December 2018

Scheduling Two Machines with Dissimilar Costs

Madhurupa Moitra
madhurupamoitra@gmail.com

SCHEDULING TWO MACHINES WITH DISSIMILAR COSTS

By

Madhurupa Moitra

Bachelor of Technology, Computer Science
Budge Budge Institute of Technology
West Bengal University of Technology, India
2016

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2018

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

November 15, 2018

This thesis prepared by

Madhurupa Moitra

entitled

Scheduling Two Machines with Dissimilar Costs

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Wolfgang Bein, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Lawrence L. Larmore, Ph.D.
Examination Committee Co-Chair

Laxmi Gewali, Ph.D.
Examination Committee Member

Venkatesan Muthukumar, Ph.D.
Graduate College Faculty Representative

ABSTRACT

Scheduling Two Machines with Dissimilar Costs

By

Madhurupa Moitra

Dr. Wolfgang Bein
Committee Chair
Professor
Department of Computer Science
University of Nevada, Las Vegas

Dr. Lawrence L. Larmore
Committee Co-Chair
Professor
Department of Computer Science
University of Nevada, Las Vegas

We consider two devices, which has states ON and OFF. In the ON state, the devices use their full power whereas in the OFF state the devices consume no energy but a constant cost is associated with switching back to ON. Such two devices are configured with different run and power-up costs on which a sequence of jobs must be processed. The object is to minimize the cost. Such systems are widely used to conserve energy, for example, to speed scale CPUs, to control data centers, or to manage renewable energy.

The problems are studied in the framework of online competitive analysis and we analyze a number of online algorithm and give lower bounds. The objective of the algorithm is to optimize the budget and analyzing how effectively it works.

ACKNOWLEDGEMENT

No endeavor is effort of an individual. I would like to thank my advisor, Dr. Wolfgang Bein with all of his mentorship throughout this entire process. His dedication, guidance, and friendly demeanor inspired me to strive ahead in my research and also inspired me to grow as a person. I would like to thank my co-mentor Dr. Lawrence L. Larmore, who not only provided his support but also provided his valuable guidance on the research. His willingness to always give me advice in various stages of my research and my academic career will not be forgotten and I will always be grateful.

I am also grateful to the members of my committee for their patience and support in overcoming numerous obstacles I have been facing through my research. I am extremely grateful to Dr. Laxmi Gewali too, with his direction, support and good nature, pursuing my graduate studies seemed easier. Anytime I required his advice he was always present no matter how busy he was. I would also like to thank Dr. Venkatesan Muthukumar upon agreeing to be on my thesis committee in spite of his extremely busy schedule.

I would also like to express my deep gratitude towards my parents Mrs. Mira Moitra and Mr. Moloy Kumar Moitra who has provided moral and emotional support in my life and non-stop encouragement throughout my years of study. I would also like to thank my sister Satarupa and my brother-in law Supratik for their consistent enthusiasm no matter what, without their support it all would have been a different story.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Let us consider a scenario where there is a moderately populated town. In order to meet their demand for power supply, the town uses renewable sources such as the wind energy and solar energy. However, such sources are not always consistent as shown in fig 1.1.

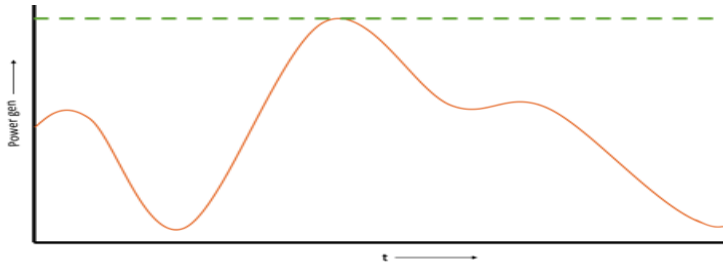

fig 1.1 Power generation by renewable sources

In the figure, the power generation via the renewable sources are shown, where the green dotted line is the power consumption demand by the town, which is assumed to be constant for simplification purposes. As we can see the power demand supply is not properly met in some cases. Therefore, a power plant, using non-renewable sources, is used in case of any power shortage. Let us say there are 2 generators at the power plant, G_1 and G_2. On sunny and windy days when the power generation is high as shown in fig 1.2, the power plant is not used. However, on other days, when either of the renewable energy sources are not enough, due to the low power generation, G_1 and G_2 are put to use.

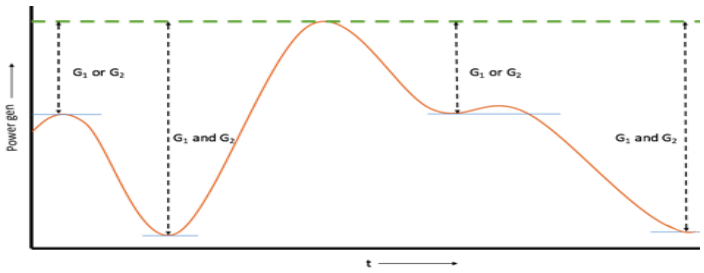

fig 1.2 Power generation lags fulfilled by non-renewable sources

Both the generators are scheduled according to the power generation lag. For example, when either of the energy sources fail to meet the consumption line, either of the two generators would be used. Similarly, in the figure above, the usage of the generators is not consistent and is required to make decision of which generator is to be used, hence the power generation lag can be treated as power assignments for the generators. Such as,

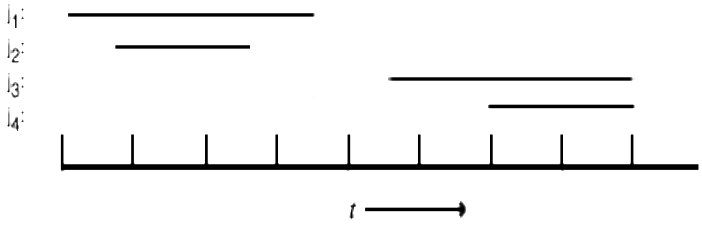

fig 1.3 Power generation lag as power assignments based on fig 1.2

Where j_i is a set of n power assignments to be assigned to the generators, G_1 and G_2. This is a typical problem in machine scheduling. It is modeled as follows: For the processing of such assignments; the 2 machines are available from time zero onwards that can handle only one assignment at a time. Each ongoing assignment cannot be stopped or be interrupted but has to be passed on to the next available machine, such that no two jobs would be handled by the same machine at the same time. Each machine has different startup costs (σ_1,σ_2) and runtime costs(ρ_1, ρ_2). Let us say, one machine's startup cost is expensive

than the other, ($\sigma_1 <= \sigma_2$). We are looking for a feasible solution for the schedule of the machines, that is, allocating each job J of a time interval of t_j, so that no two jobs would be handled by the same machine at the same time. Where online scheduling algorithm would mean, the machines have no knowledge of the upcoming inputs, and process accordingly. Given a feasible schedule, the objective is to find such an online algorithm where all the jobs are completed at the lowest possible cost.

To define the problem, in other words, let us consider a set of requests defined by:

$$J(n) = j_1, j_2, j_3, j_4, j_5, \ldots, j_n,$$

where n is the number of requests and the time interval,

$$j_i = (a_i, l_i),$$

where j_i is the ordered pair (a_i, l_i) that is, the arrival timestamp and the length of the job for the i^{th} interval and such that $a_i <= a_j$ for all i<j. With the pair, we will compute the s_i and f_i, where s_i is the start time and f_i is the finish time for the i^{th} job. There are two machines: Machine 1 and Machine 2 on which every j_i on the request sequence are to be processed. Also, every request sequence J(n) must have the property such that no three jobs should overlap, that means $[s_i, f_i[\cap [s_j, f_j[\cap [s_k, f_k[= \varnothing$ for any three distinct indices i, j, k.

Initially, every machine is assumed to be off, and that a machine must be turned on in order to accept a job. After the job is finished, the machine can be turned off immediately or left running idle, after which it will eventually accept another job or be turned off. We assume that Machine 1 costs σ_1 to turn on and ρ_1 per unit time to run, either with or without a job, while Machine 2 costs σ_2 to turn on and ρ_2 per unit time to run. It costs nothing to load a

new job onto an idle machine, and nothing to turn a machine off. The objective is to find such an online algorithm which gives out the lowest possible cost with every input adversary throws in.

1.2 Motivation

The challenge of solving environmental issues has changed many aspects of the way things operate. Power consumption in day to day life has become mandatory to us and is increasing. Even though the use of non-renewable sources is being rapidly being replaced with alternate sources. While there have been studies with exploring options for integrating sustainable and renewable energy into new or existing power grid models. Smart grid infrastructure is being designed and introduced to reduce the dependence on conventional power generation sources. However, minimizing the power consumption is still the main concern.

According to an IEA estimate, there has been a usage of 567 x 10^{20} joules of energy in 2013, equivalent to about 18.0 terawatt-hour(TWh). In this paper, given the set of incoming requests, given the idle times, the goal is to minimize the total power consumption of machines working from turning on until it shuts down while processing all the jobs in line. While scheduling the requests onto the machines is quite trivial when the given set of requests are already known. However, for this paper, we will be considering the online model, where the set of requests are unknown to the machines until it arrives. And we must decide which machine should be processing the current request.

What drives the research for this paper is that we need such an online algorithm which when compared with the optimal offline algorithm, during the competitive analysis, the lower bound and the upper bound are closer, hence providing the minimal cost at every set of inputs the adversary throws in.

1.3 Contribution

In this paper, we are considering two machines with different run and power-ups costs on which a sequence of jobs must be processed. The object is to minimize the cost. We develop a dynamic program to solve this problem optimally. However, in practice, the sequence of jobs is not known in advance; instead, the jobs arrive one at a time, and a decision as to which machine will process the jobs has to be made online, i.e., before the next job arrives. The problem is a generalization of the noted one machine power-down problem, which is important in the area of green computing. We analyze a number of online algorithms and give lower bounds. We also present a number of open problems. The sole objective of the algorithm is to optimize the budget and to analyze how effectively it works and determining the lower bound on the performance of the algorithm to that of the optimal offline and extensive simulation to find the upper bound on the worst-case scenario that the adversary can put in.

1.4 Outline

In chapter 2, we discuss offline algorithm at length, how the offline task allocation takes place. Furthermore, we will introduce a mechanism of scheduling the requests using effective scheduling techniques, such as the dynamic offline scheduling using layered

graph approach and backtracking, to obtain the minimal optimal cost for every said request. Chapter 3 stresses the online algorithm to show how different it is from offline algorithm, we describe more about the algorithm and how they difference. The chapter is more on the theoretical analysis of the online algorithms with certain useful key terms.

Chapter 4 discusses the bounds for the competitiveness, most importantly the simple lower bound. Then we propose a canonical form of the online algorithm and furthermore, provide an improved version of the same algorithm. In the later section of the chapter, it consists of extensive simulation using different criterions and inputs to finally compare the performance of the online algorithm and that of the optimal offline algorithm. Chapter 5 reflects the conclusion and future work. It emphasizes the accomplishments achieved and possible future work at the same.

CHAPTER 2

OFFLINE ALGORITHM

2.1 Definition

Let us consider two machines each with an ON state and an OFF state. ON is such a state where the machine is working until turned OFF. Each state has a base cost or unit cost. Each machine, in the OFF state, consumes zero energy; whereas it consumes up to the full energy potential in the ON state. The running cost of the machines in the ON state is proportional to the usage of the time. The machines must be in the ON state in order to process a demand. However, if one machine is in some other state while a request arrives, and the other machine is not available, it needs to control up to the ON state, else the request can be processed by the other machine. After the fulfillment of the demand, the machine can revert back to being idle or the OFF state yet will bring about a power-up cost each time a demand touches base after the machine changed to the lower control state. Moreover, when the machine is in ON state processing a request, it cannot be interrupted by another incoming request. The request may have to be delayed for processing or passed to the next available machine.

