Scheduling problems on Stars and Paths

by

Arnoosh Golestanian

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In this thesis, we consider scheduling problems in which jobs need to be processed through a (shared) network of machines according to their given paths. Formally, we are given a graph $G(V, E)$ where the edges E represent the machines. We are also given a set of jobs J and a path of edges for each of them showing that the job needs to be processed on those edges in that order. Each job can be moved to the next machine only if it has been fully processed by all the previous machines in the path. Moreover, each job takes 1 unit of time to be processed on each machine and once a machine starts processing a job, it cannot process any other jobs. Our goal is to find a schedule with minimum total completion time.

We consider two closely related scheduling problems, Star Scheduling with Unit Processing time (*SSUP*) and Generalized Path scheduling with Unit processing time (*GPUP*). As it is clear from their terminology, in SSUP the network of machines is a star and in GPUP it is a path.

The most important contribution of this thesis is a 1.796-approximation for the SSUP problem, described in Chapter 2. To achieve this we partition jobs into smaller subsets, in each subset the number of jobs that share a machine is less than a specific number which is increasing geometrically. We also prove that for the case that jobs cannot have a delay in the middle of their processing, the problem is APX-hard.

In Chapter 3, we consider GPUP problem and prove that in polynomial time one can find a schedule which minimizes the total completion time. Our analysis for the algorithm is new and is much simpler than the previous analysis.

To my beloved parents and husband.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Dr. Mohammad R. Salavatipour for his continuous support of my Master of Science study and research. I would like to thank him for his patience, motivation and for his brilliant guidance and assistance. You have shown me how I can achieve more and not to hesitate along the way of research. My special thanks to Dr. Zachary Friggstad for his great support. Without him this work would not have been possible.

I want to thank my former supervisor, Dr. Babak Farzad, for inspiring me to pursue my interest in computer science. The experience that I had with him as a masters student has been my first and biggest motivation to pursue research in algorithms.

I am thankful to my family: my husband, my parents and my brother, for supporting me spiritually throughout my life.

Last but not least, I would like to thank the University of Alberta Department of Computing Science for financially supporting me over the last two years.

Contents

List of Figures

Chapter 1

Introduction

Scheduling problems are well-studied in computer science, with application in manufacturing, resource allocation, service planning, and many other areas. In a scheduling problem, there is a collection J of n jobs and a set M of m machines. There are many different variants of scheduling problems. Here we consider those ones that are closely related to the ones considered in this thesis.

In the classical job shop scheduling problem, we are given a sequence of μ_j operations $O_{1j}, O_{2j}, \ldots, O_{\mu_j j}$ for each job $j \in J$. Operation O_{ij} takes P_{ij} time units without interruption on machine $m_{ij} \in M$. We also assume all jobs are available at time zero. Let C_j be the completion time of job j in a schedule. The typical objective functions are minimizing the makespan $C_{\max} = \max_{j \in J} C_j$, and minimizing the *total completion times* $\sum_{j \in J} C_j$. For the case that we are given weights $w_j \geq 0$ for each job $j \in J$, our goal is to minimize the total weighted completion time $\sum_{j \in J} w_j C_j$.

A feasible schedule specifies for each job the times that it must be processed on different machines such that each machine processes at most one job at any time and each job is processed by at most one machine at any time. The goal is to find a feasible scheduling of the jobs on the machines to optimize an objective function.

In Section 1.1 we introduce graph theoretical definitions and notations that are used in this thesis. For undefined definitions and notations, we use [We01] as our reference. In Section 1.2, we define some scheduling problems that are closely related to the ones considered in this thesis. Next, in Section 1.3, we give a simple introduction to approximation algorithms. We will then discuss related work in the literature in Section 1.4. Finally, in Section 1.5, we will mention the problems that we considered in this research, and the new results we obtain. Most of the results of this thesis appeared in [FGK+17].

1.1 Graph Theoretic Fundamentals

A *graph* G is a triple consisting of a vertex set $V(G)$, an edge set $E(G)$, and a relation that associates with each edge two vertices (not necessarily distinct) called its *endpoints*. We will denote e by (u, v) if and only if $u \in V$ and $v \in V$ are two endpoints of edge e; also, we say vertices u and v are *adjacent* and e is *incident* to u and v. Similarly, we say edges e_1 and e_2 are *adjacent* if and only if they are incident to a common vertex. A *loop* is an edge whose endpoints are equal. If edges have the same pair of endpoints, they are called *multiple* edges. A *simple* graph is a graph having no loops or multiple edges. However, in a *multigraph* we are allowed to have loops and multiple edges.

For every vertex $v \in V$, let $\delta(v)$ be the set of edges incident to vertex v and let $deg(v)$ be the number of incident edges to v, $|\delta(v)|$. For a graph G, let Δ and δ be the maximum and minimum degree of its vertices, respectively. We denote the set of vertices adjacent to v (*neighbours* of v) by $N(v)$. A *clique* in a graph G is a set of vertices such that every two of them are adjacent. A set of edges of graph G is a *matching* if no two edges of the set share an endpoint. A matching of a graph G is said to be *perfect* if every vertex of G is an endpoint of some edge in the matching.

A *k-regular* is a graph whose vertices all have degree k. A *path* is a simple graph whose vertices can be ordered so that two vertices are adjacent if and only if they are consecutive in the list. A *cycle* is a graph with an equal number of vertices and edges whose vertices can be placed around a circle so that two vertices are adjacent if and only if they appear consecutively along the circle. A *walk* is a list $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges such that, for $1 \leq i \leq k$, the edge e_i has endpoints v_{i-1} and v_i. A *u, v-walk* has first vertex u and last vertex v; these are its *endpoints*. A *u, v-path* is a path whose vertices of degree one (its endpoints) are u and v. A graph G is *connected* if it has a u, v-path whenever $u, v \in V(G)$; otherwise, G is *disconnected*. A graph with no cycle is *acyclic*. A *tree* is a connected acyclic graph.

A *(proper) k-vertex colouring* of graph G is a mapping $\psi : V \to \{1, 2, \ldots, k\}$ such that for any two adjacent vertices u and v, $\psi(u) \neq \psi(v)$. Similarly, a *(proper) k-edge colouring* of graph G is a mapping $\psi : E \to \{1, 2, \ldots, k'\}$ such that for any two adjacent edges e_1 and e_2, $\psi(e_1) \neq \psi(e_2)$. The smallest integer k such that G has a k-vertex (edge) colouring is called the *chromatic number* (*edge chromatic number*) of G and is denoted by $\chi(G)$ ($\chi'(G)$).

A graph G is *bipartite* if it has a 2-vertex colouring. A graph $G'(V', E')$ is a *subgraph* of $G(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$, graph G' is a *spanning* subgraph of G. Also, a spanning k-regular subgraph is called a *k-factor*. A graph G is *k-factorable* if we can partition the edges into disjoint k-factors. Let T be any subset of vertices of G; then a subgraph of G is called an *induced subgraph* by T if it is obtained by deleting vertices of set $V - T$ and all edges connected to them from graph G. We define *b-matching* for a given undirected graph G to be a subgraph with maximum degree at most b and minimum degree at least 1. For the case that $b = 2$, we simply denote it by *2-matching*.

A *directed graph* or *digraph* G is a triple consisting of a vertex set $V(G)$, an edge set $E(G)$, and a function assigning each edge an ordered pair of vertices. The first vertex of the ordered pair is the *tail* of the edge, and the second is the *head*; together, they are the endpoints. We say that an edge is an edge *from* its tail *to* its head. If vertex v is a tail (head) of an edge e, we call edge e an *outgoing* (*incoming*) edge of v. Similarly, let $\delta^{in}(v)$ ($\delta^{out}(v)$) be the set of incoming edges (outgoing edges) of vertex v. We define *in-degree* (*out-degree*) of a vertex v to be the number of incoming edges (outgoing edges) of vertex v and denote it by $deg^{in}(v)$ ($deg^{out}(v)$). In a digraph, a *loop* is an edge whose endpoints are equal. *multiple edges* are edges having the same ordered pair of endpoints. A digraph $G(V, E)$ is *regular directed* if and only if for every $v \in V, deg^{out}(v) = deg^{in}(v) = \frac{\Delta}{2}$. The *underlying graph* of a digraph D is the graph G obtained by treating the edges of D as an unordered pairs; the vertex set and the edge set remain the same.

Min sum edge colouring is an edge colouring problem with a different objective function. More formally, in the min sum edge colouring problem of a given graph $G(V, E)$, we want to find a proper edge colouring $\psi : E \to \{1, \ldots, k\}$ minimizing the function $\sum_{e \in E} \psi(e)$. We denote this problem by MSE.

1.2 Scheduling Fundamentals

There are many special cases of job shop scheduling problem studied in the literature. Here we define those ones that are related to our problems. In the *acyclic job shop scheduling problem*, each job has at most one operation per machine. In the *flow shop scheduling problem*, each job has exactly one operation on each machine and all the jobs go through machines in the same order. However, in the *generalized flow* *shop problem* there can be operations with zero processing times.

Another classification of scheduling problems is by the setting of machines. If for each job j p_{ij}'s are independent of machines, they will be equal to p_j and we have the *identical machine* setting. If they are dependent to the speed of machines then $p_{ij} = \frac{p_j}{s_i}$ where s_i is the speed of machine i and we have the *related machine* setting. Otherwise, we have the *unrelated machine* setting. Moreover, in the identical machine setting if all p_j's are equal to 1, we have jobs with *unit processing time.*

One specialization of the job shop scheduling problem is when there is an underlying network of machines. In this setting, we assume we are given a graph $G(V, M)$ where each edge $e \in M$ corresponds to a machine and each job has a specific path showing that it has to go through the corresponding machines in a specific order. If the graph G is a simple path P and the path of each job is P itself, then we get the flow shop problem.

Another variant of scheduling problem is when preemption is allowed (i.e., preemptive scheduling problem). *Preemption* of a job or operation means that processing of a job can be paused (for several times) and resumed later (even on another machine).

In the *packet routing problem*, we are given a set of packets and a network. Each packet has a specific origin and destination and it takes one unit of time for each packet to travel an edge. In this problem our goal is to first find a path for each packet between its origin and destination and then to minimize the makespan. Clearly for the case that the network is tree, the first part of the objective function will be redundant.

1.3 Approximation Algorithms

An optimization problem is the problem of finding a feasible solution with optimum objective among the given set of feasible solutions. In the minimization (maximization) problem our goal is to minimize (maximize) the objective value. Most of the scheduling problems are minimization problems. Many optimization problems are NP-hard and it is not possible to solve them optimally and efficiently unless $P = NP$. Instead we try to approximate their optimal solution. An α-approximation algorithm for a minimization problem is an algorithm which finds a feasible solution with cost SOL in polynomial time such that $SOL \leq \alpha OPT$ where OPT is the optimum value. Similarly, an α-approximation algorithm for a maximization problem finds a feasible solution in polynomial time such that $SOL \geq \frac{OPT}{\alpha}$. We call α the *approximation ratio* or *performance ratio.*

For the NP-hard optimization problems the best result that we can find is a $(1+\epsilon)$-approximation for an arbitrary constant $\epsilon > 0$. Formally, we say that a problem has a *Polynomial Time Approximation Scheme (PTAS)* if and only if it has a polynomial time $(1+\epsilon)-$approximation algorithm for every constant $\epsilon > 0$. We define class *APX* to be the class of all problems with a constant approximation algorithm. Similarly, class *PTAS* is the class of all problems that have PTAS.

Let ϕ and ϕ' be two optimizations problems. We say that there is a *PTAS* reduction from problem ϕ to ϕ' if for any instance I of problem ϕ and any fixed $\epsilon > 0$:

1. There exists an algorithm $\mathcal{A}$ and function $c : \mathbb{R}^+ \to \mathbb{R}^+$ such that $\mathcal{A}$ returns an instance $I' = \mathcal{A}(I, \epsilon)$ of ϕ' in polynomial time and if I is feasible then I' will be feasible.

2. A solution of instance I' with cost at most $(1+c\epsilon).OPT_{\phi'}(I')$ corresponds to a solution of instance I with cost at most $(1+\epsilon).OPT_{\phi}(I)$.

We say that an optimization problem is *APX-Hard* if there is a *PTAS* reduction from every other problem $\phi' \in APX$ to ϕ. In addition, if $\phi \in APX$, we say that ϕ is APX-complete.

