May 2019

Batching Problems with Constraints

Shradha Kapoor
shradha_kapoor@outlook.com

BATCHING PROBLEMS WITH CONSTRAINTS

By

Shradha Kapoor

Bachelor of Engineering in Information Science and Engineering

Visvesvaraya Technological University, India

May 2013

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

May 2019

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 12, 2019

This thesis prepared by

Shradha Kapoor

entitled

Batching Problems with Constraints

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

Lawrence Larmore, Ph.D.
Examination Committee Member

Laxmi Gewali, Ph.D.
Examination Committee Member

Zhiyong Wang, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Dean

ABSTRACT

Batching problems with constraints

by

Shradha Kapoor

Dr. Wolfgang Bein, Examination Committee Chair

Professor, Department of Computer Science

University of Nevada, Las Vegas

There is an increasing demand for a phenomenon that can manifest benefits gained from grouping similar jobs together and then scheduling these groups efficiently. Batching is the decision of whether or not to put the jobs into same group based on certain criteria. Batching plays a major role in job scheduling in Information Technology, traffic controlling systems, and goods-flow management. A list batching problem refers to batching a list of jobs in the same order or priority as given in the problem.

In this thesis we consider a one-machine list batching problem under weighted average completion. Given sequence of jobs are scheduled on single machine into distinct batches. Constraint is to batch these jobs into a fixed but arbitrary number '*k*' of batches. Each batch can have any number of jobs (within the given list) grouped without changing the order of jobs. We call it a *k-Batch* problem. This is offline form of the batching problems, and is solved by reducing to a shortest path problem. We give an improved and faster version of the algorithm to solve *k-Batch* problem in $O(n^2)$ time.

ACKNOWLEDGEMENTS

I would like to express my gratitude towards Dr. Wolfgang Bein for his continuous guidance throughout my graduate studies at University of Nevada, Las Vegas. I thank you for being my thesis committee chair. I derived confidence from the way you motivated me to move forward in the right direction of research. This immensely helped to broaden my work. Without your advice this thesis would not have had such a rich content.

I am indebted to Dr. Lawrence Larmore, Dr. Laxmi Gewali, and Dr. Zhiyong Wang for being my thesis committee members. I am highly grateful for teaching me the complex algorithmic concepts in a simplified way. With your support when I needed it, this thesis has taken a more meaningful shape. For this and for being generous with the availability during the hours of my need.

Special thanks to the faculty at the Department of Computer Science, University of Nevada Las Vegas for providing me advanced knowledge essential for a Master's degree student along with the financial support.

I would like to extend sincere appreciation towards my husband, my parents, and my brother for being the pillars of strength through thick and thin and always encouraging me to dream big and then strive for it. Without their perpetual help I could never have the capacity to reach to the spot I'm standing today in my life. Last however not the least; I thank my friend for their help and support towards completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Batching and dynamic programming are applied in wide variety of areas like traffic controlling systems, goods-flow management, job scheduling, and decision management. The solutions to batching problems occurring due to modern technologies empower the capabilities to manage operations in Information Technology efficiently. The topic analysis of batching is covered in expansive amount of literary work. Batching issues related to goods management are addressed by Kuik, Salomon, and Wassenhove [7]. These issues are divided into following levels of decision: (a) choosing/ designing the process, (b) planning the activity, and (c) controlling the activity. Decision to group or to not group similar items sequentially is referred as batching by Potts and Wassenhove [9]. Moreover, the decision of how and when to break a big group of identical items into smaller sub-groups is called batch-sizing. The paper gives description of a general model in a complex environment such that it takes into account all trade-offs during batching, batch-sizing and scheduling processes.

There is increasing demand for a phenomenon that can manifest benefits gained from grouping similar jobs together and then scheduling these groups efficiently. Consider there is one machine, where a given set of jobs have to be batched before being scheduled on the machine. All the jobs belonging to same batch are assumed to have same processing times because they are processed together in the batch. Whereas the consecutive batches can have a constant setup delay time between each other. Coffman, Yannakakis, Magazine, Santos [2] gave an algorithm for any random but fixed sequence of jobs; it batches these jobs such that the total processing time is minimal.

The scheduling analysis of a problem differs based on the assumptions of the basic model. For example, imagine the tasks belong to same group due to their similarity in some way, such as their storage requirements or their manufacturing tools. Thus, no setup time is required for the job belonging to a group, which is followed by another job from the same group. However, a 'setup time' for the 'group' will definitely be required when a given job is followed by a job belonging to a different group. The research work by Webster, Scott, and Kenneth R. Baker [8] calls this a group scheduling model. Their paper describes another variant of grouping model based on the ability of a machine to process multiple jobs simultaneously. Consider that items must be placed in a washer, for a washing operation. The washer has definite capacity, so fixed number of items can be batched together for processing at once. This model gets its name as batch processing model. Brucker and Hurink[10] solved one of the applications of batch scheduling problem, i.e. chemical batching using local search. For example, say there exist a set of items which are to be processed together by given a set of facilities. Certain given amount of product in each order is to be manufactured within the set deadline. There is a sequence of tasks involved in each production whereby each task must be performed by one of the allowed facilities for this process. The maximum and minimum size of a batch is given and manufacturing is performed in batch mode depending on the facility chosen. To solve scheduling part of this problem, the paper presented a method to compute optimal amount of batches required to fulfil the product-requests. Even though the offline version of batching has been studied widely but several applications like Transmission Control Protocol acknowledgement demand online solutions. The study by W. W. Bein et al. [11] terms an online batching problem. In this problem there exists different jobs which are lined up as an online stream, and before a new job is seen by the scheduler, each of these jobs need to be appropriately scheduled. There are two ways to schedule a job: (a) it can be made part of the current batch (b) while a new batch is scheduled to be executed, this job can be made the first job which will be executed at that time. This type of algorithms, which abide the conditions mentioned above can be termed as online algorithms to solve batching problems. An algorithm batches a job if it performs the action described in (b). Let us consider the following application of online batching problem: given tasks can be executed either on a single processor or on multiple processors. These tasks are split into batches such that their requirement of a resource is similar. We need to setup the resources of each batch before its processing starts. We acknowledge the success of processing a batch only after it terminates. As soon as the acknowledgement is sent, we may set the status of the job as completed, though it is not necessary that the job has stopped being executed. Once job processing for all the jobs is done, status of the batch can me marked as completed. In case of batching problem run on a single processor, at one point in time one job is executed and a batch is completed when all its jobs are completed. A multiprocessor batching problem is such that each job can be run independently on separate processor and we have multiple processors available at a time. In this case the runtime of a batch is the maximum processing time of any job in the batch.

CHAPTER 2

LIST BATCHING

2.1 A DESCRIPTION OF THE BATCHING PROBLEM

Partitioning and sequencing problems when combined shape into batching problems. We take into account a batching problem as

1. Set of jobs $J = \{J_i(processing\ times\ p_i, weights\ w_i)\}$ where p_i and w_i are positive integers with $i = 1, \ldots, n$.
2. There exists a single machine, which can be scheduled with a given a set of Jobs J, such that they are partitioned into batches. The completion time of every job in same batch is added to from the overall completion time of the batch.
3. A delay or setup time $s = 1$ is required every time a job from new batch is scheduled for processing.

Our aim is to find a schedule such that the overall execution time of a given set of jobs is minimized, abiding by the conditions and constraints on it. The aim of a batching algorithm is to efficiently order jobs in a batch so they are processed efficiently.

Batching problem can have two different types. First, when the jobs are executed sequentially. Second, when the jobs are executed in parallel. Note that we will not study the latter batching problems here.

Let us represent aforementioned first type of batching problem's set of jobs in $\{J_1(p_1, w_1), J_2(p_2, w_2), \ldots, J_n(p_n, w_n)\}$ manner for convenience. An example of 5 jobs can be represented as $J_1(5,2)$, $J_2(1,1)$, $J_3(4,1)$, $J_4(1,2)$, $J_5(2,3)$, see Figure 1.

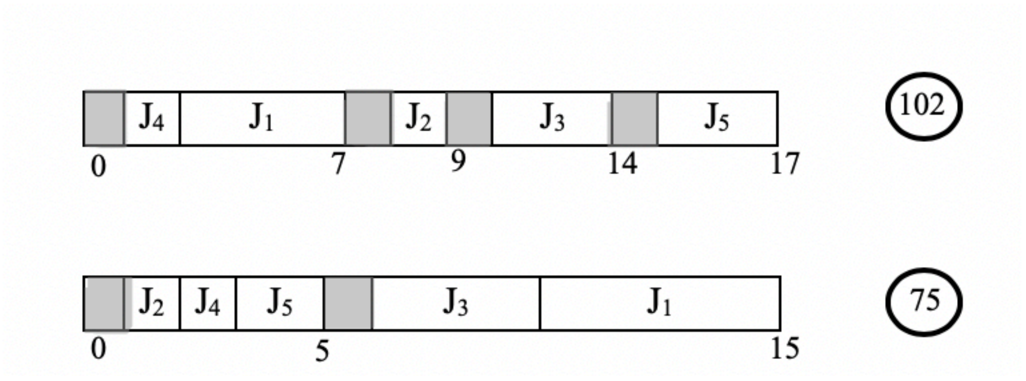


Figure 1: A batching example

Note that in this example, the order of jobs is not given. Therefore, there is a degree of freedom not only in the order but also – once an order is decided – in where the jobs are split into batches.

Think of a set of 3(= n) jobs as $J_1(p_1, w_1), J_2(p_2, w_2), J_3(p_3, w_3)$. Figure 2 shows that these jobs can be batched as *6* different sequences. Furthermore, each batch of jobs can be scheduled in *4* distinct ways. Thus, the total number of possible schedules is *24*.

