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

Φιλαλήθεις Αλγόριθμοι για την Ελαχιστοποίηση του Συνολικού Χρόνου Ολοκλήρωσης σε Ασυσχέτιστες Μηχανές.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΜΜΑΝΟΥΗΛ ΓΙΑΝΝΟΠΟΥΛΟΣ

Επιβλέπων : Δημήτριος Φωτάκης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2023

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

Φιλαλήθεις Αλγόριθμοι για την Ελαχιστοποίηση του Συνολικού Χρόνου Ολοκλήρωσης σε Ασυσχέτιστες Μηχανές.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΜΜΑΝΟΥΗΛ ΓΙΑΝΝΟΠΟΥΛΟΣ

Επιβλέπων : Δημήτριος Φωτάκης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21η Ιουλίου 2023.

.....	.....	.....
Δημήτριος Φωτάκης Καθηγητής Ε.Μ.Π.	Αριστείδης Παγουρτζής Καθηγητής Ε.Μ.Π.	Αντώνιος Συμβώνης Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2023

Εμμανουήλ Γιαννόπουλος

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

Περίληψη

Στην παρούσα διπλωματική εργασία, θα μελετήσουμε Φιλαλήθεις Αλγόριθμους Χρονοδρομολόγησης Εργασιών σε Ασυσχέτιστες Μηχανές με σκοπό την Ελαχιστοποίηση του Βεβαρημένου Χρόνου Ολοκλήρωσης. Στο μοντέλο που μας ενδιαφέρει, οι εργασίες δηλώνουν οι ίδιες τους χρόνους επεξεργασίας τους, ενώ συμπεριφέρονται ως ιδιοτελείς οντότητες, στοχεύοντας στην ελαχιστοποίηση του δικού τους χρόνου ολοκλήρωσης. Σημειώνουμε ότι στη βιβλιογραφία υπάρχουν Φιλαλήθεις Αλγόριθμοι μόνο για την πιο απλή περίπτωση των Όμοιων Μηχανών. Η γενίκευση αυτών των Αλγορίθμων σε Ασυσχέτιστες Μηχανές δεν είναι εύκολη, καθώς στην περίπτωσή μας οι εργασίες, εκτός από το να δηλώσουν έναν μικρότερο από τον πραγματικό χρόνο επεξεργασίας για να επιτύχουν έναν καλύτερο χρόνο ολοκλήρωσης, έχουν το κίνητρο να δηλώνουν μεγαλύτερους από τους πραγματικούς χρόνους επεξεργασίας σε Μηχανές στις οποίες θέλουν να αποφύγουν να ανατεθούν. Αρχικά, παραθέτουμε έναν Μη Φιλαλήθη βέλτιστο Δεσμευτικό Αλγόριθμο, καθώς και έναν Φιλαλήθη Αλγόριθμο του οποίου τον Λόγο Προσέγγισης δεν έχουμε καταφέρει να αναλύσουμε πλήρως. Υποστηρίζουμε τον παραπάνω αλγόριθμο με τη διεξαγωγή μιας πειραματικής διαδικασίας, με σκοπό τη σύγκριση της επίδοσής του με αυτή του καλύτερου επί του παρόντος Online Αλγόριθμου Χρονοδρομολόγησης, καθώς επίσης και με ένα κάτω όριο της βέλτιστης λύσης.

Λέξεις κλειδιά

Online αλγόριθμοι, Αλγόριθμοι Χρονοδρομολόγησης, Χρόνος Ολοκλήρωσης, Δεσμευτικότητα, Φιλαλήθεια, Σχεδιασμός Μηχανισμών

Abstract

In this thesis, we study the problem of online job scheduling in unrelated machines with the goal of minimizing the weighted sum of completion times, in the setting where the jobs are selfish agents, self-reporting their processing times. In the majority of the literature surrounding scheduling problems, the jobs' characteristics are assumed to be known to the algorithm. In the case where the jobs are selfish agents and the objective is the minimization of the sum of weighted completion times, only the simpler setting of identical parallel machines has been explored. Generalizing these results to the broader case of unrelated machines proves to be challenging as, apart from having an incentive to underbid their processing times to achieve a better completion time, jobs have an extra incentive to overbid some of their processing times in order to manipulate the machine assignment rule. We first propose a non-truthful tight approximation algorithm, as well as a truthful algorithm whose competitive ratio we were unable to fully analyze. Finally, we conduct an experimental evaluation of the truthful algorithm, comparing it to the state-of-the-art online algorithm for the setting of unrelated machines with the objective of minimizing the weighted sum of completion times, as well as to a close lower bound of the optimal solution.

Key words

Online Algorithms, Online Scheduling, Completion Time, Promptness, Algorithmic Mechanism Design, Truthfulness

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας περατώνονται μετά από 6 έτη οι σπουδές μου στη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα πρωτίστως να ευχαριστήσω τον κ. Φωτάκη τόσο για την βοήθειά του κατά την εκπόνηση της εργασίας, όσο και για την έμπνευση και την κατεύθυνση προς τη Θεωρητική Πληροφορική που μου προσέφερε η εξαιρετική διδασκαλία του. Θα ήθελα επίσης να ευχαριστήσω όλα τα μέλη του εργαστηρίου CoReLab για τις πολύτιμες ώρες που περάσαμε μαζί διευρύνοντας τις γνώσεις μας και μελετώντας τα διάφορα θέματα που μας ενδιαφέρουν. Επιπλέον, θέλω να ευχαριστήσω όλους τους φίλους και συμφοιτητές μου, τους οποίους αν ξεκινούσα να κατονομάζω δεν θα αρκούσε μια σελίδα, για όλες τις στιγμές που περάσαμε, η κάθεμια από τις οποίες έχει συμβάλλει κατά ένα τρόπο στη διαμόρφωση της ταυτότητας μου ως άνθρωπος. Τέλος, θέλω να ευχαριστήσω φυσικά την οικογένειά μου που πάντα στέκεται αρωγός σε κάθε δυσκολία που μπορεί να αντιμετωπίζω, βοηθώντας με με τον τρόπο τους να κοιτάω πάντα μπροστά.

Εμμανουήλ Γιαννόπουλος,

Αθήνα, 21η Ιουλίου 2023

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εκτεταμένη Ελληνική Περίληψη

Η παρούσα διπλωματική έχει εκπονηθεί κατά κύριο λόγο στην Αγγλική γλώσσα. Σε αυτό το κεφάλαιο, θα συνοψίσουμε το κυριότερο περιεχόμενό της στα Ελληνικά. Η οργάνωση της εκτεταμένης ελληνικής περίληψης συμφωνεί πλήρως με την οργάνωση του αγγλικού κειμένου, έχοντας χωριστεί στα ίδια κεφάλαια

1.1 Εισαγωγή

Το γενικότερο πρόβλημα το οποίο απασχολεί τη συγκεκριμένη διπλωματική εργασία είναι η Χρονοδρομολόγηση Εργασιών (Job scheduling). Πρόκειται για ένα πρόβλημα το οποίο απασχολεί την ανθρωπότητα από τα αρχαία χρόνια. Όλα τα μεγάλα επιτεύγματα της ανθρωπότητας, από το Μεγάλο Σινικό Τείχος μέχρι την άνθηση της αρχαίας Ελληνικής οικονομίας μέσω του θαλάσσιου εμπορίου, αποτελούν μια συλλογή από μικρές εργασίες οι οποίες έπρεπε να προγραμματιστούν στο σωστό χρονικό διάστημα και με τη σωστή σειρά, ώστε να οργανωθεί όσο το δυνατόν καλύτερα και η κατανάλωση των απαραίτητων πόρων. Στη σύγχρονη εποχή, το πρόβλημα της χρονοδρομολόγησης εργασιών έχει απασχολήσει τους ερευνητές της Επιστήμης των Υπολογιστών από τα μισά του περασμένου αιώνα, τόσο στον τομέα της Θεωρητικής Πληροφορικής όσο και στον τομέα της Επιχειρησιακής Έρευνας. Στην παρούσα διπλωματική, μελετάμε το πρόβλημα της χρονοδρομολόγησης εργασιών σε ασυσχέτιστες μηχανές με σκοπό την ελαχιστοποίηση του βεβαρημένου χρόνου ολοκλήρωσης. Μας ενδιαφέρει κυρίως το online μοντέλο, κατά το οποίο οι εργασίες έρχονται στην πορεία και δεν έχουμε πρότερη γνώση της μελλοντικής εισόδου. Επίσης, μελετάμε φιλαλήθεις αλγόριθμους, μοντελοποιώντας τις εργασίες ως ιδιοτελείς παίκτες που προσπαθούν να ελαχιστοποιήσουν τον δικό τους χρόνο ολοκλήρωσης.

1.2 Online χρονοδρομολόγηση εργασιών για την ελαχιστοποίηση του βεβαρημένου χρόνου ολοκλήρωσης

Για να ορίσουμε τυπικά το πρόβλημα, ένα πρόβλημα χρονοδρομολόγησης εργασιών αποτελείται από n εργασίες και m μηχανές. Η κάθε εργασία μπορεί να έχει ένα βάρος w_j, ένα χρόνο εκτέλεσης p_{ij} συσχετισμένο με κάθε μηχανή καθώς και έναν χρόνο άφιξης r_j. Οι Graham et al. [21] όρισαν ένα σύστημα σημειογραφίας για προβλήματα χρονοδρομολόγησης βασισμένο σε τρεις μεταβλητές $\alpha|\beta|\gamma$, των οποίων τη σημασία θα ορίσουμε:

1. α: Ορίζει το περιβάλλον των μηχανών. Οι κυριότερες κατηγορίες που μας ενδιαφέρουν στην παρούσα διπλωματική είναι οι εξής:
    - 1: Μια μηχανή
    - P: Παράλληλες (όμοιες) μηχανές. Σε αυτή την περίπτωση, η κάθε εργασία έχει μοναδικό χρόνο επεξεργασίας p_j που είναι ίδιος για όλες τις μηχανές

- R: Ασυσχέτιστες μηχανές. Η κάθε εργασία έχει έναν χρόνο εκτέλεσης p_{ij} για κάθε μηχανή. Οι χρόνοι αυτοί δεν συνδέονται με κανέναν τρόπο.

2. β: Ορίζει τους περιορισμούς του προβλήματος. Οι δυο περιορισμοί που αναφέρονται στην παρούσα διπλωματική είναι:

   - r_j: Χρόνος άφιξης της κάθε εργασίας.
   - $prec$: Ορίζει σχέσεις αρχαιότητας μεταξύ εργασιών. Συμβολίζουμε με $j < j'$ όταν για να εκκινήσει η εκτέλεση της εργασίας j', πρέπει να έχει ολοκληρωθεί η εκτέλεση της εργασίας j.

3. γ: Ορίζει τον στόχο του αλγόριθμου. Οι 3 βασικότεροι στόχοι είναι οι εξής:

   - c_{max}: Makespan. Ο χρόνος ολοκλήρωσης της τελευταίας εργασίας σε οποιαδήποτε μηχανή, δηλαδή το τέλος του προγράμματος.
   - $\sum_j w_j c_j$. Άθροισμα βεβαρημένων χρόνων ολοκλήρωσης. Το άθροισμα όλων των επιμέρους χρόνων ολοκλήρωσης πολλαπλασιασμένων με το αντίστοιχο βάρος της εργασίας.
   - $\sum F_j$. Άθροισμα χρόνων ροής. Ο χρόνος ροής μιας εργασίας είναι η διαφορά του χρόνου ολοκλήρωσής της c_j από το χρόνο άφιξής της r_j.

Αφού ορίσαμε το πρόβλημα, στην ελληνική περίληψη της παρούσας εργασίας θα παρουσιάσουμε τις βασικές δουλειές από τις οποίες αντλήσαμε έμπνευση για να ορίσουμε τους δικούς μας αλγόριθμους. Σε καμία περίπτωση δεν είναι μόνο αυτές οι πιο σημαντικές δουλειές, καθώς η έρευνα στο πρόβλημα της χρονοδρομολόγησης εργασιών είναι αρκετή για να γεμίσει μια εγκυκλοπαίδεια. Η πρώτη δουλειά που μελέτησε τη χρονοδρομολόγηση εργασιών με στόχο την ελαχιστοποίηση του βεβαρημένου χρόνου ολοκλήρωσης ήταν αυτή του Smith [49]. Μελετώντας το πρόβλημα σε περιβάλλον μιας μηχανής, ο Smith πρότεινε έναν βέλτιστο αλγόριθμο που λύνει το πρόβλημα, ο οποίος έχει μείνει γνωστός ως WSPT (Weighted Shortest Processing Time, Βεβαρημένος Συντομότερος Χρόνος Εκτέλεσης) και ως κανόνας του Smith. Ο αλγόριθμος αυτός ταξινομεί τις εργασίες σε φθίνουσα σειρά $\frac{w_j}{p_j}$ και τις εκτελεί κατά σειρά. Λόγω της βελτιστότητάς του, παραμένει κύρια ιδέα σε όλους τους online αλγορίθμους χρονοδρομολόγησης. Μεταπηδώντας στο περιβάλλον των παράλληλων μηχανών, οι Megow και Schulz [38] εισήγαγαν την τεχνική της *καθυστέρησης εργασιών* (job delaying), κατά την οποία οι εργασίες με μεγάλο χρόνο εκτέλεσης καθυστερούνται κατά ένα παράγοντα ωστέ να προσπεραστούν (αν χρειάζεται) από μικρότερες εργασίες που θα έρθουν μελλοντικά στο σύστημα. Με αυτή την τεχνική, κατάφεραν να επεκτείνουν μια παραλλαγή του αλγόριθμου του Smith στις παράλληλες μηχανές πετυχαίνοντας έναν 3.28-προσεγγιστικό αλγόριθμο για το online setting.

Στο περιβάλλον των ασυσχέτιστων μηχανών, που είναι και αυτό που κυρίως μας ενδιαφέρει, παρουσιάζουμε δυο δουλειές. Η πρώτη συγγράφηκε από τους Hall et al. [24], και περιλαμβάνει έναν 8-προσεγγιστικό αλγόριθμο βασισμένο σε ένα Γραμμικό Πρόγραμμα το οποίο οι συγγραφείς αποκαλούν χρονικά-δεικτοδοτούμενο (time-indexed),. Αυτό το γραμμικό πρόγραμμα μας δίνει ένα αρκετά καλό κάτω όριο της βέλτιστης λύσης και γι'αυτό το χρησιμοποιήσαμε σαν μέτρο σύγκρσης στην πειραματική αξιολόγηση του Κεφαλαίου 6. Παρουσιάζουμε το γραμμικό πρόγραμμα παρακάτω:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} w_j \sum_{i=1}^{m} \sum_{\ell=1}^{L} \tau_{\ell-1} x_{ij\ell} \\
\text{subject to} \quad & \sum_{i=1}^{m} \sum_{\ell=1}^{L} x_{ij\ell} = 1 && j \in [n] \\
& \sum_{j=1}^{n} p_{ij} x_{ij\ell} \leq \tau_\ell, && i \in [m] \quad \ell \in [L] \\
& x_{ij\ell} = 0 && \text{if } \tau_\ell < r_{ij} + p_{ij} \\
& x_{ij\ell} \geq 0 && i \in [m], j \in [n], \ell \in [L]
\end{aligned}
$$

Σχεδόν 20 χρόνια αργότερα, οι Gupta et al. [23] πέτυχαν την πρώτη βελτίωση στο online πρόβλημα των ασυσχέτιστων μηχανών, κατασκευάζοντας έναν 7.216-προσεγγιστικό αλγόριθμο βασισμένο σε 3 άξονες: Την τεχνική καθυστέρησης εργασιών, τον WSPT και ένα άπληστο κριτήριο ανάθεσης εργασιών σε μηχανές. Το άπληστο κριτήριο υπολογίζει το πόσο επιβαρύνει η κάθε εργασία την ήδη υπάρχουσα ουρά σε κάθε μηχανή, και ορίζεται ως παρακάτω:

Definition 1.2.1 ($cost(j \to i)$). *Για κάθε εργασία j και μηχανή i, ορίζουμε το $cost(j \to i)$ as:*

$$
cost(j \to i) := w_j \left(\left(1 + \frac{1}{c}\right) r_{ij} + p_{ij} + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} \geq \frac{w_j}{p_{ij}}} p_{ik} \right) + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} < \frac{w_j}{p_{ij}}} w_k p_{ij}
$$

Το πρώτο μέλος του αθροίσματος περιγράφει το σύνολο των εργασιών που θα προηγούνται τις j στην ουρά της μηχανής i, καθυστερώντας την εκτέλεσή της. Αντίστοιχα, το δεύτερο μέλος του αθροίσματος περιγράφει το σύνολο των εργασιών το οποίο θα προσπεράσει η j, καθυστερώντας την εκτέλεσή τους. Με βάση το παραπάνω κριτήριο, οι Gupta et al. ορίζουν τον παρακάτω αλγόριθμο.

Mechanism 1: Άπληστος online αλγόριθμος για την ελαχιστοποίηση του βεβαρημένου χρόνου ολοκλήρωσης

---

1. Ορίζουμε έναν τροποποιημένο χρόνο άφιξης r_{ij} για κάθε μηχανή i, $i \in [m]$. Ο κάθε χρόνος άφιξης ορίζεται ως ακολούθως: $r_{ij} := \max\{r_j, c \cdot p_{ij}\}$ για μια σταθερά $c = \frac{2}{3}$.

2. Αναθέτουμε κάθε εργασία σε μια μηχανή με βάση το ακόλουθο κριτήριο. Έστω $U_i(t)$ το σύνολο των εργασιών που έχουν ήδη ανατεθεί στη μηχανή i μέχρι τη χρονική στιγμή t και *δεν έχει ξεκινήσει η εκτέλεσή τους*. Υπολογίζουμε το $cost(j \to i)$ της εργασίας j σε κάθε μηχανή i με βάση το σύνολο εργασιών $U_i(r_j)$ και την αναθέτουμε στη μηχανή με το ελάχιστο κόστος.

3. Σε κάθε μηχανή, ορίζουμε το πρόγραμμα ακριβώς όπως στον WSPT. Δηλαδή, σε κάθε μηχανή, όταν γίνεται διαθέσιμη τη χρονική στιγμή t, δρομολογούμε τη δουλειά από το σύνολο $U_i(t) \cap \{j | r_{ij} \leq t\}$ με την υψηλότερη τιμή του λόγου $\frac{w_j}{p_{ij}}$.

Θεώρημα 1.2.1. *Ο αλγόριθμος* (1) *έχει competitive ratio ίσο με* 7.216

1.3 Φιλαλήθεια και Χρονοδρομολόγηση

Στο παραπάνω κεφάλαιο, ασχοληθήκαμε με το πρόβλημα της online χρονοδρομολόγησης εργασιών σε ασυσχέτιστες μηχανές, θεωρώντας ότι όλες οι πληροφορίες που αφορούν μια εργασία

είναι δημόσιες και άρα ο αλγόριθμος τις γνωρίζει επακριβώς. Σε πολλές πρακτικές εφαρμογές του προβλήματος, μια τέτοια υπόθεση δεν είναι πρακτική. Πολλές φορές, οι εργασίες μπορεί να χρειαστεί να αποκαλύψουν κάποια κρυφή πληροφορία που έχουν, και, δρώντας ιδιοτελώς, μπορούν να την αποκαλύψουν ψευδώς για να πετύχουν κάποιο δικό τους σκοπό. Γι'αυτό το λόγο, το πρόβλημα της χρονοδρομολόγησης εργασιών απασχολεί πολλούς ερευνητές στην περιοχή του *σχεδιασμού μηχανισμών*. Σκοπός τους είναι να παράξουν φιλαλήθεις αλγόριθμους, στους οποίους είναι πάντα προς όφελος της κάθε εργασίας να αποκαλύψει αληθώς την όποια πληροφορία. Για πολλά χρόνια, το κύριο πρόβλημα χρονοδρομολόγησης για τους ερευνητές στο Σχεδιασμό Μηχανισμών ήταν το εξής: Εργασίες πρέπει να δρομολογηθούν σε ασυσχέτιστες μηχανές με σκοπό την ελαχιστοποίηση του makespan. Η κάθε εργασία έχει έναν χρόνο επεξεργασίας για κάθε μηχανή p_{ij} τον οποίο γνωρίζει *μόνο η μηχανή*. Φυσικά, έχει το κίνητρο να δηλώσει ψευδώς έναν πολύ μεγαλύτερο χρόνο, ώστε η εργασία να δρομολογηθεί αλλού και έτσι η μηχανή να ελαχιστοποιήσει το δικό της φόρτο. Η πρώτη δουλειά σε αυτό το πρόβλημα ανήκει στους Nisan και Ronen [39], οι οποίοι απέδειξαν ότι δεν μπορεί να υπάρξει 2-προσεγγιστικός αλγόριθμος, και υπέθεσαν ότι το πραγματικό κάτω όριο του προβλήματος είναι $\Omega(m)$. Μετά από πολλά έτη και πολλές ενδιάμεσες βελτιώσεις τις οποίες αναφέρουμε στο κεφάλαιο 4, οι Christodoulou, Koutsoupias και Kovacs [39] απέδειξαν την παραπάνω υπόθεση κλείνοντας το πρόβλημα.