For a better understanding of approximation algorithm we show how to get a simple 2-approximation algorithm for scheduling problem on identical parallel machines as an example. In this problem we are given m parallel identical machines and n jobs such that each job needs a single operation which can be done on any of machines. Our goal is to minimize the makespan.

For a given schedule let d_i be the load of machine i which is a total processing time of jobs running on machine i for every $i \in \{1, \ldots, m\}$. It is clear that

$$\sum_{i=1}^{m} d_i = \sum_{i=1}^{n} p_i. \tag{1.1}$$

In the following simple algorithm we assign each job to the machine that has the minimum load so far.

```
   **Input   :** A set of jobs and a set of parallel machines
   **Output:** A scheduling of the jobs
1  Consider an arbitrary ordering for the jobs.
2  Schedule first $m$ jobs on machines in this order.
3  Whenever a machine is freed assign the next job to it.
```

Algorithm 1: Algorithm for minimizing makespan of scheduling problem on parallel identical machines.

Theorem 1.1. *For the makespan objective function on parallel identical machines Algorithm 1 achieves an approximation ratio of* 2.

Proof. It is clear that the optimum value, denoted by OPT, is at least the average load of a machine and the largest processing time, i.e.,

$$OPT \geq \frac{1}{m}\sum_{i=1}^{m} p_i \tag{1.2}$$

$$OPT \geq \max_i p_i. \tag{1.3}$$

Let machine i' be the machine with maximum load and job j' be the last job on machine i' in the solution of Algorithm 1. Also, let ALG be the cost of Algorithm 1, then clearly $ALG = d_{i'}$. We denote the start time of job j' on machine i' by $t_1^{j'}$ meaning that

$$d_{i'} = t_1^{j'} + p_{j'}. \tag{1.4}$$

According to Algorithm 1 all machines are busy until $t_1^{j'}$; otherwise j' would have been assigned to another machine at an earlier time; so, for every $i \in \{1, \ldots, m\}, t_1^{j'} \leq d_i$. By using (1.1) and (1.2) we get that

$$t_1^{j'} \leq \frac{1}{m}\sum_{i=1}^{m} p_i \leq OPT.$$

Moreover, by (1.3) and (1.4) the cost of Algorithm 1 will be at most $2OPT$ meaning that Algorithm 1 is a 2-approximation as claimed.

□

1.4 Related Work

Here we review the previous work on the problems we consider in this thesis. The trivial lower bound lb that is used in many related works are the congestion and dilation lower bounds. We define the *dilation* D to be $\max_{j\in J} D_j$ where D_j is a total processing time of job j regardless of the presence of other jobs. We also define the *congestion* C to be the maximum number of time units requested by all jobs on each machine. Then clearly $lb = max\{C, D\}$ is a lower bound on the makespan. Currently the best approximation algorithm known for the job shop scheduling by Shmoys et al. [SSW94] has performance ratio $O((\log lb)^2/\log\log lb)$ for minimizing the makespan. For the case of preemptive job shop problem Bansal et al. [BKS06] gave an algorithm with (better) performance ratio of $O(\log m/\log\log m)$ for the same objective function.

In [LQSY06, QS02] the authors show that any approximation algorithm with ratio ρ w.r.t. the trivial lower bound lb for makespan, can be used to obtain a $2e\rho$ approximation for total completion time. Feige and Scheideler [FS02] present an algorithm with makespan $O(lb \log lb \log\log lb)$ for the acyclic job shop problem. By giving a family of instances in which the makespan is $\Omega(lb \log lb/\log\log lb)$, they derived that this upper bound is nearly tight. Also, the approximation in [FS02] is still the best known result for the flow shop scheduling problem for both makespan and total completion time (according to [LQSY06, QS02]). However, for the generalized flow shop problem Mastrolili and Svensson [MS11] prove a hardness of approximation ratio of $\Omega(\log^{1-\epsilon} lb)$.

For the flow shop problem with identical machines, or the *proportionate flow shop*, Shakhlevich et al. [SHP98] present a polynomial time algorithm with running time $O(n^2)$ for the weighted total completion time objective. For the case that machines have unequal speed (related machines), Hou and Hoogeveen [HH03] have shown that if there are three machines and the second machine is the slowest, the problem can be solved optimally in pseudo polynomial time.

In the job shop scheduling if all jobs have unit processing time, the problem reduces to the packet routing problem. For this, the celebrated result of Leighton et al. [LMR94] shows that there is a schedule of length $O(lb)$. In [LMR99] they presented an algorithm that finds a schedule with makespan $O(lb)$ in polynomial time. Recently, in [HS13] the authors show the existence of a schedule of length

$7.26.(C+D)$ (non-constructive) and present an algorithm that finds a schedule with length $8.84.(C+D)$.

For the case of packet routing on in-trees or out-trees (directed trees in which the in-degree of each vertex is at most one, or out-degree is at most one, respectively), Leung et al. in [LTY96] have shown that the schedule in which each machine processes the job that has the longest distance to go at each time step finds the optimum solution for makespan. Based on this, in [PSW09] Peis et al. observe that one can get a 2-approximation for makespan on undirected trees. For this, they convert a tree into a rooted tree and split the problem into two subproblems, first one is the packet routing problem on an in-tree and second one is on an out-tree. For directed trees they get a better result. They show that one can get a schedule of length $C + D - 1$ which is the best bound in terms of C and D; they provide a tight example of routing C packets on a path of length D with vertices $v_1, \ldots, v_D$ where the origin (destination) of each packet is v_1 (v_D).

For the special case of packet routing problem on a path when all packets go from left-to-right the authors in [AB+14, KNSM14] show that the schedule in which each machine processes the job with shortest distance to go at each time step gives the optimum solution for total completion time. Moreover, Kowalski et al. in [KNSM14] show that the furthest-to-go strategy gives the optimum solution for makespan.

Another well studied special case of packet routing problem is when the network of machines forms a grid graph. In [PSW10] Peis et al. have shown that if the start and destination of two packets are different one can get a 2-approximation.

Im and Moseley [IM15] consider a new variation of scheduling problems with rooted tree network. In their model, the root of the tree is a job distribution center, the leaves are machines, and the interior nodes are routers which are identical. Each job needs to be routed from the root to the assigned leaf using router along the path. Each link (edge) can only move one job at each time. They consider an online scheduling (in which we do not know the set of jobs and processing times beforehand) of two cases, when the machines are identical and when they are related. For both cases, they present constant factor approximations for makespan. Bhattacharya et al. [BKM14] have looked at coordination mechanism for routing problems on a tree. Recently, Friggstad et al. in [FGK+17] showed that for the case of stars with identical machines one can get a 7.279-approximation for minimizing the total completion time. Moreover, they showed that for both makespan and total completion time, there are polynomial time $O(\min\{\log n, \log m, \log p_{max}\})$-approximation algorithms when the network of machines forms a tree.

Interestingly we can view some variants of scheduling problems from colouring problems point of view. In the *biprocessor scheduling problem* with minimizing total completion time objective, we are given a collection of unit processing time jobs such that each job needs a simultaneous use of two specific machines; moreover, we are given a multigraph for the network of machines. This problem is equivalent to the min sum edge colouring problem of the given multigraph. In [HKS11] Halldórsson et al. present a 1.8298-approximation for the min sum edge colouring problem. Prior to their work the best known ratio was 2 according to [BBH+98]. In [M09] it has been shown that the min sum edge colouring problem is APX-hard even for the case of bipartite graphs.

1.5 New Results

In the following, we define the problems we consider in this thesis.

Star Scheduling with Unit Processing time (SSUP). In this scheduling problem, we are given a set of jobs J and machines M where $|J| = n$ and $|M| = m$. Moreover, a star $G(V, M)$ with center r is given for the network of machines. We are also given a path $P_j = (s_j, t_j)$ for every $j \in J$ where s_j, t_j are two edges of G showing that job j should process distinct edges s_j and t_j in turn. We can assume each job j is located at some leaf (incident to s_j) and must finish at another leaf (incident to t_j). For an illustration see Figure 1.1. It is clear that SSUP is a special case of the packet routing problem.

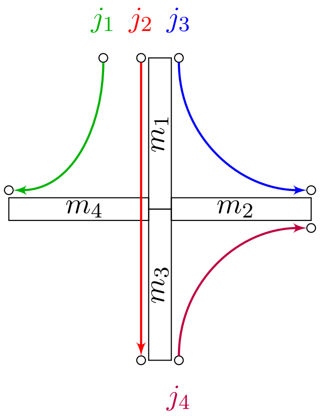


Figure 1.1: An example for star scheduling problem with unit processing time. The given set of jobs and machines are $J = \{j_1, j_2, j_3, j_4\}$ and $M = \{m_1, m_2, m_3, m_4\}$. The given paths are $P_1 = (m_1, m_4)$, $P_2 = (m_1, m_3)$, $P_3 = (m_1, m_2)$, $P_4 = (m_3, m_2)$.

Generalized Path scheduling with Unit Processing time(GPUP). In this problem, we are given a path $P(V, M)$ for the network of machines where $M = \{1, \ldots, m\}$ is the set of machines. We are also given a set of jobs J with cardinality n where each job j is specified by a subpath P_j where P_j is the sequence of edges from index s_j to t_j. Each job $j \in J$ must spend one unit of processing time on each machine in P_j. Moreover, each job will move to the next machine only if it has been fully processed by all of the previous machines in P_j. Unlike the flow shop problem, in this problem the starting machine and finishing machine for each job are not necessarily machines 1 and m. For an illustration see Figure 1.2.

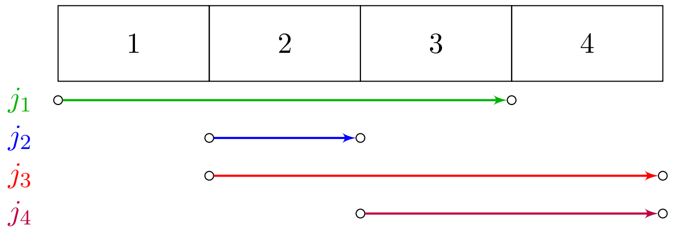


Figure 1.2: An example for generalized path scheduling problem with unit processing time. The given set of jobs and machines are $J = \{j_1, j_2, j_3, j_4\}$ and $M = \{1, 2, 3, 4\}$. The given paths are $P_1 = (1, 2, 3)$, $P_2 = (1)$, $P_3 = (2, 3, 4)$, $P_4 = (3, 4)$.

In Chapter 2, we consider the SSUP problem and some variation of it. In the beginning of the chapter we present a simple 2-approximation and by providing a tight example we show that the ratio of this algorithm cannot be improved. Next we present the following main result:

Theorem 1.2. *For minimizing the total completion time of SSUP problem there is a* 1.796*-approximation algorithm.*

One variation of SSUP is when the given auxiliary graph is regular directed. For this case we present a polynomial time algorithm which finds disjoint cycle-covers in each iteration. Another variation of this problem is when we are not allowed to have any intermediate delay for jobs in the center. We prove that this variation is APX-hard. In Chapter 3, we consider the GPUP problem and prove the following result.

Theorem 1.3. *For the GPUP problem, there is a polynomial time algorithm to compute a schedule with minimum total completion time.*

This results has been mentioned in [KNSM14, AB+14], but we used a much simpler analysis.

Chapter 2

Star Scheduling with Unit Processing Time

In this chapter we consider the star scheduling problem with identical machines and unit processing time jobs and our goal is to minimize the total completion time of jobs. Recall that in the SSUP problem we are given a set J of n jobs and a set M of m machines. Also, a star $G(V, M)$ with star r is given for the network of machines. Let digraph $H(V', E')$ be the auxiliary graph of $G(V, M)$ in which vertices correspond to machines and edges correspond to jobs in J where each edge $(s_j, t_j) \in E_H$ shows that the given path for job j in graph G is $\{s_j, t_j\}$, Figure 2.1. In this graph representation, $|E'| = |J| = n$ and $|V'| = |M| = m$. Note that in this problem the auxiliary graph $H(V', E')$ can be a multigraph.

When we schedule a job j that is supposed to run on machines u, v where $e = (u, v) \in H$ with tuple (t_1^e, t_2^e) it means that we start processing job j in graph G at time t_1^e on its first machine and at time t_2^e on its second machine. When it is clear from context, we drop parameter e from tuple, and simply assign tuple (t_1, t_2) to edge e.

Throughout this chapter, we use ALG and OPT to denote the cost of the proposed algorithm in that section and the optimum value of the star scheduling problem.