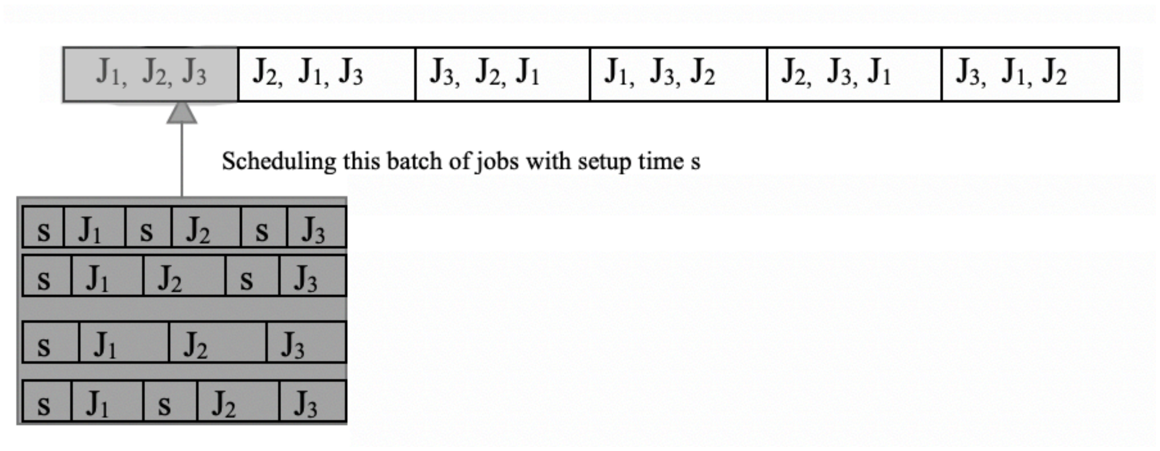


Figure 2: All possible batches of *3* jobs and further scheduling each batch.

Lemma 1 (from [1]) If the sequence of jobs is fixed, then the batch sizing for any batching problem can be performed in O(n) time. Therefore, many batching problems can be solved in polynomial time.

Proof With respect to the general objective function $T_F = \sum w_i C_i$, we can find an optimal sequence of batches for a fixed job sequence $J_1, J_2, \ldots, J_n$.

A batching solution B looks like

$$B = s\, J_{i1} \ldots J_{i2-1}\, s\, J_{i2} \ldots J_{i3-1}\, s\, J_{i3} \ldots J_{ik-1}\, s\, J_{ik} \ldots J_n$$

where solution B has k number of batches,

There is j^{th} batch, where the first job has index i_j ,

$$1 = i_1 < i_2 < i_3 < \ldots < i_k \leq n$$

The processing time of j^{th} batch is given by

$$P_j = s + \sum_{v=i_j}^{i_{j+1}-1} p_v$$

Objective function for B is computed as

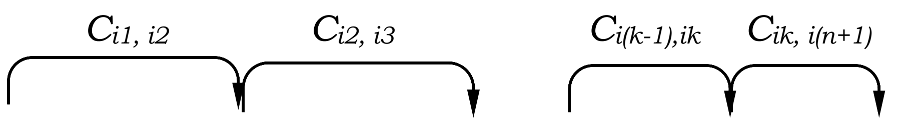
$$T_F(B) = \sum_{i=1}^{n} W_i C_i$$

$$= \sum_{j=1}^{k} \left(\sum_{v=i}^{n} w_v \right) \mathrm{P_j}$$

$$= \sum_{j=1}^{k} \left(\sum_{v=ij}^{n} w_v \right) \left(s + \sum_{v=i_j}^{i_{j+1}-1} p_v \right)$$

We compute a constant k and a sequence of indices $1 = i_1 < i_2 < i_3 < \ldots < i_k \le n$ such that the above $T_F(B)$ is minimized and thereby solves the batch sizing problem.

This issue can be diminished to a shortest path problem. All solutions for B can be represented:



$$B = s\ J_{i1} \ldots J_{i2-1}\ s\ J_{i2} \ldots J_{i3-1}\ s\ J_{i3} \ldots J_{ik-1}\ s\ J_{ik} \ldots J_n\ J_{n+1}$$

Here J_{n+1} is a dummy job. Edge (J_i, J_j) has length $C_{i,j,}$ which has costs generated as $J_i, J_{i+1}, \ldots, J_{j-1}$.

$$C_{i,j} = \left(\sum_{v=i}^{n} w_V \right) \left(s + \sum_{v=i}^{j-1} p_V \right)$$

Let i < j < k then

$$C_{i,k} - C_{i,j} = \left(\sum_{v=i}^{n} w_V \right) \left(\sum_{v=j}^{k-1} p_V \right)$$

We note that $(\sum_{v=i}^{n} w_V)$ is a monotone decreasing function and $\left(\sum_{v=j}^{k-1} p_V\right)$ is a positive integral value for any $1 \le j < k \le n+1$. This dialog focuses to the issue of finding a most brief way from vertex 1 to vertex n+1 in a system N = (V, E, C) with the highlights as given below:

1. The set of vertices, $V = \{1, 2, \ldots, n+1\}$.
2. An edge $(i, j) \epsilon E$ if and only if $i < j$.
3. The edge length $C = C_{ij}$ should satisfy

$$C_{i,k} - C_{i,j} = \left(\sum_{v=i}^{n} w_v\right)\left(\sum_{v=j}^{k-1} p_v\right) \qquad \textit{for all } i < j < k$$

   here $\left(\sum_{v=i}^{n} w_v\right)$ is a monotonic decreasing function and $\left(\sum_{v=j}^{k-1} p_v\right) > 0$ for all $j < k$.

Coffman et al. [2] explained an algorithm using a dynamic programming approach to solve such problems in polynomial time. The algorithm works only if each of the values $\left(\sum_{v=i}^{n} w_v\right)$ *and* $\left(\sum_{v=j}^{k-1} p_v\right) > 0, j < k$ can be calculated in linear time as a preprocessing step.

2.2 LIST BATCHING

Jobs $J_i = J_1, J_2, \ldots, J_n$ define the list version of the batching problem with processing time $p_i = p_1, p_2, \ldots, p_n$ and weight $w_i = w_1, w_2, \ldots, w_n$ respectively. We must process the jobs in same sequence as given in the list. These jobs are scheduled on a single machine in distinct batches. Every batch uses a setup time of $s = 1$. The completion time C_i is the completion time of job J_i in a given schedule.

Naturally, the jobs are considered according to the sequence of priorities $\frac{W_i}{P_i}$ such as $\frac{W_1}{P_1} \geq \frac{W_2}{P_2} \ldots \geq \frac{W_n}{P_n}$. Figure 3 shows four different ways to schedule 5-jobs $J_i(p_i, w_i)$ sequence as $J_1(2, 3), J_2(1, 2), J_3(1, 1), J_4(4, 3), J_5(3, 1)$. These jobs are ordered based on decreasing priority as $3/2 \geq 2/1 \geq 1/1 \geq 3/4 \geq 1/3$.

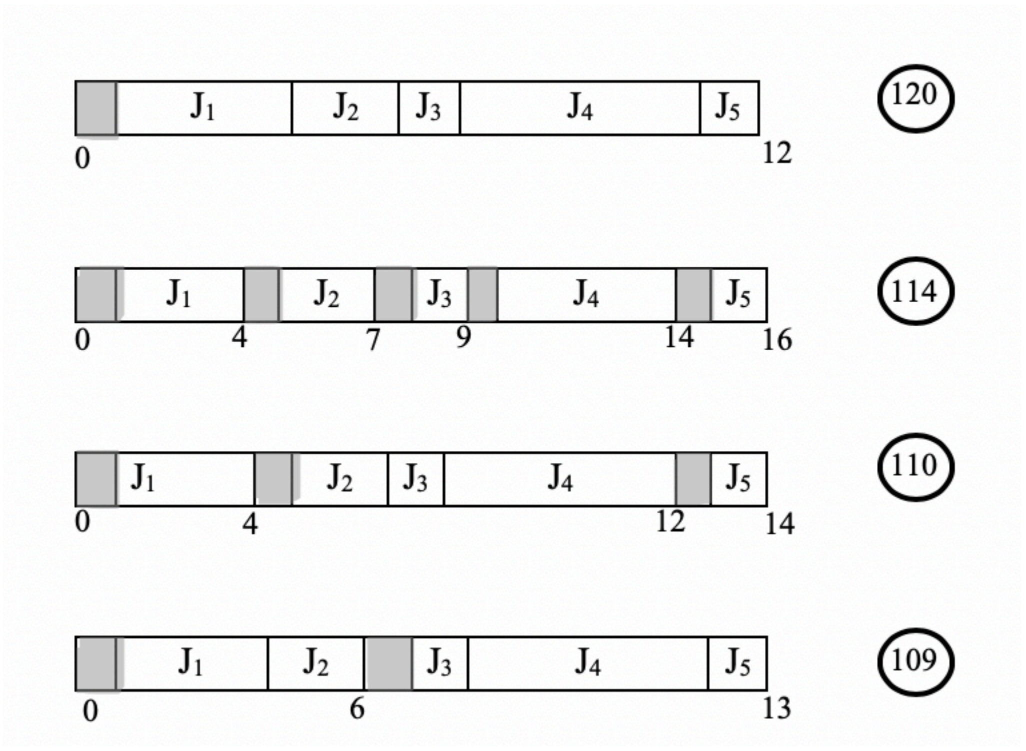


Figure 3: An example of list batching problem

2.3 LIST BATCHING PROBLEM REDUCTION TO PATH PROBLEM

The list batching problem can be reduced to a shortest path problem in the following manner:

Consider the jobs J_i from $i = 1, \dots, n$ in this order. A weighted directed acyclic graph G is constructed with one node for each job (node J_i where $i = 1, \dots, n$). Add a dummy node 0 in the beginning of the list. We can form an edge (i, j) if and only if $i < j$, see Figure 4. The edge cost $C_{i,j}$ for $i < j$ is defined as

$$C_{i,j} = \left(\sum_{l=i+1}^{n} w_l\right)\left(s + \sum_{l=1}^{j} p_l\right)$$

