Εθνικο Μετσοβιο Πολυτεχνειο
Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Λογικης και Επιστημης Υπολογισμων

Δεσμευτικοί Φιλαλήθεις Μηχανισμοί για Ελαχιστοποίηση του Βεβαρημένου Χρόνου Ολοκλήρωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αλκιβιάδης Γ. Μέρτζιος

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

Αθήνα, Νοέμβριος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΛΟΓΙΚΗΣ ΚΑΙ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΜΩΝ

Δεσμευτικοί Φιλαλήθεις Μηχανισμοί για Ελαχιστοποίηση του Βεβαρημένου Χρόνου Ολοκλήρωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αλκιβιάδης Γ. Μέρτζιος

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Νοεμβρίου 2021.

.....	.....	.....
Δημήτριος Φωτάκης	Αριστείδης Παγουρτζής	Ευάγγελος Μαρκάκης
Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ο.Π.Α.

Αθήνα, Νοέμβριος 2021.

.....

Αλκιβιάδης Γ. Μέρτζιος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σε αυτή την εργασία, ορίζουμε φορμαλιστικά την έννοια της Δεσμευτικότητας, την οποία εμπνευστήκαμε από πρόσφατες δουλειές σε Άμεση Χρονοδρομολόγηση σε Παράλληλες Μηχανές. Από όσο ξέρουμε, είμαστε οι πρώτοι που χαρακτηρίζουν με ακρίβεια αυτή την έννοια. Δεσμευτικότητα είναι η υπόσχεση ενός άμεσου αλγόριθμου δρομολόγησης να προσδιορίσει τον χρόνο ολοκλήρωσης μιας δουλειάς κατά την άφιξή της. Η δεσμευτικότητα είναι τόσο ισχυρή ιδιότητα που απαιτεί ένα πλήρως προκαθορισμένο πρόγραμμα για την επίτευξή της.

Εισάγουμε ένα μοντέλο για το πρόβλημα της Χρονοδρομολόγησης σε Παράλληλες Μηχανές το οποίο υποθέτει μία πολυωνυμική κατανομή μηκών και είναι κοντά στο άμεσο μοντέλο. Για αυτό το μοντέλο, περιγράφουμε έναν ασυμπτωτικά βέλτιστο αλγόριθμο. Περιορίζοντας το μοντέλο ώστε να έχει έναν σταθερό αριθμό από δουλειές κάθε δευτερόλεπτο και υποθέτοντας αριθμό μηχανών τουλάχιστον τόσες όσο ο αριθμός των δουλειών επί το μέσο μήκος, αποδεικνύουμε πλήρως δεσμευτικούς αλγορίθμους. Αυτοί οι αλγόριθμοι είναι ξανά ασυμπτωτικά βέλτιστοι. Αυτό είναι βελτίωση σε σχέση με το γενικό κάτω φράγμα $\Omega(\log L_{max})$ για δεσμευτικούς αλγορίθμους, όπου L_{max} είναι το μέγιστο μήκος δουλειών. Επιπλέον, αναβαθμίζουμε αυτό το μοντέλο υποθέτοντας ότι ο αριθμός των δουλειών είναι τυχαία μεταβλητή. Αυτό χειροτερεύει την προσέγγιση κατά έναν προσθετικό παράγοντα ίσο με τη ρίζα της διακύμανσης διά τον μέσο αριθμό δουλειών. Ξανά, αλλάζοντας το μοντέλο και υποθέτοντας περιοδικότητα του αριθμού δουλειών, έχουμε ένα προσθετικό παράγοντα στη προσέγγιση $O(\frac{h}{T})$, όπου h η περίοδος και T ο αριθμός των χρονικών στιγμών. Ακόμη, προσθέτουμε φιλαλήθεια σε κλασικούς αλγορίθμους που μετατρέπουν προεκτοπιστικά προγράμματα σε μη προεκτοπιστικά. Τέλος, σχεδιάζουμε έναν αλγόριθμο που είναι μερικώς δεσμευτικός και έχει σταθερή προσέγγιση για αρκετά μεγάλη χαλάρωση.

Ύστερα, χρησιμοποιούμε τις ιδέες της δεσμευτικότητας/φιλαλήθειας στο Πρόβλημα του Πλανόδιου Επιδιορθωτή (Travelling Repairman Problem ή TRP). Δείχνουμε έναν μερικώς δεσμευτικό αλγόριθμο με σταθερή προσέγγιση που βασίζεται σε γνωστούς αλγορίθμους για το άμεσο TRP. Το κύριο αποτέλεσμά μας είναι ότι κάθε πλήρως δεσμευτικός αλγόριθμος για το άμεσο TRP έχει προσέγγιση τουλάχιστον $\Omega(\frac{TSP_{tour}}{\Delta})$, όπου TSP_{tour} το μήκος της βέλτιστης περιοδείας για το Πρόβλημα Πλανόδιου Πωλητή και Δ η διάμετρος του χώρου. Τέλος, σχεδιάζουμε έναν αλγόριθμο που πετυχαίνει αυτό το κάτω φράγμα για μετρικές σε δέντρα.

Λέξεις Κλειδιά: Δεσμευτικότητα, Άμεσοι Αλγόριθμοι, Άμεση Χρονοδρομολόγηση, Σχεδιασμός Μηχανισμών, Φιλαλήθεια, Χρονοδρομολόγηση σε Παράλληλες Μηχανές, Πρόβλημα Πλανόδιου Επιδιορθωτή

Abstract

In this thesis, we formally define the notion of Promptness, inspired by recent works in Online Parallel Machine Scheduling. To the best of our knowledge, we are the first to formalize this notion. Promptness is the promise of an online scheduling algorithm to specify a job's completion time upon its arrival. Promptness is such a strong property that requires a completely fixed schedule to attain it.

In our work, we define a model for the problem of Parallel Machine Scheduling that assumes a multinomial distribution of lengths and is close to the online setting. For this model, we provide asymptotically optimal algorithms that achieve constant competitiveness. By restricting the model to have a fixed number of jobs arriving each second and assuming an amount of machines greater than the number of jobs times the average length, we provide prompt algorithms that completely specify a job's completion on arrival. This algorithm is again asymptotically optimal. This is an improvement on the general lower bound for prompt algorithms, $\Omega(\log L_{max})$, where L_{max} is the largest job length. Furthermore, we augment the model by assuming the number of jobs is a random variable and prove a competitive ratio that exacerbates by an additive factor of the square root of the variation divided by the expected number of jobs each second. Again by altering the model slightly and assuming periodicity, we give a similar bound that increases by an additive factor $O(\frac{h}{T})$, where h is the length of the period and T the number of seconds. As such, our prompt algorithms remain asymptotically optimal. Moreover, we augment the classic algorithm of Phillips et al. that converts preemptive to non-preemptive schedules with truthfulness. Because of the fixed nature of promptness, it gives rise to truthfulness in an obvious manner. Thus, all prompt algorithms in this thesis are obviously truthful. In addition, we define a relaxation of promptness and allow the specification of the completion time to be not exact, but within a certain interval. We call these algorithms semi-prompt. We then give a semi-prompt algorithm, based on our own asymptotically optimal algorithm, that is constant competitive for an interval such that the ratio of the two end points is at least two.

We also apply these ideas of promptness/truthfulness to the Travelling Repairman Problem (TRP). We provide a constant competitive semi-prompt algorithm for the TRP based on the the first constant competitiveness online algorithm for the TRP. Our main result is that any fully-prompt algorithm for the online TRP must have a competitive ratio of at least $\Omega(\frac{TSP_{tour}}{\Delta})$, where TSP_{tour} is the length of the optimal TSP path on the metric and Δ its diameter. Note that this result holds for general metrics. We then apply this idea by approximating general metric spaces with 2-HST's. Thus, we provide a fully-prompt algorithm for the TRP for general metrics that is only a $O(\log n)$ factor away from the optimal. This is based on our optimal fully-prompt TRP algorithm for tree metrics.

Keywords: Promptness, Online Algorithms, Online Scheduling, Mechanism Design, Truthfulness, Parallel Machine Scheduling, Travelling Repairman Problem

Ευχαριστίες

Σε αυτό το σημείο, θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που ήταν μέρος αυτής της περιόδου στη ζωή μου.

Αρχικά, ευχαριστώ τον κ. Φωτάκη, του οποίου η καθοδήγηση και η υποστήριξη κατέστησε δυνατή την ολοκλήρωση αυτής της διπλωματικής διατριβής. Τον ευχαριστώ για τον ενθουσιασμό που επέδειξε στη προσέγγιση του κάθε προβλήματος που συναντήσαμε στην ερευνητική μας πορεία, για την πλήρη κατεύθυνση και συμβουλή που μου προσέφερε, τόσο ακαδημαϊκά όσο και προσωπικά, και για όλες τις γνώσεις που μου μετέφερε σε καθένα από τα μαθήματα και σε κάθε μία από τις συναντήσεις μας.

Ακόμη, ευχαριστώ τον κ. Παγουρτζή για το θεωρητικό υπόβαθρο που μου παρείχε σε πολυάριθμα μαθήματα, για την εμπιστοσύνη που μου εναπόθεσε επιτρέποντας μου να εργαστώ ως βοηθός στα μαθήματα που επιβλέπει και για την πάντα προθυμία να με υποστηρίξει.

Επιπλέον, ευχαριστώ την οικογένειά μου, τον Πατέρα μου, τη Μητέρα μου και τον Αδερφό μου για την πίστη τους σε εμένα αλλά και για την έμπρακτη βοήθειά τους.

Τέλος, ευχαριστώ όλους τους φίλους που έκανα αυτά τα χρόνια για τις στιγμές που περάσαμε και για τις αναμνήσεις που δημιουργήσαμε.

Αλκιβιάδης

Contents

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

Σε αυτή την εργασία, ασχολούμαστε αποκλειστικά με προβλήματα δρομολόγησης και προσπαθούμε να βελτιώσουμε την πρακτικότητά τους μέσω μιας καινούργιας έννοιας που τη λέμε δεσμευτικότητα. Για να ορίσουμε την έννοια της δεσμευτικότητας θα πρέπει πρώτα να μιλήσουμε για προβλήματα δρομολόγησης αλλά μέχρι τώρα αρκεί να ξέρουμε ότι δεσμευτικότητα είναι η ιδιότητα των αλγορίθμων να ενημερώνουν τις δουλειές που δρομολογούν για το πότε περίπου θα τελειώσει η εκτέλεσή τους. Η λέξη "αποκλειστικά" στη πρωτη πρόταση είναι μάλλον παραπλανητική, καθώς η βιβλιογραφία ενός και μόνο προβληματος δρομολόγησης μπορεί να γεμίσει ολόκληρο βιβλίο. Παρόλο που τα προβλήματα δρομολόγησης μπορεί να διαφέρουν αρκετά στο μοντέλο τους, σε αλγορίθμους τους, στις εφαρμογές τους ή και στη φύση τους, έχουν ένα κοινό που μας επιτρέπει να περιγράφουμε τις λύσεις τους με απλό τρόπο. Η λύση σε ένα πρόβλημα δρομολόγησης μπορεί να περιγραφεί αποκλειστικά από τη σειρά με την οποία δρομολογούνται οι δουλειές. Για παράδειγμα, αν είχαμε μία μηχανή που φτιάχνει κουστούμια και δύο παραγελλίες, μία μεγάλη και μία μικρή, το καλύτερο που θα μπορούσαμε να κάνουμε είναι να ξεκινήσουμε το ράψιμο κατευθείαν. Αν θα θέλαμε να ελαχιστοποιήσουμε το άθροισμα του χρόνου ολοκλήρωσης θα δρομολογούσαμε πρώτα τη μικρότερη από της δύο δουλειές. Όπως εξηγήσαμε, η λύση καθορίζεται από το ποια παραγγελία θα ξεκινήσουμε πρώτα. Αυτό θα άλλαζε ελαφρώς αν είχαμε δύο μηχανές οι οποίες μπορούσαν να ράψουν κουστούμια, καθώς θα έπρεπε να πούμε και σε ποια μηχανή ράβεται το κάθε κουστούμι. Ένα άλλο πρόβλημα δρομολόγησης προκύπτει όταν τελειώσουμε τη παρασκευή των κουστουμιών και πρέπει να τα παραδώσουμε στους ιδιοκτήτες τους. Ύστερα, γεννιούνται διάφορα ερωτήματα. Ποιο κουστούμι να παραδώσουμε πρώτα; Έχουμε δύο υπαλλήλους που μπορούν να κάνουν μεταφορές ή μόνον έναν; Τέτοιες λεπτομέρειες δημιουργούν πολυάριθμα προβλήματα δρομολόγησης που θα παρουσιάσουμε σε αυτό το κεφάλαιο.

Το πρώτο πρόβλημα είναι αυτό της δρομολόγησης δουλειών σε παράλληλες μηχανές. Στο πρόβλημα των παράλληλων μηχανών έχουμε δουλειές που καταφθάνουν στο σύστημά μας. Κάθε δουλειά j έχει ένα μήκος l_j που δηλώνει το χρόνο εκτέλεσής της, ίδιος για όλες τις μηχανές. Ο χρόνος που η δουλειά φτάνει στο σύστημα είναι η νωρίτερη στιγμή από την οποία μπορεί να ξεκινήσει η εκτέλεση και συμβολίζεται με r_j για τη δουλειά j. Το βάρος κάθε δουλειάς είναι το w_j και ορίζει τη σημαντικότητα της δουλειάς. Αν μία δουλειά έχει μεγάλο βάρος θα θέλουμε να τελειώσει πιο νωρίς. Ο χρόνος κατά τον οποίο η δουλειά τελειώνει την εκτέλεσή της ονομάζεται χρόνος ολοκλήρωσης και συμβολίζεται με c_j. Ο αριθμός των

μηχανών συμβολίζεται με m. Υποθέτουμε ότι οι μηχανές είναι όμοιες. Μπορούμε να έχουμε δύο μοντέλα εκτέλεσης. Στο προεκτοπιστικό μοντέλο (preemptive), μπορούμε να σταματήσουμε την εκτέλεση μιας δουλειάς και να την συνεχίσουμε αργότερα, πιθανώς σε άλλη μηχανή, χωρίς να χάσουμε πρόοδο. Αντίθετα, το μη-προεκτοπιστικό μοντέλο (non-preemptive) δε μας το επιτρέπει αυτο και μια εκτέλεση δε μπορεί να διακοπεί αφού ξεκινήσει.

Το άλλο πρόβλημα που εξετάζουμε σε αυτή την εργασία είναι αυτό που σχετίζεται με τη παράδοση των κουστουμιών. Συγκεκριμένα, το πρόβλημα ονομάζεται Πρόβλημα του Πλανόδιου Πωλητή (Travelling Repairman Problem ή TRP). Στο TRP έχουμε ένα μετρικό χώρο. Ο μετρικός χώρος είναι ένα γράφημα με βάρη στις ακμές που ικανοποιούν τη τριγωνική ανισότητα. Από ένα ειδικό σημείο του χώρου, την αρχή, ξεκινά τη χρονική στιγμή $t = 0$ ο εξυπηρετητής (server). Ο server κινείται με μονοδιαία ταχύτητα και στόχος του είναι να φτάσει στα σημεία αιτήσεων που παρουσιάζονται στο μετρικό χώρο. Για μία αίτηση j, συμβολίζουμε το σημείο του μετρικού χώρου στο οποίο παρουσιάζεται με j, την απόσταση του σημείου της από την αρχή με $d(o, j)$, το βάρος της αίτησης με w_j και το χρόνο άφιξής της με r_j. Σκοπός του server είναι να κινηθεί προς τα σημεία των αιτήσεων, όπου τις εξυπηρετεί κατευθείαν. Ο χρόνος αυτός είναι ο χρόνος ολοκλήρωσης της αίτησης και συμβολίζεται με c_j, κατά τα ίδια.

Μία παραλλαγή του προβληματος είναι κάθε αίτηση να είχε ένα σημείο άφιξης και έναν προορισμό, ενώ να θεωρείται ικανοποιημένη όταν ο server τη μεταφέρει από την άφιξή της στον προορισμό. Αυτό το πρόβλημα είναι το Κάλεσε μία Κούρσα (Dial A Ride Problem ή DARP) και είναι γενίκευση του TRP, αφού το TRP είναι το DARP όπου όλες οι δουλειές έχουν κοινό προορισμό και άφιξη. Μία λεπτομέρεια που πρέπει να λάβουμε υπόψιν στο DAPR είναι η χωρητικότητα του server. Η χωρητικότητα είναι το πόσες δουλειές μπορεί να κουβαλήσει ταυτόχρονα ο server. Στην εργασία μας, όταν αναφερόμαστε στο DARP μελετάμε αποκλειστικά τη περίπτωση με απεριόριστη χωρητικότητα.

Παρόλο που περιγράψαμε το κάθε πρόβλημα, δεν έχουμε ακόμη συζητήσει με ποιον τρόπο αξιολογείται μία λύση. Η μετρική που θα χρησιμοποιήσουμε είναι το άθροισμα των (βεβαρημένων) χρόνων ολοκλήρωσης. Συγκεκριμένα, οι αλγόριθμοι μας προσπαθούν να ελαχιστοποιήσουν το άθροισμα $\sum_j w_j c_j$. Στη περίπτωση που τα βάρη είναι μονοδιαία για όλες τις δουλειές τότε οι αλγόριθμοι προσπαθούν να ελαχιστοποιήσουν το $\sum_j c_j$.

Μία σημαντική λεπτομέρεια που δεν δηλώσαμε ρητά είναι πότε ο αλγόριθμος αντιλαμβάνεται την ύπαρξη μιας δουλειάς. Θεωρούμε το πλήρως άμεσο (online) μοντέλο, όπου ο αλγόριθμος δεν γνωρίζει τίποτα για μια δουλειά πριν τον χρόνο άφιξής της r_j. Αυτό είναι το αντίθετο του κλασικού, offline μοντέλου, όπου ναι μεν ο αλγόριθμος δε μπορεί να εκτελέσει δουλειές πριν την άφιξή τους, αλλά μπορεί να προετοιμαστεί κατάλληλα καθώς γνωρίζει τον προκείμενο ερχομό τους. Παραδείγματος χάριν, στο πρόβλημα TRP, αν μια δουλειά φτάσει μακριά από την αρχή του μετρικού χώρου στο offline μοντέλο μπορούμε να αρχίσουμε να κινούμαστε προς τα εκεί χωρίς να έχει καταφθάσει ακόμη. Στο άμεσο μοντέλο δεν μπορούμε να κάνουμε το ίδιο καθώς δε γνωρίζουμε τη δουλειά πριν την άφιξή της. Τέτοιοι περιορισμοί οδηγούν σε αποτελέσματα μη προσεγγισιμότητας για διάφορα προβλήματα. Κρίνουμε τους online αλγορίθμους με βάση το competitive ratio τους. Συγκεκριμένα, λέμε ότι ένας αλγόριθμος ALG είναι c-competitive όταν για κάθε δυνατή είσοδο $\mathcal{I}$ ισχύει $cost_{ALG}(\mathcal{I}) \leq c \cdot cost_{OPT}(\mathcal{I})$, όπου OPT είναι ο βέλτιστος offline αλγόριθμος. Φυσικά, για τη περίπτωσή μας, τα κόστη των αλγορίθμων είναι το άθροισμα του βεβαρημένου χρόνου ολοκληρωσης.

Η δουλειά μας βασίζεται κυρίως στην εργασία των Eden, Feldman, Fiat και Taub [23]. Στην εργασία αυτή εισάγεται η έννοια της δεσμευτικότητας, όπου κατά την άφιξη μιας δουλειάς ο αλγόριθμος την ενημερώνει ακριβώς για το πότε θα τελειώσει. Συγκεκριμένα, δημιουργούν μία ακολουθία που πετυχαίνει competitive ratio $O(\log L_{max})$, όπου L_{max} το

μέγιστο μήκος δουλειάς. Αυτή η ακολουθία είναι σε μεγάλο βαθμό προκαθορισμένη και κάθε δουλειά διαλέγει απλώς την θέση στην οποία θέλει να μπει από αυτές που της παρουσιάζονται. Ακόμη, αποδεικνύεται πως δεν γίνεται να πετύχουμε καλύτερο competitive ratio από $\Omega(\log L_{max})$. Εμείς επεκτείνουμε αυτή τη δουλειά ορίζοντας φορμαλιστικά την έννοια της δεσμευτικότητας. Λέμε ότι ένας αλγόριθμος είναι μερικώς δεσμευτικός αν ορίζει το χρόνο ολοκλήρωσης μιας δουλειάς κατά την άφιξή της. Δηλαδή, υπολογίζει s_j, e_j τέτοια ώστε $c_j \in [s_j, e_j]$. Ως χαλάρωση του αλγορίθμου ορίζουμε την ποσότητα $\lambda = max_j \frac{e_j}{s_j}$. Αν ένας δεσμευτικός αλγόριθμος έχει χαλάρωση 1 τότε λέγεται πλήρως δεσμευτικός.

Μια άλλη πτυχή που προσθέτουμε στους αλγορίθμους μας είναι αυτή της στρατηγικότητας. Συγκεκριμένα, θεωρούμε το μοντέλο του Σχεδιασμού Μηχανισμών. Κατά το μοντέλο αυτό, οι δουλειές είναι στρατηγικοί πράκτορες που δηλώνουν τις παραμέτρους τους στο μηχανισμό. Οι παράμετροι που δηλώνουν μπορεί να είναι το βάρος ή το μήκος. Ο μηχανισμός πρέπει να εμπιστευτεί αυτές τις δηλώσεις καθώς δε ξέρει τις πραγματικές τιμές τους. Κάθε δουλειά έχει μία κοινωνική ωφέλεια $u_j = -c_j - \pi_j$ ($u_j = -w_j c_j - \pi_j$ για βεβαρημένα προβλήματα), όπου θέλει να μεγιστοποιήσει. Παρατηρούμε ότι η μεγιστοποίηση αυτή προκαλείται από την ελαχιστοποίηση του χρόνου ολοκλήρωσης. Η τιμή π_j χρεώνεται από το μηχανισμό σε κάθε δουλειά j και είναι το ποσό που πρέπει να πληρώσει κάθε δουλειά. Μια δήλωση μιας δουλειάς που μεγιστοποιεί την ωφέλειά της λέγεται κυριάρχη στρατηγική. Ένας μηχανισμός στον οποίο οι αληθινές δηλώσεις είναι κυρίαρχες στρατηγικές λεγεται φιλαλήθης. Αυτό είναι και ένας από τους κύριους στόχους του σχεδιασμού μηχανισμών, δηλ. ο σχεδιασμός αλγορίθμων που "αναγκάζουν" τις δουλειές να λένε την αλήθεια.

Παρόλο που η δεσμευτικότητα και η φιλαλήθεια φαίνονται ως διαφορετικές έννοιες με μια πρώτη ματιά, είναι κοντινά συνδεδεμένες. Συγκεκριμένα, η δεσμευτικότητα είναι μία τόσο ισχυρή απαίτηση που αναγκάζει τους αλγορίθμους να έχουν προκαθορισμένα προγράμματα, ανεξάρτητα από τις δουλειές που θα έρθουν. Στις παράλληλες μηχανές αυτό είναι μία προκαθορισμένη ακολουθία με θέσεις για διάφορες δουλειές (και κάθε δουλειά διαλέγει τη θέση που μεγιστοποιεί την ωφέλειά της) ενώ στο TRP είναι μία προκαθοριμσένη περιοδεία του μετρικού χώρου. Επειδή τα προγράμματα αυτά έχουν πλήρως οριστεί εκ των προτέρων, οι δηλώσεις των δουλειών δεν έχουν σημασία καθώς δεν επηρεάζουν τα προγράμματα. Αυτό οδηγεί σε φιλαλήθεια κατά προφανή τρόπο.

Σε αυτή την εργασία, εισάγουμε μοντέλα για να προσπαθήσουμε να πετύχουμε δεσμευτικότητα με καλύτερο competitive ratio στο πρόβλημα των παράλληλων μηχανών. Στο TRP, χρησιμοποιούμε γνωστές προσεγγίσεις μετρικών χώρων για να σχεδιάσουμε μερικώς και πλήρως δεσμευτικούς αλγορίθμους. Από αυτούς τους δύο αλγορίθμους, ο μερικώς δεσμευτικός εφαρμόζεται και σε γενικούς μετρικούς χώρους. Ακολουθούν τα αποτελέσματά μας σε μορφή θεωρημάτων. Εκτός και αν σημειώνεται διαφορετικά, όλα τα θεωρήματα αποτελούν αμιγώς δική μας εργασία.

1.1 Το Πρόβλημα των Παράλληλων Μηχανών

1.1.1 Το Κύριο Μοντέλο

Παρουσιάζουμε ένα μοντέλο που ελπίζουμε να οδηγήσει σε δεσμευτικούς μηχανισμούς με $O(1)$ competitive ratio. Σε διακριτές χρονικές στιγμές $t = 0, 1, ..., T-1$ ο αριθμός δουλειών που παρουσιάζονται στο σύστημά μας είναι n_t για κάθε χρονική στιγμή t. Ο αριθμός των δουλειών n_t επιλέγεται από έναν αντίπαλο, οπότε μπορεί να το διαλέξει έτσι ώστε να μας αναγκάσει να πετύχουμε το χειρότερα δυνατό competitive ratio. Το μήκος κάθε δουλειάς στην ομάδα n_t επιλέγεται από μία πολυωνυμική κατανομή:

$$L = \begin{cases} 2^0 & \text{με πιθανότητα } p_0 \\ 2^1 & \text{με πιθανότητα } p_1 \\ 2^2 & \text{με πιθανότητα } p_2 \\ & \vdots \\ 2^K & \text{με πιθανότητα } p_K \end{cases}$$

με 2^K να είναι το μέγιστο δυνατό μήκος. Υποθέτουμε ότι τα μήκη των δουλειών είναι ανεξάρτητα και όμοια κατανεμημένα οπότε έχουμε μία πολυωνυμική κατανομή. Ακόμη, υποθέτουμε ότι τα μήκη είναι δυνάμεις του 2 χωρίς βλάβη της γενικότητας. Αυτό γιατί μπορούμε να τα αντικαταστήσουμε με την αμέσως μεγαλύτερη δύναμη του 2 και να έχουμε μία επιβράδυνση το πολύ κατά 2.

1.1.2 Ένας Ασυμπτωτικά Βέλτιστος Αλγόριθμος για το Κύριο Μοντέλο

Σε αυτή τη ενότητα, εισάγουμε έναν αλγόριθμο που είναι ασυμπτωτικά βέλτιστος για το Κύριο Μοντέλο. Να σημειωθεί ότι αυτός ο αλγόριθμος δεν είναι δεσμευτικός. Πρώτα, δείχνουμε ότι ένας αλγόριθμος με competitive ratio $O(1)$ δε μπορεί να είναι πλήρως δεσμευτικός, δηλαδή να καθορίζει πλήρως τον χρόνο ολοκλήρωσης μιας δουλειάς κατά την άφιξή της.

Λήμμα 1.1.1. Ένας $O(1)$-competitive αλγόριθμος για το κύριο μοντέλο 1.1.1 δε μπορεί να είναι πλήρως δεσμευτικός.

Η πλήρης απόδειξη βρίσκεται στην ενότητα 6.2. Η απόδειξη ακολουθεί την εξής λογική: Έστω ότι έρχονται Kn δουλειές τη χρονική στιγμή 0. Τότε, ο αλγόριθμος θα τις δρομολογήσει βάζοντας τις μικρότερες πρώτα και ίσως αφήνοντας Kn' θέσεις για άλλες δουλειές που ίσως έρθουν αργότερα. Υποθέτουμε ότι τα μήκη δουλειών είναι ισοπίθανα. Την επόμενη χρονική στιγμή, ο αντίπαλος στέλνει $K(n + 2n')$ δουλειές. έτσι, αναγκαζόμαστε να δρομολογήσουμε τις δουλειές ως $n + n'$ δουλειές από κάθε μήκος K και ύστερα ξανά $n + n'$ δουλειές από κάθε μήκος K. Το competitive ratio είναι $O(K)$ και ο αλγόριθμος δεν έχει σταθερή προσέγγιση.

