\mathcal{A}$'s choice at time $t \in \{1, .., T\}$ and the sum of the losses incurred by the best fixed choice over X:

$$\text{Regret}(T) = \sum_{t=1}^{T} f_t(x_t) - \min_{x \in X} \sum_{t=1}^{T} f_t(x)$$

A good online algorithm should aim to have as small regret as possible, ideally a 'sublinear' one, that is Regret(T) $\in o(T)$ i.e $\lim_{T \to \infty} \frac{R(T)}{T} = 0$.
For simplicity, we assumed minimization problems. However, the very same analysis holds for maximization problems with the necessary adjustments.

4.1.1 Follow the Leader Family

Follow the Leader (FTL) refers to a family of algorithms, that make decisions based on the best strategy.
Perhaps the best way to understand this framework is the "Experts setting". In this setting, the learner may or may not have direct access to the action set $\mathcal{X}$. Instead of relying solely on direct access to available actions, the learner depends on the advice of "experts." These experts are external entities that can guide the learner.
Formally, assume that there are d experts. At each round, we need to pick an expert $j_t \in [d]$. Let $a_t(j_t)$ denote the expert's j_t advice on round t (e.g a point $x \in \mathcal{X}$ or a distribution over $\mathcal{X}$) . After making this choice, the learner follows j_t expert advice. Let f_t be a mapping from experts advice to losses on round t).
The basic version examines the suggestions of each expert on all the rounds before t and chooses the one that performed better. This can be formally expressed as:

$$x_t = \arg\min_{j \in [d]} \sum_{\tau=1}^{t-1} f_t(a_t(j)), \quad \forall t \in [T]$$

Now, a subtle proof can showcase that this algorithm may have linear regret for a

special input sequence.
For simplicity assume that there are only two experts suggesting actions each day, one of the actions receives a loss of zero (i.e., the "correct" suggestion) and the other a loss of one.

Round	**1**	**2**	**3**	**4**	**5**	**....**	**T-1**	**T**
Loss of expert 1	1/2	0	1	0	1	....	0	1
Loss of expert 2	0	1	0	1	0	....	1	0
Leader	-	2	1	2	1	....	2	1

After round 1, FTL incurs always cost 1 so at least $T-1$ in total while the best expert in hindsight incurs at most T/2 (if T is even) so the regret is $R(T) \geq T-1-T/2 = T/2-1 \in O(T)$.

To combat this, a strongly convex function $R(x)$ is added that serves as a stabilization term. This slightly different formula is named Follow-The-Regularized Leader(FTRL) :

$$x_t = \arg\min_{x \in \mathcal{X}} \sum_{\tau=1}^{t-1} \Big(f_t(x) + R(x) \Big), \quad \forall t \in [T]$$

Under the assumptions $|\mathcal{X}| \leq \mathcal{B}$ and f_t is a convex L-Lipschitz we get that FTRL satisfies the following inequality:

$$R(T) \leq \sum_{t \in T} [f_t(x_t) - f_{t+1}(x_t)] + R(x_T) - R(x_1)$$

Now if R is $\frac{1}{2\eta}||x||^2$ (an $1/2\eta$ strongly-convex function) and T is known in advance, FTRL achieves a $O(\mathcal{B}L\sqrt{2T})$ regret bound. Similar results hold even without knowledge of T (by using a time-varying learning rate η_t).
In the same spirit, introducing randomness can make the algorithm less prone to abrupt behavior changes. One way to do so is to append some noise established as "The Perturbed Leader" back in 2003 ([15]).
Over the years, many other improvements were published, the majority of them

focusing on choosing an appropriate regularizer. For a complete introduction to online learning, optimization and the relevant algorithms we refer the reader to [25, 28, 21].

4.2 Problem Modeling

Assume we divide the input in T rounds. Let N^t be the set of processes that arrived in round t. If in every round $t \in [T]$ we use the algorithm from section 3.1.3 then we have an $O(\log(p_{\max}))$ competitive algorithm. Now, we raise the question: Is there a pattern that achieves a smaller total completion time? Can we perhaps learn from past observed input sequences to devise a pattern more tailored to the specific data it encounters? Our goal is going to be to minimize the regret, defined as:

$$R(T) = \sum_{t=1}^{T} f_t(a^t) - \sum_{t=1}^{T} f_t(a^*)$$

Unlike the work of [23] that involves strategic agents in the absence of knowledge of $p_{\max}$, we assume an equivalent setting where the scheduling algorithm dictates the execution order and informs each job of its service time. This assumption simplifies the problem and allows us to focus on the learning aspects of scheduling.

Notation Summary

Symbol	Meaning
N	the set of all inputs (processes)
T	the total number of rounds
p_i	the processing time of process i
r_i	the arrival time of process i
P_i	the length of slot of type i

Defining the notion of “round"

Evaluating a pattern’s effectiveness requires the underlying algorithm to process data for a sufficient amount of time. A natural choice is to define a round with respect to a time window of a certain length w. Each round represents a specific

time interval with a fixed length, denoted by w.
Rounds:

- Round 1: $[0, w)$ - Processes arriving between time 0 and w (exclusive) are considered in this round, denoted by N^1.
- Round 2: $[w, 2w)$ - Processes arriving between time w and 2w (exclusive) belong to round 2, and so on.
- Round t: $[w(t-1), tw)$ - Round t consists of processes with arrival times within the time interval [w(t-1), tw) (exclusive).

Assuming the arrival times follow some distribution we can tune w appropriately. Otherwise, this time-window w may not be really meaningfull: there may be rounds with very few processes ("sparse rounds") and rounds with infinitely many processes ("overloaded rounds"). In the first case, we cannot pay much more than wO (where O is the optimal period). In the second case we may pay a lot more than other ("less crowded" rounds) -especially if the pattern is bad- but these costs will cancel out with the minimal cost of the rounds that fall into the first case.

