

33488

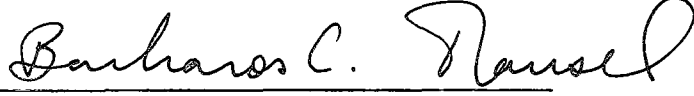
SINGLE MACHINE TOTAL TARDINESS PROBLEM:
EXACT AND HEURISTIC ALGORITHMS BASED ON
 β -SEQUENCE AND DECOMPOSITION THEOREMS

A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL
ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Bahar Kara
September, 1994

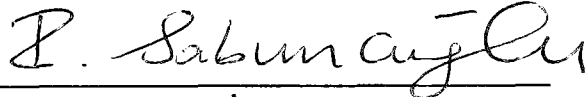
T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANASYON MERKEZİ

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



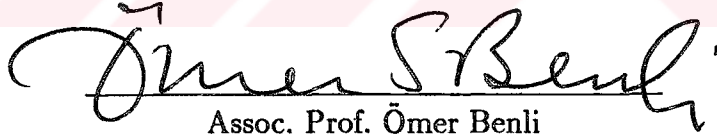
Assoc. Prof. Barbaros Ç. Tansel(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Assist. Prof. İhsan Sabuncuoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Assoc. Prof. Ömer Benli

Approved for the Institute of Engineering and Sciences:



Prof. Mehmet Baray

Director of Institute of Engineering and Sciences

ABSTRACT

SINGLE MACHINE TOTAL TARDINESS PROBLEM: EXACT AND HEURISTIC ALGORITHMS BASED ON β -SEQUENCE AND DECOMPOSITION THEOREMS

Bahar Kara

M.S. in Industrial Engineering

Supervisor: Assoc. Prof. Barbaros Ç. Tansel

September, 1994

The primary concern of this thesis is to analyze single machine total tardiness problem and to develop both an exact algorithm and a heuristic algorithm. The analysis of the literature reveals that exact algorithms are limited to 100 jobs. We enlarge this limit considerably by basing our algorithms on the β -Sequence and decomposition theorems from the recent literature. With our algorithm, we exactly solve 200 job problems in low CPU time, and we also solved 120 out of 160 test problems with 500 jobs. In addition we develop a heuristic based on our exact algorithm which results in optimum solutions in 30% of test problems and stays with 9% of the optimal in all test runs.

Key words: Single Machine Scheduling, Minimizing Total Tardiness, Exact Algorithms, Heuristics

ÖZET

TEK MAKİNEDE TOPLAM GECİKMEYİ EN AZLAMA PROBLEMİ : β -SIRALAMASI VE AYRIŞTIRMAYA DAYANAN KESİN ÇÖZÜMLÜ VE SEZGİSEL ALGORİTMALAR

Bahar Kara

Endüstri Mühendisliği Bölümü Yüksek Lisans

Tez Yöneticisi: Doç. Dr. Barbaros Ç. Tansel

Eylül, 1994

Bu tez çalışmasında tek makinede toplam gecikmeyi enazlama problemi için kesin çözümlü ve sezgisel algoritmalar önerilmiştir. Literatür incelemesinde, bilinen kesin çözümlü algoritmaların 100 iş sayısı ile sınırlı olduğu görülmektedir. Bu çalışmada yakın zamanda geliştirilmiş olan β -sıralaması ve ayrıştırma yöntemleri kullanılarak oluşturulan kesin çözümlü algoritmalar ile 200 iş sayılı problemler hızlı çözüme ulaştırılırken 500 iş sayısı içeren 160 test probleminin de 120 si çözüme ulaştırılmıştır. Ayrıca, bu çalışmada kesin çözümlü algoritmaya dayanan bir de sezgisel yöntem geliştirilmiştir. Sezgisel yöntem eniyi çözüme oldukça yakın sonuçlar vermektedir ve test problemlerinin %30 unda eniyi çözümü vermiş, bütün test problemlerinde ise optimalden sapması %9 un içinde kalmıştır.

Anahtar sözcükler: Tek makinede çizelgeleme, Toplam Gecikmeyi Enazlama, Kesin Çözümlü Algoritmalar, Sezgisel Algoritmalar.

ACKNOWLEDGEMENT

I am mostly grateful to Associate Professor Barbaros Tansel for suggesting a research topic full of enthusiasm, and who has been supervising me with patience and everlasting interest and being helpful in any way during my graduate studies.

I am also indebted to Associate Professor Ömer Benli and Assist Professor İhsan Sabuncuoğlu for showing keen interest to the subject matter and accepting to read and review this thesis. Their remarks and recommendations have been invaluable.

I wish to express my deepest gratitude Dr. Vedat Verter for his invaluable guidance.

I would like to express my deepest thanks to my family, without whom this study would not have been possible. To my father, Professor İmdat Kara for his valuable suggestions and encouragements, to my husband Kadri Yetiş for his love, understanding and moral support and finally to my mother Günay Kara, my sister Gonca Kara and my mother-in-law Pınar Yetiş for their prays.

I would also like to extend my sincere thanks to my office mates, Muhittin Hakan Demir and Sibel Salman for their encouragements and moral support.

Contents

1	INTRODUCTION	1
2	LITERATURE REVIEW	5
2.1	COMMONLY USED THEOREMS	5
2.2	OPTIMIZATION ALGORITHMS	7
2.2.1	Dynamic Programming Based Algorithms	7
2.2.2	Branch and Bound Algorithms	9
2.3	HEURISTICS	11
3	β-SEQUENCE and DECOMPOSITION THEOREMS	15
3.1	Emmons' theorems	15
3.2	β -sequence	24
3.3	Decomposition theorems	27
4	NEW EXACT ALGORITHMS	41
4.1	Algorithm Beta(TS)	41

4.2	The Algorithm Beta(PW)	51
4.3	Algorithm Beta(TS, PW)	53
4.4	Computational Results For Beta(TS, PW)	58
5	A NEW HEURISTIC : Beta	64
5.1	PSK Heuristic	64
5.2	Our Heuristic: Beta	67
5.3	Comparisons	71
6	CONCLUSIONS and FUTURE RESEARCH	75



List of Figures

1.1	Illustration of Canonical Schedule	3
3.1	Illustration for an interchange	17
3.2	The conditions of cell 7	20
3.3	Illustration for a movement	21
3.4	Graph for the illustration of enclosing rectangle	24
3.5	Figure for the illustration of decomposition theorem	28
3.6	Illustration of a forward movement	30
3.7	Illustration of a backward movement	31
3.8	Data plot of example 5	40
4.1	Flow chart for the algorithm Beta(TS)	43
4.2	Data plot for example 6	48
4.3	Tree structure	50
4.4	Flow chart for the algorithm Beta(PW)	52
4.5	Flow chart for the algorithm Beta(TS, PW)	54

4.6 Graph of the solution times of both of the algorithms 60

5.1 Figure for illustration of a swap 68



List of Tables

2.1	Table of Dynamic Programming Based Exact Algorithms	8
2.2	Table of Branch & Bound Algorithms	11
2.3	Table for the heuristics	12
3.1	Table for tardiness changes after an interchange	18
3.2	Tardiness changes after an interchange	19
3.3	Table for tardiness changes after a movement	22
3.4	Tardiness changes after a movement	22
3.5	Tardiness changes resulting from the forward movement	30
3.6	Tardiness changes resulting from the backward movement	31
4.1	Table for comparison of the CPU times	59
4.2	Table for the computational results of Beta(TS, PW)	62
5.1	Tardiness changes caused by swap	69
5.2	Average Deviations of the heuristics from optimum	72
5.3	Average Deviations of the heuristics for cases they differ	72

5.4 Average deviations without the extreme cases 73

5.5 Deviations of the heuristics from optimum 74



Chapter 1

INTRODUCTION

Scheduling may be defined as "the allocation of resources over time to perform a collection of tasks" (Baker, 1974). In this study we look at scheduling problems which arise in manufacturing systems. A schedule specifies when and on which machine each job i is to be processed. The aim is to find a schedule that optimizes some performance measure. Performance measures are mainly in two categories: *regular* performance measures and *non-regular* performance measures. If a performance measure is non-decreasing in each of the job completion times it is called a *regular* performance measure, otherwise it is called *non-regular*. In this thesis we select *total tardiness* as the performance measure and restrict ourselves to a single machine. Total tardiness is a regular performance measure. In a manufacturing system, each job has a due date at which time it needs to be ready, and if that job is not ready at its due date, it is called *tardy* and it is penalized. The sum of penalties for all jobs yields the total tardiness. If we also wanted to penalize the jobs which are completed before their due dates, then we would have a non-regular performance measure. This time the measure is not non-decreasing in each of job completion times. It may be better to force the job to wait even if the machine is idle. This is called "idle time insertion" and does not result in any improvement if a regular performance measure is used. In problems with regular performance measures, once we find the order of the processing jobs, which is called a sequence, we also

have the schedule since idle time insertion is unnecessary and so the sequence is the same with the corresponding schedule that has zero idle time between jobs. This is also the case for our problem. Our feasible set consists of the $n!$ possible permutations of the jobs.

Let us define the problem. Consider n jobs to be processed without interruption on a single machine which can handle one job at a time. Let $J = \{1, 2, \dots, n\}$ denote the indices of the job set. Each job i is available at time zero and has an integer processing time denoted by p_i . Each job i is to be completed at a given date d_i .