Αφού δε μπορούμε να έχουμε έναν αλγόριθμο με σταθερή προσέγγιση και δεσμευτικότητα, προσωρινά εγκαταλείπουμε τη δεσμευτικότητα για να περιγράψουμε έναν ασυμπτωτικά βέλτιστο αλγόριθμο. Ο αλγόριθμος βασίζεται στο κανόνα ελάχιστου χρόνου, όπου κάθε χρονική στιγμή που η μηχανή είναι άδεια, δρομολογούμε τη δουλειά ελάχιστου μήκους. Αυτό μπορεί να οδηγήσει σε μία καθυστέρηση 2^K για μικρές δουλειές που φτάνουν αμέσως μετά τη δρομολόγηση μιας μεγάλης.

Αλγόριθμος 1.1.2
Σε κάθε διακριτή χρονική στιγμή που η μηχανή είναι κενή, δρομολόγησε τη δουλειά με το ελάχιστο μήκος από αυτές που περιμένουνε.

Θεώρημα 1.1.1. Ο αλγόριθμος 1.1.2 είναι ασυμπτωτικά βέλτιστος, έχοντας competitive ratio $1 + O(\frac{2^K}{n})$.

Η πλήρης απόδειξη βρίσκεται στην ενότητα 6.2. Η απόδειξη βασίζεται στο γεγονος ότι ο χρόνος ολοκλήρωσης κάθε δουλειάς καθυστερείται το πολύ κατά 2^K. Ύστερα, φράζουμε από κάτω το άθροισμα χρόνων ολοκλήρωσης για το βέλτιστο πρόγραμμα και διαιρώντας

την ανισότητα που προκύπτει από τη καθυστέρηση κατά 2^K, έχουμε το αποτέλεσμα. Η παραπάνω προσέγγιση είναι βέλτιστη, υπό την έννοια ότι ο αντίπαλος μπορεί να μας αναγκάσει να πληρώσουμε τον παράγοντα $O(\frac{2^K}{n})$ ως εξής: στέλνει αρχικά αρκετές δουλειές ώστε να εγγυηθεί την άφιξη μιας δουλειάς 2^K και ύστερα στέλνει έναν πολύ μεγάλο αριθμό από δουλειές. Οι καινούργιες δουλειές θα αναγκαστούν να περιμένουν τη δουλειά 2^K να τελειώσει πριν δρομολογηθούν.

1.1.3 Ένας Δεσμευτικός Αλγόριθμος με Σταθερή Προσέγγιση

Τροποποιούμε το μοντέλο την ενότητας 1.1.1 για να μπορέσουμε να πετύχουμε βέλτιστους αλγορίθμους που είναι πλήρως δεσμευτικοί. Ο περιορισμός που δε μας άφηνε να πετύχουμε πλήρους δεσμευτικούς μηχανισμούς ήταν ότι δε ξέραμε πόσες δουλειές θα έρθουν. Για να αντιμετωπίσουμε αυτό, υποθέτουμε n δουλειές να έρχονται κάθε διακριτή χρονική στιγμή $t \in [T] = \{0, 1, ..., T\}$. Ακόμη, υποθέτουμε ότι έχουμε έναν μεγάλο αριθμό μηχανών $m \geq n\mathbb{E}[L]$, όπου L είναι η τυχαία μεταβλητή που ορίζει το μήκος κάθε δουλειάς που έρχεται. Με βάση αυτό το μοντέλο, μπορούμε να κατασκευάσουμε έναν αλγόριθμο που είναι πλήρως δεσμευτικός και έχει ασυμπτωτικά βέλτιστη συμπεριφορά. Για να σχεδιάσουμε τον αλγόριθμο παρατηρούμε ότι $m \geq n\mathbb{E}[L] = n(np_0 + np_1 \cdot 2 + \cdots + np_K \cdot 2^K)$. Σε κάθε χρονική στιγμή t, η αναμενόμενη τιμή δουλειών που έρχονται είναι np_i για κάθε μήκος $i \in [K + 1]$. Για μεγαλύτερες δουλειές, "παίρνουμε" και περισσότερες μηχανές. Κατά τον ίδιο τρόπο, αναθέτουμε $np_i \cdot 2^i$ μηχανές για δουλειές μήκους 2^i.

Αλγόριθμος 1.1.3
Ανάθεσε $np_i \cdot 2^i$ μηχανές για δουλειές μήκους 2^i, δηλαδή όλες οι δουλειές μήκους 2^i δρομολογούνται αποκλειστικά σε αυτές τις μηχανές.

Θεώρημα 1.1.2. Ο αλγόριθμος 1.1.3 έχει $1 + O(\frac{1}{\sqrt{n}} \max_i \frac{\sqrt{1-p_i}}{\sqrt{p_i}})$ competitive ratio για το μοντέλο με τις $m \geq n\mathbb{E}[L]$ μηχανές και $n_t = n$ σταθερά.

Η πλήρης απόδειξη βρίσκεται στην ενότητα 6.3. Το κύριο σκεπτικό είναι να φράξουμε το competitive ratio κάθε μήκους δουλειών ξεχωριστά και ύστερα να πάρουμε το μέγιστο από αυτά. Αρχικά, παρατηρούμε ότι αν έρθουν ακριβώς np_i δουλειές μήκους 2^i για κάθε χρονική στιγμή, τότε οι $np_i \cdot 2^i$ μηχανές προλαβαίνουν να αδειάσουν np_i θέσεις σε χρόνο 2^i. Συνεπώς, για ακριβώς np_i δουλειές κάθε διακριτή χρονική στιγμή, κάθε δουλειά δρομολογείται ακριβώς στο χρόνο άφιξής της και ακολουθεί ακριβώς το βέλτιστο αλγόριθμο. Αν κάποια χρονική στιγμή t έρθουν περισσότερες δουλειές, τότε παρατηρούμε ότι κάθε επιπλέον δουλειά προκαλεί μία μονάδα χρονικής καθυστέρησης για κάθε np_i δουλειές που έρχονται μετά από αυτή. Αυτό μπορούμε να το φανταστούμε ως τη δουλειά να "σπρώχνει" τις επόμενες δουλειές πιο κάτω χρονικά και έτσι προκαλεί αυτή την καθυστέρηση. Για το βέλτιστο πρόγραμμα, το φράσουμε από κάτω με τη κάθε δουλειά να δρομολογείται ακριβώς όταν έρχεται στο σύστημα.

Ακόμη, ορίζουμε παραλλαγές στο μοντέλο αυτό, διατηρώντας πάντα τη δεσμευτικότητα των αλγορίθμων μας. Η πρώτη παραλλαγή που εξετάζουμε, είναι να έχουμε περιοδικότητα στις δουλειές που έρχονται. Συγκεκριμένα, κάθε h χρονικές στιγμές έρχονται $h \cdot n$ δουλειές βάσει κάποιας ακολουθίας. Είναι προφανές ότι η ακολουθία που είναι όσο λιγότερο ισορροπημένη γίνεται θα έχει το χειρότερο competitive ratio. Αυτή η ακολουθία είναι και οι $h \cdot n$ να έρθουν την πρώτη στιγμή της περιόδου. Μία ανάλυση παρόμοια με τη προηγούμενη οδηγεί στο παρακάτω θεώρημα:

Θεώρημα 1.1.3. Ο αλγόριθμος 1.1.3 έχει $1 + O(\max_i \frac{\sqrt{1-p_i}}{\sqrt{hn}\sqrt{p_i}}) + \frac{h}{T-h}$ competitive ratio για το μοντέλο με $h \cdot n$ δουλειές κάθε h διακριτές χρονικές στιγμές.

Στο παραπάνω competitive ratio, ο παράγοντας που οφείλεται στην ανισορροπία των δουλειών είναι ο $\frac{h}{T-h}$.

Μία ακόμη παραλλαγή που αναλύουμε είναι να έχουμε N^t δουλειές κάθε χρονική στιγμή t και τα N^t να είναι ανεξάρτητες τυχαίες μεταβλητές με όμοια κατανομή. Για αυτό το μοντέλο, ισχύει το παρακάτω θεώρημα:

Θεώρημα 1.1.4. Ο αλγόριθμος 1.1.3 έχει $1+O(\frac{\sqrt{K+1}}{\sqrt{n}})+O(\frac{\sqrt{\mathbf{Var}[N^0]}}{n})$ competitive ratio για το μοντέλο με N^t δουλειές κάθε χρονική στιγμή, με N^t τ.μ. ανεξάρτητες, με κοινή κατανομή.

Πάλι, ο επιπλέον παράγοντας είναι ο $O(\frac{\sqrt{\mathbf{Var}[N^0]}}{n})$ που προέρχεται από τη τυχαία φύση του αριθμού δουλειών που έρχονται.

1.1.4 Ένας Μερικώς Δεσμευτικός Αλγόριθμος

Μέχρι στιγμής παρουσιάσαμε αλγορίθμους που ήταν πλήρως δεσμευτικοί, όπως οι αλγόριθμοι της προηγούμενης ενότητας. Σε αυτή την ενότητα, θεωρούμε ξανά το Κύριο Μοντελο 1.1.1. Θα παρουσιάσουμε έναν αλγόριθμο που είναι ένας συνδυασμός του ασυμπτωτικά βέλτιστου αλγορίθμου της ενότητας 1.1.2 και της ακολουθίας του [23]. Όταν μία δουλειά παρουσιάζεται, της δίνουμε μία υπόσχεση ότι ο χρόνος ολοκλήρωσής της θα είναι z φορές ο χρόνος ολοκλήρωσης στην ακολουθία του [23] συν το μέγιστο μήκος 2^K. Συμβολικά, $c_j \leq z \cdot c_j^{MENU} + 2^K$. Ο προσθετικός παράγοντας 2^K μας επιτρέπει να δρομολογήσουμε μία μεγάλη μηχανή χωρίς να ανησυχούμε αν θα παραβιάσουμε κάποια προθεσμία. Ο αλγόριθμος είναι ως εξής:

Αλγόριθμος 1.1.4
Όταν η μηχανή είναι κενή και δε χρειάζεται να εκλπηρώσουμε κάποια πρροθεσμία, δηλ. $t + l_j \leq z \cdot c_j^{MENU}$ για όλες τις δουλειές j που δεν έχουν δρομολογηθεί ακόμη, δρομολόγησε τη μικρότερη δουλειά διαθέσιμη, αλλιώς δρομολόγησε τη δουλειά που η προθεσμία της είναι επικείμενη.

Ο αλγόριθμος είναι σχεδόν ο ίδιος με αυτόν της ενότητας 1.1.2 με τη προσθήκη διοριών. Αναλύουμε τη συμπεριφορά του αλγορίθμου 1.1.4 στη χειρότερη περίπτωση, δηλ. όταν οι μηχανή είναι γεμάτη με μικρές δουλειές που αναγκάζονται να πληρώσουν το κόστος όλων των διοριών.

Θεώρημα 1.1.5. Ο αλγόριθμος 1.1.4 έχει $O(\frac{2z(K-1)+z(z-1)(K^2+2K-1)}{(1+(z-1)(K+1))(2+(z-1)(K+1))} + \frac{2^K}{n})$ competitive ratio για το κύριο μοντέλο 1.1.1.

Η απόδειξη βασίζεται στο να υπολογίσουμε το πρόγραμμα για τη περίπτωση που η μηχανή είναι γεμάτη στις διορίες $z \cdot c_j^{MENU}$ και όλες οι υπόλοιπες θέσεις είναι γεμισμένες με δουλειές μήκους 1. Παρατηρούμε ότι για $z \geq 2$ ο αλγόριθμος έχει ασυμπτωτικά σταθερή προσέγγιση.

1.1.5 Φιλαλήθης Εκδοχή Αλγορίθμων Δρομολόγησης

Στην εργασία της Phillips μεταξύ άλλων [46], παρουσιάζονται αλγόριθμοι σταθερής προσέγγισης για τα προβλήματα $1|r_j|\sum c_j$ και $P|r_j|\sum c_j$. Οι αλγόριθμοι αυτοί δρομολογούν τις δουλειές χωρίς να τις σταματάν και να τις ξαναξεκινάνε (preemption) τροποποιώντας ένα preemptive πρόγραμμα σε ένα non preemptive. Αυτό γίνεται ταξινομώντας τις δουλειές σε φθίνουσα σειρά βάσει του χρόνου ολοκλήρωσης στον SRPT και ύστερα τροποποιώντας το αποτέλεσμα με βάση τον αλγόριθμο CONVERT, που μετατρέπει preemptive προγράμματα σε non preemptive. Αυτό επαρκεί για να πετύχουνε προσέγγιση 2 για τη μία μηχανή και 6 για τις δύο μηχανές. Εμείς χρησιμοποιούμε το μοντέλο και τις τεχνικές του Angel μεταξύ άλλων [3] για να μετατρέψουμε τους αλγορίθμους στο [46] σε φιλαλήθεις μηχανισμούς. Ξεκινάμε χρεώνοντας κάθε δουλειά μία τιμή π_j. Στο μοντέλο αυτό κάνουμε τις παρακάτω υποθέσεις: Μία δουλειά j έχει αξία το αρνητικό χρόνο ολοκλήρωσης, δηλ. $v_j = -c_j$. Ακόμη, η κοινωνική ωφέλειά της είναι quasi-linear, $u_j = -c_j - \pi_j$. Αφού επιτρέπουμε αρνητικές ωφέλειες, μπορούμε να επιχειρηματολογήσουμε ότι η ωφέλεια της δουλειάς αν δε τελειώσει, ο χρόνος ολοκλήρωσης είναι άπειρος και η ωφέλεια $-\infty$. Τέλος, υποθέτουμε ότι μπορούμε να αφαιρέσουμε μία δουλειά από το πρόγραμμα όποτε θέλουμε. Η τελευταία υπόθεση μας επιτρέπει να αφαιρέσουμε μια δουλειά αν ανακαλύψουμε ότι μας είπε ψέματα για το μήκος της. Θέτουμε κάθε πληρωμή $\pi_j = -start_j + \sum_{i \neq j}(r_i + b_i)$ όπου $start_j$ είναι ο χρόνος που η δουλειά j ξεκινά να εκτελείται και b_i η δήλωση που μας έκανε κάθε δουλειά i για το μήκος της. Αν υποθέσουμε φιλαλήθεια, η πληρωμή αυτή είναι έγγυρη και μη αρνητική. Ακόμη, η ωφέλεια της δουλειάς j είναι $u_j = -(start_j + l_j) - (-start_j + \sum_{i \neq j}(r_i + l_i)) = -l_j - \sum_{i \neq j}(r_i + l_i)$. Αν μία δουλειά μας πει ψέματα δηλώνοντας μικρότερο μήκος, τότε θα την αφαιρέσουμε από το πρόγραμμα τη στιγμή που θα τελειώνε η εκτέλεσή της και θα έχει ωφέλεια $-\infty$. Επιπλέον, το να δήλωνε μεγαλύτερο μήκος δε θα άλλαζε την ωφέλειά του. Συμπεραίνουμε ότι κάθε δουλειά δεν έχει κέρδος αν δηλώσει ψευδές μήκος και ο μηχανισμός είναι φιλαλήθης. Σημειώνουμε ότι τα παραπάνω υποθέτουν ότι ο μηχανισμός ξέρει τους χρόνους άφιξης και δεν τους δηλώνουν οι δουλειές.

1.2 Το Πρόβλημα του Πλανόδιου Επιδιορθωτή

Σε αυτή την ενότητα, περιγράφουμε το Πρόβλημα του Πλανόδιου Επιδιορθωτή (Travelling Repairman Problem ή TRP). Το TRP εξελίσσεται πάνω σε έναν μετρικό χώρο $\mathcal{M}$. Ο μετρικός χώρος είναι ένα συνολο σημείων με μία συνάρτηση απόστασης $d(\cdot,\cdot)$ που είναι συμμετρική και υπακούει την τριγωνική ανισότητα. Ένας εξυπηρετητής μπορεί να κινείται πάνω στον μετρικό χώρο $\mathcal{M}$ με μονοδιαία ταχύτητα. Ο εξυπηρετητής ξεκινάει τη χρονική στιγμή 0 από ένα ειδικό σημείο που ονομάζουμε την αρχή του μετρικού χώρου. Στα σημεία του χώρου καταφθάνουν δουλειές με άμεσο τρόπο. Ο χρόνος άφιξης της δουλειάς j συμβολίζεται με r_j και το σημείο στο οποίο καταφθάνει, με κατάχρηση του συμβολισμού, απλά ως j. Ακόμη, κάθε δουλειά έχει ένα βάρος w_j. Λέμε ότι μια δουλειά εξυπηρετείται όταν ο εξυπηρετητής καταφθάσει στο σημείο της. Το σημείο αυτό έχει απόσταση $d(o, j)$ από την αρχή. Σκοπός του εξυπερετητή είναι να εξυπηρετήσει τις δουλειές ελαχιστοποιώντας το άθροισμα του βεβαρημένου χρόνου ολοκλήρωσης.

1.2.1 Προσέγγιση Μετρικών Χώρων με Δέντρα

Στους αλγορίθμους μας προσεγγίζουμε γενικούς μετρικούς χώρους με δέντρα. Αυτό διευκολύνει την ανάλυση και μας επιτρέπει να αξιοποιήσουμε την εγγενή δομή των δέντρων για να σχεδιάσουμε περιοδείες πάνω στο μετρικό χώρο. Αρχικά, ας δώσουμε ακριβείς ορισμούς των μετρικών χώρων και της απεικόνισης μεταξύ μετρικών χώρων.

Ορισμός 1.2.1. Ένας μετρικός χώρος $\mathcal{M}$ είναι ένα διατεταγμένο ζεύγος (M, d), όπου M είναι ένα σύνολο σημείων και $d : M \times M \to \mathbb{R}$ είναι μία συνάρτηση απόστασης τέτοια ώστε για κάθε $x, y, z \in M$ τα παρακάτω ισχύουν:

$$d(x, y) = 0 \Leftrightarrow x = y$$
$$d(x, y) = d(y, x)$$
$$d(x, z) \leq d(x, y) + d(y, z)$$

Πολλές φορές, θα καταχραζόμαστε τον συμβολισμό του μετρικού χώρου και θα χρησιμοποιούμε τις έννοιες $\mathcal{M}$ και M εναλλάξ.

Ορισμός 1.2.2. Μία ενσωμάτωση (embedding) του μετρικού χώρου (X, d_X) στο μετρικό χώρο (Υ, d_Υ) είναι μία απεικόνιση $f : X \to Y$. Η παραμόρφωση της απεικόνισης είναι το ελάχιστο $D \geq 1$ τέτοιο ώστε για όλα τα $x, y \in X$:

$$d_X(x, y) \leq d_Y(f(x), f(y)) \leq D \cdot d_X(x, y)$$

Ακόμη, για πιθανοτικές απεικονίσεις, φράσουμε την παραμόρφωση κατά αναμενόμενη τιμή.

Αφού τα δέντρα είναι απλά γραφήματα, θα θέλαμε να ενσωματώσουμε γενικούς μετρικούς χώρους σε αυτά για να χρησιμοποιήσουμε τη δομή ους. Τα δέντρα στις προσεγγίσεις μας ονομάζονται ιεραρχικά διαχωριζόμενα δέντρα (hierarchical well separated tree or HST). Το HST είναι ένα δέντρο που τα βάρη των ακμών του μειώνονται καθώς πλησιάζουμε στα φύλλα.

Ορισμός 1.2.3. Ένα k-ιεραρχικά διαχωριζόμενο δέντρο (k-HST) είναι ένα ριζωμένο δέντρο με βάρη $T = (V(T), E(T))$ που ικανοποιεί τις παρακάτω ιδιότητες:

1. Για κάθε κόμβο $v \in V(T)$, όλες οι ακμές που συνδέουν το v με τα παιδιά του έχουν ίδιο βάρος.
2. Το βάρος της ακμής κατά μήκος ενός μονοπατιού από τη ρίζα προς ένα φύλλο μειώνεται τουλάχιστον κατά ένα παράγοντα k.

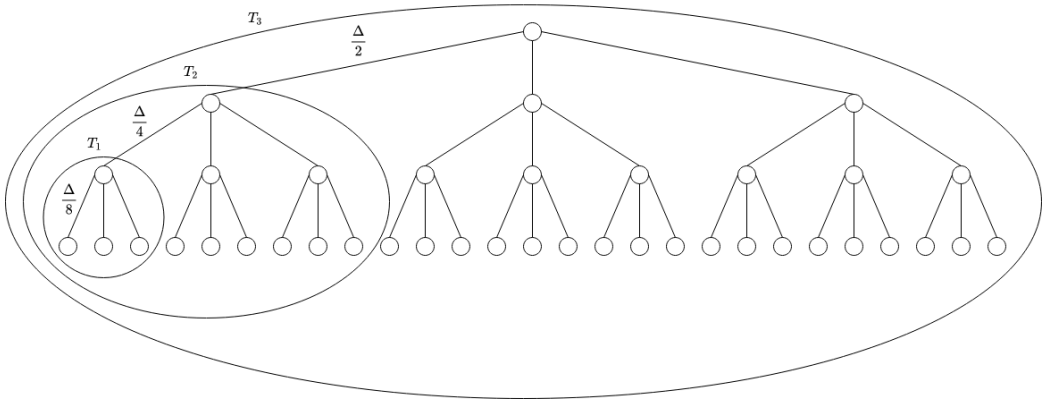
Τώρα ας παρουσιάσουμε ένα σημαντικό θεώρημα που αποτελεί βάση για την απεικόνιση σε δέντρα.

Θεώρημα 1.2.1. Έστω (X, d_X) ένας μετρικός χώρος n σημείων. Υπάρχει πολυωνυμικός πιθανοτικός αλγόριθμος που ενσωματώνει τον X στο σύνολο των φύλλων ενός 2-HST $T = (V(T), E(T))$ τέτοιο ώστε τα παρακάτω να ισχύουν:

1. Για κάθε $x, y \in X$ $d_X(x, y) \leq d_T(x, y)$
2. Για κάθε $x, y \in X$ $\mathbb{E}[d_T(x, y)] \leq O(\log n) \cdot d_X(x, y)$

Το παραπάνω θεώρημα αποδεικνύεται στην εργασία των Fakcharoenphol, Rao και Talwar [24]. Μέσω του παραπάνω θεωρήματος μπορούμε να προσεγγίσουμε μετρικούς χώρους με n σημεία με αναμενόμενη παραμόρφωση $O(\log n)$ και να χρησιμοποιήσουμε τη δομή του 2-HST για να χεδιάσουμε αλγορίθμους. Ας σημειωθεί ότι αφού τα κριτήρια αξιολόγησης αλγορίθμων που χρησιμοποιούμε μπορούν να γραφούν ως αθροίσματα αποστάσεων, η αναμενόμενη παραμόρφωση $O(\log n)$ πολλαπλασιάζει το competitive ratio c του αλγορίθμου στο 2-HST και έχουμε έναν $O(c \cdot \log n)$-competitive αλγόριθμο για γενικούς μετρικούς χώρους.

Χωρίς βλαβη της γενικότητα, υποθέτουμε ένα πλήρες δέντρο με βαθμό κορυφών d. Ακολουθεί ένα παράδειγμα για $d = 3$ και ύψος $h = 3$:



Τα βάρη των ακμών είναι τέτοια ώστε η απόσταση δύο φύλλων με κοινό πρόγονο τη ρίζα να είναι Δ. Ας σημειωθεί ότι αυτή είναι η διάμετρος του δέντρου και όχι του αρχικού μετρικού χώρου. Ακόμη, χωρίς βλάβη της γενικότητας, θεωρούμε ότι οι ακμές μεταξύ φύλλων και των προγόνων τους έχουν μήκος 1 και ότι η αρχή o είναι το αριστερότερο φύλλο. Αφού στους αλγορίθμους μας θα ασχοληθούμε με το μήκος του δέντρου, μας ενδιαφέρει το μήκος όλων των υποδέντρων. Ορίζουμε ως το δέντρο με ρίζα την αρχή ως T_0. Το δέντρο με ρίζα τον πατέρα της αρχής είναι το T_1 κ.ο.κ. Προφανώς, ολόκληρο το δέντρο είναι το $T = T_h$. Το μήκος μίας περιοδείας στο δέντρο T_1 είναι $d \cdot 1 = d$. Παρατηρούμε ότι ισχύει η παρακάτω αναδρομική σχέση:

$$|T_x| = d(|T_x| + 2^x) \tag{1.1}$$

όπου συμβολίζουμε το μήκος του δέντρου T_x με $|T_x|$. Από τη παραπάνω σχέση έχουμε:

$$|T_x| = \sum_{i=0}^{x-1} d^{x-i} 2^i \tag{1.2}$$

Από την (1.2) έχουμε ότι το μήκος του συνολικού δέντρου είναι $T_h = d^h + 2d^{h-1} + ... + 2^{h-1}d$. Για $d \geq 3$, ο όρος d^h είναι κυρίαρχος και έχουμε $|T_h| = O(d^h)$. Από την άλλη, για $d = 2$ έχουμε $|T_h| = O(h \cdot d^h) = O((\log \Delta)2^h)$. Αν χρησιμοποιήσουμε την ιδιότητα του πλήρους δέντρου, έχουμε ότι $d^h = n$, όπου n ο αριθμός σημείων του μετρικού χώρου. Οπότε, για $d \geq 3$, το μήκος μίας περιοδείας είναι $O(n)$ και $O(n \cdot \log \Delta)$ για $d = 2$. Χρησιμοποιούμε αυτό το συμβολισμό για να χαρακτηρίσουμε τα μήκη των περιοδιών μέσω παραμέτρων του μετρικού χώρου. Ξανά, ενώ n είναι ο αριθμός σημείων του αρχικού μετρικού χώρου, Δ είναι η διάμετρος του δέντρου.

1.2.2 Ένας Μερικώς Δεσμευτικός Αλγόριθμος για το TRP

Βασίζουμε τον αλγόριθμό μας στη δουλειά του Krumke μεταξύ άλλων [32, 38] για το άμεσο TRP. Ο αλγόριθμός τους χρησιμοποιεί προγράμματα γεωμετρικά αυξανόμενων μηκών, γυρνώντας στην αρχή μεταξύ προγραμμάτων. Σε κάθε τέτοιο πρόγραμμα, ο εξυπηρετητής συλλέγει τη μέγιστη ποσότητα βάρους χωρίς να ξεπεράσει ένα όριο μήκους. Συγκεκριμένα, ο αλγόριθμος χωρίζει το χρόνο σε διαστήματα $[b_{i-1}, b_i]$, όπου $b_i = (1+\sqrt{2})b_{i-1}$. Τη χρονική στιγμή b_i, ο αλγόριθμος υπολογίζει ένα πρόγραμμα μήκους το πολύ b_i το οποίο μαζεύει το μέγιστο δυνατό βάρος. Τη στιγμή b_i, ο εξυπηρετητής βρίσκεται το πολύ b_{i-1} μακριά από την αρχή και χρειάζεται άλλο b_i για το καινούργιο πρόγραμμα. Συνεπώς, $b_i + b_{i-1} + b_i = 2b_i + \frac{1}{1+\sqrt{2}} b_i = (1+\sqrt{2})b_i = b_{i+1}$ και μπορούμε να συνεχίσουμε έγκαιρα αυτή τη διαδικασία. Από εδώ προκύπτει και ο παράγοντας $1+\sqrt{2}$. Ο αλγόριθμος είναι $(1+\sqrt{2})^2$ προσεγγιστικός. Είναι προφανές, λόγω της φραγμένης φύσης του μετρικού χώρου που χρησιμοποιούμε, ότι μόλις τα μήκη των προγραμμάτων στον αλγόριθμο του Krumke ξεπεράσουν το μήκος του μετρικού χώρου, τότε όλες οι διαθέσιμες δουλειές μπορούν να δρομολογηθούν στο τωρινό πρόγραμμα που υπολογίζει ο αλγόριθμος. Αυτό οδηγεί στον παρακάτω μερικώς δεσμευτικό αλγόριθμο:

Αλγόριθμος 1.2.2

1. Ακολούθησε τον αλγόριθμο του Krumke μέχρι το μήκος των προγραμμάτων που υπολογίζει να ξεπεράσει το μήκος μιας συνολικής περιοδείας.
2. Κάνε επ' αόριστον συνολικές περιοδείες του μετρικού χώρου.