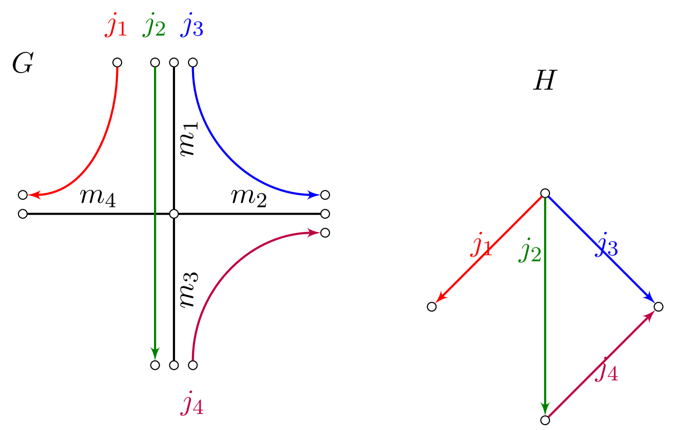


Figure 2.1: Two graph representations of star scheduling problem that is shown in Figure 1.1, an original graph G and an auxilary graph H

2.1 A Simple 2-Approximation

Let us define the following sets for each edge $e \in E$:

$$Q_e = \{j | s_j = e\} \qquad R_e = \{j | t_j = e\} \qquad L_e = Q_e \cup R_e.$$

Since graphs G and H are related, we get the following equations for every $e \in E$ and the corresponding vertex $v \in V'$:

$$deg_H(v) = deg_H^{out}(v) + deg_H^{in}(v) = |Q_e| + |R_e| = |L_e|. \tag{2.1}$$

2.1.1 Algorithm

The algorithm simply tries to send all jobs to the center first and when an edge has processed all the jobs traveling to the center, it will process the jobs that have this edge as their second machine. In the following we bring the formal representation of our algorithm. Assume that each edge $e = (u, r)$ in graph G has two buffers $b_e(u)$ and $b_e(r)$ at its endpoints where $b_e(u)$ will hold jobs that are starting from u and want to cross e; similarly, $b_e(r)$ will hold jobs that arrive at r and want to cross e.

```
Input  : The original graph $G$
Output: A scheduling of the jobs
1  $J' \leftarrow J$
2  while $J' \neq \emptyset$ do
3      foreach edge $e = (u, r)$ in original graph $G$ do
4          if $b_e(u) \neq \emptyset$ then
5              process the first job in $b_e(u)$ and pass it to its next buffer;
6          end
7          else if $b_e(r) \neq \emptyset$ then
8              process the first job in $b_e(r)$;
9          end
10     end
11 end
```

Algorithm 2: Simple approximation algorithm for the star scheduling problem with min-sum objective and unit processing time jobs.

2.1.2 Analysis

In this section we will prove Theorem 2.1.

Theorem 2.1. *Algorithm 2 is a 2-approximation for the star scheduling problem with unit processing times.*

Lemma 2.2. $OPT \geq \sum_{e \in E} \dfrac{|L_e|(|L_e|+1)}{4} + \dfrac{n}{2} = \sum_{v \in V'} \dfrac{deg_H^2(v)}{4} + n.$

Proof. The total number of jobs that will pass edge e is $|Q_e| + |R_e|$. Each of them can pass e one at a time. Assume an arbitrary ordering for these jobs. So, the k'th job will spend $k-1$ units of time in the queue and 1 unit of time for crossing edge e where $k \in \{1, \ldots, |Q_e| + |R_e|\}$. Let $f_e(j)$ be the time that edge e finishes processing job j in the optimal solution for every $e \in E, j \in J$. As a result,

$$\sum_{j \in L_e} f_e(j) \geq 1 + 2 + \ldots + (|Q_e| + |R_e|) = \frac{(|L_e|)(|L_e|+1)}{2}.$$

Let $comp(s_j)$ and $comp(t_j)$ be the finish time of job j on edges s_j and t_j in the optimal solution, respectively. Note that we can rewrite $\sum_{j \in J}(comp(s_j) + comp(t_j))$ as $\sum_{e \in E} \sum_{j \in L_e} f_e(j)$. Also let C_j^{OPT} be the completion time of job j in the optimal solution, then $OPT = \sum_{j \in J} C_j^{OPT}$. So,

$$\sum_{j \in J}(comp(s_j) + comp(t_j)) = \sum_{e \in E} \sum_{j \in L_e} f_e(j) \geq \sum_{e \in E} \frac{|L_e|(|L_e|+1)}{2}. \tag{2.2}$$

It can be seen that for any integer numbers a and b, if $a - b \geq 1$, then we have:

$$a = \max(a, b) \geq \frac{a + b + 1}{2} \tag{2.3}$$

So, clearly by using Equations (2.2) and 2.3 the following inequalities hold:

$$\begin{aligned}
C_j^{OPT} &= \max\{comp(s_j), comp(t_j)\} \geq \frac{comp(s_j) + comp(t_j) + 1}{2} \\
OPT &= \sum_{j \in J} C_j^{OPT} \geq \frac{n}{2} + \frac{1}{2}\sum_{j \in J}(comp(s_j) + comp(t_j)) \\
&\geq \sum_{e \in E} \frac{|L_e|(|L_e| + 1)}{4} + \frac{n}{2} \qquad (2.4)
\end{aligned}$$

Substituting Equation (2.1) in Equation (2.4) and simplifying it, we get the second claimed lower bound for OPT. □

Lemma 2.3 shows an upper bound for the cost of Algorithm 2.

Lemma 2.3. $ALG \leq \sum_{j \in J} \frac{|Q_{s_j}|}{2} + |Q_{t_j}| + \frac{|R_{t_j}|}{2}$.

Proof. According to the definition of star scheduling problem, job j should first cross edge s_j, and then edge t_j. In the first step of Algorithm 2, there is a queue for jobs in the set Q_{s_j}. After that job j will be in the center and cannot start the queue for passing edge t_j until all jobs of the set Q_{t_j} pass edge t_j. Next there is a queue for jobs in the set R_{t_j}.

On average, each job of set Q_{s_j} should wait for $\frac{|Q_{s_j}|}{2}$ units of time to cross edge s_j. In order to wait for all jobs of set Q_{t_j}, it requires to wait $|Q_{t_j}|$ units of time. Also, on average job j will spend at most $\frac{|R_{t_j}|}{2}$ units of time in the queue of edge t_j. So, for all jobs the time required for sending them to their destination is at most:

$$\sum_{j \in J} \frac{|Q_{s_j}|}{2} + |Q_{t_j}| + \frac{|R_{t_j}|}{2}.$$

□

Now we rewrite each term of the obtained upper bound in Lemma 2.3 in the following forms:

$$\sum_{j\in J} |Q_{s_j}| = \sum_{e\in E} |Q_e|^2 \tag{2.5}$$

$$\sum_{j\in J} |Q_{t_j}| = \sum_{e\in E} |Q_e||R_e| \tag{2.6}$$

$$\sum_{j\in J} |R_{t_j}| = \sum_{e\in E} |R_e|^2 \tag{2.7}$$

Consider the left-hand side of Equation (2.5). Intuitively, if for edge e, $|Q_e| = k$ then we consider these k jobs k times. That is why we can write it in the form of right-hand side. With a similar reason Equation (2.7) is valid. In the left-hand side of Equation (2.6), for all jobs with $t_j = e$ (there are $|R_e|$ number of such jobs), we have considered edge e and each time we add number $|Q_e|$ in the summation. Hence:

$$ALG \leq \sum_{j\in J} \frac{|Q_{s_j}|}{2} + |Q_{t_j}| + \frac{|R_{t_j}|}{2} = \frac{1}{2}\sum_{e\in E}(|Q_e| + |R_e|)^2. \tag{2.8}$$

So, by using Equation (2.8) and Lemma 2.2, Theorem 2.1 holds.

Observation 1. *The following tight example shows that the analysis of Algorithm 2 cannot be improved to get ratio better than* 2.

Consider a star $S(V, M)$ is given for the network of machines such that $M = \{s_1, s_2, \ldots, s_k\} \cup \{t_1, t_2, \ldots, t_k\}$. We are also given a set of jobs J with size qk^2 such that q jobs are using path (s_i, t_j) for every $i, j \in \{1, \ldots, k\}$. One way to schedule these jobs is to send all jobs with disjoint paths simultaneously. Hence, we partition jobs into qk sets with equal size $J = \cup_{i=1}^{k} \cup_{j=1}^{q} J_i^j$ such that jobs of each set have disjoint paths and $|J_i^j| = k$ for every $i \in \{1, \ldots, k\}, j \in \{1, \ldots, q\}$. So, by scheduling jobs of each J_i^j separately we get that,

$$\begin{aligned} OPT &\leq k\,(2 + \ldots + (q+1) + (q+2) + \ldots + (2q+1) + \ldots + ((k-1)q+2) + \ldots + (kq+1)) \\ &= k\left(\frac{(kq+1)(kq+2)}{2} - 1\right) = \frac{qk^2}{2}(kq+3) \end{aligned}$$

Now consider the case that our algorithm chooses to send all jobs that end at t_1 and then all jobs that finish at t_2 and so on up to t_k. For qk number of jobs that are finishing at t_i, we should send each one at a time where $i \in [1, k]$. Note that when we want to send jobs from s_i to t_2 we cannot send them until we send all q jobs from s_i to t_1 meaning that the first job in this set will be finished at time $q+2$. With a similar argument we can write the following for the completion time of the algorithm.

$$\begin{aligned} ALG &= 2 + \ldots + (qk+1) + (q+2) + \ldots + (q(k+1)+1) + \ldots + ((k-1)q+2) + \ldots \\ &+ ((k-1)q + 2 + qk - 1) = \sum_{j=1}^{k} \sum_{h=1}^{qk} (j-1)q + 1 + h = \frac{qk^2}{2}(2qk - q + 3) \end{aligned}$$

So, for large enough k, the following ratio approaches 2.

$$ratio \geq \frac{qk^2(2qk - q + 3)}{qk^2(qk+3)}$$

2.2 An Improved 1.796-Approximation

In this section, we obtain a better algorithm for the same problem. In the proposed algorithm, our goal is to choose a subset of edges in each iteration and schedule them such that the makespan of the corresponding jobs is minimized. The idea of this algorithm is similar to the idea of minimizing latency problem [CGRT03] where they convert a makespan objective function to min-sum objective function. Recall that *b-matching(b)* for graph H is a subgraph of H with maximum degree at most b.

2.2.1 Algorithm

Informally speaking, we partition the edges of E' into q blocks denoted by $E_i, i \in \{1, \ldots, q\}$ where q is the number of iterations of Algorithm 3. Each block contains a maximum b-matching(k_i) such that k_i is an even number and it is increasing geometrically, $k_i = 2\lfloor \frac{c^{i+\alpha}}{2} \rfloor$ where α is chosen uniformly at random from the interval $[0, 1)$ and c is some constant that will be optimized later. So, we further partition each block E_i into $\frac{k_i}{2}$ number of slots each denoted by $E_i^j, i \in \{1, \ldots, q\}, j \in \{1, \ldots, \frac{k_i}{2}\}$.

Following is the formal description of our algorithm. Procedure directed b-matching(b) in Step 6 of Algorithm 3 finds a subgraph with maximum number of edges, whose underlying undirected graph is a maximum size b-matching(b). Also, note that there are known algorithms for finding maximum b-matching(b) in polynomial time (see [CCPS98]).

```
Input  : Auxiliary graph $H$
Output: A scheduling of the edges (jobs)
1  $\alpha \sim U[0,1]$
2  $i \leftarrow 1$
3  $W \leftarrow E'$
4  while $W \neq 0$ do
5      $k_i \leftarrow 2\lfloor \frac{c^{i+\alpha}}{2} \rfloor$
6      $E_i \leftarrow$ directed b-Matching($k_i$)
7      Decompose $E_i$ into $\frac{k_i}{2}$ disjoint directed 2-matchings $E_i^1, E_i^2, \ldots, E_i^{\frac{k_i}{2}}$ (Lemma
         2.4).
8      for $j = 1$ to $\frac{k_i}{2}$ do
9          $Y_i^j \leftarrow$ clockwise edges of $E_i^j$
10         $Z_i^j \leftarrow E_i^j \setminus Y_i^j$
11         if $i = 1$ and $j = 1$ then
12             Schedule edges of $Y_1^1$ with tuple $(1, 2)$.
13             Schedule edges of $Z_1^1$ with tuple $(2, 3)$.
14             $x \leftarrow 3$.
15         end
16         else
17             $A \leftarrow$ available time step at each vertex of $E_i^j$.
18             Schedule edges of $Y_i^j$ with tuple $(A, 1 + x)$.
19             Schedule edges of $Z_1^1$ with tuple $(A, 2 + x)$.
20             $x \leftarrow x + 2$
21         end
22     end
23     $W \leftarrow W \setminus E_i$
24     $i \leftarrow i + 1$
25 end
```

Algorithm 3: Algorithm for minimizing total completion time of unit processing time jobs on stars

In step 6 of the algorithm, we find a maximum size subset of edges that can be scheduled using a bounded number of time steps. In order to schedule these edges, we divide them into a number of smaller sets (slots) such that each vertex will be an endpoint of at most 2 edges, as mentioned in Step 7. In Step 9 and 10 of Algorithm 3, we partition edges of each directed 2-matchings E_i^j into two sets according to their direction: the set of clockwise edges, denoted by Y_i^j and the set of counter clockwise edges, denoted by Z_i^j where $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, \frac{k_i}{2}\}$. In the rest of the algorithm we mentioned how to schedule these slots. In Lemma 2.5 we claim that we can schedule almost all of these slots by using at most 2 new time steps each.