If we denote by $T_i(S)$ the tardiness of job i and by $C_i(S)$ the completion time of job i in a schedule S then

$$T_i(S) = \max\{0, C_i(S) - d_i\}.$$

Defining \mathcal{S} to be the set of all permutations of $1, 2, \dots, n$ the problem is to

$$\min_{S \in \mathcal{S}} \sum_{i=1}^{i=n} T_i(S)$$

If we want to assign different priorities to jobs for being tardy, we have the *Total Weighted Tardiness Problem* which is

$$\min_{S \in \mathcal{S}} \sum_{i=1}^{i=n} w_i T_i(S)$$

where w_i is the weight associated with job i .

The weighted tardiness problem is shown to be NP - Hard in the strong sense by Lenstra, Rinnooy Kan, and Brucker in 1977 [19]. The complexity status of the unweighted case remained open until 1990. Then Du and Leung [9] showed that the problem is NP-hard in the ordinary sense. It is instructive to give the main idea of the proof of Du & Leung.

Du & Leung showed the NP-Hardness of the total tardiness problem by a reduction from a restricted version of the NP-Complete *Even-Odd Partition* problem. The Even-Odd partition problem and the Restricted Even-Odd partition problem can be stated as follows.

Even-Odd partition : Given a set of $2n$ positive integers $B = \{b_1, b_2, \dots, b_{2n}\}$ such that $b_i > b_{i+1}$ for $1 \leq i < 2n$, is there a partition of B into two subsets B_1 and B_2 such that $\sum_{i \in B_1} b_i = \sum_{i \in B_2} b_i$ and such that for each $1 \leq i \leq n$, B_1 (and hence B_2) contains exactly one of $\{b_{2i-1}, b_{2i}\}$?

Restricted Even-Odd partition : Given a set of $2n$ positive integers $B = \{a_1, a_2, \dots, a_{2n}\}$ such that $a_i > a_{i+1}$ for $1 \leq i < 2n$, $a_{2j} > a_{2j+1} + \delta$ for each $1 \leq j < n$ and $a_j > n(4n+1)\delta + 5n(a_1 - a_{2n})$ for each $1 \leq i \leq 2n$ where $\delta = 0.5 \sum_{i=1}^{i=n} (a_{2i-1} - a_{2i})$, is there a partition of A into two subsets A_1 and A_2 such that $\sum_{i \in A_1} a_i = \sum_{i \in A_2} a_i$ and such that for each $1 \leq i \leq n$, A_1 (and hence A_2) contains exactly one of $\{a_{2i-1}, a_{2i}\}$?

The additional constraints on A are imposed to facilitate the NP - Hardness proof of the total tardiness problem. First the authors showed that the restricted Even-Odd partition problem is NP - Complete.

The authors showed that the total tardiness problem is NP - Hard by showing the corresponding decision problem to be NP - Complete. The decision version of the total tardiness problem can be stated as follows.

Given an integer k and a set $J = \{1, \dots, n\}$ of n independent jobs, process times $p_i \in \mathbb{Z}^+ \forall i \in J$ and due dates $d_i \in \mathbb{Z} \forall i \in J$, is there a permutation $S \in \mathcal{S}$ such that $\sum_{i=1}^{i=n} T_i(S) \leq k$?

The authors first describe a reduction from the Restricted Even-Odd Partition problem to the total tardiness problem. For that, the authors constructed an instance of the total tardiness problem with $3n + 1$ jobs labeled as $V_1, V_2, \dots, V_{2n}, W_1, W_2, \dots, W_{n+1}$. Letting $V = \{V_1, V_2, \dots, V_{2n}\}$ and $W = \{W_1, W_2, \dots, W_{n+1}\}$ partition set V in two subsets $\{V_{1,1}, V_{2,1}, \dots, V_{n,1}\}$ and $\{V_{1,2}, V_{2,2}, \dots, V_{n,2}\}$. With these sets, the authors defined the term *Canonical Schedule* as a schedule of the type below :



Figure 1.1: Illustration of Canonical Schedule

The schedule can be considered in two parts. The first part is composed of two

tuples of jobs, the first element supplied from the first partition of the V set and the second element supplied from the W set. The second part contains only the jobs in the second partition of the V set.

With $v_{i,j}$ denoting the process time of job $V_{i,j}$ for $j = 1, 2$ and $\forall i \in \{1, 2, \dots, n\}$, we have $\{v_{i,1}, v_{i,2}\} = \{a_{2i-1}, a_{2i}\}$ for each $1 \leq i \leq n$. The authors proved that there is always an optimal schedule which is a canonical one. For this proof and in the construction of the canonical schedule, they used the theorems of Emmons [11] and some results from Baker [2]. Then they showed that the total tardiness of a canonical schedule S , denote by $TT(S)$, satisfies : $TT(S) \geq k$. Moreover, the equality holds if and only if $\sum_{i=1}^{i=n} v_{i,1} = \sum_{i=1}^{i=n} v_{i,2}$.

It follows that the total tardiness problem is NP-Complete \square .

In this thesis, we give computationally effective exact and heuristic algorithms for the single machine total tardiness problem. In chapter 2 we review the literature on the single machine total tardiness problem. Then we analyze in chapter 3 some important results from the literature which we base our research on. These are the β -Sequence of Tansel & Sabuncuoğlu [34] and decomposition theorems of again Tansel & Sabuncuoğlu and Potts & Wassenhove [23]. We also discuss the well known theorems of Emmons [11]. Then we give the exact algorithms that we have developed. There are three different algorithms and the most improved one, which we call Beta(TS, PW), is capable of handling 500 jobs, even though it cannot solve all of the instances to optimality, whereas the maximum number of jobs in the literature is limited to 100. These algorithms together with an explanatory example and computational results are given in chapter 4. We give a new heuristic in chapter 5 whose observed performance is at least as good as or better than the heuristic of Panwalkar et al. [22] which is the most successful heuristic in the literature. The last chapter gives conclusions and outlines future research.

Chapter 2

LITERATURE REVIEW

Before explaining what we have done in this thesis, it will be better to review the literature first so as to identify some of the deficiencies in the area. The first section discusses important theorems, the second section is devoted to a discussion of exact algorithms, and the third section is devoted to heuristics developed so far.

2.1 COMMONLY USED THEOREMS

This section gives the theoretical background for the single machine total tardiness problem. The results that we give in this section have been used by nearly all researchers in this area.

It will be better to begin this section with the well known lemma of Elmaghraby given in 1968 [10]. The lemma says that among a subset S of unscheduled jobs (each available at time 0), if there is a job $k \in S$ such that $d_k \geq \sum_{i \in S} p_i$ then there exist an optimal schedule in which k is last among all jobs in S . This is very intuitive. If job k has such a due date d_k then it will not be tardy if we process it last among the ones in hand.

An important study which found many applications is the well known

paper of Emmons 1969 [11]. In that study, Emmons derived three basic theorems that establish precedence relations between job pairs according to their process times and due dates. Emmons also derived some corollaries which identified any jobs, if possible, being first or last among the unscheduled ones.

Later, in 1975, Rinnooy Kan et al. [28] derived similar relations but for arbitrary nondecreasing cost functions.

In a recent work, Tansel and Sabuncuoğlu [35] interpreted the theorems of Emmons with a geometric viewpoint which makes the theorems easy to handle and more understandable. Following that study Tansel and Sabuncuoğlu [34] derived some conditions which identify certain sequences as being optimal or not.

In 1977, Lawler [16] developed a theorem which is also applicable to the weighted tardiness case. This theorem is not for finding precedence relations; instead, it gives a decomposition principle. Decomposing refers to dividing the problem into two or more sets and solving each set separately. The decomposition theorem of Lawler assumes that the jobs are in EDD order and then finds alternative decompositions which result from moving the longest unscheduled job to different places. In order to find the optimal sequence, all alternative decompositions must be carried on. The decomposition takes pseudopolynomial time. It is in $O(n^4 * p_{max})$ where p_{max} is the maximum process time or in $O(n^3 * P)$ where P is the total processing time.

Later, Potts and Wassenhove [23, 26] worked on the decomposition theory of Lawler and they decreased the number of possible decompositions by imposing some extra conditions on the decomposition theorem of Lawler.

Then in 1994, Tansel and Sabuncuoğlu [34] give a different type of decomposition theorem. Their decomposition theorem does not assume EDD ordering and it does not try to decompose the problem according to the possible places of the longest job. This decomposition theorem finds one exact decomposition according to the job it applies, but this time there is no guarantee that the problem decomposes. That is, the theorem may not apply for

any job in hand in which case we will not have any decomposition.

2.2 OPTIMIZATION ALGORITHMS

The algorithms which search for the optimum for single machine total tardiness problem are mainly in two categories: dynamic programming algorithms and branch & bound algorithms. Branch & bound algorithms usually suffer from high running times and dynamic programming algorithms suffer from high core storage requirements. The best algorithms in the literature can go up to 100 jobs both for branch & bound and dynamic programming.

2.2.1 Dynamic Programming Based Algorithms

Dynamic programming based algorithms are the earliest available algorithms for the single machine total tardiness problem [3]. In finding an optimal sequence with dynamic programming, the typical approach is to identify a set of jobs, say S , with S to be scheduled at the last $m = |S|$ places of the sequence and to find the job in S which will be scheduled first by evaluating all. This is repeated until all jobs are scheduled.

There are many dynamic programming algorithms. Srinivasan 1972 [33], Lawler 1977 [16], Schrage and Baker in 1978 [4, 29] Potts and Wassenhove in 1982 and 1987 [23, 25] are the main ones. The table 2.1 summarizes their results.