In between the two states ON and OFF, the machines can process as many requests as they arrive. With all the pointers keeping in mind, the objective is to limit the power utilization expected to process all the demands. There are two types of algorithms dedicated to

machine scheduling; one being the offline algorithm. Offline schedule algorithms are often applied to such set of inputs already given, in which we already are aware of what is yet to come. Offline algorithms or also called OPT to give the optimal schedule results since the set of inputs are already known, hence is easier to strategize the inputs accordingly for processing such that the power utilization is minimum at the optimal condition. In this paper, we will discuss more the offline optimal algorithm, and how the algorithm schedules each input request sequence at length using layered approach.

In this algorithm, we assume that there are two machines A and B, where the startup or the ignition cost for the machines is defined by σ, more likely as σ_A and σ_B without loss of generality $\sigma_A <= \sigma_B$. Each machine has a runtime cost which we name as ρ, which is the usual cost bore by the machines while being turned on and running. We name the runtime cost for machine A and machine B as ρ_A and ρ_B. As we already know about the set of requests arrive it is easier to determine how the incoming requests can be distributed. Following on, we will discuss the offline scheduling algorithm at length.

We first redefine the input sequence in such a way that there are only two machines required at any given time.

2.2 No-Overlap Pre-processing

Let us consider a set of requests defined by:

$$J(n) = j_1, j_2, j_3, j_4, j_5, \ldots, j_n,$$

where n is the number of requests and the time interval,

$$j_i = (a_i, l_i),$$

where j_i is the ordered pair (a_i, l_i) that is, the arrival timestamp and the length of the job for the i^{th} interval. With the pair, we will compute the s_i and f_i, where s_i is the start time and f_i is the finish time for the i^{th} job.

J is a set of jobs <s_i, f_i> with the following operators:

- makeempty(J) initializes J to be empty.
- min(J) returns the finishing time of the item on J having minimum finish time but does not delete it.
- deletemin(J) deletes the item of J having minimum finish time, but does not return a value.
- size(J) returns the number of items in J. The size of J is never more than 3, the number of machines plus 1.
- insertJ(s, l) inserts a job <s, s+l> into J

This algorithm schedules the jobs and computes the values of s_i and f_i:

```
makeempty(J)
   insertJ(0, infinity)
   for i := 1 to n do
     if min(J) le ai
     then si := ai
          deletemin(J)
          insertJ(si, li)
     else if size(J) < 3
     then si := ai
```

insertJ(s_i, l_i)

else

s_i := min(J)

deletemin(J)

insertJ(s_i, l_i)

The above algorithm redefines the input sequence in such a way that at any given time there are only two machines required. In the next section, we will discuss weighted layered graph and its applications on the optimal scheduling with certain examples.

2.3 The Offline Two Machine Scheduling Problem

We consider a sample request sequence J, which is a set of jobs $j_i = (s_i, l_i)$ and is defined by:

$$J(n) = j_1, j_2, j_3, j_4,$$

Where $f_i = s_i + l_i$,

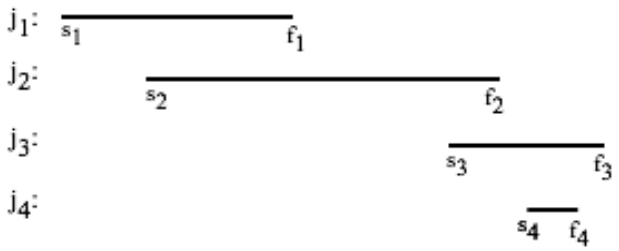

fig 2.1 Request interval for each job, which begins at s_i and ends at f_i.

At any given time, each machine, say Machine 1 and Machine 2, is in one of the three states: off, idle or working. We introduce critical points, c_k, which defines the change in the machine states, where k is the number of state changes, as shown fig 2.2 below:

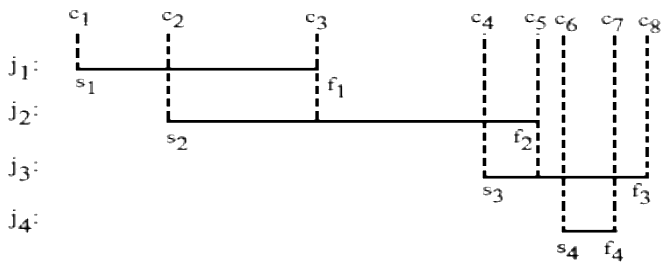

fig 2.2 Machine state changes marked by critical points

such that,

$$s_1 = c_1 < s_2 = c_2 < s_3 = c_3 < s_4 = c_4 < s_5 = c_5 < s_6 = c_6 < s_7 = c_7 < s_8 = c_8$$

If c_k and c_{k+1} are consecutive critical points, and Δ corresponds to the length of the maximal time interval during which the machine configuration does not change, such that,

$$\Delta_k = c_{k+1} - c_k$$

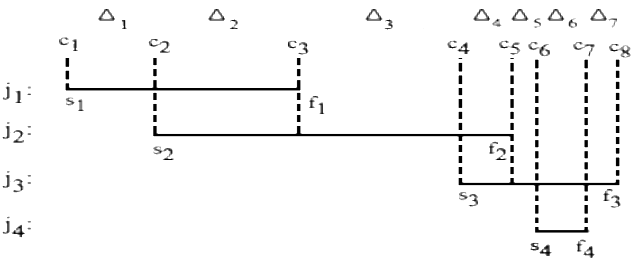

fig 2.3 Maximal time intervals marked by Δ

The machine states: off, idle and working can be abbreviated as F, I and W. Since there are two machines, there are nine combinations of states of the two machines, abbreviated as FF, IF, WF, FI, II, WI, FW, IW and WW, where the first letter refers to Machine 1. However, we split the state WW into two configurations, which we call as AB and BA. For example, if two jobs running are j_i and j_y, where $i < y$, we say j_i is the older job and j_y is the newer job. Such that, AB refers to such a configuration if the older job is on Machine 1 and BA, when the older job is on machine 2. Thus there are ten machine configurations.

There are five types of critical points:

- A critical point c has type 01 if there is no job running before c and one job starts at c.
- A critical point c has type 10 if there is one job running before c and that job finishes at c.
- A critical point c has type 12 if there is one job running before c and another job starts at c.
- A critical point c has type 21A if there are two jobs running before c and the older job finishes at c.
- A critical point c has type 21B if there are two jobs running before c and the newer job finishes at c.

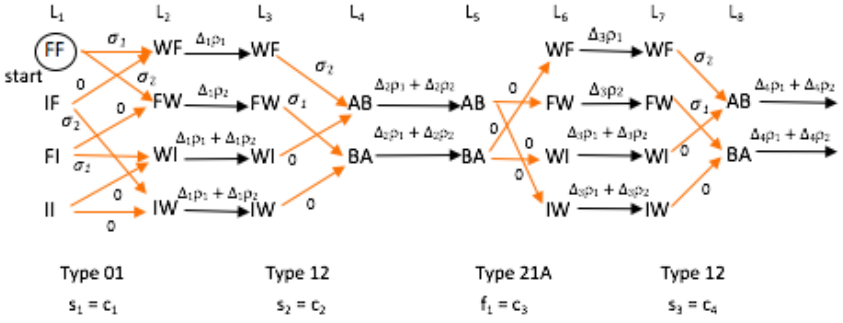

fig 2.4a Weighted Layered Graph R for the request sequence at fig 2.1

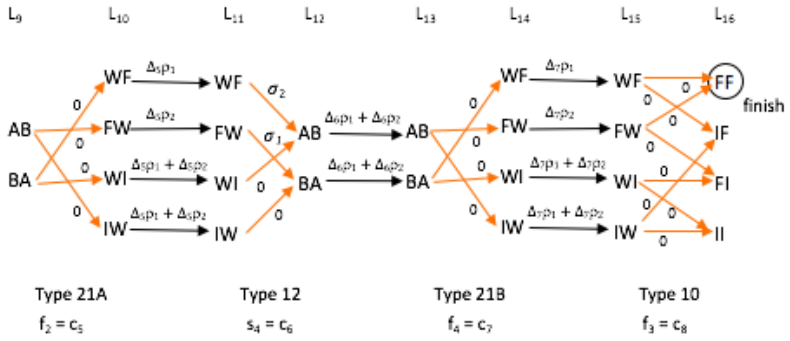

fig 2.4b Weighted Layered Graph R for the request sequence at fig 2.1(contd.)

In the figure above, L represents layers of the graph. Graph R has two consecutive layers for each critical point c, L_c^- and L_c^+. Each layer is a copy of C_m, where C_m be the set of machine configurations for m jobs running. For m = 0, 1, 2, C_0=[FF, IF, F1, II], C_1=[WF, FW, WI, IW] and C_2 = [AB, BA].

- If c has type 01, then L_c^- is a copy of C_0 and L_c^+ is a copy of C_1.
- If c has type 10, then L_c^- is a copy of C_1 and L_c^+ is a copy of C_0.
- If c has type 12, then L_c^- is a copy of C_1 and L_c^+ is a copy of C_2.
- If c has type 21A or 21B, then L_c^- is a copy of C_2 and L_c^+ is a copy of C_1.

The weights of the vertices joining L_{ck}^- and L_{ck+1}^+ is

- 0 if the machine configuration is FF during the interval.
- $\Delta\rho_1$ if the machine configuration is IF or WF during the interval.
- $\Delta\rho_2$ if the machine configuration is FI or FW during the interval.
- $\Delta\rho_1 + \Delta\rho_2$ if the machine configuration is II, WI, IW, AB or BA during the interval.

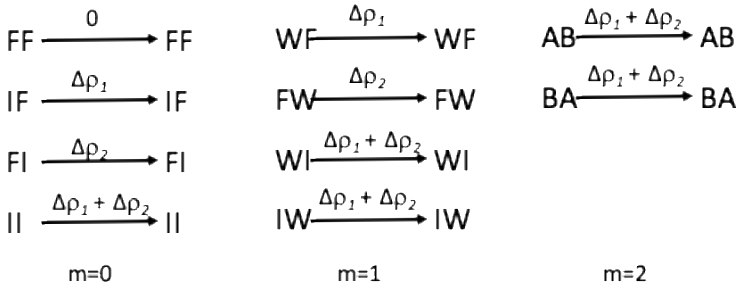

fig 2.5 Adjoined vertices, where Δ is the length of the time interval, for C_1, C_2, and C_3

2.4 Dynamic Program Layered Graph

In this section, we discuss weighted layered graph, in order to reduce the optimal schedule problem to the minimum path problem. We define a layered graph to be a directed graph

G whose set of vertices is the union of a sequence of layers, L_1, L_2,…,L_n, such that every arc of G is between consecutive layers, i.e., every arc is of the form (x,y) where $x \in L_i$ and $y \in L_{i+1}$ for some i. A weighted layered graph is a layered graph G together with a weight on each arc. If $|L_i| = O(1)$ and a start vertex in L_1 is given, the minimum path problem for G can be solved in O(k) time by dynamic programming.