Θεώρημα 1.2.2. Ο αλγόριθμος 1.2.2 είναι $O(1)$-προσεγγιστικός για το άμεσο TRP με χαλάρωση $O(\frac{TSP_{tour}}{\Delta})$, όπου TSP_{tour} είναι το μήκος μίας συνολικής περιοδείας του μετρικού χώρου και Δ η διάμετρος.

Παρατηρούμε ότι ο παραπάνω αλγόριθμος μπορεί να εφαρμοστεί σε γενικούς μετρικούς χώρους. Η πλήρης απόδειξη βρίσκεται στην ενότητα 7.3. Η απόδειξη ακολουθεί την ανάλυση του Krumke για το βήμα 1. Για το βήμα 2, αρκεί η απλή παρατήρηση ότι θα περάσουν το πολύ 2 περιοδείες μέχρι την εξυπηρέτηση οποιουδήποτε σημείου και έχουμε προσέγγιση 3 σε σχέση με το βέλτιστο. Η χαλάρωση είναι τέτοια γιατί όταν καταφθάνει μία δουλειά σε απόσταση Δ υπάρχουν δύο ενδεχόμενα. Αν δεν έρθει καμία άλλη δουλειά ο αλγόριθμος θα την εξυπηρετήσει σε χρόνο $\Theta(\Delta)$. Αλλιώς, αν έρθουν πολλές δουλειές κοντά στην αρχή, η δρομολόγηση της μαρκινής δουλειάς μπορεί να καθυστερήσει μέχρι χρόνο $\Theta(TSP_{tour})$.

Επιπλέον, σημειώνουμε πως ο αλγόριθμος αυτός ίσως είναι μονόδρομος αν πρέπει να υποσχεθούμε στη δουλειά άνω φράγμα για το χονο ολοκλήρωσής της στη μορφή $c \cdot \max\{d(o,j), r_j\}$, όπου c είναι μία παράμετρος που εξαρτάται από την υπόσχεση του αλγορίθμου. Υποθέτουμε ότι μία δουλειά j έρχεται σε απόσταση Δ από την αρχή. Αν ένας αλγόριθμος έπρεπε να υποσχεθεί $c_j \leq c \cdot \max\{d(o,j), r_j\} = c \cdot \max\{\Delta, r_j\}$, τότε το c θα έπρεπε να λάβει υπόψιν και όλες τις άλλες δουλειές που μπορεί να έρθουν πιο κοντά. Αν μία δουλειά i ερχόταν σε απόσταση κοντινότερη $d(o,i) < d(o,j)$ τότε θα ήταν και $c \cdot \max\{d(o,i), r_i\} < c \cdot \max\{d(o,j), r_j\}$. Αυτό αληθεύει για όλες τις δουλειές που είναι κοντινότερα στην αρχή από τη j, δηλ. όλα τα σημεία του μετρικού χώρου, αφού η j είναι η

πιο μακρινή δουλειά. Οπότε έχουμε:

$$c \cdot d(o,j) \geq |T_h| \Leftrightarrow c \geq \frac{|T_h|}{\Delta} = O(\frac{d^h}{\Delta}) = O(\frac{n}{\Delta}) \tag{1.3}$$

ή $c = O(\frac{2^h \log \Delta}{\Delta}) = O(\log \Delta)$ για $d = 2$. Αυτό σημαίνει πως δε μπορούμε να δώσουμε πιο ακριβή υπόσχεση από τον αλγόριθμο 1.2.2.

1.2.3 Ένας Πλήρως Δεσμευτικός Αλγόριθμος για το TRP

Σε αυτή την ενότητα, θα παρουσιάσουμε έναν πλήρως δεσμευτικό αλγόριθμο σε 2-HST με competitive ratio $O(\frac{d^h}{\Delta})$ για $d \geq 3$ και $O(\log \Delta)$ για $d = 2$. Ο αλγόριθμός μας είναι απλός: κάνει διαδοχικές περιοδείες του 2-HST. Για να μην παραμελήσουμε δουλειές κοντά στην αρχή για πάρα πολύ, κάνουμε μία περιοδεία στο T_1 πρώτα, μετά στο T_2 κ.ο.κ. μέχρι να αρχίσουμε να κάνουμε περιοδείες ολόκληρου του δέντρου. Παρόλο που αυτός ο αλγόριθμος φαίνεται να έχει μέγαλο competitive ratio, δε μπορούμε να πετύχουμε κάτι καλύτερο για πλήρως δεσμευτικούς αλγορίθμους. Συγκεκριμένα, δείχνουμε ότι κάθε πλήρως δεσμευτικός αλγόριθμος για το άμεσο TRP έχει competitive ratio τουλάχιστον $\Omega(\frac{TSP_{tour}}{\Delta})$, όπου TSP_{tour} είναι το μήκος της βέλτιστης περιοδείας του μετρικού χώρου. Πρώτα αποδεικνύουμε το κάτω φράγμα και ύστερα αναλύουμε τον αλγόριθμο.

Θα παρουσιάσουμε μία ακολουθία δουλειών που αποτρέπει κάθε αλγόριθμο να πετύχει competitive ratio αυστηρά καλύτερο του $\Omega(\frac{TSP_{tour}}{\Delta})$. Δουλειές έρχονται σε ομάδες, μία ομάδα ανά σημείο του μετρικού χώρου. Η ακολουθία σταματάει όταν σε κάποια ομάδα ο αλγόριθμος υποσχεθεί χρόνο ολοκλήρωσης μεγαλύτερο ή ίσο του TSP_{tour}. Για αυτή την ομάδα, έχουμε φροντίσει από πριν ο αριθμός των δουλειών να είναι επαρκώς μεγάλος ώστε ο βέλτιστος αλγόριθμος να τις εξυπηρετεί πρώτες και αυτές μόνο να καθορίζουν το competitive ratio. Έτσι, το competitive ratio μπορεί να είναι οσοδήποτε κοντά στο $\Omega(\frac{TSP_{tour}}{\Delta})$.

Θεώρημα 1.2.3. Για φραγμένο μετρικό χώρο $\mathcal{M}$, ένας πλήρως δεσμευτικός αλγόριθμος είναι τουλάχιστον $\Omega(\frac{TSP_{tour}}{\Delta})$ competitive.

Η πλήρης απόδειξη βρίσκεται στην ενότητα 7.4. Τώρα, ας περιγράψουμε τον αλγόριθμο που πετυχαίνει το ίδιο πάνω φράγμα με αυτό το κάτω φράγμα:

Αλγόριθμος 1.2.3
Για ένα πλήρες δέντρο:

1. Περίμενε στην αρχή μέχρι $t = 1$.
2. Για $x = 1, 2, ..., h$ κάνε περιοδείες στο T_x, επιστρέφοντας στην αρχή ενδιάμεσα.
3. Κάνε επαναλαμβανόμενες περιοδείες σε όλο το δέντρο (T_h).

Θεώρημα 1.2.4. Ο αλγόριθμος 1.2.3 είναι πλήρως δεσμευτικός με competitive ratio $O(\frac{d^h}{\Delta})$ για $d > 2$ και $O(\log \Delta)$ για $d = 2$ για το άμεσο TRP.

Η απόδειξη βασίζεται στον υπολογισμό του μήκους των συνολικών περιοδειών. Για τη περίπτωση $d > 2$ ο παράγοντας d^h κυριαρχεί και έχουμε τη παραπάνω προσέγγιση. Ακόμη,

ο ίδιος αλγόριθμος έχει την ίδια προσέγγιση για το άμεσο Dial A Ride Problem (DARP). Στο DARP κάθε δουλειά που έρχεται έχει ένα σημείο άφιξης και ένα σημείο προορισμού. Σκοπός του εξυπηρετητή είναι να μεταφέρει κάθε δουλειά από το σημείο άφιξης στο σημείο προορισμού. Ο αλγόριθμος μας δουλεύει για το DARP όπου ο εξυπηρετητής έχει απεριόριστη χωρητικότητα, δηλ. μπορεί να κουβαλήσει όσες δουλειές θέλει ταυτόχρονα.

Θεώρημα 1.2.5. Ο αλγόριθμος 1.2.3 είναι πλήρως δεσμευτικός με competitive ratio $O(\frac{d^h}{\Delta})$ για $d > 2$ και $O(\log \Delta)$ για $d = 2$ για το άμεσο DARP με απεριόριστη χωρητικότητα.

Για τους μετρικούς χώρους 2-HST για οποιοδήποτε d, το κάτω και το άνω φράγμα ταυτίζονται. Ακόμη, ο αλγόριθμος των Feuerstein και Stougie [25] για μετρικούς χώρους στη γραμμή είναι πλήρως δεσμευτικός και έχει επίσης ίδιο άνω και κάτω φράγμα. Συγκεκριμένα, ο αλγόριθμος περνάει από τα σημεία $(-2)^i$ για φυσικό i και συνεπώς κάθε δουλειά ξέρει ακριβώς πότε θα εξυπηρετηθεί. Ο αλγόριθμος είναι 9-competitive για το TRP και 15-competitive για το DARP με απεριόριστη χωρητικότητα. Το κάτω φράγμα από το θεώρημά μας είναι $\Omega(1)$ αφού $TSP_{tour} = \Theta(\Delta)$, κάτι που είναι όμοιο με το άνω φράγμα $O(1)$. Συνεπώς, ο αλγόριθμος αυτός μαζί με τον δικό μας είναι όσο το δυνατόν καλύτεροι και μπορούμε να πετύχουμε επακριβώς το κάτω φράγμα για αυτούς τους απλούς μετρικούς χώρους.

Chapter 2

Introduction

This thesis is concerned with the design of prompt, truthful mechanisms for scheduling problems. Scheduling problems are by no means a new area of research. Countless works have been done in the area of scheduling, which in essence is the act of trying to arrange different tasks to minimize some time criteria. We enhance the regular scheduling problems by adding the constraint of promptness, a notion which restrains the completion time of a job. Promptness is a guarantee given by the scheduler upon the arrival of a job. This guarantee is of the form of a time interval, within the completion time of the job lies. Specifically, the scheduler specifies the completion time of a job to be between two certain values. The measure of how large is the gap between the two extreme values of the interval is the slackness. In particular, slackness is the ratio of these two extremes. For a scheduler that tells a job exactly its completion time, the slackness is 1 and the algorithm is called fully-prompt. On the other hand, the algorithm is semi-prompt.

Promptness is naturally a desired property of a scheduling algorithm. If an algorithm is prompt, a job knows exactly when its scheduling will take place. In the real world, such a guarantee would be of great magnitude for the people submitting the jobs to the system, as they can accordingly plan their day. Promptness is not only of practical importance but also allows us to extend our algorithms to truthful mechanisms. In the field of mechanism design, it is the aim of the algorithm designer to compute the outcome as to maximize a certain function denoting the gain of all jobs combined. This gain is called the social welfare. But jobs also have selfish objectives. Jobs submit values for their parameters to the mechanism, such as their length and weight. They might choose to submit the true values or try to lie and give false declarations. The jobs try and maximize their own function, which is their utility. They might try to achieve this by declaring a false value to the mechanism. For example, if the mechanism asks the length of a job on a machine, the job might lie to accomplish a greater utility. The aim of mechanism design is to reconcile the two, by designing truthful mechanisms, i.e., mechanisms that entice the jobs to declare their true values while also achieving an adequate social welfare. Truthfulness might have an impact on the attainment of the optimal social welfare as we will see.

One might ask what does this have to do with promptness. It turns out, promptness is such a strict requirement that, to achieve the fully-prompt version of it, one needs to specify the schedule exactly, independent of the jobs. This means that a machine accepting jobs will have a fixed sequence of jobs or a server that travels a metric space will have a set path it follows. The predetermined nature of this schedule means jobs have no motive to declare a false value as this will not change the schedule. As such, promptness gives rise

to truthfulness in an obvious way. The connection between promptness, truthfulness and the constraints one imposes on the other is the focus of this thesis and our main interest in resolving the relationship of the two.

2.1 Previous Work

In this section, we summarize the ideas and previous work that this thesis is based upon. One of the most prominent concepts in our work is that of online algorithms. In online algorithms, or the online setting as we call it, the input is not presented all at once to the algorithm. Instead, the input is revealed piece by piece. The extra hardness of this setting is that the algorithm has to make choices based on the input observed so far, without knowing what will come next. In the algorithms we discuss, our main goal is to minimize some cost function. We denote the cost of algorithm ALG by $cost_{ALG}(\mathcal{I})$ on online input $\mathcal{I}$. We asses the performance of an online algorithm by the competitive ratio, i.e., the ratio of the cost of the online algorithm to the cost of the optimal offline algorithm, which operates as if it knew the input beforehand. The competitive ratio is the maximum of the aforementioned ratios across all possible inputs. In accordance to this, we say that an algorithm ALG is c-competitive if $cost_{ALG}(\mathcal{I}) \leq c \cdot cost_{OPT}(\mathcal{I})$ [15].

We are to explore two problems in the online setting, the first of which is parallel machine scheduling [29]. In parallel machine scheduling we have m identical machines, which have the ability to execute tasks, or jobs as we call them. Jobs have a certain length, denoted by l_j for job j, meaning the amount of time their execution takes in each of the m machines. They also have a weight, w_j for job j. Jobs arrive at certain time points called arrival times, denoted by r_j for job j. Note that, since we are in the online setting, the algorithm is not aware of a job before its arrival time, in contrast to the offline setting. Of course, in both settings, a job can only begin execution after its arrival. The goal of the algorithm is to minimize the sum of weighted completion times, i.e., $\sum_{j=1}^{n} w_j c_j$, where n is the total number of jobs that arrive and c_j is the time at which execution of job j is completed. Furthermore, we allow some algorithms to pause execution of a job and continue it at a later time, possibly on another machine. We call this property preemptiveness. For the preemptive setting, one algorithm is to always schedule the m jobs (one in each machine) that have the highest priority, meaning the highest weight to remaining processing time ratio. This algorithm is called WSRPT and achieves a 2-competitive guarantee [57]. A simplification of the above algorithm is SRPT, were the weights of the jobs are unitary and the algorithm schedules the jobs with the least remaining processing time. For one machine, SRPT achieves the optimal result [8]. Another algorithm for parallel machine scheduling that is prominent in our work is the one by Phillips et al. [46]. In [46], the authors give the first constant approximation algorithm for parallel machine scheduling in the offline setting by converting preemptive schedules to non preemptive. For a detailed overview of offline and online machine scheduling please study chapter 3.

Another problem which concerns us is the Travelling Repairman Problem (TRP) [25]. The TRP takes place on a metric space, a graph with edge weights that obey the triangle inequality. Again, jobs arrive at points of the metric space (the nodes of the graph) at certain time points called arrival times r_j. Each job has a weight w_j. A server is positioned at a special point called the origin at time zero and can move through the metric space at unit speed. The objective of the server is to reach the position of each job, upon which the job is satisfied instantly. Again, the server aims to minimize the sum

of weighted completion times. Algorithms for the online TRP include the algorithm by Krumke et al. [38], which was the first constant approximation algorithm for the online TRP. The algorithm works in exponentially increasing schedules, within each accumulating the most job weight. Specifically, the algorithm executes a schedule of length at most b_i and gathers the maximum weight of jobs. To do this, solving the orienteering problem is needed [12]. The server then returns to the origin. The length of the schedules follows the following relation $b_i = (1 + \sqrt{2})b_{i-1}$. This algorithm achieves a $(1 + \sqrt{2})^2$ competitive ratio. Another algorithm that has an $O(1)$ competitive ratio is that of Feuerstein and Stougie [25]. Their algorithm works on line metrics and achieves the constant competitive ratio by performing a criss-cross movement around the zero point. More results on offline and online TRP variants are detailed in chapter 3.

A construction that enables the design of online algorithms for metric spaces is that of k-hierarchically well-separated trees (k-HST) [9, 24]. A k-HST is a rooted, weighted tree such that the weight of the edges along a path from the root to the leaves decrease by a factor of at least k. The practicality of HST's stems from their ability to approximate metric spaces. Specifically, we can embed metric spaces to the leaves of a 2-HST such that the expected distance of any two leaves-points on the tree is at most $\log n$ the distance on the original metric [24], where n is the number of points on the original metric. We use the term "expected" because the metric spaces are embedded to a distribution of trees through a random algorithm.

The first work to mention the notion of promptness is that of Eden et al. [23]. Their work tries to insert promptness in the problem of online parallel machine scheduling. Their algorithm constructs a sequence of exponentially increasing lengths. Since the sequences are predetermined, they are trivially prompt. Specifically, available slots are offered to jobs arriving, who choose the slot that minimizes their completion time. The adaptive version of the sequence is proved $O(\log L_{max})$-competitive for the online setting, while the static version of it is $O(\log L_{max} + \log n)$-competitive, where n is the number of jobs. Furthermore, they prove that the static sequence is $\Omega(\sqrt{L_{max}})$-competitive and that every prompt algorithm has a lower bound of $\Omega(\log L_{max})$.

Finally, ideas from mechanism design [42] are of prime focus in our work. Mainly, the work of Li [39] is one of interest. In [39], Li introduces the concept of obviously strategy-proof mechanisms, meaning mechanisms that the agents participating can easily understand what their best strategy is without expending a significant cognitive cost. The algorithm of Eden et al. is one such mechanism as the jobs just choose the earliest possible slot from those that are offered to them. Note that in this thesis, the social value of each outcome for the jobs is the negative completion time ($-c_j$ for job j) and maximizing the social value (utility) means minimizing the completion time.

2.2 Our Contribution

Our aim is to formalize the ideas of [23] and extend them to other problems and settings. We achieve this by formally defining the concept of promptness. In particular, we define a prompt algorithm to be one that specifies the completion time c_j of a job to be within a certain interval. If c_j is specified completely, meaning the algorithm tells the job exactly its completion time on arrival, the algorithm is called fully-prompt. Else, it is called semi-prompt. Next, we introduce a model where, at each discrete time point, an adversarially chosen number of jobs arrive, with their lengths following a multinomial distribution. The model assumes jobs of length 2^i, with $i = 0, 1, ..., K$. For this model, we

prove that no fully-prompt algorithm can achieve a competitive ratio of $O(1)$, in accordance with the results of Eden et al. [23]. We continue by introducing an asymptotically optimal algorithm for this model, which is $1 + O(\frac{2^K}{n})$-competitive, where 2^K is the maximum job length and n is the total number of jobs. The algorithm is based on a non-preemptive variant of WSRPT and makes use of the multinomial distribution to bound the competitive ratio. Furthermore, we explore the case of having a large number of machines and a constant number of jobs arriving each time point, which allows us to design a fully-prompt algorithm with an asymptotically optimal competitive ratio of $1 + O(\frac{1}{\sqrt{n}} \max_i \frac{\sqrt{1-p_i}}{\sqrt{p_i}})$, where this time n is the fixed number of jobs arriving each time point and p_i is the probability of a job of length 2^i. We also experiment with this model and give results for modifications based on the model. These modifications include the number of jobs arriving each second to be a random variable, for which we prove a $1 + O(\frac{\sqrt{K+1}}{\sqrt{n}}) + O(\frac{\sqrt{\mathbf{Var}[N_0]}}{n})$ upper bound, where N_t jobs arrive at time point t and N_t are i.i.d random variables. Another modification is having larger cycles of jobs arriving (with a period greater than one second), for which the competitive ratio is $1 + O(\max_i \frac{\sqrt{1-p_i}}{\sqrt{hn}\sqrt{p_i}}) + O(\frac{h}{T})$, where h is the period and T the total number of time points.

Moreover, we introduce truthful versions of the algorithms presented in Phillips et al. [46] using the utility model introduced by Angel et al. [3]. We, also, provide a semi-prompt version of the algorithm in Eden et al. [23] which is $O(1)$-competitive for a slackness value of at least 2. This semi-prompt algorithm cobines the ideas of our $1 + O(\frac{2^K}{n})$-competitive algorithm with the schedule of Eden et al.

Finally, we apply the ideas of promptness in the latency version of the *Travelling Repairman Problem* (TRP). We simplify the metric space by approximating with a *k-hierarchically well-separated* tree (k-HST) [9, 24]. More specifically, we approximate a general, bounded metric space with a 2-HST [24] and use this simplified metric space to construct prompt algorithms. We design two prompt algorithms: One is fully-prompt with a competitive ratio of $O(\frac{TREE_{tour}}{\Delta})$, where $TREE_{tour}$ is the optimal TSP tour of the 2-HST and Δ is the diameter of the tree. The other algorithm is a semi-prompt $O(1)$-competitive algorithm with an overall slackness of $\Theta(\frac{TREE_{tour}}{\Delta})$ and follows closely the online TRP algorithm in Krumke et al. [38]. In addition, we prove that for a fully-prompt algorithm in a general, bounded metric space the competitive ratio is at least $\Omega(\frac{TSP_{tour}}{\Delta})$, where TSP_{tour} is the optimal TSP tour of the metric space and Δ is the diameter of the space. This means that the fully prompt algorithm we describe has an optimal competitive ratio for tree metrics. We also draw a parallel with an already known algorithm that gives a constant competitive ratio on a line metric. This algorithm was first presented by Feuerstein and Stougie in [25] and turns out to be fully-prompt.

2.3 Organization of this Thesis

First of all, we give an in depth summary of this thesis in Greek in the chapter 1. The rest of this work is in English. In Chapter 3, we give a historic account of work on the Travelling Repairman Problem (TRP) and Parallel Machine Scheduling. In Chapter 4, we describe the notion of promptness while in 5, we connect promptness with truthfulness and explain why promptness in an property that can produce truthful mechanisms. In Chapter 6, we describe a probabilistic model of input jobs and present prompt mechanisms that deal with this model. Moreover, we generalize the work of Eden et al. [23] and introduce a semi-prompt variant of their sequence. In Chapter 7, we describe the TRP design tight

prompt algorithms in the case of 2-HST metrics for the online TRP while also giving a lower bound for fully-prompt algorithms.

Chapter 3

A History of Completion Time

In this thesis, we exclusively deal with scheduling problems and try to elevate their applicability by introducing promptness constraints. We are to explain the notion of promptness in later chapters, but its essence is that tasks ought to know, more or less, the time at which they will be completed. The word "exclusively" in the first sentence is, admittedly, misleading, as the bibliography of a single, specific scheduling problem could fill a book on its own. While scheduling problems can vary greatly in their setup, inner workings, algorithms that attempt to solve them, their motivation or their real world applications, they have a common trope that allows us to describe a solution in a very simple manner. The solution of a scheduling problem can be almost exactly described by an ordering of the tasks. To illustrate this, if we have a machine that sews suits and we have two orders, one large and one small suit, the best thing to do would be to start right away. We need to decide which suit to start sewing first. Later, we will find out that if we want to minimize the sum of completion times of the two suits, we need to start with the small one. As explained, the solution is determined by which piece of clothing we start to manufacture first. This would be slightly altered if we had two sewing machines and another worker, which allowed us to process requests in parallel. Then, we need also specify the machine at which each suit will be crafted at. Another scheduling problem arises when we have completed the production of the suits and we want to deliver them to the owners' homes. Which home should we go to first? Do we have two delivery employees or one? Can they both carry the same weight? Nuances like these create a whole array of scheduling problems, which we will try to succinctly present in this chapter.

The first problem that was made apparent in our aforementioned example is that of job scheduling on parallel machines. In this problem, requests come one by one at a time specific to them, called the arrival time. This was not the case in the above example, as both suit orders were available as soon as we could start sewing. In the general case, jobs arrive at a time of their choosing and inform the the algorithm of their presence. This is quite a natural setting, as the algorithm should not be expected to know when the jobs will arrive beforehand. After their arrival, the jobs are available to begin execution. The execution takes place in an entity called a machine, which can handle at most one job at a time. There could be many machines or just one. After a machine starts the execution of a job, the machine is committed to to that job until its completion. Where that not the case, meaning a job can be stopped and continued at a later time, possibly on another machine, then we would called the algorithm preemptive.

The second problem, which is to be analyzed in this thesis, is the Travelling Repairman

Problem or TRP. In the TRP, there is a server which can move across a metric space. The metric space is a distance relation between points, which assigns a positive, symmetric distance between two points and satisfies the triangle equality. The server starts at a special point, called the origin, at time zero and moves with constant speed. Requests present themselves at points of the metric space one by one. The earliest moment in which a request is presented is called its arrival time. A request is said to be served, if the server reaches its location after its arrival. This correlates to the second problem illustrated in the above example, i.e., the delivery of the suits. The delivery man, server, needs to reach the houses of the suit owners, requests, and deliver the suits. Again, the requests are known beforehand. A better example would be that of a locksmith. A locksmith receives requests to travel to a specific house and unlock a door. If we assume the locksmith is a master at his/her trade and can unlock a door almost instantly, then this problem is exactly the TRP.

Another problem, which is closely related to the TRP, is the dial a ride problem or DARP. DARP is very similar to TRP, with the slight alteration that requests have a source and a destination in the metric space. To serve a request, the server needs to pick it up at its source and deliver it to its destination. This problem produces more variants, as the question of the capacity of the server comes into play. How many requests can the server carry at a time? While important, we will not focus on DARP. It is mentioned, however, because the TRP is a special case of it (source and destination coincide) and, as such, lower bounds for the TRP also apply to DARP.

While the basis of the problems was described, we have not yet discussed the gauge in which these problems will be evaluated. We are to solely judge the performance of an algorithm based on its sum of completion times, or else, average completion time. A request's completion time is the time at which the request has been served or handled. For a job in the machine scheduling setting, this is the time at which a machine finished the execution of the job. For the TRP, it is the time at which the server reached the request, and as such, served it. Obviously, the lesser the sum of completion times is, the better the algorithm performs. This is because the jobs had to cumulatively wait for less time for their overall completion. A small variant of the average completion time is the weighted version of it. In the weighted version, each job has a weight which corresponds to its importance. We would like important jobs to complete first, so we multiply each completion time by the weight of its job. The weighted completion time is the sum of weight and completion time products across all jobs. As such, the weighted completion time is small when large weight jobs complete first.

A small, yet important, detail which we have not explicitly stated, is what does the algorithm know of the jobs arriving. More specifically, *when* is the algorithm made aware of a job that arrives. In this thesis, we solely explore the online setting of the problems, meaning the algorithm learns of a job's presence only after its arrival. Before a job's arrival time, the algorithm is unaware of its impending coming. This is in contrast to the offline setting, in which the algorithm knows that a job arrives at a certain time in the future and simply cannot schedule it until then, but can prepare beforehand for its arrival. For example, in the offline TRP the server could start to move towards a future job's location to service it immediately. In the online setting, the server could not know that the job would appear there before its arrival time. We judge online algorithms based on their competitive ratio c. We say that an algorithm ALG is c competitive, if for any input of requests $\mathcal{I}$, it holds that $cost_{ALG}(\mathcal{I}) \leq c \cdot cost_{OPT}(\mathcal{I})$, where OPT is the optimal, offline algorithm [15]. Of course, the costs are measured as the sum of (weighted) completion times. For randomized algorithms, the costs are substituted by expected costs.

Another idea, which we will try to incorporate to existing algorithms, is the notion of promptness. We will exactly define promptness in the following chapter, but for now it suffices to know that promptness promises a job that its scheduling will happen at approximately a given time. Specifically, on the job's arrival, the algorithm promises the job that its completion (service) will happen at certain point in time (or within a certain time interval). This is, of course, a much desired property, if we are to consider daily settings. When you go to the tailor to order a new suit, you expect to be informed on the spot about when the order will be ready. If the tailor's only consideration was minimizing the sum of completion times, then the tailor would like to schedule smaller orders first. Subsequently, our suit would never be made, provided enough smaller tasks presented themselves adequately often. Such a behaviour would, of course, be unacceptable and would cause many to avoid the particular shop. Promptness tries to remedy this by promising jobs that their scheduling will occur at a specified time in the future. This has a detrimental impact on the competitive ratio, which deteriorates because of it. We will try to give accurate upper and lower bounds that promptness has on the problems of online scheduling and TRP.

Promptness, in turn, ties with the idea of strategic choice by the jobs that arrive. In following chapters, we will describe jobs as strategic agents that report their attributes (weight, length, etc.) to our algorithm. They might misreport their true numbers, if that means they will achieve greater value from the outcome of the algorithm. This value is measured in the sum of (weighted) completion times negated. The lesser the sum is, the greater the value of the jobs. This makes the choice of minimizing completion time especially relevant, as minimization of completion time leads to maximization of value collected by the jobs. Promptness helps achieve truthfulness, a property of algorithms that means jobs have an incentive to report their true attributes, through fixing a schedule an constricting each job to a specific spot. If the algorithm needs to specify exactly a job's completion time, then the job's position in the schedule is set in stone, and thus cannot gain more by lying.