The algorithm of Lawler [16] is a pseudopolynomial time algorithm which evaluates all possible decompositions via dynamic programming. The author did not give any computational results but later Potts and Wassenhove [23] made a computational study of this algorithm. The algorithm could not go further than 50 jobs. Lawler later developed a fully polynomial approximation scheme for his algorithm [17]. The bound of $O(n^3P)$ is transformed to $O(n^7/\epsilon)$ with some manipulations

Authors	Any Important property	Computational results
Srinivasan 1972	-	50 jobs in 0.36 CPU secs at UNIVAC 1108
Lawler 1977	Decomposition based	Done by Potts & Wassenhove in 1987. 50 jobs in 8.74 CPU secs on CDC 7600
Baker & Schrage 1978	Developed for problems restricted with precedence relations	Core storage requirements permits up to 30 jobs, solved in 0.2 CPU secs
Schrage & Baker 1978	Good labeling techniques	50 jobs
Potts & Wassenhove 1982	Decomposition based	100 jobs in 27.07 CPU secs
Potts & Wassenhove 1987	Slight modifications	100 jobs in 27.07 secs case is chosen at the end

Table 2.1: Table of Dynamic Programming Based Exact Algorithms

In 1978, Baker and Schrage gave two algorithms on this topic. These algorithms are the best known dynamic programming algorithms and have been used by many subsequent researchers. The first one [4] is for problems in which jobs have precedence restrictions. The authors give an approach which can also be used for problems with no precedence restrictions. The authors also give importance to labeling of the sets.

The second paper of Schrage and Baker [29] works on better labeling techniques than the one they proposed in their previous paper. They work out a good algorithm for labeling.

Both algorithms have low CPU time and the core storage permitted handling as many as 50 jobs which are solved in less time than any branch and bound algorithm. The technique given in the second paper is even better than the one based on the chain structure because of the improved labeling. With this algorithm it becomes easier to retrieve the sets when they are needed. This dynamic programming algorithm is used as a subroutine by some subsequent researchers.

In 1987, Potts and Wassenhove [25] proposed a method which makes some modifications on the dynamic programming algorithms of Lawler and then of Schrage & Baker's. Their decomposition theorem is applied to the problem in hand until it can be solved by the dynamic programming algorithm of Schrage & Baker. Then, they made some modifications such as using Elmagraby's lemma [10] in branching, using a technique which prevented solving the same problem more than once, and recognizing easily solvable cases like the ones giving SPT optimum or EDD optimum, before attempting to solve the entire set. With these kinds of modifications, the authors managed to solve 100 jobs in less time than they did before. The final step they reach in this study is the best known exact algorithm. It solves 100 jobs in 27.07 seconds on the average on a CDC 7600 computer. The code is in FORTRAN IV.

2.2.2 Branch and Bound Algorithms

Branch and bound algorithms appeared with the development of theorems that give precedence relations. The first branch and bound algorithm which is based on precedence relations is developed by Elmagraby in 1968 [10] and by Emmons in 1969 [11]. Then Schwimmer in 1972 [30], Rinnooy Kan et al. in 1975 [28], Fisher in 1976 [12], Picard and Queyranne in 1978 [21], Potts & Wassenhove in 1985 [24] are the other studies which find an optimum via a branch & bound algorithm. The table 2.2 summarizes the state of the art on branch and bound.

In 1976, Fisher [12] developed the best known branch and bound algorithm in terms of the tightness of the lower bound used. He considers the single machine as posing a constraint on the feasible set of the problem. He performed a Lagrangean relaxation on that constraint and the solution of the relaxed problem gave a lower bound for the initial one. The branch and bound algorithm is based on backwards scheduling with depth first search. Nodes correspond to the set of scheduled jobs up to that time. Begin branching by the possible set of last jobs (known by the use of Emmons' theorems) and apply depth first search by taking into account the precedence relations (resulting

from Emmons theorems) until fathoming occurs. The fathoming criteria are:

- i) an initial solution for the problem is first calculated with any heuristic. Use Carroll's heuristic [5] which is a construction heuristic based on Emmons theorems. At each node the calculated lower bound is compared with the solution of the heuristic and if the lower bound is greater than the total tardiness of the solution, then the node is fathomed.

- ii) if the Lagrangean resulted in an optimal solution then the node is fathomed. You have the solution.

- iii) if for any two different nodes, the nodes contain the same set of scheduled jobs but have different costs, then the one with higher cost is fathomed (fathom with respect to dominance criteria).

This algorithm can solve up to 50 jobs in reasonable time but it works with small p_i only since the solution to the Lagrangean is obtained in $O(n^2 p_{avg})$ where p_{avg} is the average processing time. The algorithm can solve 50 job problems and only if jobs have small process times.

The algorithm of Potts and Wassenhove [24] is the fastest one among the known branch & bound algorithms. Their algorithm is again based on backwards scheduling with depth first search. They also calculate a lower bound for each node. Their lower bound is easy to calculate, is not as good as Fisher's, but not bad either. In calculating the lower bound they relax the problem so that the resulting problem becomes the minimization of the total completion times. The method is successful up to 50 jobs for the weighted case.

Authors	Problem	Any Assumption	Computational results
Elmaghraby 1968	Weighted tardiness	-	None reported
Emmons 1969	Total tardiness	-	None reported
Schwimmer 1972	Weighted tardiness	small p_i	20 jobs in 0.29 secs
Rinnooy Kan et al 1975	Any nondecreasing cost function.	Alg. tested for weighted tardiness	20 jobs but could not solve all
Fisher 1976	Total tardiness	Small p_i	50 jobs 63.49 CPU secs
Picard et al. 1977	Weighted tardiness	Time dependent TSP application	20 jobs in 136.6 secs.
Sen et al. 1983	Total tardiness	-	The experiments are not comparable. Solved 5 problems with 100 jobs.
Potts & Wassenhove 1985	Weighted tardiness	-	50 jobs in 7.9 secs.

Table 2.2: Table of Branch & Bound Algorithms

2.3 HEURISTICS

The research after 1990, at which time the NP - Completeness is proved, focused mainly on finding good heuristics. There were studies on heuristics before that time also, but those were mainly in construction type heuristics which may help in upper bounding. In construction heuristics, the schedule is built from scratch by fixing the position of one job at each step. Carroll 1965 [5], Wilkerson and Irwin 1971 [36], Morton and Rachamadugu in 1982 [20], Baker and Bartrand [1] have construction heuristics for the problem. The Wilkerson & Irwin heuristic is in fact a combination of a construction plus some improvement heuristic.

Different types of improvement heuristics began to appear after 1990. Potts and Van Wassenhove in 1991 [26], Lowe et al in 1991 [6] are among those studies. The table 2.3 summarizes existing heuristics in the literature.

Authors	Name of heuristic	Any Property	Complexity
Caroll 1965	COVERT	Construction	$O(n^2)$
Morton & Rachamadugu 1982	Apparent Urgency	Construction	$O(n^2)$
Baker & Bartrand	Modified Due Date	Construction	$O(n \log n)$
Wilkerson & Irwin 1971	WI	Construction plus Interchange	-
Potts & Wassenhove 1991	-	Decomposition incorporated with any heuristic	-
Lowe et al. 1991	-	Relaxing Emmons theorems and solving with dynamic programming of Schrage and Baker in sets.	-
Fry et al. 1989	API	9 different interchanges. Better than WI	-
Holsenback & Russel 1992	-	Construction. Very simple, even hand calculation is possible for $n < 20$ Better than API	$O(n^2)$
Panwalkar et al. 1993	PSK	Construction. Best known	$O(n^2)$

Table 2.3: Table for the heuristics

Any of the scheduling rules can be accompanied with interchange heuristics. That is, the resultant schedule of any of the above heuristics can be put into local search heuristics. In 1990, Chang et al. [7] analyzed many types of local search heuristics and derived the worst case behavior for them. They analyzed four main local search heuristics:

ADJ : Adjacent Interchange - Only interchanging a pair of two adjacent jobs
 K - INT : K - Interchange, Interchanging any pair of jobs at most k times successively

B - S : Backwards Shift, Moving one of the jobs backwards.

G - S : General Shift, Moving one of the jobs forwards or backwards.

They defined a local search heuristic to be any method that starts with an initial sequence and searches for another permutation of the jobs which results in less tardiness.

The authors found out that for $n > 4$, in the worst case, the first two interchange based heuristics, ADJ and K - INT can have arbitrarily large (may be ∞) relative error (the relative error is defined as the ratio of the worst local optimum value to the global optimum). The authors also showed that shift based heuristics, B - S and G - S, have finite relative error in the worst case.

Another heuristic is developed by Fry et al. [14] in 1989 which is based on adjacent pairwise interchanges (API). Since adjacent pairwise interchanges can only result in a local optimum, the authors tried to evaluate different solutions and select the best at the end. The initial sequence in starting the local search affects the solution, so the authors tried three different initial sequences, namely, SPT, EDD, and SLK (Smallest Slack Rule) which schedules jobs in nondecreasing order of $C_i - d_i$ where C_i denotes the completion time of job i . The interchange strategy will also affect the result. The authors matched the three initial sequences with three different interchange strategies so there were nine different solutions for each problem. At the end they select the best of these nine solutions as the result of the API heuristic.