Defining the loss function per-round

Unlike common online learning settings, here we need to be careful, because depending on the choice of a^t and time-window w, there is a chance that processes from the next round $t+1$ arrive before finishing with the input of round t. To resolve this issue we define as $f_t(a^t)$ the total completion time of all the processes in N^t using as a basic pattern a^t plus any process from N^{t+1} that happened to arrive before the completion of N^t (due to the promptness constraint we cannot do otherwise). This additional workload can potentially increase the cost associated with the chosen pattern a^t for that round.On the positive side, if a^t is a "poor" choice for N^t since the algorithm will incur significant costs on this round, it will also learn that this sequence was not the optimal choice. In the long-run, when T is large enough, this cost will not affect $\frac{R(T)}{T}$.

Assumptions

In our model, we will assume that:

1. Processing times take integer values in $\{1, 2, .., 2^l\}$ for some $l \in \mathbb{N}$ (this assumption makes the analysis easier while at the same time considering the real job lengths adds up only a constant to the computed cost) and weights are identical (e.g $w_j = 1$)

2. We are looking for periodic non-decreasing sequences

Since processes's lengths are assumed to be powers of 2 then also slot lengths are powers of this which offers an extra benefit. It enables us to use an alternative sequence representation and therefore reduce the search space substantially: Instead of looking for vectors $a \in \mathbb{N}^{p_{\max}}$ where a_i is the length of the i-th slot, we are looking for vectors $a \in \mathbb{N}^l$ where $l = \lceil \log p_{\max} \rceil$ where $a_i \in [0, l]$ represents the number of slots of length 2^i in the sequence.
In other words, we are looking for a sequence P defined as

$$P = \langle p_1, \dots p_1, \dots, p_l, \dots p_l \rangle$$

that can be reduced into a vector a defined as:

$$a = (a_1, \dots a_l)$$

where a_i represents the number of slots of length p_i in P such that $P = \langle p_1^{a_1}, \dots, p_l^{a_l} \rangle$.

4.2.1 Convex program formulation

Assume we know that the optimal period is L and that the maximum length of any process that we will serve is $P_{\max}$ then without loss of generality the optimal sequence is made from $l = \lceil \log_{P_{\max}} \rceil$ different types/lengths of slots $\{1, 2, \dots, 2^l\}$. Therefore we are looking for a sequence $a \in \mathbb{N}^l$. Denote by r_i the arrival time of

process i.

$$
\begin{aligned}
\text{minimize} \quad & C^1(y) \\
\text{subject to} \quad & \sum_{j=1}^{l} a_j P_j \leq L && (1) \\
& \sum_{r_i \leq tL} y_{it} \geq 1, \qquad \forall i \in N && (2) \\
& \sum_{i \in S^j} y_{it} \leq \sum_{k=j}^{l} a_k, \qquad \forall j \in [l],\ \forall t \in T && (3) \\
& y_{it} \in [0,1], \qquad \forall i \in N, \forall t \in T && (4) \\
& a_i \in [0, \lfloor L/P_i \rfloor] \qquad \forall i \in [l] && (5)
\end{aligned}
\tag{C^1}
$$

where $C^1(y)$ is an approximation of the total completion time using a periodic schedule y (defined in 4.2.1 in the next section).

Explaining the formulation:

Variables:

- Binary variable y_{it} denotes whether process i was scheduled on the t-th sequence.
- The variable $a_j \in N$ denotes the number of slots of type j in the basic pattern.

Constraints:

- Constraint (1) ensures that the sequence a will have a length (duration) of at most L.
- Constraint (2) ensures that if a process i is scheduled in the t-th sequence, it arrives no later than the end of this sequence. However, this constraint does not account for processes that might have arrived after the start time of their assigned slot, possibly underestimating their completion time. We

account for that through an error-correcting term in the objective function (see paragraph 4.2.1).

- Constraint (3) ensures that if S^j is the set of processes with size at most P_j (i.e., $S_j = \{i \mid i \in [N] : p_i \leq P_j\}$), these processes cannot occupy more than the sum of slots in a that can fit them (i.e., slots greater than or equal to their sizes). For example, if we consider $i = \arg\max P_i$, then S_i contains all the input processes, and we ensure that the number of processes scheduled in the t-th sequence does not exceed the available slots in the sequence.

Relation between the objective and the real function

If $F(y)$ is the real total completion time (TCP) using a periodic schedule y, we want to relate : our estimation of TCP with $\min_y F(y)$.
Let w_i^r denote the real-time step at which process i begins processing according to a solution y to C^1: break it into the waiting time because of the previous sequences plus the waiting time due to the waiting time $r(i)$ due to the relative order we schedule i inside a copy of the sequence.

$$y_{it} w_i^r = y_{it}[(t-1)L + r(i)]$$

Now let's consider an approximation w_i for the waiting time of process i defined as:

$$w_i = (t-1)L \tag{4.1}$$

This is an underestimation for two reasons:

1. We ignore the fact that the relative waiting time $r(i)$ may not be 0
2. We ignore the fact in our set of constraints $\mathcal{C}$, we have the constraint: $\sum_{t:\, r_i \leq tL} y_{it} \geq 1$ which assumes that process i should be scheduled on the $t-th$ schedule that ends at tL if and only if $r_i \leq tL$. However, this does not take into consideration that the process may be scheduled on a slot j that starts earlier than r_i.