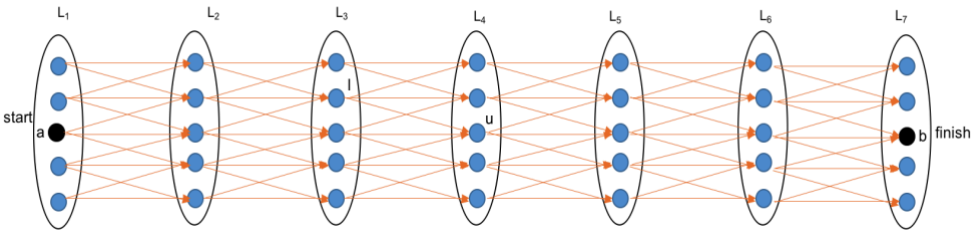

fig 2.6 Construction of layered graph G

Each arc connects vertices of successive layers. The arcs from L_{ck}^- to L_{ck+1}^+ as shown in fig 2.6, are called horizontal, they correspond to intervals between critical points. Whereas the arcs from L_c^- and L_c^+ we call vertices, they correspond to the change of the machine state configurations, which we assume takes zero time. In the fig above, there are seven layers L_i, consisting of nodes or vertices V_1, V_2, V_3, V_4, and V_5. Since layered graphs are used to find the minimum path between each layer. It is defined as,

E[V] = Shortest path from a to b

Let us assume, we already found the minimum path till L_3, and the node l has been found to be among the nodes in the minimum path. Let us particularize from node l to the next shortest path node:

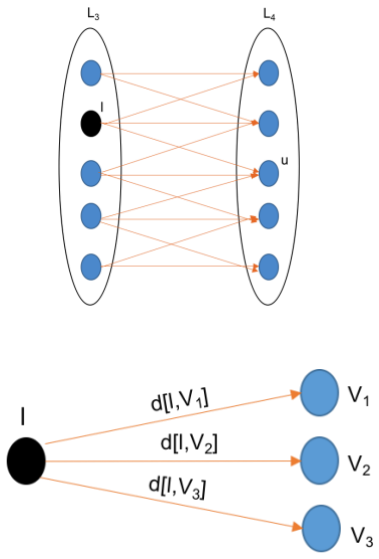

fig 2.7 Shortest distance from L_3 to L_4

Assuming we know the weights of each arcs connecting between the two layers. In the fig above, node l is connected to 3 more nodes, therefore,

Let u be the next node containing the minimum path amongst the three nodes, and let $d[l,V_i]$ be the weight of the arc. That is,

$$E[u]= \min\{E[l]+ d [l, V_1], E[l]+ d [l, V_2], E[l]+ d [l, V_3]\}$$

Calculating the above formula would provide the minimum path from node l to node u. This is, however, a small snippet of how layered graph are useful in determining the shortest distance from one layer to another. In the next section, we will try to determine minimum path using the previous example as discussed in section 2.3.

2.5 Calculating Minimum Path for each Critical Point using Layered Graph Approach

So far we have discussed how the layered graph approach works, and how can it be applied to determine the shortest path between each layer, we have also discussed various key terms

used in the graph. Arcs, as discussed earlier, are such lines which connect each vertex of successive layers. We also specified the weights of each arc as seen in the example in section 2.3. A vertical arc is between a node of L_c^- and L_c^+. Each such arc corresponds to a change of machine state configurations. Taking the same weights specified, the cost of the arc is σ_1 if Machine 1, but not Machine 2 is turned on, similarly, σ_2 is Machine 2, and not Machine 1, is turned on, and 0 if neither of the Machines is turned on. The 28 possible changes of the states, along with weights of corresponding arcs, as shown in fig 2.8. For each critical point c, the subgraph of R consisting of L_c^- and L_c^+ and the arcs between them are illustrated as one of the five subfigures, one corresponding to each of the five types of critical point.

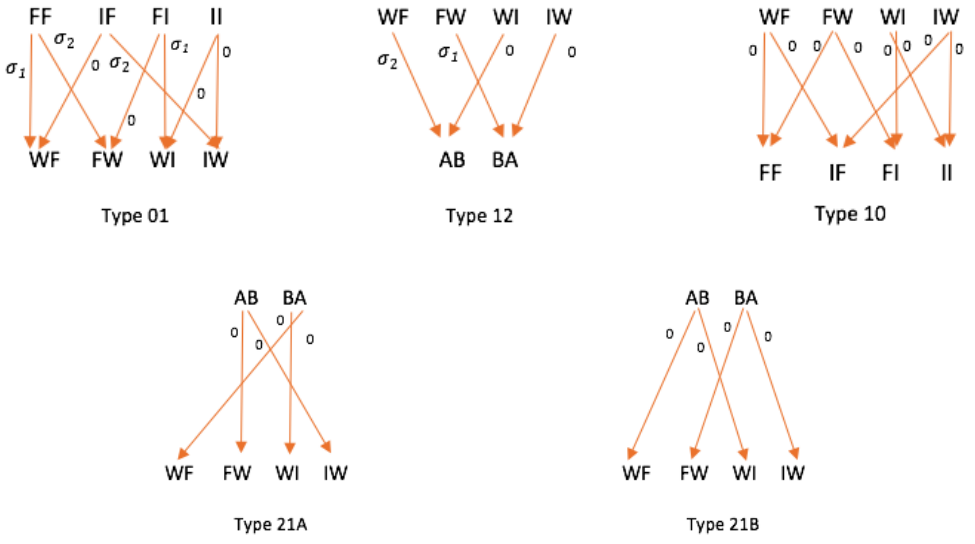

fig 2.8 Vertical arcs between layers L_c^- and L_c^+ for each of the five types of critical points

The formulations for the shortest distance of the vertical arcs is as below:

For Type 01:

$E[WF] = \min\{E[FF] + \sigma_1, E[IF]\}$

E[FW] =min{E[FF]+ σ_2 , E[FI]}

E[WI] =min{E[FI]+ σ_1 , E[II]}

E[IW] =min{E[IF]+ σ_2 , E[II]}

For Type 12:

E[AB] =min{E[WF]+ σ_2 , E[WI]}

E[BA] =min{E[FW]+ σ_1 , E[IW]}

For Type 10:

E[FF] =min{E[WF], E[FW]}

E[IF] =min{E[WF], E[IW]}

E[FI] =min{E[FW], E[WI]}

E[II] =min{E[WI], E[IW]}

For Type 21A:

E[WF] =min{E[BA]}

E[FW] =min{E[AB]}

E[WI] =min{E[BA]}

E[IW] =min{E[AB]}

For Type 21B:

E[WF] =min{E[AB]}

E[FW] =min{E[BA]}

E[WI] =min{E[AB]}

E[IW] =min{E[BA]}

Given the shortest paths for each critical point, one can successfully backtrack from the FF, as in fig 2.4b, that is the destination, to FF, as in fig 2.4a, that is the source.

CHAPTER 3

ONLINE ALGORITHM

In this chapter, we give an informal introduction to the online competitive model. We begin by revisiting a classical problem in the area of online competitive algorithms: The Ski Rental Problem. This problem is related to the one-machine version of the problem studied in this thesis. The chapter then continues to elaborate further on the one-machine problem and uses this scenario to illustrate the terminology around online competitive analysis.

3.1 Ski Rental Problem

We consider such a machine with two states, that are the ON and OFF state. The problem we will be referring to is to a well-known problem: ski-rental problem. [1, 5, 6] The ski-rental problem introduces a scenario where you are new to skiing, and you ask yourself if you are to buy or rent the skis. Ignoring the other factors, such as which model are you going for, or which ski place you want to go, the problem arises when you are not aware of how often will you go skiing. Since the future is quite unclear, you must make a decision which would seem viable in terms of being financially better off at that point.

In this problem, we need to decide whether one should keep on renting every time, or switch to buying the skis irrespective of the frequency one goes for skiing. The decision we take affects how the present works out and how the future will turn out to be. The basic version of the problem can be explained as that you are suddenly invited by a friend for

skiing, who by instance did not inform you about the days you will be staying and skiing, due to a number of factors such as extremely bad weather.

Since you are going skiing for an unknown number of days, let us assume, if one chooses to rent skis, they have to pay $30, whereas buying them costs $300. If you know how many times you would go skiing, making the decision would have been clearer. If you are to ski at least 10 times, buying the pair of skis would seem to be a viable option, whereas, if you are to go less than 10 times, you would be financially better off renting the skis every time. But due to the uncertainty in the future, how would you proceed with the decision, and what decision would it be?

3.2 Online Algorithm

[10] In other words, you are faced with an online problem, as the problem arrives without any knowledge of future events. Offline problems are such problems where we already know what is yet to come, hence makes it easier to react at the present point of time. These problems are somehow of ideal conditions and give out the optimal solution. Here the other type of power down strategy comes in, online problems dealing with real time. In an online problem, data is continuously supplied as the input; data supplied at time i, would give out the i^{th} output. Since outputs are formed without complete knowledge of the future inputs, online algorithms cannot produce the optimal solution. The main goal of an online algorithm is to approximate the performance by an offline algorithm, which gives out the optimal results. Thus, online algorithms are said to be competitive if its performance is close to that of the optimal offline algorithm on each input.

An online algorithm is such an algorithm which should satisfy an online problem. Online algorithms are in a way approximation algorithms, where they tend to approximate the performance of an optimal offline algorithm for the same problem. An optimal offline algorithm is an algorithm which is aware of the future events in its entirety and deals with the problem at a better financial position.

In the ski rental problem, one needs to choose whether to rent the skis or to buy them, depending on how frequent one is to go skiing. The online algorithm tries to determine whether to rent the ski or to buy them, that is whether to pay a small chunk every time one goes for skiing or to pay a hefty sum of money and be as is. When an online algorithm is analyzed, the performance of the algorithm will be gauged with the help of competitive analysis. Since competitive analysis is a method for worst-case analysis, the adversary is called in.

Regardless of any strategies used, the adversary determines exactly at what point you will never go skiing again. Hence, making the online algorithm to pay more than what the optimal offline algorithm. Below are two such online algorithm strategies where the adversary makes the frequency of skiing more or less with a view to creating the worst possible outcomes by the online algorithm.

A. Always OFF Algorithm

In an online algorithm scheduling, since we are not always sure about the future events that might take place. [3] One approach is to always turn off the machine after every other processing of requests coming in. This has a competitive ratio of infinity. Though it works

perfectly fine if the interval between the requests is quite apart. The algorithm falls apart when the requests are consecutive.

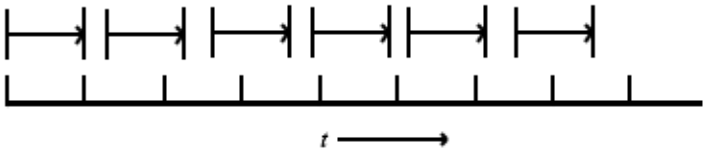

Fig 3.1 Always OFF approach for the worst-case analysis

In the fig above, it shows how every job after its completion, the machine switches to its OFF state. If we are to apply the ski rental problem to the algorithm and increasing the frequency of the above set of requests or skiing, for example, if renting a pair of ski costs $30 and buying costs $300, and Always OFF algorithm in relation to the problem means renting the ski every time. So the adversary increases the frequency of going for skiing. Let the frequency be j. If j <= 10, the cost of renting would be 30j, which is, 30j <= 300. But if j>10, instead of paying 300, the algorithm ends up paying 30j which twice than that of the optimal offline algorithm.

B. Always ON Algorithm

While scheduling an online algorithm, another strategy is to keep the machines on after they have been turned on. Staying ON between the requests has a competitive ratio of infinity. With requests being consequent that provide for the best case. But the worst-case would when the interval between the requests is quite big.

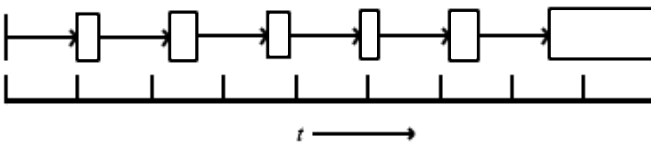

Fig 3.2 Always ON approach for worst-case analysis

The Always ON algorithm works in opposition to the previous algorithm, the fug above shows that after processing every job request in the sequence, the machine is kept ON. Though it works perfectly fine for the first six incoming sequences. However, the algorithm completely falls apart when there are no incoming requests. In the ski rental scenario, when renting a pair of skis is $30 and buying them is $300, the online algorithm decides to buy the ski, the adversary now makes sure that one never goes skiing.

In other words, let the frequency be j. If j <= 10, the cost of renting would be 30j, which is, 30j<=300. According to the optimal offline cost, one can pay min (30j, 300), since the algorithm ends up paying 300, the adversary makes it a point of never going skiing. If j = 1, instead of paying 30, the algorithm pays 300, ten times what the optimal offline algorithm paid.

We can clearly see that for a machine with 2 states, OFF and ON, the power-down problem has a competitive ratio of 2. The worst-case occurs when:

- The machine turns off just before the next request.
- The machine stays on and requests do not arrive.