Στην παρούσα εργασία μας ενδιαφέρει ένα άλλο setting: n εργασίες εισέρχονται στο σύστημα για να δρομολογηθούν σε m παράλλλες μηχανές, διατηρώντας ως κρυφή πληροφορία τον χρόνο εκτέλεσης τους p_j. Λόγω του κανόνα WSPT, οι εργασίες έχουν κίνητρο να δηλώσουν μικρότερο χρόνο εκτέλεσης από τον πραγματικό για να δρομολογηθούν νωρίτερα στο πρόγραμμα. Στο πλαίσιο αυτό, παρουσιάζουμε τις 2 βασικές δουλείες που ασχολούνται με αυτό το πρόβλημα. Οι Angel et al. [1] ορίζουν μια τεχνική με το όνομα *preventive preemption*, κατά την οποία μια εργασία που ψεύδεται για το χρόνο επεξεργασίας δηλώνοντας ένα $p'_j < p_j$ τιμωρείται ως εξής: Αν δεν έχει ολοκληρωθεί μετά από χρόνο p'_j, μετατίθεται σε μια ουρά στο τέλος του προγράμματος μαζί με τις υπόλοιπες τιμωρημένες εργασίες, η οποία ουρά εκτελείται με αλγόριθμο round robin. Ο αλγόριθμος τους για μια μηχανή παρουσιάζεται παρακάτω:

Mechanism 2: WSPT-PP

---

1. Ταξινομούμε όλες τις εργασίες σε σειρά WSPT ($\frac{b_1}{w_1} \leq \frac{b_2}{w_2} \leq ... \leq \frac{b_n}{w_n}$)
2. Προγραμματίζουμε το διάστημα εκτέλεσης της κάθε εργασίας ως $l_1^j = \sum_{i=1}^{j-1} b_j$ and $r_1^j = l_1^j + b_j$
3. Μετά από χρόνο $t = \sum_j b_j$ δρομολογούμε τις τιμωρημένες εργασίες με αλγόριθμο round robin: Για κάθε $x \geq 2$ (όπου x ο αριθμός των εργασιών), αν η εργασία i δεν έχει ολοκληρωθεί σε χρόνο $\sum_j b_j + n(x-2) + i - 1$ την δρομολογούμε στο διάστημα $[l_i^x, r_i^x]$, με $l_i^x = \sum_j b_j + n(x-2) + i - 1$ και $r_i^x = \sum_j b_j + n(x-2) + i$

Θεώρημα 1.3.1. *Ο αλγόριθμος* (2) *είναι βέλτιστος για το περιβάλλον της μιας μηχανής.*

Οι συγγραφείς ορίζουν την επέκταση του αλγορίμου για περιβάλλον παράλληλων μηχανών ως εξής: Αρχικά γίνεται μια τυχαία και ομοιόμορφη ανάθεση εργασιών σε μηχανές. Αυτό είναι απαραίτητο καθώς σε αντίθετη περίπτωση (όπως δείχνουμε στο κεφάλαιο 4) οι εργασίες έχουν κίνητρο να δηλώσουν ψευδή χρόνο εκτέλεσης. Έπειτα, δρομολογούμε τις εργασίες σε κάθε μηχανή ακριβώς όπως στον WSPT-PP.

Θεώρημα 1.3.2. *Η επέκταση του WSPT-PP σε παράλληλες μηχανές είναι $\frac{3}{2}$-competitive.*

Η δεύτερη δουλειά που προτείνει έναν φιλαλήθη αλγόριθμο για χρονοδρομολόγηση εργασιών σε παράλληλες μηχανές είναι αυτή των Eden et al. Ο συγκεκριμένος αλγόριθμος έχει και μια παραπάνω ιδιότητα, αυτή της *δεσμευτικότητας*.

Ορισμός 1.3.1 (Δεσμευτικοί αλγόριθμοι). *Ένας αλγόριθμος είναι δεσμευτικός όταν για κάθε εργασία j με χρόνο άφιξης r_j, ο χρόνος ολοκλήρωσής της c_j ορίζεται οριστικά και αμετάκλητα την χρονική στιγμή r_j.*

Θεώρημα 1.3.3. *Κάθε δεσμευτικός αλγόριθμος έχει competitive ratio $\Omega(\log P_{max})$ όπου P_{max} είναι ο μέγιστος χρόνος εκτέλεσης μιας εργασίας που εισέρχεται στο σύστημα.*

Για να επιτύχουν δεσμευτικότητα, οι συγγραφείς χωρίζουν το χρονικό ορίζοντα σε προκαθορισμένα χρονικά διαστήματα, τα οποία παρουσιάζουν στις εργασίες σε μορφή μενού επιλογών, χωρίς πρότερο input από τις εργασίες. Έτσι, επιτυγχάνεται αυτόματα η φιλαλήθεια, εφόσον η εργασία επιλέγει μόνη της το διάστημα που επιθυμεί από το μενού. Ο τρόπος χωρισμού του χρονικού ορίζοντα σε διαστήματα βασίζεται στην παρακάτω ακολουθία:

$$S_0 = \langle 1 \rangle$$
$$S_1 = S_0 || S_0 || 2^1 = \langle 1, 1, 2 \rangle$$
$$S_2 = S_1 || S_1 || 2^2 = \langle 1, 1, 2, 1, 1, 2, 4 \rangle$$
$$S_k = S_{k-1} || S_{k-1} || 2^k$$

Αντίστοιχα, μια άπειρη ακολουθία ορίζεται όπως παρακάτω, έχοντας κάθε S_k ως πρόθεμα:

$$S_\infty = \langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 16, ... \rangle$$

Η αιτία επιλογής της συγκεκριμένης ακολουθίας συνοψίζεται στο ακόλουθο λήμμα. Στο κεφάλαιο 4 αναλύουμε εκτενέστερα και διαισθητικά τη χρησιμότητα του λήμματος:

Λήμμα 1.3.1. *Για κάθε $d \geq 0$ και για κάθε $0 \leq k \leq d$, ο συγχωνευμένος όγκος όλων των διαστημάτων μήκους 2^k στην S_d ισούται με 2^d:*

$$\sum_{i:S_d[i]=2^k} 2^k = 2^d$$

Με βάση την παραπάνω ακολουθία, οι συγγραφείς ορίζουν τον τρόπο με τον οποίο γίνεται ο χωρισμός του ορίζοντα σε διαστήματα κάθε φορά που εισέρχεται στο σύστημα μια εργασία j, με βάση παράγοντες: Τον χρόνο άφιξης της εργασίας r_j, το χρόνο ολοκλήρωσης της c_j (κριτήριο που έμμεσα μας αποκαλύπτει το χρόνο εκτέλεσής της p_j), και το μέγεθος και το τελικό σημείο της ακολουθίας μέχρι τώρα τα οποία συμβολίζουμε με l_j και $e(A^j_{l_j})$ αντίστοιχα. Η ακολουθία μεταβάλλεται ως εξής:

- Αν $c_j \leq e(A^j_{l_j})$, τότε η εργασία επέλεξε ένα ήδη προκαθορισμένο διάστημα και δεν μεταβάλλεται η ακολουθία.
- Αν $r_j \geq e(A^j_{l_j})$, τότε η εργασία αφίχθη στο σύστημα μετά το τέλος της τελευταίας ακολουθίας. Η $A^j_{l_j}$ μονιμοποιείται, και μια νέα ακολουθία $A^j_{l_{j+1}}$ δημιουργείται ξεκινώντας από την $S_k(r_j)$. Η περίπτωση αυτή δημιουργεί κενό στο πρόγραμμα και δεν μπορεί να προκύψει με adversarial είσοδο
- Αν $r_j \leq e(A^j_{l_j})$ και $c_j > e(A^j_{l_j})$ αλλά η $A^j_{l_j}$ είναι μεγαλύτερη από k, τότε η $A^j_{l_j}$ επεκτείνεται με την προσθήκη της $S_k(e(A^j_{l_j}))$ (παρατηρούμε ότι η $A^j_{l_j}$ παραμένει πρόθεμα της $S_\infty(t)$)
- Τέλος, αν $r_j \leq e(A^j_{l_j})$ και $c_j > e(A^j_{l_j})$ αλλά η $A^j_{l_j}$ έχει μέγεθος μικρότερο από k, τότε $A^j_{l_j}$ επεκτείνεται, μετατρέποντάς την στην $S_k(b(A^j_{l_j}))$.

Με βάση τις παραπάνω μεθόδους τροποποίησης της ακολουθίας κατασκευάζουμε σε κάθε στιγμιότυπο την κατάσταση του αλγορίθμου την οποία συμβολίζουμε με $\tau(A_{l_{j-1}}^{j-1}, r_j, j)$. Μέσω αυτής ορίζεται ο παρακάτω αλγόριθμος

Mechanism 3: Αλγόριθμος δημιουργίας μενού με δυναμικές ακολουθίες

Δοθείσης μιας εργασίας j με χρόνο άφιξης r_j, και μιας κατάστασης $A_{l_{j-1}}^{j-1}$, κατασκευάζουμε την $\tau(A_{l_{j-1}}^{j-1}, r_j, j)$ και κάνουμε το ακόλουθο:

$a \leftarrow 1$;
while *η εργασία δεν έχει επιλέξει διάστημα* **do**
    Πρόσφερε το πρώτο διάστημα μεγέθους 2^a σε κάθε μηχανή;
    $a \leftarrow a * 2$;
end

Θεώρημα 1.3.4. *Ο αλγόριθμος* (3) *είναι* $O(\log P_{max})$*-competitive όπου* P_{max} *ο χρόνος επεξεργασίας της μεγαλύτερης εργασίας που εισέρχεται στο σύστημα.*

1.4 Φιλαλήθεια σε ασυσχέτιστες μηχανές

Στην προηγούμενη ενότητα αναφέραμε 2 φιλαλήθεις αλγόριθμους για το πρόβλημα χρονοδρομολόγησης σε παράλληλες μηχανές. Ο αλγόριθμος των Angel et al. βασιζόταν σε πληροφορία που ανέφεραν οι εργασίες για τον χρόνο εκτέλεσής τους. Στο setting των ασυσχέτιστων μηχανών αυτό δεν μπορεί να συμβεί, καθώς οι εργασίες μπορούν να χειραγωγήσουν κάθε κανόνα ανάθεσης με τρόπο παρόμοιο με το ακόλουθο παράδειγμα:

Παράδειγμα 1.4.1. *Μια εργασία με χρόνους εκτέλεσης* p_{ij} *μπορεί να χειραγωγήσει την μηχανή* m *στην οποία θα ανατεθεί αναφέροντας ταυτόχρονα αληθώς τον χρόνο εκτέλεσης της για τη συγκεκριμένη μηχανή ως ακολούθως:*

1. *Δήλωσε χρόνο εκτέλεσης* p_{mj} *για τη μηχανή* m*.*
2. *Δήλωσε ψευδώς ότι* $p_{m'j} = \infty$ *για κάθε άλλη μηχανή* $m' \neq m$

Λόγω του παραπάνω, για την χρονοδρομολόγηση σε ασυσχέτιστες μηχανές η ερευνητική μας δουλειά επικεντρώθηκε σε μηχανισμούς μενού. Παρόλα αυτά, παρουσιάζουμε έναν μη-φιλαλήθη δεσμευτικό αλγόριθμο που πετυχαίνει το βέλτιστο δυνατό competitive ratio, βασισμένο στους αλγορίθμους (1) και (3).

Mechanism 4: Μη-Φιλαλήθης δεσμευτικός αλγόριθμος μενού

Τρέχουμε ένα "αντίγραφο" του αλγορίθμου (1) στο παρασκήνιο. Για κάθε εργασία που έρχεται στο σύστημα με χρόνους εκτέλεσης p_{ij}:

Υπολογίζουμε την ανάθεσή της σε μια μηχανή με βάση τα $cost(j \rightarrow i)$;
Υπολογίζουμε την ακολουθία $\tau(A_{l_j}^{j}, r_j)$ για τη μηχανή στην οποία ανατέθηκε (τη συμβολίζουμε με i);
$a \leftarrow 1$;
while *η εργασία δεν έχει επιλέξει διάστημα* **do**
    Πρόσφερε το πρώτο διάστημα μεγέθους 2^a στην μηχανή i;
    $a \leftarrow a * 2$;
end

Θεώρημα 1.4.1. *Ο αλγόριθμος 4 είναι* $O(\log P_{max})$*-competitive.*

1.5 Φιλαλήθης αλγόριθμος και πειραματική αξιολόγηση

Στο τελευταίο σκέλος της διπλωματικής εργασίας παρουσιάζουμε έναν φιλαλήθη αλγόριθμο του οποίου το competitive ratio δεν επιτύχαμε να αποδείξουμε. Περισσότερες πληροφορίες για τη διαδικασία που ακολουθήσαμε και τις δυσκολίες που προκύπτουν δίνονται στο κεφάλαιο 6. Ο αλγόριθμος είναι όμοιος με τον αλγόριθμο (3) με τη διαφορά ότι κάθε μηχανή έχει τη δική της ακολουθία εφόσον έχουμε ασυσχέτιστες μηχανές. Τυπικά, ο αλγόριθμος ορίζεται παρακάτω:

Mechanism 5: Αλγόριθμος δημιουργίας μενού με δυναμικές ακολουθίες για ασυσχέτιστες μηχανές

Δοθείσης εργασίας j με χρόνο άφιξης r_j, και μιας κατάστασης $A_{l_{j-1}}^{j-1}$, κατασκευάζουμε την $\tau(A_{l_{j-1}}^{j-1}, r_j, j)$ και κάνουμε το ακόλουθο:

$a \leftarrow 1$;
while *η εργασία δεν έχει επιλέξει διάστημα* **do**
    Πρόσφερε το πρώτο διάστημα μεγέθους 2^a σε κάθε μηχανή;
    $a \leftarrow a * 2$;
end

Εφόσον δεν καταφέραμε να αποδείξουμε το competitive ratio του παραπάνω αλγόριθμου, πραγματοποιήσαμε μια πειραματική αξιολόγηση συγκρίνοντας τους αλγορίθμους (5), (**??**) και τη βέλτιστη λύση. Οι είσοδοι των πειραμάτων χωρίστηκαν σε 2 γκρουπ. Το πρώτο δημιουργήθηκε από εμάς και τα χαρακτηριστικά του αναλύονται στο κεφάλαιο 6. Το δεύτερο προέκυψε από την κατάλληλη επεξεργασία δημόσιων δεδομένων της βιβλιοθήκης PSPLIB [32, 33, 13] για να προσαρμοστούν στο πρόβλημα μας. Τα βασικά ευρήματα της πειραματικής αξιολόγησης ήταν τα εξής 3:

1. Ο αλγόριθμος 1 έχει πολύ καλύτερη επίδοση από το θεωρητικό άνω όριο του 7.216, αφού παρατηρήσαμε ένα competitive ratio περίπου 4, όταν όλες οι εργασίες είναι διαθέσιμες τη χρονική στιγμή 0, και 1.5 έως 2, όταν οι εργασίες έχουν χρόνους άφιξης.

2. Ο μέγιστος χρόνος επεξεργασίας P_{max}^* που εμφανίζεται στη βέλτιστη λύση είναι σε μέγεθος συγκρίσιμο (με λόγο γύρω στο 1) με τον μέγιστο χρόνο επεξεργασίας που εμφανίζεται στον αλγόριθμο 5 $P_{max,ALG}$. Αυτό σημαίνει ότι ένα άνω όριο $\log P_{max}$ έχει κάποιο νόημα (το οποίο δεν θα ίσχυε αν το $P_{max,ALG}$ δεν φρασσόταν από τίποτα).

3. Ο αλγόριθμος 5 έχει αρκετά καλή επίδοση ώστε να υποστηρίζει την υπόθεση ότι στη μέση περίπτωση, το competitive ratio του είναι $\Theta(\log P_{max})$

Chapter 2

Introduction

Project scheduling and resource allocation challenges have captivated the attention of scholars and practitioners throughout history, making them an enduring and significant area of study. From monumental construction endeavors like the Great Wall of China to the intricate trade networks of Ancient Greece, the effective allocation and scheduling of individual tasks that collectively contribute to the accomplishment of a larger objective have been prevalent in diverse and complex scenarios. In the modern world, job scheduling problems have captured the attention of both practitioners in the operations research field, as well as computer scientists aiming to propose efficient algorithms that handle the influx of diverse tasks and optimally schedule them among different machines, often with different objectives. Historically, the main objectives of interest for researchers have been the *makespan*, which is the completion time of the final job that finished its execution in any machine, the *sum of completion times*, which is the sum of all times at which a job's execution was completed, and the *sum of flow times*, i.e. the sum of each job's time where it was awaiting its execution. In this thesis, we mostly concern ourselves with the *sum of completion times*. Perhaps the more natural way of showcasing the importance of the sum of completion times as an objective is by presenting it as an equivalent objective to the *average completion time* of all jobs. It is evident, that as the above objective is minimized, it can be used to measure how "happy" the average participant is with the result of the scheduling algorithm. We shall first go through a review of the most important works on this objective, in the regular case where the jobs are mostly known and either are all available at once or arrive online. Subsequently, we shall present the case where each job is submitted by an agent who might have the ulterior motive of maximizing their own "happiness", i.e. minimizing their own completion time. This setting of the problem poses extra challenges and has captured the attention of researchers working in the field of *algorithmic mechanism design*. Finally, we shall propose a new algorithm for a specific setting of this problem, and also compare the performance of different scheduling algorithms by performing an experimental evaluation.

2.1 Previous Work

2.1.1 Online scheduling to minimize the sum of completion times

The seminal work that introduced job scheduling as a problem in the field of Operations Research was that of Smith [49]. Proposing the now very famous SPT algorithm that minimizes the sum of completion times on a single machine by scheduling the shortest jobs first, Smith was the first in a long line of works spanning decades and tackles the scheduling problems in different settings and under different sets of constraints. The single-machine problem was studied in much more detail in the online version of the problem, where the jobs arrive over time and the algorithm cannot know which or even how many jobs will arrive in the future. To formalize the setting, we shall use the notation proposed by Graham et al. [21]. There exists a set $J = \{j \in \{1, 2, \ldots, n\}\}$ of jobs to be scheduled among a set $M = \{i \in \{1, 2, \ldots, m\}\}$ of machines, obtaining a completion time c_j. Each job has m processing times p_{ij} associated with each machine (if all p_{ij} are the same, the machines are called identical), a weight w_j and a release time r_j. The goal of the algorithms we will examine

is to minimize the sum of weighted completion times $\sum_j w_j c_j$. In some algorithms, preemption is allowed, meaning that a job's execution can be paused and continued at a later time. To assess the performance of an online algorithm, we usually denote by $cost(ALG)$ the cost of the online algorithm, and by $cost(OPT)$ the total cost of the optimal offline algorithm (in this case, schedule), who possesses knowledge of the entire input a priori. Then, the online algorithm is considered c-competitive, if for every single possible input, it holds that $cost(ALG) \leq c \cdot cost(OPT)$. Parameter c can be a constant, or an asymptotic function of any input variable (for example the number of jobs n). The first constant-factor approximation for the online single machine setting was given by Phillips et al. [41], who first used the technique of creating the optimal preemptive schedule and then transforming it into a good enough non-preemptive one. In the case of parallel machines, Megow and Schulz [38] are the first to use the technique of *job delaying*. As we stated with Smith's rule, when scheduling to minimize the completion time, the algorithm wants to schedule the smaller jobs first and the larger jobs last. Job delaying aims to create a "buffer" between a large job's arrival and its earliest possible execution time, during which smaller jobs may arrive and bypass it. In the unrelated machine case, Hall et. al [24], present the "interval-indexed" LP relaxation, an LP whose variations are still used today to achieve close-to-optimal approximation algorithms in the offline case (with the state-of-the-art algorithm being that of Im and Li [29]). With this LP, they divide the time horizon into intervals with sizes of different powers of two, and define a decision variable for each combination of job-machine-interval, adding the appropriate constraints. The rounding happens in a deterministic manner, by creating a bipartite graph of job-machine assignments based on the fractional load of each decision variable. Their algorithm can be adapted to work in the online case of the problem yielding an 8-approximation. The first (and only at the time of writing this thesis) improvement to this bound was achieved by Gupta et al. [23], who propose a greedy online 7.216-competitive algorithm using a combination of greedy assignments based on the load added to a machine, the technique of job delaying, and a modified WSPT schedule.