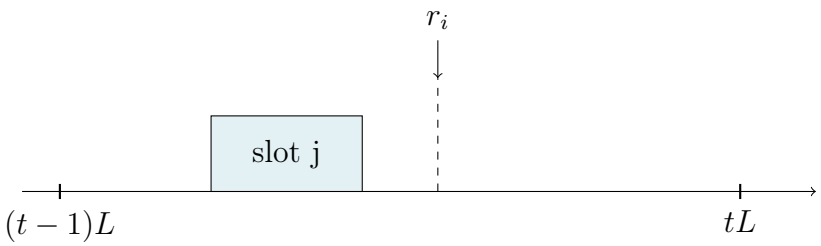


Figure 4.1: process i arrives after its assigned slot

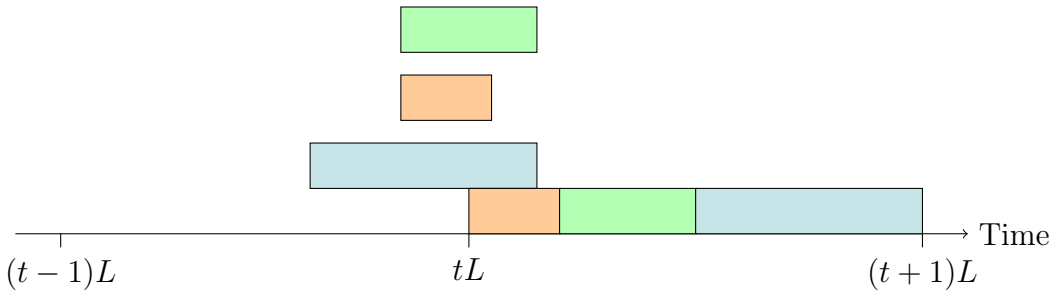
These two assumptions are independent. To see why, consider the case where all of the processes we scheduled on the t-th sequence arrived exactly at its start (so we do not need to worry about the second bullet). That still does not guarantee that all the processes had relative waiting time $r(i) = 0$ (in fact that only can happen if only one process was scheduled on the t-th sequence).
So if w_i^r is the real waiting time of a process i that was scheduled according to the solution y to C^1:

$$w_i^r = \tilde{w}_i + \epsilon_1 + \epsilon_2$$

where $\epsilon_1 =$ *error due to release times* and $\epsilon_2 =$ *error due to* $r(i)$
We address them as follows:

First, we know that $0 \leq r(i) \leq L$, so $0 \leq \epsilon_2 \leq L$

As for handling the release times we observe that in the worst-case scenario every process i that was scheduled on t arrived on the end of it, that is $r_i \approx [(t+1)L - \epsilon]$. But since all these processes did fit on a copy of the sequence and had $r_i < (t+1)L$ they can be scheduled on the $(t+1)$-th sequence, so $\epsilon_1 \leq L$. Finally, in the best-case scenario, it arrived no later than its start time (that is $\epsilon_1 = 0$). Therefore: $0 \leq \epsilon_1 \leq L$



So in reality

$$w_i \leq w_i^r \leq w_i + 2L \tag{4.2}$$

Summing over all processes and rounds:

$$F(y) = \sum_{i \in N} \sum_{t \in T} (w_i^r + p_i) y_{it} = \sum_{i \in N} \sum_{t \in T} w_i^r y_{it} + \sum_{i \in N} \sum_{t \in T} p_i y_{it}$$

Name $P(y) = \sum\limits_{i \in N} \sum\limits_{t \in T} y_{it} p_i$. Now using (2) and summing over N and t, we get :

$$\sum_{i \in N} \sum_{t \in T} y_{it} w_i + P(y) \leq F(y) \leq \sum_{i \in N} \sum_{t \in T} y_{it} w_i + \sum_{i \in N} \sum_{t \in T} y_{it} 2L + P(y)$$

$$\iff \sum_{i \in N} \sum_{t \in T} y_{it}(t-1)L + P(y) \leq F(y) \leq \sum_{i \in N} \sum_{t \in T} y_{it}(t-1)L + 2LN + P(y) \tag{2}$$

where in the last equation we substituted w_i with it's definition and used the fact that $\sum\limits_{i \in N} \sum\limits_{t \in T} y_{it} = N$.
Now consider the CP with objective function :

$$C^1(y) = \sum_{i \in N} \sum_{t \in T} y_{it}[(t-1)L] + \sum_{i \in N} \sum_{t \in T} y_{it} p_i \qquad \text{(Objective of program } C^1\text{)}$$

Now by substituting on (2), we can write:

Lemma 4.2.1:

For any feasible solution y to program C^1 it holds:

$$C^1(y) \leq F(y) \leq C^1(y) + 2LN \quad (A)$$

Discussion on the LP

Minimizing function C^1 gives an underestimation of the total completion time. While it considers how long a process waits before its assigned sequence starts, it doesn't consider any additional waiting time that might occur within the sequence itself (also as it was explained earlier the constraints of the convex program assume that the process release time was before the start time of the slot it was scheduled, which is not always the case). Another way to put it is to say that the objective function is oblivious to the troubles introduced by trying to arrange the processes with release dates in a copy of the sequence. Essentially, this program returns an arrangement of the processes such that they fit in terms of lengths in their assigned copy, taking only partially into consideration their release dates (by constraint (1) a process will not be assigned to a sequence t that ends before the process arrives in the system).

In addition, it is a relaxation since we consider non-preemptive scheduling and therefore $y_{it} \in \{0, 1\}$ and also integer sequences i.e $a_j \in N^l$ (*the number of slots* a_j *of type* j *that belong to sequence* a *is an integer*).

4.2.2 Rounding and Approximation

Approximation of the rounded solution

Assumption We know the optimal period L (that is the period the optimal (offline) algorithm using a fixed sequence would use to schedule the entire input N).