2.2.2 Analysis

In the following lemmas we will show an upper bound for the cost of Algorithm 3.

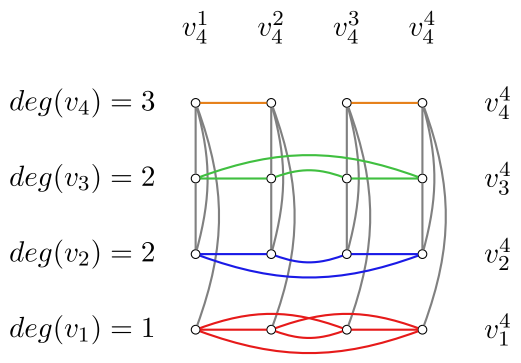


Figure 2.2: An example of how to build a regular graph F' from graph $F(\mathcal{V}', \mathcal{E}')$ as shown in Figure 1.1 where $\mathcal{V}' = \{v_1, v_2, v_3, v_4\}$ with degrees $deg(v_1) = 1, deg(v_2) = 2, deg(v_3) = 2, deg(v_4) = 3$.

Lemma 2.4. *Each b-matching(k_i) can be partitioned into $\frac{k_i}{2}$ disjoint 2-matchings in polynomial time.*

Proof. For an arbitrary $i \in \{1, \ldots, q\}$, let subgraph $H_i(V_i, E_i)$ denote the maximum b-matching(k_i) in graph H with m' number of vertices.

In the following, it will be shown that H_i is a subgraph of some $2d$-regular graph, where $d = \frac{k_i}{2}$. It has been shown that any $2d$-regular graph is 2 factorable [Pe1891] and edges can be partitioned into 2-factors in polynomial time. This means that we can decompose each subgraph H_i into $\frac{k_i}{2}$ 2-matchings.

Consider an ordering $v_1, v_2, \ldots, v_{m'}$ for vertices of V_i such that $deg(v_1) \leq \deg(v_2) \ldots \leq deg(v_{m'})$. Note that according to the definition of b-matching, $deg(v_1) \geq 1$ and $deg(v_{m'}) \leq 2d$.

We make $2d$ copies of subgraph H_i, shown by $H_i^1, \ldots, H_i^{2d}$ and we add an edge between every two copies of vertex v_j for every $j \in \{1, \ldots, m'\}$. We denote this new graph by $F(\mathcal{V}', \mathcal{E}')$. We can see that $\mathcal{E}'$ can be partitioned into m' cliques of size $2d$ and $2d$ copies of E_i. We denote these cliques by $F_1, \ldots, F_{m'}$.

In order to get a $2d$-regular graph, we need to remove some edges from graph F. It is known that every complete graph with an even number of vertices is 1-factorable, i.e., its edges can be partitioned into disjoint perfect matchings in polynomial time [Ha69]. Hence, we can remove $deg(v_j)$ perfect matchings from each clique F_j of graph F, where $j = 1, \ldots, m'$. We call this new graph F'. It can be seen that the degree of each vertex in F' is $2d$ and $H_i \subseteq F'$, as desired.

□

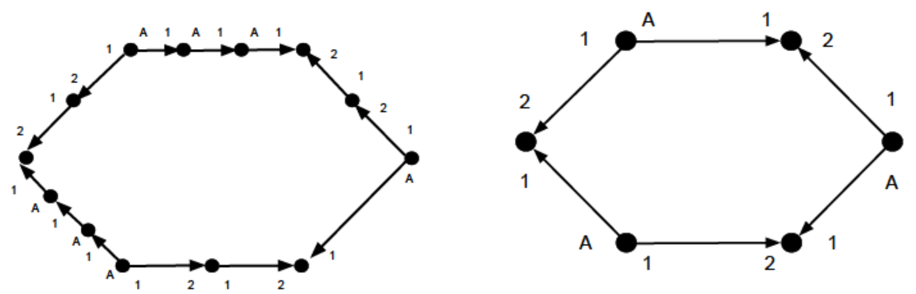
Lemma 2.5. *Consider the partitioning of $E' = \bigcup_{i=1}^{q} \bigcup_{j=1}^{\frac{k_i}{2}} E_i^j$. One can find a scheduling in polynomial time in which every slot E_i^j uses 2 new time steps except slot E_1^1 which uses 3 new time steps.*

Proof. Recall that each slot is a directed 2-matching and as we know a 2-matching is a collection of directed cycles and paths. Let C be a directed component of slot E_i^j. Without loss of generality, we assume that underlying undirected graph of C is a cycle; since for the case that it is a path, we can add a dummy edge.

To simplify analysis we modify cycle C into a new graph, call it C'. Let maximal directed subgraph of C be a maximal set of consecutive edges with the same direction in C. In order to build C', we contract all edges of a maximal directed subgraph into a one new edge. Note that if we schedule an arbitrary edge of C' with tuple (t_1, t_2), we can schedule all the edges of the corresponding maximal directed subgraph in C with tuple (t_1, t_2). According to the definition of maximal directed subpath, we can see that on each vertex of C distinct time steps will appear; so, this will be a proper scheduling for C.

Let positive (negative) vertices of C' be vertices with in-degree (out-degree) 2. Clearly, no two negative vertices or no two positive vertices can be adjacent. Hence, C' is a bipartite graph and must be an even cycle. We consider the partitioning of edges of cycle C' into two sets: clockwise edges and counter clockwise edges. Since C' is an even cycle, the number of clockwise edges and counter clockwise edges will be equal.

If C belongs to slot E_1^1, we will schedule edges of C' in the following pattern: Schedule all clockwise edges with tuple $(1, 2)$ and schedule the rest of edges with tuple $(2, 3)$. So, slot E_1^1 uses at most 3 time steps.



(a) An original cycle C

(b) modified cycle C' after edge contraction

Figure 2.3: An illustration of how to convert cycle C to cyle C' and how to schedule each of them.

Now, assume that C belongs to any slot other than E_1^1. Since degree of each vertex in any slot is at most 2 and we used 3 time steps for E_1^1 and 2 time steps for all subsequent slots, always one time step is available for any vertex in C which means that this time step can be further used for an outgoing incident edge in the next slot (if it exists).

Let E_i^j be an arbitrary slot other than the first and consider all of the slots prior to E_i^j. Clearly this slot belongs to the i'th block. So, except the $j-1$ slots before E_i^j, other slots are from blocks $E_1, E_2, \ldots, E_{i-1}$. As a result, the degree of each node in block E_i^j is at most $\sum_{\ell=1}^{i-1} k_\ell + 2(j-1)$. However, the number of time steps we have assigned so far is:

$$3 + 2\left(\frac{k_1}{2} - 1\right) + 2\frac{k_2}{2} + \ldots + 2\frac{k_{i-1}}{2} + 2(j-1) = \sum_{\ell=1}^{i-1} k_\ell + 2(j-1) + 1$$

So, in slot E_i^j for each vertex at least one time step from set $\{1, 2, \ldots, \sum_{\ell=1}^{i-1} k_\ell + 2j - 1\}$ is available. Let A be the available colour at each vertex of slot E_i^j which may be different for vertices within each slot. We schedule all clockwise edges with tuple $(A, 1^+)$ and counter clockwise edges with tuple $(1^+, 2^+)$ where $1^+ = \sum_{\ell=1}^{i-1} k_\ell + 2j$ and $2^+ = \sum_{\ell=1}^{i-1} k_\ell + 2j + 1$. Hence, in this case slot E_i^j uses at most 2 new time steps. Also, the makespan of each job in slot E_i^j is at most

$$\sum_{\ell=1}^{i-1} k_\ell + 2j + 1. \qquad (2.9)$$

Moreover, similar to the previous case, the given scheduling is proper as each vertex will receive two different time steps, since the number of clockwise and counter clockwise edges are equal in the modified cycle. $\square$

Lemma 2.6. *For the partitioning of E' such that $E' = \bigcup_{i=1}^{q} E_i$, each job in block E_q has a completion time of at most $1 + \sum_{\ell=1}^{q} k_\ell$.*

Proof. Using Lemma 2.4 and Algorithm 3, block E_q will be partitioned into slots $E_q^1, \ldots E_q^{\frac{k_q}{2}}$. So, clearly the makespan of each job in block E_q is at most the makespan of each job in the last slot which is $E_q^{\frac{k_q}{2}}$. According to Equation (2.9) in the proof of Lemma 2.5, the makespan of each job in slot E_i^j is at most $\sum_{\ell=1}^{i-1} k_\ell + 2j + 1$. Hence, the following will be an upper bound on the makespan of each job in block E_q:

$$\sum_{\ell=1}^{q-1} k_\ell + 2(\frac{k_q}{2}) + 1 = \sum_{\ell=1}^{q} k_\ell + 1$$

$\square$

In order to decrease the cost of the algorithm, between any ordering of edges in the last block and its reverse we choose the better one. In other words, for scheduling each slot, we can do the reverse of the mentioned pattern in the proof of Lemma 2.5 and choose the better one .i.e., assign tuple $(A, 2^+)$ to all clockwise edges and $(A, 1^+)$ to all counter clockwise edges; moreover, in each block we can schedule the slots $E_q^1, \ldots, E_q^{\frac{k_q}{2}}$ in the reverse order. Hence, by taking the better of these two reverse orderings (both in the last block and each slot), Observation 2 will be obvious.

Observation 2. *If we consider the reverse ordering of edges in block E_q and run Algorithm 3 again, then the average completion time of jobs in E_q will be at most $1 + \sum_{\ell=1}^{q-1} k_\ell + \dfrac{k_q + 1}{2}$.*

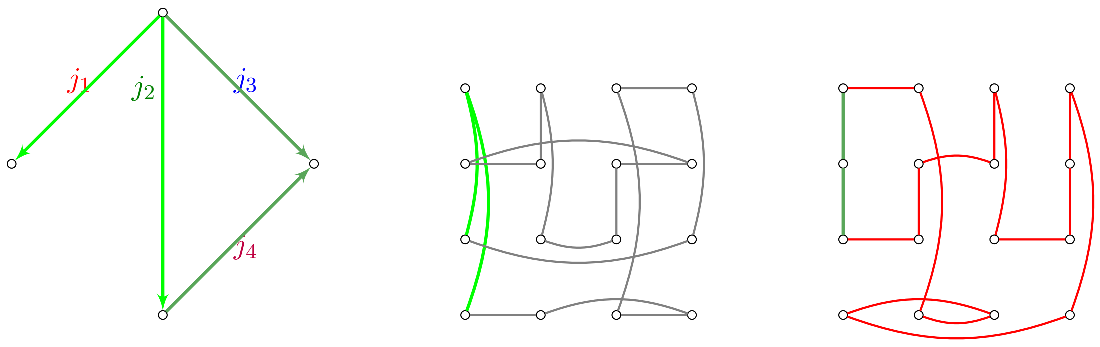


Figure 2.4: In this example we show how to schedule edges of the shown graph in Figure 2.1. Green edges of each cycle are the edges of the original graph that should be scheduled in two different slots.

Now we want to compare the cost of Algorithm 3 against the cost of a solution with optimum total completion time. Let C_j^{ALG} and C_j^{OPT} be the completion time of j'th job in the schedule given by Algorithm 3 and a schedule with optimum total completion time, respectively. If job j is scheduled in block q in the solution given by Algorithm 3 then according to Lemma 2.6 and Observation 2, we get that $C_j^{ALG} \leq 1 + \sum_{\ell=1}^{q-1} k_\ell + \dfrac{k_q+1}{2}$.