The above ideas may seem disparate at first, but they are intrinsically interlinked. Promptness leads to truthfulness and truthfulness would like for jobs to maximize their value, thus minimizing the completion time. For the rest of this chapter, we are to give a historical presentation of the advancements made in the minimization of completion time in the online setting, for the TRP and the machine scheduling problem.

3.1 Parallel Machine Scheduling

For the problem of minimizing the sum of completion times in parallel machines, parallel meaning all machines are identical and every job has the same length on all machines, we shall focus on both offline and online results. The simplicity of the earliest known algorithms, such as Smith's rule [55], allow us to construct algorithms for more complex settings then the online one, settings that involve probabilistic distribution of job lengths, fixed number of jobs coming at each second, etc. The analysis of such settings will be done in chapter 6. In this section, we shall give a historic presentation of the most important and pertinent to this thesis results in both online and offline settings. But, before this historic recollection, we shall introduce some notation to better signify these problems.

We shall use the standard notation for machine scheduling problems, introduced by Graham et al. in [28]. A machine scheduling problem is defined by three parameters

α, β, γ and denoted by $\alpha|\beta|\gamma$. The parameters can take on the following values:

- α
  - 1: Jobs are scheduled only on one machine.
  - P: Jobs are scheduled on m identical machines.
  - R: Jobs are scheduled on m unrelated machines, i.e., they may have different lengths on different machines.
- β
  - $pmtn$: Preemption is allowed, meaning jobs can be stopped and continued at a later date, possibly on another machine.
  - $prec$: A set of precedence constraints on the jobs. If $i \prec j$ then starting time of j should be no earlier than completion time of i.
  - r_j: A job can only be scheduled after time r_j, its arrival time.
  - $online - r_j$: A job can only be scheduled after time r_j, its arrival time and the algorithm is not aware of the job's arrival until time r_j.
- γ
  - $\sum c_j$: The objective is to minimize the sum of completion times.
  - $\sum w_j c_j$: The objective is to minimize the sum of weighted completion times.

For example, $1|r_j, pmtn|\sum c_j$ is the problem of minimizing the sum of completion times on one machine with arrival times and preemption allowed and $P|online - r_j|\sum w_j c_j$ is the weighted completion time minimization with arrival times and multiple, parallel machines in the online setting.

For single machine scheduling, the minimization of weighted completion time comes from ordering jobs according to Smith's rule, i.e., scheduling jobs in non-decreasing order of $\frac{l_j}{w_j}$, where w_j is the weight and l_j is the length of job j. Using the above notation, the problem would be denoted as $1||\sum w_j c_j$. Smith's rule was first introduced in [55], and it is one of the most important ideas in scheduling theory. In [55], the author proves that this ordering is optimal, meaning no other order can achieve a better sum of weighted completion times. This is an anomaly; as we will discover later in this thesis, one rarely should hope for an optimal algorithm. Most of the time, we shall be content with constant approximations on the optimal value, and even these will prove to be out of reach in many scenarios.

The problem discussed so far dealt with jobs all available at time zero, a simplification that restricts this model from having real world applications. A generalization would be to allow jobs to arrive at any time, and being able to start execution only after their arrival. We denote this arrival time by r_j for job j. Even in the case of one machine with unit weights, the problem of finding an optimal schedule is NP-hard [16]. However, even in the general case of m machines and weighted jobs with arrival times there exists an near optimal algorithm if we allow jobs to be preempted. If we allow preemption, jobs can be stopped and continued (without having to start executing from the beginning) at a later time, possibly on another machine. The Weighted Shortest Remaining Processing Time algorithm (or WSRPT), [57], schedules jobs based on smallest remaining weight-to-size ratio. The authors in [57] prove that this algorithm is 2-competitive for identical machines,

with jobs having arrival times and weights. Furthermore, the algorithm is optimal in the case of $1|r_j, pmtn|\sum c_j$ [8]. In this thesis, we will use this algorithm (and its unweighted version) to measure the performance of our algorithms. It is easy to see that WSRPT can be extended to the online setting as it only considers jobs available at the system at any time.

Besides the simple algorithms mentioned before, there have been many results in the area of parallel machine scheduling. Megow and Schulz [40] present slight variations of the WSRPT that achieve constant competitive ratios. Their algorithm P-WSPT achieves a 2 competitive ratio for $P|online - r_j, pmtn|\sum w_j c_j$ by scheduling the m jobs with the highest weight-to-length ratio among the available. This result is also tight. For $P|online - r_j|\sum w_j c_j$, they argue that scheduling a job with high weight-to-length ratio might not be a good idea because a job with larger ratio might appear right after. They remedy this by delaying jobs according to their length. Specifically, SHIFTED WSPT delays a job with arrival time r_j until a point r'_j between $\max\{r_j, al_j\}$ and $r_j + al_j$ for some a. Whenever a machine becomes idle, schedule the job with the highest weight-to-length ratio among those who have arrived (with respect to the new arrival times r'_j). SHIFTED WSPT achieves 3.28-competitiveness for any number of machines m.

Schulz and Skutella [49], present a 2-competitive randomized algorithm for the problems $P|online - r_j, pmtn|\sum w_j c_j$ and $P|online - r_j|\sum w_j c_j$. The algorithm, named RANDOM ASSIGNMENT, assumes a virtual, fast machine with m-speedup, meaning each job runs m times as fast on this machine, Then, the jobs are scheduled preemptively, with regards to weight-to-length ratio $\frac{w_j}{l'_j}$, where $l'_j = \frac{l_j}{m}$ is the processing time on the fast machine. The schedule on the fast machine will be used to determine the actual schedule on the m machines. Each job j is independently and uniformly assigned to a machine, while also a_j is uniformly sampled from $[0, 1]$. The jobs then, are scheduled non-preemptively on their machines in order of non-decreasing $c^S_j(a_j)$, where $c^S_j(a_j)$ is the time at which a_j fraction of the job has been completed on the fast machine. Note that the non-preemptive algorithm RANDOM ASSIGNMENT is also 2-competitive with regards to the optimal preemptive solution.

For the non-preemptive problem $P|online - r_j|\sum w_j c_j$, Correa and Wagner [21] improve the competitive ratio to 2.62 with the deterministic algorithm NAS (Non-preemptive a scheduling). This algorithm solves an LP-schedule on a fast machine as RANDOM ASSIGNMENT and schedules jobs on machines, again, according to their a_j points. Finally, Sitters [51] gave a $(1.79+O(1/\sqrt{m}))$-competitive algorithm by combining the technique of delaying jobs from [40] and the scheduling on a fast machine from [21, 49].

One special setting that deserves to be mentioned is non-migration. In the preemptive setting, jobs can be stopped and continued later on another machine. Non-migration restricts this by disallowing jobs to change machine; they may be stopped and continued later, but it has to be on the same machine. Avrahami and Azar [7] give a 14-approximate algorithm for completion time with immediate dispatching, meaning jobs are instantly assigned to a machine when they arrive. This restriction is close to our own work, as it constrains preemption by forbidding jobs to change machines, while also having a close to promptness idea in the form of assigning immediately to a machine. In our work, we assign jobs immediately to a slot.

In the offline setting, the first constant approximation algorithms for many setting were given by Phillips et al. [46]. They prove two lemmas that help them convert preemptive schedules to non preemptive. Specifically, they Phillips et al. show that that ordering the jobs non-preemptively in order of their completion time in the preemptive schedule, only increases the completion time by a factor of 2 for one mahine and 3 for

many machines. They then use this result to design non-preemptive algorithms based on preemptive schedules. By converting the schedule of SRPT on $1|r_j, pmtn|\sum c_j$ to a non-preemptive, they achieve a 2 approximation algorithm for $1|r_j|\sum c_j$. This is because SRPT is optimal. Furthermore, this algorithm can also be applied in the online setting, giving a 2-competitive algorithm for $1|online-r_j|\sum c_j$. Since Hoogeveen and Vestjens [30] showed one cannot achieve a better competitive ratio than 2 on $1|online - r_j|\sum c_j$, the algorithm is optimal. For $P|r_j|\sum c_j$, the authors give a preemptive, 2-approximate algorithm (same as P-WSPT in [40]) and attain 6-approximation by converting the schedule to a non-preemptive. Finally, by rounding the fractional solutions generated by a linear programming relaxation of an integer program to valid preemptive schedules, they give a $8+\epsilon$ approximation to $R|r_j, pmtn|\sum w_j c_j$, converting it to $16+\epsilon$ and $24+\epsilon$ approximations for $1|r_j|\sum w_j c_j$ and $P|r_j|\sum w_j c_j$ respectively.

Improvements on the above were given by Sitters [53], showing a 1.698 approximation on $P|r_j, pmtn|\sum w_j c_j$, while an upper bound of 1.5 was proven by Skutella [54] for the non-preemptive case using convex quadratic programming relaxations. In addition, a PTAS is given by Afrati et al. [1] for a constant number of machines.

3.2 The Travelling Repairman Problem

The first attempt at an online algorithm for the TRP was made by Feuerstein and Stougie [25]. In their work, they give an online algorithm, named Blindly Construct the Route, or BCR, that achieves a 9-competitive ratio for the TRP. Unfortunately, BCR is only valid for line metrics, a special case of metric spaces that consist only of points in a straight line. BCR works as follows: Let σ be the earliest possible time that a request can be served. This could mean that the request has a small enough arrival time and is relatively close to the origin. The algorithm starts following a back and forth motion, going through points $(-2)^k\sigma$ for $k = 0, 1, 2, ...$, where it is assumed that the origin is at the zero point in the line. Intuitively, BCR follows a fixed (up to the normalization constant σ) route in a criss-cross motion. As we shall later discuss, this provides the prompt property as each job knows the exact route that the server is going to take, and thus it knows its exact completion time on arrival. Furthermore, it is noteworthy that the same algorithm is 15-competitve for the DARP with an infinite capacity server.

A more general algorithm, able to handle general metric spaces, was given by Krumke et al. [38]. Their algorithm, $\textbf{INTERVAL}_{1+\sqrt{2}}$, achieves a competitive ratio of $(1+\sqrt{2})^2 < 5.8285$. To get such a result, $\textbf{INTERVAL}_{1+\sqrt{2}}$ again uses the parameter σ discussed earlier, which is the earliest time at which a request can be served. The algorithm, specifies the points in time of the form $B_k = (1+\sqrt{2})^k\sigma$ for all natural k. At time B_k, the server returns to the origin and computes a schedule of maximum weight and of total length less than B_k. Since the distance from the origin at time B_k is less than or equal to B_{k-1} (this comes from the previous schedule computed at time B_{k-1}) we have that $B_{k-1} + B_k + B_k = \frac{1}{1+\sqrt{2}}B_k + B_k + B_k = (1+\sqrt{2})B_k = B_{k+1}$. This means we are in time to start computing the next schedule at time B_{k+1}, having length B_{k+1} (after spending B_k to return to the origin) and we can continue this process indefinitely.

$\textbf{INTERVAL}_{1+\sqrt{2}}$ was the first constant competitive ratio algorithm for general metric spaces. However, a detail of the algorithm might render it impractical for ordinary use. The algorithm of [38] requires the solution of an NP-hard problem. The calculation of a schedule that accumulates the most weight while having a constraint on the total distance to travel (schedule cannot exceed B_k) is the well known orienteering problem [12].

However, this restriction is not within the scope of our work. We eschew computational complexity problems and are more concerned with the information theoretic aspects of such problems.

The algorithm of Krumke et al. [38], was elevated by Jaillet and Wagner in [32]. In [32], the authors imbue $\mathbf{INTERVAL}_{1+\sqrt{2}}$ with the added power of advance notice, creating the *BREAK* algorithm. Specifically, they assume a model in which each job's arrival is made known to the algorithm a units of time before its actual arrival. This is a compromise between the offline and the online setting. Using the same technique as [38], the authors achieve a competitive ratio of $(1+\sqrt{2})^2 - \frac{\alpha\beta}{\alpha+\beta}$, where $\alpha = \frac{a}{OPT_{TSP}}$, $\beta = \frac{a}{r_{max}}$, OPT_{TSP} is the length of the optimal TSP tour and r_{max} is the maximum arrival time.

Both $\mathbf{INTERVAL}_{1+\sqrt{2}}$ and *BREAK* have an inherent property of promptness. If they are left to run for long enough, their schedule length will continue to increase. For bounded metric spaces, this length will eventually surpass the total tour length over all points. This means that the algorithm will degenerate to repeated optimal TSP tours. Hence, we can make a very vague promise to every arriving job: Your scheduling will occur between $\max\{d_j, r_j\}$ and $O(OPT_{TSP})$, where d_j is the distance from the origin and r_j the job's arrival time. This promise does, indeed, give a tenuous promptness guarantee. However, in later chapters, will give an exact promise of completion time for every job and prove a lower bound for it.

An improvement on Krumke et al. for line metrics was given by Bienkowski and Liu [11]. In [11], the authors construct a schedule with a slight modifications, that allow it to service jobs further in the line a little earlier compared to OPT. The authors achieve this by allowing the server to move further away from the origin during later parts of a exponential phase. The analysis follows from a set of linear inequalities (from comparing the weight of serviced jobs between their algorithm and OPT) which leads to a maximization factor-revealing linear program, whose solution is an upper bound on the competitive ratio. Finally, they give a solution to the dual linear program. This results in a competitve ratio of $2\sqrt{2} + \frac{13}{5} < 5.429$.

Furthermore, Hwang and Jaillet [31] use geometrically increasing schedules that lead to a factor-revealing LP. This in turn achieves a 5.14-competitive algorithm. The most recent advancement in the field was made by Bienkowski, Kraska and Liu [10]. Again, they derive a factor-revealing LP from auxiliary schedules that gives a 4-competitive algorithm for TRP and DARP, no matter the number of machines or the capacity of the servers (in the case of DARP).

For randomized algorithms, Krumke et al. [37] give a $\frac{2+\sqrt{2}}{\ln(1+\sqrt{2})} < 3.8738$-competitive algorithm, while Hwang and Jaillet [31] achieve a competitive ratio of $\frac{4}{\ln 3} < 3.641$. Finally, in Bienkowski et al. [10] the randomized competitive ratio is lowered to 2.821, for all version of TRP and DARP (number of machines and capacity of servers for the DARP).

To get a sense how close these results are to the best possible, we briefly mention some lower bounds for the online TRP. Feuerstein and Stougie [25] give a lower bound of $1+\sqrt{2}$. For many servers, a lower bound of 2 is proved by Bonifaci and Stougie [14]. Finally, a lower bound of $\frac{7}{3}$ is given in Krumke et al. [38], in the randomized setting.

There has also been much work on the offline variant of the TRP. From the perspective of computational complexity, TRP was shown to be NP-hard on general graphs by Sahni and Gonzales [48]. Specifically, their reduction proved that no bounded approximation, polynomial time algorithm can exist for the TRP unless P=NP. The reduced the Hamiltonian Cycle on the original graph to the TRP as follows: They augment the original graph by adding all missing edges and assigning them with weight k. It is obvious that if the

TRP has a solution of sum of completion times less than $\frac{n(n+1)}{2}$, then the original graph has a Hamiltonian Cycle. If the original graph has no Hamiltonian Cycle, then it must use at least one edge of weight k. By choosing k sufficiently high, one can conclude from the ϵ-approximate solution to the TRP if the original graph has a Hamiltonian Cycle. Thus, for any ϵ, there cannot be a polynomial time ϵ-approximate algorithm for the TRP on general graphs.

Even on trees, the weighted TRP is NP-hard even if all edges have unit length [50]. In [50], Sitters showed that the problem of 3-Partiton can be reduced to weighted TRP on trees, by assigning a linear branch to each value in the set to be partitioned. It was shown, that the set can be partitioned if and only if the optimal sum of completion times is sufficiently smaller than the sum of completion times if every branch is served one by one, without turning back. Furthermore, TRP was shown to be MaxSNP-hard by a reduction from the Traveling Salesman Problem with distances 1 and 2 [13, 45].

There have been many algorithms for special cases of the offline TRP. Afrati et al. [2] gave an exact algorithm with $O(n^2)$ running time for line metrics, where n is the number of points on the line. The algorithm is based on dynamic programming, as at each step the algorithm can make only one of two choices; go right or left. This result was improved by García et al. [44], who gave an linear time algorithm for line metrics by reformulating the problem in a different dynamic programming way than [2]. Similar ideas can be applied to tree metrics. Minieka [41] and Koutsoupias et al. [35] gave dynamic programming algorithms for the TRP on trees with $O(n^{k+1})$ and $O(n^k)$ time complexity, respectively, where k is the number of leaves. Finally, for some special cases of metrics (Euclidean plane, planar graphs or weighted trees), the TRP admits a PTAS [5, 52].

Approximations for the offline weighted TRP are possible when the graph is a metric space, i.e., it obeys the triangle inequality. In Blum et al. [13], the authors present a 144-approximation on the offline TRP, the first constant approximation for the TRP. The approximation is based on bounding the latency of nodes between k-MST's, meaning spanning trees with k nodes. Using as k-MST's approximations for the TSP, the authors are able to achieve the aforementioned approximation. Goemans and Kleinberg [27], strengthen the bound of the TSP approximators introduced in [13] and combined with a 2 approximation for the k-MST [4, 6], they give a 7.18-approximate algorithm. The best known ratio for the TRP in general metrics is thanks to Chaudhuri et al. [17], where the authors use the primal-dual method to find a near optimal k-tree, thus lowering the approximation for the TRP to 3.59.

Chapter 4

Promptness

The algorithms discussed so far have a common theme: The job scheduled is not aware when its execution will take place. We can think of them as telling the job when it arrives "OK- hang around for some time and you will be serviced at a later date". Of course, there are many real life settings where such an uncertainty is undesirable. Consider the following example: You want to refill your electric car's battery at a recharging station. When you reach the station hopefully you are serviced immediately so you can be on your way. However, if such a lucky event does not take place, you want to at least have knowledge of the time when your service will occur. Were the station to use the simple Smith's rule to minimize the sum of completion times, then your execution could possibly be postponed indefinitely if your recharge time was high enough to surpass all available jobs at the station.

This problem was, to the best of our knowledge, challenged first in [23] where the authors introduced the notion of promptness. This notion captures exactly the need for jobs to know their exact completion time upon arrival. The authors managed this by creating an adaptive schedule of slots of various lengths. When a new job arrives, it can pick one slot which suffices for its length. Naturally, a selfish agent will pick the slot that minimizes its completion time. If it were to pick a slot of smaller length then its execution would end abruptly, before it can finish. Using this adaptive schedule, the authors managed to prove a competitive ratio of $O(\log L_{max})$, where L_{max} is the length of the largest job that appears in the input. Note that this L_{max} is not known to the algorithm in advance.

4.1 A Definition of Promptness

We shall generalize and formalize promptness guarantees of algorithms. In [23], the jobs scheduled know their exact completion time on arrival. This constraint can be relaxed to allow a level of uncertainty: With maximum uncertainty, the algorithm behaves exactly as a classical algorithm as jobs are not at all aware of when their execution will occur. Conversely, with minimum uncertainty, the jobs know exactly their scheduling time. We formalize this in the following way: For a job j that arrives at our system, an algorithm is *semi-prompt* if it promises the job that its completion time will lie in a specific interval with endpoints s_j and e_j. Symbolically, the algorithm is semi-prompt if, on the arrival of job j, it determines s_j, e_j such that $\forall j\ c_j \in [s_j, e_j]$. We define as *slackness* λ_j the ratio of

the two endpoints of the interval, meaning $\lambda_j = \frac{e_j}{s_j}$. Furthermore, we define as the *overall slackness* λ of the algorithm the maximum of all the individual λ_j's: $\lambda = sup\{\lambda_j : j \in [n]\}$, where $[n] = \{1, 2, \ldots, n\}$ and n being the number of jobs that arrive.

Definition 4.1 (Semi-Promptness)**.** A scheduling algorithm is *semi-prompt* if, for every job j, it computes s_j, e_j on the arrival of job j such that $c_j \in [s_j, e_j]$, where c_j is the completion time of job j. The value $\lambda_j = \frac{e_j}{s_j}$ is called the *slackness* of job j and the *overall slackness* λ of the algorithm is the supremum over all λ_j.

Obviously, if an algorithm is to announce the exact completion time to a job on its arrival then $s_j = e_j$ for that job. In this case, the slackness is exactly 1 and the algorithm is called *fully-prompt*.

Definition 4.2 (Full-Promptness)**.** A semi-prompt scheduling algorithm is *fully-prompt* if $s_j = e_j$ for every job j, i.e., it has an overall slackness $\lambda = 1$.

To better understand the concept of promptness, let us describe an example in the TRP setting. In the TRP, jobs arive in an online manner at points in a metric space. A job is served when a server moves to its location. The server begins at a special point called the origin and moves at unit speed. Let us say that, at time t=0, the server is located at the origin. Furthermore, a job j arrives far away from the origin, at distance Δ. The server is presented with a problem: go serve the job immediately and possibly delay future jobs that arrive close to the origin or wait at the origin to accommodate future jobs that arrive there. Either way, if the algorithm is semi-prompt or fully-prompt, it needs to give job j a guarantee on its completion time. In chapter 7, we shall present two algorithms. One is semi-prompt with large slackness. This algorithm tells job j that it might be serviced near optimally at time $t = \Theta(\Delta)$ or at the latest possible time, after it has serviced all other jobs at time $t = \Theta(TSP_{tour})$, where TSP_{tour} is a TSP path for the metric space. Thus, we have $s_j = \Theta(\Delta), e_j = \Theta(TSP_{tour})$ and the slackness is $\lambda_j = \Theta(\frac{TSP_{tour}\Delta}{)}$. On the other hand, we describe a fully-prompt algorithm that services j at an exact time. This algorithm follows an fixed path on the metric space and thus its slackness is $\lambda_j = 1$.

In the parallel machine setting, the problem is similar. We need to inform a job bout its approximate completion time on arrival. However, in contrast to the TRP on bounded metrics, a large job can be postponed indefinitely if smaller jobs keep coming. This is the nature of SRPT after all. This is remedied by fixing a sequence of job lengths that can be scheduled, thus achieving slackness $\lambda = 1$ and competitive ratio $O(L_{max})$, where L_{max} is the largest possible job length. The sequence is explained in the next section.

4.2 The Prompt Sequence

The first attempt at a prompt algorithm was made by Eden et al. in [23]. In their work, they provide an $O(\log L_{max})$-competitive algorithm. This algorithm is based on an certain sequence of slots, which creates a menu of possible options, that is presented to the jobs upon arrival. The jobs, as strategic agents, choose the slot that minimizes their completion time and as such, maximizes their utility. Since the jobs choose exactly their slot in the menu, the algorithm is fully-prompt.

To create the menu of options, Eden et al. create a series of slots to accommodate all possible lengths of jobs. In fact, they create a series of exponentially increasing sizes of slots. The slots have a length of powers of 2, but this does not hinder the mechanism, as

jobs can just choose their best fit. This is without loss of generality, as a general input can only have a better competitive ratio than one with just powers of 2.

The exponential series, as we call them, work as follows: Let $S_0 = \langle 1 \rangle$ and $S_k = S_{k-1}||S_{k-1}||2^k$ for each $k > 0$, where $||$ is the concatenation of two sequences. This gives the following sequences:

$$
\begin{aligned}
S_0 &= \langle 1 \rangle \\
S_1 &= S_0||S_0||2^1 = \langle 1, 1, 2 \rangle \\
S_2 &= S_1||S_1||2^2 = \langle 1, 1, 2, 1, 1, 2, 4 \rangle \\
S_3 &= S_2||S_2||2^3 = \langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8 \rangle \\
&\dots
\end{aligned}
$$

Furthermore, we define an infinite sequence that has, as a prefix, every S_k:

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

This sequence might seem to appear out of the blue, but its has a clear reasoning behind it. Suppose we have $K + 1$ machines instead of 1. This means we can assign a job size on each machine and have a super-optimal schedule (compared to the one with only 1 machine). This schedule would look as follows:

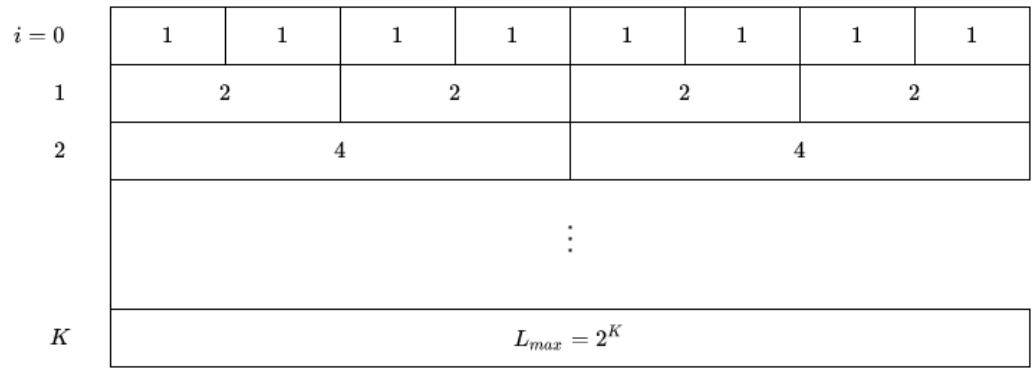


Figure 4.1: Interleaving machine example

If we were to "push" all of these jobs on one machine, then we would get the sequence S_K. This is because we would like to have more small jobs in the beginning of the schedule and leave the large ones for the end. In S_K, all job lengths have equal volume 2^K. This is because we allocate equal volume for each job length. From the above example, the upper bound of the sequence is trivially $O(L_{max}) = O(K)$, because we shrink $K + 1$ machines into only 1. Now let us prove this bound formally.

Obviously, sequence S_k has a length of $2^{k+1} - 1$. The first $2^{k+1} - 1$ elements of S_∞ are exactly S_k. The algorithm in [23] adopts the fully online setting and thus does not know the maximum input length L_{max}. A competitive ratio of $O(\log L_{max})$ is achieved by adapting the exponential sequence for every new job that arrives. Moreover, this result is optimal as any fully-prompt algorithm is $\Omega(\log L_{max})$. If one where to use the infinite, static sequence instead, then the authors prove a lower bound of $\Omega(\sqrt{L_{max}})$. If we allow the competitive ratio to be a function of the number of jobs as well as the maximum

length, then the competitive ratio improves to $O(\log L_{max} + \log n)$ for the static sequence, where n is the number of jobs. In our setting, we assume a relaxation on the online model and assume we know an upper bound on the maximum length a job can have. With L_{max} considered known, we present a $O(\log L_{max})$-competitive algorithm, that repeats the sequence $S_{\lceil \log L_{max} \rceil}$. To prove the above, we will use two lemmas from [23]:

Lemma 4.1. For all $d \geq 0$, for every $0 \leq k \leq d$, the sum of all 2^k value items in S_d is 2^d:

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

where $n_d = 2^{d+1} - 1$ is the length of sequence S_d and $S_d[i]$ is the i-th element of S_d.

Proof. We shall prove this with induction over d. For S_0 the only value $2^0 = 1$ sums to $2^0 = 1$ and therefore the claim us true. Assume that the claim is true for S_{d-1}, meaning:

$$\sum_{1 \leq i \leq n_{d-1} : S_{d-1}[i] = 2^k} 2^k = 2^{d-1}$$

for all $0 \leq k \leq d-1$. Since $S_d = S_{d-1} || S_{d-1} || 2^d$, the total length of 2^k elements is doubles because we have two S_{d-1} sequences:

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

for all $0 \leq k \leq d-1$. Furthermore, the claim is true for $k = d$ since we only have one item of length 2^d. □

Before we state and prove the second lemma, we introduce some notation. Let γ_i be the sum of the first i entries in S_∞, i.e., $\gamma_i = \sum_{j=1}^{i} S_\infty[j]$. Furthermore, we define $S_k(t)$, $t \geq 0$, to be the sequence of n_k consecutive intervals that are created by the elements of S_k. Meaning, we interpret S_k as a sequence of the length of intervals and $S_k(t)$ the sequence of those intervals starting at t:

$$S_k(t) = \langle [t, t+\gamma_1], [t+\gamma_1, t+\gamma_2], ..., [t+\gamma_{n_k-1}, t+\gamma_{n_k}] \rangle$$

For example, $S_2(3)$ is:

$$S_2(3) = \langle [3,4], [4,5], [5,7], [7,8], [8,9], [9,11], [11,15] \rangle$$

In addition, if j is the index of a job that arrives at our system, we define $D(j) = \{j' \leq j : l_{j'} \leq l_j\}$, meaning the jobs that arrived before j and have length less than it. Finally, $I(j)$ is the interval job j occupies, $M(j)$ is the machine job j executes on and with $b(S_k(t)), e(S_k(t))$ we denote the endpoints of the first and last interval in $S_k(t)$ respectively.