Lemma 4.2.2:

Assume $C^1(y^\sigma)$ is (an estimation according to $C^1()$ of) the total completion time using pattern $\sigma \in R^d$ on an input sequence N. Now let $a = \lceil \sigma \rceil$ be the rounded pattern, then:

$$C^1(y^a) \leq 3C^1(y^\sigma), \quad \forall y^\sigma \in \{0,1\}^{|N|}, a = \lceil \sigma \rceil$$

Proof)

When we round σ to a we do not change the schedule y^σ but only the length of the period. If the initial period is L then for the new period L' we have:

$$L' = \sum_{i=1}^{l} a_i P_i \leq \sum_{i=1}^{l} (\sigma_i + 1) P_i \leq L + \sum_{i=1}^{l} P_i$$

We are going to prove that :

If a^* is the optimal pattern of integers then

$$a_l^* > 0 \implies \sum_{i=1}^{l} P_i \leq 2L$$

Proof):

$$\sum_{i=1}^{n-1} P_i = \sum_{i=1}^{n-1} 2^i = 2^n - 2 < 2^n = P_n, \quad \forall n \in \mathbb{N}$$

Therefore if $a_l^* > 0 \implies L^* \geq P_l > \sum_{i=1}^{l-1} P_l$

$$\sum_{i=1}^{l} P_i = \sum_{i=1}^{l-1} P_i + P_l < 2L$$

So for the new period L' we get:

$$L' < 3L \tag{4.3}$$

Let $P = \sum_{i=1}^{l} P_i$. The total completion time is:

$$\begin{aligned} C^1(y^a) &= \sum_{\tau=1}^{T}\sum_{i=1}^{N}[(\tau-1)L' + p_i]y^a_{it} = \sum_{\tau=1}^{T}\sum_{i=1}^{N}[(\tau-1)L' + p_i]y^\sigma_{it} \\ &< 3\sum_{\tau=1}^{T}\sum_{i=1}^{N}[(\tau-1)L + p_i]y^\sigma_{it} \\ &= 3C^1(y^\sigma) \end{aligned}$$

Here we used the fact that (offline) we can schedule the input greedily using a as well as using σ (since a is only “an augmentation" of σ).

Theorem 1

If (σ, y^σ) is any feasible (fractional) solution of (C^1) then for the schedule y^a induced by $a = \lceil \sigma \rceil$ it holds:

$$F(y^a) \leq 3C^1(y^*) + 2LN \leq 3F(y^*) + 2LN$$

where y^* is an optimal solution

Proof):
Let y^* be the schedule that occurs from the optimal periodic scheduling.

$$\begin{aligned} F(y^a) &\leq C^1(y^a) + 2LN, && \text{using inequality (A) 4.2.1} \\ &\leq 3C^1(y^\sigma) + 2LN, && \text{using lemma 4.2.2} \\ &\leq 3C^1(y^*) + 2LN, && \text{since } y^\sigma \text{is a minimizer of } C^1 \\ &\leq 3F(y^*) + 2LN && \text{by definition of } C^1(y) \end{aligned}$$

Corollary: Assuming that our observing window w is tuned properly with respect to the distribution of arrival times:

$$F(y^a) \leq 4F(y^*)$$

Imagine a partition of the time horizon of interest into intervals of lengths w. As we said earlier we reevaluate the pattern every w units of time and we do that T times. Since L is optimal it may be the case that we need exactly one copy of the pattern at each round (we can cover wT perfectly with intervals of length L). Even in that case, the optimal prompt algorithm incurs a total completion time $F(y^*) \geq \sum_{i=1}^{N}(i-1)L = N(N-1)L/2 > 2LN$ (assuming $N \geq 3$) units of time.
Therefore we get:

$$F(y^a) \leq 4F(y^*)$$

4.3 The online scheduling algorithm

Let $C_t^1(y^\sigma) = \sum_{\tau=1}^{n} \sum_{i \in N^t} y^\sigma(\tau - 1)L$ be the cost of the "underestimation LP" (C^1) for the input sequence N^t, using the pattern σ and a schedule y^σ that respects σ.

Note that we have two different notions of rounds in this context. First, there are the T rounds defined by the observation windows, w, during which we reconsider the pattern. Second, within each such round t, we need $n = n(t)$ rounds/copies of the pattern σ to schedule the input N^t.

Algorithm 2 Online Prompt Scheduling Algorithm

Input: Period L, maximum processing time $p_{\max}$, regularizer $U : R^l \to R$
Output: A prompt schedule $(y^{\sigma_1}, \ldots, y^{\sigma_T})$

1: Set $l \leftarrow \log p_{\max}$ (for formulating C^1 constraints)
2: **for** $t \in [T]$ **do**
3: Compute $(\tilde{y}, \tilde{\sigma}_t) = \arg\min_{y,\sigma} \left(\sum_{\tau=1}^{t-1} C_\tau^1(y_\tau^\sigma) + \frac{U(\sigma)}{\eta} \right)$
4: Round $\tilde{\sigma}_t$ to $a_t = \lceil \tilde{\sigma}_t \rceil$
5: Schedule processes (greedily) in N^t using a_t
6: **end for**

To analyze this algorithm effectively, observe that it encompasses three distinct components: a learning algorithm, a rounding algorithm, and a scheduling algorithm.

Measuring Perfomance

We want to compare $F_t(y_t^{a_t})$ with the optimal real cost $F_t(y_t^*)$ of the best fixed sequence σ^*. Using Theorem 1 () we get that the real cost $F_t(y_t^{a_t})$ is at most $4C_t^1(y_t^{\sigma_t}) + L$. We will use an alternative measure of regret, namely $c - regret$ as defined in relevant cases ([29], ...). For $c = 1$ the measure coincides with the "standard" notion of regret. In particular, we define as:

$$R(T) = \sum_{t=1}^{T} F_t(y_t^{a_t}) - c \sum_{t=1}^{T} F_t(y_t^{p^*})$$

where p^* is the optimal fixed adversary i.e $p^* = \arg\min \sum_{t=1}^{T} F_t(y_t^{p^*})$.
We also define:

$$\tilde{R}(T) = \sum_{t=1}^{T} C_t^1(y_t^{a_t}) - c\sum_{t=1}^{T} C_t^1(y_t^{\sigma^*})$$

where (σ^*, y^{σ^*}) is the solution to the convex program with objective function $\min_{y,\sigma} \sum_{t=1}^{T} C_t^1(y^\sigma)$ and the same constraints as before.