Assume that $C_j^{OPT} = dc^q$ for some $d < c$ and some integer $q \geq 1$ and recall that $k_i = 2\lfloor \dfrac{c^{i+\alpha}}{2} \rfloor$. Now the following two cases will arise:

Case 1: $d < c^\alpha$. In this case, because $C_j^{OPT} \in \mathbb{Z}$, we get the following:

$$C_j^{OPT} = dc^q \leq 2\lfloor \frac{dc^q}{2} \rfloor + 1 \leq 2\lfloor \frac{c^{q+\alpha}}{2} \rfloor + 1.$$

$$C_j^{OPT} \leq 2\lfloor \frac{c^{q+\alpha}}{2} \rfloor = k_q.$$

It means that in a schedule with optimum objective function, the completion time of j'th job is at most k_q. In other words, if we send jobs in our solution in the order of optimum scheduling, then j'th job (in the optimum schedule) will be scheduled at latest in iteration q. Since in iteration q of Algorithm 3 we find a maximum number of edges that can be completed within k_q units of time (maximum b-matching(k_q)), the j'th edge in Algorithm 3 must be scheduled before or in iteration q. As a result,

$$C_j^{ALG} \leq 1 + \sum_{i=1}^{q-1} k_i + \frac{k_q+1}{2}.$$

Case 2: $d \geq c^\alpha$. Since $d < c$ and $c^\alpha \geq 1$, we get that:

$$dc^q \leq 2\lfloor \frac{dc^q}{2} \rfloor + 1 < 2\lfloor \frac{c^{q+1}}{2} \rfloor + 1 \leq 2\lfloor \frac{c^{q+\alpha+1}}{2} \rfloor + 1$$
$$C_j^{OPT} \leq 2\lfloor \frac{c^{q+1+\alpha}}{2} \rfloor = k_{q+1}.$$

Hence for the same reason as Case 1, $C_j^{ALG} \leq 1 + \sum_{i=1}^{q} k_i + \dfrac{k_{q+1}+1}{2}$.

Next we simplify the upper bounds for each case. For the first case we get that:

$$\begin{aligned}
C_j^{ALG} &\leq 1 + \sum_{i=1}^{q-1} k_i + \frac{k_q + 1}{2} \leq 1 + 2\lfloor \frac{c^{1+\alpha}}{2} \rfloor + \sum_{i=2}^{q-1} c^{i+\alpha} + \frac{c^{q+\alpha}+1}{2} \\
&= 1 - c^\alpha + c^\alpha - c^{1+\alpha} + c^{1+\alpha} + 2\lfloor \frac{c^{1+\alpha}}{2} \rfloor + \sum_{i=2}^{q-1} c^{i+\alpha} + \frac{c^{q+\alpha}+1}{2} \\
&= \sum_{i=0}^{q-1} c^{i+\alpha} - c^\alpha - c^{1+\alpha} + \frac{c^{q+\alpha}}{2} + \frac{3}{2} + 2\lfloor \frac{c^{1+\alpha}}{2} \rfloor \\
&= c^\alpha \frac{c^q - 1}{c-1} + \frac{c^{q+\alpha}}{2} + \frac{3}{2} - c^\alpha - c^{1+\alpha} + 2\lfloor \frac{c^{1+\alpha}}{2} \rfloor \\
&= c^{q+\alpha}\left(\frac{1}{c-1} + \frac{1}{2}\right) + \frac{3}{2} - \frac{c^\alpha}{c-1} + \beta_j,
\end{aligned}$$

where $\beta_j = 2\lfloor \frac{c^{1+\alpha}}{2} \rfloor - c^\alpha - c^{1+\alpha}$. For the second case, following similar steps we get that:

$$C_j^{ALG} \leq c^{q+1+\alpha}\left(\frac{1}{c-1} + \frac{1}{2}\right) + \frac{3}{2} - \frac{c^\alpha}{c-1} + \beta_j.$$

Recall that $\int_0^1 c^\alpha d\alpha = \frac{c-1}{\ln c}$. So, by taking the expectation of C_j^{ALG} over α, we get the following:

$$\begin{aligned}
\mathbf{E}[C_j^{ALG}] &\leq \int_{\log_c d}^{1} \left(c^{q+\alpha}\frac{c+1}{2(c-1)} - \frac{c^\alpha}{c-1} + \frac{3}{2} + \beta_j\right) d\alpha \\
&+ \int_0^{\log_c d} \left(c^{q+1+\alpha}\frac{c+1}{2(c-1)} - \frac{c^\alpha}{c-1} + \frac{3}{2} + \beta_j\right) d\alpha \\
&= \int_0^1 \left(\frac{3}{2} - \frac{c^\alpha}{c-1} + \beta_j\right) d\alpha + \frac{c+1}{2(c-1)} c^q \int_{\log_c d}^1 c^\alpha d\alpha + \frac{c+1}{2(c-1)} c^{q+1} \int_0^{\log_c d} c^\alpha d\alpha \\
&= \frac{3}{2} - \frac{1}{\ln c} + \int_0^1 \beta_j d\alpha + \frac{c+1}{2\ln c} dc^q. \qquad (2.10)
\end{aligned}$$

Next we will find $\int_0^1 \beta_j d\alpha$. Note that in the following lemma the range of q has been chosen with some foresight.

Lemma 2.7. *For* $3 \leq c < \sqrt{14}$,

$$\int_0^1 \beta_j d\alpha = -(c+1)\frac{c-1}{\ln c} + 22 - 2\log_c 23040.$$

Proof. Since $c < \sqrt{14}$, $\log_c 14 - 1 > 1$ and we can write the following for values of $\lfloor \frac{c^{1+\alpha}}{2} \rfloor$:

$$\lfloor \frac{c^{1+\alpha}}{2} \rfloor = \begin{cases} 1, & \alpha \in [0, \log_c 4 - 1) \\ 2, & \alpha \in [\log_c 4 - 1, \log_c 6 - 1) \\ 3, & \alpha \in [\log_c 6 - 1, \log_c 8 - 1) \\ 4, & \alpha \in [\log_c 8 - 1, \log_c 10 - 1) \\ 5, & \alpha \in [\log_c 10 - 1, \log_c 12 - 1) \\ 6, & \alpha \in [\log_c 12 - 1, 1). \end{cases}$$

Hence,

$$\begin{aligned} \int_0^1 2\lfloor \frac{c^{1+\alpha}}{2} \rfloor d\alpha &= 2\left(\int_0^{\log_c 4 - 1} 1 d\alpha + \int_{\log_c 4 - 1}^{\log_c 6 - 1} 2 d\alpha + \ldots + \int_{\log_c 12 - 1}^{1} 6 d\alpha \right) \\ &= 2\left(\log_c 4 - 1 + 2(\log_c 6 - \log_c 4) + \ldots + 6(1 - \log_c 12 + 1)\right) \\ &= 2(11 - \log_c(32 \times 6!)) = 22 - 2\log_c 23040. \end{aligned}$$

Therefore,

$$\begin{aligned} \int_0^1 \beta_j d\alpha &= \int_0^1 \left(2\lfloor \frac{c^{1+\alpha}}{2} \rfloor - c^{\alpha} - c^{1+\alpha} \right) d\alpha \\ &= -(c+1)\frac{c-1}{\ln c} + 22 - 2\log_c 23040. \end{aligned}$$

□

So by using Lemma 2.7 in Equation (2.10), we get that:

$$\begin{aligned} C_j^{ALG} &\leq \frac{c+1}{2\ln c} C_j^{OPT} + \frac{47}{2} - 2\log_c 23040 - \frac{c^2}{\ln c} \\ &\leq \frac{c+1}{2\ln c} \cdot C_j^{OPT}; \end{aligned}$$

since $\frac{47}{2} - 2\log_c 23040 - \frac{c^2}{\ln c}$ is a negative term for $c > 0$, we can write the second inequality. For $c = 3.59$, we get the approximation ratio of 1.796 as claimed.

2.3 Special Case: Regular Directed Auxiliary Graph

Recall that we say the given graph $H(V', E')$ is regular directed if and only if $\forall v \in V', deg^{out}(v) = deg^{in}(v) = \frac{\Delta}{2}$. As we know $|V'| = m, |E'| = n$. One simple observation is that $2n = m\Delta$. Recall that cycle-cover of digraph H is a spanning directed subgraph of H in which every vertex has in-degree and out-degree 1.

Note that the existence of cycle-cover is not always guaranteed in general, but in Lemma 2.8 we show that in this case we can always find one in polynomial time.

Consider Algorithm 4 for this problem in which we partition a graph into a collection of paths or cycles and schedule each edge in 2 time steps.

```
Input  : Regular directed auxiliary graph $H$
Output: A scheduling of the jobs
1 $i \leftarrow 1$
2 $W \leftarrow E'$
3 while $W \neq 0$ do
4     $C_i \leftarrow$ cycle-cover
5     Schedule edges of $C_i$ with tuple $(2i-1, 2i)$
6     $W \leftarrow W \setminus C_i$
7     $i \leftarrow i + 1$
8 end
```

Algorithm 4: Polynomial algorithm for minimizing total flow time of unit processing time jobs on stars for the special case that auxiliary graph H is a regular directed graph.

Lemma 2.8. *When digraph $H(V', E')$ is a $\frac{\Delta}{2}$-regular directed graph, we can find $\frac{\Delta}{2}$ edge disjoint cycle-covers in graph H in polynomial time.*

Proof. We construct a bipartite graph $B(\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = L' \cup R'$ and $L' = \{x_1, \dots, x_m\}$, $R' = \{y_1, \dots, y_m\}$. Moreover, there exists an undirected edge (x_i, y_j) in $\mathcal{E}$ if and only if a directed edge (v_i, v_j) belongs to E' for every $i, j \in \{1, \dots, m\}, i \neq j$. Clearly, $|E'| = |\mathcal{E}| = n$ and B is a $\frac{\Delta}{2}$-regular graph. Hence, by using Hall's theorem we can see that graph B has $\frac{\Delta}{2}$ perfect matchings [Ha35]. The following proposition has been proved for undirected graphs in [Pe1891], but the proof can be easily extended to directed graphs.

Proposition 1. *Each of $\frac{\Delta}{2}$ perfect matchings in graph B, represent a cycle-cover in digraph H.*

According to Hopcroft-Karp Algorithm [HK71], we can find bipartite matchings in graph B in $O(n\sqrt{m})$ and convert each of them to a cycle-cover of graph H in $O(m)$ time.

□

Now we can compute the cost of Algorithm 4 as the following:

$$\begin{aligned} ALG = \sum_{i=1}^{\frac{\Delta}{2}} 2i|C_i| &= 2m\sum_{i=1}^{\frac{\Delta}{2}} i \\ = m.(\frac{\Delta}{2})(\frac{\Delta}{2}+1) &= \frac{m\Delta^2}{4} + \frac{m\Delta}{2} = \frac{m\Delta^2}{4} + n, \end{aligned}$$

By using Lemma 2.2, the lower bound of optimal solution for this case is:

$$OPT \geq \sum_{v\in V'} \frac{\Delta(\Delta+1)}{4} + \frac{n}{2} = \frac{m\Delta^2}{4} + n.$$

Hence, Algorithm 4 produces the optimal solution.

2.4 Min Sum Edge Colouring versus Star Scheduling

Recall that in the min sum edge colouring problem of a given graph $H(V', E')$, our goal is to find a proper edge colouring $\psi : E' \to \{1, \ldots, k\}$ minimizing the function $\sum_{e\in E'} \psi(e)$. As we can see the objective function of this problem is similar to the star scheduling problem. Consider both of these problems for digraph $H(V', E')$ (and its underlying undirected graph).

In this section, our goal is to show the relation between optimal solution of min sum edge colouring and optimal solution of star scheduling problem on the same instances which will be shown by OPT_{MSE} and OPT_{SSUP}, respectively. In other words, we will prove Lemma 2.9 and Lemma 2.10.

Lemma 2.9. $OPT_{MSE} \leq 2OPT_{SSUP}$.

Proof. To show this, we use the given upper bound for optimal solution of MSE problem in [BBH+98] and the lower bound of star scheduling problem that we have proved in Lemma 2.2.