2.1.2 Scheduling under the presence of selfish agents

As we mentioned earlier in the introduction, assuming that the processing times of the jobs that entered the systems are public information is an assumption that cannot be made in a lot of real-life applications. Consider the case of an open-to-public supercomputer, where researchers have to submit their tasks and wait for them to be completed in order to yield their results. Of course, every researcher would like to minimize their own job's completion time; they are not interested in the collective sum of completion times. Therefore, they have the incentive to lie about their jobs' processing times in order to achieve a better completion time. In this case, the scheduling problem of minimizing the sum of completion times can be studied from the perspective of *algorithmic mechanism design*. In fact, when Nisan and Ronen introduced the term *algorithmic mechanism design* in [39], they considered the problem of scheduling jobs in unrelated machines in the case where machines are the selfish agents, reporting the time they would need to finish a task. The objective was the minimization of the makespan. Before reviewing their line of work, we shall introduce the basic notions of mechanism design. As Hurwicz introduced it in [28] a *mechanism* is a communication system in which participants exchange messages with each other and/or with a central administration and, based on a predetermined rule, an outcome (for example, an allocation of resources) is decided for a set of messages. The mechanism should always guarantee *truthfulness*, meaning that participants who lie about their information should never benefit because of it, while agents who present their true information should never be damaged for playing fairly. As we mentioned above, the problem of truthful scheduling to minimize the makespan in the case where machines are selfish agents is a well-studied hard problem. To start with, Nisan and Ronen conjectured that any truthful algorithm has an approximation ratio of at least $\Omega(n)$, where n is the number of machines, and actually proved a lower bound of 2 [39]. Since then, there was a plethora of works gradually improving this lower bound [9, 18, 34, 12, 7] until the conjecture was proven in [8] closing a long-standing open

problem. Of course, this is not the only setting under which truthful scheduling has been investigated. In [19], Gkatzelis, Markakis and Roughgarden propose a constant-approximation mechanism that minimizes the sum of completion times when the private information is the job's weight. Furthermore, in the setting where the private information is the jobs' processing times, Feldman et al. [17] propose a mechanism using posted prices to approximately minimize the makespan. In this thesis, we consider the case where the private information is the jobs' processing times, they are to be reported by the jobs themselves, and the objective is to minimize the sum of completion times. We review two lines of work that tackle this scheduling problem from two very different perspectives. The first is the work of Angel et al. [1], which utilizes a *direct revelation mechanism*. A direct revelation mechanism works in the following way:

- First, each agent reports their individual valuation functions to the mechanism
- Then, the mechanism decides on an outcome based on the above reports.

In our scheduling problem, each agent (job) reports their processing time to the mechanism, and then the mechanism decides on a close-to-optimal schedule. Naturally, due to Smith's SPT rule [49], the jobs have no incentive to report a higher than true value. In order to guarantee that the jobs do not underbid their processing time, the authors propose a punishment rule they call *preventive preemption*, which ensures that any lying job will complete later than any truthful job. Using a completely different philosophy, Eden et al. [15] provide a truthful *menu-based mechanism*, with the main goal of also making it *prompt*, meaning that every incoming job immediately finds out its completion time on arrival. Their menu-based mechanism presents each job with a number of possible timeslots where they can be executed and lets each job choose one by itself, therefore also guaranteeing privacy to the jobs. The sequence of timeslots is built in a way such that larger jobs are slightly delayed in order to quicker accommodate smaller jobs, similar to *job delaying*. Also, truthfulness is achieved as any job that chooses a slot smaller than its processing time in order to "cheat" is never actually completed. As we will see in chapter 4, this prompt mechanism has a significantly worse competitive ratio than the simple truthful mechanism in [1]. Naturally, the question arises; why would we want to sacrifice performance to guarantee promptness? The main motivation behind prompt scheduling algorithms is the uncertainty that large jobs have to tolerate in any other case. If we consider Smith's rule, one can observe that a large job can be indefinitely postponed by smaller jobs that arrive while the machine is occupied. This large job has zero guarantees as to when it will be executed; the algorithm simply lets it hang around until it decides that it is time for it to be assigned. A prompt algorithm combats this inconvenience by immediately determining the completion time c_j of an incoming job. To better illustrate how useful this property is in a lot of real-life scenarios, let us revisit the example of the supercomputer; a prompt algorithm operates as if it is informing the researcher: "Come back at time t, your job will be ready then".

2.2 Our contribution

Drawing motivation from the results of [15] and [1] in the problem of truthful scheduling of jobs in identical machines, our goal was to explore truthful scheduling algorithms for the much more general setting of unrelated machines. We note that, in the case of unrelated machines, a direct revelation mechanism would not be easy to implement. In addition to having an incentive to underbid their processing times in order to get a better completion time (which can get solved by introducing punishments for underbidding), in our case, jobs also have an incentive to overbid their processing times in some machines in order to manipulate their assignment to a specific machine. Since this incentive is inherently very hard to counteract, we shifted our focus toward the family of prompt menu-based mechanisms. In chapter 5 we propose a tight $O(\log P_{max})$ non-truthful prompt scheduling algorithm for the case of unrelated machines. Moreover, in chapter 6 we propose

a truthful algorithm for the same setting and present the indications we have that its performance is comparable to the algorithm in chapter 5. While we were unable to prove its actual competitive ratio, we support the above claim by conducting an experimental evaluation of our proposed algorithm, comparing it to the state-of-the-art greedy online algorithm 7 and also to a lower bound of the optimal solution obtained by solving the LP 3.3.1. This experimental evaluation also shows (somewhat surprisingly) that the simple greedy algorithm 7 performs much better than its theoretical guarantees, in many cases being almost optimal given that it is designed to be non-preemptive.

2.3 Organization

In chapter 3, we give a brief review of the most important works on minimizing the sum of completion times both in the offline and in the online version of the problem. We analytically present two LP-relaxations that are commonly used as a backbone to most offline approximation algorithms, and also present the state-of-the-art greedy online algorithm of Gupta et al. [23]. In chapter 4, we present the two main works of Angel et al. [1] and Eden et al. [15] on truthful scheduling for selfish agents. In chapter 5 we present a non-truthful algorithm based on the work of Eden et al. [15] for the setting of unrelated machines, using some ideas from Gupta et al. [23]. Finally, in chapter 6 6 we present a truthful algorithm whose competitive ratio we have been unable to analyze, supporting it with an experimental evaluation that suggests that its performance is close to the theoretical optimal for a prompt algorithm. In those experiments, we also measure the performance of algorithm 7, noting that it performs a lot better than its theoretical upper bound.

Chapter 3

Scheduling for the Minimization of Weighted Completion Times

Job scheduling is a fundamental and very well-studied problem in Theoretical Computer Science, having a plethora of real-world applications. For example, in manufacturing, a different schedule of specific tasks amongst different machines can have implications on the total time needed to complete them, the total cost needed to complete them and other objectives that might be of interest. Moreover, the algorithm that should be used to define the optimal schedule for these tasks is sensitive to the setting in which they have to be completed. The number of machines that are available, the different processing speeds they might have, the fact that some tasks might need to be completed after others are all factors that drastically change the approach that should be used to tackle the problem.

In order to organize the different challenges that might arise when trying to solve a scheduling problem, we cite the universal notation provided by Graham et al. [21], according to which a scheduling problem can be classified into a category based on a triplet of characteristics $\alpha|\beta|\gamma$. There are n jobs $(j = 1, 2, ..., n)$ that should be processed on m machines $(i = 1, 2, ..., m)$. We assume that each machine can only process one job at a time and that each job can only be processed by one machine (non-malleable setting). For each job, the following information might be provided:

1. m_j : the number of operations the job requires to be deemed completed.
2. r_j : the job's release time; the time at which it becomes available to the system for processing.
3. p_{ij} : the job's processing time for machine i. Note that a job can have different processing times for different machines (we elaborate on that later).
4. d_j : the job's due date. If it is not completed before d_j the objective we are trying to optimize might suffer a penalty.
5. w_j : the job's weight, indicating it's importance (higher weight means the job should be completed earlier rather than later).
6. f_j : a non-decreasing cost function measuring the cost $f_j(t)$ if job j is completed at time t.

We will now describe what the fields $\alpha|\beta|\gamma$ might entail. α describes the machine environment and might have one or two entries (i.e. $\alpha = \alpha_1\alpha_2$) from the following:

- $\alpha_1 = 1$: Single machine, $p_{1j} = p_j$.
- $\alpha_1 = P$: Identical parallel machines, $p_{ij} = p_j$, $i \in [m]$.
- $\alpha_1 = U$: Uniform parallel machines, $p_{ij} = q_i p_i$ where q_i is the speed factor of machine i.
- $\alpha_1 = R$: Unrelated machines.

- $\alpha_1 = O$: Open shop. Each job j consists of a set of m operations $(O_{1j}, O_{2j}, ..., O_{mj}$. Operation O_{ij} has to be processed by machine i in p_{ij} time units (note that p_{ij} might be zero) and there are no constraints limiting the order in which the different operations have to be completed.
- $\alpha_1 = F$: Flow shop. Same as open shop, but now the set of operations is ordered, meaning O_{ij} must be completed before $O_{i'j}$ for $i < i'$
- $\alpha_1 = J$: Job shop. Same as flow shop, but the order in which the operations must be completed is not necessarily the same arbitrary order with which we numbered the machines (i.e. the set of operations is $(O_{m_1 j}, O_{m_2 j}, ..., O_{m_i j})$.
- α_2 can be a positive integer denoting that the number of machines is fixed. If α_2 is not given we assume that the number of machines is a variable.

The second field β will be a subset of 6 possible constraints that might be imposed:

- $\beta_1 \in \{pmtn, \emptyset\}$: If $pmtn$ then preemption is allowed, which means that at any time we might interrupt a job's execution and continue it later. If empty, then each job that starts its execution must be completed with no interruption.
- $\beta_2 \in \{res, res1, \emptyset\}$: Used in resource allocation problems. If res, then it is assumed that there exists a set of resources R_h, $h = (1, 2, ..., s)$ and that each job j requires the use of r_{hj} units of R_h at all times during its execution. A resource cannot be used by more than one job at a specific time index. If $res1$ it is assumed that there exists a single resource. If empty, there are no resource constraints.
- $\beta_3 \in \{prec, tree, \emptyset\}$: If $prec$ then a precedence relation $<$ is defined between different jobs. It can be described by a directed acyclic graph G. If G has a path from i to j then $J_i < J_j$ which means that job j has to be completed before job i can start being executed. If $tree$ then G is a rooted tree with having either outdegree at most one or indegree at most one for all vertices. If empty, then there are no precedence constraints.
- $\beta_4 \in \{r_j, \emptyset\}$: If r_j then each job has a release time r_j at which it becomes available for execution. If empty, we assume that all release times $r_j = 0$.
- $\beta_5 \in \{m_i \leq \bar{m}, \emptyset\}$: In the first case, m_i is upper-bounded by a constant (can only be used when $\alpha_1 = J$).
- $\beta_6 \in \{p_{ij} = 1, \underline{p} \leq p_{ij} \leq \overline{p}, \emptyset\}$: If $p_{ij} = 1$ then we assume unit length jobs. If $\underline{p} \leq p_{ij} \leq \overline{p}$ then the job processing times are lower and upper-bounded respectively. If empty, there are no limits to the processing times.

Finally $\gamma \in \{f_{max}, \sum f_j\}$ describes the optimality criterion chosen, and can be one of the following (among others that are omitted):

- C_j : Completion time. When minimizing $C_{j,max}$ the problem is called makespan minimization. When minimizing the sum, the problem is called minimization of the sum of completion times.
- $L_j = C_j - d_j$: Lateness.
- $T_j = \max 0, C_j - d_j$: Tardiness.
- $U_j =$ if $C_j \leq d_j$ then 0, else 1. Unit penalty.

In this chapter, we will review the most notable previous work on the minimization of the sum of completion times $\sum C_j$ in different (α, β) settings.

3.1 Weighted Completion Times Minimization on A Single Machine

In Graham notation, the problem of scheduling a single machine in order to minimize the sum of completion times is denoted by $1|\sum_j c_j$. Smith [49] showed that scheduling the jobs in a non-decreasing order based on their processing times solves the problem optimally in $O(nlogn)$ time. This rule is commonly referred to as Shortest Processing Time (SPT). Similarly, when the jobs are assigned weights, it is optimal to schedule the jobs in a non-decreasing order based on the ratio $\frac{c_j}{w_j}$. When we add release times to the problem but preemption is allowed $\left(1|r_j, pmtn|\sum_j w_j c_j\right)$, the above rules can be modified to schedule the job with the Weighted Shortest Remaining Processing Time (WSRPT) at each step, solving the problem optimally. These two rules are a benchmark for job scheduling problems, as they can be used in harder instances to derive good approximation algorithms.

In the case where preemption is not allowed, the problem of minimizing the sum of weighted completion times with release times is strongly NP-Hard, even for unit weights ($w_j = 1$ for all jobs j) (Lenstra et al. [35]). For unit weights, Phillips, Stein and Wein [41] proposed the first constant-factor approximation algorithm, deriving a schedule based on SRPT. First, the optimal preemptive schedule P is computed and used as a guide. The jobs are scheduled in an increasing order based on their completion times in the SRPT schedule c_j^P, adding some idle time where needed due to the release times r_j. With an elegant proof, they show that each job completes in at most $2c_j^P$ time, which means the algorithm is a 2-approximation. It is useful to note that their algorithm works in an online fashion, as being oblivious about job j before r_j does not change its behavior. Subsequently, Hoogeveen and Vestjens [26] prove that 2 is actually the lower bound in the online setting for deterministic algorithms, which means the above algorithm is the best we can do without introducing randomization. In [6] Chekuri et al. describe an $\frac{e}{e-1}$-approximation deterministic algorithm for the offline problem, and an online randomized algorithm with the same competitive ratio, also proving that the latter is optimal. Their work introduces the notion of α-schedules. Given a preemptive schedule P and a constant $\alpha \in [0, 1]$, they define $c_j^P(\alpha)$ as the time at which an $\alpha-$fraction of job j has been completed. Then, an $\alpha-$schedule is the non-preemptive schedule obtained by ordering the jobs increasingly based on their $c_j^P(\alpha)$. Choosing α with the density function $f(\alpha) = \frac{e^\alpha}{e-1}$ results in an optimal online $\frac{e}{e-1}$-competitive randomized algorithm.

In the case of minimizing the sum of weighted completion times with no preemption, forming a non-preemptive schedule based on the optimal preemptive schedule is not the best solution. Instead, several works use linear programming (LP)-based algorithms, with several LP relaxations having been proposed (non-preemptive and preemptive time-indexed LP relaxation, completion time relaxation and more, see Hall et al. [24], Queyranne [42], Queyranne and Schulz [43], Dyer and Wolsey [14], Belouadah et al. [4]). Goemans et al. [20] combine an LP relaxation with an α-schedule to provide an 1.6853-approximation algorithm for this problem.

A final technique that should be noted is derived from the online versions of single machine job scheduling problems and is called *job delaying*. In [51] Vestjens describes an instance in which any online algorithm that schedules jobs as soon as the machine is idle can have an arbitrarily bad performance. Assume the following instance: A job j with processing time 1 is available at time 0. Then, at time ε a set of $n-1$ jobs with processing times 0 arrive. Obviously the optimal offline schedule has a total cost of $n\varepsilon + 1$ whereas any online algorithm that does not delay the first job has cost at least n, which means the algorithm is $\frac{1}{\varepsilon}-$competitive where ε can be arbitrarily small. Vestjens proves that it is optimal to delay the first job until the time index $1 - \frac{1}{n} - \varepsilon$ for this instance, proving in the process that no online algorithm can have a competitive ratio better than 2 for the online version of the problem $1|r_j|\sum_j w_j c_j$.

All of the above techniques have been used to provide algorithms for the generalizations of these problem in the settings of identical and unrelated machines, some of which we will review in the following sections.

3.2 Weighted Completion Times Minimization on Identical Machines

In this section, we will briefly go through the literature of scheduling parallel identical machines to minimize the sum of completion times. The problem of scheduling identical machines is of significant importance, as even in the case of two machines ($P2|\sum_j c_j$) it can be shown that the problem is NP-Hard, by reducing its instance to an instance of *Knapsack* (Bruno et. al [5]). The hardness results hold even when preemption is allowed, as shown by McNaughton [37]. Recall that we mentioned that the SRPT criterion that solves one-machine scheduling optimally can be used to derive good approximation algorithms for the NP-Hard problems in identical machines. In [38] Megow and Schulz prove that WSRPT is a 2-approximations for the problem $P|pmtn|\sum_j w_j c_j$ in the online setting. Their algorithm has convenient properties and is very commonly used to compare and analyze the competitive ratios of algorithms for different identical machine scheduling problems, one of which we will describe later on. Therefore, it is useful to present the algorithm and its proof.

Mechanism 6: Weighted Shortest Remaining Processing Time (WSRPT)

At each point in time, interrupt all jobs that are being scheduled. Then, among all available and not completed jobs, schedule the m (or fewer if less than m jobs are available) jobs with the highest priority based on their weight and *remaining* processing time ($\frac{w_j}{p_j}$).

Note that this algorithm works for the online version as well, as its decisions are only based on the characteristics of available jobs. We will now prove the following theorem:

Theorem 3.2.1. *Algorithm WSRPT produces a schedule where the sum of completion times is at most 2-times the optimal value of the solution for the offline problem* $P|r_j, pmtn|\sum_j w_j c_j$.

Proof. To prove the competitive ratio of WSRPT we will consider the time interval between a job's release time and its completion time $(r_j, c_j]$. We partition this interval into two disjoint sets of subintervals $I(j), \bar{I}(j)$. The first one contains all subintervals where job j is being processed by some machine, whereas the second one contains all other subintervals in (r_j, c_j). Note that in all subintervals belonging to $\bar{I}(j)$, since job j is not being processed, all machines should be busy processing some other jobs. By definition of the two sets, we obtain that:

$$c_j \leq r_j + |I(j)| + |\bar{I}(j)|$$

where $|I(j)|$ and $|\bar{I}(j)|$ denote the total length of all subintervals in each set. Also, by definition of $I(j)$ we have that $|I(j)| = p_j$. Note that by definition of WSRPT, during all subintervals of $\bar{I}(j)$ all machines are processing some jobs with a greater weight to processing time ratio than job j. In the worst case, all higher priority jobs are fully processed during these subintervals, meaning that $|\bar{I}(j)| = \frac{\sum_{k<j} p_k}{m}$. Now, we can give an upper bound to the sum of completion times:

$$\sum_j w_j c_j \leq \sum_j w_j (r_j + p_j) + \sum_j w_j \sum_{k<j} \frac{p_k}{m}$$

The proof concludes when we note that both parts of the sum are actually a lower bound to the optimal solution. First, for each job's completion time it obviously holds that $c_j \geq r_j + p_j$. For the second term, we can observe that $\sum_j w_j \sum_{k<j} \frac{p_k}{m}$ can be modeled as the objective function when relaxing the problem to scheduling an m-speed scaled single machine (a single machine that has m-times the speed of each individual machine in our identical machines environment). As the m-speed scaled $1|\sum_j w_j c_j$ is a relaxation of our problem, its solution is at least as good as our problem's optimal solution. Therefore we conclude that:

$$\sum_j w_j c_j \leq 2OPT$$

and we have proved that WSRPT is 2-competitive. □

Furthermore, we can use an instance provided in [44] by Schulz and Skutella to prove that the above algorithm is tight:

Lemma 3.2.1. *For any number of machines, the competitive ratio of WSRPT is lower bounded by 2 for the online problem* $P|r_j, pmtn| \sum_j w_j c_j$.

Proof. We will first describe the instance where WSRPT cannot achieve a better solution than 2 times the optimal one. Assume we have m copies of $n+1$ jobs (practically, each one of these m copies is fully assigned to one machine). These $n+1$ jobs have the following characteristics: $w_j = 1$, $p_j = n - \frac{j}{n}$, $r_j = jn - \frac{j(j+1)}{2n}$ for all $0 \leq j \leq n$. WSRPT preempts every job exactly when it has just $\frac{1}{n}$ processing time left, and only finishes it when all jobs with a greater release time have been finished. Therefore, it holds that:

$$cost(ALG) = \sum_j w_j c_j = m \left(\sum_{j=0}^{n} r_n + p_n + \frac{j}{n} \right) = n^3 + n^2 - \frac{(n+1)^2}{2} + n^2 - 1 + \frac{n+1}{2}$$

In contrast, the optimal schedule is actually non-preemptive and has the value:

$$cost(OPT) = m \left(\sum_{j=0}^{n} r_j + p_j + \frac{j}{n} \right) = \frac{n^3}{2} + \frac{n^2}{2} - \frac{(n+1)(n+2)}{6} + n - \frac{n+1}{2} + \frac{n+1}{2}$$

It is clear that as n goes to infinity, the limit of the ratio between ALG and OPT becomes 2. □

From the above, we derive that WSRPT is a tight 2-competitive algorithm for the online problem of scheduling n jobs in m identical machines. We will use this fact to prove the competitive ratio of other scheduling algorithms in later chapters. Now, we shall provide an overview of the most notable results for offline and online job scheduling problems in the identical machines setting. In the same paper ([38]) Megow and Schulz provide a 2-competitive algorithm for the online problem without preemption. They argue that scheduling a "long" job with a high priority ratio $\frac{w_j}{p_j}$ might be very suboptimal, as a job with an even higher priority ratio might arrive right after and be delayed for a long time. To combat this, they propose SHIFTED WSPT, in which they assign a modified release time r'_j to each job, which is a number between $\max\{r_j, \alpha p_j\}$ and $r_j + \alpha p_j$ for some constant $\alpha \in (0, 1]$. During all time indices prior to r'_j (even the ones after r_j) the algorithm acts as if it is oblivious to the existence of job j. Technically, this algorithm works exactly the same as using WSRPT to solve $P|r'_j| \sum_j w_j c_j$. It is 3.28-competitive for a variable number of machines m.