Lemma 4.3.1:

If $R(T)$ is the c-regret defined above then for $c = 4$ it holds:

$$R(T) \leq 3\tilde{R}(T)$$

Proof:

First observe that using Collorary 4.2.2 we have that: $\sum_{t=1}^{T} F_t(y_t^{p^*}) \geq 2LN$ where p^* is the optimal integer pattern in the hindshight. In addition since C_t^1 is a lower bound for F_t and σ^* is the minimizer of $\sum_{t=1}^{T} C_t^1(y^{\sigma^*})$ it holds:

$$\sum_{t=1}^{T} F_t(y_t^{p^*}) \geq \sum_{t=1}^{T} C_t^1(y_t^{p^*}) \geq \sum_{t=1}^{T} F_t(y_t^{\sigma^*})$$

Now we have that :

$$\begin{aligned}
R(T) &= \sum_{t=1}^{T} F_t(y_t^{a_t}) - 3\sum_{t=1}^{T} F_t(y_t^{p^*}) - \sum_{t=1}^{T} F_t(y_t^{p^*}) \\
&\leq \sum_{t=1}^{T} [3C^1(y_t^{a_t}) + 2LN^t] - 3\sum_{t=1}^{T} C_t^1(y_t^{p^*}) - \sum_{t=1}^{T} F_t(y_t^{p^*}), \quad \text{from Theorem 1 (4.2.2)} \\
&= 3\sum_{t=1}^{T} [C^1(y_t^{a_t}) - C_t^1(y_t^{\sigma^*})] + [2LN - \sum_{t=1}^{T} F_t(y_t^{p^*})] \\
&\leq 3\tilde{R}(T)
\end{aligned}$$

The learning algorithm

In each round t, our proposed algorithm utilizes the well-known Follow-The-Regularized-Leader (FTRL) algorithm (refer to the relevant bibliography for more details) as a subroutine (step 3) to compute a fractional pattern σ_t. This pattern σ_t represents a variation of the best-fixed pattern from the previous $t-1$ rounds if we had computed the losses using our underestimation σ_t. The reason it does not match exactly with the best-fixed pattern from the previous $t-1$ rounds is that we include an additional term $U(\sigma)$ in the minimization problem. This term serves as a stabilization factor (see Section 4.1.1).

Let $R_u(T)$ be the regret FTRL incurs against a fixed-adversary u, i.e:

$$R_u(T) = \sum_{t=1}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=1}^{T} C_t^1(y_t^u)$$

Lemma 4.3.2: The FTL-BTL Lemma

Take any $u \in R^d$ and assume σ_t is the FTRL choice, then:

$$R_u(T) \leq \sum_{t=1}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=1}^{T} C_t^1(y_t^{\sigma_{t+1}}) + \frac{\max_{\sigma} U(\sigma) - \min_{\sigma} U(\sigma)}{\eta}$$

This lemma is due to Kalai and Vempala ([15]) but because of its importance, its proof has been rewritten many times. A standard proof can be found in section 4.4.2 of the Appendix.

Lipschitzness of the loss function

We can write :

$$C_t^1(y) = \sum_{i \in N} \sum_{\tau=1}^{n_t} [(\tau - 1)L + p_i] y_{i,\tau} = \langle A, y \rangle_F$$

where $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product between matrices $A \in R^{N^t} \times R^T$ and $y \in R^{N^t} \times R^T$ defined as:

$$A = \begin{bmatrix} p_1 & L + p_1 & .. & L(n_t - 1) + p_1 \\ p_2 & L + p_2 & .. & L(n_t - 1) + p_2 \\ \vdots & \vdots & .. & \vdots \\ p_{N^t} & L + p_{N^t} & .. & L(n_t - 1) + p_{N^t} \end{bmatrix} \text{ and } y = \begin{bmatrix} y_{1,1} & y_{1,2} & .. & y_{1,n_t-1} \\ y_{2,1} & y_{2,2} & .. & y_{2,n_t-1} \\ \vdots & \vdots & .. & \vdots \\ y_{N^t,1} & y_{N^t,2} & .. & y_{N^t,n_t-1} \end{bmatrix}$$

To be precise, assume $T^y(t)$ and $T^y(t)$ is the number of copies needed to schedule input N^t using sequences x and y respectively. Now name $n_t = \max\{T^y(t), T^x(t)\}$ and assume without loss of generality that $T^x(t) < T^y(t)$. We can write $C_t^1(x) = A^T \cdot y_x$ by adding to y_x extra columns to expand it to horizon n_t with zero entries.

Lemma 4.3.3: F

unction $C_t^1(y^\sigma) = \langle A, y \rangle_F$ is $\|A^T\|_F$ Lipschitz with respect to any norm, where $\| \cdot \|_F$ is the Frobenious norm, i.e:
$\|A^T\|_F = (N^t)^{1/2} L n_t^{3/2} \sqrt{\frac{1}{3} - \frac{1}{2n_t} + \frac{1}{6n_t^2} + \frac{2}{N^t L} \sum_{i=1}^{N^t} p_i \cdot \left(\frac{1}{2n_t} - \frac{1}{2n_t^2} \right) + \frac{1}{N^t L^2 n_t^2} \sum_{i=1}^{N^t} p_i^2}$
where N^t is the input sequence on round t and $n_t = n_t(\sigma)$ is the number of copies of the pattern σ algorithm needed to schedule the input.