Following is the definition of compact edge colouring according to [BBH+98]. An edge colouring $\psi : E' \to \{1, \ldots, k\}$ is compact if and only if every edge e with colour $\psi(e) = i$ is adjacent to some edge e' with $\psi(e') = j$ for all $1 \leq j \leq i$ (note that k is not necessarily equal to $\chi'(H)$).

Let *comp*(H) denote the cost of any compact edge colouring in graph H. It has been shown in [KKK91] that *comp*$(H) \leq \sum_{v\in V'} \binom{deg(v)}{2} + |E(H)|$.

$$OPT_{MSE} \leq comp(H) \leq \frac{1}{2}\sum_{v\in V'} deg^2(v).$$

Also, it has been proved in Lemma 2.2 that,

$$OPT_{SSUP} \geq \frac{1}{4}\sum_{v\in V'} deg^2(v) + n.$$

So, the Lemma holds.

□

Lemma 2.10. $OPT_{SSUP} \leq OPT_{MSE} + |E'|$.

Proof. In order to prove this lemma, we need to show that there exists a solution for star scheduling problem with cost at most $OPT_{MSE} + |E'|$.

Consider an optimal solution of min sum edge colouring problem. By denoting edges of each colour class i as a matching M_i with $\mathcal{M}$ being a maximum colour in optimal solution, we get a partitioning of $\bigcup_{i=1}^{\mathcal{M}} M_i$ on E'. Also, the cost of optimal solution of MSE problem will be:

$$OPT_{MSE} = \sum_{i=1}^{\mathcal{M}} i|M_i|.$$

Our goal is to show that there exists a schedule in which the completion time of all edges in M_i is at most $i+1$, for every $i \in \{1, \ldots, \mathcal{M}\}$. To do that, we use strong induction on i. Let predicate $P(n)$ be the following:

$P(n)$: There is a schedule such that all edges of $M_1, \ldots, M_n$ are finished by time $n+1$. The base case of $n = 1$ is true, since we can schedule edges of M_1 with tuple $(1, 2)$. Now suppose that $P(1)$ up through $P(k)$ are all true for some integer $k < n$. We need to show that $P(k+1)$ is true.

Let H' be the induced subgraph of graph H by $M_1, \ldots, M_k$. For all edges in $M_1, \ldots, M_k$ the completion time of edges are at most $k+1$. However, the degree of vertices in graph H' is at most k. So, for edges in M_{k+1} their start point can use an unused time from set $\{1, \ldots, k+1\}$ and their finish time will be $k+2$.

□

2.5 Hardness Result

In this section, we consider a variant of star scheduling problem and we show a hardness result for this new problem.

Definition 2.11. *Let SSND be the Star Scheduling problem with No intermediate Delay meaning that once we schedule job j on its first machine, we have to finish it in the next time step, i.e., tuple (t_1^j, t_2^j) is assigned for job j if and only if $t_2^j - t_1^j = 1$.*

Theorem 2.12. *The SSND problem is APX-hard.*

Proof. It has been proved in [M09] that min-sum edge colouring problem is APX-hard even for bipartite graphs with maximum degree 3. We denote MSE problem on bipartite graphs by MSEB. Also, let us denote the optimal cost of MSEB problem and SSND problem by OPT_M and OPT_S, respectively.

In order to prove this theorem, we use L-reduction from MSEB problem to SSND problem which is defined in the following:

Definition 2.13. *For two optimization problems MESB and SSND, there is an L-reduction with parameters a and b from MSEB problem to SSND problem if for some $a, b > 0$*

1. *for each instance I of MSEB problem, we can compute in polynomial time an instance I' of SSND problem.*
2. $OPT_S(I') \leq aOPT_M(I)$
3. *given a solution of I' with cost SOL_S, we can compute in polynomial time a solution of I with cost SOL_M such that:*

$$SOL_M - OPT_M(I) \leq b\left(SOL_S - OPT_S(I')\right)$$

In the following Lemma we will show that why this reduction is useful in the proof.

Lemma 2.14. *Assume that there is an L-reduction with parameters a and b from MSEB to SSND. Then if there is no ρ-approximation algorithm for MSEB problem, SSND cannot be approximated with ratio better than $\frac{\rho-1}{ab}+1$, unless $\mathbf{P} = \mathbf{NP}$.*

Proof. Assume otherwise, there exists a ρ' approximation algorithm for SSND with cost SOL_S where $\rho' < \dfrac{\rho-1}{ab}+1$. Hence,

$$\begin{aligned} SOL_S &< (\frac{\rho-1}{ab}+1)OPT_S(I') \\ SOL_S - OPT_S(I') &< (\frac{\rho-1}{ab})OPT_S(I') \qquad (2.11) \\ SOL_M - OPT_M(I) &< (\rho-1)OPT_M(I), \qquad (2.12) \end{aligned}$$

where Equations (2.11) and (2.12) are by using Conditions 2 and 3 of Definition 2.13. According to Equation (2.12), we get that $SOL_M < \rho OPT_M(I)$ which means that the obtained solution for MSEB is a ρ-approximation, a contradiction. $\square$

It can be seen that if $\rho > 1$ and $a, b > 0$ then $\frac{\rho-1}{ab} + 1 > 1$.

Instances of MSEB problem are bipartite graphs $\mathcal{G}(L \cup R, E'')$. However, instances of $SSND$ problem are a set of jobs and machines such that the network of machines create a star. In order to convert a given instance I for $MSEB$ to an instance I' for $SSND$, we convert I to an auxiliary graph H of star scheduling problem and as we have seen earlier in Section 2.1, a graph H can be converted to a star in polynomial time.

We do that by directing each edge from set L to R in graph $\mathcal{G}$, meaning that condition 1 is valid. Clearly, $|E''| = |E'|$.

Recall that a solution to $MSEB$ problem for the graph $\mathcal{G}$ is a function $\psi : E'' \to \{1, 2, \ldots, k\}$. Next we want to show that $OPT_S(I') \leq OPT_M(I) + |E''|$. In other words, we claim that there exists a solution for $SSND$ problem with value $OPT_M(I) + |E''|$. To get that solution, we assign tuple $(i, i+1)$ to edge e, if $\psi(e) = i$ in the optimal $MSEB$ solution, for $i = 1, \ldots, k$ and for every $e \in E''$. It can be seen that this solution is a proper scheduling since edges that share the same endpoint or start point are assigned with different tuples (as they have different colours in optimal solution of MSEB problem).

Also, $OPT_M(I) \geq |E''|$, since each edge contributes at least one to the edge colouring summation. Hence, we get the following:

$$OPT_S(I') \leq OPT_M(I) + |E''| \leq 2OPT_M(I), \tag{2.13}$$

meaning that for parameter $a = 2$ Condition 2 holds.

The last thing that remains to show is how to compute a solution for MSEB problem with value SOL_M given a solution to $SSND$ with value SOL_S and makespan Δ' such that $SOL_M = SOL_S - |E''|$. Recall that a solution to $SSND$ problem with the auxiliary graph H is an assignment of tuples (t_1^e, t_2^e) to each edge $e \in E'$ where $|t_2^e - t_1^e| = 1$. Hence, we colour each edge e of E'' with time step t_1^e if and only if the corresponding edge in auxiliary graph H has been scheduled with tuple (t_1^e, t_2^e). We show by contradiction that the obtained solution is a proper edge colouring solution.

If the obtained solution is not a proper edge colouring, it means that there were at least two edges with the same start point or end point such that both of them get colour q, for some $q \leq \Delta' - 1$. So, two jobs are scheduled on the same machine at time k or $k+1$ in the first case or second case, respectively. This is a contradiction to the fact that our schedule was proper. Hence, we get that the obtained solution for $MSEB$ is a proper edge colouring such that

$$\begin{aligned} SOL_S &= \sum_{e \in E'} t_2^e = \sum_{e \in E'} (t_1^e + 1) = \sum_{e \in E'} t_1^e + |E'| \\ &= \sum_{e \in E''} t_1^e + |E''| = SOL_M + |E''|. \end{aligned}$$

Hence, we have the following:

$$\begin{aligned} SOL_M - OPT_M(I) &= SOL_S - |E''| - OPT_M(I) \\ &\leq SOL_S - |E''| - \left(OPT_S(I') - |E''|\right), \end{aligned} \tag{2.14}$$

where Equation (2.14) is by rearranging Equation (2.13). As a result, Equation (2.14) shows that the third condition is valid for parameter $b = 1$.

As we know there is a constant $\rho > 1$ such that MSEB problem cannot be approximated with ratio ρ unless $\mathbf{P} = \mathbf{NP}$ [M09]. According to Lemma 2.14, we get that SSND cannot be approximated with ratio $\frac{\rho+1}{2}$, unless $\mathbf{P} = \mathbf{NP}$.

□

Chapter 3

Generalized Path Scheduling with Unit Processing Time

3.1 Problem Overview

In this chapter we consider the generalized path scheduling problem with unit processing time jobs and our goal is to find a schedule with the minimum total completion time. Recall that in the GPUP problem, we are given a set J of n jobs and a set $M = \{1, \ldots, m\}$ for machines. Moreover, we are given a path $P(V, M)$ for the network of machines.

Since each job takes one unit of time to be processed on any of the machines on its path, we can think of a schedule as a synchronized schedule in which at each tick of the clock each machine processes at most one job (among jobs that were available to be run on that machine).

Consider an arbitrary machine $g \in M$ and job $j \in J$ such that $g \in P_j$; now let L_j^g or *remaining length of job j after machine g* be the total processing time of job j on machines g to t_j that is $L_j^g = t_j - g + 1$. Let B_g^t be the set of jobs that are ready to be processed on machine g at time t where $t \in \{0, \ldots, m + n\}$.

3.1.1 Our Results

For the GPUP problem we show that there exists a polynomial time algorithm for minimizing total completion time.

Let *shortest remaining length algorithm* be the algorithm in which each machine $g \in M$ at any time t will process the job with the shortest remaining length from set B_g^t (ties are broken arbitrary).

In [KNSM14] and [AB+14], it has been shown that the shortest remaining length algorithm produces an optimal solution. In fact, they have considered the on-line version of this problem. In their analysis, for each machine they compared a job that will be processed on it based on a valid scheduling and the proposed algorithm; then they considered two possible scenarios: whether these two jobs will be available to be processed on the same machine in the future or not. They showed that in both scenarios one can modify the valid schedule without increasing the total completion time such that it satisfies the shortest remaining length strategy.

In this section we will prove the same result but with a simpler analysis.

Theorem 3.1. *There is an optimal solution for the GPUP problem such that each machine $g \in J$ will first process jobs with the shortest remaining length from set B_g^t at any $t \in \{0, \dots, m+n\}$.*

Proof. Suppose for the sake of contradiction that for all optimal solutions the property "processing the job with the shortest remaining length first" has been violated. For an arbitrary schedule, let *conflicting time* be the last time that "processing the job with the shortest remaining length first" has been violated and *conflicting machine* be the rightmost (last) machine that violation has been occurred on. Among all optimal solutions consider those that have the smallest conflicting time, and among them consider the one with the leftmost conflicting machine. We call this solution the *minimal* optimal solution, and denote by sol_1.

Our goal is to modify sol_1 such that the conflicting time of the new solution is smaller or the conflicting machine is closer to machine 1 without increasing the total completion time which is a contradiction.

Without loss of generality let time b_1 and machine 1 be the conflicting time and conflicting machine in sol_1; so, at time b_1 there were two available jobs i and j on machine 1 such that $L_i^1 < L_j^1$, but sol_1 sends job j before job i on machine 1. Let $t_i = k$ and job i is processed on machines $1, \dots, k$ at times $a_1, \dots, a_k$ in sol_1, respectively. Also, assume that for job j, $t_j \geq k+1$ and it is processed on machines $1, \dots, k+1, \dots, t_j$ at times $b_1, \dots, b_{k+1}, \dots, b_{t_j}$ in sol_1, respectively. For an illustration see Figure 3.1.