Most recently, in 1992, Holsenback and Russel [15] developed another heuristic. In their review of the literature the authors say that the API heuristic of Fry et al. (explained before) was the best of the existing heuristics in terms of solution quality where solution quality is defined as the mean percentage deviation of the heuristic from the optimum. The authors claim that they developed a better one, both in solution quality and running time. The heuristic starts with an EDD schedule and tries to improve the sequence. The *reducible tardiness* criterion is used in the improvement. Any job which has tardiness greater than its processing time ($T_k > p_k$ where T_k is the tardiness of job k) is said to have reducible tardiness. In the algorithm, inspecting jobs from the last to first, the first job having reducible tardiness, say job k , is identified. Given k , the predecessors of job k (in the current sequence) are inspected in the order $k-1, k-2, ..$ until the first job is found whose movement to position $k+1$ (i.e. right after k) improves the total tardiness. With this movement the tail end of the sequence from positions $k+1$ up to n is fixed. For the remaining jobs (i.e. jobs $1, \dots, k$) the same procedure is applied. The

algorithm continues until no job with reducible tardiness can be found. The complexity of the algorithm is $O(n^2)$. The important property of this heuristic is its easiness. Even hand calculation is possible with this heuristic for $n < 20$.

In 1993, a better heuristic is developed by Panwalkar et al. [22]. This heuristic is the best known one for the single machine total tardiness problem, both in solution time and in solution quality. The authors compared their heuristic with that of Wilkerson and Irwin, Holsenback and Russel and the API method of Fry et al. and showed, by making many experiments, that their heuristic is better than all of the others. The algorithm makes n passes from left to right and at the k^{th} pass it picks one job and schedules it to the k^{th} position. Each pass starts with the smallest job in hand, calls it the active job, and tries to fix this job by considering some inequalities or changes the active job.



Chapter 3

β -SEQUENCE and DECOMPOSITION THEOREMS

In this chapter we analyze the earlier results from the literature which we use in our algorithms. The first section gives Emmons theorems [11]. The second section is on the β -Sequence of Tansel & Sabuncuoğlu [34] and the third section gives the decomposition theorems of Lawler [16], Potts & Wassenhove [23], and Tansel & Sabuncuoğlu [34]

3.1 Emmons' theorems

The theorems of Emmons are the basic building blocks in the scheduling theory of total tardiness. Nearly all subsequent researchers used these theorems in their studies.

To state the theorems assume jobs are indexed according to the nondecreasing processing times by breaking ties with nondecreasing due dates. That is $j < k$ implies $p_j < p_k$ or $p_j = p_k$ and $d_j \leq d_k$. This indexing will

be referred to as the SPT (Shortest Processing Time) indexing. Throughout the thesis, we use SPT indexing unless otherwise mentioned and we denote $p^* = \sum_{i \in J} p_i$.

We define a subset B_j of J to be a before-set of j if there exists an optimal sequence in which every job in B_j precedes job j . Similarly, a subset A_j of J is defined to be an after-set of job j if there is an optimal sequence in which every job in A_j succeeds job j . We also define $\bar{A}_k = \{i : i \notin A_k\}$

Theorem 1 For $j < k$ if B_k is a before set of job k and

$$d_j \leq \max \left\{ \sum_{i \in B_k} p_i + p_k, d_k \right\}$$

then $B_k \cup \{j\}$ is also a before set of job k (i.e there exist an optimal sequence in which every job in B_k as well as job j precede job k).

Theorem 2 For $j < k$ if B_k and A_k are before and after sets of job k ,

respectively, and

$$\begin{aligned} \text{i)} \quad & d_j > \max \{ \sum_{i \in B_k} p_i + p_k, d_k \} \\ \text{ii)} \quad & d_j + p_j \geq \sum_{i \in \bar{A}_k} p_i \end{aligned}$$

then $A_k \cup \{j\}$ is also an after set of job k .

Theorem 3 For $j < k$ if A_j is an after set of job j and

$$d_k \geq \sum_{i \in A_j} p_i$$

then $A_j \cup \{k\}$ is also an after set of job j .

We use the notation $j \leftarrow k$ to mean there exist an optimal sequence in which job j precedes job k .

The first theorem gives the conditions for a smaller job being a predecessor of a larger one whereas the second theorem is for a larger job being a predecessor of a smaller one. Repeated use of the theorems will give many relations and those relations will form succesively expanding before-sets and after-sets of each job. That is once we find j is before k we insert job j in

the most recent before set of job k and insert job k in the most recent after set of j .

The proofs of the theorems are based on either interchanging two jobs or moving one job to a later position. While making these movements or interchanges, say for proving $j \leftarrow k$, the author assumes the opposite (that is, job k is placed before job j in a sequence) and shows that the tardiness of the sequence will not get worse by interchanging jobs j and k or by moving job j right after job k . The following discussion give the main ideas that lead to Emmons' theorems.

Let $j < k$ and suppose we have a sequence S in hand with job k placed before job j . Jobs j and k may or may not be adjacent. Now we study the changes in the tardiness function when we interchange jobs j and k and we derive conditions under which the interchange decreases the total tardiness.

Let the sequence after interchanging job j and job k be \bar{S} .

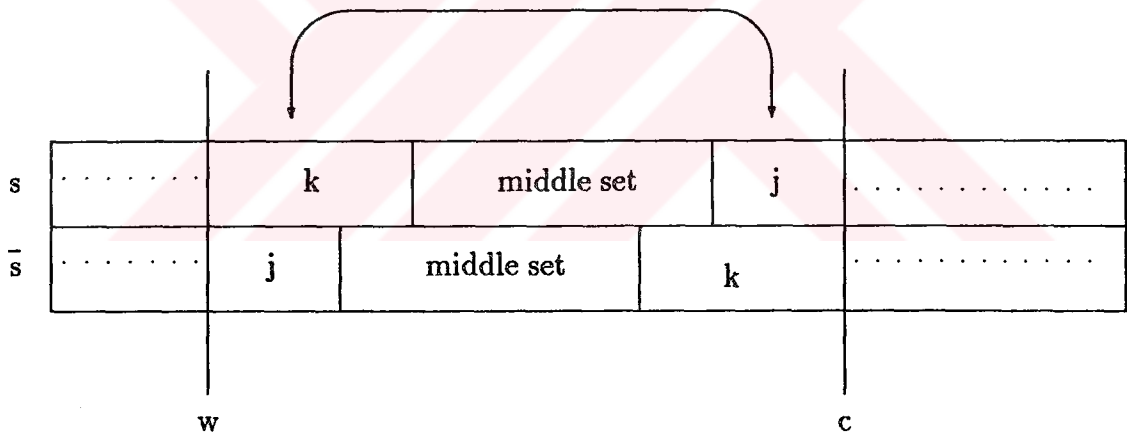


Figure 3.1: Illustration for an interchange

Let W be the waiting time of job k in S and C be the completion time of job j in S . So only the jobs whose completion times are between W and C are affected from this movement. We call these jobs as the *middle set* and denote the tardiness of these jobs in sequence \bar{S} by $\bar{T}(\text{middle set})$. If we define $T(S)$ to be the tardiness of sequence S and $T(\bar{S})$ to be tardiness

of \bar{S} , we have

$$T(S) = \max\{0, W + p_k - d_k\} + \max\{0, C - d_j\} + T(\text{middle set}) + K$$

$$T(\bar{S}) = \max\{0, W + p_j - d_j\} + \max\{0, C - d_k\} + \bar{T}(\text{middle set}) + K$$

where K is the total tardiness of the unaffected jobs.

Since $j < k$ implies $p_j \leq p_k$ the tardiness of the middle set can not get worse. The change in tardiness is

$$\bar{\Delta} \triangleq T(S) - T(\bar{S})$$

$$\geq \max\{0, W + p_k - d_k\} + \max\{0, C - d_j\} - \max\{0, W + p_j - d_j\} - \max\{0, C - d_k\}$$

where the inequality follows from $T(\text{middle set}) - \bar{T}(\text{middle set}) \geq 0$. Define the right hand side of the inequality to be Δ .

We consider 16 cases corresponding to two possibilities for each maximand. The following table identifies the 16 possibilities.

Aft. Move \rightarrow Bef Move \downarrow	j=Tardy k=Tardy	j=Tardy k=NotTardy	j=NotTardy k=Tardy	j=NotTardy k=NotTardy
j=Tardy k=Tardy	1	X - (a)	2	X - (a)
j=Tardy k=NotTardy	3	4	5	6
j=NotTardy k=Tardy	X - (b)	X - (a)(b)	7	X - (a)
j=NotTardy k=NotTardy	X - (b)	X - (b)	8	0

Table 3.1: Table for tardiness changes after an interchange

In the table, we marked some of the cells by X to indicate that the cell will not arise. For example, for $X(a)$, the condition of the cells mean job k is tardy in S and it becomes not tardy in \bar{S} which is impossible. Also for $X(b)$, the condition of the cells show job j is not tardy in S whereas it becomes tardy in \bar{S} which is impossible.

Now let us see the change in the tardiness for each of the cell in the table above. We look at Δ .

Cell 1 $\Delta = (W + p_k - d_k + C - d_j) - (W + p_j - d_j + C - d_k) = p_k - p_j \geq 0$
 since $j < k$

Cell 2 $\Delta = (W + p_k - d_k + C - d_j) - (C - d_k) = W + p_k - d_j$

Cell 3 $\Delta = (C - d_j) - (W + p_j - d_j + C - d_k) = d_k - W - p_j \geq d_k - W - p_k \geq 0$
 since k is not tardy in S in this cell so that $d_k \geq W + p_k$.