Proof):

$$|C_t^1(y) - C_t^1(x)|_p = |\langle A^T, (y - x) \rangle_F|_p \leq \left| \|A^T\|_F \cdot \|y - x\|_F \right|_p \leq \left| \|A^T\|_F \right|_p \cdot \left| |y - x|_F \right|_p$$

$$= \|A^T\|_F \cdot |\|y - x\|_F|_p$$

Now using the definition of the Frobenius norm 4.4.1 we compute:

$$\|A^T\|_F = \Big(\sum_{\tau=1}^{n_t}\sum_{i=1}^{N}|A_{i\tau}|^2\Big)^{1/2} = \Big(\sum_{\tau=1}^{n_t}\sum_{i=1}^{N}(\tau-1)^2L^2 + 2L(\tau-1)L + p_i^2\Big)^{1/2}$$

$$= \Big(NL^2\cdot\frac{n_t}{6}(2{n_t}^2 - 3n_t + 1) + 2L\sum_{i=1}^{N^t}p_i\cdot\frac{(n_t-1)n_t}{2} + n_t\sum_{i=1}^{N^t}p_i^2\Big)^{1/2}$$

$$= \Big(n_t^3NL^2[\frac{1}{3} - \frac{1}{2n_t} + \frac{1}{6n_t^2} + \frac{2}{N^tL}\sum_{i=1}^{N^t}p_i\cdot(\frac{1}{2n_t} - \frac{1}{2n_t^2}) + \frac{1}{N^tL^2n_t^2}\sum_{i=1}^{N^t}p_i^2]\Big)^{1/2}$$

$$= (N^t)^{1/2}Ln_t^{3/2}\sqrt{\frac{1}{3} - \frac{1}{2n_t} + \frac{1}{6n_t^2} + \frac{2}{N^tL}\sum_{i=1}^{N^t}p_i\cdot(\frac{1}{2n_t} - \frac{1}{2n_t^2}) + \frac{1}{N^tL^2n_t^2}\sum_{i=1}^{N^t}p_i^2}$$

Observe how $n_t \leq N^t$ i.e we cannot need more copies than the number of processes, so we can bound $\|A\|_F$ with a function of N^t, L only.

Lemma 4.3.4:

closeness Consider two functions $f, g : R^k \to R$ that are m-strongly convex with respect to some norm k and such that their difference $h(p) = g(p) - f(p)$ is an L-Lipschitz function with respect to the same norm. Then if $pf = \arg\min_{p\in R^k} f(p)$ and $p_g = \arg\min_{p\in R^k} g(p)$, it must hold that:

$$||p_f - p_g|| \leq \frac{L}{m}$$

Proof:

By definition of m-strong convexity we have that:

$$f(p) - f(p_f) \geq \langle\nabla f(p), p - p_f\rangle + \frac{1}{2m}\|p - p_f\|^2$$

But p_f is a minimizer of f and therefore $\langle\nabla f(p), p - p_f\rangle \geq 0$ and therefore we can

omit this term from the right-hand site:

$$f(p) - f(p_f) \geq \frac{m}{2}\|p - p_f\|^2, \quad (1)$$

The same holds for g:

$$g(p) - g(p_g) \geq \frac{m}{2}\|p - p_g\|^2, \quad (2)$$

Now :

$$\begin{aligned} h(p_f) - h(p_g) = (f(p_f) - g(p_f)) - (f(p_g) - g(p_g)) = (f(p_f) - f(p_g)) + (g(p_g) - g(p_f)) \\ \geq \frac{m}{2}\|p_f - p_g\|^2 + \frac{m}{2}\|p_g - p_f\|^2 = m\|p_f - p_g\|^2 \end{aligned}$$

But h is also Lipschitz :

$$\begin{aligned} m\|p_f - p_g\|^2 \leq h(p_f) - h(p_g) \leq L\|p_f - p_g\| \\ \implies \|p_f - p_g\| \leq \frac{L}{m} \end{aligned}$$

4.3.1 Regret analysis (Putting everything together)

Theorem 2

Algorithm 2 has sublinear 4-regret with respect to T.

Proof):

We proved $C_t^1(\sigma)$ is $M = \|a^T\|_F$ -Lipschitz. Name $\tilde{F}_t(\sigma) = \sum_{\tau=1}^{t} C_\tau^1(y_\tau^\sigma) + \frac{U(\sigma)}{\eta}$, where y_t^σ is the greedy schedule that occurs using σ. Assume $U(\sigma)$ is m strongly convex then since $\sum_{\tau=1}^{t} C_\tau^1(y_\tau^\sigma)$ is convex $\tilde{F}_t$ is m/η strongly convex on $\mathbb{R}^k$ (the sum of a convex function with an m-strongly convex function is an m-strongly convex function). In addition name $\tilde{\sigma}_t$ and $\tilde{\sigma}_{t+1}$ the choices of FTRL for rounds t and $t+1$ respectively. Since $\tilde{\sigma}_t = \arg\min_\sigma F_t(\sigma)$ and $\tilde{\sigma}_{t+1} = \arg\min_\sigma F_{t+1}(\sigma)$ are the minimizers of two strongly convex functions and their difference $h(\sigma) = C_{t+1}^1(\sigma)$ is M-Lipschitz, we can apply lemma **??**.

$$|\tilde{\sigma}_t - \tilde{\sigma}_{t+1}| \leq \frac{\eta}{m} \cdot M$$

Now we are ready for the final result:

- Name $\sigma^* = \arg\min_{\sigma \in R^k} \sum_{t=1}^{T} C_t^1(y_t^\sigma)$ and $\tilde{R}(T) = \sum_{t=1}^{T} C_t^1(y_t^{\tilde{\sigma}_t}) - \sum_{t=1}^{T} C_t^1(y_t^{\sigma^*})$. Then by the FTL-BTL lemma 4.3:

$$\tilde{R}(T) \leq \sum_{t=1}^{T} C_t^1(y_t^{\tilde{\sigma}_t}) - \sum_{t=1}^{T} C_t^1(y_t^{\tilde{\sigma}_{t+1}}) + \frac{\max_\sigma U(\sigma) - \min_\sigma U(\sigma)}{\eta}$$

- By combining lemma **??** with the fact that C_t^1 is M-Lipschitz we get:

$$\tilde{R}(T) \leq T \cdot M^2 \frac{\eta}{m} + \frac{\max_\sigma U(\sigma) - \min_\sigma U(\sigma)}{\eta}$$