Optimal offline algorithms due to their added advantage of having knowledge on the future. It provides the minimum cost possible. Since we mostly deal with real-time situations, we are always faced with online problems. Although it may seem like a good idea for the online algorithm to take such an action which is cheaper in the short term, however, the algorithm might have to pay more with all the cheaper actions taken together. Or it may take an expensive action, and pay less in the future activities. This principle underlying the algorithm can also be called the ski principle. The principle also says once the online

algorithm has incurred enough cost taking up cheaper decisions, the algorithm can also afford to take expensive decisions in that essence.

3.3 Online vs OPT

The optimal offline algorithm or OPT can be contrasted to online algorithms, as such an algorithm which already has full knowledge of the entire set of requests. Since it has the entire set of the incoming requests, it already knows how long it would require to process each request and how can it schedule it so that it can the minimum cost possible. An offline algorithm also is required to take an action in response to each request, but the choice of each action can depend on either the entire set of request sequence or the request itself. In other words, an offline algorithm knows the future whereas, the online algorithm does not. For example, if we take sorting algorithms into consideration; say, selection sort. Selection sort repetitively takes the minimum element from the remaining unsorted set of inputs and places it at the beginning, which demands access to the entire input. It is thus an offline algorithm. An offline algorithm is given the entire problem data from the start and is expected to provide an output which solves the problem at hand.

Taking the same example for sorting algorithms, as explained earlier selection sort work as an offline algorithm. On the other thing, insertion sort deems only one element at a time every iteration giving out partial output every time until the entire set of future inputs are met. Hence, insertion sort is typically an online algorithm. Even though, insertion sort gives out the optimal solution in this case. However, this may not be the case for every problem, where online algorithms cannot compete with the performance of an offline counterpart.

We use the term competitive if the ratio of the performance of an online algorithm to the performance of an optimal offline algorithm is bounded.

3.4 Key Terms

3.4.1 Competitive Analysis

[10,11] Defining a measure of performance of any online algorithm is more challenging, since whatever decision the algorithm takes, usually ends up affecting the rest of the unseen incoming requests. One approach is to evaluate online algorithms is to assume a specific stochastic model of the source of requests. With such models, online algorithms may perform optimally, if it chooses its actions so as to minimize the cost, where the cost depends on the set of requests generated by the stochastic source and the decisions taken by the online algorithm as it proceeded. However, stochastic models are not always consistent with their generation of requests and the choice on the model depends on the data that may not be readily available about the request sequence observed in the past.

This paper stresses the alternative to the stochastic models, a worst-case approach in which online algorithm is evaluated by comparing its cost with that of an offline counterpart processing the same set of sequence optimally. We follow, which we define as the competitive ratio of an online algorithm, over all possible input sequences, of the ratio between the cost incurred by the online algorithm and the cost incurred by the optimal offline algorithm. Though the approach of competitive ratio avoids the commitment to any stochastic models. However, the approach is always cynical about the performance of the

online algorithm inducing the minimax regret concept in game theory, in effect the set of requests would be chosen by an all-knowing clever adversary.

For many algorithms, performance is dependent not only on the size of the inputs but on their values. For example, a quicksort algorithm, such data-dependent algorithms are analyzed for their average case and worst-case data. Competitive analysis is a method for worst-case analysis for online and randomized algorithms, which are usually data-dependent. In other words, competitive analysis is a method invented for analyzing data-driven algorithms such as online algorithms, which must satisfy an unpredictable sequence of requests, as compared to the performance of an optimal offline algorithm that can view the sequence of the incoming requests in advance. An online algorithm is said to be competitive if its competitive ratio, which is further discussed in 3.4.2, is bounded.

3.4.2 Competitive Ratio

As discussed in 3.4.1, to check if the online algorithm is competitive, we need to evaluate the performance of the algorithm with that of an optimal offline algorithm. The idea of computing the competitive ratio is to place an online algorithm in competition with such an algorithm that receives more information. A competitive ratio can be defined as,

$$\max\left\{\frac{\text{Cost of the online algorithm}}{\text{Cost of the optimal ofline algorithm}}\right\}$$

where the cost is defined as the total resource usage by both the algorithms in order to process any given set of requests.

3.4.3 Adversary

When an online algorithm compares its competitiveness, that is when an adversary comes in. The adversary generates a request sequence for both the online algorithm and itself. For the type of algorithms, the competitiveness can depend on the adversary models used. There are three common adversaries:

The **oblivious adversary** is also referred to as the weak adversary. This adversary knows how the online algorithm works or is coded but does not get to know the randomized results of the algorithm. Whereas, the **adaptive online adversary** or the medium adversary, must make its own decision before it is allowed to know the decision of the algorithm. And lastly, the **adaptive offline adversary**, also known as the strong adversary, knows everything, even the arbitrary number generator. The adversary is so strong that randomization does not help against it.

3.4.4 Lower Bound

While performing the competitive analysis of the algorithms, since we work on the data-driven algorithms, competitive ratios are different for every set of inputs. Determining the best-case, average-case and the worst-case ratios will help us determine the actual performance of the online algorithm we are dealing with. Therefore, we take bounds. Being lower bounded on an algorithm is used to indicate the lowest growth rate. In other words, lower bound on an algorithm is how it would perform in any scenario with least amount of resources, that is, in terms of cost, it will produce (at least) no less than the given bound.

3.4.5 Upper Bound

Though, lower bound is important to be taken into account, but we will, however, know how the algorithm works in the worst possible cases, and this paper will stress on the upper bound on an algorithm. Upper bounds on an algorithm mean, the algorithm solves a set of requests (at most) no more than the given bound, that is, the best the algorithm can do.

CHAPTER 4

COMPARISON BETWEEN OPT AND ONLINE ALGORITHM

So far we have discussed about both the types of algorithms. Offline problems are such problems where we already know what is yet to come, hence makes it easier to react at the present point of time. These problems are somehow of ideal conditions and give out the optimal solution. Whereas, online problems deal with real time. In an online problem, data is continuously supplied as the input; data supplied at time i, would give out the i^{th} output. Since outputs are formed without complete knowledge of the future inputs, online algorithms cannot produce the optimal solution. The main goal of an online algorithm is to approximate the performance by an offline algorithm, which gives out the optimal results. Thus, online algorithms are said to be competitive if its performance is close to that of the optimal offline algorithm on each input.

In this chapter, we use both the algorithms to show experimental results depending on the schedules of the two machines resulting in the final costs respectively. Both the Online algorithm and Optimal Offline algorithm OPT are run against a set of arbitrary set of request sequence

4.1 The Simple Lower Bound

Let us first consider the case that the execution lengths l_i of a job j_i is not known in advance. The standard 2-machine scheduling problem with start costs 0 already gives a lower bound depending on the ratio between the execution costs of both machines, which is as

Theorem 5.1 The competitiveness of any online algorithm S is at least $\max\{\rho_1/\rho_2, \rho_2/\rho_1\}$.

Proof: Without loss of generality one may assume $\rho_1 \geq \rho_2$. We define a simple request sequence I of 2 jobs and an adversary A against which no online algorithm S can be less than ρ_1/ρ_2 -competitive. Both jobs start almost simultaneously and overlap. S has to decide whether to schedule j_1 on Machine 1 or Machine 2 and then j_2 has to go to the other machine. Depending on this decision A then chooses l_i to be very large for the job executed on Machine 1 and l_j very small for the other job. A schedules both jobs just the other way around.

This gives $cost_S(I) \geq \sigma_1 + \sigma_2 + \rho_1 l + \rho_2 l'$, where $l >> l'$, and $cost_A(I) = \sigma_1 + \sigma_2 + \rho_1 l' + \rho_2 l$. Since l can be chosen arbitrarily large the ratio $cost_S(I)/ cost_A(I)$ cannot be bounded by anything less than ρ_1 / ρ_2.

This lower bound can be extended to the case of known execution lengths by introducing a third job j_0 of small fixed length l_0 that starts before the two other jobs. j_1 starts shortly before j_0 is finished, and j_2 shortly after the execution of j_0. Now, the decision where to schedule j_0 determines the machines for j_1, j_2, but at that time their execution lengths are not known to S.

4.2 Proposed Online Scheduling Algorithms

As discussed, an online algorithm is one that can process its input one by one in a sequential manner. An online algorithm can be implemented in several other ways. Some examples include insertion sort, greedy algorithm, page replacement algorithm and so on. Let us now define a new online algorithm for which we will later provide simulation results with a contrast to its offline counterpart.

We consider two machines: Machine 1 and Machine 2. Each machine has their own respective startup costs named as (σ_1, σ_2) without loss of generality $\sigma_1 <= \sigma_2$; and runtime costs namely, (ρ_1, ρ_2). Let us a naïve approach for the scheduling algorithm. We will name the algorithm as the canonical online scheduling algorithm, the pseudocode is defined as below:

Some definitions:

dt_1 and dt_2 = difference

$temp_1$ and $temp_2$ = e[i] (the end time of the processed job)

$span_1$ and $span_2$ = e[i] + idle time

s[i] = start time of the current job

- If Machine 1 is OFF, 2 is running, turn on Machine 1
- If Machine 2 is OFF, 1 is running, turn on Machine 2
- If Machine 1 is OFF, 2 is idle, use Machine 2
- If Machine 2 is OFF, 1 is idle, use Machine 1
- If Machine 1 is idle, machine 2 is running, use Machine 1

- If machine 2 is idle, machine 1 is running, use machine 2
- if machine 1 and machine 2 are OFF, use machine 1
- if machine 1 and machine 2 are idle,
    - Check the duration since the last processed job

{ if((s[i]<= $span_1$) && (s[i]<= $span_2$))

dt_1=s[i]- $temp_1$

dt_2=s[i]- $temp_2$

if (dt_1<= dt_2)

use machine 1

else

use machine 2 }

- if machine 1 and machine 2 is working, a new job arrives
    - Check the duration of the jobs:

{ if((s[i]<= $temp_1$) && (s[i]<= $temp_2$))

dt_1= $temp_1$-s[i]

dt_2= $temp_2$-s[i]

if (dt_1<= dt_2)

use machine 1 (after it has finished processing the current job)

else

use machine 2 (after it has finished processing the current job) }

Since the idle time value is always an arbitrary number, we introduce another algorithm where we can always determine the idle to be constant corresponding to the startup and the runtime costs. The pseudocode of the new improved scheduling algorithm is defined as below:

- If machine 1 is available, use that machine.
- If machine 2 is available, use that machine.
- If machine 1 is idle, 2 is OFF: use machine 1.
- If machine 2 is idle, 1 is OFF: use machine 2.
- If both machine 1 and machine 2 are idle, then:
    - Compare the runtime cost for the machines, that is,

      If $\rho_1 > \rho_2$, use machine 2.

      If $\rho_1 <= \rho_2$, use machine 1.
- If both machine 1 and machine 2 are OFF, then:
    - Compare the start-up cost for the machines, that is,

      If $\rho_1 <= \rho_2$, use machine 1

      If $\rho_1 > \rho_2$,

      If $\rho_1/\rho_2 >= \sigma_1/\sigma_2$, use machine 1

      If $\rho_1/\rho_2 < \sigma_1/\sigma_2$, use machine 2
- For all jobs, once the job ends, then after job on machine 1, machine 1 runs idle for σ_1/ρ_1.
- For all jobs, once the job ends, then after job on machine 2, machine 2 runs idle for σ_2/ρ_2.

In the following section, we will take a look at the different algorithms through extensive simulations.

4.3 Simulation Results

4.3.1 Canonical Online Algorithm versus Offline Algorithm

We will take a look at the comparison of the performance of the canonical online algorithm with the OPT, and how it behaves when introduced different set of idle time values:

With the parameters set as,

σ_1	σ_2	ρ_1	ρ_2
1.00	1.50	1.00	1.00

Table 4.3.1.1 Parameters set I

The Lower Bound for this parameter should be at least $\max\{\rho_1/\rho_2, \rho_2/\rho_1\}$

Therefore, the competitive ratio should be at least 1.