Cell 4 $\Delta = (C - d_j) - (W + p_j - d_j) = C - W - p_j \geq p_k \geq 0$

Cell 5 $\Delta = (C - d_j) - (C - d_k) = d_k - d_j$

Cell 6 $\Delta = (C - d_j) \geq 0$

Cell 7 $\Delta = (W + p_k - d_k) - (C - d_k) = W + p_k - C \leq 0$ since $C \geq W + p_k + p_j$

Cell 8 $\Delta = -(C - d_k) < 0$

Let us summarize the results of Δ in the next table

Aft. Move \rightarrow Bef Move \downarrow	j=Tardy k=Tardy	j=Tardy k=NotTardy	j=NotTardy k=Tardy	j=NotTardy k=NotTardy
j=Tardy k=Tardy	(1) ≥ 0	X	(2) $W + p_k - d_j$	X
j=Tardy k=NotTardy	(3) ≥ 0	(4) ≥ 0	(5) $d_k - d_j$	(6) ≥ 0
j=NotTardy k=Tardy	X	X	(7) < 0	X
j=NotTardy k=NotTardy	X	X	(8) < 0	0

Table 3.2: Tardiness changes after an interchange

In order for the interchange to improve tardiness we want $\Delta \geq 0$ for all of the cells. Since there are some cells with $\Delta < 0$ and others where Δ can be negative, zero or positive we need to impose conditions to avoid them. We only need to look at the cells 2, 5, 7 and 8. Since we have $d_k - d_j$ in one of the cells, let us first impose the condition $d_k \geq d_j$.

Cell 2 $W + p_k - d_j = ?$ Since we use $d_k \geq d_j$

$$W + p_k - d_j \geq W + p_k - d_k \geq 0 \text{ since } k \text{ is tardy in } S \text{ and } W \geq 0$$

Cell 5 $d_k - d_j \geq 0$ by the imposed condition.

For the cells 7 and 8 we need conditions for avoiding them. Let us see if $d_k \geq d_j$ works here or not

Cell 7 In this cell job j is not tardy in both sequences and job k is tardy in both of them.

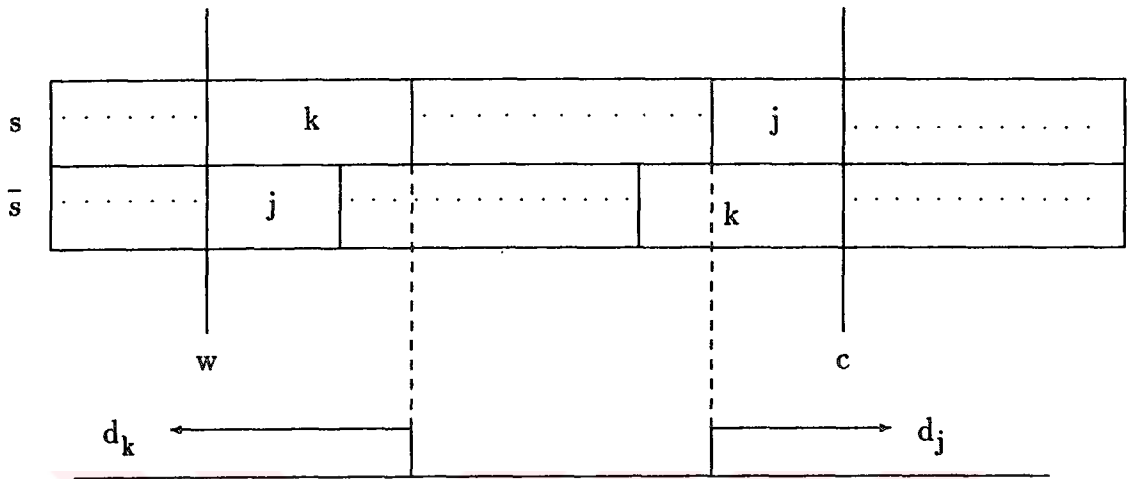


Figure 3.2: The conditions of cell 7

So if $d_k \geq d_j$ this case will not arise.

Cell 8 is similar to cell 7 and the condition $d_k \geq d_j$ avoids this cell also.

So if $p_j \leq p_k$ and $d_j \leq d_k$ then the sequence \bar{S} has no more tardiness than S and so we can say that in some optimal sequence job j will be scheduled before job k (otherwise, interchanging those jobs will not increase the tardiness).

If we define $P(B_k)$ as the total process times of the jobs in the before-set of job k , with similar arguments, $\Delta \geq 0$ condition can also be satisfied with $P(B_k) + p_k \geq d_j$ condition.

So for $p_j \leq p_k$ either $d_j \leq d_k$ or $d_j \leq P(B_k) + p_k$ is needed to have an improvable interchange. Hence, if $p_j \leq p_k$ and $d_j \leq \max\{d_k, P(B_k) + p_k\}$ then job j is before job k in some optimal sequence. This is the first theorem of Emmons.

Now suppose we know that the above inequality is not satisfied for jobs j and k . Since we cannot say that j is before k in some optimal sequence, let us see if we can say that k is before j in some optimal sequence. This time put j before k in a sequence S and form the second sequence \bar{S} by moving job j to the position right after job k . The figure below will be helpful.

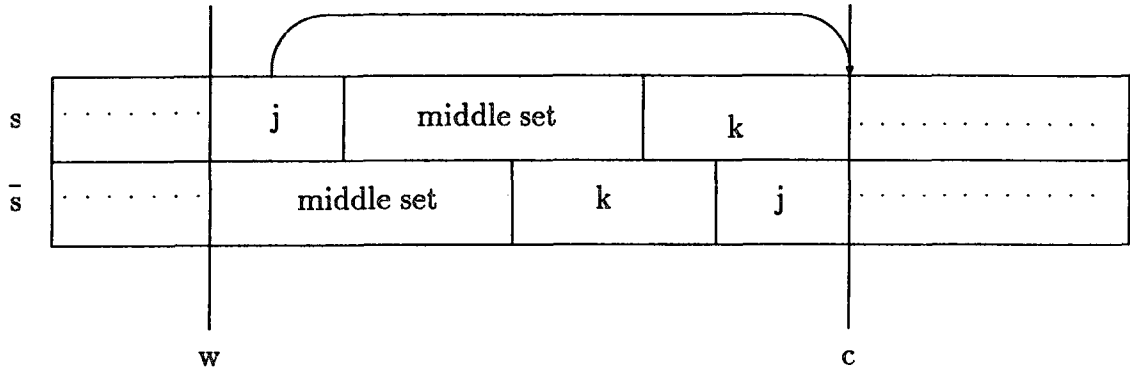


Figure 3.3: Illustration for a movement

Again, the total tardiness for S and \bar{S} are

$$T(S) = \max\{0, W + p_j - d_j\} + \max\{0, C - d_k\} + T(\text{middle set}) + K$$

$$T(\bar{S}) = \max\{0, C - d_j\} + \max\{0, C - p_j - d_k\} + \bar{T}(\text{middle set}) + K$$

We assume that the first theorem is violated; that is, assume

$$p_j \leq p_k, d_j > \max\{d_k, P(B_k) + p_k\}$$

Since $p_j \leq p_k$, the tardiness of jobs in the middle set cannot increase.

The table below illustrates the cases to consider. Since, now we are moving job j right after job k the table has a slightly different form.

Aft. Move \rightarrow Bef Move \downarrow	j =Tardy k =Tardy	j =Tardy k =NotTardy	j =NotTardy k =Tardy	j =NotTardy k =NotTardy
j =Tardy k =Tardy	1	2 X (c)	X (a)	X (a)
j =Tardy k =NotTardy	X (b)	3 X (c)	X (a)(b)	X (a)
j =NotTardy k =Tardy	4	5	6	7
j =NotTardy k =NotTardy	X (b)	8 X (c)	X (b)	0

Table 3.3: Table for tardiness changes after a movement

The X's are again denoting the cases which cannot arise. X (a) cannot occur because if job j is tardy before, it will continue to be tardy; X (b) cannot occur because if job k was not tardy before, then it will continue to be not tardy, and finally X (c) cannot occur due to the condition $d_j > \max\{d_k, P(B_k) + p_k\} \geq d_k$.

The table below summarizes the results found for the remaining cases.

Aft. Move \rightarrow Bef Move \downarrow	j =Tardy k =Tardy	j =Tardy k =NotTardy	j =NotTardy k =Tardy	j =NotTardy k =NotTardy
j =Tardy k =Tardy	(1) $W + 2p_j - C$	X	X	X
j =Tardy k =NotTardy	X	X	X	X
j =NotTardy k =Tardy	(4) $p_j + d_j - C$	(5) ≥ 0	(6) ≥ 0	(7) ≥ 0
j =NotTardy k =NotTardy	X	X	X	0

Table 3.4: Tardiness changes after a movement

We again want to have $\Delta \geq 0$ for all cells so we need to analyze cases 1 and 4. Let A_k be the after set of job k and let $\bar{A}_k = J - A_k$. Then we can say that $C \leq P(\bar{A}_k)$ and $P(\bar{A}_k) \geq W + p_j + p_k$. Suppose $d_j + p_j \geq P(\bar{A}_k)$ then $d_j + p_j \geq P(\bar{A}_k) \geq W + p_j + p_k$ and so $d_j \geq W + p_k \geq W + p_j$. Then j was not tardy before which eliminates cell 1.

For cell 4, since we assume $p_j + d_j \geq P(\bar{A}_k)$ and since $P(\bar{A}_k) \geq C$ this cell is satisfied by the condition imposed.

So for $p_j \leq p_k$ if $d_j > \max\{d_k, P(B_k) + p_k\}$ and $p_j + d_j \geq P(\bar{A}_k)$ then job k is before job j in some optimal sequence, and this is the second theorem of Emmons.

In 1976, Fisher [12] relaxed these theorems, by first removing $j < k$ condition from the third theorem and removing one of the maximands from part (i) of the second theorem. That is (i) can be taken as $d_j > d_k$.