- Name $D = \max_\sigma U(\sigma) - \min_\sigma U(\sigma)$. By picking $\eta = \sqrt{\frac{mD}{TL^2}}$, we get:

$$\tilde{R}(T) \leq M\sqrt{\frac{DT}{m}} + M\sqrt{\frac{DT}{m}} = 2M\sqrt{\frac{DT}{m}}$$

which is sublinear on T:

$$\frac{\tilde{R(T)}}{T} \to 0$$

- In lemma 4.3 we show that if $R(T)$ is the (real) c-regret then :

$$R(T) \leq 3\tilde{R}(T)$$

4.4 Conclusions and future directions

In this work, we expanded the idea of the algorithm from [23] to an online learning framework. We gained the insight that, unlike other classical computer science theory problems, scheduling demands extra effort to achieve a meaningful formulation as a learning problem. One main reason is that after making a single decision, its effect on cumulative losses is not immediately clear. Our model is only a first step in this direction. However, our work is promising for further exploration.
Scheduling problems can be further explored through the lens of learning theory. Specifically, for our problem, there are still many questions to study. Some examples include:

- How can we attain the optimal length L?
- How does the analysis change when this optimal length is not constant over time?
- What are the properties of inputs for which our algorithm has a constant competitive ratio?

These questions represent avenues for further research, aiming to deepen our understanding and improve the performance of online scheduling algorithms.

Appendix

4.4.1 Mathematical Background

Norms

$$\|A\|_1 = \max_{1\leq j\leq m} \sum_{1\leq n} |a_{ij}|$$

The Frobenious norm Let $A \in \mathbb{R}^m \times \mathbb{R}^n$ be an $m \times n$ matrix then we define it's Frobenius norm $\|A\|_F : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ as:

$$\|A\|_F = \Big(\sum_{j=1}^{n} \sum_{i=1}^{m} |a_{ij}|^2 \Big)^{1/2} = (\mathrm{Tr}(A'A))^{1/2}$$

4.4.2 FTRL-BTRL lemma

Let $R_u(T)$ be the regret FTRL incurs against a fixed-adversary u, i.e:

$$R_u(T) = \sum_{t=1}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=1}^{T} C_t^1(y_t^u)$$

Lemma 4.4.1:

Take any $u \in R^d$ and assume σ_t is the FTRL choice, then:

$$R_u(T) \leq \sum_{t=1}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=1}^{T} C_t^1(y_t^{\sigma_{t+1}}) + \frac{\max_{\sigma} U(\sigma) - \min_{\sigma} U(\sigma)}{\eta}$$

Proof):
Assume an imaginary round 0 where : $C_0^1(y) = \frac{1}{\eta}\min_\sigma U(\sigma)$. Now the FTRL algorithm we play can be written equivalently:

$$\sigma_t = \arg\min_\sigma \Big(\sum_{\tau=1}^{t-1} C_\tau^1(y) + \frac{1}{\eta} U(\sigma) \Big) = \arg\min_\sigma \sum_{\tau=0}^{t-1} C_\tau^1(y)$$

where y respects σ_t. Basically, we solve the convex program, and we find an appropriate tuple (y, σ_t).
By induction on T:

- For T=0: observe that $C_0^1(\sigma_1) = \min_\sigma C_0^1(y)$ therefore $C_0^1(\sigma_0) - C_0^1(u) \leq C_0^1(\sigma_0) - C_0^1(\sigma_1)$
- **Inductive Hypothesis:** Assume it holds for T:

$$\sum_{t=0}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=0}^{T} C_t^1(y_t^u) \leq \sum_{t=1}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=1}^{T} C_t^1(y_t^{\sigma_{t+1}})$$

- **Inductive step:** Since $\sigma_{T+2} = \arg\min_\sigma \sum_{t=0}^{T+1} C_t^1(y_t^\sigma)$

$$\sum_{t=0}^{T+1} C_t^1(y_t^{\sigma_t}) - \sum_{t=0}^{T+1} C_t^1(y_t^u) \leq \sum_{t=0}^{T+1} C_t^1(y_t^{\sigma_t}) - \sum_{t=0}^{T+1} C_t^1(y_t^{\sigma_{T+2}})$$

$$= \Big[\sum_{t=0}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=0}^{T} C_t^1(y_t^{\sigma_{T+2}}) \Big] + C_{T+1}(y_{T+1}^{\sigma_{T+1}}) - C_{T+1}(y_{T+1}^{\sigma_{T+2}})$$

$$\overset{\text{ind. hypoth.}}{\leq} \sum_{t=0}^{T} C_t^1(y_t^{\sigma_t}) - \sum_{t=0}^{T} C_t^1(y_t^{\sigma_{t+1}}) + C_{T+1}(y_{T+1}^{\sigma_{T+1}}) - C_{T+1}(y_{T+1}^{\sigma_{T+2}})$$

$$= \sum_{t=0}^{T+1} C_t^1(y_t^{\sigma_t}) - \sum_{t=0}^{T+1} C_t^1(y_t^{\sigma_{t+1}})$$

$$\implies \sum_{t=1}^{T+1} C_t^1(y_t^{\sigma_t}) - \sum_{t=1}^{T+1} C_t^1(y_t^u) \leq \sum_{t=1}^{T+1} C_t^1(y_t^{\sigma_t}) - \sum_{t=1}^{T+1} C_t^1(y_t^{\sigma_{t+1}}) + \frac{U(u) - \min_\sigma U(\sigma)}{\eta}$$

Bibliography

[1] Wayne E. Smith. "Various optimizers for single-stage production". In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 59–66. DOI: https://doi.org/10.1002/nav.3800030106. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800030106. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106.

[2] Ronald L Graham. "Bounds for certain multiprocessing anomalies". In: *Bell system technical journal* 45.9 (1966), pp. 1563–1581.

[3] Linus Schrage. "Letter to the Editor—A Proof of the Optimality of the Shortest Remaining Processing Time Discipline". In: *Operations Research* 16.3 (1968), pp. 687–690. DOI: 10.1287/opre.16.3.687. eprint: https://doi.org/10.1287/opre.16.3.687. URL: https://doi.org/10.1287/opre.16.3.687.