Lemma 4.2. Let $d \geq 0$. Let t and q be such that job j, with $r_j \leq t$, $l_j = 2^k$, chose the last interval in $S_d(t)$ 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 (WSRPT without weights), that execute on machine q and occupy some interval in $S_d(t)$. More concisely, $D(j, q, S_d(t)) = D(j) \bigcap \{j' : I(j') \in S_d(t), M(j') = q\}$. Then:

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

where $vol(D(j, q, S_d(t)))$ denotes the total length of all intervals of jobs in $D(j, q, S_d(t))$.

Proof. If $l_j = 2^k = 2^d$ then $j \in D(j, q, S_d(t))$ and the claim is true. Also, the claim is true for $d = 0$. We will prove the case $l_j = 2^k < 2^d$ with induction over d. Let $d > 0$ and assume the claim is true for all $0 \leq d' < d$. If $k < d$ then all intervals of size 2^k are occupied because job j would have preferred those. But these intervals are the last interval of some subsequence of $S_d(t)$ that ends in them and are occupied by a job $j' < j$. Denote these intervals by $D(j', q, S_k(t'))$ for some t'. From induction hypothesis we have that $vol(D(j', q, S_k(t'))) \geq 2^k$. Since we have 2^{d-k} sequences S_k in $S_d(t)$ the total volume of occupied intervals is $vol(D(j, q, S_d(t))) \geq 2^k \cdot 2^{d-k} = 2^d$, which ends the proof. □

As we mentioned before, an upper bound of $O(\log L_{max} + \log n) = O(K + \log n)$ is proven for the case of the static sequence $S_\infty(0)$ in [23]. W.l.o.g, we assume only lengths that are powers of 2. Furthermore, by assuming the largest possible length of a job is at most 2^K and known in advance, we are able to improve this to $O(K) = O(L_{max})$ by modifying the sequence, dropping the dependence on the number of jobs.

Theorem 4.1. For a known upper bound L_{max} on the length of incoming jobs, a repetition of sequences $S_{\lceil \log L_{max} \rceil}$ yields an $O(\log L_{max})$-competitive algorithm.

Proof. First, we modify the sequence to take into account the assumption we made about the maximum possible length. Let us denote the largest possible length of a job by 2^K, where we assume w.l.o.g. that all jobs have lengths of powers of 2. We define the new static sequence as:

$$G_1 \,||\, G_2 \,||\, G_3 \,||\, \dots$$

where $G_i = S_K(e(G_{i-1}))$ and G_0 is the empty sequence with $e(G_0) = 0$. Simply, the sequence we construct is a concatenation of exponential sequences that go up to 2^K. For every job j there exist x, l such that:

$$e(G_x) < r_j + l_j \leq e(G_{x+1})$$
$$e(G_{x+l}) < c_j \leq e(G_{x+l+1})$$

Since job j chooses an interval at sequence G_{x+l} and job j can fit in the last interval of every sequence G_i (because $l_j \leq 2^K$) then every interval of size 2^K must be taken for sequences G_{x+1}, G_{x+2}, $\dots$, G_{x+l}. By applying 4.2 to each one of these G_i separately we have that:

$$vol(D(j, \bigcup_{i=x+1}^{x+l} G_i, q)) \geq l \cdot 2^K$$

for every machine q. Subsequently, the total volume of jobs on all machines in the above sequences must be at least $m \cdot l \cdot 2^K$. Combining the above result with 4.1 we have:

$$c_j^* \geq l \cdot 2^K + r_j + l_j \geq l \cdot 2^K + e(G_x) \geq l \cdot 2^K + x \cdot (K + 1) \cdot 2^K$$

Furthermore from our initial assumption for the value of c_j and 4.1 we get:

$$c_j \leq e(G_{x+l+1}) \leq (x + l + 1) \cdot 2^K \cdot (K + 1)$$

Thus:

$$\begin{aligned} c_j &\leq \frac{(x+l+1)\cdot(K+1)}{l+x\cdot(K+1)}\cdot c_j^* \\ &= \frac{l+x\cdot(K+1)+l\cdot K+K+1}{l+x\cdot(K+1)}\cdot c_j^* \\ &= \left(1+\frac{l\cdot K+K+1}{l+x\cdot(K+1)}\right)\cdot c_j^* \\ &\leq \left(1+\frac{l\cdot K+K+1}{l}\right)\cdot c_j^* \\ &= O(K)\cdot c_j^* \end{aligned}$$

for $l \geq 1$. For $l = 0$:

$$\begin{aligned} c_j^* &\geq r_j + l_j \geq e(G_x) = x\cdot K\cdot 2^K \\ c_j &\leq e(G_{x+1}) = (x+1)\cdot K\cdot 2^K \end{aligned}$$

and we have a constant approximation. Lastly, for $x = 0, l = 0$, the job lies in sequence G_1. If a slot is the last slot in interval $S_k(0)$ for some k, we call it a "last slot". Let the last slot of $S_k(0)$ be the first "last slot" to the left of job j. Then, from the nature of $S_\infty(0)$, we have that $c_j \leq 2(k+1)2^k$. Moreover, from lemma 4.2, we have that $c_j^* \geq 2^k$. Thus, $c_j \leq 2(k+1)c_j^* \leq 2(K+1)c_j^* = O(K)c_j^*$ and the algorithm is $O(K)$-competitive in all cases. □

For example, the sequence with $L_{max} = 2^K = 4$ would be like follows:

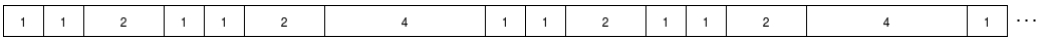


Figure 4.2: Slot sequence for $K = 2$

Jobs arrive and choose the first interval in the sequence that fits them and is not taken by another job. This means they can also choose a bigger interval than their length, but not a smaller one, since they will not be able to finish execution.

Chapter 5

Mechanism Design and Promptness

Throughout this thesis, we consider jobs arriving as strategic agents, meaning the jobs have a willingness to lie in order to gain more. For example, we can think of a job in the job scheduling model as an agent with a private length l_j, that only it knows. If the job can achieve a smaller completion time by announcing a different length to the mechanism than it is happy to do so. This can be extended to jobs with both private lengths l_j and weights w_j. To roughly see how this relates to promptness, observe that if a job has a fixed completion time and cannot alter it by lying then it has no incentive to declare a spurious length.

We shall begin this chapter by introducing the basic definitions of *Mechanism Design* in the general setting and, afterwards, bridge the gap between the general setting and ours. Furthermore, we will provide a reasoning for the connection between promptness and truthfulness.

5.1 Preliminaries

Definition 5.1 (The basic setup)**.** Assume n jobs (strategic agents) arrive at our system in an online fashion and a set of possible outcomes $\mathcal{O}$. Each job has a valuation function $v_j : \mathcal{O} \to \mathbb{R}^+$, which is unknown to us and a set of actions $\mathcal{A}_j$. Each job submits an action $a_j \in \mathcal{A}_j$ to the mechanism upon arrival. Also, let $\mathbf{a} = (a_1, a_2, ..., a_n)$ the vector of submitted actions. The mechanism uses a function $f : (\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n) \to \mathcal{O}$ to map the information declared by the jobs to a schedule. The mechanism also uses a pricing policy, which is a vector of pricing functions $\pi_j : (\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n) \to \mathbb{R}^+$. Finally, each job has a utility function u_j, which depends on its valuation function v_j, the outcome of the mechanism and the pricing function π_j. The pair (f, π) defines a mechanism.

This setting is as general as possible and allows us to describe a scheduling scenarios where jobs can be modeled as strategic agents. We consider direct revelation mechanisms, where a job simply declares its type on arrival. By type, we refer to the parameters that exactly specify a job. For example, in the job scheduling setting, a job's type is its arrival time, its length and its weight. Symbolically, $a_j = (r_j, l_j, w_J)$. In the game theoretic setting, a job can declare a false type to the mechanism in order to achieve greater utility.

We assume that a job cannot create a phantom presence, i.e., it cannot present itself to the mechanism prior to its arrival. Hence, a job will always declare its true arrival time r_j, since our algorithms do not benefit a job that is delayed. Moreover, in the TRP setting, a job's type is its position in the metric space and its weight, if we consider the weighted TRP. If a job could create an earlier presence in the TRP, it could then achieve a better completion time as the server would begin towards its position earlier.

For the rest of this thesis, we will solely consider a special type of utility functions called quasi-linear. A quasi-linear utility is the valuation of the job given the outcome minus the price charged by the mechanism. A rigorous definition is given below.

Definition 5.2 (Quasi-Linear Utility Function)**.** We say that a job j has *quasi-linear utility* if:

$$u_j = v_j(f(\mathbf{a})) - \pi_j(\mathbf{a})$$

Of course, jobs are interested to maximize their utility. If the other players were to have fixed strategies (types), then job j would like to declare such a type, as to maximize its utility u_j. Such a type would be called a dominant strategy:

Definition 5.3 (Dominant Strategy)**.** Given type a_j of job j, let the rest of the jobs have fixed declared types $\mathbf{a}_{-j}$. We say that type a_j is a *dominant strategy* for job j if:

$$v_j(f(a_j, \mathbf{a}_{-j})) - \pi_j(a_j, \mathbf{a}_{-j}) \geq v_j(f(a'_j, \mathbf{a}_{-j})) - \pi_j(a'_j, \mathbf{a}_{-j}),\ \forall a'_j \in \mathcal{A}_j$$

assuming quasi-linear utilities.

We want for the jobs to gain the most by declaring their true types. This allows the mechanism to make correct decisions as it knows the true values of the problem. A mechanism such that truth telling is a dominant strategy for all jobs is called truthful:

Definition 5.4 (Truthfulness)**.** Let a_j be the true type of job j. A mechanism (f, π) is *truthful* if for every job, declaring their true type a_j is a dominant strategy, i.e., for every declared type a'_j and every vector of declared types $\mathbf{a}_{-j}$:

$$v_j(f(a_j, \mathbf{a}_{-j})) - \pi_j(a_j, \mathbf{a}_{-j}) \geq v_j(f(a'_j, \mathbf{a}_{-j})) - \pi_j(a'_j, \mathbf{a}_{-j}),\ \forall a'_j \in \mathcal{A}_j$$

assuming quasi-linear utilities.

The property of truthfulness is fundamental in mechanism design. It alleviates the participating strategic agents from the burden of having to compute a dominant strategy since telling the truth yields the most utility for the agents. Furthermore, it allows the mechanism to make the correct decisions, not having to worry about possible spurious declarations by the agents.

Another property that might be desired is that an agent should not be damaged by participating in the mechanism, i.e., having negative utility. Mechanisms that guarantee non negative utility are called individually rational:

Definition 5.5 (Individually Rational)**.** A mechanism (f, π) is *individually rational* if for every job, every declared type a_j and every vector of declared types $\mathbf{a}_{-j}$ the utility is non negative:

$$v_j(f(a_j, \mathbf{a}_{-j})) - \pi_j(a_j, \mathbf{a}_{-j}) \geq 0,\ \forall a_j \in \mathcal{A}_j$$

assuming quasi-linear utilities.

A mechanism that possesses both the properties of truthfulness and individual rationality is called Dominant Strategy Incentive Compatible (DSIC):

Definition 5.6 (DSIC)**.** A mechanism (f, π) is *dominant strategy incentive compatible*, or DSIC, if it is both truthful and individually rational.

Mechanisms that we design and analyze might not always be individually rational. For example, one might choose to model the valuation from having a completion time of c_j as $v_j = -c_j$. The larger the completion time, the lesser the utility. Thus, a mechanism that maximizes social welfare, minimizes the sum of completion times. We will see later in this chapter, that this model prevents us from using the VCG mechanism (to be discussed later). Finally, we define notions that allow us to measure the performance of mechanisms. These notions gauge the efficiency of the mechanism based on the value that each individual agent received.

Definition 5.7 (Social Welfare)**.** Let (f, π) be a mechanism and let $\mathbf{a}$ be the vector of declared types. We define as *social welfare* the cumulative value that all agents gain:

$$SW = \sum_{j=1}^{n} v_j(f(\mathbf{a}))$$

We also want to design mechanisms that raise a lot of revenue. The revenue is the sum of all prices paid by the agents:

Definition 5.8 (Revenue)**.** Let (f, π) be a mechanism and let $\mathbf{a}$ be the vector of declared types. We define as *social welfare* the cumulative value that all agents gain:

$$Rev = \sum_{j=1}^{n} \pi_j(\mathbf{a})$$

Though important, our main concern in this thesis will be the maximization of social welfare rather than revenue. As we shall see, we will present such a model that social welfare will coincide with the objective of maximizing weighted completion time. Now, let us introduce the VCG mechanism and its application in auction theory.

5.2 The VCG mechanism

A silent point in the above presentation of mechanism design basics, was *what* do the agents (jobs) declare. In a single item auction, bidders bid their values for a single, indivisible good, that is awarded to only one participant. Thus, in a single item auction, bidders ought to bid their valuations for the item. A truthful auction that achieves maximization of social welfare is the Vickrey auction [56], or second-price auction. The auction is called this because it charges the winning bidder (the one with the greatest bid) the price of the second highest bid. It can be proven that this results in a truthful auction.

If more goods where to be auctioned, then we would have a combinatorial auction [47]. In a combinatorial auction, there exist many, distinct, goods, each to be awarded to a certain bidder. Another way to look at this, is that each bidder gets a specific subset

of the items and each of those subsets does not intersect with others. In such a setting, a bidder can have a different valuation for each one of the subsets, which are 2^n for n distinct items. A breakthrough, truthful mechanism, that achieves optimal social welfare, is the Vickrey–Clarke–Groves auction, or VCG [20, 33, 56]. In this section, we will explain the inner workings of the VCG mechanism. In later sections we shall make the connection between job scheduling and mechanism design and explain why VCG is not applicable.

Theorem 5.1 (Vickrey–Clarke–Groves mechanism)**.** In every general mechanism design environment, there is a DSIC welfare-maximizing mechanism.

Proof. Now let us specify the inner workings of the mechanism and prove its guarantee. Since the mechanism is used in the auction setting we discussed before, each agent (now bidder), submits a function $b_j : \mathcal{O} \to \mathbb{R}^+$ describing its preference of items. Again, the complete vector of bids is denoted by $\mathbf{b}$. Furthermore, the set of outcomes is an allocation of the items to be auctioned, where each item is given to exactly one or zero bidders. The allocation of the mechanism is computed to maximize the social welfare:

$$f(\mathbf{b}) = arg\, max_{\omega \in \mathcal{O}} \sum_{j=1}^{n} b_j(\omega)$$

The outcome of the mechanism needs be optimal since that is the guarantee of theorem 5.1. We use the bids as substitutes for the true (unknown) valuation functions. This will prove to be not important, as the mechanism will be proven truthful. Hence, the bids will coincide with the true valuations for strategic agents.

The next part of the mechanism to be determined is the payment of each bidder. As payments, the mechanism charges an agent its "externality", i.e., the impact his allocation had on the welfare of other agents. Simply put, the externality shows us how the presence of a player reduced the welfare other players would have gotten on his/her absence. Thus, the payment rule is:

$$\pi_j(\mathbf{b}) = max_{\omega \in \mathcal{O}} \sum_{i \neq j} b_i(\omega) - \sum_{i \neq j} b_i(\omega^*)$$

where ω^* is the optimal outcome of the VCG mechanism. The payments are always non-negative since $\omega^* \in \mathcal{O}$. We will prove that the mechanism is DSIC. To show this, we analyze the utility of player j:

$$v_j(\omega^*) - \pi_j(\mathbf{b}) = v_j(\omega^*) + \sum_{i \neq j} b_i(\omega^*) - max_{\omega \in \mathcal{O}} \sum_{i \neq j} b_i(\omega)$$

The last term of the second hand of the equation is not dependent on j's bid and can be ignored. Thus, the problem of maximizing utility comes down to maximizing the first two terms $v_j(\omega^*) + \sum_{i \neq j} b_i(\omega^*)$. we observe that these two terms are maximized for $v_j = b_j$, since for $v_j = b_j$ the quantity $v_j(\omega^*) + \sum_{i \neq j} b_i(\omega^*) = \sum_{i=1}^{n} v_i(\omega^*)$ becomes exactly the social welfare, which the mechanism wants to maximize. Subsequently, bidding truthfully results in the mechanism choosing an outcome that maximizes agent j's utility. This concludes the proof of theorem 5.1. □

5.3 Mechanism Design and Completion Time

We are to give a model for the strategic agents (jobs) that arrive at our system. In order to specify the model, we need to describe the actions that jobs declare to the mechanism while also describing their valuation function. The utility is considered to be quasi-linear as stated before. In both settings, the job scheduling and the TRP, jobs do not decalre actions to the mechanism, rather their parameters that describe them. We call the tuple of these parameter the type of the job.

For the job scheduling setting, the jobs' types are composed of their arrival time r_j, their weight w_j and their length l_j. In the unweighted case, the weights are omitted. We assume that jobs cannot misrepresent their arrival times and that they cannot create a phantom presence, i.e., make the mechanism think that they have arrived before they actually did. We indicate the type of a job as a tuple $a_j = (r_j, w_j, l_j)$. When we make a distinction between the declared type and the actual type of the job, we denote the declared types with the prime symbol. For example, the declared weight of job j would be w'_j, the declared type a'_j an so on. We assume that the valuation function of a job can take two forms: weighted and unweighted. The weighted valuation of job j is $v_j = -w_j c_j$, while he unweighted is simply $v_j = -c_j$. This is not to be confused with the weighted version of the problems; in fact we could use the unweighted valuation function even when the jobs have weights to declare, although we abstain from such practises to have a meaningful relation between the objective of the scheduling problem and the social welfare. To be precise, the minimization of the weighted completion times is the maximization of social welfare with weighted valuation functions, accordingly for the unweighted case. Unless stated otherwise, we are to assume valuations that coincide with the problem's objective (weighted or unweighted).

In the TRP setting, the jobs declare their weight and their arrival time as above and also their position. We assume that jobs cannot create a phantom presence in another position. Again, the valuation function is $v_j = -w_j c_j$ in the weighted case and $v_j = -c_j$ in the unweighted case.

In general, one of the primary goals of mechanism design is the maximization of social welfare. Since the social welfare is exactly the minimization of (weighted) completion times, our algorithms are able to kill two birds with one stone. Both the central objective of scheduling and mechanism design are satisfied concurrently. However, there exist limitations when designing mechanisms due to the complex nature of the utility function. Even the powerful VCG mechanism cannot be applied to job scheduling as we shall see.

This is not a novel restriction. Truthfulness can often impose a constraint on our ability to achieve optimal social welfare. This is the case in the Nisan-Ronen conjecture [42, 43]. Nisan and Ronen concern themselves with the minimization of makespan, i.e., the largest completion time in a schedule, in the case of unrelated machines. Specifically, assume n tasks (jobs) and m machines, where each machine i reports time t_i^j for task j. The Nisan-Ronen conjecture states that no truthful mechanism can achieve an approximation better than m on the makespan. Nisan and Ronen applied the VCG mechanism [20, 33, 56] to the problem, and achieved an approximation of m. They then boldly conjectured that this approximation is the best possible for truthful mechanisms. This conjecture is yet unproven. Some work has been done; Nisan and Ronen themselves proved an impossibility of $1+\sqrt{2}$. This lower bound was gradually improved to 3 by a series of papers [19, 22, 26, 36]. The first super-constant lower bound was given by Christodoulou et al. [18] who proved a lower bound of $1+\sqrt{m-1}$. This shows us that truthfulness can indeed obstruct our ability to approximate well the optimum social welfare, as we explain further in the following

section.

5.4 Futility of using the VCG mechanism

In [3] it was shown that the VCG method cannot be applied to job scheduling without preemtpion of the jobs.

Theorem 5.2. There is no optimal truthful mechanism with payments for the single machine case even in the unweighted case.

Proof. Assume, by contradiction, that there exists an optimal truthful mechanism minimizing the sum of completion times of the tasks on a single machine. Also, assume two jobs that arrive at time zero. The optimal social welfare is achieved by scheduling the shortest job first [55]. There are exactly two ways to schedule the tasks; since no preemption is allowed, the scheduling is a simple ordering of the tasks. First, we examine the following scenario: job 2 tells the truth and declares $l_2 = b_2 > b_1$, where b_j is the declared length of job j for simplicity. The mechanism will schedule job 1 before 2 with job 2 having utility $u_2 = -c_2 - p_2 = -l_1 - l_2 - p_2$. On the other hand, if job 2 lies and declares $b'_2 < b_1$ then it is scheduled first and gains utility $u'_2 = -c'_2 - p'_2 = -l_2 - p'_2$. For the mechanism to be truthful it is needed that $u'_2 \leq u_2 \Rightarrow -l_2 - p'_2 \leq -l_1 - l_2 - p_2 \Rightarrow l_1 \leq p'_2 - p_2$. However, l_1 is not known to the mechanism when the payments are computed and no payment can satisfy the above constraint. □

Corollary 5.1. The VCG method cannot be applied for the single-machine case.

To understand the above restriction, we need to think what the valuation of each job is. In the classic VCG setting, the jobs are expected to declare their value for each possible outcome of the mechanism. Even in the case with only two jobs, the jobs are not aware of their valuation functions. That is, they know the nature of these functions but not their exact values. If job 1 is scheduled first, its valuation is $-l_1$. Although, if the job is scheduled after job 2, its value is $-l_1 - l_2$. Hence, job 1 cannot know its exact valuation function, as it needs to know the length of job 2 beforehand.

5.5 Promptness and Truthfulness

One might ask what is the actual purpose of promptness. Sure, promptness can help those impatient to await for their time for scheduling, but this does not help the overall sum of completion times. As we saw, there exist constant approximation online algorithms that help reach near optimality regarding the sum of completion times. Even from a social welfare point of view, these algorithms should be better, as any fully-prompt algorithm for job scheduling has a competitive ratio of at least $\Omega(\log L_{max})$. So why sacrifice the competitive ratio and in turn the social welfare to give some large, impatient jobs a guarantee? As we proved above, no socially optimal and truthful mechanism can exist. So we cannot use VCG and we need another way to design truthful mechanisms.

Remind yourself of the sequence we shown in 4.2: The sequence is fixed and only depends on the maximum length, which we assume we know beforehand. We construct the sequence before the actual jobs arrive. This makes the sequence set in stone; no declaration of weight or length by the jobs will force us to change the way the sequence

is constructed. This means that the algorithm is trivially truthful. Truthfulness arose from the fixed nature of the sequence. Thus, promptness gives fixed execution paradigms, which in turn give truthfulness. This is a marvellous property of promptness that enables us to design truthful algorithms with ease.

Furthermore, in the TRP setting we will bolster this claim, as again our near optimal algorithm will be based on a fixed traversal of the metric space. This fixed traversal makes the false declaration of weights or positions meaningless from the side of the jobs. Thus, again the fixed schedule gives rise to truthfulness.

Both fully-prompt approaches, the fixed traversal of the tree in the TRP setting and the menu based mechanism of [23], have something in common; they do not require any effort on the part of the jobs to compute a strategy. While many mechanisms can be DSIC, not all of them are as easily understood by the agents participating. For example, choosing when to quit in an ascending clock auction might be equivalent to the Vickrey second price auction with sealed bids [56], but experiments showed players are more likely to play the dominant strategy under the clock auction rather than the sealed bids [34]. This was modelled by Li [39] in the sense of obviously strategy-proof mechanisms. Our fully-prompt mechanisms follow a similar idea. Since the path of the server is fixed in TRP and in machine scheduling the job chooses the best slot for it on its own, there is no calculation that requires a significant cognitive cost; the agents simply tell the truth because their bid does not affect the outcome. This is an advantage of promptness, as its fixed/strict nature designs mechanisms with predetermined schedules and thus gives rise to truthfulness that is obvious, both to prove and for the agent who participates.

Another parallel to known ideas of Mechanism Design is to posted prices. A posted price mechanism presents each agent with a (possibly different) price for each outcome, and the agent can either accept or reject the mechanism's offer. This is similar to the menus that Fledman et al. [23] offers to jobs. Its offer is a time slot, out of which the agents chooses the one that maximizes their utility, i.e., the one which results in the minimum completion time. Again, we see why these type of mechanisms are obviously strategy proof. An agent does not need to think very hard; they must simply choose the interval (or set of items for the posted prices case) that maximizes their utility.

Chapter 6

Promptness and Truthfulness in Machine Scheduling

In this chapter, we will present machine scheduling algorithms for various settings, which consist of relaxed versions of the online setting. These settings will be based on a model which assumes an adversarially chosen number of jobs each second, that follow a multinomial distribution. Specifically, for the T first discrete time points since time zero, a number of n_t jobs arrive each second. These n_t jobs (for time t) follow a multinomial distribution with lengths of powers of 2. While this model might seem unnatural, it can capture many real life scenarios without infringing on the generality of the online setting too much. For example, assume an electric car recharge station. A number of n_t cars arrive at each time point t and follow a certain distribution which is known in advance. This is not an illogical assumption; the recharge station might have observed that such a distribution does in fact describe the jobs arriving, while the adversarially chosen n_t do not constrict the total number of jobs that are expected to arrive.

Algorithms in this chapter will mainly focus on this model and variations thereof. First, we give an asymptotically optimal algorithm for the model we discussed above. This algorithm is based on Smith's Rule [55] and has a competitive ratio of $1 + O(\frac{L_{max}}{n_{total}})$ for the case of one machine, where $n_{total} = \sum_{i=0}^{T-1} n_t$ is the total number of jobs across all time points. Furthermore, we should note that this algorithm is not at all prompt. In fact, slackness is infinite, meaning a job has no upper bound on its completion time upon arrival. We reconcile this by reasoning that an $O(1)$-competitive algorithm in this model cannot achieve full promptness. Continuing, we modify the main model by assuming that the jobs arriving each discrete time point are constant and known in advance, i.e., $n_t = n$ for all t and provide an $1 + O(\frac{1}{\sqrt{n}} \max_i \frac{\sqrt{1-p_i}}{\sqrt{p_i}})$-competitive algorithm in the case of m machines, for $m \geq n\mathbb{E}[L]$, where L is the random variable that gives the length of a job. Consequently, we give upper bounds for generalizations of the above setting with many machines, which include n being a random variable and having periodicity in the number of jobs arriving.

Moreover, we provide a truthful version of the constant approximation algorithm in Phillips et al. [46] based on techniques of [3]. Finally, we present a convex combination of our first, asymptotically optimal algorithm and the exponential sequence in 4.1. This results in a semi-prompt algorithm which achieves an asymptotically optimal competitive

ratio for large values of slackness.

6.1 The Main Model

At discrete time points $t = 0, 1, ..., T-1$ the number of jobs to arrive at our system at time t, n_t, is adversarially chosen. Each batch of jobs arriving must follow a multinomial distribution defined as follows:

$$L = \begin{cases} 2^0 & \text{with prob. } p_0 \\ 2^1 & \text{with prob. } p_1 \\ 2^2 & \text{with prob. } p_2 \\ & \vdots \\ 2^K & \text{with prob. } p_K \end{cases}$$

where L is the length of every job arriving at a certain time and K is such that the maximum allowed job length is 2^K. Furthermore, we assume that these lengths are i.i.d, thus a multinomial distribution is formed for each batch of jobs n_t. The model does not assume the number of machines and we shall present results for both $m = 1$ and $m > 1$, where m denotes the number of machines. Furthermore, we denote by $[K]$ the set $\{0, 1, ..., K-1\}$.