Here it will be better to give the geometric view of Tansel & Sabuncuoğlu [34]. With their point of view theorems of Emmons become really easy to handle. Especially their look at theorem 1 is worth mentioning. Define $E_k = \sum_{i \in B_k} p_i + p_k$ to be the earliest completion time of job i and $L_k = \sum_{i \in \bar{A}_k} p_i$ be the latest completion time of it. Recall that B_k and A_k are the before and after sets of job k and $\bar{A}_k = J - A_k$. Plot the data on a graph with each data point being represented by $(p_k, \max\{d_k, E_k\}) \forall k \in J$. Initially, no relation is known, and $E_k = p_k$. E_k becomes larger when we find relations. Each job k has an *enclosing rectangle* which is defined by the following points as the corners :

$$(0, 0), (p_k, 0), (0, \max\{d_k, E_k\}), (p_k, \max\{d_k, E_k\})$$

A point $(p_j, \max\{E_j, d_j\})$ is said to be in the enclosing rectangle of job k if the point is in the interior or on the boundary of the enclosing rectangle of job k . An equivalent statement of Theorem 1 of Emmons is as follows : With SPT indexing, if the point corresponding to job j is in the enclosing rectangle of k then job j is before job k in some optimal sequence. The figure 3.4 illustrates the idea.

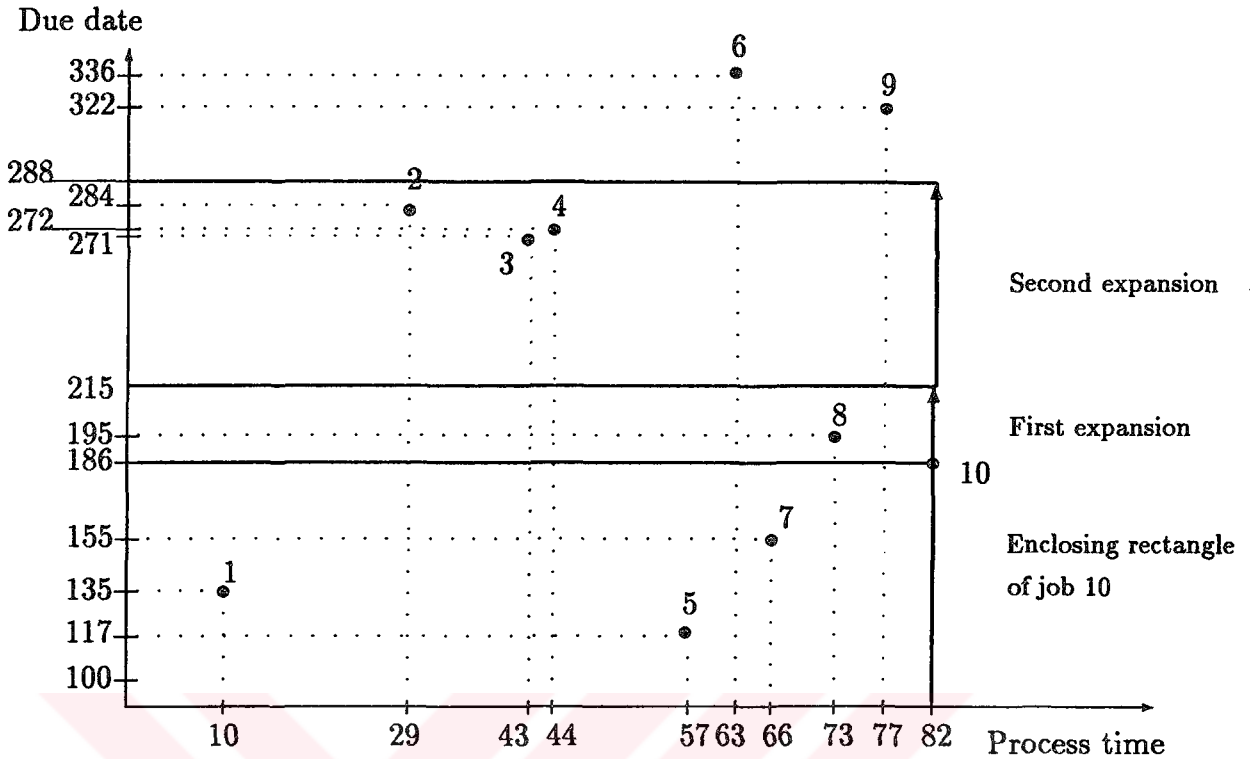


Figure 3.4: Graph for the illustration of enclosing rectangle

For example in Figure 3.4, initially $B_{10} = \emptyset$ and $E_{10} = 82$. Since jobs 1, 5, 7 are in the enclosing rectangle of job 10, B_{10} now expands to $\{1, 5, 7\}$ and E_{10} becomes $82 + 10 + 57 + 66 = 215$. The enclosing rectangle of job 10 expands vertically indicated by the arrow in the figure and now job 8 is also in the rectangle. Now $B_{10} = \{1, 5, 7, 8\}$ and $E_{10} = 215 + 73 = 288$ and jobs 2, 3, 4 fall in the enclosing rectangle of job 10. With these rectangular movements, the relations become very easy to handle.

3.2 β -sequence

With the repeated use of Emmons' theorems, a new sequence, called the β -Sequence, is developed by Tansel & Sabuncuoğlu. The β -Sequence happens to be an optimal sequence under certain conditions. Let us give some definitions

first.

For any job j four different sets are constructed. These are referred to as right-down, left-down, right-up, and left-up sets of job j , denoted by RD_j, LD_j, RU_j, LU_j , respectively. Let $\alpha_i = \max(p_i, d_i) \forall i$.

The sets are defined as

$$RD_j = \{i : i \in J, i > j, \text{ and } \alpha_i < \alpha_j\}$$

$$LD_j = \{i : i \in J, i < j, \text{ and } \alpha_i \leq \alpha_j\}$$

$$RU_j = \{i : i \in J, i > j, \text{ and } \alpha_i \geq \alpha_j\}$$

$$LU_j = \{i : i \in J, i < j, \text{ and } \alpha_i > \alpha_j\}$$

For example, for job 7 in Figure 3.4 we have

$$LD_7 = \{1, 5\}, \quad RD_7 = \emptyset, \quad LU_7 = \{2, 3, 4, 6\}, \quad RU_7 = \{8, 9, 10\}$$

In fact LD_j is the set of jobs in the enclosing rectangle of job j .

The β -Sequence is generated by using the earliest completion times of the jobs resulting from the before-sets. While forming the before-sets the left-down and right-down sets are used. The usage of the left-down set corresponds to the use of Theorem 1 of Emmons in finding the before-sets. That is, for $i \in LD_j$, the relation $i \leftarrow j$ is available from Tansel & Sabuncuoğlu's interpretation of Theorem 1. The use of right-down set corresponds to the use of Theorem 2 of Emmons. For $i \in RD_j$, a job being in RD_j means that $p_j < p_i$ and $\max\{d_j, E_j\} > d_i$ which are the first two requirements of theorem 2 of Emmons. So for any $i \in RD_j$, if $p_j + d_j \geq L_i$ then $i \leftarrow j$ can be concluded.

We form the earliest completion time E_j of job j in two steps after initialisation:

Initial: Assign $\beta_j = \max(p_j, d_j) \forall j$ and form LD_j, RD_j with respect to point $(p_j, \beta_j) \forall j$.

Step 1 - For each newly included k in the most recent LD_j , we increment E_j by p_k while decreasing L_k by p_j , assign $\beta_j = \max\{d_j, E_j\}$ and redefine LD_j

with respect to point (p_j, β_j) .

Step 2 - For each newly included k in the most recent RD_j , if $p_j + \beta_j \geq L_k$ then we increment E_j by p_k while decreasing L_k by p_j , assign $\beta_j = \max\{d_j, E_j\}$ and redefine RD_j with respect to point (p_j, β_j) .

These steps are repeated as many times as possible. Termination occurs when no more incrementation can be done.

The β -Sequence is the sequence of the jobs when we order them in nondecreasing order of their final β values with ties broken by nondecreasing order of process times. Let $D_j(\beta)$ denote the jobs sequenced before job j in the β -sequence and $RD_j(\beta)$ be the most recent right down set of job j at termination of steps 1 and 2.

The β -Theorem of Tansel & Sabuncuoğlu is :

Theorem If $\beta_j \geq P(D_j(\beta)) \forall j$ with $RD_j(\beta) \neq \emptyset$ then the β -Sequence is optimal for the original problem.

If (i) $RD_j(\beta) = \emptyset$ or (ii) $RD_j(\beta) \neq \emptyset$ and $\beta_j \geq P(D_j(\beta))$, then we say job j passes the β -test, otherwise we say job j fails the β -test. Note that failure occurs if and only if $RD_j(\beta) \neq \emptyset$ and $\beta_j < P(D_j(\beta))$. If all jobs pass then the β -Sequence is optimal. Otherwise nothing can be said about the optimum sequence. That is, a failed β -Sequence may or may not be optimal.

Let me give an example here to illustrate the idea.

Ex 1 Suppose we have 7 jobs to schedule. The process time and due dates are shown in the table below. Applying Emmons' theorem to find precedence relations between job pairs we find earliest completion times for the jobs. The resultant early completion times of those 7 jobs are below:

Job No	Process time	DueDate	Final E_i 's	Final β_i 's
1	19	246	19	246
2	26	250	105	250
3	60	246	79	246
4	63	275	168	275
5	64	309	319	319
6	77	328	396	396
7	87	280	255	280

The beta sequence is 1 3 2 4 7 5 6

For job 1 down set total is zero (passes).