Input Sequence 1:

s_i	e_i
0.00	5.00
0.10	0.20
1.80	1.90
3.50	3.60

Table 4.3.1.2 Input Sequence 1

Below is the cost comparison and competitive ratio based on the idle times taken, we will be generating the idle time values using arbitrary values and σ/ρ respectively,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.00	14.80	9.80	1.510
1.00	1.50	16.30	9.80	1.663

Table 4.3.1.3 Cost comparison and competitive ratio based on Input Sequence 1

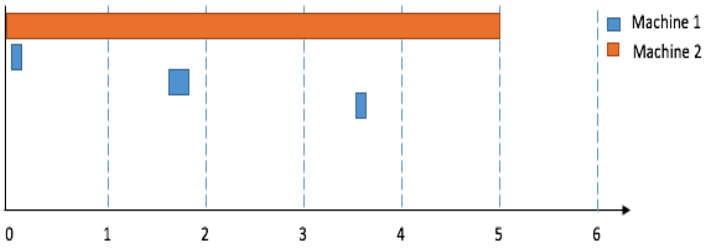

Fig 4.3.1.1 OPT for Input Sequence 1

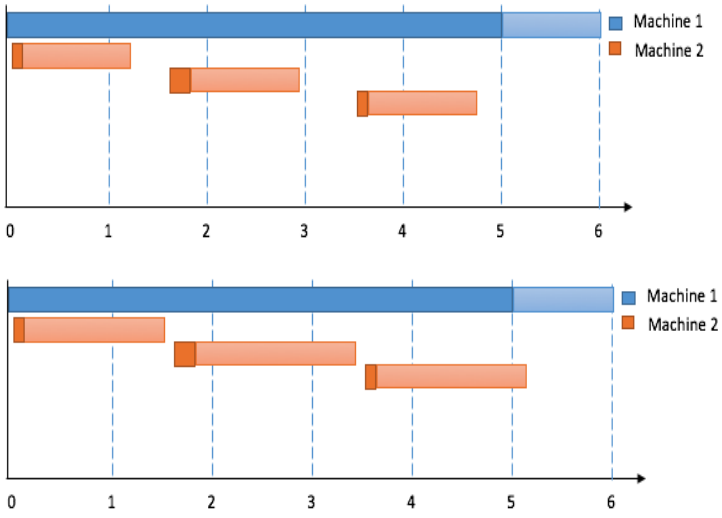

Fig 4.3.1.2 Canonical Online Algorithm for Input Sequence 1

Input Sequence 2:

s_i	e_i
0.00	5.00
0.01	0.02
2.03	2.04
4.05	4.06

Table 4.3.1.4 Input Sequence 2

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.00	14.53	9.53	1.524
1.00	1.50	16.03	9.53	1.682

Table 4.3.1.5 Cost comparison and competitive ratio based on Input Sequence 2

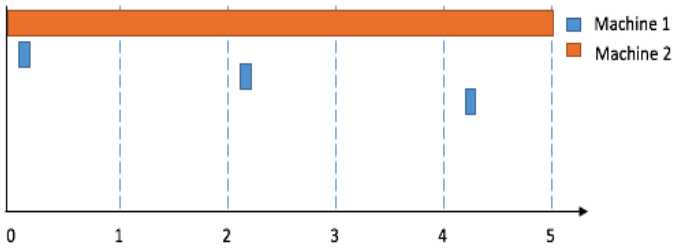

Fig 4.3.1.3 OPT for Input Sequence 2

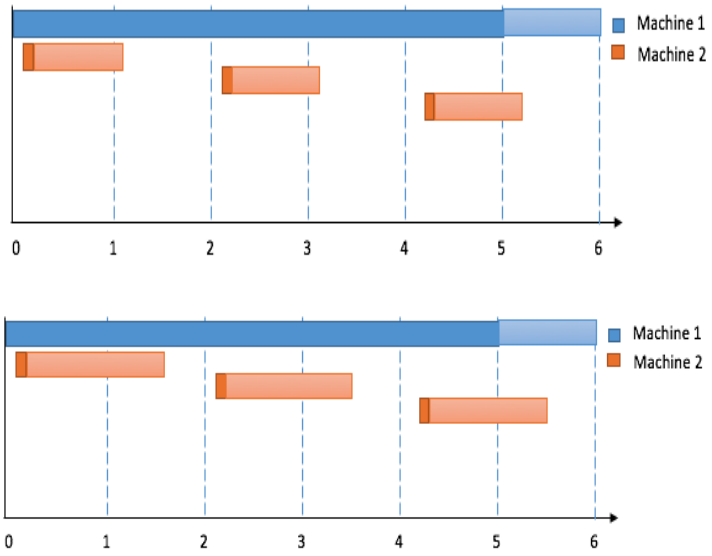

Fig 4.3.1.4 Canonical Online Algorithm for Input Sequence 2

Input Sequence 3:

s_i	e_i
4.00	10.00
4.10	4.20
5.80	5.90
7.50	7.60
9.20	9.30

Table 4.3.1.6 Input Sequence 3

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.00	18.40	11.90	1.546
1.00	1.50	20.40	11.90	1.714

Table 4.3.1.7 Cost comparison and competitive ratio based on Input Sequence 3

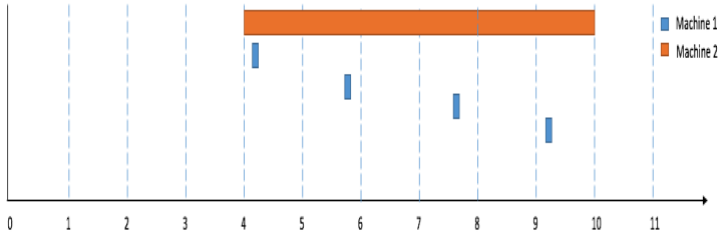

Fig 4.3.1.5 OPT for Input Sequence 3

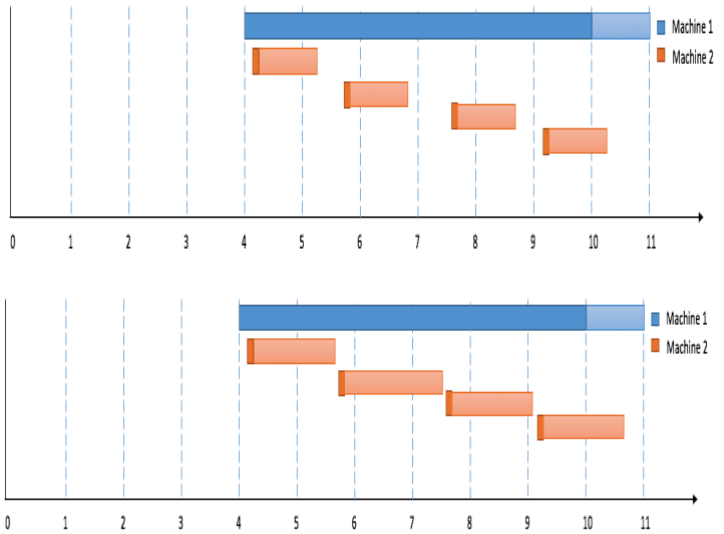

Fig 4.3.1.6 Canonical Online Algorithm for Input Sequence 3

Input Sequence 4:

s_i	e_i
0.00	6.00
0.10	0.20
1.80	1.90
3.50	3.60
5.20	5.30

Table 4.3.1.8 Input Sequence 4

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.00	18.40	11.90	1.546
1.00	1.50	20.40	11.90	1.714

Table 4.3.1.9 Cost comparison and competitive ratio based on Input Sequence 4

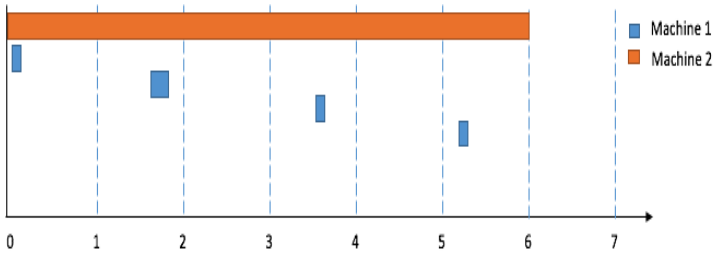

Fig 4.3.1.7 OPT for Input Sequence 4

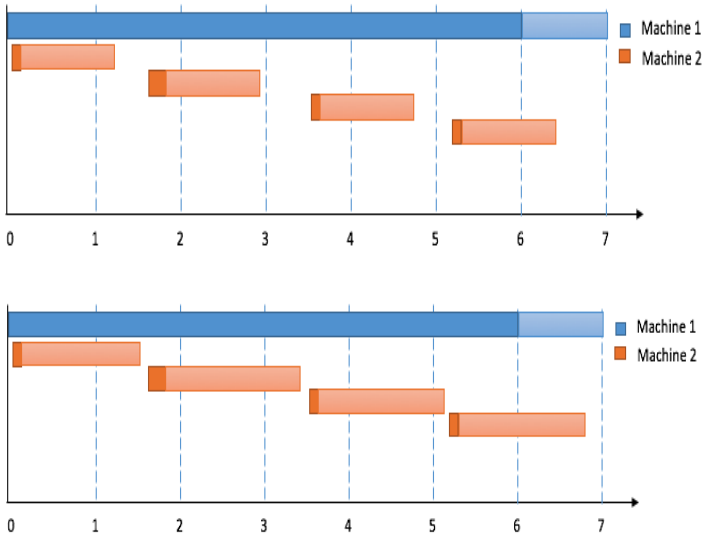

Fig 4.3.1.8 Canonical Online Algorithm for Input Sequence 4

Input Sequence 5:

s_i	e_i
0.00	5.00
0.01	0.02
1.53	1.54
3.05	3.06
4.57	4.58

Table 4.3.1.10 Input Sequence 5

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.00	17.04	10.54	1.616
1.00	1.50	19.04	10.54	1.806

Table 4.3.1.11 Cost comparison and competitive ratio based on Input Sequence 5

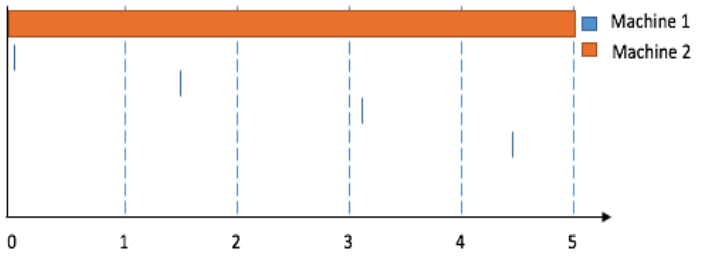

Fig 4.3.1.9 OPT for Input Sequence 5

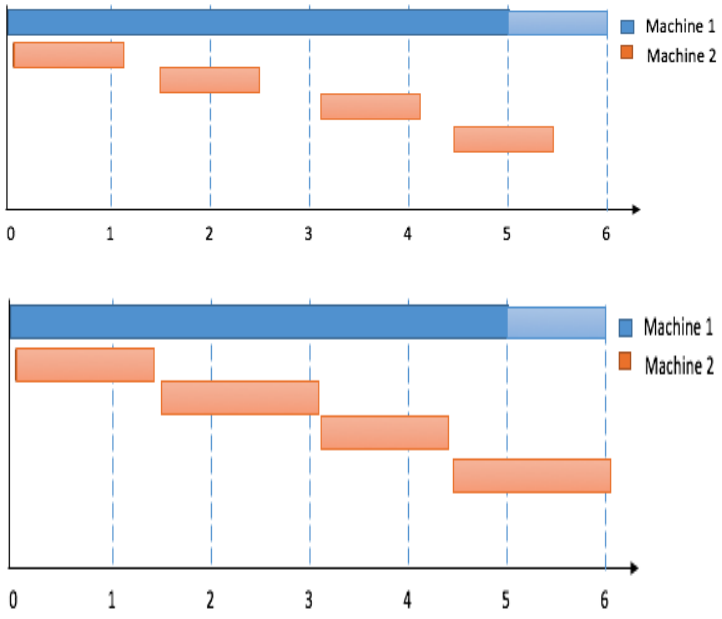

Fig 4.3.1.10 Canonical Online Algorithm for Input Sequence 5

With the parameters set as,

σ_1	σ_2	ρ_1	ρ_2
1.00	2.00	1.00	1.00

Table 4.3.1.12 Parameters set II

The Lower Bound for this parameter should be at least 1.