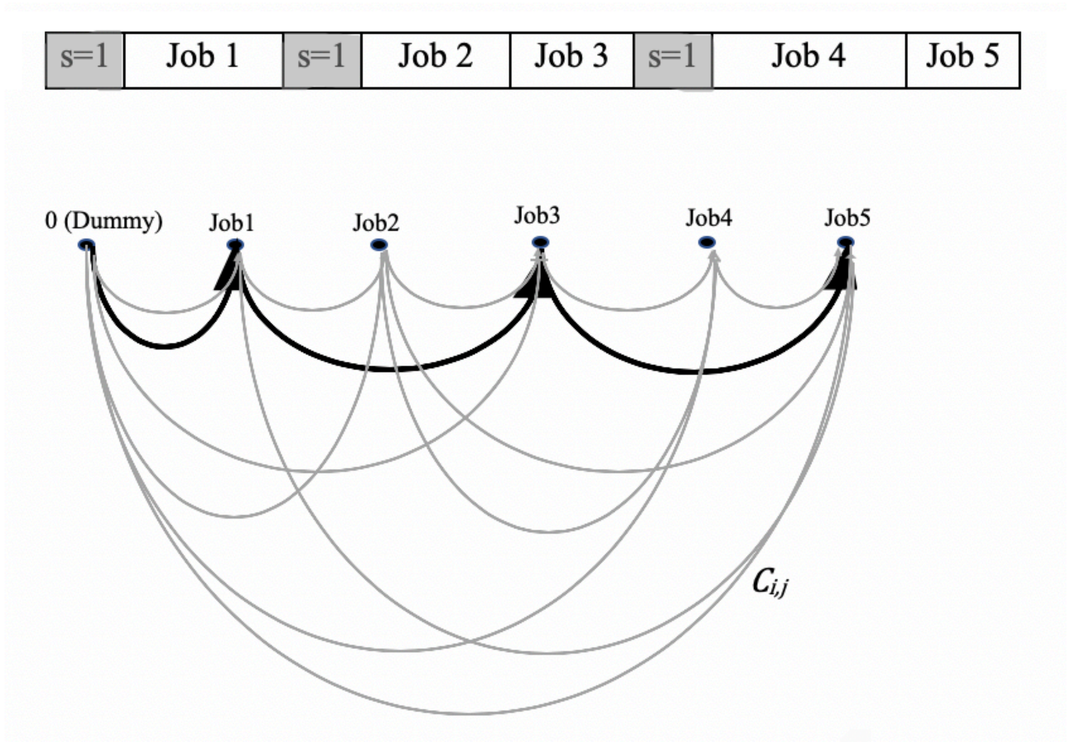


Figure 4: Reduction of the list batching problem to a shortest path problem

We briefly note:

Lemma 2 (from [3]) The matrix $C = C_{i,j}$ (as described in 2.1) is Monge (refer to 4 for definition) for all $p_i, w_i \geq 0$. Additionally, this matrix C can be computed in constant time given the linear time preprocessing.

Proof Let $\mathrm{W}_i = \sum_{v=1}^{i} w_v$ and $\mathrm{P}_i = \sum_{v=1}^{i} p_v$ be the partial sum of the p_i and w_i values. So, we have

$$C[i,j] = C_{i,j} = (W_n - W_i)(s + P_j - P_i)$$

For $i < i` and j < j`$

$$C[i,j] + C[i`,j`] - C[i`,j] - C[i,j`] = (P_{j`} - P_{j`})(W_{i`} - W_i) \geq 0$$

Furthermore, observe that these values of matrix can be computed in constant time after linear time preprocessing by setting up arrays of partial sums for W_i and P_i in linear time.

Going back to description of the reduction, it is clearly seen (see Albers and Brucker (1993) [1] for details) that the $\sum C_i w_i$ value of the schedule is given by the cost of path $< 0, i_1, i_2, \ldots, i_k, n >$ which batches at each job $i_1, i_2, \ldots, i_k$. Conversely, any batching with cost A corresponds to a path in graph G with path length A.

Following dynamic algorithm can be used to compute the shortest path in time $O(n^2)$:

Let

$$E[m] = cost\ of\ the\ shortest\ path\ from\ node - to\ node\ m$$

then

$$E[m] = \min_{1\leq k\leq m}\{E[k] + C_{k,m}\}\ with\ E[0] = 0$$

As a result of above equation, we get a table shown in Figure 5, where elements can be generated row by row.

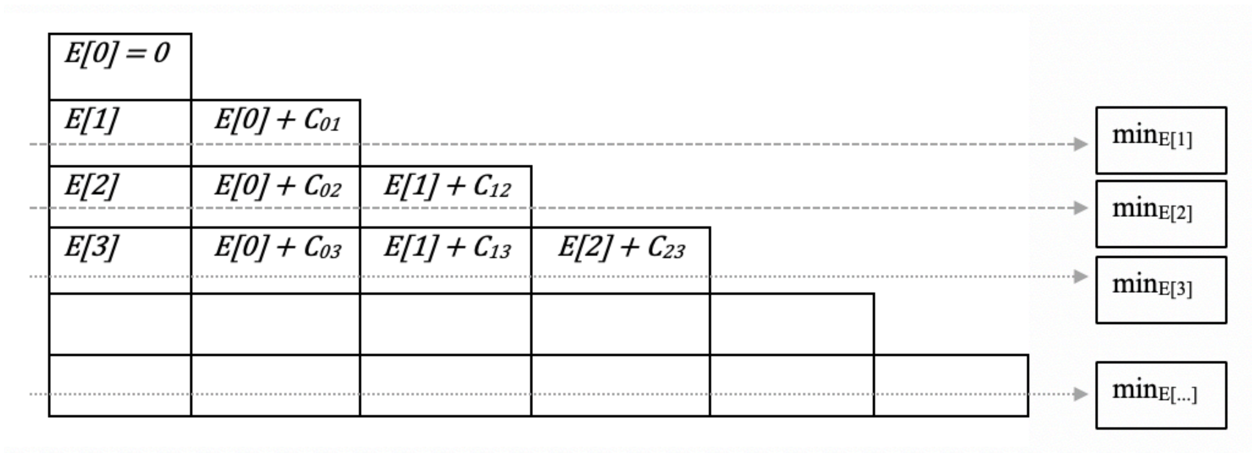


Figure 5: Dynamic programming tableau for list batching problem.

We see that this dynamic program can calculate the minima of each row of n×n matrix E, such that

$$E[m] = \begin{cases} E[k] + C_{k,m} & if\ k < m \\ \infty & otherwise \end{cases}$$

with $k = 0, \dots, m-1$ *and* $m = 1, \dots, n$.

All row minima values obtained from the shortest path matrix E, are used to form the final shortest path graph.

Lemma 3 (from [3]) The matrix $E = E_{m,k}$ is Monge.

Proof Monge property of matrix E is preserved under addition and finding the minimum.

2.4 ILLUSTRATION OF THE LIST BATCHING PROBLEM REDUCED TO SHORTEST PATH PROBLEM

We are given a list of 9-jobs $J_i(p_i, w_i)$ in decreasing order as $J_1\,(4, 1) > J_2\,(4, 2) > J_3\,(2, 1) > J_4\,(3, 2) > J_5\,(4, 3) > J_6\,(2, 2) > J_7\,(2, 3) > J_8\,(2, 4) > J_9\,(1, 3)$.

The setup time is $s = 1.$

Partial sum of the processing times $P_i = \sum_{v=1}^{i} p_v$ for all $i = 1, 2, \ldots, 9$ and $P_0 = 0$ is

$P_1 = 1$	$P_2 = 3$	$P_3 = 4$	$P_4 = 6$	$P_5 = 9$	$P_6 = 11$	$P_7 = 14$	$P_8 = 18$	$P_9 = 21$

Partial sum of the weights $W_i = \sum_{v=1}^{i} w_v$, for all $i = 1, 2, \ldots, 9$ and $W_0 = 0$ is

$W_1 = 4$	$W_2 = 8$	$W_3 = 10$	$W_4 = 13$	$W_5 = 17$	$W_6 = 19$	$W_7 = 21$	$W_8 = 23$	$W_9 = 24$

Completion times of the jobs = Cost matrix = $C_{i,j}$

$$= \left(\sum_{m=i+1}^{n} w_m\right)\left(s + \sum_{m=1}^{j} p_m\right)$$

$$= (W_n - W_j)(w + P_j - P_i)$$

Cost of edge from i^{th} node to j^{th} node is given by $C_{i,j}$, where $i < j$, $i = 0, 2, \ldots, 8$, and $j = 1, 2, \ldots, 9$ as

C_{01}	48																
C_{02}	96	C_{12}	60														
C_{03}	120	C_{13}	80	C_{23}	32												
C_{04}	168	C_{14}	120	C_{24}	64	C_{34}	42										
C_{05}	240	C_{15}	180	C_{25}	112	C_{35}	84	C_{45}	44								
C_{06}	288	C_{16}	220	C_{26}	144	C_{36}	112	C_{46}	66	C_{56}	21						
C_{07}	360	C_{17}	280	C_{27}	192	C_{37}	154	C_{47}	99	C_{57}	42	C_{67}	20				
C_{08}	456	C_{18}	360	C_{28}	256	C_{38}	210	C_{48}	143	C_{58}	63	C_{68}	40	C_{78}	15		
C_{09}	528	C_{19}	420	C_{29}	304	C_{39}	252	C_{49}	176	C_{59}	91	C_{69}	55	C_{79}	24	C_{89}	4

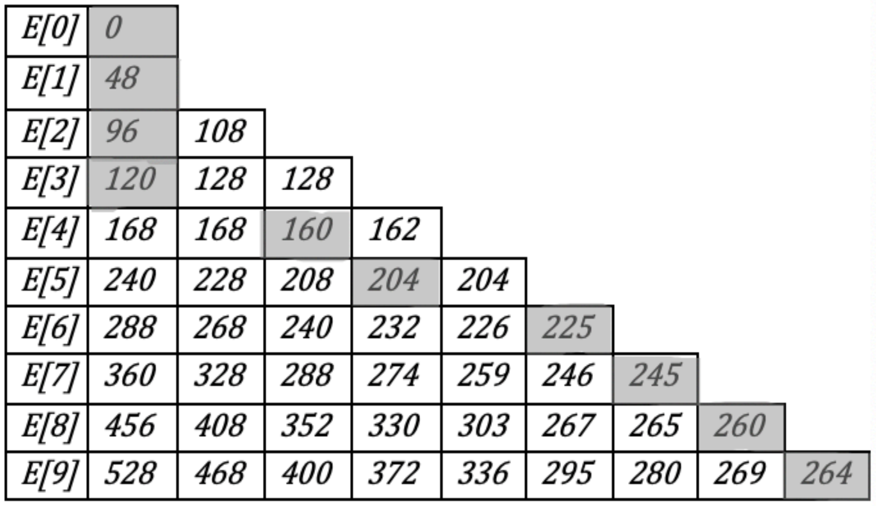
Shortest path can be computed using dynamic program depicted in Figure 6

$$E[m] = \text{cost of the shortest path from node 0 to node } m$$

$$= \min_{1 \le k \le m} \{E[k] + C_{k,m}\}$$

$when\ k < m, k = 0, \ldots, 8\ and\ m = 1, \ldots, 9.$

$E[0] = 0.$



E[0]	0								
E[1]	48								
E[2]	96	108							
E[3]	120	128	128						
E[4]	168	168	160	162					
E[5]	240	228	208	204	204				
E[6]	288	268	240	232	226	225			
E[7]	360	328	288	274	259	246	245		
E[8]	456	408	352	330	303	267	265	260	
E[9]	528	468	400	372	336	295	280	269	264

Figure 6: $E = E_{min}$ matrix with the highlighted value as each row minima.