We observe that again we only assume lengths of powers of 2. This is without loss of generality, as we can assume the general model, augment its job's length to the closest power of 2 and get an at most 2 increase in competitive ratio. Note that, because of the way our algorithms are constructed, this multiplicative factor only affects the asymptotically zero part of the competitive ratio.

6.2 An Asymptotically Optimal Algorithm for the Main Model

In this section, we provide an asymptotically optimal algorithm for the model of 6.1. Note that the algorithm which we will describe is not prompt. First, we provide a reasoning for our dismissal of promptness by proving that any $O(1)$-competitive algorithm cannot be fully-prompt, i.e., specify exactly a job's completion time on arrival.

Lemma 6.1. An $O(1)$-competitive algorithm for the main model 6.1 cannot be fully prompt.

Proof. Let as assume an input of the following form: $n_0 = Kn$ and each job length is equiprobable. This means that, in expectation, the number of jobs of length 2^i arriving at time $t = 0$ is n $\forall i \in [K+1]$. If a mechanism is prompt, it must decide on intervals immediately for the jobs arriving at $t = 0$. Since we aim to achieve a constant competitive ratio, we should schedule these jobs in ascending job length. Thus, we should create a menu that leaves enough slots for each job of n_0 and it could possibly leave space for future jobs. These future jobs should also be scheduled in ascending length. Subsequently, the menu will contain $n+n'$ slots for each job length in ascending order, where the first n slots of each length are filled by jobs of n_0 and the rest are reserved for future jobs (of the same length). An adversary, then, can choose to send $n_1 = K(n+2n')$ to our system, resulting

in a $\Theta(K)$ competitive ratio. Note that, the values of n_0 and n_1 are scaled according to n and n', so the multiplicative factor of K is only a matter of convenience and serves no actual purpose in the proof.

Now, let us prove that this input does in fact result in an $\Theta(K)$ competitive ratio. The total number of jobs is $n_0 + n_1 = 2K(n + n') = 2KN$, where we denote $(n + n')$ by N. Since all jobs are available at time $t = 1$ and there is at least one job of length 1 available at $t = 0$, we do not need to take into account each job's arrival time. The optimal algorithm OPT schedules the $2KN$ jobs in ascending order of lengths:

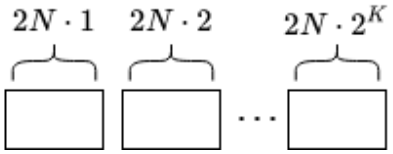


Figure 6.1: Optimal algorithm for the $2KN$ jobs

where $N \cdot l$ denotes N jobs of length l. The sum of completion times for OPT is $O(N^2 2^K)$:

$$\begin{aligned} OPT &= \sum_{i=0}^{K} \frac{2N(2N+1)}{2} 2^i + \sum_{i=1}^{K} 2N(2N)(\sum_{j=0}^{i-1} 2^j) \\ &= \frac{2N(2N+1)}{2}(2^{K+1} - 1) + (4N^2)\sum_{i=1}^{K}(2^i - 1) \\ &= \frac{2N(2N+1)}{2}(2^{K+1} - 1) + (4N^2)(2^{K+1} - 2 - K) \\ &= O(N^2 2^K) \end{aligned}$$

On the other hand, our algorithm schedules jobs as follows:

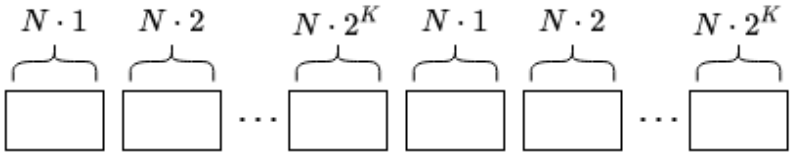


Figure 6.2: Prompt algorithm for the $2KN$ jobs

The right side of the schedule has NK jobs. Thus, the completion time is at least $\Omega((NK)2^K N)) = \Omega(KN^2 2^K))$ and the competitive ratio is at least $\Omega(K)$. □

Since an $O(1)$-competitive prompt algorithm is not possible for our model, we are happy to temporarily abandon promptness in order to achieve a better competitive ratio. The algorithm which we are to describe is based on SRPT with the difference that we do

not preempt jobs. The algorithm, which is based on SRPT, is as follows:

Algorithm 6.2
At each discrete time point, schedule the job with the smallest length among the jobs who are waiting.

Theorem 6.1. Algorithm 6.2 is asymptotically optimal.

Proof. Comparing this schedule to the optimal preemptive schedule, we observe that a job can only be delayed by a maximum of $L_{max} = 2^K$ if a job of such length is scheduled exactly before the new one arrives. Hence, for all $j \in [n] = [\sum_{i=0}^{T-1} n_i]$:

$$\begin{aligned} c_j \leq c_j^* + 2^K \Rightarrow \\ \sum_{j\in[n]} c_j \leq \sum_{j\in[n]} c_j^* + n2^K \end{aligned} \tag{6.1}$$

We then give a lower bound for the optimal preemptive schedule. Let $X_0, X_1, \ldots, X_K$ be random variables denoting the total number of jobs of length $1, 2, \ldots, 2^K$ respectively. Since the jobs arriving are i.i.d, the total number of jobs follows a multinomial distribution with $n = \sum_{i=0}^{T-1} n_i$. We then have:

$$\begin{aligned} \sum_{j\in[n]} c_j^* &\geq \sum_{i\in[K+1]} \frac{X_i(X_i+1)}{2} 2^i \\ &\geq \sum_{i\in[K+1]} \frac{X_i^2}{2} 2^i \end{aligned}$$

Taking expectations on both sides of the inequality we get:

$$\begin{aligned} \mathbf{E}[\sum_{j\in[n]} c_j^*] &\geq \mathbf{E}[\sum_{i\in[K+1]} \frac{X_i^2}{2} 2^i] \\ &= \sum_{i\in[K+1]} \frac{\mathbf{E}[X_i^2]}{2} 2^i \\ &\geq \sum_{i\in[K+1]} \frac{\mathbf{E}[X_i]^2}{2} 2^i \\ &= \frac{n^2 2^K}{2} \sum_{i\in[K+1]} p_i^2 2^{i-K} \end{aligned} \tag{6.2}$$

We note that the term lost in the third line is the variance of each r.v, which is of magnitude O(n) and will not contribute much to the volume of the schedule. From (6.1) and (6.2) we have that:

$$\frac{\mathbf{E}[\sum_{j\in[n]} c_j]}{\mathbf{E}[\sum_{j\in[n]} c_j^*]} \leq 1 + \frac{n2^K}{\frac{n^2 2^K}{2} \sum_{i\in[K+1]} p_i^2 2^{i-K}}$$
$$= 1 + \frac{2}{n \sum_{i\in[K+1]} p_i^2 2^{i-K}} \tag{6.3}$$

Since the probabilities sum to 1 and all probabilities are non-negative, we can bound the term $F(\mathbf{p}) = \sum_{i\in[K+1]} p_i^2 2^{i-K}$ using KKT conditions:

$$\min_{\mathbf{p}\in\mathcal{U}} \sum_{i\in[K+1]} p_i^2 2^{i-K}$$

where $\mathcal{U} = \{\mathbf{p} \in \mathbb{R}^3 : -p_i \leq 0, \sum_{i\in[K+1]} p_i = 1\}$ and $\mathbf{p}$ is the probability vector. The necessary conditions for $\mathbf{p}$ being a minimum are:

$$2p_i 2^{i-K} - \lambda_i + \lambda = 0 \qquad \forall i \in [K+1] \tag{6.4}$$
$$\lambda_i p_i = 0 \qquad \forall i \in [K+1] \tag{6.5}$$
$$\sum_{i\in[K+1]} p_i - 1 = 0 \tag{6.6}$$

where the constraints of (6.5) are derived from the inequalities $-p_i \leq 0$ and the constraint (6.6) is from the definition of our model (probabilities must sum to 1). Solving (6.4) an substituting in (6.5) we get:

$$(2p_i 2^{i-K} + \lambda)p_i = 0 \qquad \forall i \in [K+1] \tag{6.7}$$

From (6.7), if $p_i > 0$ we have that:

$$p_i = -\lambda 2^{K-i-1} \qquad \forall i \in [K+1] \tag{6.8}$$

Subsequently, we can express λ in terms of the probabilities using (6.6):

$$\sum_{i:p_i>0} p_i - 1 = 0 \Leftrightarrow$$
$$-\sum_{i:p_i>0} \lambda 2^{K-i-1} = 1 \Leftrightarrow$$
$$\lambda = -\frac{1}{\sum_{i:p_i>0} 2^{K-i-1}} \tag{6.9}$$

Substituting λ in (6.8) we get:

$$p_i = -\lambda 2^{K-i-1} = \frac{2^{K-i-1}}{\sum\limits_{i:p_i>0} 2^{K-i-1}} \qquad \forall i \in [K+1] \tag{6.10}$$

Because of the nature of $F(\mathbf{p}) = \sum\limits_{i\in[K+1]} p_i^2 2^{i-K}$, the function is minimized when all p_i are positive. Thus, the probabilities are:

$$p_i = \frac{2^{K-i-1}}{\sum\limits_{i\in[K+1]} 2^{K-i-1}} = \frac{2^{K-i}}{2^{K+1}-1} \qquad \forall i \in [K+1] \tag{6.11}$$

and substituting in $F(\mathbf{p})$ we have:

$$\begin{aligned}
\sum_{i\in[K+1]} p_i^2 2^{i-K} &\geq \sum_{i\in[K+1]} (\frac{2^{K-i}}{2^{K+1}-1})^2 2^{i-K} \\
&= \sum_{i\in[K+1]} \frac{2^{K-i}}{(2^{K+1}-1)^2} \\
&= \frac{1}{2^{K+1}-1}
\end{aligned} \tag{6.12}$$

$F(\mathbf{p})$ can also be bounded with respect to the probabilities:

$$\begin{aligned}
\sum_{i\in[K+1]} p_i^2 2^{i-K} &= p_K^2 + \sum_{i\in[K]} p_i^2 2^{i-K} \\
&\geq p_K^2 + p_{min_{<K}}^2 \frac{2^K-1}{2^K} \\
&\geq p_K^2 + \frac{p_{min_{<K}}^2}{2}
\end{aligned} \tag{6.13}$$

Combining (6.3) with (6.12) and (6.13) we get:

$$\frac{\mathbf{E}[\sum\limits_{j\in[n]} c_j]}{\mathbf{E}[\sum\limits_{j\in[n]} c_j^*]} \leq 1 + \frac{2}{n} min\{2^{K+1}-1,\ \frac{2}{2p_K^2 + p_{min_{<K}}^2}\}$$

□

The above result is optimal in the sense that an adversary can force us to pay the $\sum\limits_{i\in[K+1]} p_i^2 2^{i-K}$ factor. First, the adversary sends enough jobs to guarantee with high probability the existence of a job of length 2^K. They then await for us to schedule the job and then they send a large number of jobs, overpowering the initial number. Thus, the coefficients of the $\Theta(n^2)$ terms in (6.2) are the ones controlling the numerator of the competitive ratio.

Let us try to generalize this algorithm in the case of m machines. Because SRPT is a 2-approximation on $P|r_j, pmtn|\sum C_j$, we have that:

$$\begin{aligned} c_j &\leq 2c_j^* + 2^K \Rightarrow \\ \sum_{j\in[n]} c_j &\leq 2\sum_{j\in[n]} c_j^* + n2^K \end{aligned} \tag{6.14}$$

In addition, the terms X_i in (6.2) are divided by the number of machines m, because the best case scenario is a uniform distribution of jobs on the machines. Thus, we get:

$$\begin{aligned} \mathbf{E}[\sum_{j\in[n]} c_j^*] &\geq \mathbf{E}[\sum_{i\in[K+1]} \frac{(X_i/m)^2}{2} 2^i] \\ &= \frac{n^2 2^K}{2m^2} \sum_{i\in[K+1]} p_i^2 2^{i-K} \end{aligned} \tag{6.15}$$

Combining (6.14) and (6.15), in the case of m machines, we have a final competitive ratio of $2 + \frac{2m^2}{n} min\{2^{K+1} - 1, \frac{2}{2p_K^2 + p_{min<K}^2}\}$. While it may seem paradoxical that an increased number of machines deteriorates the performance, it is actually the contrary; A larger number of machines allows each job to finish earlier, but it takes a larger number of jobs to achieve the same guarantee for the volume of the optimal schedule in each machine using the above analysis.

6.3 A Fully-Prompt Algorithm with Constant Competitive Ratio

Let n be the constant number of jobs that arrive at each time point $t \in [T]$. If we are to assume a sufficiently large number of machines $m \geq n\mathbf{E}[L]$, where L is the r.v denoting the length of a job, we can construct an $O(1)$-competitive, asymptotically optimal algorithm that is also fully-prompt. First, we explain the intuition that leads to the algorithm. For $\mathbf{E}[L] = \sum_{i\in[K+1]} p_i 2^i$ we have that m is at least $np_0 + np_1 \cdot 2 + \cdots + np_K \cdot 2^K$. At each time point t, the jobs that arrive consist of np_i jobs of length 2^i for all $i \in [K+1]$, in expectation. For larger job sizes, we also "get" more machines. In the same spirit, we assign $np_i \cdot 2^i$ machines solely for jobs of size 2^i for all $i \in [K+1]$.

Algorithm 6.3
Assign $np_i \cdot 2^i$ machines for jobs of length 2^i, meaning all jobs of size 2^i are scheduled exclusively on these machines.

Theorem 6.2. Algorithm 6.3 achieves a competitive ratio of $1 + O(\frac{1}{\sqrt{n}} \max_i \frac{\sqrt{1-p_i}}{\sqrt{p_i}})$ for the model with $m \geq n\mathbf{E}[L]$ machines and $n_t = n$ constant.

Proof. We observe that, if there were to arrive exactly np_i jobs at each time point, then the jobs would follow a stair like pattern, with np_i jobs at each step and 2^i steps until all

the "lower" steps are full and we need to fill the first one again. Note that the first step will be free by then. Subsequently, the start time of each job is r_j, unless in a previous time point more jobs than expected arrived. If there were to arrive more jobs at a given time point, then those jobs will shift the ones that follow by one place. This delay is counted every np_i steps. For example, if at time $t = 2$ arrive $np_i + 2$ jobs, then we will have a delay of 2, added for every np_i additional jobs. The way in which the machines are filled is illustrated in figure 6.3.

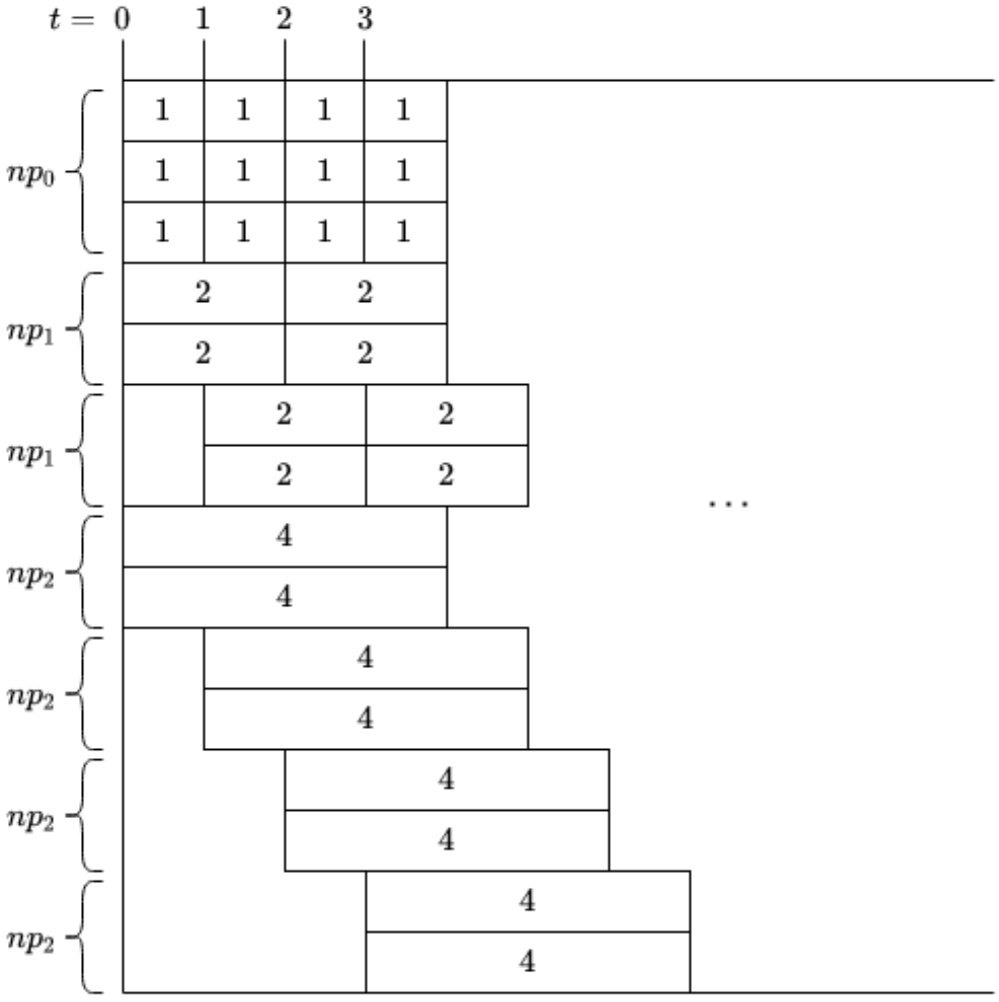


Figure 6.3: Many machine example for $K = 2$

In the above example, each time point exactly np_i jobs arrive for every $i \in [K + 1]$. To show how the schedule alters for an increased number of jobs arriving, the following example has 3 extra jobs of length 2 arrive at $t = 2$ and 1 less job of length 4 at $t = 2$:

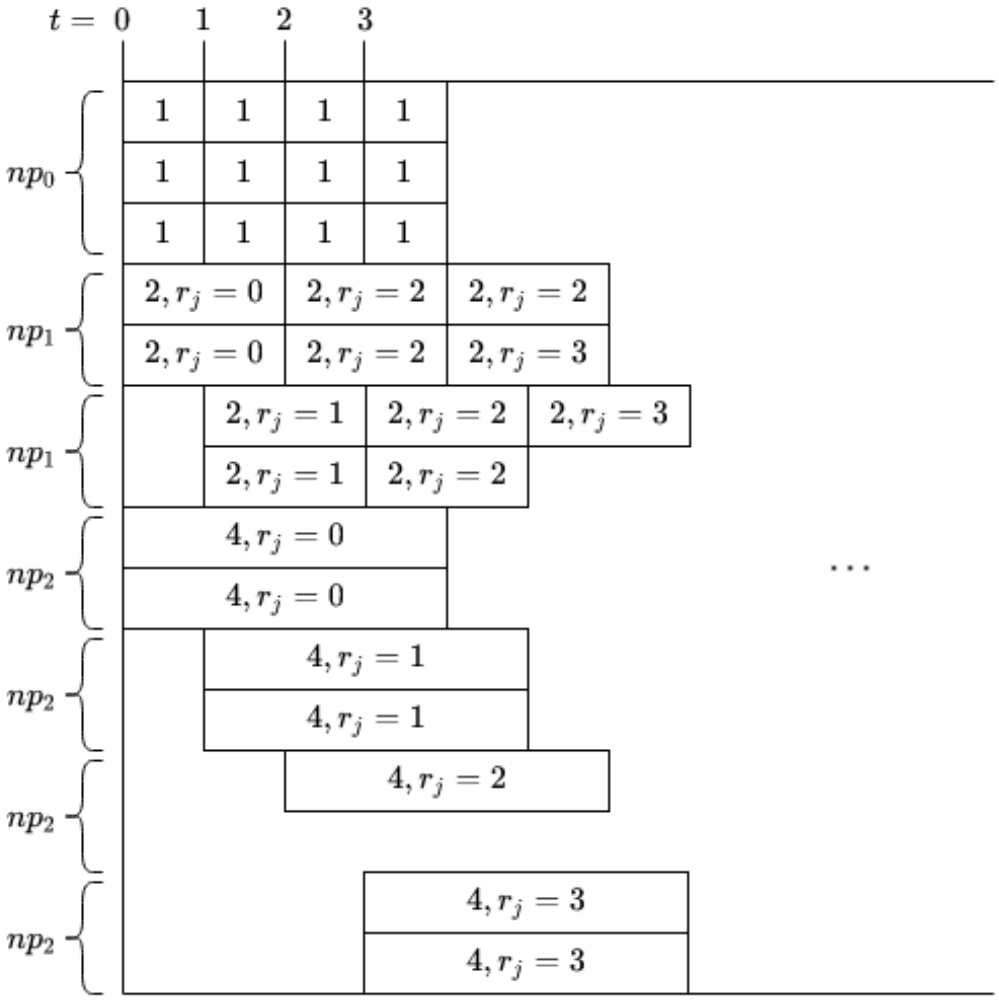


Figure 6.4: Many machine example for $K = 2$ with extra jobs arriving

We observe that for every extra job that arrives at a certain time, there is a delay added to the overall completion time for that particular length added every np_i jobs. Furthermore, missing jobs do not add a delay. Bounding the sum of completion times of each job size separately, denoted by ALG_i, with respect to the earliest completion time c_j^*, which is at most that of OPT_i and is exactly the arrival time plus the length of the job, we have that:

$$ALG_i \leq OPT_i + \sum_{t=0}^{T-1} \max(X_i^t - np_i, 0) \frac{\max(X_i^t - np_i, 0) + \sum_{l=t+1}^{T-1} X_i^l}{np_i} \tag{6.16}$$

where X_i^t is the number of jobs of length 2^i that arrive at time t. Moreover, let $X_i = \sum_{t=0}^{T} X_i^t$

be the total number of jobs of length 2^i that arrive at our system, nTp_i in expectation. The total delay is the sum of the delay the extra jobs cause to the ones that come after them and to themselves. Recall that, at this point, we only care to bound the jobs of size 2^i, using $2^i np_i$ machines. In order to bound the expectation of our algorithm, we need to calculate the expected value of the added delays. Since X_i^t are i.i.d for every t we get:

$$\mathbf{E}\left[\sum_{t=0}^{T-1}\max(X_i^t-np_i,0)\frac{\max(X_i^t-np_i,0)+\sum_{l=t+1}^{T-1}X_i^l}{np_i}\right]=$$

$$\frac{1}{np_i}\cdot\mathbf{E}\left[\sum_{t=0}^{T-1}\max(X_i^t-np_i,0)\left(\max(X_i^t-np_i,0)+\sum_{l=t+1}^{T-1}X_i^l\right)\right]=$$

$$\frac{1}{np_i}\cdot\mathbf{E}\left[\sum_{t=0}^{T-1}\left(\max(X_i^t-np_i,0)^2+\max(X_i^t-np_i,0)\sum_{l=t+1}^{T-1}X_i^l\right)\right]=$$

$$\frac{T}{np_i}\cdot\mathbf{E}\left[\max(X_i^0-np_i,0)^2\right]+\frac{1}{np_i}\sum_{t=0}^{T-1}\mathbf{E}[\max(X_i^t-np_i,0)]\sum_{l=t+1}^{T-1}\mathbf{E}[X_i^l]=$$

$$\frac{T}{np_i}\cdot\mathbf{E}\left[\max(X_i^0-np_i,0)^2\right]+\frac{1}{np_i}\mathbf{E}[\max(X_i^0-np_i,0)]\sum_{t=0}^{T-1}\sum_{l=t+1}^{T-1}np_i=$$

$$\frac{T}{np_i}\cdot\mathbf{E}[\max(X_i^0-np_i,0)^2]+\frac{T(T-1)}{2}\cdot\mathbf{E}[\max(X_i^0-np_i,0)] \tag{6.17}$$

Bounding the above expectations:

$$\mathbf{E}[|X_i^0-np_i|]^2\le\mathbf{E}[|X_i^0-np_i|^2]=\mathbf{Var}[X_i^0]=np_i(1-p_i) \tag{6.18}$$

which gives us:

$$\mathbf{E}[\max(X_i^0-np_i,0)]\le\mathbf{E}[|X_i^0-np_i|]\le\sqrt{n}\sqrt{p_i(1-p_i)} \tag{6.19}$$

Similarly:

$$\mathbf{E}[\max(X_i^0-np_i,0)^2]\le\mathbf{E}[|X_i^0-np_i|^2]=\mathbf{Var}[X_i^0]=np_i(1-p_i) \tag{6.20}$$

The optimal schedule can be bounded as follows:

$$OPT_i\ge\sum_{t=0}^{T-1}(t+2^i)X_i^t \tag{6.21}$$

where X_i^j is the number of jobs of length 2^i arriving at time j. Note that OPT_i's value is exactly each arrival time plus the job's length. Taking expectations, we have that:

$$\begin{aligned}\mathbf{E}[OPT_i]&\ge nTp_i2^i+np_i\sum_{t=0}^{T-1}t\\&\ge nTp_i2^i+np_i\frac{T(T-1)}{2}\end{aligned} \tag{6.22}$$

Substituting (6.19), (6.20) in (6.17) and dividing with (6.22):

$$\begin{aligned}\frac{\mathbf{E}[ALG_i]}{\mathbf{E}[OPT_i]} &\leq 1 + \frac{\frac{T}{np_i}np_i(1-p_i) + \frac{T(T-1)}{2}\sqrt{n}\sqrt{p_i(1-p_i)}}{nTp_i2^i + np_i\frac{T(T-1)}{2}} \\ &\leq 1 + \frac{T(1-p_i) + \frac{T(T-1)}{2}\sqrt{n}\sqrt{p_i(1-p_i)}}{np_i\frac{T(T-1)}{2}} \\ &= 1 + \frac{2(1-p_i)}{np_i(T-1)} + \frac{\sqrt{1-p_i}}{\sqrt{n}\sqrt{p_i}} \end{aligned} \tag{6.23}$$

Considering T to be adequately large, $\frac{2(1-p_i)}{np_i(T-1)}$ is dominated and our algorithm has a competitive ratio of $1 + O(\frac{1}{\sqrt{n}}\max_i \frac{\sqrt{1-p_i}}{\sqrt{p_i}})$, where we take the maximum ratio over all job lengths. □

Observe that, because we know how many jobs will come at a certain time beforehand, we are able to design a fully-prompt algorithm. This should come as no surprise; Recall the impossibility result of lemma 6.1. To give an example of an algorithm that cannot be prompt, we fill the machine with jobs according to how many jobs arrived before and how many empty space it left for extra jobs. Since we are not obliged to send a specific number of jobs, we choose adversarially to send enough jobs to fill the empty space and the ratio of $\Omega(K)$ is achieved. However, if the number of jobs that arrive is known in advance, then we can allocate exactly as many spots for jobs as we expect. Consequently, the limiting factor that impedes full-promptness with constant competitiveness is the online model; If we are not aware of how many jobs will arrive we cannot allocate space commensurately. This will be made apparent in the lower bound for fully-prompt algorithms that we will present next chapter.

Following the example of our prompt schedule with many machines and known number of arriving jobs, we present slight variations of the the model and modify the analysis of the algorithm accordingly.

6.3.1 Assuming periodicity of jobs arriving

In this generalization, we assume that n_t is a periodic sequence with period h. In our above analysis it was the case that at each time point the same number of jobs arrived, allowing the jobs to be uniformly distributed in time. It is clear that the input that deviated the most from the above, optimal, result, will occur when the sequenced is unbalanced. The most unbalanced sequence is when all the jobs in a period arrive at the first time point. Thus, the sequence is as follows:

$$n_0 = n \cdot h,\ n_1 = 0,\ n_2 = 0,\ ...,\ n_{h-1} = 0,\ n_h = n \cdot h,\ ... \tag{6.24}$$

where n is the mean number of jobs in a given period. We are to prove that algorithm 6.3 performs well on this variation of the model.

Theorem 6.3. Algorithm 6.3 achieves a $1 + O(\max_i \frac{\sqrt{1-p_i}}{\sqrt{hn}\sqrt{p_i}}) + \frac{h}{T-h}$ competitive ratio for the model of section 6.3.1.