In [44] Schulz and Skutella provide a randomized 2-competitive algoithm for the online problem $P|r_j| \sum_j w_j c_j$ by combining the α-schedule technique with randomization during the phase of assigning jobs to machines. Their algorithm, RANDOM ASSIGNMENT, first solves the problem $1|r_j| \sum_j w_j c_j$ assuming an m-speed scaled single machine and then assigns each job to a machine randomly and uniformly. Using the shadow α-schedule produced by the single machine, it schedules each job j in order of their non-decreasing α-fraction completion times in the same way that Chekuri et. al [6] described for the single machine problem, as stated in the previous section.

Using both the single machine α-schedule technique and LP relaxations, Correa and Wagner [11] improved the deterministic bound of the online non-preemptive version of the problem to 2.62 with their algorithm NAS. In the same paper, they provide a $(2 - \frac{1}{m})$-competitive randomized algorithm named CW2005. Combining the speed-scaled single machine α-schedule technique and the job delaying technique proposed in [38] (Megow and Schulz), Sitters [47] gave an $(1.79(1 + \frac{1}{\sqrt{m}})^2)$ upper bound with an algorithm she named ONLINE. Using the best of ONLINE and CW2005 for different values of m (the threshold being 320) gives a 1.997-competitive algorithm for the non-preemptive scheduling of identical machines.

3.3 Weighted Completion Times Minimization on Unrelated Machines

The third machine environment that will be analyzed in this thesis is that of unrelated machines. In this setting each job j has m different processing times p_{ij}, $i \in [m]$, all of which are completely unrelated to each other. It is clear that this setting is a generalization of identical machines, an instance of identical machines can be replicated by choosing all p_{ij} to be equal. In contrast to the identical machines setting, where a PTAS exists, scheduling unrelated machines to minimize the sum of weighted completion times is APX-Hard (Hoogeveen et. al [27]).

3.3.1 Offline version

The first constant-factor approximation ($\frac{16}{3}-$approximation) of the problem $R|r_{ij}|\sum_j w_j c_j$ was given by Hall et al. [24]. In their work, they use an *interval-indexed* linear programming formulation, where the decision variables indicate the time interval during which a job is completed. The time horizon is split into intervals whose endpoints follow a geometric progression. Then, for each interval, a set of jobs is scheduled whose sum of processing times is no larger than the interval's length. The so-called "interval-indexed" LP relaxation is the following:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} w_j \sum_{i=1}^{m} \sum_{\ell=1}^{L} \tau_{\ell-1} x_{ij\ell} \\
\text{subject to} \quad & \sum_{i=1}^{m} \sum_{\ell=1}^{L} x_{ij\ell} = 1 && j \in [n] \\
& \sum_{j=1}^{n} p_{ij} x_{ij\ell} \leq \tau_\ell, && i \in [m] \quad \ell \in [L] \\
& x_{ij\ell} = 0 && \text{if } \tau_\ell < r_{ij} + p_{ij} \\
& x_{ij\ell} \geq 0 && i \in [m], j \in [n], \ell \in [L]
\end{aligned}
$$

In order to round the LP solution, the authors at first completely disregard the interval indices in order to match jobs to their respective machines. They form a bipartite graph in the following way (first proposed in Tardos and Shmoys [46]):

On a high level, they create k copies of each machine node where $k = sum_j x_{ij}$. They also create a node for each job j. A bipartite graph is created in the following manner: Add job-machine edges to the first copy of a machine until the first index j' where $sum_{j=1}^{j'} x_{ij} \geq 1$. Then, continue by adding the edges between the remaining jobs and the second machine until it also reaches 1 and so on. Finally, when the bipartite graph is constructed, find the minimal cost matching that exactly matches each job to one machine. Finally, schedule each job in the time interval that the LP calculated for the specific variable x_{ij}, with the ordering being arbitrary if two jobs are to be processed in the same machine during the same interval.

In [44] Schulz and Skutella use a similar time-indexed LP but with randomized rounding to provide a (2+ε)-approximation. Independently, Skutella [48] improved the bound, giving a 2-approximation based on the randomized rounding of a very simple convex quadratic program's solution. Further improving upon this 2-approximation had been a challenging open problem for many years since then. 15 years later, Im and Li [30] provided a 1.8786-approximation algorithm using yet another time-indexed LP paired with a novel rounding and scheduling technique, which uses the idea of job delaying paired with randomization. First, they solve the LP and randomly assign each job to a machine i_j based on the values of its decision variables. Then, they fix a distribution Θ over $[0, 1]$ where no number appears with positive probability and independently draw a θ_j for each job j from Θ, assuming that each job has a distinct θ_j. Finally, they invoke a modified release time $r'_j = s_j + \theta_j p_{i_j j}$, and then schedule the jobs in WSPT order based on the modified release times.

In [48] Skutella also gave a long-standing 1.5-approximation algorithm for the offline problem

with no release times $R|\sum_j w_j c_j$. Coincidentally, this bound also lasted exactly 15 years, until Bansal, Srinivasan and Svensson [3] improved upon it with a $\left(1.5 - 10^{-7}\right)$-approximation using a lift-and-project based SDP relaxation. Li [36] provided a $\left(1.5 - \frac{1}{6000}\right)$ follow-up result using the same technique as a black box. The current state-of-the-art result comes from Im and Li [29], with a 1.45-approximation based on the same SDP relaxation and bipartite-rounding techniques.

A completely different approach to the problem is that of the "Configuration LP". Sviridenko and Wiese [50] designed a PTAS for the problem $R||r_{ij}||\sum_j w_j c_j$ using the following LP relaxation:

$$\begin{aligned}
\text{min} \quad & \sum_{i \in M} \sum_{S \in S(i)} y_{i,S} \cdot W_{i,S} \\
\text{subject to} \quad & \sum_{S \in S(i)} y_{i,S} \leq 1 && \forall i \in M \\
& \sum_{i \in M} \sum_{S \in S(i)} y_{i,S} \geq 1 && \forall j \in J \\
& y_{i,S} \geq 0 && \forall i \in M,\, S \in S(i)
\end{aligned}$$

In the above relaxation, M is the set of machines, J is the set of jobs, and $S(i)$ is the set of all possible schedules of machine i. Note that $j \in S$ if job j appears on the schedule S. The configuration LP might have only a linear number of constraints, but it has an exponential number of variables. Therefore, it is not directly solvable. To tackle this issue, they solve the dual LP by the ellipsoid method and a polynomial separation routine. The dual LP is the following:

$$\begin{aligned}
\text{max} \quad & \sum_{j \in J} \beta_j - \sum_{i \in M} \alpha_i \\
\text{subject to} \quad & -\alpha_i + \sum_{j \in J} \beta_j \leq W_{i,S} \forall i \in M,\, \forall S \in S(i) \\
& \alpha_i \geq 0 && \forall i \in M \\
& \beta_i \geq 0 && \forall j \in J
\end{aligned}$$

For this separation problem, we want to either find a schedule $S \in S(i)$ such that $-\alpha_i + \sum_{j \in J} \beta_j > W_{i,S}$ or assert that for all schedules it holds that $-\alpha_i + \sum_{j \in J} \beta_j \leq W_{i,S}$. The authors provide a reduction to the problem of scheduling one machine with rejections and minimizing the sum of weighted completion times plus the sum of rejections, where each job's rejection penalty e_j equals β_j. This problem is a well-studied NP-Hard problem [25, 16, 45]. However, in order to approximate the original configuration LP within a factor of $(1 + \varepsilon)$ we can solve the following relaxed version of the dual LP:

$$\begin{aligned}
\text{max} \quad & \sum_{j \in J} \beta_j (\sum_t x_{s,t} + \sum_{s \in Q(P)} x_{s,j}) - \sum_{j \in J} x_{t,j} \cdot R_{t+1} - \sum_{j \in J} \sum_{s \in Q(P)} x_{x,j} \cdot end(s) \\
\text{subject to} \quad & \sum_t x_{t,j} + \sum_{s \in Q(P)} x_{s,j} \leq 1 && \forall j \in J \\
& \textstyle\sum_{j \in J} xs, j \leq 1 && \forall s \in Q(P) \\
& \textstyle\sum_{j \in J} p_j \cdot x_{t,j} \leq rem(t) && \forall t \\
& x_{t,j} \geq 0 && \forall t,\, \forall j \in J :\, r_j \leq R_t \wedge p_j \leq \varepsilon \cdot I_t \\
& x_{s,j} \geq 0 && \forall s \in Q(P),\, \forall j \in J :\, p_j \leq size(s) \wedge r_j \leq begin(s)
\end{aligned}$$

Elaboration on the notation used above is needed. We define $R_x = (1+\varepsilon)^x$ and $I_x = [R_x, R_{x+1}]$. $Q(P)$ denotes the set of jobs in a schedule S that start in an interval I_x such that $p_j \geq \varepsilon \cdot I_x$ (which means they are large enough to not be contained solely between that interval). All jobs that are not in $Q(P)$ are considered *small* jobs. $size(s)$ denotes the size of slot s, $begin(s)$ and $end(s)$ denote its begin time and ending time, and $rem(t)$ denotes the remaining time of interval I_t to be

allocated to small jobs (as a fraction of it has been allocated to a large job not completed within a previous interval). The above LP is the dual to an analogous relaxation of the original configuration LP and can be solved exactly. Finally, to obtain a feasible solution for the original configuration LP, the authors provide a lemma that allows to compute for each schedule S of the relaxed LP a convex combination of configurations $S_1, \ldots, S_B$ and coefficients $\lambda_1, \ldots, \lambda_B$ in order to satisfy the original set of feasible schedules.

3.3.2 Online version

For the online version of $R|r_{ij}|\sum_j w_j c_j$ the first improvement upon the results obtained in [24] was achieved by Gupta et al. [23] 20 years later. Surprisingly, they give a simple online greedy algorithm which is a 7.216-approximation of the optimal solution. Because of its simple nature, it can be used as an auxiliary algorithm in order to prove the competitive ratios of other algorithms that will be analyzed later. Therefore, it is useful analyze its own proof to gain useful insight on why it works. First, we shall describe the algorithm:

Consider a job j that is released at time r_j and has processing time p_{ij} for each machine i, $i \in [m]$. The algorithm works by going through the following steps:

Mechanism 7: Greedy online algorithm minimizing sum of weighted completion times

---

1. Define a modified release time r_{ij} for each machine i, $i \in [m]$. The release times are defined as follows; $r_{ij} := \max\{r_j, c \cdot p_{ij}\}$ for a constant c. Parameter c will be optimized later, as it also depends on other constants.

2. Assign each job to a machine based on the following criterion. Let $U_i(t)$ be the set of jobs that have already been assigned to machine i at time t and *have not yet been started*. Define $cost(j \to i)$ as an upper bound of the cost that the assignment of job j to machine i would impose on a theoretical WSPT schedule of jobs $U_i(r_j)$. We will define $cost(i \to j)$ and explain the need to use an upper bound later.

3. On each machine, schedule the jobs exactly as in WSPT. That is, as soon as machine i falls idle at time t, schedule the job among $U_i(t) \cap \{j | r_{ij} \leq t\}$ with the highest ratio $\frac{w_j}{p_{ij}}$.

Definition 3.3.1 ($cost(j \to i)$). *For job j and machine i, we define $cost(j \to i)$ as:*

$$cost(j \to i) := w_j \left(\left(1 + \frac{1}{c}\right) r_{ij} + p_{ij} + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} \geq \frac{w_j}{p_{ij}}} p_{ik} \right) + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} < \frac{w_j}{p_{ij}}} w_k p_{ij}$$

We will now prove the following lemma. In the proof, when considering a job j assigned to a machine i, a *higher priority* job j' is one where $\frac{w_{j'}}{p_{ij'}} \geq \frac{w_j}{p_{ij}}$. Similarly, a *lower priority* job j' is one where $\frac{w_{j'}}{p_{ij'}} < \frac{w_j}{p_{ij}}$:

Lemma 3.3.1. *Define $m(j)$ as the machine to which job j was assigned by the greedy algorithm. Then:*

$$ALG \leq \sum_j cost(j \to m(j))$$

Proof. To prove the lemma, we shall give an upper bound to the contribution of job j to $\sum_j w_j c_j$. Recall that when a job j is released at time r_j it is immediately assigned to the machine i that minimizes $cost(i \to j)$. First, we shall derive the latest time at which job j might start being

executed. By definition of the algorithm, it cannot start at a time earlier than r_{ij}. At time r_{ij} another job h could be under execution in machine i, which will further delay job j's execution by $X_i(r_{ij})$ time units. Finally, job j will be delayed by all jobs in $U_i(r_j)$ with *higher priority*. Therefore, the increase of $\sum_j w_j c_j$ attributed to job j will be at most:

$$w_j \left(r_{ij} + X_i(r_{ij}) + p_{ij} + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} \geq \frac{w_j}{p_{ij}}} w_k p_{ij} \right) + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} < \frac{w_j}{p_{ij}}} w_k p_{ij} \leq cost(j \to i)$$

To prove that the inequality holds, it remains to show that $X_i(r_{ij}) \leq \frac{r_{ij}}{c}$. Indeed, since job h is being processed at time r_{ij}, and therefore $r_{ih} \leq r_{ij}$, it holds that:

$$X_i(r_{ij}) \leq p_{ih} \leq \frac{r_{ih}}{c} \leq \frac{r_{ij}}{c}$$

Notice that our proof only uses the jobs released prior to r_j. At the time indices between r_j and r_{ij}, some (or several) job j' with *higher priority* than job j might be assigned to machine i, and it might delay the execution job j further. However, the upper bound we defined does not account for these jobs. That is done on purpose as these delays will be accounted for when calculating the individual costs $cost(j' \to i)$. For the set of jobs with *lower priority* released between r_j and r_{ij}, the maximal delay that can be imposed upon job j is $X_i(r_{ij})$ as we defined it above, as job j will be scheduled ahead of all other non-scheduled *lower priority* jobs. Summing over all jobs j it follows that $\sum_j w_j c_j \leq \sum_j cost(j \to m(j))$. □

We shall now prove the following theorem:

Theorem 3.3.1. *ALG has a 7.216 competitive ratio for minimizing the sum of weighted completion times $\sum_j w_j c_j$ on unrelated machines. That means that $ALG \leq 7.216 \cdot OPT$*

Proof. Let $m(j)$ be the machine to which job j got assigned. We define

$$\alpha_j := cost(j \to m(j))$$

$$\beta_{i,s} := \sum_{k: m(k)=i;\ r_k \leq s\ c_k \geq s} w_k$$

By definition $\sum_{i,s} \beta_{i,s} = ALG$ and by (3.3.1) $ALG \leq \sum_j \alpha_j$. Now, consider an f-speed-scaled instance, where we modify the release times and processing times by a factor f as follows:

$$r_j^f := \frac{r_j}{f} \quad \text{and} \quad p_{ij}^f := \frac{p_{ij}}{f}$$

. It follows that:

$$r_{ij} := \frac{r_{ij}}{f}$$

.

Note that the presence of f only serves to scale time by this factor. All assignments of jobs to machines remain unchanged. Therefore, we can define a new upper bound on the increase of the sum of weighted completion time α_j^f and a new total weight of assigned-but-unfinished jobs on machine i at time s $\beta_{i,s}^f$ as follows:

$$\alpha_j^f := \frac{\alpha_j}{f} \beta_{i,s}^f := \beta_{i,(f \cdot s)} = \frac{\beta_{i,s}}{f} \tag{3.1}$$

Finally, we define the new total cost of ALG in the speed-scaled instance as $ALG^f = \sum_{i,s} \beta_{i,s}^f \leq \sum_j \alpha_j^f$. Using (3.3.2), which gives us a lower bound on the value of the optimal solution OPT, we can derive that:

$$OPT \geq \sum_j \frac{\alpha_j^f}{\alpha} - \sum_{i,s} \frac{\beta_{i,s}^f}{b} \geq \frac{ALG}{f}\left(\frac{1}{\alpha} - \frac{1}{b}\right)$$

$$\Rightarrow ALG \leq \frac{f \cdot OPT}{1/\alpha - 1/b}$$

Setting the algorithm's parameters to $c = 2/3, a = 32/23, b = 16/3, f = 23/6$, a combination that is a feasible choice to use (3.3.2), it follows that $ALG \leq 7.216 \cdot OPT$. □

To state Lemma (3.3.2), we shall first define a linear programming relaxation Pr of the problem at hand, as well as its dual linear program Dr, that will be used to derive a lower bound for the optimal solution. Consider the following linear program:

$$\begin{aligned}
\min \quad & z^{Pr} = \sum_{j \in J} w_j c_j^{Pr} & & \\
\text{s.t.} \quad & \sum_{j \in J} y_{ijs} \leq 1 & & \text{for all } i \in M, s \in \mathbb{Z}_{\geq r_j} \\
& \sum_{i \in M} \sum_{s \in \mathbb{Z}_{\geq r_j}} \frac{y_{ijs}}{p_{ij}} & & \text{for all } j \in J \\
& c_j^{Pr} = \sum_{i \in M} \sum_{s \in \mathbb{Z}_{\geq r_j}} \left(\frac{y_{ijs}}{p_{ij}}(s + \tfrac{1}{2}) + \tfrac{y_{ijs}}{2}\right) & & \text{for all } j \in J \\
& y_{ijs} \geq 0 & & \text{for all } i \in M,\ j \in J,\ s \in \mathbb{Z}_{\geq r_j}
\end{aligned} \tag{3.2}$$

As it is a relaxation of the original problem, its solution will be at most as large as the solution given by OPT. Now, consider its dual linear program:

$$\begin{aligned}
\max \quad & z^{Dr} = \sum_{j \in J} \alpha_j - \sum_{i \in M} \sum_{s \in \mathbb{Z}_{\geq 0}} \beta_{i,s} & & \\
\text{s.t.} \quad & \frac{\alpha_j}{p_{ij}} \leq \beta_{i,s} + w_j \left(\frac{s + \frac{1}{2}}{p_{ij}} + \frac{1}{2}\right) & & \text{for all } i \in M,\ j \in J,\ s \in \mathbb{Z}_{\geq r_j} \\
& \beta_{i,s} \geq 0 & & \text{for all } i \in M,\ s \in \mathbb{Z}_{\geq r_j}
\end{aligned} \tag{3.3}$$

By duality, any feasible solution to the dual linear program Dr is a lower bound to the optimal solution of the problem. We finally state Lemma (3.3.2) as follows:

Lemma 3.3.2. *With α^f and β^f as defined in* (3.1)*, the values $\left(\frac{\alpha^f}{\alpha}, \frac{\beta^f}{b}\right)$ are a feasible solution to* (3.3)*, if $\alpha f \leq 2(2 + c)$, $\frac{1}{c} \leq f(\alpha - 1)$, and $\alpha f \geq b$. For the values $c = 2/3$, $\alpha = 32/23$, $b = 16/3$, $f = 23/6$ the objective function value of* (3.3) *yields $z^{D_r}\left(\frac{\alpha^f}{\alpha}, \frac{\beta^f}{b}\right) \geq \frac{ALG}{f}\left(\frac{1}{\alpha} - \frac{1}{b}\right) = \frac{ALG}{7+11/51}$.*

Proof. We have only to prove the feasibility of the solution $\left(\frac{\alpha^f}{\alpha}, \frac{\beta^f}{b}\right)$. From the constraints of the dual program, for any job j and machine i it holds that:

$$\frac{\alpha_j}{p_{ij}} \leq \beta_{is} + w_j \frac{s + \frac{1}{2}}{p_{ij}} + w_j \cdot \frac{1}{2}$$

Substituting in the solution, it suffices to show that:

$$\frac{\alpha_j^f}{\alpha \cdot p_{ij}} \leq \frac{\beta_{is}^f}{b} + w_j \frac{s + \frac{1}{2}}{p_{ij}} + w_j \cdot \frac{1}{2}$$

Using the facts that $\alpha^f = \frac{\alpha}{f}$, $\beta_{is}^f = \beta_{i,fs}$, $s + \frac{1}{2} > s$ it is enough to show that:

$$\frac{\alpha_j}{p_{ij}} \leq \alpha f \cdot \frac{\beta_{i,fs}}{b} + w_j \frac{s}{p_{ij}} \cdot \alpha f + w_j \cdot \frac{\alpha f}{2} \tag{3.4}$$