For job 3 down set total is $p_1 = 19$ and if $\beta_3 = 246 \geq p_1 = 19$ (passes)

For job 2 $\beta_2 = 250 \geq 19 + 60 = 79$ (passes)

For job 4 $275 \geq 79 + p_2 = 105$ (passes)

For job 7 $280 \geq 105 + p_4 = 168$ (passes)

For job 5 $319 \geq 168 + p_7 = 255$ (passes)

For job 6 $396 \geq 255 + p_5 = 319$ (passes)

So the β -Sequence is optimal.

3.3 Decomposition theorems

The decomposition idea, first developed by Lawler [16], is the next tool that we use in our algorithms. Decomposing means handling each part alone, independent of the other. So the number of jobs in hand at a time decreases. The problem is that the decomposition theorem of Lawler results in many possible alternative decompositions and there is no *a priori* information on which particular decomposition(s) yields an optimal sequence.

The decomposition theorem of Lawler is also applicable to weighted tardiness problem. For the decomposition principle to apply, jobs are assumed to be *agreeably weighted*; that is, if $p_i < p_j$ then $w_i \geq w_j$. For the total

tardiness problem since all $w_i = 1$ this condition passes immediately. Begin with reindexing the jobs in EDD order, i.e. $d_1 \leq d_2 \leq d_3 \leq \dots \leq d_n$ and break ties by nondecreasing process times. Let job k be the largest indexed job with the largest processing time. The decomposition theorem states that there exists an integer δ , $0 \leq \delta \leq n - k$, such that there is an optimal sequence in which k is preceded by all jobs $j : j \leq k + \delta$ and followed by all jobs $j : j > k + \delta$. So using this principle the problem can be decomposed into subproblems without losing from optimality. That is, there is an optimum sequence which is in the following order according to δ :

- i) $1, 2, 3, \dots, k - 1, k + 1, \dots, k + \delta$
- ii) k
- iii) $k + \delta + 1, k + \delta + 2, \dots, n$

In fact, this theorem can be seen from theorem 1 of Emmons if we look at it from the view of Tansel & Sabuncuoglu. Since k is taken to be the largest job, there will be no point in the right side of job k in the graph. The figure below will be helpful in the explanations.

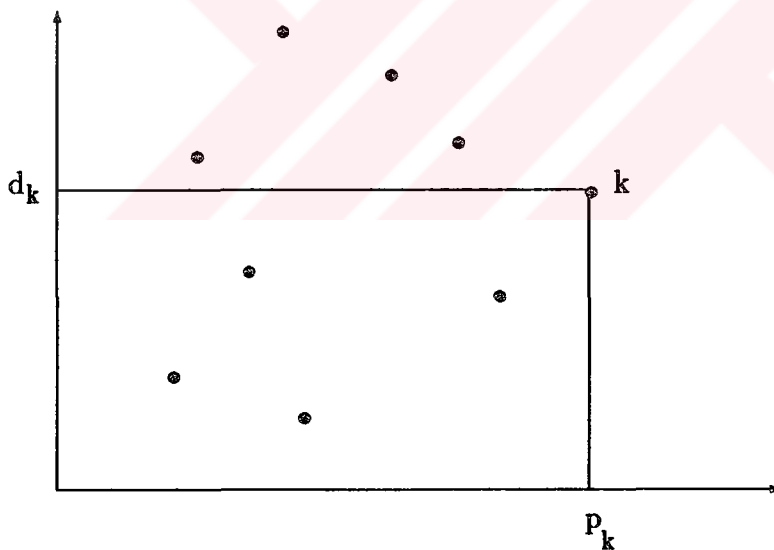


Figure 3.5: Figure for the illustration of decomposition theorem

The jobs in the enclosing rectangle of job k will be sequenced before job k and for the ones above the rectangle, nothing can be said in terms of theorem 1. The decomposition theorem of Lawler uses this idea. The jobs in the enclosing rectangle of job k are the ones which have less due date than that of job k . Since we are using the EDD indexing, then those jobs will be the ones from 1 to $k - 1$. So in some optimal sequence, job k will be sequenced after the jobs which have less due date than itself, regardless of how the jobs in the enclosing rectangle are sequenced. The jobs which have $d_i \leq d_k$ will be surely before job k . The rest of the jobs should be checked in both before k and after k . So there are many possible decompositions, in terms of this decomposition theorem. \dots

Then Potts and Wassenhove [23] worked on the theorem and decreased the search space. In order to understand what is going on let us drive the conditions from scratch. We are trying to find places $k + \delta$ to which moving job k is not profitable. That is, we assume job k is at place $k + \delta$ and then derive conditions for which moving job k forwards or backwards decreases the tardiness. Then we conclude that with those conditions, the place $k + \delta$ will not be an alternative for job k .

We know that $p_k \geq p_i \forall i$ and

$$d_k \leq d_{k+1} \leq d_{k+2} \leq \dots \leq d_{k+\delta-1} \leq d_{k+\delta} \leq \dots \leq d_n.$$

Let us first look at the case of forward movement from place $k + \delta$. Job k is at place $k + \delta$ and we derive the conditions under which moving k from $k + \delta$ will be profitable. The figure 3.6 shows the sequences, S before the movement and the sequence \bar{S} after the movement.

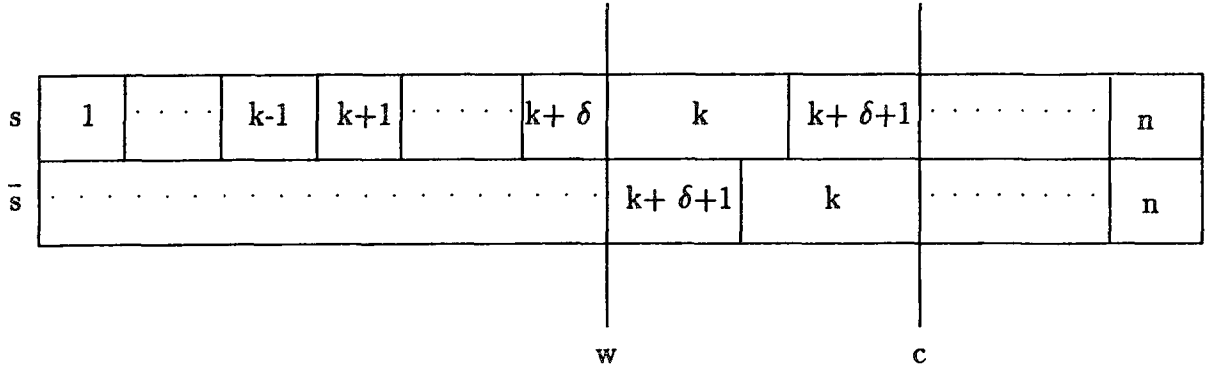


Figure 3.6: Illustration of a forward movement

The table below illustrates the idea. Let $\Delta = T(S) - T(\bar{S})$ and let $j = k + \delta + 1$.

After Move \rightarrow Bef Move \downarrow	k=Tardy j=Tardy	k=Tardy j=NTardy	k=NTardy j=Tardy	k=NTardy j=NTardy
k=Tardy j=Tardy	(1) $p_k - p_j \geq 0$	(2) $W + p_k - d_j$	X	X
k=Tardy j=NTardy	X	(3) < 0	X	X
k=NTardy j=Tardy	X	(5) $d_k - d_j < 0$	X	X
k=NTardy j=NTardy	X	(8) < 0	X	0

Table 3.5: Tardiness changes resulting from the forward movement

We are trying to find conditions for which moving job k is profitable, so we want $\Delta \geq 0$. So we need to eliminate cases 3, 5 and 8 while trying to make cell 2 ≥ 0 . If we impose $W + p_k - d_{k+\delta+1} \geq 0$ then $d_{k+\delta+1} \leq W + p_k$ and so job $k + \delta + 1$ is tardy in the sequence S which eliminates cases 3 and 8 immediately.

For case 5: Since $d_{k+\delta+1} \leq W + p_k \leq d_k$ where the last inequality follows from job k 's being not tardy in S and since $d_k < d_{k+\delta+1}$ this case is also eliminated.

So if

$$d_{k+\delta+1} \leq W + p_k \leq \sum_{i=1}^{i=k+\delta} p_i - p_k + p_k = \sum_{i=1}^{i=k+\delta} p_i \quad \text{for } \delta < n - k$$

then job k will not stay at place $k + \delta$.

Now, let us derive a similar relation from the backwards movement of the job k assumed to stay at place $k + \delta$. The figure below will show the sequence before the move, S , and after the move \bar{S} .