Input Sequence 6:

s_i	e_i
0.00	1.00
0.50	1.00
3.01	5.00
3.02	3.03

Table 4.3.1.13 Input Sequence 6

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.50	14.5	9.50	1.526
1.00	2.00	15.5	9.50	1.631

Table 4.3.1.14 Cost comparison and competitive ratio based on Input Sequence 6

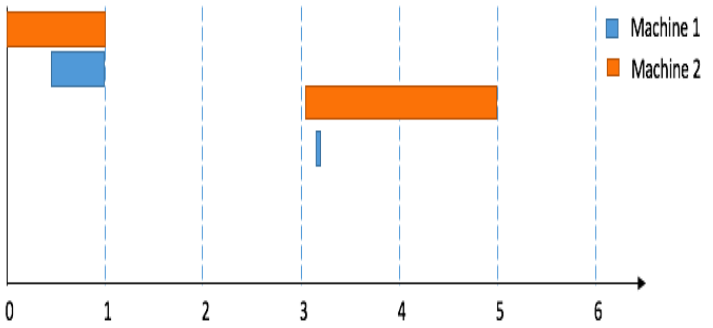

Fig 4.3.1.11 OPT for Input Sequence 6

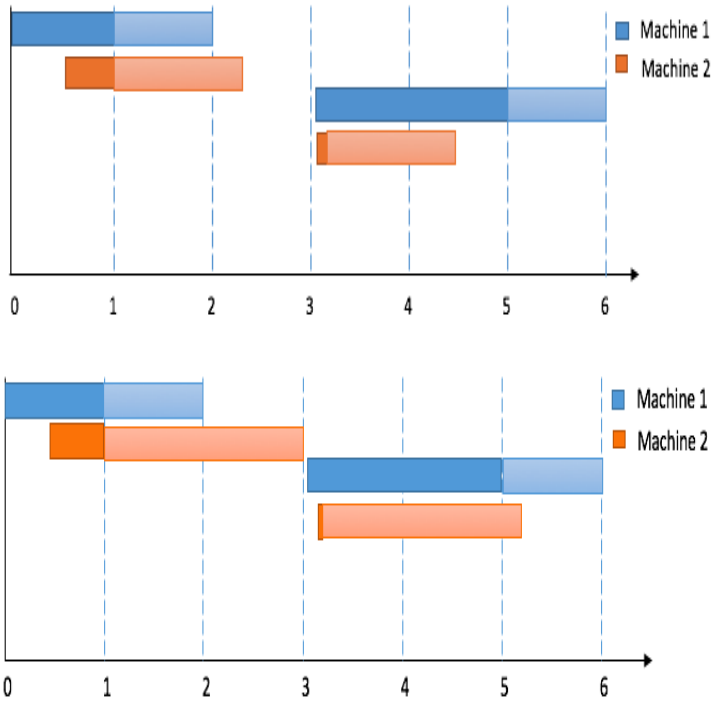

Fig 4.3.1.12 Canonical Online Algorithm for Input Sequence 6

Input Sequence 7:

s_i	e_i
0.00	1.00
0.50	1.50
3.40	3.68
5.60	5.80

Table 4.3.1.15 Input Sequence 7

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.50	11.98	7.48	1.601
1.00	2.00	12.30	7.48	1.644

Table 4.3.1.16 Cost comparison and competitive ratio based on Input Sequence 7

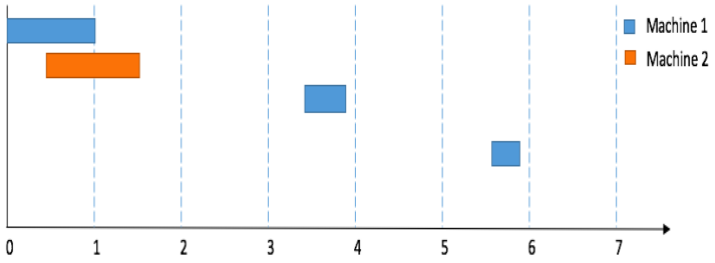

Fig 4.3.1.13 OPT for Input Sequence 7

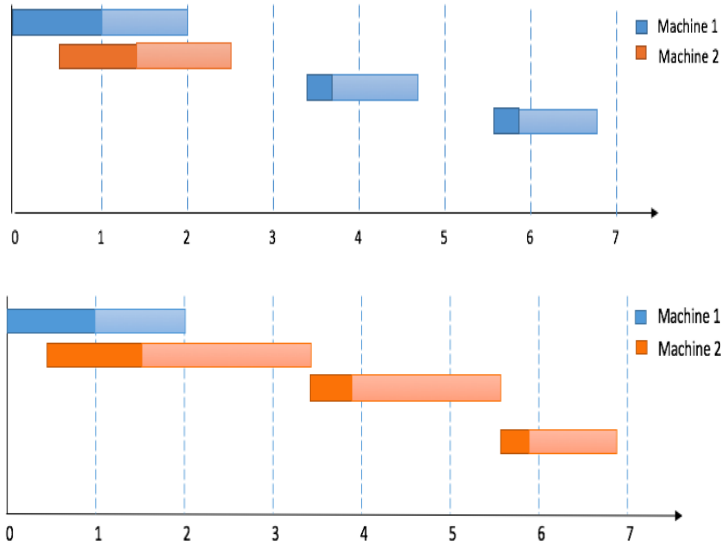

Fig 4.3.1.14 Canonical Online Algorithm for Input Sequence 7

Input Sequence 8:

s_i	e_i
0.00	5.00
0.01	0.011
2.10	2.11
4.30	5.00

Table 4.3.1.17 Input Sequence 8

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.50	18.211	10.81	1.684
1.00	2.00	19.811	10.81	1.832

Table 4.3.1.18 Cost comparison and competitive ratio based on Input Sequence 8

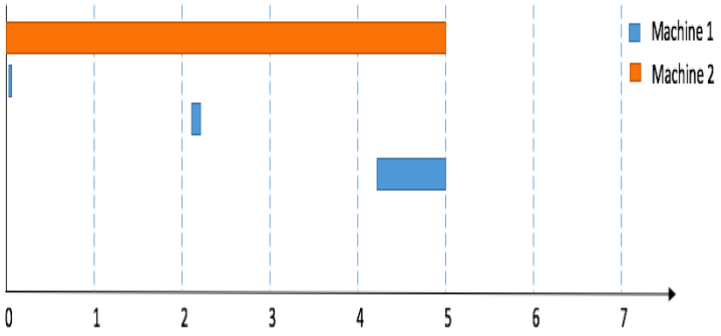

Fig 4.3.1.15 OPT for Input Sequence 8

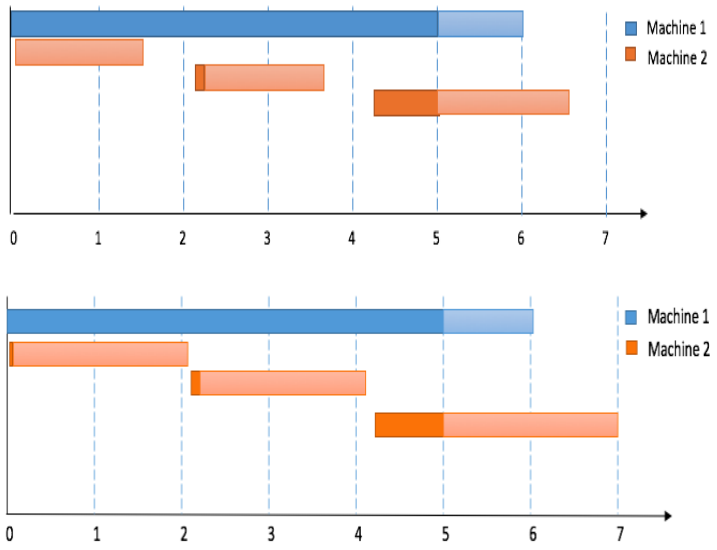

Fig 4.3.1.16 Canonical Online Algorithm for Input Sequence 8

Input Sequence 9:

s_i	e_i
0.00	5.00
0.01	0.02
2.03	2.04
4.05	4.06

Table 4.3.1.19 Input Sequence 9

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.50	17.53	10.03	1.747
1.00	2.00	19.03	10.03	1.897

Table 4.3.1.20 Cost comparison and competitive ratio based on Input Sequence 9

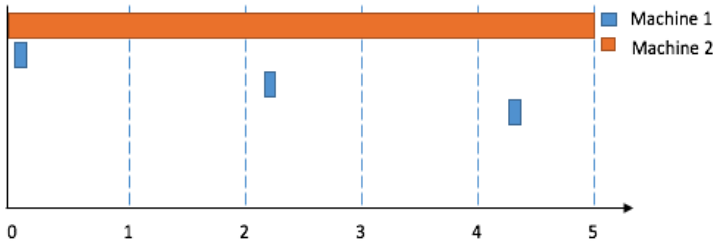

Fig 4.3.1.17 OPT for Input Sequence 9

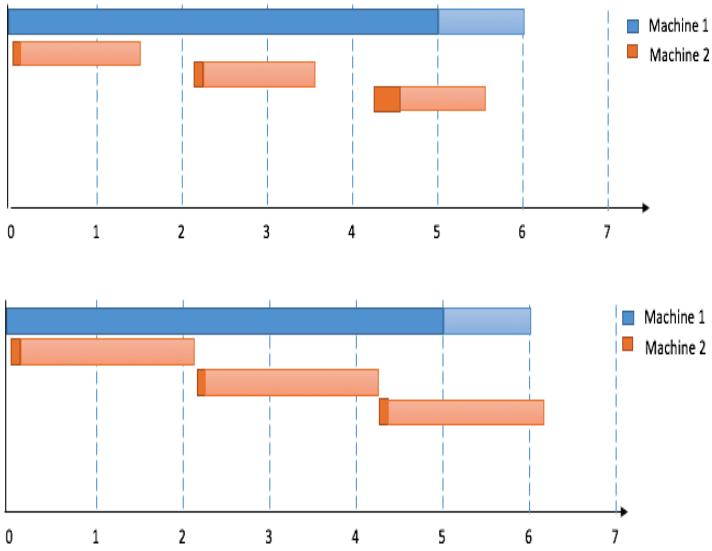

Fig 4.3.1.18 Canonical Online Algorithm for Input Sequence 9

Input Sequence 10:

s_i	e_i
0.00	5.00
0.01	0.09
2.10	2.11
4.12	4.30
6.21	10.00
6.22	6.30
8.50	9.00

Table 4.3.1.21 Input Sequence 10

Generating the idle time values using arbitrary values and σ/ρ,

w_1	w_2	Canonical Online Algorithm	OPT	CR
1.00	1.50	31.14	17.35	1.795
1.00	2.00	33.54	17.35	1.933

Table 4.3.1.22 Cost comparison and competitive ratio based on Input Sequence 10

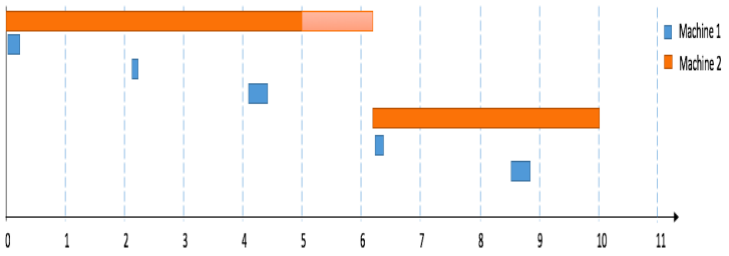

Fig 4.3.1.19 OPT for Input Sequence 10

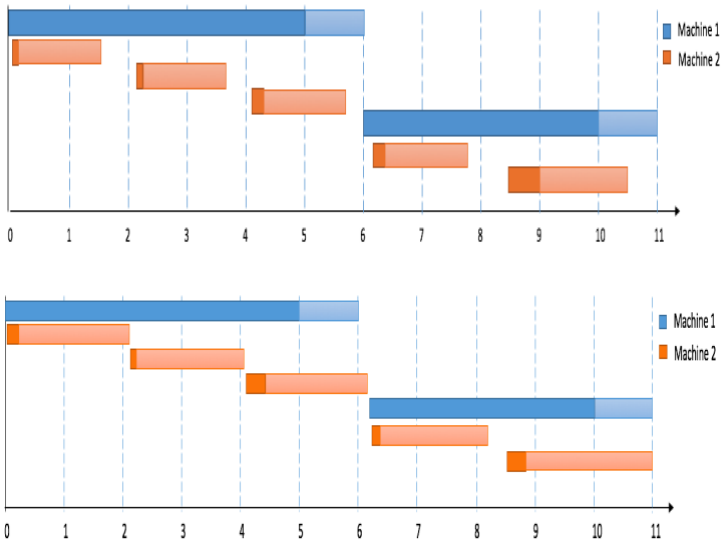

Fig 4.3.1.20 Canonical Online Algorithm for Input Sequence 10

4.3.2 Improved Online Algorithm versus Offline Algorithm

With the parameters set as,

σ_1	σ_2	ρ_1	ρ_2
1.00	1.50	1.00	1.00

Table 4.3.2.1 Parameters set I

The Lower Bound for this parameter should be at least 1.