Figure 7 represents the graph with final shortest path after we have reduced the 9-jobs list batching problem to shortest path problem with a dummy node *0*.

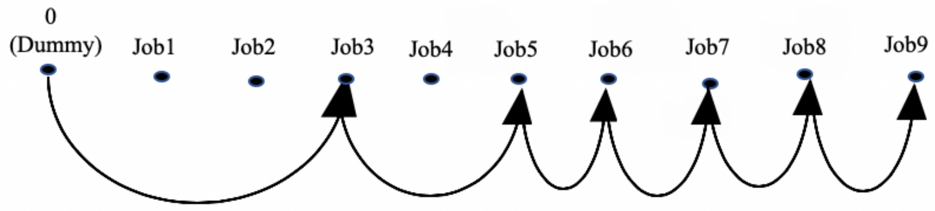


Figure 7: Shortest path graph for the given 9-jobs list batching problem.

CHAPTER 3

BATCHING WITH CONSTRAINTS

We now turn to the list batching problems with constraints. The solution to this is very useful in modern day batching problems which differ from the traditional ones in complexity.

3.1 BATCHING WITH BOUNDED NUMBER OF JOBS IN A BATCH

The list batching problem in which every batch has at least k number of jobs. We can compute cost matrix $C_{i,j}$ (as defined in 2) by making edge cost $C_{i,j} = \infty$ for batches where

$$|i - j| < k, \qquad \text{for } i < j,\ i = 0, 1, \ldots, n\text{-}1 \text{ and } j = 1, 2, \ldots, n.$$

Another list batching problem with constraint is the one in which every batch has at most k number of jobs. Cost matrix $C_{i,j}$ can be calculated using edge cost $C_{i,j} = \infty$ for the batches where

$$|i - j| > k, \qquad \text{for } i < j,\ i = 0, 1, \ldots, n\text{-}1 \text{ and } j = 1, 2, \ldots, n.$$

3.2 BATCHING WITH A FIXED NUMBER OF BATCHES (k-BATCH)

In this paper, a list batching problem with the condition that there must be exactly *k* number of batches is named as *k-Batch* problem. Each batch can have any number of jobs from $1, \ldots, n$. We can schedule the list of jobs by reducing this batching problem to a path problem (discussed in 2.3).

Consider a sequential list of jobs as $J_1(p_1, w_1), J_2(p_2, w_2), \ldots, J_n(p_n, w_n)$. The jobs $J_1, J_2, \ldots, J_n$ can be depicted by nodes $1, 2, \ldots, n$. We must add a dummy node 0 so that the path starts from 0^{th} node. Cost matrix $C = C_{i,j}$ is determined by the process explained in 2.2. The total number of batches is fixed to exactly k. Thus the k will be the total number of edges in graph. When number of batches k equals 1 then we have only one edge from source node 0 to destination node i. So, we evaluate minimum cost matrix $E_k[i]$ as

$$E_1[i] = C_{0,i}, \qquad \text{where } i = 1, 2, \ldots, n \text{ and } k = 1.$$

The matrix of minimum cost of paths from dummy node 0 to every other node can be calculated as

$$E = E_k[m, i] = (\text{cost of path from node } 0 \text{ to node } m, \text{ with } k\text{-}1 \text{ number of edges})$$
$$+ (\text{cost of one edge from node } m \text{ to last node } i).$$

i.e.,

$$E_k[m,i] = \begin{cases} \infty & if\ m \leq i < k \\ E_{k-1}[m] + C_{m,i} & otherwise \end{cases}$$

with $k = 1, 2, \ldots, n$. Since m is the last but one node, $m = 0, 1, \ldots, n\text{-}1$ and i is the last node, $i = 1, 2, \ldots, n$. These properties form a directed acyclic graph G such that the m^{th} node is always less than the i^{th} node. Starting from 0^{th} node to i^{th} node, the total number of edges is equal to k.

Figure 8 depicts the minimum cost matrix of paths formed by *'path from node 0 to node m with k-1 edges and last edge from node m to node i'.* Such matrix can be constructed for every k number of batches where $k = 1, \ldots, n$.

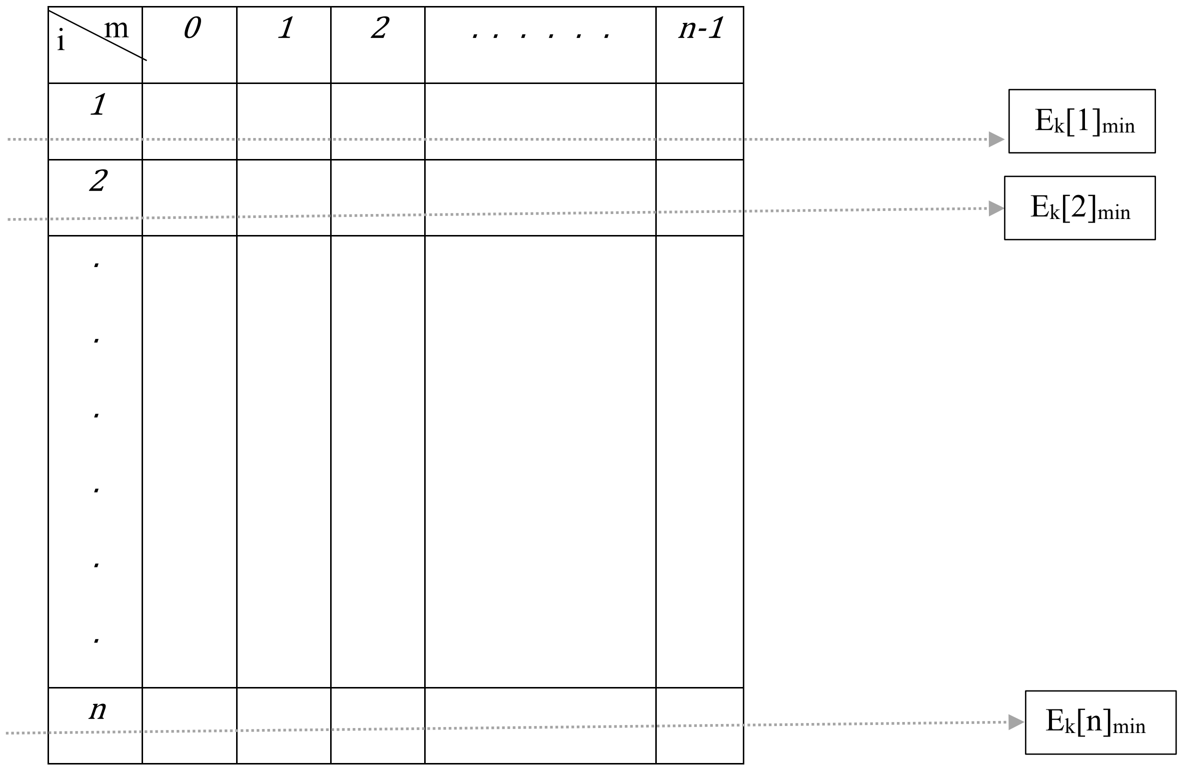


Figure 8: Minimum cost matrix of paths to node i with m as last but one node.

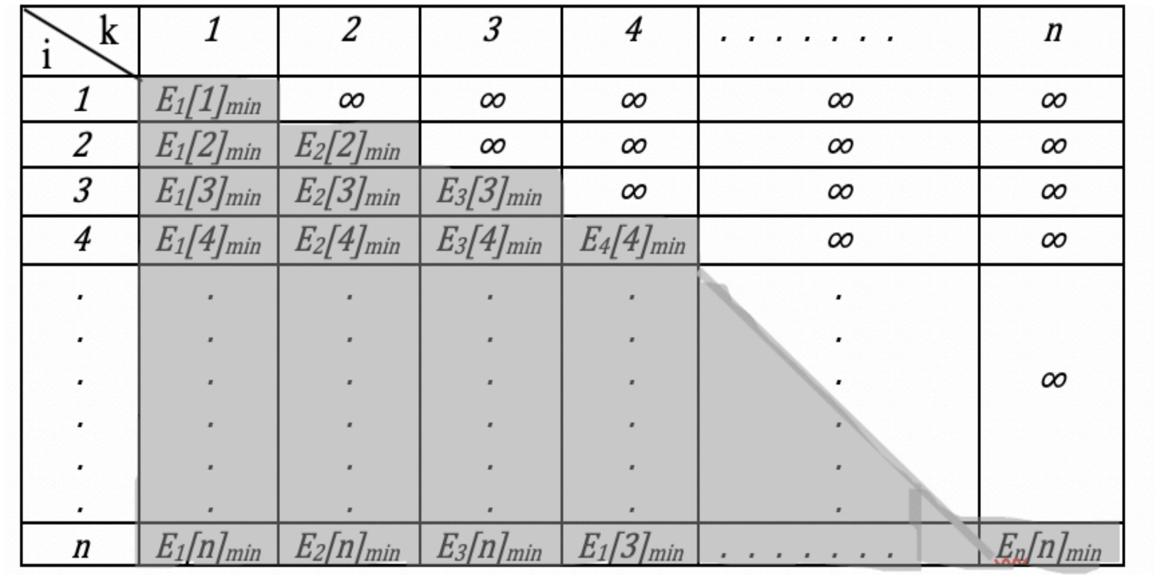
We can get the matrix $E_{min} = E_k[i]_{min}$ by consolidating the minima for all $k = 1, \ldots, n$ number of batches having last node $i = 1, \ldots, n$. Each cell value in a row of the matrix E can be evaluated using the minima of its previous row. This is depicted with the equation as

$$E_{min} = E_k[i]_{min} = \min_{0 \leq m < i} E_{k-1}[m, i],$$

for all $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, n$ such that $i \geq k$, see Figure 9.