Due to (3.3.1) and having chosen α_j to be the one that minimizes $cost(i \to j)$ it holds that:

$$\frac{\alpha_j}{p_{ij}} \leq \frac{w_j}{p_{ij}} \cdot \left(\left(1 + \frac{1}{c}\right) r_{ij} + p_{ij} + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} \geq \frac{w_j}{p_{ij}}} p_{ik} \right) + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} < \frac{w_j}{p_{ij}}} w_k \tag{3.5}$$

Therefore we have to show that the RHS in (3.5) is upper bounded by the value of the RHS in (3.4). We can achieve that by showing a slightly stronger inequality. Because $\beta_{i,fs}$ is the total weight of jobs k assigned to machine i and unfinished at time fs but with $r_k \leq fs$, keeping $f \geq 1$ and since $r_j \leq s$ it holds that $r_j \leq fs$. Consequently, we have that:

$$\beta_{i,fs} \geq \sum_{k:m(k)=i, r_k \leq r_j, c_k \geq fs} w_k \geq \sum_{k \in U_i(r_j), c_k \geq fs} w_k$$

Therefore (3.4) is actually bounded from above by:

$$\begin{aligned} &\frac{\alpha f}{b} \cdot \sum_{k \in U_i(r_j), c_k \geq fs} w_k + w_j \frac{s}{p_{ij}} \cdot \alpha f + w_j \cdot \frac{\alpha f}{2} \\ &= \left(\frac{\alpha f}{b} \cdot \sum_{k \in U_i(r_j), c_k \geq fs} w_k + \frac{w_j}{p_{ij}} (fs - r_j) \right) + \frac{w_j}{p_{ij}} (fs(\alpha - 1) + r_j) + w_j \cdot \frac{\alpha f}{2} \end{aligned}$$

Thus, finally, we have to prove that:

$$\begin{aligned} &w_j \cdot \left(\left(1 + \frac{1}{c}\right) r_{ij} + p_{ij} + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} \geq \frac{w_j}{p_{ij}}} p_{ik} \right) + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} < \frac{w_j}{p_{ij}}} w_k p_{ij} \\ &\leq \left(\frac{\alpha f}{b} \cdot \sum_{k \in U_i(r_j), c_k \geq fs} w_k p_{ij} + w_j (fs - r_j) \right) + w_j (fs(\alpha - 1) + r_j) + w_j p_{ij} \cdot \frac{\alpha f}{2} \end{aligned}$$

We can rewrite this as:

$$\begin{aligned} &w_j \cdot \underbrace{\left(\left(1 + \frac{1}{c}\right) r_{ij} + p_{ij} \right)}_{I} + \underbrace{\sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} \geq \frac{w_j}{p_{ij}}} w_j p_{ik} + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} < \frac{w_j}{p_{ij}}} w_k p_{ij}}_{II} \\ &\leq \underbrace{\frac{\alpha f}{b} \cdot \sum_{k \in U_i(r_j), c_k \geq fs} w_k p_{ij} + w_j (fs - r_j)}_{II^*} + w_j \underbrace{\left((fs(\alpha - 1) + r_j) + p_{ij} \cdot \frac{\alpha f}{2} \right)}_{I^*} \end{aligned}$$

Now we can prove separately that:

1. $I \leq I^*$: We split the proof in two cases. If $r_{ij} = r_j$ then $I = (1 + \frac{1}{c})r_j + p_{ij}$, and since $s \geq r_j$, $I^* = \left((fs(\alpha - 1) + r_j) + p_{ij} \cdot \frac{\alpha f}{2}\right) \geq r_j + f(\alpha - 1)r_j + p_{ij} \cdot \frac{\alpha f}{2}$. Therefore we get that $I \leq I^*$ if $1/c \leq f(\alpha - 1)$ and $\alpha f \geq 2$. If $r_{ij} = cp_{ij}$ then $I = (2 + c)p_{ij}$ and we get that $I \leq I^*$ when $2(2 + c) \leq \alpha f$ for every $\alpha \geq 1$. To get that $I \leq I^*$ in both cases, we need the conditions $1/c \leq f(\alpha - 1)$ and $2(2 + c) \leq \alpha f$

2. $II \leq II^*$: By definition of $U_i(r_j)$ it holds that:

$$w_j(fs - r_j) \geq w_j \sum_{k \in U_i(r_j), c_k < fs} p_{ik}$$

.

Therefore, conditioned on $\frac{\alpha f}{b} \geq 1$ we have that $II \leq II^*$, as:

$$II^* - II = \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} \geq \frac{w_j}{p_{ij}}, c_k \geq fs} (w_k p_{ij} - w_j p_{ik}) + \sum_{k \in U_i(r_j), \frac{w_k}{p_{ik}} < \frac{w_j}{p_{ij}}, c_k < fs} (w_j p_{ik} - w_k p_{ik}) \geq 0$$

□

Chapter 4

Truthfulness and Promptness in Scheduling

Assuming that every job's metadata is publicly available at all times is an assumption that does not work in every real-life application of scheduling problems. On many occasions, a job's processing time or other useful information might be held privately, and exposed to the algorithm by the jobs themselves. In these cases, each job can be modeled as a selfish agent, that might reveal false information if that would cause it to receive a better result from the algorithm. To combat this, an important research question in the area of Mechanism Design has been to create *truthful* algorithms in order to guarantee that the jobs will have no incentive to lie about any private information they hold. Once again, the primary objective researchers are interested in is the minimization of the makespan. Since the volume of literature involving makespan minimization is vast, we shall only briefly review the most important works on this problem.

4.1 Truthful Scheduling for the Minimization of the Makespan

Minimizing the makespan when the jobs' processing time is private information is inherently a hard problem. Nisan and Ronen [39], where the first in line to tackle the problem of minimizing the makespan in unrelated machines, in the case where the machines are selfish agents that seek to minimize their load. When presented with a task, each machine has to declare t_i^j, the time it would need to process task j. Obviously, the machines have incentive to overbid this processing time, in order for it to be scheduled somewhere else. The authors applied the VCG [52, 10, 22] mechanism (that we will analyze later) and showed that it achieves an $m-$approximation where m is the number of available machines. They then formed a conjecture that no truthful mechanism can achieve an approximation better than m for this problem. A long line of works had tackled this conjecture, gradually proving tighter lower bounds. Nisan and Ronen were the first to prove a lower bound of $1 + \sqrt{2}$. A series of papers led to the improvement of this lower bound to 3 [9, 18, 34, 12]. The first super-constant lower bound of $1 + \sqrt{m-1}$ was given by Christodoulou, Koutsoupias and Kovacs [7]. Finally, the same three authors proved the conjecture to be true in [8]. The above line of works shows that the need for truthfulness can have a significant impact on a problem's hardness. In the next section, we shall further investigate the impacts of truthfulness constraints on the objective of minimizing the sum of weighted completion times.

4.2 Truthful Scheduling for the Minimization of the Sum of Weighted Completion Times

As has been stated above, the objective of interest in this thesis is the minimization of the sum of weighted completion times. In a mechanism design setting, this objective could theoretically be presented as a social welfare maximization problem, assuming the utility function of each of the jobs is modeled as $u_j = -w_j \cdot c_j$. The *de facto* family of truthful mechanisms that is used when tackling such problems is known as the VCG mechanism [52, 22, 10]. However, this family of mechanisms cannot be applied to scheduling problems. To understand the reasoning behind this, we shall

first analyze when and why the VCG mechanism works. Then, we will present two fundamentally different truthful mechanisms for scheduling parallel identical machines, based on two different philosophies. The first one is a *direct* mechanism, where the agents report their information to a central entity, which then decides on an outcome. The second is a *menu* mechanism, where an agent never really reports their privately-held information. Instead, each agent enters the system and chooses one of the possible outcomes they are presented with.

4.2.1 Preliminaries

First, we shall provide some notation and definitions to be used when describing the VCG mechanism. Assume there are n agents and a set X of possible outcomes. Each agent has a valuation function $v_i : X \to \mathbb{R}^+$ that maps each outcome to the value the agent receives from it. For the VCG mechanism to work, it is also assumed that all agents have a quasi-linear utility function:

Definition 4.2.1 (Quasi-Linear Utility). *An agent i has a* quasi-linear utility *function iff when receiving a negative or positive payment p_i, their utility becomes:*

$$u_i(v_i) = v_i + p_i$$

Now we shall define *truthfulness*:

Definition 4.2.2 (Truthfulness). *Let v_i be the true valuation function of agent i, v_i' be the one the agent presents to the mechanism and $\mathbf{v}$ be the vector of valuations presented by all agents. The mechanism is truthful iff:*

$$u_i(v_i, \boldsymbol{v}_{-i}) \geq u_i(v_i', \boldsymbol{v}_{-i}),\ \forall v_i' \neq v_i$$

Another property that is often useful in mechanisms is *individual rationality*, which dictates that an agent should not incur a negative utility by participating in the mechanism:

Definition 4.2.3 (Individual Rationality). *A mechanism $\mathcal{M}$ is* individually rational *iff for every agent i and every declared valuation function v_i*

$$u_i(v_i, \boldsymbol{v}_{-i}) \geq 0$$

A mechanism that is both truthful and individually rational is said to be Dominant Strategy Incentive Compatible (DSIC).

4.2.2 The VCG Mechanism

A VCG mechanism works as follows: First, every agent submits a valuation function $v_i : X \to \mathbb{R}^+$, describing their preferences over all possible outcomes. Based on the vector of valuation functions $\mathbf{v}$, the mechanism calculates $x^* = x_{OPT}(v)$ as follows:

$$x_{OPT}(v) = \arg\max_{x \in X} \sum_i v_i(x)$$

In the final step of the mechanism, the payments each agent has to make are calculated as:

$$p_i = \max_{x \in X}(\sum_{j \neq i} v_j(x)) - \sum_{j \neq i} v_j(x^*)$$

The above value represents the *externality* of agent i, i.e. the total utility reduction all other agents had to suffer due to agent i participating in the mechanism. Since $x^* \in X$, the payments are

always non-negative, therefore the mechanism has a net gain in the end. Individual rationality is also satisfied, as no agent is "forced" to participate if their payment would outweigh their valuation. Now, we shall prove the truthfulness of the mechanism by analyzing the utility of each agent:

$$u_i = v_i(x^*) - p_i(v) = v_i(x^*) + \sum_{j \neq i} v_j(x^*) - \max_{x \in X}(\sum_{j \neq i} v_j(x))$$

The last term is fixed for all possible actions of agent i. Therefore, to maximize their utility, each agent seeks to maximize the quantity $v_i(x^*) + \sum_{j \neq i} v_j(x^*) = \sum_j v_j(x^*)$. Observe that this quantity is exactly the quantity that the mechanism seeks to maximize, the social welfare. Therefore, each agent has the incentive to submit their true valuation function in order to "aid" the mechanism in achieving its objective and consequently maximizing each individual agent's own utility.

Angel et al. [1] proved that the VCG mechanism cannot be used even in the single machine case when preemption is not allowed:

Theorem 4.2.1. *There exists no truthful non-preemptive mechanism for minimizing the sum of completion times in a single machine.*

Proof. We shall prove the above theorem by contradiction. Assume that there exists a truthful mechanism and that the input consists of 2 jobs with processing times t_1, t_2 where $t_1 < t_2$. It is known that Smith's rule [49] produces the optimal schedule for a single machine. Let us assume job 1 declares a processing time $t'_1 = t_1$ truthfully, and job 2 declares a processing time t'_2. If $t'_2 < t_1$ then job 2 is scheduled before job 1 and obtains the utility $u_2 = -t_2 - p_2$. If job 2 does not lie, then $t'_2 > t_1$ and job 2 is scheduled after job 1, which means $u'_2 = -t_1 - t_2 - p'_2$. If the mechanism is truthful, then $u'_2 \geq u_2 \Rightarrow -t_1 - t_2 - p'_2 \geq -t_2 - p_2 \Rightarrow p_2 - p'_2 \geq t_1$. However, t_1 is not known by the algorithm (as job 1 could have also lied) and therefore the payments cannot be calculated. □

4.2.3 Preventive preemption

In order to obtain a truthful mechanism that minimizes the sum of weighted completion times in identical machines, Angel et al. [1] propose a "punishment" for jobs that lie about their processing times, which they call preventive preemption.

Definition 4.2.4. *An algorithm uses preventive preemption if it constructs a schedule in which a job j is preempted (and resumed later), if and only if, $b_j < p_j$, where b_j is the presented processing time and p_j is the true processing time.*

Having defined *preventive preemption*, we can describe the algorithm they use to optimally schedule jobs truthfully on a single machine, called WSPT-PP. WSPT-PP creates a schedule in which each job is executed during a set of time intervals $\rho_j = \{(l_1, r_1), ..., (l_k, r_k)\}$, being preempted in between them iff $b_j < p_j$.

Mechanism 8: WSPT-PP

1. Sort all jobs in WSPT order ($\frac{b_1}{w_1} \leq \frac{b_2}{w_2} \leq ... \leq \frac{b_n}{w_n}$)
2. Schedule the first interval of each job j such that $l_1^j = \sum_{i=1}^{j-1} b_j$ and $r_1^j = l_1^j + b_j$
3. After time $t = \sum_j b_j$ schedule the jobs that have been preempted using a round robin (RR) policy: For each $x \geq 2$ (where x is the number of jobs), if job i is not completed at time $\sum_j b_j + n(x-2) + i - 1$ schedule this job in the time interval $[l_i^x, r_i^x]$, with $l_i^x = \sum_j b_j + n(x-2) + i - 1$ and $r_i^x = \sum_j b_j + n(x-2) + i$

In essence, preventive preemption "punishes" a job that lies about its true processing time by sending it in a queue at the very back of the schedule, where it alternates its execution with the other lying jobs. Therefore, presenting a shorter-than-reality processing time always hurts the job's completion time. At the same time, it is never optimal for a job to overbid its size, as that will only push it further in the schedule as well. Formally, the authors prove the following theorem:

Theorem 4.2.2. *Algorithm WSPT-PP is a truthful, optimal algorithm for the single machine case where each job's private data is their processing time and the objective is the minimization of the weighted sum of completion times.*

Proof. Proving that $b_i > p_i$ does not favor the job is trivial; in that case, the job will start at a later time than if it had bid $b_i = p_i$ and therefore also finish at a later time. If $b_i < p_i$, then the job will be preempted at time b_i later than its execution start, and by definition of the algorithm it will be continued at least at time $\sum_{j=1}^{n} b_j$. Therefore for its completion time the following will hold: $c_j \geq \sum_{j=1}^{n} b_j + p_i - b_i$. If it had bid $b_i = p_i$ its starting time would be at most $\sum_{j=1}^{n} b_j$ and thus for its completion time it would be true that $c_j \leq \sum_{j=1}^{n} b_j + p_i - b_i$. We conclude that job j has no incentive to lie. Finally, if no job has an incentive to lie then the produced schedule is identical to WSPT, therefore WSPT-PP is optimal. □

We have shown that preventive preemption can be used to produce an optimal truthful algorithm for minimizing the sum of weighted completion times on a single machine. One could propose to use WSPT-PP as is, sorting the tasks in descending WSPT order and executing them in order whenever a machine becomes available, preempting any jobs that underbid and starting them again after the completion of the last task in any machine. However, the above algorithm is not truthful. Consider the following case:

Counterexample. Consider two machines and three jobs with $w_1 = w_2 = p_1 = p_2 = 1$ and $w_3 = 2, p_3 = 2 + \varepsilon$. If job 3 bids truthfully, its completion time will be $3 + \varepsilon$ in one of the two machines. However, if job 3 bids $2 - \varepsilon$, its execution will start on machine 1 and jobs 1 and 2 will be scheduled on machine 2. Therefore, even if we preempt job 3, its completion time will be $2 + 2\varepsilon$ at worst, if we continue its execution on machine 2. □

It is evident that preemptive preemption does not work in the parallel machine case as we defined it above. In [31], it is shown that imposing a large enough penalty (i.e. continuing all deferred jobs at time at least $\sum_{j=1}^{n}$) is a truthful $1 + \sqrt{2}$ approximation. In [1], the authors propose a simple, yet efficient trick to derive a truthful $\frac{3}{2}$-approximation: the assignment of jobs to machines simply happens randomly and uniformly, stripping the jobs of any agency they could have over their assignment if they lied.

4.3 Promptness in Scheduling

A second line of work that combined truthfulness with the minimization of the sum of weighted completion times in scheduling parallel machines, with a notion they call *promptness* is that of Eden et al. [15]. A scheduling algorithm is considered *prompt*, when it immediately determines (on a job's arrival) the starting time and ending time of a job's execution, without preemption. The need for prompt algorithms was born by an inherent weakness of Smith's WSPT rule and all of its online variants: a long job has zero guarantees as to when its execution will begin. In fact, if we allow preemption, even when a very large job begins getting executed, it might be indefinitely deferred by smaller jobs that arrive later.

Definition 4.3.1 (Prompt algorithms). *An algorithm is* prompt *when for every job j with release time r_j, its completion time c_j is determined definitively and irrevocably at time r_j.*

Theorem 4.3.1. *Every prompt algorithm is at least $\Omega(\log P_{max})$-competitive, where P_{max} is the maximal processing time of a job that entered the system.*

To combat this, the authors propose the following idea: Divide the time horizon into predetermined time slots, provide the incoming job with a menu of options, and simply let the job itself choose the time slot in which it will be executed. If a job chooses a time slot smaller than its own processing time, it is punished harshly, as it will never finish its execution. This fully prompt algorithm achieves a tight bound of $O(\log P_{max})$, where P_{max} is the maximal processing time out of all the jobs that arrive. Note that it is not known to the algorithm a priori. We should also note that since the algorithm uses timeslots with lengths powers of two, we round the jobs' processing time to the nearest power of two, which multiplies the algorithm's sum of completion times at most by a factor of 2.

4.3.1 The prompt sequence

To divide the time horizon into the appropriate slots, Eden et al. introduce a recursive sequence of exponentially increasing timeslots defined as such:

$$S_0 = \langle 1 \rangle$$
$$S_1 = S_0 || S_0 || 2^1 = \langle 1, 1, 2 \rangle$$
$$S_2 = S_1 || S_1 || 2^2 = \langle 1, 1, 2, 1, 1, 2, 4 \rangle$$
$$S_k = S_{k-1} || S_{k-1} || 2^k$$

Similarly, an infinite sequence is defined that contains every S_k as a prefix:

$$S_\infty = \langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 16, ... \rangle$$

The sequence's shape might not seem intuitive at first, but the reasoning behind it becomes clear with the following lemma:

Lemma 4.3.1. *For all $d \geq 0$ and for all $0 \leq k \leq d$, the volume of all 2^k time slots in S_d combined is equal to 2^d:*

$$\sum_{i:S_d[i]=2^k} 2^k = 2^d$$

The proof is achieved via induction over d:

Proof. The claim is true for S_0. Assume that it holds for S_{d-1}. Then for $S_d = S_{d-1} || S_{d-1} || 2^d$ it holds that:

$$\sum_{i:S_d[i]=2^k} 2^k = 2 \cdot 2^{d-1} = 2^d$$

□

Corollary 4.3.1. *The following hold:*

- *S_k appears in S_d 2^{d-k} times*
- *The sum of lengths of intervals in S_d is $(d+1)2^d$*

Remember that we assume every job has a processing time that is a perfect power of 2. If we divide the jobs into groups based on their processing times, the above lemma inherently suggests that each distinct group contributes equally to the sum of completion times over a fixed sequence. To better illustrate that, we can imagine that $d+1$ different groups of jobs (with processing times $2^0, 2^1, \ldots, 2^d$ are divided in equal volume into $d+1$ machines, as shown below:

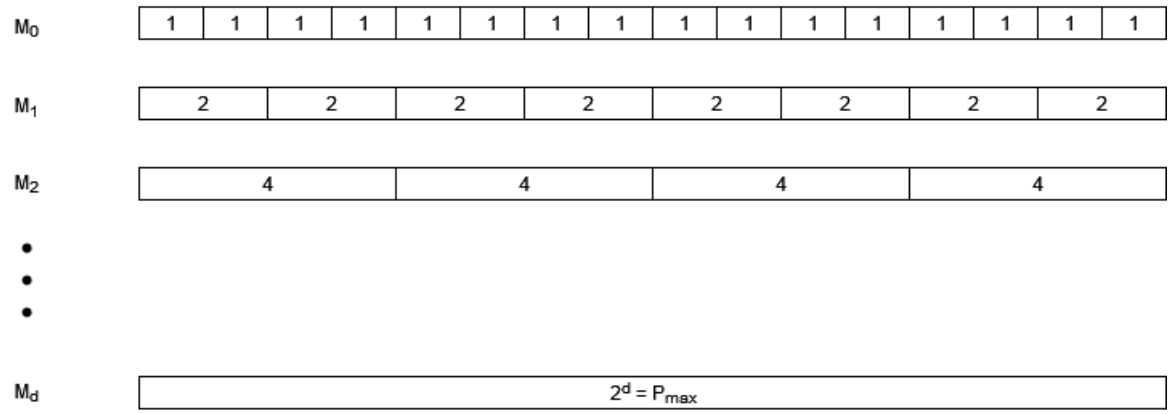

Figure 4.1: Equivalence of volume for different sizes

We shall denote by $S_k(t)$ the sequence of timeslots S_k that starts at time t (i.e. $S_1(t) = \langle [t, t+1], [t+1, t+2], [t+2, t+4] \rangle$). We denote by $b(S_k)$ the beginning of sequence S_k and by $e(S_k)$ its ending time. Also, for a job j, $D(j)$ will denote the set of jobs that were assigned to the same machine and completed no later than job j, and $I(j)$ will denote the specific interval in which job j was executed in. The authors prove that using the infinite sequence $S_\infty(0)$ on every machine yields an $\Omega(\sqrt{P_{max}})$ competitive algorithm. If we instead repeat the sequence $S_{\log P_{max}}$ on all machines, the algorithm is $O(\log P_{max})$-competitive. To prove this, we shall first prove the following lemma:

Lemma 4.3.2. *Let $d \geq 0$. Let t and q be such that some job j with $r_j \leq t, p_j = 2^k$ chose the last interval of $S_d(t)$ on machine q ($2^k \leq 2^d$). Let $D(j, q, S_d(t))$ be the set of jobs, completing no later than job j under SRPT, that execute on the same machine as job j, and occupy some interval in $S_d(t)$. I.e., $D(j, q, S_d(t)) = D(j) \cap \{j' | I(j') \in S_d(t), M(j') = q\}$. Note that $j \in D(j, q, S_d(t))$. Then,*

$$vol(D(j, S_d(t), q)) \geq 2^d$$

.