Proof. As above, we bound the optimal schedule and the schedule produced by our algorithm:

$$OPT_i \geq \sum_{t=0}^{T-1} (t + 2^i) X_i^t \tag{6.25}$$

where now X_i^t have an expected value of $h \cdot n$ at integer multiples of h and zero everywhere else. Taking expectations:

$$\mathbf{E}[OPT_i] \geq hn\frac{T}{h}p_i 2^i + hnp_i(0 + h + 2h + ... + T - h) \tag{6.26}$$

$$= hn\frac{T}{h}p_i 2^i + h^2 np_i(1 + 2 + ... + \frac{T}{h} - 1) \tag{6.27}$$

$$= nTp_i 2^i + hnp_i \frac{T(\frac{T}{h} - 1)}{2} \tag{6.28}$$

In order to bound the delays of the schedule, let us first present an example of the sequence. In the following illustration, we have a period of $h = 12$. We observe that, for $h > 2^i$, a delay of 2^i is added to every batch of $2^i np_i$ jobs for the second batch, $2 \cdot 2^i$ for the third batch and so on. Obviously, there are $\frac{h}{2^i}$ batches in each period, as there are hnp_i jobs in each period.

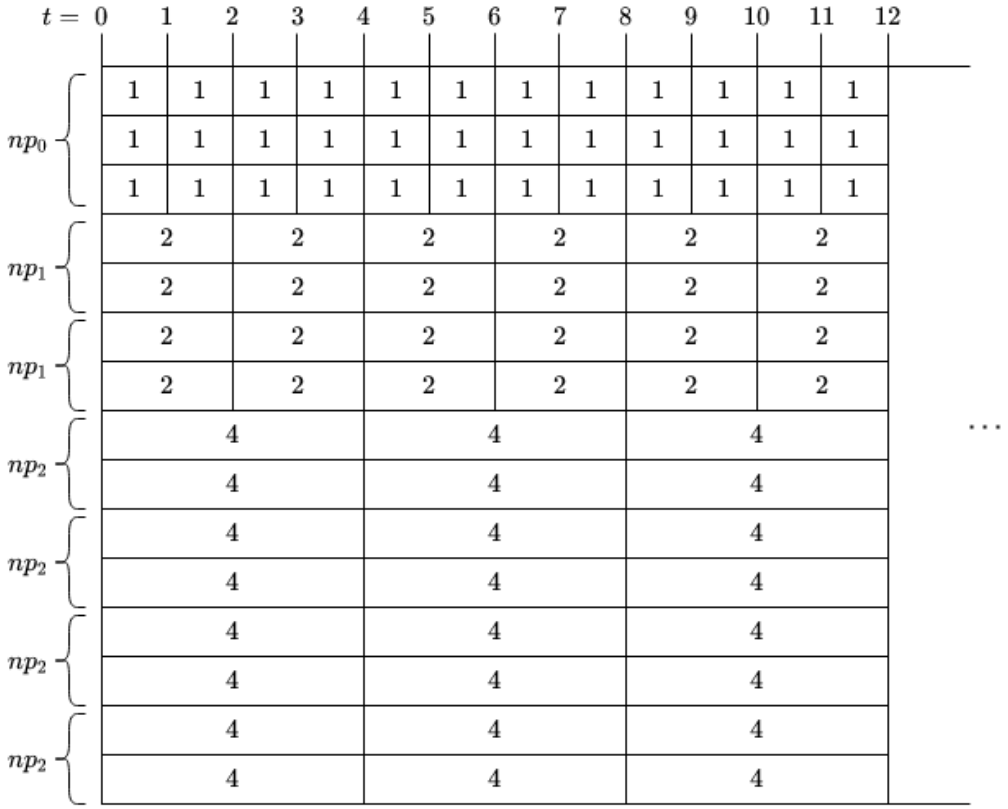


Figure 6.5: Many machine example for $h = 12, K = 2$

The delays induced by the unbalanced input (by the jobs on themselves) can be bounded as such:

$$\begin{aligned}\mathbf{E}[delay_i] &\leq \sum_{periods} 2^i np_i 2^i (0+1+2+...+\frac{h}{2^i}-1) \\ &= \frac{T}{h}\frac{\frac{h}{2^i}(\frac{h}{2^i}-1)}{2}2^{2i}np_i \\ &\leq \frac{Th}{2}np_i \end{aligned} \tag{6.29}$$

resulting in a competitive ratio:

$$\begin{aligned}\frac{\mathbf{E}[delay_i]}{\mathbf{E}[OPT_i]} &\leq \frac{hnp_iT}{hnp_iT(\frac{T}{h}-1)} \\ &= \frac{1}{\frac{T}{h}-1} \\ &= \frac{h}{T-h} \end{aligned} \tag{6.30}$$

This ratio refers to the delays induced by each batch of jobs to themselves. Moreover, since an extra job causes a delay of 2^i for one job, added every $2^i np_i$ jobs, the delays induced by the deviation of arrivals from their mean value in the rest of the jobs are bounded as follows:

$$\begin{aligned}\mathbf{E}[delay\ from\ extra\ jobs_i] &\leq \mathbf{E}\left[2^i \sum_{t=0}^{\frac{T}{h}-1} \max(X_i^{t\cdot h} - hnp_i, 0)\frac{\sum_{l=t+1}^{\frac{T}{h}-1} X_i^{l\cdot h}}{2^i np_i}\right] \\ &= \frac{1}{np_i}\sum_{t=0}^{\frac{T}{h}-1}\sum_{l=t+1}^{\frac{T}{h}-1}\mathbf{E}[\max(X_i^{t\cdot h}-hnp_i,0)]\mathbf{E}[X_i^{l\cdot h}] \\ &= h\frac{\frac{T}{h}(\frac{T}{h}-1)}{2}\mathbf{E}[\max(X_i^0-hnp_i,0)] \end{aligned} \tag{6.31}$$

Similar as before we have:

$$\mathbf{E}[\max(X_i^0 - hnp_i, 0)] \leq \sqrt{hn}\sqrt{p_i(1-p_i)} \tag{6.32}$$

This results in a total competitive ratio of:

$$\begin{aligned}\frac{\mathbf{E}[ALG_i]}{\mathbf{E}[OPT_i]} &\leq 1 + \frac{\frac{T(\frac{T}{h}-1)}{2}\sqrt{hn}\sqrt{p_i(1-p_i)}}{nTp_i2^i + hnp_i\frac{T(\frac{T}{h}-1)}{2}} + \frac{h}{T-h} \\ &\leq 1 + \frac{\sqrt{1-p_i}}{\sqrt{hnp_i}} + \frac{h}{T-h} \\ &= 1 + O(\frac{\sqrt{1-p_i}}{\sqrt{hn}\sqrt{p_i}}) + \frac{h}{T-h} \end{aligned} \tag{6.33}$$

Taking the maximum for every possible job length, the theorem follows.

□

6.3.2 Assuming the number of jobs arriving is a random variable

The algorithm is the same as section 6.3, assign $2^i np_i$ machines for jobs of size 2^i and probability p_i. Furthermore, we assume that at each time t, the number of jobs arriving is drawn from a distribution with expected value n. Symbolically, let N^t be the i.i.d random variables denoting the jobs arriving at time t, with $\mathbf{E}[N^t] = n \; \forall t \in [T]$. Furthermore, let Y_i^t denote the number of jobs of length 2^i arriving at time t. It is clear that the r.v Y_i^t follows a binomial distribution with parameters (N^t, p_i).

Theorem 6.4. Algorithm 6.3 achieves a $1 + O(\frac{\sqrt{K+1}}{\sqrt{n}}) + O(\frac{\sqrt{\mathbf{Var}[N^0]}}{n})$ competitive ratio for the model of section 6.3.2.

Proof. Again, we bound the schedule of our algorithm for each job length separately:

$$ALG_i \leq OPT_i + \sum_{t=0}^{T-1} \max(Y_i^t - np_i, 0) \frac{\max(Y_i^t - np_i, 0) + \sum_{l=t+1}^{T-1} Y_i^l}{np_i}$$

Taking expectations we get:

$$\mathbf{E}[ALG_i] \leq \mathbf{E}[OPT_i] + \frac{1}{np_i} \left(\sum_{t=0}^{T-1} \mathbf{E}[\max(Y_i^t - np_i, 0)^2] + \sum_{t=0}^{T-1} \mathbf{E}[\max(Y_i^t - np_i, 0)] \sum_{l=t+1}^{T-1} \mathbf{E}[Y_i^l] \right) \tag{6.34}$$

since Y_i^t are independent for different times t. We can bound each of these expected values separately:

$$\begin{aligned} \mathbf{E}[\max(Y_i^t - np_i, 0)^2] &\leq \mathbf{E}[|Y_i^t - np_i|^2] \\ &= \mathbf{E}[(Y_i^t)^2] - 2 \cdot \mathbf{E}[Y_i^t \cdot (np_i)] + \mathbf{E}[(np_i)^2] \\ &= \mathbf{E}[(Y_i^t)^2] - 2(np_i)\mathbf{E}[Y_i^t] + (np_i)^2 \end{aligned} \tag{6.35}$$

Calculating the above expectations:

$$\mathbf{E}[Y_i^t] = \mathbf{E}[\mathbf{E}[Y_i^t | N^t]] = \mathbf{E}[N^t p_i] = np_i \tag{6.36}$$

$$\begin{aligned} \mathbf{E}[(Y_i^t)^2] &= \mathbf{E}[Y_i^t]^2 + \mathbf{Var}[Y_i^t] \\ &= (np_i)^2 + \mathbf{E}[\mathbf{Var}[Y_i^t | N^t]] + \mathbf{Var}[\mathbf{E}[Y_i^t | N^t]] \\ &= (np_i)^2 + \mathbf{E}[\mathbf{E}[(Y_i^t - \mathbf{E}[Y_i^t | N^t])^2 | N^t]] + \mathbf{Var}[N^t p_i] \\ &= (np_i)^2 + \mathbf{E}[\mathbf{E}[(Y_i^t - N^t p_i)^2 | N^t]] + p_i^2 \, \mathbf{Var}[N^t] \\ &= (np_i)^2 + \mathbf{E}[N^t p_i (1 - p_i)] + p_i^2 \, \mathbf{Var}[N^t] \\ &= (np_i)^2 + np_i(1 - p_i) + p_i^2 \, \mathbf{Var}[N^t] \end{aligned} \tag{6.37}$$

Substituting in (6.35):

$$\begin{aligned}\mathbf{E}[\max(Y_i^t - np_i, 0)^2] &\leq (np_i)^2 + np_i(1-p_i) + p_i^2\,\mathbf{Var}[N^t] - 2(np_i)^2 + (np_i)^2 \\ &= np_i(1-p_i) + p_i^2\,\mathbf{Var}[N^t] \end{aligned} \tag{6.38}$$

Similarly:

$$\begin{aligned}\mathbf{E}[\max(Y_i^t - np_i, 0)] &\leq \mathbf{E}[|Y_i^t - np_i|] \\ &= \sqrt{\mathbf{E}[|Y_i^t - np_i|]^2} \\ &\leq \sqrt{\mathbf{E}[|Y_i^t - np_i|^2]} \\ &= \sqrt{np_i(1-p_i) + p_i^2\,\mathbf{Var}[N^t]} \\ &\leq \sqrt{np_i(1-p_i)} + p_i\sqrt{\mathbf{Var}[N^t]} \end{aligned} \tag{6.39}$$

Substituting (6.34) we get:

$$\begin{aligned}\mathbf{E}[ALG_i] &\leq \mathbf{E}[OPT_i] + \frac{1}{np_i}\left(\sum_{t=0}^{T-1}\mathbf{E}[\max(Y_i^t - np_i, 0)^2] + \sum_{t=0}^{T-1}\mathbf{E}[\max(Y_i^t - np_i, 0)]\sum_{l=t+1}^{T-1}\mathbf{E}[Y_i^l]\right) \\ &\leq \mathbf{E}[OPT_i] + \frac{1}{np_i}\left(\sum_{t=0}^{T-1}(np_i(1-p_i) + p_i^2\,\mathbf{Var}[N^t]) + \sum_{t=0}^{T-1}\left(\sqrt{np_i(1-p_i)} + p_i\sqrt{\mathbf{Var}[N^t]}\right)\sum_{l=t+1}^{T-1}(np_i)\right) \\ &= \mathbf{E}[OPT_i] + \left(\sum_{t=0}^{T-1}((1-p_i) + \frac{p_i\,\mathbf{Var}[N^t]}{n}) + \sum_{t=0}^{T-1}\sum_{l=t+1}^{T-1}\left(\sqrt{np_i(1-p_i)} + p_i\sqrt{\mathbf{Var}[N^t]}\right)\right) \\ &= \mathbf{E}[OPT_i] + \left(T\left((1-p_i) + \frac{p_i\,\mathbf{Var}[N^0]}{n}\right) + \frac{T(T-1)}{2}\left(\sqrt{np_i(1-p_i)} + p_i\sqrt{\mathbf{Var}[N^0]}\right)\right)\end{aligned} \tag{6.40}$$

Bounding OPT_i we get:

$$\begin{aligned}\mathbf{E}[OPT_i] &\geq \mathbf{E}\left[\sum_{t=0}^{T-1}(2^i + t)Y_i^t\right] \\ &= Tnp_i2^i + np_i\sum_{t=0}^{T-1}t \\ &= Tnp_i2^i + np_i\frac{T(T-1)}{2}\end{aligned} \tag{6.41}$$

Summing for all job lengths we have that:

$$\begin{aligned}
\mathbf{E}[OPT] &= \sum_{i=0}^{K} \mathbf{E}[OPT_i] \\
&\geq \sum_{i=0}^{K} \left(Tnp_i 2^i + np_i \frac{T(T-1)}{2} \right) \\
&\geq Tn\mathbf{E}[L] + n\frac{T(T-1)}{2} \qquad (6.42)
\end{aligned}$$

Similarly, summing the delays for each job length we get:

$$\begin{aligned}
\mathbf{E}[delay] &\leq \sum_{i=0}^{K} \left(T\left((1-p_i) + \frac{p_i \,\mathbf{Var}[N^0]}{n} \right) + \frac{T(T-1)}{2} \left(\sqrt{np_i(1-p_i)} + p_i\sqrt{\mathbf{Var}[N^0]} \right) \right) \\
&\leq \left(T\left(K + \frac{\mathbf{Var}[N^0]}{n} \right) + \frac{T(T-1)}{2} \left(\sqrt{n} \sum_{i=0}^{K} \sqrt{p_i(1-p_i)} + \sqrt{\mathbf{Var}[N^0]} \right) \right) \\
&\leq \left(T\left(K + \frac{\mathbf{Var}[N^0]}{n} \right) + \frac{T(T-1)}{2} \left(\sqrt{n} \sqrt{\left(\sum_{i=0}^{K} 1 \right) \left(\sum_{i=0}^{K} p_i(1-p_i) \right)} + \sqrt{\mathbf{Var}[N^0]} \right) \right) \\
&\leq \left(T\left(K + \frac{\mathbf{Var}[N^0]}{n} \right) + \frac{T(T-1)}{2} \left(\sqrt{n}\sqrt{K+1} + \sqrt{\mathbf{Var}[N^0]} \right) \right) \qquad (6.43)
\end{aligned}$$

From the above, we can calculate the total competitive ratio of the algorithm:

$$\begin{aligned}
\frac{\mathbf{E}[ALG]}{\mathbf{E}[OPT]} &\leq 1 + \frac{T\left(K + \frac{\mathbf{Var}[N^0]}{n} \right) + \frac{T(T-1)}{2} \left(\sqrt{n}\sqrt{K+1} + \sqrt{\mathbf{Var}[N^0]} \right)}{Tn\mathbf{E}[L] + n\frac{T(T-1)}{2}} \\
&\leq 1 + \frac{T\left(K + \frac{\mathbf{Var}[N^0]}{n} \right) + \frac{T(T-1)}{2} \left(\sqrt{n}\sqrt{K+1} + \sqrt{\mathbf{Var}[N^0]} \right)}{n\frac{T(T-1)}{2}} \\
&\leq 1 + \frac{2\left(K + \frac{\mathbf{Var}[N^0]}{n} \right) + T\left(\sqrt{n}\sqrt{K+1} + \sqrt{\mathbf{Var}[N^0]} \right)}{nT} \\
&= 1 + \frac{2K}{nT} + \frac{2\,\mathbf{Var}[N^0]}{n^2T} + \frac{T\left(\sqrt{n}\sqrt{K+1} + \sqrt{\mathbf{Var}[N^0]} \right)}{nT} \\
&= 1 + \frac{2K}{nT} + \frac{2\,\mathbf{Var}[N^0]}{n^2T} + \frac{\sqrt{K+1}}{\sqrt{n}} + \frac{\sqrt{\mathbf{Var}[N^0]}}{n} \qquad (6.44)
\end{aligned}$$

The dominant factors in the competitive ratio are $\frac{\sqrt{K+1}}{\sqrt{n}}$ and $\frac{\sqrt{\mathbf{Var}[N^0]}}{n}$. Thus the competitive ratio is $1 + O(\frac{\sqrt{K+1}}{\sqrt{n}}) + O(\frac{\sqrt{\mathbf{Var}[N^0]}}{n})$.

□

6.4 A Semi-Prompt Algorithm for Machine Scheduling

So far, we presented fully prompt algorithms in the case where we had $n\,\mathbf{E}[L]$ machines. Let us consider again the main model: Jobs of length $2^i, i \in [K+1]$ arrive at discrete time points, where the number of jobs is chosen adversarially. We shall present a convex combination of the algorithm of section 6.2 and of the exponential sequence in theorem 4.1. When a job arrives, we give the job a guarantee that its completion time will be less than or equal to its completion time in the exponential sequence times a constant z plus the maximum job length. Symbolically, $c_j \leq z \cdot c_j^{MENU} + 2^K$. Note that the additive factor 2^K enables us to schedule a large job without worrying about violating a deadline. The algorithm is the following:

Algorithm 6.4
When the machine is idle and no deadline needs to be satisfied, i.e., $t + l_j \leq z \cdot c_j^{MENU}$ for all jobs j, where t is the current time, schedule the shortest job available, otherwise schedule the job whose deadline is imminent.

The algorithm is essentially the same as in section 6.2 with the added constraint of the deadlines. We shall analyze the behavior of this scheme in the worst case scenario i.e when the machine is full and the smaller jobs are forced to pay the delay induced by deadlines being fulfilled.

Theorem 6.5. Algorithm 6.4 achieves a $O(\frac{2z(K-1)+z(z-1)(K^2+2K-1)}{(1+(z-1)(K+1))(2+(z-1)(K+1))}+\frac{2^K}{n})$ competitive ratio for model of 6.1.

Proof. The worst case schedule looks as follows:

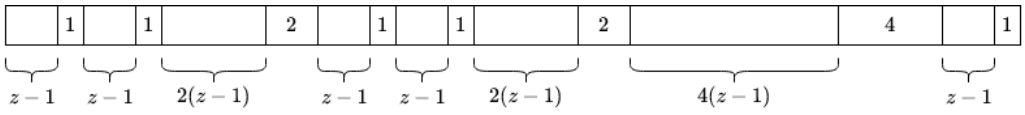


Figure 6.6: Semi-prompt algorithm sequence

where the $(z-1)$ is empty space and we can schedule any job we want, while the other intervals are reserved for the deadlines of each job. We will compute the sum of completion times in the case where all the deadline intervals are filled with their respective jobs and all other intervals are filled with jobs of size 1. This schedule will give the worst approximation as smaller jobs are forced to pay the price of waiting for a deadline and this price is added as many times as possible. First, we compute the sum of completion times in the general case. Furthermore, if the machine was not full at all times, we could improve the completion time of a job. Let a_x denote the value of the schedule when the largest job in it has size 2^x. We then solve the following linear recurrence equation:

$$
\begin{aligned}
a_x = 2a_{x-1} &+ ((2^x - 1) + x2^{x-1}(z-1))zx2^{x-1} \\
&+ z(x+1)2^x + (z-1)2^x(zx2^{x-1} \cdot 2) \\
&+ \frac{2^x(z-1)(2^x(z-1)+1)}{2}
\end{aligned}
$$

Let us explain each term of the above equation separately:

- $2a_{x-1}$: We are doubling the sequence and adding the previous value two times. This accounts only for the completion times of the appended a_{x-1} schedule between the jobs that are inside it, not taking into account the starting time of the schedule, which is equal to the end of the first a_{x-1} schedule.
- $(2^x - 1)zx2^{x-1}$: The jobs which complete on their deadline are exactly as many as the jobs in the exponential sequence in Eden et al. i.e $(2^x - 1)$. These jobs are offseted by the end of the previous schedule a_{x-1}, which has length $zx2^{x-1}$.
- $x2^{x-1}(z-1)zx2^{x-1}$: Similar as above, but now the non-deadline intervals are filled with $x2^{x-1}(z-1)$ jobs of length one.
- $z(x+1)2^x$ This is the completion time of the new largest job with length 2^x at the last deadline of the schedule.
- $(z-1)2^x(zx2^{x-1} \cdot 2)$ The offset for the new jobs of size one before the deadline of 2^x. There are $(z-1)2^x$ jobs while the length of the schedule is two times $zx2^{x-1}$.
- $\frac{2^x(z-1)(2^x(z-1)+1)}{2}$ The relative completion times of the jobs of size one before the deadline of 2^x.

Solving the equation while requiring that $a_0 = \frac{z(z+1)}{2}$ we have that:

$$
\begin{aligned}
a_x = &\ 4^x \cdot z(x-1) \\
&+ 4^x \cdot z(z-1)(\frac{1}{2}x^2 - x + \frac{3}{2}) \\
&+ 2^x \cdot zx(\frac{1}{4}x + \frac{5}{4}) \\
&+ 4^x \cdot z(z-1)(2x-2) \\
&+ 4^x \cdot (z-1)^2 \\
&+ 2^x \cdot (z-1)\frac{x}{2} \\
&+ 2^x \cdot (3z-1)
\end{aligned}
$$

where the third and second from last terms are both from the relative completion times in the free space, before the deadline of 2^x and the last term is the one that gives us $a_0 = \frac{z(z+1)}{2}$. Every other term is derived from the additive factors of the difference equation in that order.

Calculating the optimal schedule, we order the jobs in increasing job length and have:

$$OPT = \frac{2^K(1+(z-1)(K+1))2^K(2+(z-1)(K+1))}{2} + 2^K(1+(z-1)(K+1))(2^K-1) + \sum_{i=1}^{K}\left(\frac{X_i(X_i+1)}{2}2^i + X_i\sum_{j=1}^{i-1}X_j 2^j\right) \quad (6.45)$$

where $X_i = 2^{K-i}$ is the number of jobs of length 2^i. The last term is $O(4^K)$ and can be ignored. The first term is the relative sum of completion times among the jobs of size one, while the second is the delay induced on larger jobs by the jobs of size one, which were scheduled first. We shall calculate the competitive ratio of the dominant terms. Note, we will also use the dominant terms when $z = 1$ since this shows the trade-off between the algorithm in 4.1 and the algorithm of section 6.2. Thus, the dominant factors when enforcing the deadlines are $4^x \cdot z(x-1) + 4^x \cdot z(z-1)(\frac{1}{2}x^2 - x + \frac{3}{2}) + 4^x \cdot z(z-1)(2x-2) = 4^x \cdot z(x-1) + 4^x \cdot z(z-1)(\frac{1}{2}x^2 + x - \frac{1}{2})$

$$\frac{SCH}{OPT} \leq \frac{4^K \cdot 2z(K-1) + 4^K \cdot z(z-1)(K^2+2K-1)}{4^K(1+(z-1)(K+1))(2+(z-1)(K+1))} = \frac{2z(K-1) + z(z-1)(K^2+2K-1)}{(1+(z-1)(K+1))(2+(z-1)(K+1))}$$

When $z = 1$, the above competitive ration degrades to $O(K)$, while when $z \to \infty$ the combined ratio approaches 1. The final competitive ratio is the above plus the factor $\frac{2^K}{n}$. This occurs when a large job arrives at our system and we schedule it, not knowing many smaller jobs will arrive immediately after. This factor is computed exactly as in section 6.2, giving us the final ratio $O(\frac{2z(K-1)+z(z-1)(K^2+2K-1)}{(1+(z-1)(K+1))(2+(z-1)(K+1))} + \frac{2^K}{n})$.

□

Also, for $z = 2$, we have a competitive ratio of $\frac{4(K-1)+2(K^2+2K-1)}{(1+(K+1))(2+(K+1))} = O(1)$. This shows us that, even if we give a little leeway to the exponential sequence of 4.1, we can achieve constant competitive ratio while also having a guarantee of $c_j \leq zc_j^{MENU} + 2^K$ for the job compared to its completion time in 4.1.

Corollary 6.1. Algorithm 6.4 is $O(1 + \frac{2^K}{n})$-competitive for $z \geq 2$.

6.5 Adding Truthfulness to the Algorithm in Phillips et al.

In Phillips et al. [46], algorithms are presented that achieve an $O(1)$ competitive ratio by converting preemptive schedules to non preemptive, for the problems $1|r_j|\sum C_j$ and $P|r_j|\sum C_j$. These algorithms schedule the jobs in decreasing order of completion times that are calculated by the WSRPT algorithm and then converting the schedule to a non preemptive one using their algorithm CONVERT. This is sufficient to achieve a 2-approximation for the case with only one machine and 6-approximation for identical parallel machines. Using the techniques developed by Angel et al. [3], we can achieve

truthfulness for the above algorithms by charging each job a payment π_j. In our model, we make the following assumptions: A job j has valuation equal to its negative completion time i.e $-c_j$. Furthermore, its utility is quasilinear and equal to $u_j = -c_j - \pi_j$. Since we allow negative valuations we can argue that if a job is not completed its completion time is infinite and its valuation negative infinity. Lastly, we assume that we can forcibly remove jobs from their schedule if we choose to do so. This last assumption frees us to remove a job from its machine if we discover that it lied about its processing time. We set each payment $\pi_j = -start_j + \sum_{i \neq j}(r_i + b_i)$ were $start_j$ is the time job j starts processing and b_j is the bid referring to the processing time of job j. If we were to assume truthfulness, this payment function is valid, since the start time of job j cannot be greater than the sum of all the other arrival times and all the other processing times. Furthermore, the utility of each agent would be $u_j = -(start_j + l_j) - (-start_j + \sum_{i \neq j}(r_i + l_i)) = -l_j - \sum_{i \neq j}(r_i + l_i)$. If an agent were to lie about his processing time, bidding a lower value, then we could forcibly remove him from the schedule the moment his processing were to end, resulting in $-\infty$ utility. Moreover, a greater bid would not impact his/her utility. We conclude that an agent has no incentive to bid different from his true parameters and the mechanism is truthful. We note that the above require the mechanism to know the arrival times; r_j are not reported by the agents.

Chapter 7

Promptness and Truthfulness in the Travelling Repairman Problem

In this chapter, we try to insert the property of promptness into algorithms for the TRP. We shall start by explaining the problem in detail. In the TRP, jobs arrive at points of a metric space $\mathcal{M}$. The machine, or server, travels through the metric space at unit speed. The server also starts at time zero at a fixed point of $\mathcal{M}$ called the origin. The objective of the server is to reach all jobs at their locations and serve them. We evaluate the performance of our algorithm by the sum of (weighted) completion times. To illustrate this, think back to our example with the locksmith. The locksmith receives online requests to go to houses and unlock a door. The houses are the jobs located at points of a metric space. In order for the locksmith to please his costumers, he needs to serve them as soon as possible. This is achieved when all costumers have a small completion time. The challenge for the locksmith is to know when to travel to a far away location, as doing so might delay houses that are located near him/her. We also add the extra constraint of promptness; the locksmith must inform the costumers about their completion time upon their arrival.

Our contribution to the problem is designing a fixed path that the server takes in order to serve jobs. We construct this path by approximating the metric space $\mathcal{M}$ with its Hierarchical Well Separated Tree (HST) [9,24]. This approximation embeds the metric space in a tree metric while increasing the distances by at most a factor of $\log n$, where n is the number of points in the metric space. This method achieves promptness but might have a bad competitive ratio, as it is essentially a space filling curve. We mitigate this by proving a lower bound of $\Omega(\frac{TSP_{tour}}{\Delta})$, where TSP_{tour} is the optimal TSP tour of the metric space and Δ is the diameter of the space. This means that our algorithm, which achieves a $O(\frac{TREE_{tour}}{\Delta})$ competitive ratio, is within $\log n$ of the lower bound, with the $\log n$ factor deriving from the HST approximation of the original metric space. We also give a $O(1)$-competitive, semi-prompt algorithm where we allow slackness to reach its highest possible value while still being finite. This is possible due to the finite nature of the metric space.