Lemma 4 (from [3]) The matrix $E_{min} = E_k[i]_{min}$ is Monge.

Proof Monge property of matrix is preserved under addition and finding the minimum.



i \ k	1	2	3	4	. . . .	n
1	$E_1[1]_{min}$	∞	∞	∞	∞	∞
2	$E_1[2]_{min}$	$E_2[2]_{min}$	∞	∞	∞	∞
3	$E_1[3]_{min}$	$E_2[3]_{min}$	$E_3[3]_{min}$	∞	∞	∞
4	$E_1[4]_{min}$	$E_2[4]_{min}$	$E_3[4]_{min}$	$E_4[4]_{min}$	∞	∞
⋮	⋮	⋮	⋮	⋮	⋮	∞
n	$E_1[n]_{min}$	$E_2[n]_{min}$	$E_3[n]_{min}$	$E_1[3]_{min}$	. . . .	$E_n[n]_{min}$

Figure 9: Minimum cost matrix E_{min} shows the shortest path to node *i* with exactly *k* number of edges, where $k = 1, \ldots, n$.

3.3 ILLUSTRATION OF A k-BATCH PROBLEM REDUCED TO SHORTEST PATH PROBLEM

Consider a list of 5-jobs $J_i(w_i, p_i)$ in the same sequence as $J_1\,(1, 3) > J_2\,(1, 2) > J_3\,(2, 3) > J_4\,(1, 1) > J_5\,(2, 1)$. We will find the optimal solution to schedule these jobs into batches when the number of batches *k* is fixed.

The setup time *s* is *1*.

Partial sum of the processing times $P_i = \sum_{v=1}^{i} p_v$ *, for all* $i = 1, 2, \ldots, 5$ *and* $P_0 = 0$ *is*

$P_1 = 1$	$P_2 = 2$	$P_3 = 4$	$P_4 = 5$	$P_5 = 7$

Partial sum of the weights $W_i = \sum_{v=1}^{i} w_v$ *, for all* $i = 1, 2, \ldots, 5$ *and* $W_0 = 0$ is

$W_1 = 3$	$W_2 = 5$	$W_3 = 8$	$W_4 = 9$	$W_5 = 10$

The cost of each edge from i^{th} node (for all $i = 0, 2, \ldots, 8$) to j^{th} node (for all $j = 1, 2, \ldots, 9$) is given by $C_{i,j}$ as

C_{01}	20								
C_{02}	30	C_{12}	14						
C_{03}	50	C_{13}	28	C_{23}	15				
C_{04}	60	C_{14}	35	C_{24}	20	C_{34}	4		
C_{05}	80	C_{15}	49	C_{25}	30	C_{35}	8	C_{45}	3

We will now compute a matrix for minimum cost paths from dummy node 0 to node i, where $i = 1, 2, \ldots, 5$, given a fixed number of edges k. We will generate n such matrices taking into consideration each value of k as $1, 2, \ldots, 5$.

For the case when $k = 1$, the matrix $E_1[i]_{min}$ means the minimum costs of one edge from node 0 to node i. This is obtained from $C_{0,i}$*, for* $i = 1, \ldots, 5$ as

$E_1[i]_{min} =$

$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
20	30	50	60	80

Now for cases when $1 < k \leq 5$, we can do the calculations using

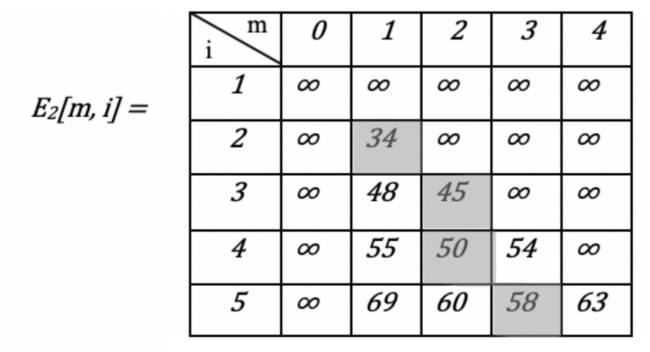
$$E_k[m,i] = \begin{cases} \infty & if\ m \leq i < k \\ E_{k-1}[m] + C_{m,i} & otherwise \end{cases}$$

with $k = 1, 2, \ldots, 5,$

m is the last but one node thus $m = 0, 1, \ldots, 4$, node 0 is the dummy node,

i is the last node so $i = 1, 2, \ldots, 5.$

When $k = 2$ then $E_2[i]_{min}$ represents the minimum costs of path with 2 edges from node 0 to node i. Each path consists of one edge from dummy node 0 to node m and the last edge from node m to node i, *for* $i = 1, 2, \ldots, 5.$



$E_2[m, i] =$

i \ m	0	1	2	3	4
1	∞	∞	∞	∞	∞
2	∞	34	∞	∞	∞
3	∞	48	45	∞	∞
4	∞	55	50	54	∞
5	∞	69	60	58	63

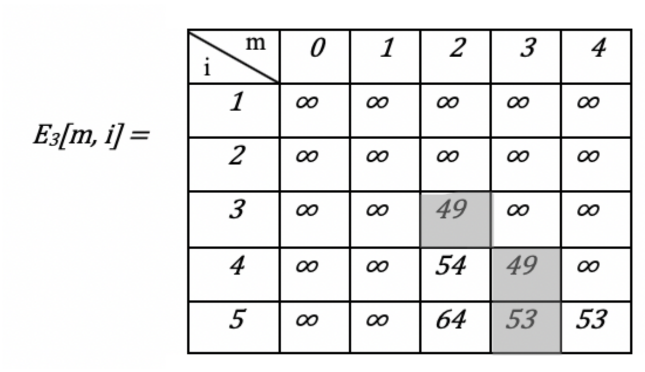
The minima of each row of matrix $E_2[m, i]$ is responsible to form the row 2 of the matrix $E_k[i]_{min}$. Thus

$E_2[i]_{min} =$

$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
∞	34	45	50	58

Similarly, when *k =3* then $E_3[i]_{min}$ represents the minimum costs of path with 3 edges from node *0* to node *i.* Each path consists of two edges from dummy node *0* to node *m* and the last edge from node *m* to node *i*, for *i = 1, 2, . . ., 5.*

$E_3[m, i] =$



m i	*0*	*1*	*2*	*3*	*4*
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	49	∞	∞
4	∞	∞	54	49	∞
5	∞	∞	64	53	53

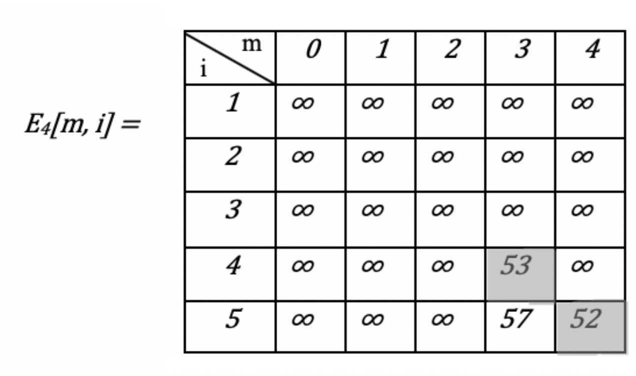
The minima of each row of matrix $E_3[m, i]$ is responsible to form the row 3 of the matrix $E_k[i]_{min}$. Thus

$E_3[i]_{min} =$

i = 1	*i = 2*	*i = 3*	*i = 4*	*i = 5*
∞	∞	*49*	*49*	*53*

When *k =4* then $E_4[i]_{min}$ represents the minimum costs of path with 4 edges from node *0* to node *i.* Each path consists of three edges from dummy node *0* to node *m* and the last edge from node *m* to node *i,* for *i = 1, 2, . . ., 5.*

$E_4[m, i] =$



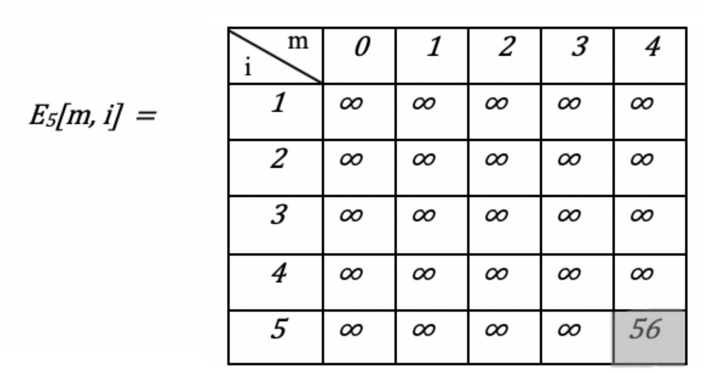
m i	*0*	*1*	*2*	*3*	*4*
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	53	∞
5	∞	∞	∞	57	52

The minima of each row of matrix *$E_4[m, i]$* is responsible to form the row 4 of the matrix *$E_k[i]_{min}$*. Thus

$E_4[i]_{min} =$

$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
∞	∞	∞	53	52

When *$k = 5$* then *$E_5[i]_{min}$* represents the minimum costs of path with 5 edges from node *0* to node *i*. Each path consists of four edges from dummy node *0* to node *m* and the last edge from node *m* to node *i*, for *$i = 1, 2, \ldots, 5$.*