Proof. The claim is trivially true for $k = d$ and also for $d = 0$. We shall prove it by induction over d for the case when $k < d$. Assume that the claim holds for all $0 \leq d' < d$. Since $k < d$, every interval of length 2^k in $S_d(t)$ must be occupied by some other job j', otherwise job j would have chosen one of them. By the induction hypothesis, for every such job j' it holds that $vol(D(j', S_k(t'), q)) \geq 2^k$, and since there are 2^{d-k} 2^k sized slots in $S_d(t)$, adding them up leads to $vol(D(j, S_d(t), q)) \geq 2^d$. □

Corollary 4.3.2. *In the above lemma, if $d \geq 1$, replacing the condition $r_j \leq t$ with the weaker condition $r_j \leq e(S_{d-1}(t))$ gives us the weaker guarantee $vol(D(j, S_d(t), q)) \geq 2^{d-1}$.*

Proof. Recall that by construction $S_d(t) = S_{d-1}(t) || S_{d-1}(t') || 2^d$ and $t' = e(S_{d-1}(t))$. Once again, if $k = d$ the claim is trivially true since $j \in D(j)$. For the case where $k \leq d-1$, since the job arrived at time $r_j \leq t'$ and did not choose the last interval of $S_{d-1}(t')$, it means that the interval was occupied. Therefore $vol(D(j, S_d(t), q)) \geq 2^{d-1}$. □

Theorem 4.3.2. *For $P_{max} = 2^d$, the repeating static sequence $\overline{S_d} = S_d(0) || S_d((d+1)2^d) || \ldots$ produces an $O(\log P_{max})$ competitive schedule with respect to the sum of completion times.*

Proof. For simplicity, we shall notate the repeating sequences as $S_1, S_2, \ldots$, with the indices representing their order of appearance. For a job j there exists an $\alpha \geq 0$ such that:

$$e(S_\alpha) \leq r_j + p_j < e(S_{\alpha+1}(t))$$
$$e(S_{\alpha+x}) < c_j \leq e(S_{\alpha+x+1})$$

Since P_{max} is known a priori, job j can fit in every single interval of $\bigcup_{i=\alpha+1}^{\alpha+x} S_i$, which means that every one of these intervals must be occupied. Applying (4.3.2) x times leads to:

$$vol(D(j), \bigcup_{i=\alpha+1}^{\alpha+x} S_i, q) \geq x \cdot 2^d$$

Since, $c_j^* \geq r_j + p_j$, the above implies that:

$$c_j^* \geq \max\left\{r_j + p_j, x \cdot 2^d\right\} = \max\left\{\alpha 2^d(d+1), x \cdot 2^d\right\}$$

Also, from our assumption, the following holds for c_j:

$$c_j \leq e(S_{\alpha+x+1}) = (\alpha + x + 1)2^d(d+1)$$

The proof is now split into the following two cases:

1. $\alpha(d+1)2^d \geq x \cdot 2^d \Rightarrow x \leq \alpha(d+1)$. In that case, we obtain:

$$c_j \leq \frac{\alpha + x + 1}{\alpha} c_j^* \leq c_j \leq \frac{(\alpha d + 2\alpha + 1)}{\alpha} c_j^* \leq O(d) \cdot c_j^*$$

2. $\alpha(d+1)2^d \leq x \cdot 2^d \Rightarrow x \geq \alpha(d+1)$. In that case, we obtain:

$$c_j \leq \frac{(\alpha + x + 1)(d+1)}{x} c_j^* \leq \left(\frac{\alpha}{x} + 1 + \frac{1}{x}\right)(d+1)c_j^* \leq \left(\frac{1}{d+1} + 1 + \frac{1}{\alpha(d+1)}\right)(d+1)c_j^* \leq O(d) \cdot c_j^*$$

In both cases, $c_j \leq O(d) \cdot c_j^*$. □

Note that the above algorithm requires a priori knowledge of P_{max}, which is an undesired constraint for an online algorithm. In order to achieve a competitive ratio of $O(\log P_{max})$ without knowing P_{max} before the input ends, the authors propose a dynamic way of using the sequences, which we shall now describe.

4.3.2 $O(\log P_{max})$-competitive dynamic menu, unit weights

First, we will describe the algorithm used to minimize the sum of (unweighted) completion times for scheduling identical machines and then we will provide the natural extension for the weighted case. Since the machines are identical, every machine shares the same schedule. Whenever a job arrives, every machine updates its division of $[0, \max c_j]$ into time slots. A division is compiled of a set of disjoint sequences. All sequences are set (and will never change) except for the last one, which is tentative. The tentative last sequence is the largest suffix that can be a prefix of $S_\infty(t)$. Let us denote the state of the algorithm when job $j+1$ arrives as $\psi^j = (A^j, X^j)$, where A^j is the set of disjoint sequences and X^j is the set of occupied time intervals. We also denote with l_j the size of the sequence set A^j and with A_i^j the i-th sequence. When job $j+1$ with processing time 2^{p_k} arrives, it chooses a timeslot by itself and its completion time c_j is determined (in a way that we shall describe later) and the new state $\psi^{j+1} = (A^{j+1}, X^{j+1})$ is calculated in the following way:

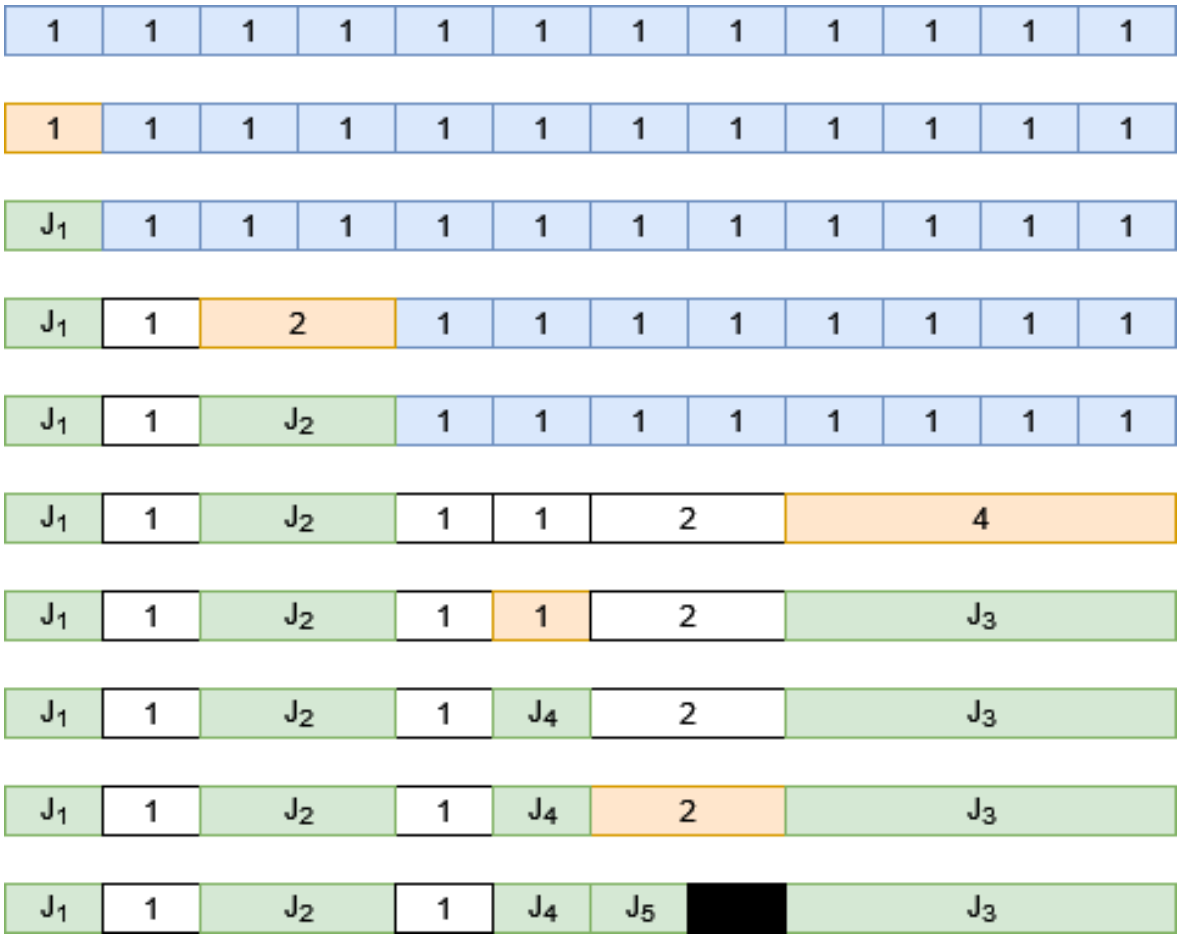

Figure 4.2: One-machine schedule for the following input: $J_1 : p_j = 1, r_j = 0$, $J_2 : p_j = 2, r_j = 0$, $J_3 : p_j = 4, r_j = 0$, $J_4 : p_j = 1, r_j = 5$, $J_5 : p_j = 1, r_j = 6$. The colors represent the following states: Blue - Tentative Sequence, Orange - Timeslot the job chooses, Green - Job locked in, White - Sequence is fixed, Black - Slot's unused fraction (cannot be occupied)

- If $c_j \leq e(A^j_{l_j})$, then the job chose an already computed unoccupied interval and nothing changes.
- If $r_j \geq e(A^j_{l_j})$, then the job arrives at a later time than has been computed. $A^j_{l_j}$ becomes set, and a new tentative sequence $A^j_{l_{j+1}}$ is created with $S_k(r_j)$. Note that this introduces a gap in the schedule, which can never happen when the input is chosen adversarially, as it benefits our algorithm's competitive ratio.
- If $r_j \leq e(A^j_{l_j})$ and $c_j > e(A^j_{l_j})$ but $A^j_{l_j}$ is of size larger than k, then $A^j_{l_j}$ is extended by plugging in $S_k(e(A^j_{l_j}))$ (note that $A^j_{l_j}$ remains a prefix of $S_\infty(t)$)
- Finally, if $r_j \leq e(A^j_{l_j})$ and $c_j > e(A^j_{l_j})$ but $A^j_{l_j}$ is of size smaller than k, then $A^j_{l_j}$ is extended, becoming $S_k(b(A^j_{l_j}))$.

We shall now define (on a high level) the algorithm that provides the arriving job with a set of time intervals from which the job chooses its actual execution timeslot. Obviously, the job chooses the earliest timeslot that is at least as big as its processing time, making the algorithm trivially truthful.

Given a state $A^j_{l_j}$ and a time t, we construct the following sequence:

$$\tau(A,t) = A_1||A_2||...||A_{l_j}||\overline{S_0}(t)$$

After job j arrives, $\tau(A,t,j)$ is updated based on the three rules outlined above. Intuitively, this means that every machine uses the repeating static sequence $\overline{S_0}$ (i.e. repeating slots of unit size) until a job j arrives that requires a bigger timeslot. Then, each machine updates to the appropriate sequence needed to serve job j, and repeats $\overline{S_0}$ after $I(j)$. Finally, we define the algorithm as the following:

Mechanism 9: Dynamic menu-offering algorithm, parallel machines

Given a job j with processing time p_j and release time r_j, and a state $A_{l_{j-1}}^{j-1}$, construct $\tau(A_{l_{j-1}}^{j-1}, r_j, j)$ and do the following:

$a \leftarrow 1$;
while *job has not picked an interval* **do**
    Offer the first interval of size 2^a on every machine;
    $a \leftarrow a * 2$;
end

The proof of this algorithm's $O(\log P_{max})$ competitive ratio is achieved by comparing its performance to SPT. Since it is very similar to the proof we construct for algorithm 10, we will simply present a proof sketch and refer to the appropriate lemmas in the next section. Formally, the theorem is the following:

Theorem 4.3.3. *For every job j, it holds that:*

$$c_j \leq O(\log P_{max}) \cdot c_j^*$$

The proof is achieved by using slightly altered versions of lemmata 5.1.1, 5.1.2, 5.1.3 and 5.1.4. First, it is proven that when job j is scheduled very close to its arrival time, its completion time is naturally very close to its corresponding completion time in the optimal solution. The second part of the proof consists of finding a lower bound to the volume of jobs that are scheduled between the arrival time of job j and its actual completion time, both in algorithm 9 as well as in the optimal solution. This is achieved by noticing that in each sequence between job j's "arrival" sequence and "completion" sequence, every single timeslot of size $\geq p_j$ has to be occupied by a job that is completed earlier than job j in SRPT.

4.3.3 Arbitrary weights

To solve the weighted case of the problem, we require prior knowledge of the maximum weight W_{max}. Then, we can simply repeat each time interval $logW_{max}$ times, with the $i-th$ repeating interval having been designed to only hold jobs of weight $w_j \geq 2^i$. The analysis of algorithm 9 using the static sequence holds when multiplying everything with $\log W_{max}$, meaning that the following theorem holds.

Theorem 4.3.4. *The prompt algorithm that schedules weighted jobs is $O((\log n + \log P_{max}) \cdot \log W_{max})$-competitive*

Chapter 5

Prompt Scheduling in Unrelated Machines

5.1 A non-truthful $O(\log P_{max})$-competitive algorithm

5.1.1 Unit weights

In this chapter, we shall provide an $O(\log P_{max})$-competitive non-truthful algorithm for minimizing the sum of weighted completion times in the unrelated machines setting. To achieve this, we combine the algorithm from Gupta et al. [23], together with the algorithm from Eden et al. [15], with some modifications to make it applicable to unrelated machines. Specifically, each machine has its own set of sequences (that is computed exactly as in Eden et al., described in the previous section). The algorithm is the following:

Mechanism 10: Non-truthful dynamic menu-based scheduling algorithm

Run a "ghost" copy of the algorithm of Gupta et al. in the background. Whenever a job arrives and declares its processing times p_{ij}:

Determine the machine it is assigned to by computing $cost(j \to i)$ for every machine ;
Compute $\tau(A^j_{l_j}, r_j)$ for the machine the job was assigned to (denoted by i);
$a \leftarrow 1$;
while *job has not picked an interval* **do**
    Offer the earliest interval of size 2^a on machine i ;
    $a \leftarrow a * 2$;
end

The idea of the above algorithm is simple: As long as we can manage to achieve the exact same job-to-machine assignments as a good-working algorithm A, the properties of the prompt sequence will guarantee that our algorithm will be at most an $O(\log P_{max})$-factor worse than A.

Let us denote by c^*_j the completion time of job j in the algorithm of Gupta et al. and by c_j the completion time of job j in our algorithm. We shall prove that $c_j \leq O(\log P_{max}) \cdot c^*_j$. This implies that $\sum_j c_j \leq O(\log P_{max}) \cdot \sum_j c^*_j$ and as algorithm (7) is a constant approximation of the optimal, our algorithm's competitive ratio will be $O(\log P_{max})$. Note that as each job has m different processing times (for each of the m unrelated machines) P_{max} here will denote the maximal realized processing time by any job. Before analyzing the proof, we should note that the algorithm is non-truthful, as the jobs can manipulate the assignment criterion. While underbidding a processing time will only damage a job (as it will never be completed if it chooses an interval smaller than its processing time), a job can overbid its processing time on some machine, in order to be assigned to a different machine that is preferable to it. Consider figure 5.1 below: A job arrives with $r_j = 0, p_1 = 2, p_2 = 4$. If it bids truthfully, the job will be assigned in machine 1, with $c_j = 14$. If it bids $p'_1 = 4, p'_2 = 4$, it will be assigned in machine 2, with $c_j = 12$.

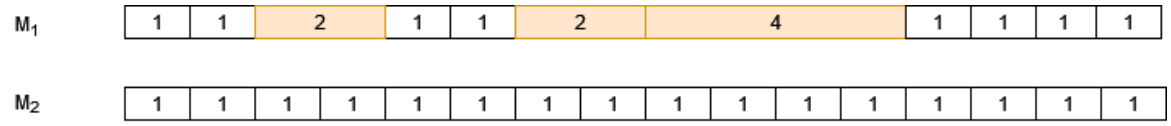

Figure 5.1: A job with $r_j = 0, p_1 = 2, p_2 = 4$ can benefit by bidding $p'_1 = 4, p'_2 = 4$

Now we shall prove the following theorem:

Theorem 5.1.1. *Assuming unit weights, for every job j, it holds that $c_j \leq O(\log P_{max})c^*_j$*

Let us denote by A_{α_j} the sequence during which job j arrived, and $A_{\alpha_j+\rho_j}$ the sequence during which job j was completed:

$$b(A_{\alpha_j}) \leq r_j < e(A_{\alpha_j}) \qquad b(A_{\alpha_j+\rho_j}) \leq r_j < e(A_{\alpha_j+\rho_j})$$

For the sake of simplicity, we will use the shorthand notation:

$$A_0 = A_{\alpha_j}, A_1 = A_{\alpha_j+1}, ..., A_{\rho_j} = A_{\alpha_j+\rho_j}$$

We shall now prove the following lemma:

Lemma 5.1.1. *Assume $\rho_j > 0, p_j = 2^k$. Let $A_i = S_d(t)$ for some $i < \rho_j$ and some d, t. If $r_j \leq b(A_i)$ then $vol(D(j) \cap \{j' | I(j') \in A_i\}) \geq 2^d$.*

Proof. The proof is separated into two cases depending on d:

- If $d \geq k$, then for job j to chose an interval in A_{ρ_j}, all intervals of size 2^k in A_i should be taken by other jobs, otherwise job j would have chosen one of them. Applying Lemma (4.3.2) directly leads to $vol(D(j) \cap \{j' | I(j') \in A_i\}) \geq 2^d$.
- If $d < k$, consider the minimal index j' with $c_{j'} > e(A_i)$. Due to the structure of the algorithm, j' chose an interval in A_{i+1}, which is disjoint from A_i, and appears no later than A_{ρ_j}. It must also be that $r_{j'} \leq r_j$ (otherwise j' is not the minimal) and that $p_{j'} = 2^z \leq 2^d$ (or else the sequences would not be disjoint). Therefore, job j' could fit in any 2^d sized interval, which means every 2^d sized interval in A_i was occupied. Once again, applying Lemma (4.3.2) leads to $vol(D(j') \cap \{j'' | I(j'') \in A_i\}) \geq 2^d$, which then leads to $vol(D(j) \cap \{j' | I(j') \in A_i\}) \geq 2^d$, since $r_{j'} \leq r_j$.

□

The above lemma gives us a lower bound for the total volume occupied by jobs between the release time and completion time of job j. The following lemma will cover the case when job j arrives in the middle of A_0, but is scheduled in a later sequence.