Input Sequence 1:

s_i	e_i
0.00	1.00
0.99	2.01
3.52	5.00
6.03	9.00

Table 4.3.2.2 Input Sequence 1

Below is the cost comparison and competitive ratio based on the idle times taken, we will be generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	1.50	15.47	10.97	1.410

Table 4.3.2.3 Cost comparison and competitive ratio based on Input Sequence 1

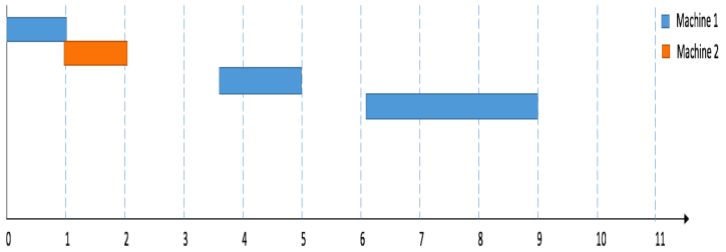

Fig 4.3.2.1 OPT for Input Sequence 1

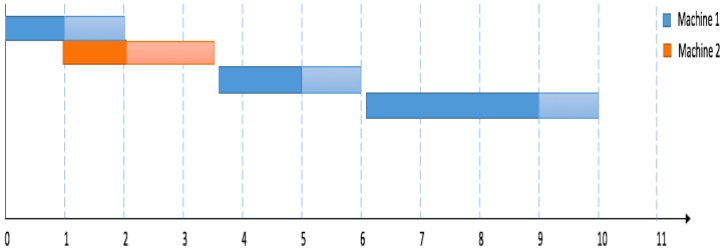

Fig 4.3.2.2 Improved Online Algorithm for Input Sequence 1

Input Sequence 2:

s_i	e_i
0.00	5.00
0.01	0.02
1.53	1.54
3.05	3.06
4.99	5.00
6.01	10.00
6.02	6.03
7.54	7.55
9.06	9.07

Table 4.3.2.4 Input Sequence 2

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	1.50	30.07	18.57	1.619

Table 4.3.2.5 Cost comparison and competitive ratio based on Input Sequence 2

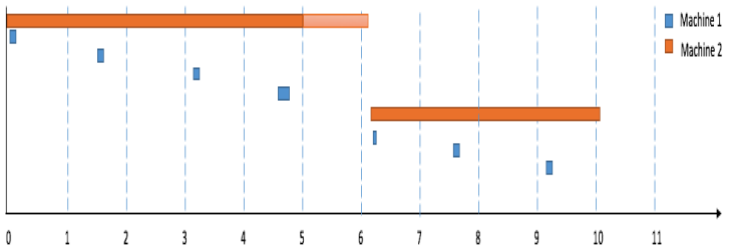

Fig 4.3.2.3 OPT for Input Sequence 2

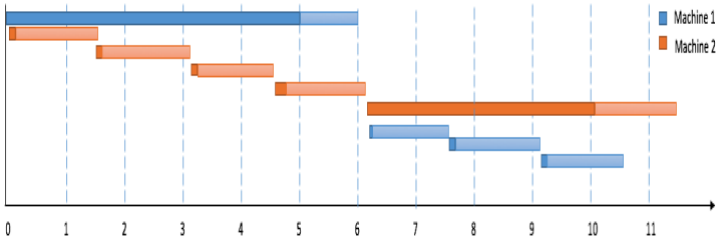

Fig 4.3.2.4 Improved Online Algorithm for Input Sequence 2

Input Sequence 3:

s_i	e_i
0.00	10.00
0.10	0.20
1.80	1.90
3.50	3.60
5.20	5.30
6.90	7.00
8.60	8.70

Table 4.3.2.6 Input Sequence 3

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	1.50	30.60	18.10	1.690

Table 4.3.2.7 Cost comparison and competitive ratio based on Input Sequence 3

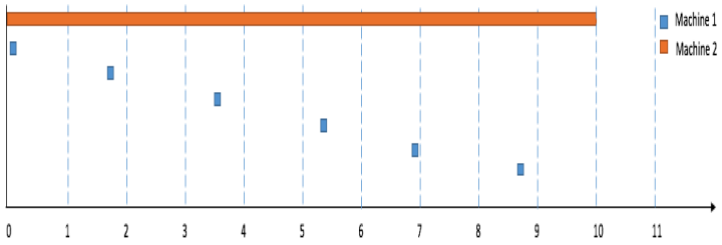

Fig 4.3.2.5 OPT for Input Sequence 3

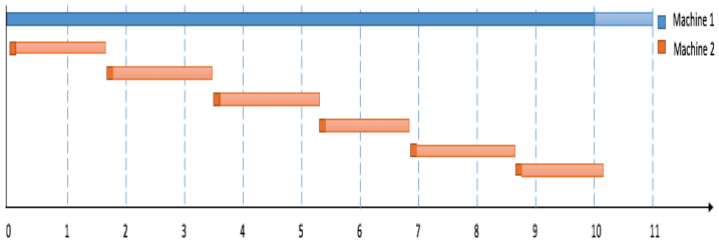

Fig 4.3.2.6 Improved Online Algorithm for Input Sequence 3

Input Sequence 4:

s_i	e_i
0.00	1.00
0.50	1.00
2.51	2.75
3.78	3.79
3.781	3.782

Table 4.3.2.8 Input Sequence 4

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Algorithm	OPT	CR
1.00	1.50	13.751	7.75	1.774

Table 4.3.2.9 Cost comparison and competitive ratio based on Input Sequence 4

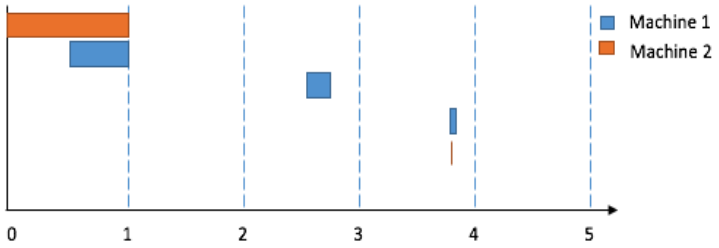

Fig 4.3.2.7 OPT for Input Sequence 4

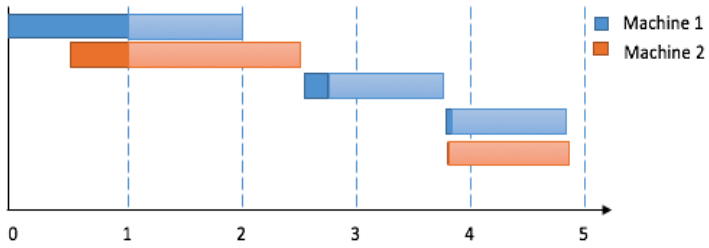

Fig 4.3.2.8 Improved Online Algorithm for Input Sequence 4

Input Sequence 5:

s_i	e_i
0.00	5.00
0.01	0.02
1.53	1.54
3.05	3.06
4.57	4.58

Table 4.3.2.10 Input Sequence 5

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	1.50	19.04	10.54	1.806

Table 4.3.2.11 Cost comparison and competitive ratio based on Input Sequence 5

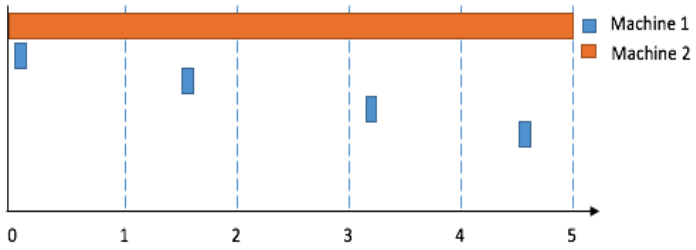

Fig 4.3.2.9 OPT for Input Sequence 5

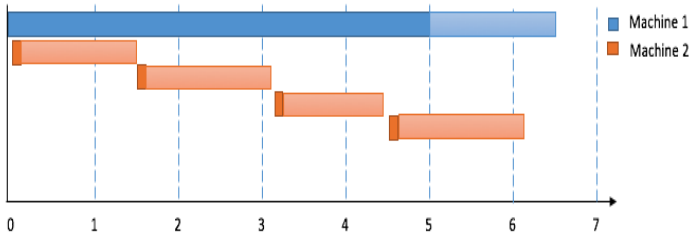

Fig 4.3.2.10 Improved Online Algorithm for Input Sequence 5

Now, setting the parameters as,

σ_1	σ_2	ρ_1	ρ_2
1.00	2.00	1.00	1.00

Table 4.3.2.12 Parameters set II

The Lower Bound for this parameter should be at least 1.