$E_5[m, i]$ =

i \ m	0	1	2	3	4
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	56

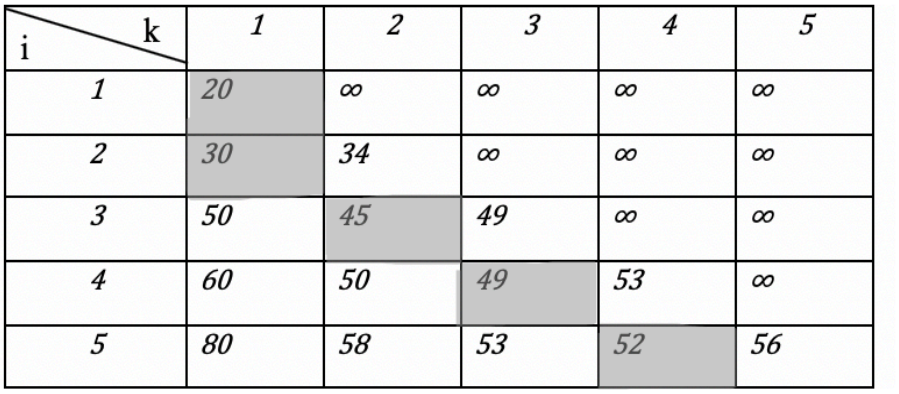
The minima of each row of matrix *$E_5[m, i]$* is responsible to form the row 5 of the matrix *$E_k[i]_{min}$*. Thus

$E_5[i]_{min} =$

$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
∞	∞	∞	∞	56

We can get the final *$E_{min} = E_k[i]_{min}$* matrix, see Figure 10, by putting together all the minima matrices computed earlier when *$k = 1, 2, \ldots, 5$.* The shaded parts in

Figure 10 represent shortest path from node *0* to node *i* with exactly *k* number of edges.



i \ k	1	2	3	4	5
1	20	∞	∞	∞	∞
2	30	34	∞	∞	∞
3	50	45	49	∞	∞
4	60	50	49	53	∞
5	80	58	53	52	56

Figure 10: Dynamic programming tableau of $E_k[i]_{min}$. Each row minima is highlighted in gray color.

CHAPTER 4

SPEEDING UP WITH LARSCH/ SMAWK ALGORITHM

The dynamic programming speedup can be illustrated by two important properties: total monotonicity and the Monge property.

Definition 1 A Monge matrix X is defined as

$$X[i,j] + X[i`,j`] \leq X[i`,j] + X[i,j`] \qquad for\ all\ i < i`\ and\ j < j`$$

Definition 2 A 2×2 matrix $\begin{pmatrix} p & q \\ r & s \end{pmatrix}$ is monotone if $q \leq p$ implies that $s \leq r$. In other words, a matrix is considered to be a monotone if following conditions hold true. We have the right-most minima of the upper row say R^{UR} and right-most minima of the lower row say R^{LR}, such that R^{UR} is not to the right of R^{LR}.

Definition 3 If all the 2×2 submatrices of a matrix X are monotone, then the matrix X is called totally monotone.

Observation All instances of every Monge matrix is a total monotone. Refer to the Figure 11. Thus, Monge matrices has tendency to occur routinely. This is evident from example, where we are trying to reduce the batching problem to the shortest path in section 2.3, the cost matrix C (see figure 5) is a Monge matrix with total monotonicity.

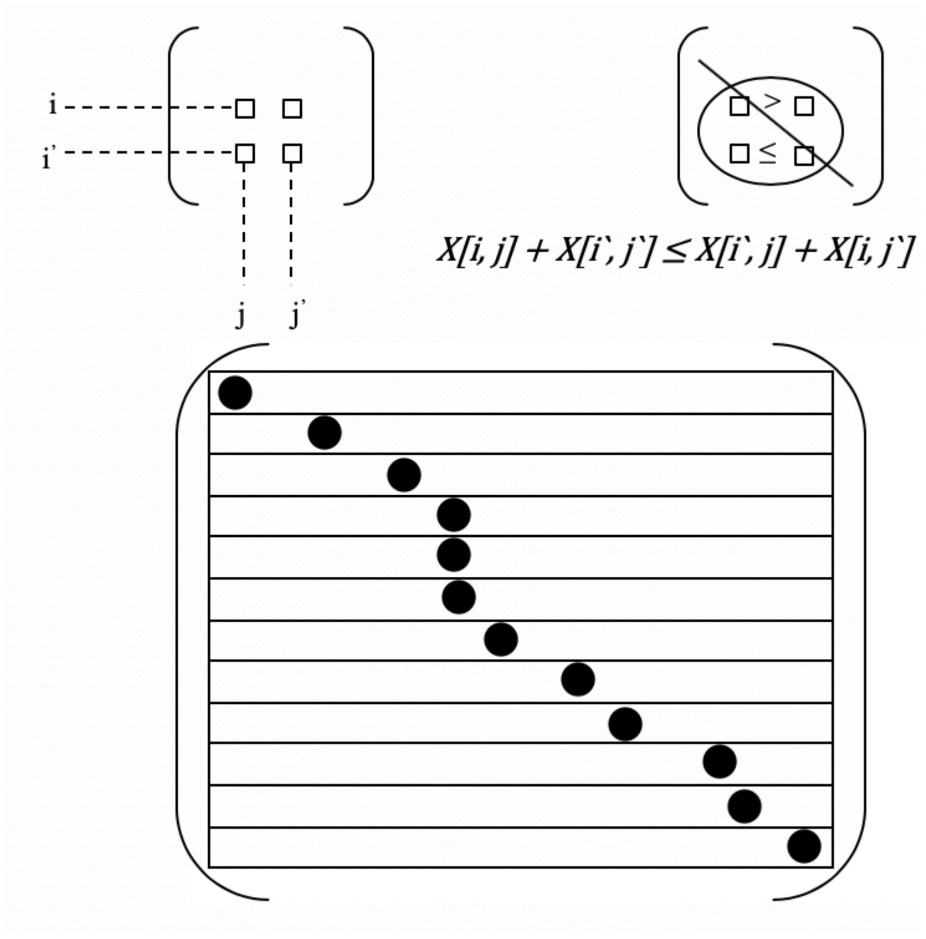


Figure 11: Monge property is shown in top left of the figure. This forbid the instance in top right. Top-left shows the Monge property, which prohibits the situation in top-right. This is evident at the bottom of figure where the row minima is inclined towards the right

The non-speedup method calculates all row minima of a totally monotone or Monge matrix in the runtime efficiency of *$O(n^2)$.* This runtime can be improved to *$O(n \log n)$* as depicted in Figure 12.

The trivial *$O(n \log n)$* algorithm works only when the entire matrix is available offline. This algorithm takes linear time to find the minimum value in the middle row of given matrix. Then row minima is computed by using recursion which is done on upper left and lower right matrix considering all rows have a minima Refer to Figure 12.

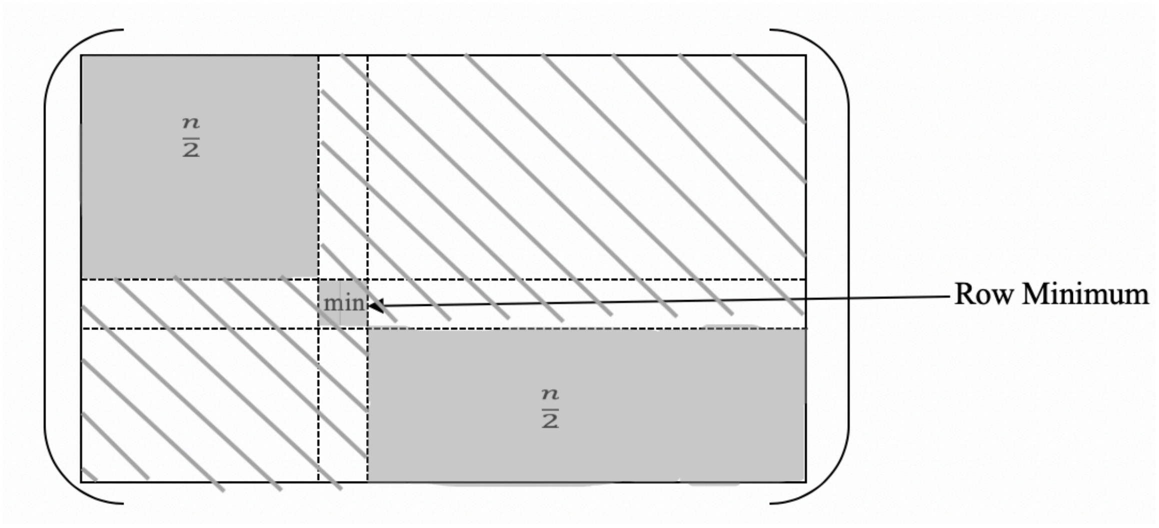


Figure 12: Computation of row minima for a Monge matrix. Time taken is *O(n log n)*.

We can use a complex recursive algorithm by Shor, Moran, Agarwal, Wilber, Klawe [3] known as SMAWK algorithm to further increase the speed. This can find all row minima in *O(n)* time. This method also requires the matrix to be ready offline as a pre-computational step.

The *k-Batch* problem discussed in 3.2, is one of the applications of the offline algorithms. As the first computational step we find all row minima in matrix $E_k[m, i]$. Perform first step for all values of *k*, where $k = 1, \ldots, n$ in runtime *O(n)*. Finally generate matrix $E_k[i]_{min}$ using *SMAWK* algorithm to improve overall runtime to $O(n^2)$ from $O(n^3)$, when using non-speedup algorithm.

Figure 13 demonstrates an online protocol which can be used to query every element: It will take constant time to generate elements in column 1, provided minima of row 1 is computed; following on, columns 1 and 2 can be easily generated if we can compute minima of row 2, and so forth.

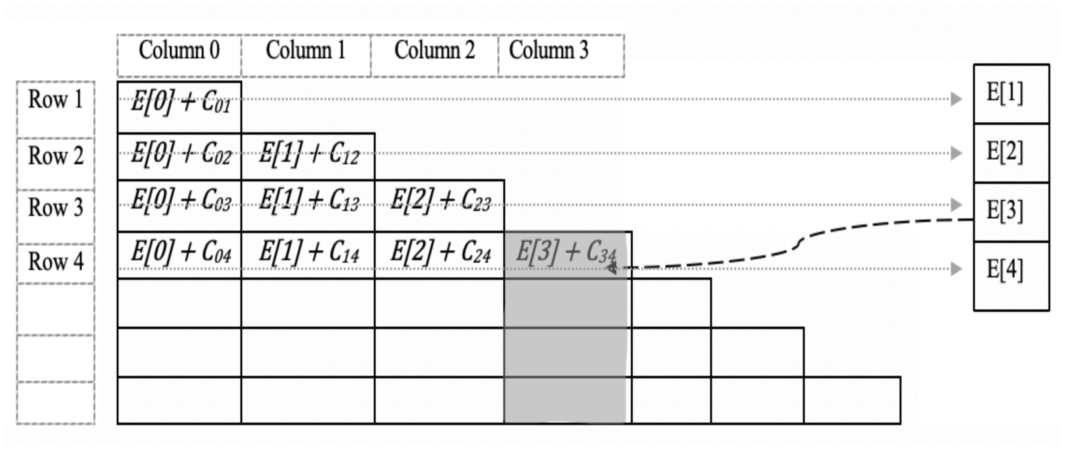


Figure 13: Online protocol tableau. Column 3 is ‘knowable’ if minimum of row 3 can be computed.