Lemma 5.1.2. *Assume $\rho_j > 0, p_j = 2^k, A_0 = S_d(t)$. If $0 < d \leq k$ and $r_j \leq e(S_{d-1}(t))$ then $vol(D(j) \cap \{j' | I(j') \in A_0\}) \geq 2^{d-1}$.*

Proof. Since A_0 is disjoint from A_{ρ_j}, consider the minimal index j' such that $c_{j'} > e(A_0)$. Once again, $r_{j'} \leq r_j$ and $p_{j'} = 2^z \leq 2^d$. Since the job fits in the 2^d sized last interval of A_0, but did not choose it, it must have been occupied. Using Lemma (4.3.2), gives us the desired $vol(D(j) \cap \{j' | I(j') \in A_i\}) \geq 2^{d-1}$. □

We shall finally prove two lemmata that, when combined, produce Theorem (5.1.1)

Lemma 5.1.3. *For every j such that $\rho_j = 0$, $c_j \leq O(\log P_{max})c^*_j$.*

Proof. $\rho_j = 0$ means that $b(A_0) \leq r_j < c_j \leq e(A_0)$. Let $A_0 = S_d(t)$ and as j fits in this sequence, $k \leq d$. We split the proof in the following cases:

1. If j occupies the first interval of size S_k in A_0, then by Corollary (4.3.1) $c_j \leq b(A_0)+(k+1)2^k$ and $c_j^* \geq b(A_0) + 2^k$, therefore:

$$c_j \leq (k+1)c_j^* \leq O(\log P_{max})c_j^*$$

2. Otherwise, $e(S_\alpha(t)) \leq r_j + p_j < e(S_{\alpha+1}(t))$ for some α. Again we distinguish the following cases:

   (a) If $c_j \leq e(S_{\alpha+2}(t))$, then by Corollary (4.3.1), $c_j \leq b(A_0) + (\alpha+3)2^{\alpha+2}$ while $c_j^* \geq b(A_0) + (\alpha+1)2^\alpha$. Thus:

$$c_j \leq \frac{4(\alpha+3)}{\alpha+1}c_j^* \leq 12c_j^*$$

   (b) Otherwise there exists a x such that $e(S_{\alpha+x}(t)) < c_j \leq e(S_{\alpha+x+1}(t))$. This implies that every interval of size 2^k in the set $S_{\alpha+x}(t) \setminus S_{\alpha+1}$ must have been occupied, or else job j would have chosen it. Once again using Corollary (4.3.1), we can deduce that S_k appears $2^{\alpha+x-k}$ times in $S_{\alpha+x}(t)$ and $2^{\alpha+1-k}$ times in $S_{\alpha+1}$. Therefore S_k appears $2^{\alpha+x-k} - 2^{\alpha+1-k} \geq 2^{\alpha+x-k-1}$ times in $S_{\alpha+x} \setminus S_{\alpha+1}$.

   Using Lemma (4.3.2) that bounds the volume of jobs completed before j, we can deduce that $c_j^* \geq \max\left\{b(A_0), 2^{\alpha+x+1}\right\}$. By Corollary (4.3.1) it holds that:

$$c_j \leq e(S_{\alpha+x+1}(t)) = b(A_0)+(\alpha+x+2)2^{\alpha+x+1} \leq \max\left\{b(A_0), 2^{\alpha+x+1}\right\}(5+4(\alpha+x+1))$$

   .

   Also since $S_{\alpha+x+1}$ is fully contained within $S_d(t)$, $P_{max} > \alpha+x+1$, which means that:

$$c_j \leq (5 + 4\log P_{max})c_j^*$$

□

The above lemma fully covers the case where job j is scheduled in the same sequence in which it arrives. The following lemma will cover all of the other cases:

Lemma 5.1.4. *If $\rho_j \geq 1$, then for every j, $c_j \leq O(\log P_{max})c_j^*$.*

Proof. We will prove the above lemma by deriving a lower bound for the volume of jobs in every A_i between A_0 and A_{ρ_j}. Let D_i be the set of jobs in $D(j)$ that are completed in some A_i (obviously $\bigcup_{0 \leq i \leq \rho_j} D_i \subseteq D(j)$). We can deduce that:

$$c_j^* \geq \max\left\{r_j + p_j, \sum_0^{\rho_j} vol(D_i))\right\}$$

We distinguish the following cases for A_i:

1. $i = 0$ when $r_j > b(A_0)$, which means that $r_j \geq (\alpha+1)2^\alpha$ for some α.

   (a) If $d = \alpha + 1$ then $e(A_0) = b(A_0) + (\alpha+2)2^{\alpha+1} \leq 4r_j$

   (b) If $d \geq \alpha + 2$ then $e(A_0) = b(A_0) + (d+1)2^d \leq b(A_0) + 2(\log P_{max} + 1)vol(D_0)$ from Lemma (5.1.2).

In both cases:

$$e(A_0) - b(A_0) \leq \max\{4r_j, 2(\log P_{max} + 1)vol(D_0)\} \leq 4(\log P_{max} + 1)\max\{r_j, vol(D_0)\}$$

2. $i \in \{1, 2, ..., \rho_j - 1\}$ or $i = 0$ when $r_j = b(A_0)$

   In that case $r_j \leq b(A_i) < e(A_i) < c_j$. Let $A_i = S_d(t)$. As the sequence is disjoint from A_{ρ_j}, $P_{max} \geq d$ and in conjuncture with (5.1.1):

$$e(A_i) - b(A_i) = (d+1)2^d \leq (\log P_{max} + 1)vol(D_i)$$

3. $i = \rho_j$

   Let $A_{\rho_j} = S_d(t)$. If j chooses the first 2^k sized interval in A_{ρ_j} then $c_j = b(A_{\rho_j}) + (k+1)2^k \leq A_{\rho_j} + (\log P_{max} + 1)p_j$. Otherwise let α be the maximal integer such that $e(S_\alpha(t)) < c_j \leq e(S_{\alpha+1}(t)$. Once again, every 2^k sized interval that comes before then must be occupied, and by Lemma (4.3.2) and Corollary (4.3.1) $c_j \leq b(A_{\rho_j}) + (\alpha+2)2^{\alpha+1} \leq b(A_{\rho_j}) + (\log P_{max} + 1)vol(D_{\rho_j})$. In both cases:

$$c_j - b(A_{\rho_j}) \leq (\log P_{max} + 1)\max\left\{p_j, vol(D_{\rho_j})\right\}$$

Notice that we can assume that in the worst case, there are no gaps between intervals in the schedule, as the idle time caused by the gaps would exist in the optimal solution as well, and therefore would be beneficial to our algorithm's competitive ratio. Thus, we assume $e(A_i) = b(A_{i+1})$ and finally it holds that:

$$\begin{aligned}
c_j &= e(A_0) - b(A_0) + \sum_{i=1}^{\rho_j - 1} e(A_i) - b(A_i) + c_j - b(A_{\rho_j}) \\
&\leq r_j + 4(\log P_{max} + 1)\max\{r_j, vol(D_0)\} + \sum_{i=1}^{\rho_j - 1}(\log P_{max} + 1)vol(D_i) + (\log P_{max} + 1)\max\left\{p_j, vol(D_{\rho_j})\right\} \\
&= r_j + (\log P_{max} + 1)\left(4\max\{r_j, vol(D_0)\} + \sum_{i=1}^{\rho_j - 1} vol(D_i) + \max\left\{p_j, vol(D_{\rho_j})\right\}\right) \\
&\leq c_j^* + (\log P_{max} + 1)(4c_j^* + c_j^* + c_j^*) \\
&= O(\log P_{max})c_j^*
\end{aligned}$$

□

Theorem (5.1.1) follows directly from Lemmata (5.1.3) and (5.1.4)

Corollary 5.1.1. *Given the assignments of an α-competitive scheduling algorithm minimizing the sum of completion times, we can obtain a $O(\alpha \cdot \log P_{max})$-competitive prompt algorithm.*

The above corollary implies that any constant-factor competitive online truthful scheduling algorithm for unrelated machines will inherently provide us with an optimal prompt menu-based algorithm as well.

5.1.2 Arbitrary weights

To cover the case of arbitrary weights, we can simply propose an extension. Given a maximum weight W_{max} (note that the algorithm needs to know it in advance this time), create $logW_{max}$ copies of every interval that appears. The i-th copy (zero-indexed) is designed to hold jobs of weight

$w_j \geq 2^i$. Since we do not care about truthfulness, the analysis in the above section holds when multiplying everything with $logW_{max}$, providing a $O(\log W_{max} \cdot (\log P_{max} + \log n)$-competitive online scheduling algorithm with the objective of minimizing the sum of weighted completion times. Formally, the following theorem holds.

Theorem 5.1.2. *For every job j, it holds that $c_j \leq O((\log n + \log P_{max}) \cdot \log W_{max})c_j^*$*

Chapter 6

A Truthful Algorithm and Experimental Results

In the final section of this thesis, we will present a truthful prompt scheduling algorithm for unrelated machines with the objective of minimizing the sum of weighted completion times, whose competitive ratio we have not managed to analyze fully. Then, we shall compare this algorithm to the state-of-the-art online algorithm of Gupta et al. [23] and to the lower bound given by the LP of [24], by solving some instances found in literature, as well as some we created on our own.

6.1 The truthful algorithm

6.1.1 Unit weights

We propose a truthful menu-based algorithm for scheduling unrelated machines, which we consider a natural extension of the algorithm for parallel machines proposed by Eden et al. [15]. The algorithm maintains a dynamic sequence of timeslots for each of the unrelated machines, and offers a menu of options to each incoming job. Each job is left to choose an interval on its own, and the punishment of never finishing its execution if it chooses an interval smaller than its processing time remains, making the algorithm trivially truthful. The algorithm is the exact same as 9, the only difference being that for job j, each machine i updates its sequence individually based on p_{ij}:

Mechanism 11: Dynamic menu-offering algorithm, unrelated machines, unit weights

Given a job j with processing times p_{ij} and release time r_j, and a state $A_{l_{j-1}}^{j-1}$, construct $\tau(A_{l_{j-1}}^{j-1}, r_j, j)$ and do the following:

$a \leftarrow 1$;
while *job has not picked an interval* **do**
    Offer the first interval of size 2^a on every machine;
    $a \leftarrow a * 2$;
end

While we have not been able to fully analyze the above algorithm's competitive ratio, our survey as well as the experimental results below lead us to believe that its performance may be bounded by $O(\log P_{max})$ (making it an optimal prompt algorithm), with P_{max} being the maximal realized processing time of a job.

6.1.2 Arbitrary weights

As is the case in the parallel machine setting, there exists a natural extension of the above algorithm for handling arbitrary weights in unrelated machines. Once again, we simply add $\log W_{max}$ copies of each timeslot to the sequence, with the $i-$th copy having been designed to allow jobs of weight $\geq 2^i$ to occupy it. The analysis is the exact same as the parallel machines setting, as the jobs have the same weight for all machines.

6.1.3 Main Remarks

While we were unable to fully analyze the competitive ratio of algorithm 11, our attempts at a proof have given us valuable insight on some of its key parameters. As mentioned in section 5.1, as long as our algorithm follows the assignments of algorithm 7, its performance is optimal given the lower bound of $\Omega(\log P_{max})$ of prompt algorithms. Therefore, it makes sense to understand the patterns and circumstances under which our algorithm differentiates itself from the assignments done in algorithm 7. Our first goal was to try to quantify these differences and measure the "damage" that each of these could cause as more jobs arrived. Note that since each job minimizes its own completion time, at the time of a discrepancy, our algorithm is optimizing the current sum of completion times; therefore the implications of such a discrepancy can only be measured by the damage caused to the completion time of future jobs. Furthermore, since the sequence's design intuitively means that each job is delayed by $\log P_{max}$ (and therefore the expected competitive ratio would be $O(\log P_{max})$, our main goal was to give an upper bound to the value of $P_{max,ALG}$ as opposed to P^*_{max}, as the optimal algorithm has different assignments to our algorithm, and we are scheduling in unrelated machines. When testing how much larger than P^*_{max}, $P_{max,ALG}$ can become, we had the following indications:

1. In all our attempts at creating a "bad" instance, $P_{max,ALG}$ was never larger than $(P^*_{max})^2$
2. Even in the case where $P_{max,ALG}$ reached a size of $O((P^*_{max})^2)$, the very long setup of such an instance made it so our algorithm is practically identical to the optimal algorithm. More specifically, these instances had a very long line of same assignments between the 2 algorithms, followed by a discrepancy that was not enough to damage our algorithm's performance.
3. Due to the dynamic nature of the sequence, any "bad" assignment that leads us to an extended large sequence is quickly forgotten; the resetting of each sequence to $\overline{S}_0$ whenever no bigger jobs arrive (and in general, the fact that the sequence only extends itself whenever it needs to), means we never pay more than one factor of $\log P_{max}$ when a very large job is scheduled.

All of the above, have led us to the assumption that algorithm 11 is actually a tight $O(\log P_{max})$ competitive scheduling algorithm. The following section aims to provide an experimental evaluation to further support the above claim.

6.2 Experimental Results

In literature, the works conducting an experimental evaluation of scheduling algorithms in the setting of unrelated machines with the objective of minimizing the sum of weighted completion times are scarce [53, 2, 54]. In order to get a better understanding of the algorithms presented in this thesis in practice (and not only in their worst case, which may consist of a very specific, unnatural instance), we conducted experiments based on known instances in literature as well as instances that we created specifically for this comparison. The main results of the experiments are the following:

1. Algorithm 7 has a much closer to optimal performance than its theoretical upper bound of 7.216, mainly hovering between 4, when all jobs are available at time 0, and 1.5 to 2, when jobs arrive over time.
2. The maximal realized processing time in the optimal solution P^*_{max} is always comparable (with a ratio of around 1) in size to the maximal realized processing time in 11 $P_{max,ALG}$. That means that the $\log P_{max}$ bound is valuable (which would not be the case if $P_{max,ALG}$ was unbounded).
3. Algorithm 11 has a good enough performance to support the conjecture that in the average case, its competitive ratio is $\Theta(\log P_{max})$

6.2.1 Input files

The inputs used to measure the performance of the compared algorithms can be divided into two groups. The first group consists of instances downloaded from the library PSPLIB, a library of instances for the Resource Constrained Project Scheduling problem [32, 33, 13]. We downloaded 100 instances of the multi-mode (resembling unrelated machines) resource-constrained scheduling problem for a varying number of jobs n and machines m ($n = \{30, 50, 100\}, m = \{3, 6, 9\}$) and erased all constraints except from the precedence constraints. Note that the maximum processing time of a job in these instances is 50, which tunes the parameter P_{max} of algorithm 11 to 64. The second group of instances are the ones we created by sampling processing times (differing for each machine) randomly and uniformly from the interval $[0, 100]$ (which tunes the P_{max} parameter of algorithm 11 to 128), sampling weights randomly and uniformly from the interval $[0, 10]$ and sampling arrival times simulating a Poisson stable queue with $\lambda = 0.2$. Once again there are instances for a varying number of jobs ($n = \{10, 20, 50, 100, 500\}$ and machines ($m = \{2, 5, 7, 10\}$). The algorithm of Gupta et al. [23] and algorithm 11 based on Eden et al. [15] were implemented in *Python 3.6* with the needed modifications to support precedence constraints where applicable. We note that the language of implementation does not matter, as the algorithms are deterministic and our experiments only measure their competitive ratio and not their running time. The LP from [24] solved to obtain the lower bound was solved using Google's OR Tools [40].

6.2.2 Results

Comparing algorithm 7 to algorithm 11

In principle, these two algorithms follow the same logic. Whenever a job j with processing times p_{ij} arrive, they delay the job by an adequate amount of time in order to fit smaller jobs or jobs of higher priority before it. Algorithm 7 delays the job by a maximum of $\frac{2}{3}p_{ij}$ for each machine, whereas algorithm 11 delays it by $\log P_{max}$ due to the nature of the prompt sequence. Below, we present four figures showcasing the competitive ratio of algorithm 11 compared to algorithm 7 (i.e. $\frac{cost(ALG_6)}{cost(ALG_2)}$) for the instances that we created (Group 2).

It is evident that as more jobs and machines are included in the instances, the closer the performance of the two algorithms becomes. Also, as was expected, adding release times to the problem brings the two algorithms closer, as the job's arrival times become more and more spaced out.