Input Sequence 6:

s_i	e_i
0.00	1.00
0.50	1.00
3.01	5.00
3.02	3.03

Table 4.3.2.13 Input Sequence 6

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	2.00	15.5	9.50	1.632

Table 4.3.2.14 Cost comparison and competitive ratio based on Input Sequence 6

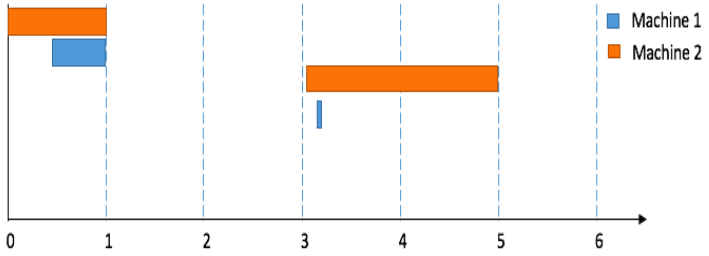

Fig 4.3.2.11 OPT for Input Sequence 6

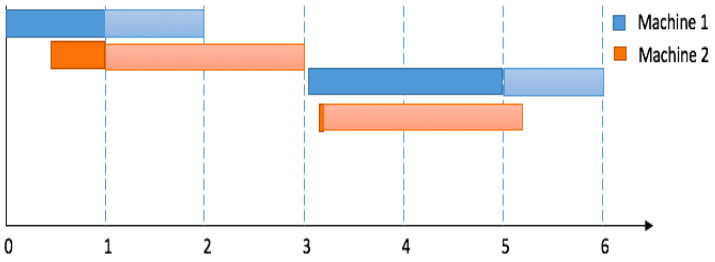

Fig 4.3.2.12 Improved Online Algorithm for Input Sequence 6

Input Sequence 7:

s_i	e_i
0.00	1.00
0.50	1.50
3.40	3.68
5.60	5.80

Table 4.3.2.15 Input Sequence 7

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	2.00	12.30	7.48	1.644

Table 4.3.2.16 Cost comparison and competitive ratio based on Input Sequence 7

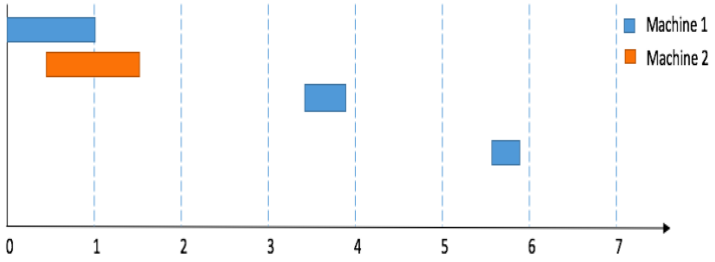

Fig 4.3.2.13 OPT for Input Sequence 7

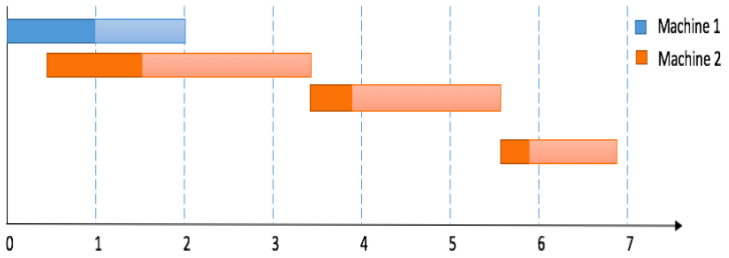

Fig 4.3.2.14 Improved Online Algorithm for Input Sequence 7

Input Sequence 8:

s_i	e_i
0.00	5.00
0.01	0.011
2.10	2.11
4.30	5.00

Table 4.3.2.17 Input Sequence 8

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	2.00	19.811	10.81	1.832

Table 4.3.2.18 Cost comparison and competitive ratio based on Input Sequence 8

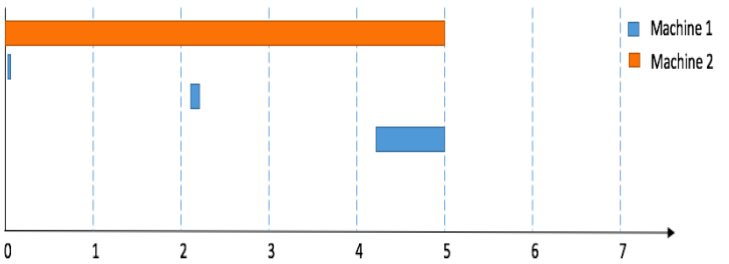

Fig 4.3.2.15 OPT for Input Sequence 8

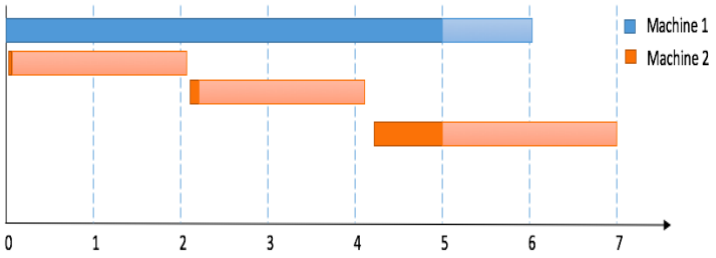

Fig 4.3.2.16 Improved Online Algorithm for Input Sequence 8

Input Sequence 9:

s_i	e_i
0.00	5.00
0.01	0.02
2.03	2.04
4.05	4.06

Table 4.3.2.19 Input Sequence 9

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	2.00	19.03	10.03	1.897

Table 4.3.2.20 Cost comparison and competitive ratio based on Input Sequence 9

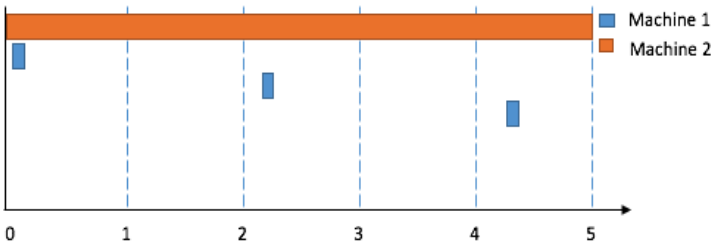

Fig 4.3.2.17 OPT for Input Sequence 9

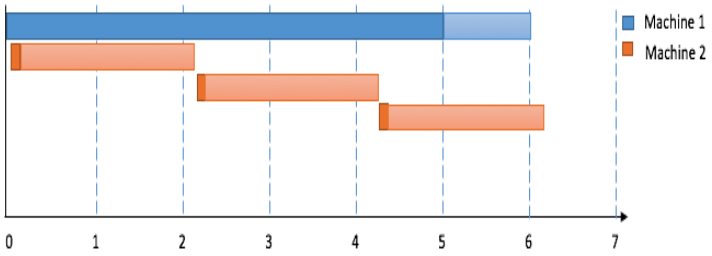

Fig 4.3.2.18 Improved Online Algorithm for Input Sequence 9

Input Sequence 10:

s_i	e_i
0.00	5.00
0.01	0.09
2.10	2.11
4.12	4.30
6.21	10.00
6.22	6.30
8.50	9.00

Table 4.3.2.21 Input Sequence 10

Generating the idle time value at σ/ρ,

w_1	w_2	Improved Online Cost	OPT	CR
1.00	2.00	33.54	17.35	1.933

Table 4.3.2.22 Cost comparison and competitive ratio based on Input Sequence 10

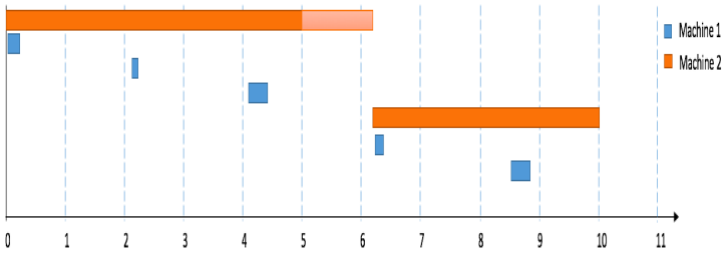

Fig 4.3.2.19 OPT for Input Sequence 10

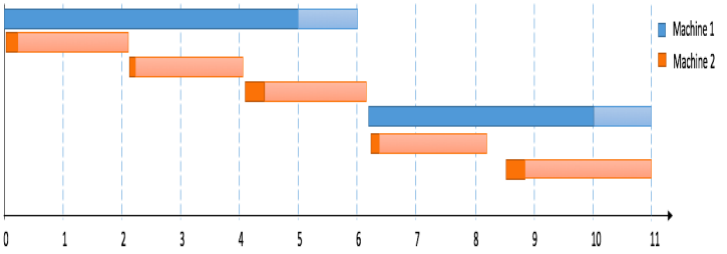

Fig 4.3.2.20 Online Algorithm for Input Sequence 10

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Summary

In this paper, we analyzed the costs incurred by two machines with dissimilar costs when scheduled with an unknown set of requests. We first presented the motivation which drove the research of the paper. We framed our analysis through the types of algorithmic models; online and offline. Given a sequence I of jobs, we define, say, $cost_{opt}(I)$ to be the minimum cost of any schedule for I. A schedule minimizing $cost_{opt}(I)$ can easily be constructed in linear time since knowing the finish time of a job scheduled on a machine and the start time of the next job on this machine one can then decide whether the cost is lower to keep it on in between or to turn it off and then restart. Whereas, in real-life situations, the request sequence of jobs is not known beforehand. n online algorithm S must decide at time s_i, which machine to assign to job j_i, if both machines are available, and must also decide for a machine after it has executed a job how long to let it run idle before turning it off.

We propose a naïve approach to the problem definition and then improve the existing algorithm for a better result. We did extensive simulations and compared it with the $cost_{opt}(I)$, and checked for the simple lower bounds for each.

5.2 Future Work

Our future work will consist of refining and combing our ideas from this thesis. In no way have we solved the problem in theoretical terms, and we have not given any general upper bounds. It would be desirable to devise an online algorithm with provable competitive ratio. However, in this thesis we have seen that there are strategies that work well under various simulations. The work done on the dissimilar cost problem is quite sparse, since not much work has been done on that subject. We will continue to develop new strategies to attempt to come up with upper or lower bounds. Another line of investigation would be to devise upper or lower bounds for more than two machines with dissimilar costs. Also, tapering down strategy techniques have been considered (see [3]) and we suggest applying such techniques in the case of multiple machines. Such problem is an applicable problem in many areas, even beyond the area of information technology, and we seek to gain a great deal of understanding of this problem and how it can be used in many areas of study.

REFERENCE:

[1]. Sandy Irani and Anna R. Karlin. 1996. Online computation. In Approximation algorithms for NP-hard problems, Dorit S. Hochbaum (Ed.). PWS Publishing Co., Boston, MA, USA 521-564.

[2]. Karp, Richard M. (1992). "On-line algorithms versus off-line algorithms: How much is it worth to know the future?". IFIP Congress (1). **12**: 416–429. Retrieved 17 August 2015.

[3]. J. Andro-Vasko, W. Bein, D. Nyknahad, and H. Ito. Evaluation of online power-down algorithms. In 2015 12th International Conference on Information Technology - New Generations, pages 473–478, April 2015

[4]. https://en.m.wikipedia.org/wiki/

[5]. Sandy Irani and Gaurav Singh. An overview of the competitive and adversarial approaches to designing dynamic power management strategies. IEEE Transactions Very Large Scale Integration. 13(12). December 2005.

[6]. Susanne Albers. Energy-efficient algorithms. Communications Of The ACM, 53:86–96, 2010.

[7]. Mumey, Brendan & Tang, Jian & Hashimoto, Saiichi. (2012). Enabling green networking with a power down approach. 2867-2871. 10.1109/ICC.2012.6364006.

[8]. Y. Wati and C. Koo, "The Green IT Practices of Nokia, Samsung, Sony, and Sony Ericsson: Content Analysis Approach," 2010 43rd Hawaii International

Conference on System Sciences, Honolulu, HI, 2010, pp. 1-10.doi: 10.1109/HICSS.2010.480

[9]. Bonichon, Nicolas & Bose, Prosenjit & De Carufel, Jean-Lou & Perkovic, Ljubomir & Van Renssen, André. (2015). Upper and Lower Bounds for Competitive Online Routing on Delaunay Triangulations. 10.1007/978-3-662-48350-3_18.

[10]. Allan Borodin and Ran El-Yaniv. 1998. Online Computation and Competitive Analysis. Cambridge University Press, New York, NY, USA

[11]. S. Phillips and J. Westbrook. chapter 10 of Algorithms and Theory of Computation Handbook, chapter On-line algorithms: "Competitive analysis and beyond". CRC Press, Boca Raton, 1999.

[12]. Pathak, Govind. "Analysis of power-down systems with five states." University of Nevada, Las Vegas, 2016.

CURRICULUM VITAE

Graduate College

University of Nevada, Las Vegas

Madhurupa Moitra

madhurupamoitra@gmail.com

Degrees:

Bachelor of Technology in Computer Science, 2016

Budge Budge Institute of Technology, Kolkata, India

West Bengal University of Technology, Kolkata, India.

Master of Science in Computer Science, 2016

University of Nevada Las Vegas

Thesis Title: **Scheduling Two Machines with Dissimilar Costs.**

Thesis Examination Committee:

Chair Person, Dr. Wolfgang Bein, Ph.D.

Co-Chair Person, Dr. Lawrence L. Larmore, Ph.D.

Committee Member, Dr. Laxmi Gewali, Ph.D.

Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D.