According to our assumption $b_1 < a_1$. Now if machine $k' \in [2, k]$ is the leftmost machine in which $b_{k'} > a_{k'}$, we will build sol_2 from sol_1 by swapping jobs i and j on machines $1, \dots, k'-1$. In other words, job j will be processed on machines $1, \dots, k'-1, k' \dots, t_j$ at times $a_1, \dots a_{k'-1}, b_{k'}, \dots, b_{t_j}$. This scheduling is valid because $a_{k'-1} <$

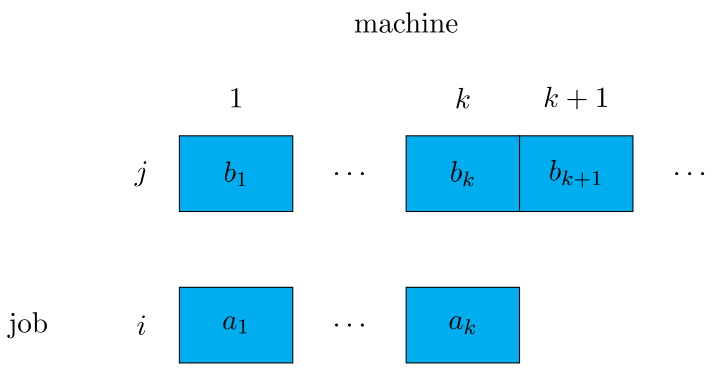


Figure 3.1: Scheduling of jobs i, j in sol_1. Start time of each job on different machines is shown in the corresponding box.

$a_{k'} < b_{k'}$. Similarly, job i will be processed on machines $1, \ldots, k'-1, k', \ldots, k$ at times $b_1, \ldots, b_{k'-1}, a_{k'}, \ldots, a_k$ and it is a valid scheduling since $b_{k'-1} < a_{k'-1} < a_{k'}$. Clearly, the completion times of jobs i and j will not change.

Now consider the case that on every machine $g \in \{1, \ldots, k\}$, $b_g < a_g$. In this case by swapping jobs i and j on machines $1, \ldots, k$ in sol_1, we will build sol_2. Moreover, for $1 \leq z \leq t_j - k$ if $a_k + z - 1 < b_{k+z}$ job j will be processed on machine $k+z$ similar to sol_1 and if $a_k + z - 1 \geq b_{k+z}$ it will be processed at time $a_k + z$. In this way, the completion time of job j will not change and the completion time of job i will decrease. Also, in the worst case the summation of completion times of jobs i and j will be at most $b_k + a_k + t_j - k$; since $b_k + t_j - k \leq b_{t_j}$, the total completion time of sol_2 will not increase.

It is clear that by processing job j in sol_2 similar to sol_1, there will not be any conflict on machines $1, \ldots, k$. However, for the case that machine $k+z$ is processing job j at time $a_k + z$, there can be some other job that is processed on machine $k+z$ at the same time. In Claim 3.2 we will show that such job cannot exist. Hence, sol_2 is a valid solution and the conflicting machine is closer to machine 1 which contradicts the minimality of sol_1.

Claim 3.2. *For $1 \leq z \leq t_j - t_i$, if $a_k + z - 1 \geq b_{k+z}$ then in sol_2 machine $k+z$ can process job j at time $a_k + z$.*

Proof. We use induction on the index of machines. Let predicate $p(z)$ be the following:

If $a_k + z - 1 \geq b_{k+z}$ then in sol_2 machine $k+z$ can process job j at time $a_k + z$.

We prove $p(z)$ for all $1 \leq z \leq t_j - t_i$. The base case is when $z = 1$. In the following we will show the correctness of the base case.

We need to show that in sol_2 job j can be processed on machine $k+1$ at time a_k+1. Assume otherwise. If there is any conflict it must be on machine $k+1$ in which some job j_{k+1} is being processed on machine $k+1$ at time a_k+1. Now if machine $k+1$ is the starting machine of job j_{k+1}, then simply it can be processed on this machine at time b_{k+1} which is available (in sol_1 this machine has been used by job j at time b_{k+1}). So, we consider the case that the starting machine of job j_{k+1} is before machine $k+1$ which means that it will be processed on machine k at some time that is less than or equal to a_k-1 (in sol_1 machine k is busy with job i at time a_k).

Let ℓ be the largest available time in which there is no job on machine $k+1$ where $\ell \in [b_{k+1}, a_k]$. Since ℓ is the largest available time, there will be $a_k - \ell + 1$ number of jobs $j_{\ell+1}, \ldots, j_k, j_{k+1}$ which are processed on machine $k+1$ at time steps $\ell+1, \ldots, a_k, a_k+1$, respectively.

Now assume that job $j_{\ell^*} \in \{j_{\ell+1}, \ldots, j_k, j_{k+1}\}$ is the closest job to j_{k+1} with the starting machine $k+1$ such that it is processed on machine $k+1$ at time ℓ^*; so by processing job j_{ℓ^*} at time step b_{k+1} the largest available time on machine $k+1$ will be ℓ^* where $\ell^* > \ell$, contradicting the maximality of ℓ. So, the starting machine of all jobs $\{j_{\ell+1}, \ldots, j_{k+1}\}$ are before machine $k+1$.

As we mentioned earlier jobs $j_{\ell+1}, \ldots, j_{k-1}, j_k$ will be processed on machine $k+1$ at times $\ell+1, \ldots, a_k-1, a_k$. Moreover, jobs $j_{\ell+1}, \ldots, j_{k-1}, j_k$ will be processed on machine k at some times that are less than or equal to $\ell, \ldots, a_k-2, a_k-1$, respectively. For an illustration see Figure 3.2. Now if job $j_{\ell+1}$ is being processed on machine k at some time less than ℓ then it could have been processed on machine $k+1$ at time ℓ and the largest available time for machine $k+1$ would be $\ell+1$, contradicting the maximality of ℓ. So, consider the case that $j_{\ell+1}$ is being processed on machine k at time ℓ.

Similarly, if machine k is processing job $j_{\ell+2}$ at some time less than ℓ then $j_{\ell+2}$ could have been processed on machine $k+1$ at time ℓ and the largest available time for machine $k+1$ would be $\ell+2$, contradicting the maximality of ℓ. Hence, we consider the case that $j_{\ell+2}$ is being processed on machine k at time $\ell+1$. Note that job $j_{\ell+2}$ cannot be processed on machine k at time ℓ as it is busy with job $j_{\ell+1}$.

Inductively, machine k is processing jobs $j_{\ell+3}, \ldots, j_{k-1}, j_k$ at times $\ell+2, \ldots, a_k-2, a_k-1$. Recall that job j_{k+1} is being processed on machine k at some time that

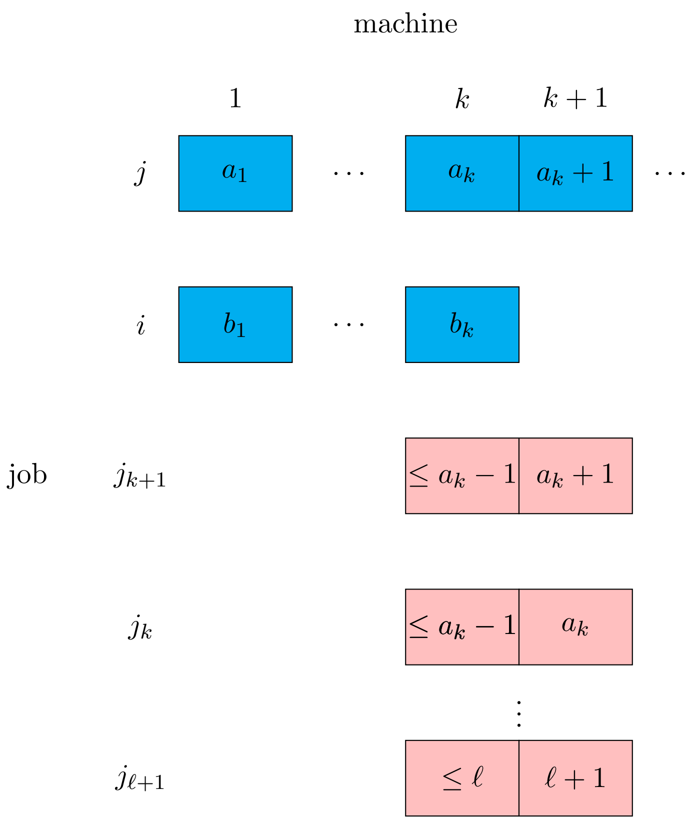


Figure 3.2: Scheduling of jobs $i, j, j_{k+1}, j_k, \dots, j_{\ell+1}$ in sol_2. Start time of each job on different machines is shown in the corresponding box.

is less than or equal $a_k - 1$. Now if job j_{k+1} is being processed on machine k at some time less than ℓ, then it would have been processed on machine $k+1$ at time ℓ and the largest available time would be $a_k + 1$, contradicting the maximality of ℓ. Also, it cannot be processed on machine k at times $\ell, \dots, a_k - 2, a_k - 1$ as machine k is busy with jobs $j_{\ell+1}, \dots, j_{k-1}, j_k$. This contradiction shows that job j_{k+1} cannot exist and there will be no conflict in sol_2.

Now suppose that $p(q)$ is true for all $q < z$, and next we will show that $p(z)$ is true. In other words, our goal is to show that if $a_k + z - 1 > b_{k+z}$ then in sol_2 machine $k + z$ can process job j at time $a_k + z$. The following observation can be verified easily.

Observation 3. *If* $a_k + z - 1 > b_{k+z}$ *then* $a_k + z - 2 > b_{k+z-1}$.

Hence, if $a_k + z - 1 > b_{k+z}$, by using Observation 3 and the fact that $p(z - 1)$ is true we get that job j can be processed on machine $k + z - 1$ at time $a_k + z - 1$.

Now for the sake of contradiction assume that $a_k + z - 1 > b_{k+z}$ but job j cannot be processed on machine $k + z$ at time $a_k + z$ meaning that some other job j_z has been processed on it.

The starting machine of job j_z must be before machine $k+z$; otherwise, it could have been processed on machine $k + z$ at time b_{k+z} and there will be no conflict. Also, machine $k + z - 1$ is processing job j_z at some time less than or equal to $a_k + z - 2$ because machine $k + z - 1$ is busy with job j at time $a_k + z - 1$.

Similar to the base case, we define ℓ to be the largest available time in which there is no job on machine $k + z$ where $\ell \in [b_{k+z}, a_k + z)$.

It can be seen that in this case there will be $a_k + z - 1 - \ell$ number of jobs $j_{\ell+1}, \ldots, j_{z-2}, j_{z-1}$ which are being processed on machine $k + z$ at time steps $\ell + 1, \ldots, a_k + z - 2, a_k + z - 1$, respectively. Also, with an argument similar to the base case, jobs $j_{\ell+1}, \ldots, j_{z-2}, j_{z-1}$ will be processed on machine $k + z - 1$ at some times that are less than or equal to $\ell, \ldots, a_k + z - 3, a_k + z - 2$, respectively. For an illustration see Figure 3.3.

Now if job $j_{\ell+1}$ is being processed on machine $k + z$ at some time less than ℓ then it could have been processed on machine $k + z$ at time ℓ and the largest available time for machine $k + z$ would be $\ell+1$ contradicting the maximality of ℓ. So, consider the case that job $j_{\ell+1}$ is being processed on machine $k + z - 1$ at time ℓ.

Inductively, machine $k + z - 1$ is processing jobs $j_{\ell+2}, \ldots, j_{z-2}, j_{z-1}$ at times $\ell + 1, \ldots, a_k + z - 3, a_k + z - 2$. As we know job j_z is being processed on machine $k + z - 1$ at some time that is less than or equal to $a_k + z - 2$. Now if job j_z is being processed on machine $k + z$ at some time less than ℓ, then it could have been processed on machine $k + z$ at time ℓ in sol_2 and the largest available time would be $a_k + z$ which contradicts the maximality of ℓ. Moreover, machine $k + z$ cannot process job j_z at times $\ell, \ldots, a_k + z - 3, a_k + z - 2$ as it is busy with jobs $j_{\ell+1}, \ldots, j_{z-2}, j_{z-1}$.

This contradiction shows that job j_z cannot exist meaning that there will be no conflict in sol_2 as claimed.