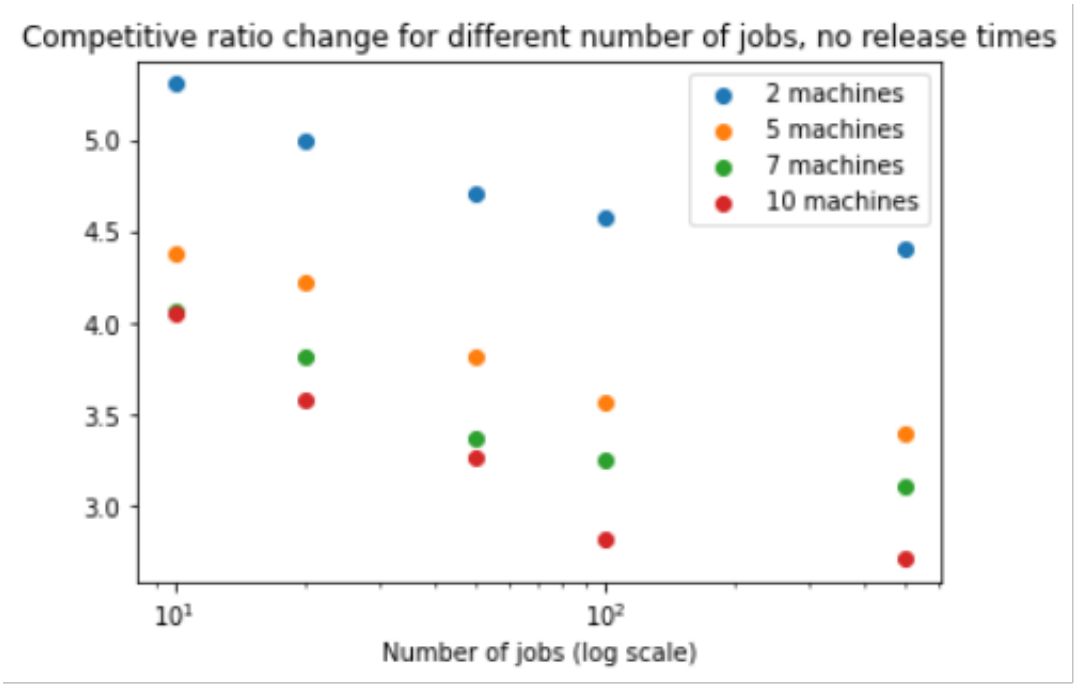

Figure 6.1: Competitive ratio wrt number of jobs, no release times

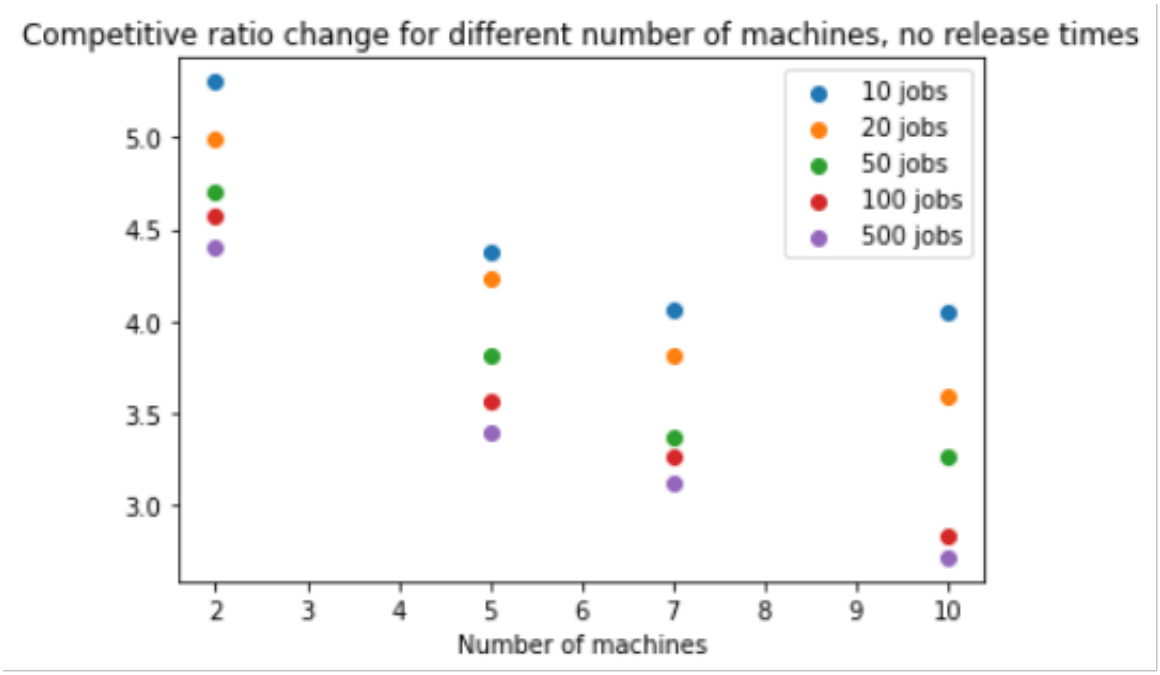

Figure 6.2: Competitive ratio wrt number of machines, no release times

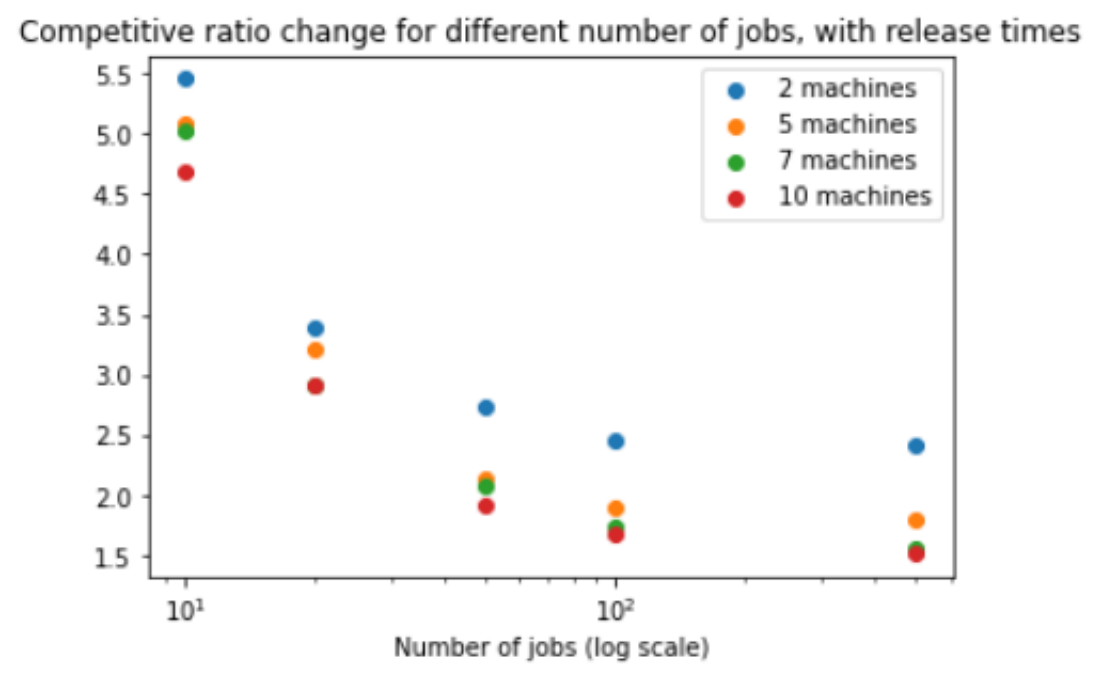

Figure 6.3: Competitive ratio wrt number of jobs, with release times

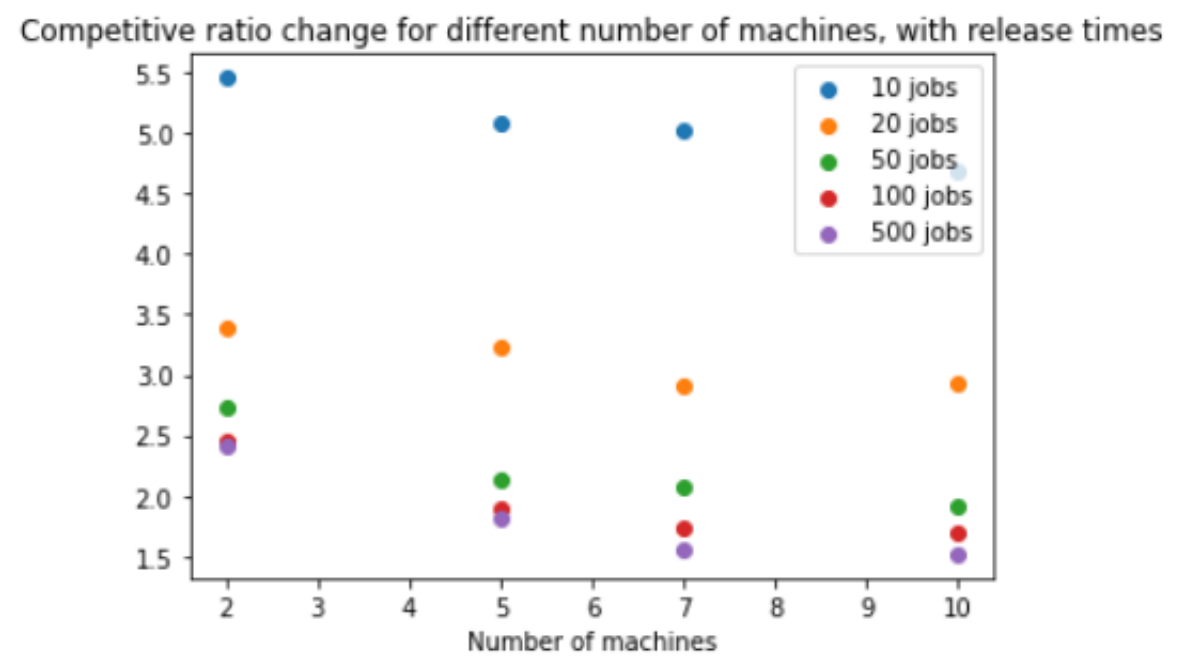

Figure 6.4: Competitive ratio wrt number of jobs, with release times

Comparing the algorithms to a lower bound obtained by solving LP 3.3.1, we can observe the following results:

- When jobs have release times simulating a poisson stable queue, ALG 7 is extremely close to being optimal. Practically, the job delaying it introduces only causes some minor damage at the start, and as times passes the algorithm simulates the optimal solution. ALG 11 also achieves a very good performance, better than the asymptotical factor $O(\log P_{max})$.
- When jobs have no release times, meaning they are all available to be scheduled from time

0 and just arrive online one after the other, ALG 7 is still better than its theoretical upper bound, but loses some performance due to the mandatory job delaying it introduces. At the same time, ALG 11 has a considerably worse performance, but it still remains within a logical constant margin of the asymptotical factor $O(\log P_{max})$. This is important, as in the case of release times being 0, promptness is a much more desired trait since the jobs can wait for a long time before their execution is initiated.

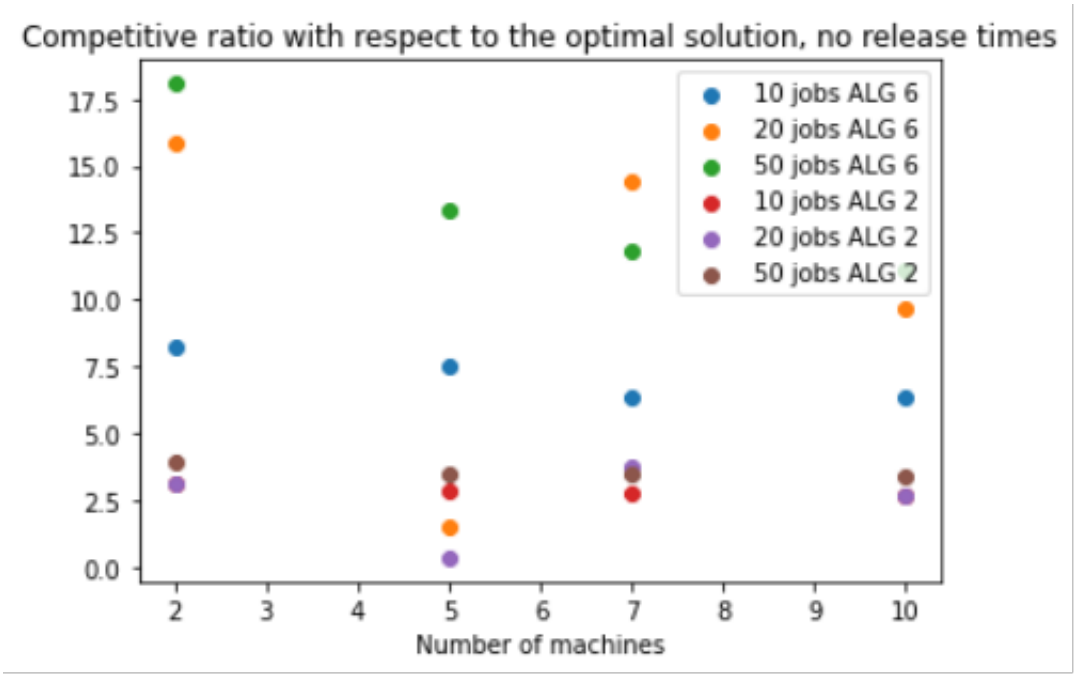

Figure 6.5: Competitive ratio compared to optimal, no release times

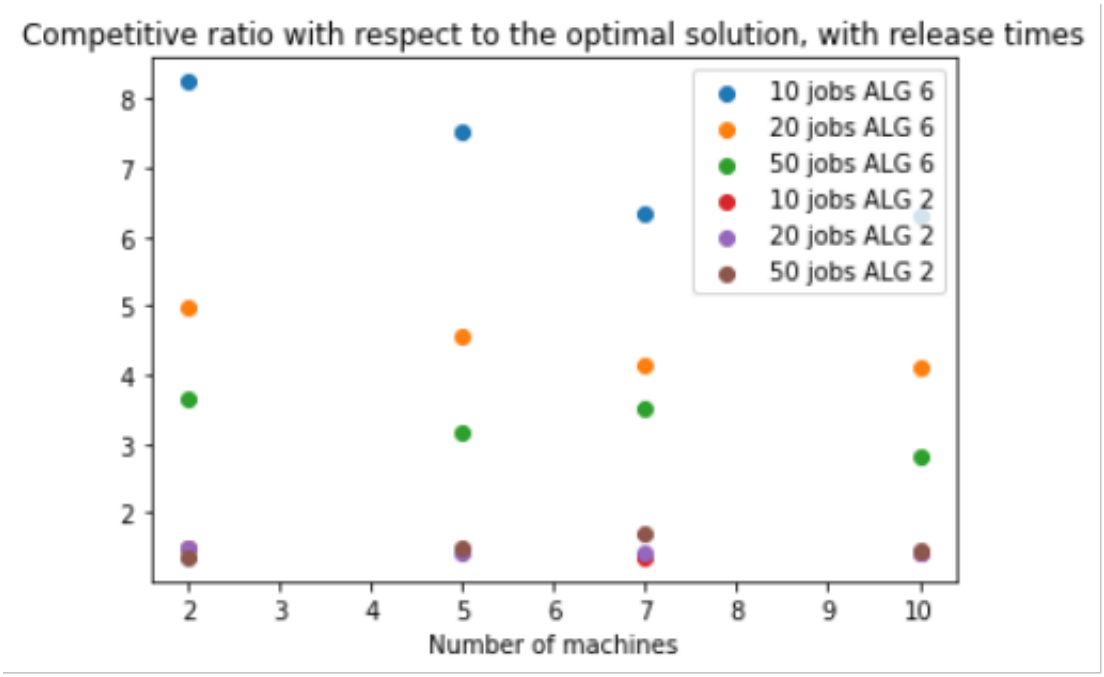

Figure 6.6: Competitive ratio compared to optimal, with release times

We also present the results comparing ALG 11 to ALG 7 obtained from the experiments con-

ducted on Group 1 of the instances, noting that precedence constraints (as expected) also bring the algorithms closer in performance, as in a lot of cases they limit the damage caused by a larger than needed delay.

Competitive Ratio	min	median	max
$n = 30, m = 3$	1.62	1.92	2.27
$n = 50, m = 3$	1.47	1.96	2.15
$n = 50, m = 6$	1.47	1.87	2.51
$n = 50, m = 9$	1.51	1.87	2.65
$n = 100, m = 3$	2.23	2.67	3.13
$n = 100, m = 6$	1.91	2.25	2.49
$n = 100, m = 9$	1.38	2.02	2.46

Bibliography

[1] E. Angel, E. Bampis, F. Pascual, and N. Thibault, "Truthfulness for the sum of weighted completion times," in *Computing and Combinatorics*, T. N. Dinh and M. T. Thai, Eds. Cham: Springer International Publishing, 2016, pp. 15–26.

[2] I. D. Baev, W. M. Meleis, and A. Eichenberger, "An experimental study of algorithms for weighted completion time scheduling," *Algorithmica*, vol. 33, no. 1, pp. 34–51, May 2002. [Online]. Available: https://doi.org/10.1007/s00453-001-0103-x

[3] N. Bansal, A. Srinivasan, and O. Svensson, "Lift-and-round to improve weighted completion time on unrelated machines," in *STOC 2016 - Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*. United States: Association for Computing Machinery, Inc, Jun. 2016, pp. 156–167, 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2016) ; Conference date: 19-06-2016 Through 21-06-2016.

[4] H. Belouadah, M. Posner, and C. Potts, "Scheduling with release dates on a single machine to minimize total weighted completion time," *Discrete Applied Mathematics*, vol. 36, no. 3, pp. 213–231, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0166218X92902559

[5] J. Bruno, E. G. Coffman, and R. Sethi, "Scheduling independent tasks to reduce mean finishing time," *Commun. ACM*, vol. 17, no. 7, p. 382–387, jul 1974. [Online]. Available: https://doi.org/10.1145/361011.361064

[6] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, "Approximation techniques for average completion time scheduling," *SIAM Journal on Computing*, vol. 31, no. 1, pp. 146–166, 2001. [Online]. Available: https://doi.org/10.1137/S0097539797327180

[7] G. Christodoulou, E. Koutsoupias, and A. Kovacs, "On the nisan-ronen conjecture," 2021.

[8] ——, "A proof of the nisan-ronen conjecture," 2023.

[9] G. Christodoulou, E. Koutsoupias, and A. Vidali, "A lower bound for scheduling mechanisms," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. USA: Society for Industrial and Applied Mathematics, 2007, p. 1163–1170.

[10] E. H. Clarke, "Multipart pricing of public goods," *Public Choice*, vol. 11, no. 1, pp. 17–33, Sep 1971. [Online]. Available: https://doi.org/10.1007/BF01726210

[11] J. R. Correa and M. R. Wagner, "Lp-based online scheduling: From single to parallel machines," in *Integer Programming and Combinatorial Optimization*, M. Jünger and V. Kaibel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 196–209.

[12] S. Dobzinski and A. Shaulker, "Improved lower bounds for truthful scheduling," 2020.

[13] A. Drexl, R. Nissen, J. H. Patterson, and F. Salewski, "Progen/πx – an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions," *European Journal of Operational Research*, vol. 125, no. 1, pp. 59–72, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221799002052

[14] M. E. Dyer and L. A. Wolsey, "Formulating the single machine sequencing problem with release dates as a mixed integer program," *Discrete Appl. Math.*, vol. 26, no. 2–3, p. 255–270, feb 1990. [Online]. Available: https://doi.org/10.1016/0166-218X(90)90104-K

[15] A. Eden, M. Feldman, A. Fiat, and T. Taub, "Prompt scheduling for selfish agents," 2018.

[16] D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. Uma, and J. Wein, "Techniques for scheduling with rejection," *Journal of Algorithms*, vol. 49, no. 1, pp. 175–191, 2003, 1998 European Symposium on Algorithms. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196677403000786

[17] M. Feldman, A. Fiat, and A. Roytman, "Makespan minimization via posted prices," 2017.

[18] Y. Giannakopoulos, A. Hammerl, and D. Poças, "A new lower bound for deterministic truthful scheduling," 2020.

[19] V. Gkatzelis, E. Markakis, and T. Roughgarden, "Deferred-acceptance auctions for multiple levels of service," in *Proceedings of the 2017 ACM Conference on Economics and Computation*, ser. EC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 21–38. [Online]. Available: https://doi.org/10.1145/3033274.3085142

[20] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang, "Single machine scheduling with release dates," *SIAM Journal on Discrete Mathematics*, vol. 15, no. 2, pp. 165–192, 2002. [Online]. Available: https://doi.org/10.1137/S089548019936223X

[21] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Discrete Optimization II*, ser. Annals of Discrete Mathematics, P. Hammer, E. Johnson, and B. Korte, Eds. Elsevier, 1979, vol. 5, pp. 287–326. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016750600870356X

[22] T. Groves, "Incentives in teams," *Econometrica*, vol. 41, no. 4, pp. 617–631, 1973. [Online]. Available: http://www.jstor.org/stable/1914085

[23] V. Gupta, B. Moseley, M. Uetz, and Q. Xie, "Greed works—online algorithms for unrelated machine stochastic scheduling," *Mathematics of Operations Research*, vol. 45, no. 2, pp. 497–516, may 2020. [Online]. Available: https://doi.org/10.1287%2Fmoor.2019.0999

[24] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Mathematics of Operations Research*, vol. 22, no. 3, pp. 513–544, 1997. [Online]. Available: http://www.jstor.org/stable/3690391

[25] H. Hoogeveen, M. Skutella, and G. J. Woeginger, "Preemptive scheduling with rejection," *Mathematical Programming*, vol. 94, no. 2, pp. 361–374, Jan 2003. [Online]. Available: https://doi.org/10.1007/s10107-002-0324-z

[26] J. A. Hoogeveen and A. P. A. Vestjens, "Optimal on-line algorithms for single-machine scheduling," in *Integer Programming and Combinatorial Optimization*, W. H. Cunningham, S. T. McCormick, and M. Queyranne, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 404–414.

[27] J. Hoogeveen, P. Schuurman, and G. Woeginger, "Non-approximability results for scheduling problems with minsum criteria," in *Integer Programming and Combinatorial Optimization (Proceedings 6th International IPCO Conference, Houston TX, USA, June 22-24, 1998)*, ser. Lecture Notes in Computer Science, R. Bixby, E. Boyd, and R. Rios-Mercado, Eds. Germany: Springer, 1998, pp. 353–366.

[28] L. Hurwicz, *Optimality and informational efficiency in resource allocation processes*. Cambridge University Press, 1977, p. 393–460.

[29] S. Im and S. Li, "Improved approximations for unrelated machine scheduling," 2022.

[30] ——, "Better unrelated machine scheduling for weighted completion time via random offsets from non-uniform distributions," *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 138–147, 2016.

[31] T. Kawaguchi and S. Kyan, "Worst case bound of an lrf schedule for the mean weighted flow-time problem," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1119–1129, 1986. [Online]. Available: https://doi.org/10.1137/0215081

[32] R. Kolisch, C. Schwindt, and A. Sprecher, "Benchmark instances for project scheduling problems," 1999.

[33] R. Kolisch and A. Sprecher, "Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program," *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221796001701

[34] E. Koutsoupias and A. Vidali, "A lower bound of $1 + \phi$ for truthful scheduling mechanisms," in *Mathematical Foundations of Computer Science 2007*, L. Kučera and A. Kučera, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 454–464.

[35] J. Lenstra, A. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[36] S. Li, "Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations," 2017.

[37] R. McNaughton, "Scheduling with Deadlines and Loss Functions," *Management Science*, vol. 6, no. 1, pp. 1–12, October 1959. [Online]. Available: https://ideas.repec.org/a/inm/ormnsc/v6y1959i1p1-12.html

[38] N. Megow and A. S. Schulz, "On-line scheduling to minimize average completion time revisited," *Operations Research Letters*, vol. 32, no. 5, pp. 485–490, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167637703001573

[39] N. Nisan and A. Ronen, "Algorithmic mechanism design," *Games and Economic Behavior*, vol. 35, no. 1, pp. 166–196, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S089982569990790X

[40] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/

[41] C. A. Phillips, C. Stein, and J. Wein, "Minimizing average completion time in the presence of release dates," *Mathematical Programming*, vol. 82, pp. 199–223, 1998.

[42] M. Queyranne, "Structure of a simple scheduling polyhedron," *Math. Program.*, vol. 58, no. 1–3, p. 263–285, jan 1993.

[43] M. Queyranne and A. S. Schulz, "Polyhedral approaches to machine scheduling," 2008.

[44] A. S. Schulz and M. Skutella, "Scheduling unrelated machines by randomized rounding," *SIAM Journal on Discrete Mathematics*, vol. 15, no. 4, pp. 450–469, 2002. [Online]. Available: https://doi.org/10.1137/S0895480199357078

[45] S. S. Seiden, "Preemptive multiprocessor scheduling with rejection," *Theoretical Computer Science*, vol. 262, no. 1, pp. 437–458, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397500002887

[46] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, no. 1, pp. 461–474, Feb 1993. [Online]. Available: https://doi.org/10.1007/BF01585178

[47] R. Sitters, "Efficient algorithms for average completion time scheduling," in *Integer Programming and Combinatorial Optimization*, F. Eisenbrand and F. B. Shepherd, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 411–423.

[48] M. Skutella, "Convex quadratic and semidefinite programming relaxations in scheduling," *J. ACM*, vol. 48, no. 2, p. 206–242, mar 2001. [Online]. Available: https://doi.org/10.1145/375827.375840

[49] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, vol. 3, no. 1-2, pp. 59–66, March 1956. [Online]. Available: https://ideas.repec.org/a/wly/navlog/v3y1956i1-2p59-66.html

[50] M. Sviridenko and A. Wiese, "Approximating the configuration-lp for minimizing weighted sum of completion times on unrelated machines," in *Integer Programming and Combinatorial Optimization*, M. Goemans and J. Correa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 387–398.

[51] A. Vestjens, "On-line machine scheduling," Ph.D. dissertation, Mathematics and Computer Science, 1997.

[52] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *The Journal of Finance*, vol. 16, no. 1, pp. 8–37, 1961. [Online]. Available: http://www.jstor.org/stable/2977633

[53] T. Vredeveld and C. Hurkens, "Experimental comparison of approximation algorithms for scheduling unrelated parallel machines," *INFORMS Journal on Computing*, vol. 14, no. 2, pp. 175–189, 2002. [Online]. Available: https://doi.org/10.1287/ijoc.14.2.175.119

[54] E. Xavier and F. Miyazawa, "Practical comparison of approximation algorithms for scheduling problems," *Pesquisa Operacional*, vol. 24, 05 2003.