[4] 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". In: *Discrete Optimization II*. Ed. by P.L. Hammer, E.L. Johnson, and B.H. Korte. Vol. 5. Annals of Discrete Mathematics. Elsevier, 1979, pp. 287–326. DOI: https://doi.org/10.1016/S0167-5060(08)70356-X. URL: https://www.sciencedirect.com/science/article/pii/S016750600870356X.

[5] Cynthia Phillips, Clifford Stein, and Joel Wein. "Scheduling jobs that arrive over time". In: *Algorithms and Data Structures*. Ed. by Selim G. Akl, Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 86–97. ISBN: 978-3-540-44747-4.

[6] J. A. Hoogeveen and A. P. A. Vestjens. "Optimal on-line algorithms for single-machine scheduling". In: *Integer Programming and Combinatorial Optimization*. Ed. by William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 404–414. ISBN: 978-3-540-68453-4.

[7] C. Chekuri, R. Motwani, B. Natarajan, and C. Stien. "Approximation techniques for average completion time scheduling". In: *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '97. New Orleans, Louisiana, USA: Society for Industrial and Applied Mathematics, 1997, pp. 609–618. ISBN: 0898713900.

[8] A.P.A. Vestjens. "On-line machine scheduling". English. Phd Thesis 1 (Research TU/e / Graduation TU/e). Mathematics and Computer Science, 1997. ISBN: 90-386-0571-4. DOI: 10.6100/IR500043.

[9] Susanne Albers and Stefano Leonardi. "On-line algorithms". In: *ACM Comput. Surv.* 31.3es (Sept. 1999), 4–es. ISSN: 0360-0300. DOI: 10.1145/333580.333583. URL: https://doi.org/10.1145/333580.333583.

[10] René Sitters. "Two NP-hardness Results for Preemptive Minsum Scheduling of Unrelated Parallel Machines". In: *Integer Programming and Combinatorial Optimization*. Ed. by Karen Aardal and Bert Gerards. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 396–405.

[11] Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang. "Single Machine Scheduling with Release Dates". In: *SIAM Journal on Discrete Mathematics* 15.2 (2002), pp. 165–192. DOI: 10.1137/S089548019936223X. eprint: https://doi.org/10.1137/S089548019936223X. URL: https://doi.org/10.1137/S089548019936223X.

[12] Andreas S. Schulz and Martin Skutella. "The power of α-points in preemptive single machine scheduling". In: *Journal of Scheduling* 5.2 (2002), pp. 121–133. DOI: https://doi.org/10.1002/jos.93. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jos.93. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jos.93.

[13] 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.

[14] 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.

[15] Adam Kalai and Santosh Vempala. "Efficient algorithms for online decision problems". In: *Journal of Computer and System Sciences* 71.3 (2005). Learning Theory 2003, pp. 291–307. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2004.10.016. URL: https://www.sciencedirect.com/science/article/pii/S0022000004001394.

[16] Martin Skutella. "List Scheduling in Order of α-Points on a Single Machine". In: *Efficient Approximation and Online Algorithms: Recent Progress on Classical Combinatorial Optimization Problems and New Applications*. Ed. by Evripidis Bampis, Klaus Jansen, and Claire Kenyon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 250–291. ISBN:

978-3-540-32213-9. DOI: 10.1007/11671541_9. URL: https://doi.org/10.1007/11671541_9.

[17] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall international series in industrial and systems engineering. Springer, 2008. ISBN: 9780387789347. URL: https://books.google.gr/books?id=EkpDak9kEs0C.

[18] Susanne Albers. "Online Scheduling". In: *Introduction to Scheduling*. Ed. by Yves Robert and Frédéric Vivien. CRC computational science series. CRC Press / Chapman and Hall / Taylor & Francis, 2009, pp. 51–77. DOI: 10.1201/9781420072747-C3. URL: https://doi.org/10.1201/9781420072747-c3.

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844.

[20] David Karger, Cliff Stein, and Joel Wein. "Scheduling algorithms". In: *Algorithms and Theory of Computation Handbook: Special Topics and Techniques*. 2nd ed. Chapman & Hall/CRC, 2010, p. 20. ISBN: 9781584888208.

[21] Sébastien Bubeck. *Introduction to Online Optimization*. Dec. 2011. URL: https://www.microsoft.com/en-us/research/publication/introduction-online-optimization/.

[22] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[23] Alon Eden, Michal Feldman, Amos Fiat, and Tzahi Taub. "Prompt Scheduling for Selfish Agents". In: (2018). arXiv: 1804.03244 [cs.DS].

[24] MR Mazaheri Habibi, FM Abadi, H Tabesh, H Vakili-Arki, A Abu-Hanna, and S Eslami. "Evaluation of patient satisfaction of the status of appointment scheduling systems in outpatient clinics: Identifying patients' needs". In: *Journal of Advanced Pharmaceutical Technology & Research* 9.2 (Apr. 2018), pp. 51–55. DOI: 10.4103/japtr.JAPTR_134_18.

[25] Francesco Orabona. "A Modern Introduction to Online Learning". In: *CoRR* abs/1912.13213 (2019). arXiv: 1912.13213. URL: http://arxiv.org/abs/1912.13213.

[26] Maria-Florina Balcan. *Data-driven Algorithm Design*. 2020. arXiv: 2011.07177 [cs.DS]. URL: https://arxiv.org/abs/2011.07177.

[27] Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. *Pandora's Box with Correlations: Learning and Approximation*. 2020. arXiv: 1911.01632 [cs.DS]. URL: https://arxiv.org/abs/1911.01632.

[28] Elad Hazan. *Introduction to Online Convex Optimization*. 2023. arXiv: 1909.05207 [cs.LG]. URL: https://arxiv.org/abs/1909.05207.

[29] Abhishek Sinha, Ativ Joshi, Rajarshi Bhattacharjee, Cameron Musco, and Mohammad Hajiesmaili. *No-regret Algorithms for Fair Resource Allocation*. 2023. arXiv: 2303.06396 [cs.LG].

[30] Rahul Vaze. *Online Algorithms*. Cambridge University Press, 2023.