LARSCH an algorithm developed by Larmore and Schieber [4], can process the elements of matrix one-by-one in a serial manner, without having the entire matrix available from the start. The LARSCH algorithm generalizes SMAWK, which can be executed as an online algorithm with *O(n)* time. This runtime is greatly improved from the non-speedup online algorithm to find all row minima taking *O(n^2)* time.

Note that because of the previous Lemma construction, the following theorem is true.

Theorem 1 If the number of jobs is *n*, then it will take O(*n^2)* time to solve the *k-Batch* problem.

Proof Refer to Lemma 2 in section 2

CHAPTER 5

BATCHING WITH ARBITRARY ORDER

The problems in which there is no priority among the jobs to be batched, are categorized in this paper under general batching problems.

General batching problems have the following properties:

1. A fixed but random number of jobs are given in the problem.
2. The jobs can be independently scheduled irrespective of any order. There are no restrictions to batch and process any job.

The solution to general batching problems must have the given threshold value *T* as upper bound of the objective value.

Brucker and Albers [1] proved these general batching problems to be *NP*-hard and proved it by reduction from the *3-PARTITION* problem.

Definition 4 A *3-PARTITION* problem can be defined with a positive integer bound *P* and *3a* non-negative integers $m_1, m_2, \ldots, m_{3a}$ which satisfy the conditions

$$\frac{P}{4} < m_i < \frac{P}{2}$$

and

$$\sum_{i=1}^{3a} m_i = aP$$

where the integers m_i for $i = 1, 2, \ldots, 3a$.

Definition 5 For an instance of *3-PARTITION*, the general batching problem X can be defined as follows:

1. A job to be partitioned J_i (processing time $p_i = m_i$, weight $w_i = p_i$) for $i = 1, 2, \ldots, 3a$.
2. A 'dummy' job J_{3a+i} ($p_{4a+1} = 2P$, $w_{4a+1} = p_{4a+1}$) for $i = 1, 2, \ldots, 3a$.
3. Machine setup/delay time is $s = 2P$.
4. The optimal objective function T_F *of* the problem with $P(a+1)(a+2)$ number of jobs gives Threshold value T. Each job is independent with processing time of 1, $p_i = 1$ and flow time of f_i. This is the problem of $\sum f_i$ with setup time $s = 2P$.

The problem X has a solution S with $T_F(S) \leq T$ only when there exists a solution to the *3-PARTITION* problem. If it has the solution then every batch must have one dummy job along with all partition jobs J_i *where* $i \in I_j$ disjoint batches with $1 \leq j \leq a$.

For example, for a problem X we have an arbitrary solution S. Consider I_j which is set of jobs in batch j, and total K batches:

$$I_j := \{v \mid J_v \text{ is in batch } j\} \quad \text{where } j = 1, 2, \ldots, K.$$

It is known

$$\left(\sum_{v=I_1}^{I_j} w_v\right) = \left(\sum_{v=I_1}^{I_j} p_v\right)$$

and for *j = 1, 2, . . ., K* these aggregations end up as positive integer values. The solution *S* contains *n* jobs scheduled with processing time *p = 1* and weight *w = 1* such that

$$n = \sum_{j=1}^{K}\left(\sum_{v=I_1}^{I_j} w_v\right) = \sum_{j=1}^{K}\left(\sum_{v=I_1}^{I_j} p_v\right) = P(a+1)(a+2)$$

and *j*th batch contains $\left(\sum_{v=I_1}^{I_j} w_v\right) = \left(\sum_{v=I_1}^{I_j} p_v\right)$ jobs*, for all j = 1, 2, . . ., K.*

Lemma 5 Assume *X`* is a general batching problem with setup time of *2P.* An optimal solution $S_K$ to the problem *X`* is unique, when number of batches *K = a + 1* and batch sizes

$$n_j = (a+2-j)2P, \qquad for\ all\, J = 1,2,\ldots,K$$

Proof The solution S_k is an optimal solution of problem *X`*, as described in Proposition 2 by Dobson, Karmarkar and Rummel [5]. We must show that there is no other optimal solution than S_k.

Another optimal solution S_L to the problem *X`* with *L* number of batches such that $S_L \neq S_K$ and batch sizes m_j*, for all j = 1, 2, . . ., L.*

If $K < L$ then in that case $n_j > m_j$ should exist with index $1 < j < K$.

At the point when a job in batch set j is planned for another group K + 1 then S_K` is solution acquired from S_K. Due to the optimality of S_K we have,

$$T_F(S_K`) - T_F(S_K) = (K+1-j)2P - (n_j - 1) \geq 0$$

Similarly, for S_L` we have,

$$T_F(S_L`) - T_F(S_L) = -(L-j)2P + m_j - (m_L - 1) \geq 0$$

Two inequalities above, if added leads to,

$$(K + 1 - L)2P + (m_j + 2 - m_L - n_j \geq 0$$

The inequality $m_j + 2 - m_L - n_j = m_j - m_L + 1 - n_j + 1 \leq 0$ holds true because of our assumption of $L > K$ leading to $K + 1 - L \leq 0$, $n_j > m_j$ and $m_L \geq 1$. Thus, $K + 1 = L$ and $m_j - (m_L - 1) = n_j - 1$. Substituting these terms in the two inequalities and solving them further we obtain,

$$(K + 1 - j)2P = n_j - 1$$

put the first values for K and n_j we get we get a contradiction by the following equation,

$$(a + 2 - j)2P = (a + 2 - j)2P - 1$$

If we compare another parameter, given $L \leq K$, $n_j < m_j$ holds true, which has index as $1 \leq j \leq L$. If $L = K$ holds true, it will cause $n_k > m_k$ which will have index of $1 \leq k \leq L$. For the case when $L < K$ we set $k = L + 1$ and $m_k = 0$ for the rest of this proof. Let S_K` be the solution we get from $S_K$ when a job in batch $k$ is planned to run in batch $j$. Also, let $S_L$` be the solution acquired from S_L if a job in batch j is planned to run in batch k.

For $j < k$ we have,

$$T_F(S_K`) - T_F(S_K) = -(K - j)2P + n_j - (n_K - 1) \geq 0$$

And

$$T_F(S_L`) - T_F(S_L) = (K - j)2P - (m_j - 1) + m_k \geq 0$$

After adding these two inequalities and using our assumption of $n_j + 1 - m_j \leq 0$ *and* $m_k + 1 - n_k \leq 0$ *we get*

$$(K-j)2P-n_j+(n_k-1)\geq 0$$

We conclude that $0=(K-j)2P-n_j+(n_K-1)=(K-j)2P+(j-K)2P-1=-1$, resulting in a contradiction.

For $k<j$ then

$$T_F(S_K`)\ -\ T_F(S_K)=(j-k)2P-(n_K-1)+n_j\geq 0$$

and

$$T_F(S_L`)\ -\ T_F(S_L)=-(j-K)2P+m_k-(m_k-1)\geq 0$$

We add these two inequalities. Then if we apply resultant equation to summed up inequalities we have

$$(j-K)2P-(n_k-1)+n_j=0$$

Substituting n_k and n_j by original values to get

$$0=(j-K)2P-(n_K-1)+n_j=(j-K)2P+(K-j)2P+1=1$$

The above equation is a contradiction. Hence proved that S_K is the only optimal solution of X.

Theorem 2 If and only if *3-PARTITION* has a solution then the problem X has a solution S such that $T_F(S)\leq T$

Proof Consider X has an arbitrary solution which is S and the count of batches in S be K and batch j has set of job indices which are I_j

$$I_j\coloneqq\{v\mid J_v\ is\ scheduled\ in\ batch\ j\}\quad where\ j=\ 1,2,\dots,K$$

Solution S is assumed to be a solution with scheduled $n=P(a+1)(a+2)$ jobs which have running times of $p_i=1$ with job weights as $w_i=1, i=1, 2, \dots, n.$

As per the Lemma 5, S is a solution of X, given $T_F(S) \leq T$. This condition holds true only in the event that it is conceivable to plan the dummy jobs along with partition jobs such that the subsequent solution comprises of precisely $a + 1$ groups and the accompanying condition is fulfilled for $j = 1, 2, \ldots, a + 1$

$$\sum_{v=I_1}^{I_j} p_v = (a + 2 - j)2P$$

In order to get the solution discussed above, it is mandatory to schedule the dummy job J_{3a+i} in batch $i = 1, 2, \ldots, a + 1$. Also, it is mandatory to schedule the batch on the first position, which has total processing time of P. *3-PARTITION* should have a solution for above to hold true. Say we have a total sum of weights P for given sets $I_1, I_2, \ldots, I_a$, then partition jobs J_i which have $i \in I_j$, $1 \leq j \leq K$ can be put together with *jth* batch.

3-PARTITIO when transformed to X takes $O(a)$ steps. Hence, for the given general batching problem *NP*-hardness proof is complete.

Bein, Noga and Wiegley [6] explained the approximation algorithm to solve the general batching problems. One way is to re-organize the jobs according to their priorities $\frac{W_i}{P_i}$, where n is total number of jobs and $i = 1, 2, \ldots, n$. When the order of jobs is adjusted based on their priorities such that $\frac{W_1}{P_1} \geq \frac{W_2}{P_2} \ldots \geq \frac{W_n}{P_n}$, then the jobs are said to be in canonical order. CANONICALBEST approximation algorithm are those which can schedule the given jobs in their respective canonical order.