□

□

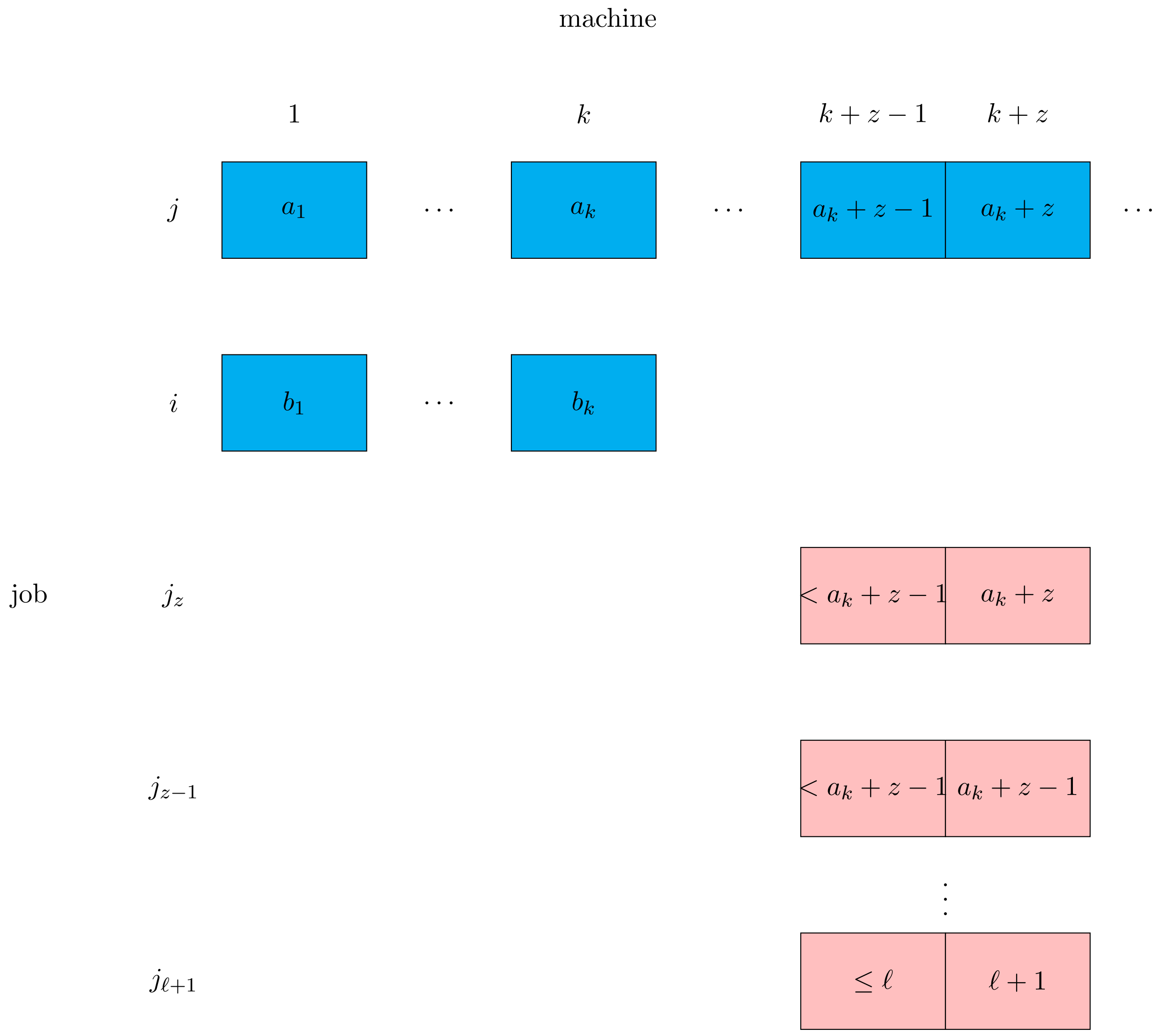


Figure 3.3: Scheduling of jobs $i, j, j_z, j_{z-1}, \ldots, j_{\ell+1}$ in sol_2. Start time of each job on different machines is shown in the corresponding box.

Chapter 4

Conclusion

4.1 Summary

In this thesis, we gave a 1.796-approximation for the star scheduling problem with unit processing time jobs when the objective function is minimizing the total completion time. This algorithm works as the following: in each iteration it finds a maximum b-matching(b) in the auxiliary graph of the network of machines and schedules the corresponding jobs such that the makespan will be small. To achieve this goal, in Lemma 2.4 we proved that each b-matching(b) is a collection of some 2-matchings. Moreover, in Lemma 2.5 we proved that there is a scheduling in which we can schedule almost all of the corresponding jobs of each 2-matching by using 2 new time steps.

For the case that the auxiliary graph of the network of machines is regular directed, we showed that there is a polynomial time algorithm with running time $O(nm\sqrt{m})$. We also defined a new variation of star scheduling problem in which no jobs can have delay in the center of the star and we showed that this variation is APX-hard.

In addition, we considered the generalized path scheduling problem with unit processing time jobs when our goal is to find a schedule with the minimum total completion time. For this problem, we presented a polynomial time algorithm with running time $O(mn \lg n)$ with a new analysis. In this algorithm, each machine will first process jobs with the shortest remaining length among the jobs that are ready to be processed.

Besides the above results, we also gave a simple 2-approximation for the SSUP problem, for minimizing the total completion time and by providing a tight example we showed that this algorithm cannot be improved. Another result that we came up with during this research was the relation between optimal solution of MSE and SSUP problem, shown in Lemmas 2.9, 2.10.

4.2 Future Work

We present a 1.796-approximation for the SSUP problem. However, we did not find any tight example for this algorithm. An obvious question is if there is an approximation algorithm for the SSUP problem with ratio better than 1.796. Consider the case that the auxiliary graph of the network of machines has equal in-degree and out-degree for every vertex. Can we get a better approximation for this case? We showed that if in-degree and out-degree of all vertices are equal to $\frac{\Delta}{2}$, there is a polynomial time algorithm.

In Chapter 2 of this thesis, we considered a variation of the SSUP problem, star scheduling problem with no intermediate delay (SSND) and proved that it is APX-hard. Though, we believe that the SSUP problem is also APX-hard, proving this still needs new ideas and remains an open problem.

In Chapter 3, we considered the generalized path scheduling problem with unit processing time jobs. An interesting question is whether there is any $O(1)$-approximation algorithm for the case that machines are identical but jobs have different processing times, for minimum total completion time. Consider the case that the network of machines form a spider (tree with one vertex of degree at least 3 and other vertices have degree at most 2). Another interesting question is whether there exists any $O(1)$-approximation algorithm when machines are identical.

Bibliography

[AB+14] Antoniadis, Antonios and Barcelo, Neal and Cole, Daniel and Fox, Kyle and Moseley, Benjamin and Nugent, Michael and Pruhs, Kirk. "Packet forwarding algorithms in a line network." Latin American Symposium on Theoretical Informatics. Springer Berlin Heidelberg, (2014): 610-621.

[BBH+98] Bar-Noy, Amotz, et al. "On chromatic sums and distributed resource allocation." Information and Computation 140.2 (1998): 183-202.

[BKM14] Bhattacharya, Sayan, Janardhan Kulkarni, and Vahab Mirrokni. "Coordination mechanisms for selfish routing over time on a tree." International Colloquium on Automata, Languages, and Programming. Springer, Berlin, Heidelberg, (2014): 186-197.

[BKS06] Bansal, Nikhil, Tracy Kimbrel, and Maxim Sviridenko. "Job shop scheduling with unit processing times." Mathematics of Operations Research 31.2 (2006): 381-389.

[CCPS98] Cook, William J and Cunningham, William H and Pulleyblank, William R and Schrijver, Alexander. "Combinatorial optimization." Vol. 605. New York: Wiley, 1998.

[CGRT03] Chaudhuri, Kamalika and Godfrey, Brighten and Rao, Satish and Talwar, Kunal. "Paths, trees, and minimum latency tours." Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on. IEEE, (2003): 36-45.

[FGK+17] Friggstad, Zachary, Arnoosh Golestanian, Kamyar Khodamoradi, Christopher Martin, Mirmahdi Rahgoshay, Mohsen Rezapour, Mohammad R. Salavatipour, and Yifeng Zhang. "Scheduling problems over network of machines." APPROX 2017.

[FS02] Feige, Uriel, and Christian Scheideler. “Improved bounds for acyclic job shop scheduling.” Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, 1998.

[GK00] Giaro, Krzysztof, and Marek Kubale. “Edge-chromatic sum of trees and bounded cyclicity graphs.” Information Processing Letters 75.1-2 (2000): 65-69.

[Ha35] Hall, Philip. “On representatives of subsets.” Journal of the London Mathematical Society 1.1 (1935): 26-30.

[Ha69] Harary, Frank. “Graph theory. 1969.”

[HH03] Hou, Sixiang, and Han Hoogeveen. “The three-machine proportionate flow shop problem with unequal machine speeds.” Operations Research Letters 31.3 (2003): 225-231.

[HK71] Hopcroft, John E., and Richard M. Karp. “A $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite.” Switching and Automata Theory, 1971., 12th Annual Symposium on. IEEE, 1971.

[HKS11] Halldórsson, Magnús M., Guy Kortsarz, and Maxim Sviridenko. “Sum edge coloring of multigraphs via configuration LP.” ACM Transactions on Algorithms (TALG) 7.2 (2011): 22.

[HS13] Harris, David G., and Aravind Srinivasan. “Constraint satisfaction, packet routing, and the lovasz local lemma.” Proceedings of the forty-fifth annual ACM symposium on Theory of computing. ACM, 2013.

[IM15] Im, Sungjin, and Benjamin Moseley. “Scheduling in bandwidth constrained tree networks.” Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures. ACM, (2015): 171-180.

[KKK91] Kubicka, Ewa, Grzegorz Kubicki, and Dionisios Kountanis. “Approximation algorithms for the chromatic sum.” Computing in the 90’s. Springer, New York, NY, 1991. 15-21.

[KNS12] Kowalski, Dariusz, Zeev Nutov, and Michael Segal. “Scheduling of vehicles in transportation networks.” International Workshop on Communication Technologies for Vehicles. Springer, Berlin, Heidelberg, 2012.

[KNSM14] Kowalski, Dariusz R and Nussbaum, Eyal and Segal, Michael and Milyeykovsky, Vitaly. “Scheduling problems in transportation networks of line topology.” Optimization Letters 8.2 (2014): 777-799.

[LMR94] Leighton, Frank Thomson, Bruce M. Maggs, and Satish B. Rao. “Packet routing and job-shop scheduling in O(Congestion + Dilation) steps.” Combinatorica 14.2 (1994): 167-186.

[LMR99] Leighton, Tom, Bruce Maggs, and Andrea W. Richa. “Fast algorithms for finding O(Congestion + Dilation) packet routing schedules.” Combinatorica 19.3 (1999): 375-401.

[LP86] Lovász, László and Plummer, Michael D. “Matching Theory, volume 29 of Annals of discrete mathematics.” North-Holland 5 (1986): 42-46.

[LQSY06] Li, Wenhua and Queyranne, Maurice and Sviridenko, Maxim and Yuan, Jinjiang. “Approximation algorithms for shop scheduling problems with minsum objective: A correction.” Journal of Scheduling 9.6 (2006): 569-570.

[LTY96] Leung, JY-T., Tommy W. Tam, and Gilbert H. Young. “On-line routing of real-time messages.” Real-Time Systems Symposium, 1990. Proceedings., 11th. IEEE, 1990.

[M09] Marx, Dániel. “Complexity results for minimum sum edge coloring.” Discrete Applied Mathematics 157.5 (2009): 1034-1045.

[MS11] Mastrolilli, Monaldo, and Ola Svensson. “Hardness of approximating flow and job shop scheduling problems.” Journal of the ACM (JACM) 58.5 (2011): 20.

[Pe1891] Petersen, Julius. “Die Theorie der regulären graphs.” Acta Mathematica 15.1 (1891): 193-220.

[PSW09] Peis, Britta, Martin Skutella, and Andreas Wiese. “Packet routing: Complexity and algorithms.” International Workshop on Approximation and Online Algorithms. Springer, Berlin, Heidelberg, 2009.

[PSW10] Peis, Britta, Martin Skutella, and Andreas Wiese. “Packet routing on the grid.” LATIN 2010: Theoretical Informatics (2010): 120-130.

[QS02] Queyranne, Maurice, and Maxim Sviridenko. “Approximation algorithms for shop scheduling problems with minsum objective.” Journal of Scheduling 5.4 (2002): 287-305.

[SHP98] Shakhlevich, Natalia, Han Hoogeveen, and Michael Pinedo. “Minimizing total weighted completion time in a proportionate flow shop.” Journal of Scheduling 1.3 (1998): 157-168.

[SSW94] Shmoys, David B., Clifford Stein, and Joel Wein. “Improved approximation algorithms for shop scheduling problems.” SIAM Journal on Computing 23.3 (1994): 617-632.

[Vi64] Vizing, Vadim G. “On an estimate of the chromatic class of a p-graph.” Diskret. Analiz 3.7 (1964): 25-30.

[We01] West, Douglas Brent. “Introduction to graph theory.” Vol. 2. Upper Saddle River: Prentice hall, 2001.