Again, we stress the importance of promptness in designing algorithms. Our algorithms are obviously truthful, as the server travels a fixed path; no matter the declared weight or location of the job, the server will reach it at the same time. It also does

not help to declare false jobs next to your location to temp the server into coming earlier. Promptness constraint forces us to follow a fixed path and this in turn gives rise to obvious truthfulness.

7.1 The Model

Before formally describing our problem, let us first define the notion of a metric space.

Definition 7.1 (Metric Space)**.** A metric space $\mathcal{M}$ is an ordered pair (M, d), where M is a set of points and $d : M \times M \to \mathbb{R}$ is a distance function such that for any $x, y, z \in M$, the following holds:

$$d(x, y) = 0 \Leftrightarrow x = y$$
$$d(x, y) = d(y, x)$$
$$d(x, z) \leq d(x, y) + d(y, z)$$

From the above definition, it follows that the distance metric d is non negative. Moreover, we use the terms $\mathcal{M}$ and M interchangeably, meaning we might say $x \in \mathcal{M}$, when in reality we mean $x \in M$.

Now that we have defined metric spaces, let us formally describe the Travelling Repairman Problem. In the TRP, a server moves at unit speed through points of a metric space $\mathcal{M}$. The server starts out at time $t = 0$ at a special point of the metric space called the origin, denoted by o. Jobs arrive at points of the metric space, waiting to be served. The arrival time of job j is denoted by r_j and its point on the metric space, simply by j. The server's goal is to reach the position of these requests and to minimize the sum $\sum_{j=1}^{n_{requests}} w_j c_j$, where w_j is the weight of a request, c_j its completion time and $n_{requests}$ the number of requests. In the unweighted version of the problem all jobs have unit weight and the server aims to minimize $\sum_{j=1}^{n_{requests}} c_j$. By n, we denote the number of points in the metric space $\mathcal{M}$ and by Δ its diameter, meaning the largest distance between any two points.

Augmenting our setting with notions from mechanism design, the utility of each job is $u_j = -c_j - p_j$ or $u_j = -w_j c_j - p_j$ for weighted jobs and the welfare of the mechanism is the negative sum of (weighted) completion time $-\sum_{j=1}^{n_{requests}} w_j c_j$.

7.2 Tree Embeddings for Metric Spaces

In our algorithms, we approximate general metric spaces by tree metrics. This makes the analysis easier and allows us to use the inherent structure of the tree to design algorithms and tours.

Definition 7.2 (Embeddings and Distortion)**.** An embedding of a metric space (X, d_X) into a metric space (Y, d_Y) is a map $f : X \to Y$. Its (bi-Lipschitz) distortion is the least $D \geq 1$ such that for all $x, y \in X$:

$$d_X(x, y) \leq d_Y(f(x), f(y)) \leq D \cdot d_X(x, y)$$

Furthermore, randomized embeddings are embeddings for which we can bound the distortion in expectation.

Definition 7.3 (Tree Metric)**.** A tree metric is a metric space whose points are nodes on a tree and its distance function can be described by the distances on a tree with non negative weights on its edges.

Since the tree is a simple graph, we would like to embed arbitrary metric spaces into tree metrics to make use of the structure of the tree. The trees we use in our approximations are called hierarchically well-separated trees (HST). An HST is a tree metric whose weights on the edges grow smaller as we approach the leaves.

Definition 7.4 (k-Hierarchically Well-Separated Trees)**.** A k-hierarchically well-separated tree (k-HST) is a rooted weighted tree $T = (V(T), E(T))$ satisfying the following properties:

1. For every node $v \in V(T)$, all edges connecting v to a child are of equal weight.
2. The edge weight along a path from the root to a leaf decrease by a factor of at least k.

We now present an important theorem that serves as the basis for diverting to a tree metric.

Theorem 7.1 (Tree Metric)**.** Let (X, d_X) be an n-point metric space. There exists a randomized polynomial-time algorithm that embeds X into the set of leaves of a 2-HST $T = (V(T), E(T))$ such that the following holds (we may assume that $X \subseteq V(T)$).

1. For every $x, y \in X$ $\ d_X(x, y) \leq d_T(x, y)$.
2. For every $x, y \in X$ $\ \mathbb{E}[d_T(x, y)] \leq O(logn) \cdot d_X(x, y)$.

The above theorem is thanks to Fakcharoenphol, Rao and Talwar [24]. Through the above theorem, we can approximate metric spaces with n points with expected $O(\log n)$ distortion and use the structure of the 2-HST to design algorithms. Note that, since our criteria for evaluating algorithms is linear and can be expressed as a sum of distances (a jobs completion time is the length the server has travelled until it reached the job) the expected distortion factor $O(\log n)$ carries on and multiplies our competitive ratio for algorithms on the tree metric. Thus, with an extra overhead of $O(\log n)$ we can design algorithms on tree metrics for general metric spaces.

Without loss of generality, we assume the tree to be complete with degree d. Here is a depiction of the tree for $d = 3$ and height $h = 3$:

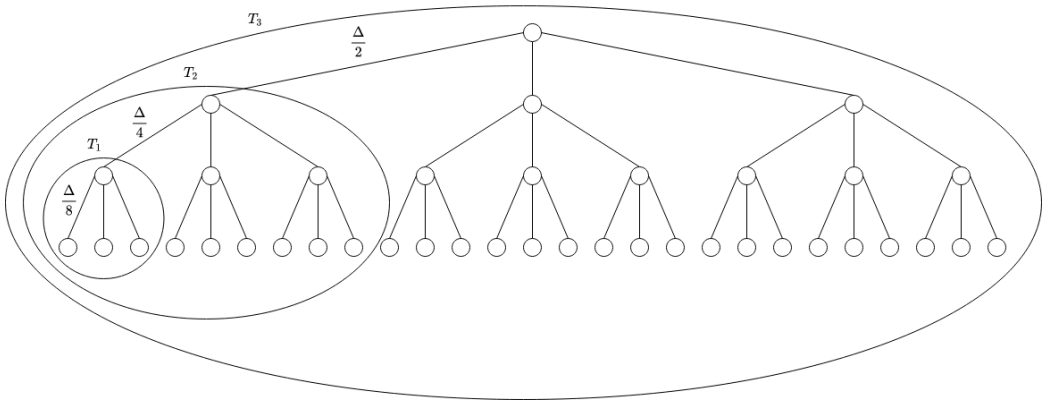


Figure 7.1: A 2-HST with height $h = 3$ and degree $d = 3$

We omit the edge weights of different children as they have the same weight as the ones depicted. The weights are such that the total distance across two leaves with earliest common ancestor the root, have distance Δ. Note that this is the diameter of the tree, not the original metric. Moreover, without loss of generality, we assume the edges connecting to leaves have length 1 and that the origin o is the leftest leaf. Since in our algorithms we concern ourselves with the length of a particular path, we are interested what is the length of a sub-tree. We denote the tree rooted at the origin as T_0. The tree rooted at the father of the origin is T_1 and so on. Obviously, the whole tree is $T = T_h$, where h is the height of the tree. The length of a tour of T_1 (without returning to the origin) is $d \cdot 1 = d$. When we go up a level, the following recursive relation holds:

$$|T_{x+1}| = d(|T_x| + 2^x) \tag{7.1}$$

where we denote the length of the tour on T_x by $|T_x|$. Solving the above recursive relation, we get that:

$$|T_x| = \sum_{i=0}^{x-1} d^{x-i} 2^i \tag{7.2}$$

From (7.2) we have that the length of the tree is $|T_h| = d^h + 2d^{h-1} + ... + 2^{h-1}d$. For $d \geq 3$, the d^h term dominates and we have $|T_h| = O(d^h)$. On the other hand, for $d = 2$ we have that $|T_h| = O(h \cdot d^h) = O((\log \Delta)d^h)$. If we use the complete tree property, we have that $d^h = n$, where n is the number of point in the metric space. So, for $d \geq 3$, the length of a tour on the tree is $O(n)$ and $O(n \cdot \log \Delta)$ for $d = 2$. We use these relations to refer to the lengths of tree parts with parameters that exhibit the nature of the metric space. Again, while n is the number of points in the original space, Δ is the diameter of the tree metric.

7.3 A Semi-Prompt Algorithm for the TRP

We base our approach on the algorithm of Krumke et al. [32, 38] for the online TRP. Their algorithm, as explained in 3.2, uses schedules of geometrically increasing length. In each of these schedules, the server gathers the largest amount of job weight it can, without

surpassing an exponentially increasing length. Specifically, the algorithm divides time in intervals $[b_{i-1}, b_i]$, where $b_i = (1+\sqrt{2})b_{i-1}$. At time b_i the algorithm computes a schedule of length at most b_i that gathers the maximum weight of jobs. At time b_i, the server needs to take at most b_{i-1} to return to the origin and another b_i for the new schedule. We have that $b_i + b_{i-1} + b_i = b_i + \frac{1}{1+\sqrt{2}}b_i + b_i = (1+\sqrt{2})b_i = b_{i+1}$ and we are in time to compute the next schedule. This is where the need for $1+\sqrt{2}$ stems from. This algorithm is $(1+\sqrt{2})^2$-competitive. Because of the finite nature of the metric space we use, it is clear that once the increasing length in Krumke's algorithm surpasses the length of the metric space, all available jobs are scheduled in the current schedule the algorithm computes. This gives rise to a trivially semi-prompt algorithm:

Algorithm 7.3

1. Follow Krumke's algorithm until the maximum length of the schedules is greater than a global tour.
2. Make sequential global tours on the tree forever.

Theorem 7.2. Algorithm 7.3 is $O(1)$-competitive with an overall slackness of $O(\frac{d^h}{\Delta})$.

Proof. From Krumke's analysis, our algorithm achieves a constant competitive ratio of $(1+\sqrt{2})^2$ for jobs that arrived and finished during step 1. For jobs that arrived and completed in step 2 we have that algorithm $c_j^* \geq |T_h|$ while it takes at most 2 tours to reach a point (one going and one returning). Thus, $c_j \leq 3|T_h| \leq 3c_j^*$. Lastly, there can be no job that arrived in step 1 and completes during step 2, since the schedule length in step 1 is large enough to be able to reach it. Hence, the algorithm is $(1+\sqrt{2})^2$-competitive. As for the slackness of the algorithm, it is determined by the behaviour of the algorithm on jobs that are far from the origin. If a job at diameter distance arrives early, its earliest completion time might be Δ, but it can also wait until we are able to complete a whole tour to be reached. As such, $s_j = \Delta, e_j = |T_h| = O(d^h)$, and the slackness is $\lambda_j = O(\frac{d^h}{\Delta})$. For jobs that are closer, e_j is divided by d (going one sub-tree lower), while s_j is only divided by 2, so the overall slackness is $\lambda = O(\frac{d^h}{\Delta})$. □

Note that this algorithm might be a necessity to make a guarantee to a job that its completion time will be less than $c \cdot \max\{d(o,j), r_j\}$, where c is a constant determined by the promise of the algorithm. Assume a job j at distance Δ from the origin. If a prompt algorithm were to promise a $c \cdot \max\{d(o,j), r_j\} = c \cdot \max\{\Delta, r_j\}$ bound on j's completion time, then c would have to account for other possible jobs that should be finished earlier than j. If job i was closer to the origin then $d(o,i) < d(o,j)$ and its promise $c \cdot \max\{d(o,i), r_i\} < c \cdot \max\{d(o,j), r_j\}$ would be less than j's. This is true for every job in the tree to the left of j, meaning all points of the metric space, if we assume j to be the farthest job. So we have that:

$$c \cdot d(o,j) \geq |T_h| \Leftrightarrow c \geq \frac{|T_h|}{\Delta} = O(\frac{d^h}{\Delta}) = O(\frac{n}{\Delta}) \tag{7.3}$$

or $c = O(\frac{2^h \log \Delta}{\Delta}) = O(\log \Delta)$ for $d = 2$. This means that, since we cant promise anything better, we might as well use algorithm 7.3 and achieve a constant competitive ratio.

7.4 A Fully-Prompt Algorithm for the TRP

In this section, we describe a fully-prompt algorithm that achieves a competitive ratio of $O(\frac{d^h}{\Delta}) = O(\frac{n}{\Delta})$ for $d \geq 3$ and $O(\log \Delta)$-competitive for $d = 2$. Our algorithm is simple: it just makes successive tours of the tree. To not leave jobs close to the origin hanging for too long, we make tours on T_1 first, then T_2 and so on until we tour the whole tree repeatedly. While this algorithm might seem to have a large competitive ratio, it is the best we can do for fully-prompt algorithms. Specifically, we prove that any fully prompt algorithm must be at least $\Omega(\frac{TSP_{tour}}{\Delta})$ where TSP_{tour} is the optimal value for the TSP path problem on the metric space. First, we prove the lower bound and then we formalize the algorithm.

We are to present an instance of jobs that prevents the algorithm from achieving any competitive ratio strictly bellow $\Omega(\frac{TSP_{tour}}{\Delta})$, where we remind that TSP_{tour} is the optimal value for the TSP path problem and Δ is the diameter. Jobs arrive at consecutive batches, one batch of jobs for each point in the metric space at time $t = 1$. The sequence stops if a batch of jobs is scheduled to be served at time greater or equal than TSP_{tour}. We choose an arrival time of $t = 1$ to eschew paradoxes that arise from having an optimal completion time of zero for the job(s) at the origin. The number of jobs arriving at a point in the metric space will be chosen as to achieve the desired lower bound. First, we prove the following lemma:

Lemma 7.1. There exists at least one point in the metric space which is served at time $t \geq TSP_{tour}$, where TSP_{tour} is the optimal value for the TSP path problem.

Proof. Since the length of the optimal tour is TSP_{tour}, there has to be at least on point in the space which is reached at time at least TSP_{tour}. Suppose this is not true; then the last job served will be done so at time $r < TSP_{tour}$. Since no job's completion time reaches TSP_{tour} then all jobs (for all metric space points) will be served before r. This is a TSP tour with length $r < TSP_{tour}$, a contradiction. □

Theorem 7.3. For a bounded metric space $\mathcal{M}$, a fully-prompt algorithm for the TRP is at least $\Omega(\frac{TSP_{tour}}{\Delta})$, where TSP_{tour} is the optimal value for the TSP path problem on $\mathcal{M}$.

Proof. Using lemma 7.1, we are to prove the main lower bound for general metric spaces. We use the number of jobs in the batch to appropriately weigh the job with which the sequence stops. Since we do not know which job this is, as our sequence is determined before the algorithm makes its choice, we give a sufficient number of jobs at each step. This sufficient number can be determined as follows. Assume the algorithm so far has latency cost ALG and the optimal schedule has latency OPT. By the triangle inequality, the new batch's, j, amount of jobs can be chosen sufficiently high to make the optimal algorithm serve it by time $c_j^* = d(\mathcal{O}, j) + 1$, where $d(\mathcal{O}, j)$ is the distance from the origin. Subsequently, the new competitive ratio becomes $\frac{ALG+n_j \cdot c_j}{OPT+n_j \cdot c_j^*}$. The number of jobs n_j is set sufficiently large and the new ratio asymptotically approaches $\frac{c_j}{c_j^*}$. This can be done in each iteration of batches arriving until the sequence stops. When the sequence stops at batch k we have that $c_k \geq TSP_{tour}$ while $c_k^* \leq d(\mathcal{O}, j) + 1$. This proves a competitive ratio arbitrarily close to $\frac{TSP_{tour}}{\Delta+1}$. □

The above result gives a lower bound of $\Omega(\frac{n}{\Delta})$ for a 2-HST with maximum degree $d > 2$, $\Omega(\log \Delta)$ for a 2-HST with $d = 2$ and a $\Omega(1)$ lower bound for line metrics. All of these results are tight. For the former results, a simple tour spanning all points and returning

every level to the origin suffices while for the latter, the algorithm by Feuerstein and Stougie [25] achieves an upper bound of $O(1)$.

Let us describe the algorithm for a 2-HST:

Algorithm 7.4
For a complete tree metric:

1. Wait at the origin until $t = 1$.
2. For $x = 1, 2, ..., h$ make tours on T_x, returning to the origin between tours.
3. Make successive tours on the whole tree (T_h).

Theorem 7.4. Algorithm 7.4 is $O(\frac{d^h}{\Delta})$-competitive for $d > 2$ and $O(\log \Delta)$-competitive for $d = 2$ for the online TRP.

Proof. This algorithm runs a fixed route on the tree. The algorithm's main source of delay is when a job that is far from the origin arrives and no other jobs arrive. In this case, the job would be handled by the optimal algorithm by $c_j = \Delta$. For a job in T_x, it will have to wait until $\sum_{i=1}^{x} \left(|T_i| + 2^{i-1}\right)$, where the 2^{i-1} terms stem from the servers return to the origin. Calculating the above sum, we have that:

$$\begin{aligned}
\sum_{i=1}^{x} \left(|T_i| + 2^x\right) &= \sum_{i=1}^{x} \left(\sum_{j=0}^{i-1} d^{i-j} 2^j \right) + \sum_{i=1}^{x} 2^{i-1} \\
&= \sum_{i=1}^{x} (d^i + 2d^{i-1} + ... + 2^{i-1}d) + 2^x - 1 \\
&= \sum_{i=1}^{x} d^i (2^{x-i+1} - 1) + 2^x - 1 \\
&= (d^x + 3d^{x-1} + 7d^{x-2} + ... + (2^x - 1)d) + 2^x - 1 \qquad (7.4)
\end{aligned}$$

The term of (7.4) is $O(d^x)$ for $d > 2$ and $O(xd^x)$ for $d = 2$. This means that a job located in T_x but not in T_{x-1} will have completion time $c_j = O(d^x)$ for $d > 2$. The earliest the job could be served would be at its distance from the origin, i.e., $c_j^* \geq 2 \cdot (1+2+4+...+2^x) = \Omega(2^x)$. For $d > 2$, the competitive ratio is $\frac{c_j}{c_j^*} = O(\frac{d^h}{2^h}) = O(\frac{d^h}{\Delta})$, while for $d = 2$ the competitive ratio is $O(h) = O(\log \Delta)$. In both cases, the ratios are of the form $O(\frac{TSP_{tour}}{\Delta})$ and our algorithms are optimal for the tree metric. However, if we are to embed a general space into a tree metric, we would suffer by the $\log n$ factor. □

Furthermore, we note that the same algorithm is also optimal for the Dial-A-Ride Problem with infinite capacity.

Theorem 7.5. Algorithm 7.4 is $O(\frac{d^h}{\Delta})$-competitive for $d > 2$ and $O(\log \Delta)$-competitive for $d = 2$ for the online DARP with infinite capacity.

Proof. Let s_j, g_j be the source and destination of request j respectively. Let $r = \max\{d(o, s_j), d(o, g_j)\}$. Obviously, $c_j^* \geq r$. We note the smallest T_i tree that s_j is contained by T_s and T_g the smallest tree that g_j is contained. We analyze three cases:

- $s < g$: The server picks up the request first and delivers it by $\sum_{i=1}^{g} \left(|T_i| + 2^{i-1}\right)$. Also, $r \leq 2^g$. As in theorem 7.4, $c_j = O(d^g)$ and $c_j^* \geq 2^g$. This gives a competitive ratio of $O(\frac{d^g}{2^g})$. For the case of a binary tree, $c_j = O(g2^g)$ and the competitive ratio is $O(g)$.
- $s > g$: Now the server picks the request up by time $\sum_{i=1}^{s} \left(|T_i| + 2^{i-1}\right)$. He then returns to the origin, and reaches the destination after a further $|T_g|$. Thus, $c_j \leq \sum_{i=1}^{s} \left(|T_i| + 2^{i-1}\right) + 2^s + |T_g|$. This value is $O(d^s)$ for $d > 2$ and $O(s2^s)$ for $d = 2$. Again, $r \leq 2^s$ and the competitive ratio is $O(\frac{d^s}{2^s})$ for $d > 2$ and $O(s)$ for binary trees.
- $s = g$: This case is in accordance with the aforementioned two. If we reach the destination first, the delivery of the job will happen on the next tour, while if we reach the source first, the completion will happen on the same tour. In any case, the bound is the same as before.

A point is furthest from the origin when it is in the T_h tree and as such, $s, g \leq h$. The above ratios are maximized when s and g are close to h and the final competitive ratio is $O(\frac{d^h}{2^h})$ for $d > 2$ and $O(\log \Delta)$ for $d = 2$. In both cases, this is the same as $O(\frac{TSP_{tour}}{\Delta}) = O(\frac{|T_h|}{\Delta})$. □

Expanding on the above way of thinking, we argue that the algorithm in [25] is also fully-prompt. The algorithm therein, follows a criss-cross movement between points $(-2)^i$ to serve jobs on a line metric. This achieves a 9-competitive algorithm for the TRP and a 15-competitive algorithm for the DARP with infinite capacity. This algorithm follows a fixed schedule and thus is fully-prompt. Again, the lower bound given by theorem 7.3 is $\frac{TSP_{tour}}{\Delta} \leq \frac{2 \cdot \Delta}{\Delta} = 2$. This means that their algorithm is fully prompt and its competitiveness is tight up to a constant multiplicative factor, among other fully-prompt algorithm on line metrics.

Keep in mind that the above algorithms are obviously truthful. Since the server follows a fixed path, a job's declared weight has no effect on its completion time and thus its utility. More generally, even if a job created phantom presences in points near it to beguile the server to come earlier towards its location, this again would have no effect on the fixed path of the server.

List of Figures

Bibliography

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 32–43, 1999.

[2] Afrati, Foto, Cosmadakis, Stavros, Papadimitriou, Christos H., Papageorgiou, George, and Papakostantinou, Nadia. The complexity of the travelling repairman problem. *RAIRO-Theor. Inf. Appl.*, 20(1):79–87, 1986.

[3] Eric Angel, Evripidis Bampis, Fanny Pascual, and Nicolas Thibault. Truthfulness for the sum of weighted completion times. volume 9797, pages 15–26, 08 2016.

[4] Aaron Archer and David P. Williamson. Faster approximation algorithms for the minimum latency problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, page 88–96, USA, 2003. Society for Industrial and Applied Mathematics.

[5] Sanjeev Arora and George Karakostas. Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, 32, 04 1999.

[6] Sanjeev Arora and George Karakostas. A $2 + \varepsilon$ approximation algorithm for the k mst problem. *Mathematical Programming*, 107:491–504, 07 2006.

[7] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. *Algorithmica*, 47, 06 2003.

[8] K.R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New York, 1974.

[9] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193, 1996.

[10] Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu. Traveling repairperson, unrelated machines, and other stories about average completion times, 02 2021.

[11] Marcin Bienkowski and Hsiang Hsuan Liu. An improved online algorithm for the traveling repairperson problem on a line. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[12] A. Blum, S. Chawla, D.R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 46–55, 2003.

[13] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, page 163–171, New York, NY, USA, 1994. Association for Computing Machinery.

[14] Vincenzo Bonifaci and Leen Stougie. Online k-server routing problems. volume 45, pages 83–94, 01 2007.

[15] Allan Borodin and Ran El-Yaniv. Online computation and competitive analysis. 1998.

[16] P. Brucker, J. Lenstra, and A. Kan. Complexity of machine scheduling problems. 1975.

[17] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 36–45, 2003.

[18] George Christodoulou, Elias Koutsoupias, and Annamária Kovács. On the nisan-ronen conjecture. *CoRR*, abs/2011.14434, 2020.

[19] George Christodoulou, Elias Koutsoupias, and Angelina Vidali. A lower bound for scheduling mechanisms. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1163–1170, USA, 2007. Society for Industrial and Applied Mathematics.

[20] Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[21] José R. Correa and Michael R. Wagner. Lp-based online scheduling: from single to parallel machines. *Mathematical Programming*, 119:109–136, 2009.

[22] Shahar Dobzinski and Ariel Shaulker. Improved lower bounds for truthful scheduling. *CoRR*, abs/2007.04362, 2020.

[23] Alon Eden, Michal Feldman, Amos Fiat, and Tzahi Taub. Prompt scheduling for selfish agents. *CoRR*, abs/1804.03244, 2018.

[24] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004. Special Issue on STOC 2003.

[25] Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001. On-line Algorithms '98.

[26] Yiannis Giannakopoulos, Alexander Hammerl, and Diogo Poças. A new lower bound for deterministic truthful scheduling. *CoRR*, abs/2005.10054, 2020.

[27] Michel X. Goemans and Jon M. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82:111–124, 1996.

[28] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[29] Leslie Hall, David Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 09 1996.

[30] Han Hoogeveen and Arjen Vestjens. Optimal on-line algorithms for single-machine scheduling. pages 404–414, 01 1996.

[31] Dawsen Hwang and Patrick Jaillet. Online scheduling with multi-state machines. *Networks*, 71, 12 2017.

[32] Patrick Jaillet and Michael Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transportation Science*, 40:200–210, 05 2006.

[33] Theodore Jr and Theodore Groves. Incentives in teams. *Econometrica*, 41:617–31, 02 1973.

[34] John Kagel, Ronald Harstad, and Dan Levin. Information impact and allocation rules in auctions with affiliated private values: A laboratory study. *Econometrica*, 55:1275–1304, 02 1987.

[35] Elias Koutsoupias, Christos H. Papadimitriou, and Mihalis Yannakakis. Searching a fixed graph. In *ICALP*, 1996.

[36] Elias Koutsoupias and Angelina Vidali. A lower bound of $1 + \phi$ for truthful scheduling mechanisms. In Luděk Kučera and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2007*, pages 454–464, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[37] Sven Krumke, Willem Paepe, Diana Poensgen, and Leen Stougie. Erratum to "news from the online traveling repairman" [theoret. comput. sci. 295 (1–3) (2003) 279–294]. *Theoretical Computer Science*, 352:347–348, 03 2006.

[38] Sven O. Krumke, Willem E. de Paepe, Diana Poensgen, and Leen Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1):279–294, 2003. Mathematical Foundations of Computer Science.

[39] Shengwu Li. Obviously strategy-proof mechanisms. *American Economic Review*, Forthcoming.

[40] Nicole Megow and Andreas S. Schulz. On-line scheduling to minimize average completion time revisited. *Oper. Res. Lett.*, 32:485–490, 2004.

[41] Edward Minieka. The delivery man problem on a tree network. *Ann. Oper. Res.*, 18(1–4):261–266, May 1990.

[42] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 35, 11 2000.

[43] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1):166–196, 2001.

[44] Alfredo Olaverri, Pedro Jodrá, and Javier Tejel. A note on the traveling repairman problem. *Networks*, 40:27–31, 08 2002.

[45] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[46] Cynthia Phillips, Clifford Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:82–199, 1995.

[47] Stephen Rassenti, Vernon Smith, and R.L. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, 13(2):402–417, 1982.

[48] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. j acm 23(3): 555-565. *J. ACM*, 23:555–565, 07 1976.

[49] Andreas S. Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM J. Discret. Math.*, 15:450–469, 2002.

[50] René Sitters. The minimum latency problem is np-hard for weighted trees. In William J. Cook and Andreas S. Schulz, editors, *Integer Programming and Combinatorial Optimization*, pages 230–239, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[51] René Sitters. Efficient algorithms for average completion time scheduling. In Friedrich Eisenbrand and F. Bruce Shepherd, editors, *Integer Programming and Combinatorial Optimization*, pages 411–423, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[52] René Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. *CoRR*, abs/1307.4289, 2013.

[53] René Sitters. Approximability of average completion time scheduling on unrelated machines. *Mathematical Programming*, 161(1):135–158, Jan 2017.

[54] M. Skutella. Semidefinite relaxations for parallel machine scheduling. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, pages 472–481, 1998.

[55] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[56] William S. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[57] Bo Xiong and Christine Chung. Completion time scheduling and the wsrpt algorithm. In A. Ridha Mahjoub, Vangelis Markakis, Ioannis Milis, and Vangelis Th. Paschos, editors, *Combinatorial Optimization*, pages 416–426, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.