Quality of approximation by CANONICALBEST is measured by its approximation ratio A_R. For an optimization problem O_P, an algorithm C has approximation ratio A_R if for every instance $i \in O_P$,

$$A_R \leq \frac{cost\ of\ the\ solution\ given\ by\ C\ for\ instance\ i}{cost\ of\ the\ optimal\ solution\ for\ instance\ i}$$

Approximation ratio is A_R of 2, for the given priority algorithm. In this algorithm batches are made such that they have decreasing order of priority. It was also discussed that approximation ratio on any priority algorithm will have a lower bound of $\frac{2+\sqrt{6}}{4} \approx 1.1124$. There exists a conjecture that matches this bound. As the algorithm requires the priorities to be sorted firth, this will have a runtime of at least $O(n \log n)$.

We can solve the prioritized (canonical order) list of jobs in runtime of $O(n)$ using the List batching process as discussed in 2.2. Only the order of jobs in the optimal solution should be known for the list batching process.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In many practical situations, batching the problem and scheduling helps in processing efficiently. Though, there are much faster ways to solve such problems efficiently, but generally these are solved by simple- dynamic programming methodologies. We have studied here a one-machine list batching problem where the number of batches is arbitrary but fixed. We have given the algorithm to solve an offline form of this problem. The solution uses a dynamic program that has improved the speed to a runtime of $O(n^2)$. It is often realized, that while solving extremely big real world problems, having an algorithm with solution, which works in cubic time is equivalent to having no solution at all.

However, it is an interesting open research problem to find an optimal and feasible solution to the *k-Batch* problems with additional constraints, for example, when each batch can have a given fixed number of jobs, when the setup time is included as a part of each batch, etc. Another thought-provoking topic to investigate would be the problems where structure of batch is predefined. For example, a given batch structure allows to have two jobs with processing time < 2 and other two jobs with processing time > 2. The applications to these kinds of problems will also be of great interest. These problems have a relationship to knapsack and bin packing. Dynamic programming plays a vital role in the optimal solution to such problems.

APPENDIX A

SELECTED SOURCE CODE

```
''' Calculate initial cost matrix, i.e. cost from node 0 to node i'''
    # s -- selection time
    s = 1

    # n -- number of jobs
    n = int(input('\nEnter the total number of jobs:'))
    self.n = n

    # p -- matrix of processing times of jobs
    p = [int (x) for x in input('\nEnter the processing time for each job:').split()]

    # w -- matrix of weights of jobs
    w = [int (x) for x in input('\nEnter the weight for each job:').split()]

    # P -- partial sum of processing times p
    P = [0] * (n+1)
    for i in range(1, n+1):
      sum = 0
      for j in range(i):
        sum += p[j]
      P[i] = sum
    print ('\nP -- partial sum of processing times p:', P)

    # W -- partial sum of weights w
    W = [0] * (n+1)
    for i in range(1, n+1):
      sum = 0
      for j in range(i):
        sum += w[j]
      W[i] = sum
    print ('\nW -- partial sum of weights w:', W)

    # C -- matrix of edge costs; C[i,j] -- cost of edge (i,j)
    # C[i,j] = (Wn - Wi)(s + Pj - Pi)
    C = dict()
    print('\nInitial Matrix of edge costs:')
    for i in range(n+1):
      for j in range(i+1, n+1):
        C[i,j] = ((W[n] - W[i]) * (s + P[j] - P[i]))
```

```
            print ('cost of edge (',i, ',', j, ') :',C[i,j])

        # Path always start from 0
        self.cost_zero_to_i = C

''' Compute the minimum cost matrix Emin of shortest path to last node i with exactly k
number of edges'''
def minimum_cost(self):
        # map to store the minimum value of E in each k
        minimumE = dict()
        # map to store current calculations for E
        currentE = dict()

        for y in range(1, self.n+1):
          minimumE[1,y] = self.cost_zero_to_i[0,y]

        # iterate to increment number_of_edges(k) by one till k<=n, start with k=2
        for k in range(2, self.n+1):

          # iterate to increment last_node_number(i) by one till i<=n, start with i=2
          for i in range(1, self.n+1):

 # iterate to increment last_but_one_node_number(l) by one till l<=n, start with l=1
            for l in range(1, self.n+1):
              if i < k or l >= i:
                currentE[i,l] = float('inf')
              else:
                currentE[i,l] = minimumE[k-1,l] + self.cost_zero_to_i[l,i]

          # add next row to minimum_E matrix
          for y in range(1, self.n+1):
            minimumE[k, y] = self.find_minimum(currentE, y)

        return minimumE

    def find_minimum(self,curr_E, row):
        mini = float('inf')
        for i in range(1, self.n+1):
          if curr_E[row, i] < mini:
            mini = curr_E[row, i]

        return mini
```

BIBLIOGRAPHY

[1] Albers, Susanne & Brucker, Peter. (1993). The complexity of one machine batching problem. Discrete Applied Mathematics. 47. 87-107. 10.1016/0166-218X(93)90085-3.

[2] Coffman, Ed & Yannakakis, Mihalis & Magazine, Michael & Santos, Cipriano. (1990). Batch Sizing and Job Sequencing on a Single Machine. Annals of Operations Research. 26. 135-147. 10.1007/BF02248589.

[3] A. Aggarwal, M.M. Klawe, S. Moran, P.W. Shor, R.E. Wilber, Geometric applications of a matrix-searching algorithm. Algorithmica 2(1), 195–208 (1987); A preliminary version appeared, in Proceedings of the 2nd Annual Symposium on Computational Geometry (1986), pp. 285–292.

[4] L. Larmore, Lawrence & Schieber, Baruch. (1990). On-Line Dynamic Programming with Applications to the Prediction of RNA Secondary Structure. Journal of Algorithms - JAL. 12. 503-512. 10.1016/0196-6774(91)90016-R.

[5] Dobson, Gregory, et al. “Batching to Minimize Flow Times on One Machine.” Management Science, vol. 33, no. 6, 1987, pp. 784–799.

[6] Bein, Wolfgang & Noga, John & Wiegley, Jeff. (2007). Approximation for Batching via Priorities. Scientific Annals of Computer Science. 17.

[7] R. Kuik, M. Salomon, and L. N. van Wassenhove. Batching decisions: structure and models. European journal of operational research, 75:243– 263, 1994.

[8] Webster, Scott, and Kenneth R. Baker. “Scheduling Groups of Jobs on a Single Machine.” Operations Research, vol. 43, no. 4, 1995, pp. 692–703.

[9] C. N. Potts and L. N. Van Wassenhove. Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. The Journal of the Operational Research Society, 43:395–406, 1992.

[10] Brucker, Peter & Hurink, Johann. (1999). Solving a Chemical Batch Scheduling Problem by Local Search. Annals of Operations Research. 96. 10.1023/A:1018959704264.

[11] Bein W.W., Epstein L., Larmore L.L., Noga J. (2004) Optimally Competitive List Batching. In: Hagerup T., Katajainen J. (eds) Algorithm Theory - SWAT 2004. SWAT 2004. Lecture Notes in Computer Science, vol 3111. Springer, Berlin, Heidelberg.

[12] Kovalyov M.Y., Oulamara A., Soukhal A. (2012) Two-Agent Scheduling on an Unbounded Serial Batching Machine. In: Mahjoub A.R., Markakis V., Milis I., Paschos V.T. (eds) Combinatorial Optimization. ISCO 2012. Lecture Notes in Computer Science, vol 7422. Springer, Berlin, Heidelberg.

[13] Integer and combinatorial optimization, George L. Nemhauser and Laurence A. Wolsey, Wiley-Interscience Series in Discrete Mathematics and Optimization, New York, 1988, ISBN 0-471-82819-X, 763pp.

[14] Burkard, Rainer & Klinz, Bettina & Rudolf, Rüdiger. (1997). Perspectives of Monge Properties in Optimization. Discrete Applied Mathematics. 70. 95-161. 10.1016/0166-218X(95)00103-X.

[15] Agnetis, Alessandro & Pacciarelli, Dario & Pacifici, Andrea. (2007). Multi-agent single machine scheduling. Annals OR. 150. 3-15. 10.1007/s10479-006-0164-y.

[16] Brucker, Peter & Jurisch, Bernd & Krämer, Andreas. (1997). Complexity of scheduling problems with multi-purpose machines. Annals of Operations Research - Annals OR. 70. 57-73. 10.1023/A:1018950911030.

[17] Baptiste, P & Brucker, P & Knust, S & Timkovsky, Vadim. (2004). Ten notes on equal-execution-time scheduling. 4OR. 2. 111-127.

[18] Uzsoy, Reha & Yang, Yaoyu. (2009). Minimizing total weighted completion time on a single batch processing machine. Production and Operations Management. 6. 57 - 73. 10.1111/j.1937-5956.1997.tb00415.x.

[19] Yuan, Jinjiang & P. Shang, W & Feng, Q. (2005). A note on the scheduling with two families of jobs. J. Scheduling. 8. 537-542. 10.1007/s10951-005-4997-z.

[20] Du, Donglei. (2008). Scheduling Algorithms by Peter Brucker. SIAM Review. 50. 169-171. 10.2307/20454082.

[21] Bein, Wolfgang & Golin, Mordecai & L. Larmore, Lawrence & Zhang, Yan. (2006). The Knuth-Yao Quadrangle-Inequality Speedup is a Consequence of Total-Monotonicity. ACM Transactions on Algorithms. 6. 31-40. 10.1145/1109557.1109562.

[22] Baptiste, Philippe. (2000). Batching Identical Jobs. Mathematical Methods of Operations Research. 52. 355-367. 10.1007/s001860000088.

CURRICULUM VITAE

GRADUATE COLLEGE
UNIVERSITY OF NEVADA, LAS VEGAS

SHRADHA KAPOOR
shradha_kapoor@outlook.com

Degrees:

- Bachelor of Engineering in Information Science, 2013
  Visvesvaraya Technological University, India
- Master of Science in Computer Science, 2019
  University of Nevada, Las Vegas

Thesis Title: **Batching problems with constraints**

Thesis Examination Committee:

- Committee Chair, Dr. Wolfgang Bein, Ph.D
- Committee Member, Dr. Lawrence Larmore, Ph.D
- Committee Member, Dr. Laxmi Gewali, Ph.D
- Graduate College Representative, Dr. Zhiyong Wang, Ph.D