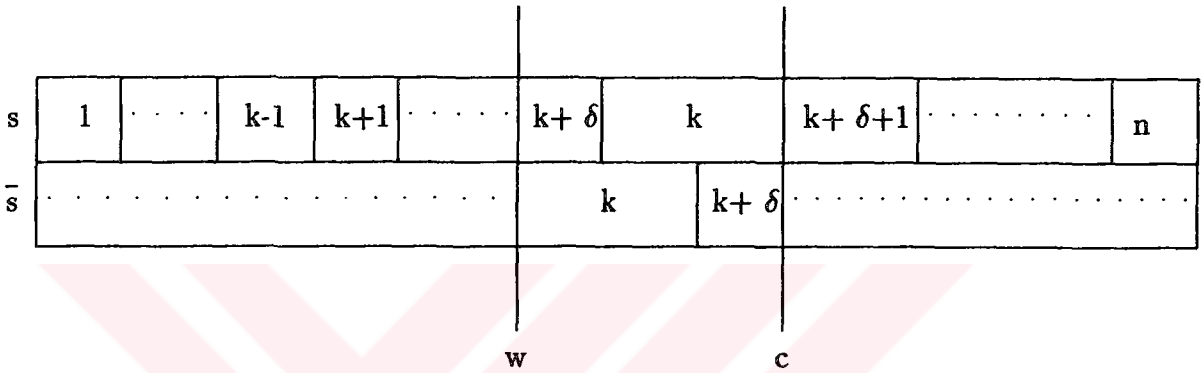


Figure 3.7: Illustration of a backward movement

The table below shows the resultant $\Delta = T(S) - T(\bar{S})$

After Move \rightarrow Bef Move \downarrow	k=Tardy k+delta=Tardy	k=Tardy k+delta=NTardy	k=NTardy k+delta=Tardy	k=NTardy k+delta=NTardy
k=Tardy k+delta=Tardy	(1) $p_{k+\delta} - p_k < 0$	X	(2) X	X
k=Tardy k+delta=NTardy	(3) $d_{k+\delta} - W - p_k$	(4) > 0	(5) $d_{k+\delta} - d_k > 0$	(6) > 0
k=NTardy k+delta=Tardy	X	X	X	X
k=NTardy k+delta=NTardy	X	X	X	0

Table 3.6: Tardiness changes resulting from the backward movement

Since we want the cells to be ≥ 0 we want to eliminate cell 1 and want cell 3 to have $\Delta \geq 0$. Try the condition of cell 3 which is $d_{k+\delta} \geq W + p_k$.

Since

$$d_{k+\delta} \geq W + p_k \geq W + p_{k+\delta}$$

then job $k + \delta$ is not tardy in sequence S which eliminates cell 1 while causing cell 3 to be ≥ 0 . But this time we need to pay attention to the equality case since we moved the larger job backwards. Suppose $d_{k+\delta} = W + p_k$. The corollary 2.3 of Emmons states that

if jobs j and k with $j < k$ are to occur consecutively after a waiting time W then they should be sequenced according to the rule

$$j \leftarrow k \text{ if and only if } d_j \leq \max\{W + p_k, d_k\}.$$

We will see if $d_{k+\delta} = W + p_k$ is feasible for our aim or not by checking this corollary. For our case $k + \delta < k$. If $d_{k+\delta} \leq \max\{W + p_k, d_k\}$ then $k + \delta \leftarrow k$ will be concluded which contradict with our aim.

In cell 3 k is tardy in S and \bar{S} and so $d_k \leq W + p_k$. Then the maximand in the inequality becomes $W + p_k$ and since

$$d_{k+\delta} = W + p_k = \max\{W + p_k, d_k\}$$

then we conclude that $k + \delta \leftarrow k$.

So, the condition needed here is to have

$$d_{k+\delta} > W + p_k = \sum_{i=1}^{i=k+\delta} p_i \text{ for } \delta > 0.$$

If this is so, then job k will not stay at place $k + \delta$.

Finally, if

$$d_{k+\delta} > \sum_{i=1}^{i=k+\delta-1} p_i \text{ for } \delta > 0 \quad (1)$$

or

$$d_{k+\delta+1} \leq \sum_{i=1}^{i=k+\delta} p_i \text{ for } \delta < n - k \quad (2)$$

then job k will not go to place $k + \delta$.

So, if

$$d_{k+\delta} \leq \sum_{i=1}^{i=k+\delta-1} p_i < d_{k+\delta-1} - p_{k+\delta} \text{ for } 0 < \delta < n - k$$

then place $k + \delta$ will be an alternative for job k .

The inequality (2) is not valid for $\delta = n - k$ case since at $\delta = n - k$ the place $k + \delta + 1 = n + 1$ will be meaningless. So for $\delta = n - k$ only the condition (1) will be valid, that is, the job will go to place n if $d_n \leq \sum_{i=1}^{i=n-1} p_i$. For condition (1), $\delta = 0$ is meaningless since we are trying to move job k at place $k + \delta$ backward, with $\delta = 0$ the movement will be meaningless. So for $\delta = 0$, only the condition(2) will be used, that is, the job will stay at its original place if $d_{k+1} > \sum_{i=1}^{i=k} p_i$

Now let us state the decomposition theorem of Potts & Wassenhove in the formal form . First reindex in EDD. For any $k, l \in J$ a problem is said to *decompose with job k in position l* if there exist an optimal sequence in which jobs $1, \dots, k - 1, k + 1, \dots, l$ are sequenced before job k and jobs $l + 1, \dots, n$ are sequenced after job k .

Theorem(Potts & Wassenhove (1982)) The problem decomposes with job k in position l for some l satisfying one of the below conditions:

$$\begin{aligned}
 (i) \quad l = k & \qquad \text{and} \quad \sum_{i=1}^{i=l} p_i < d_{l+1} \\
 (ii) \quad l = k + 1, \dots, n - 1 & \text{ and } d_l \leq \sum_{i=1}^{i=l-1} p_i < d_{l+1} - p_l \\
 (iii) \quad l = n & \qquad \text{and} \quad \sum_{i=1}^{l-1} p_i \geq d_l
 \end{aligned}$$

The theorem says that by moving job k to the l^{th} position we decompose the problem into two sets. The ones in the first $l - 1$ positions forming the before set and the rest forming the after set. Both can be solved independent of the other by taking the ready times for the after set into consideration.

It would be better to show the decomposition theorem on an example here.

Ex 2 Suppose we have 16 jobs to be scheduled. The processing times and the due dates are below:

Job No	Proc. time	DueDate
1	85	241
2	15	246
3	77	292
4	23	325
5	15	385
6	41	390
7	43	417
8	35	418
9	14	432
10	66	432
11	42	440
12	1	456
13	49	475
14	15	490
15	59	506
16	26	519

Since the theorem is based on an EDD schedule we need to have EDD indexed jobs. Our current data satisfies it. The job with the largest process time is job 1. So places between 1 to n will be searched.

$$l = 1 \text{ if } \sum_{i=1}^{i=1} p_i = 85 < d_2 = 246 \rightarrow \text{PASS}$$

$$l = 2 \text{ if } d_2 \leq \sum_{i=1}^{i=1} p_i = 85 \rightarrow \text{FAIL}$$

$$l = 3 \text{ if } d_3 = 292 \leq \sum_{i=1}^{i=2} p_i = 100 \rightarrow \text{FAIL}$$

$$l = 4 \text{ if } d_4 = 325 \leq \sum_{i=1}^{i=3} p_i = 177 \rightarrow \text{FAIL}$$

$$l = 5 \text{ if } d_5 = 385 \leq \sum_{i=1}^{i=4} p_i = 200 \rightarrow \text{FAIL}$$

$l = 6, 7, 8, 9, 10$ and 11 also FAIL

$$l = 12 \text{ if } d_{12} = 417 \leq \sum_{i=1}^{i=11} p_i = 456 \text{ and } 456 < d_{13} - p_{12} = 475 - 1 \rightarrow \text{PASS}$$

$$l = 13 \text{ if } d_{13} = 475 \leq \sum_{i=1}^{i=12} p_i = 457 \rightarrow \text{FAIL}$$

$$l = 14 \text{ if } d_{14} = 490 \leq \sum_{i=1}^{i=13} p_i = 506 \text{ and } 506 < d_{15} - p_{13} = 506 - 49 \rightarrow \text{FAIL}$$

$$l = 15 \text{ if } d_{15} = 506 \leq \sum_{i=1}^{i=14} p_i = 521 \text{ and } 521 < d_{16} - p_{15} = 519 - 59 \rightarrow \text{FAIL}$$

$$l = 16 \text{ if } \sum_{i=1}^{i=15} p_i = 580 \geq d_{16} = 519 \rightarrow \text{PASS}$$

So, the available places for job 1 are place 1, place 12, and place 16. This means there are three possible decompositions for this problem. The first one fixing job 1 to the first place and scheduling the rest, 15 jobs with ready time of 85. The other one is fixing job 1 to the last place and schedule the rest. This time no ready time is needed(i.e ready time is zero). These two decompositions only decrease the number of unscheduled jobs by one. Placing job 1 to place 12 decomposes the problem into two sets. jobs 2 to 12 in the before set and the rest in the after set. The after set, the one with jobs 13, 14, 15, and 16 will have ready times as the total process time of the before set. That is, this time we have two independent sets to schedule, as shown:

$$\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \cup \{13, 14, 15, 16\}$$

A careful analysis of the decomposition theorem shows that the conditions in the theorem turned to be essentially the pass, fail conditions of the β -Sequence.

Recall that the conditions in the decomposition theorem of Potts and Wassenhove were in terms of summation of process times and the due dates of the jobs. We can obtain an equivalent problem by changing the due dates to β values (due to Equivalence Theorem of Tansel & Sabuncuoğlu [34]) for each job. With k being the index of the job with the maximum process time, the conditions of the theorem can be stated in the following equivalent form:

$$(i) \quad l = k \quad \text{and} \quad \sum_{i=1}^l p_i < \beta_{l+1}$$

$$(ii) \quad l \in \{k + 1, k + 2, \dots, n - 1\} \quad \text{and} \quad \beta_l \leq \sum_{i=1}^{l-1} p_i < \beta_{l+1} - p_l$$

$$(iii) \quad l = n \quad \text{and} \quad \sum_{i=1}^{l-1} p_i \geq \beta_l$$

These conditions are similar to the pass / fail conditions of the β -test. The first condition says that if the β value of the largest process timed job

is greater than its down set total, the position it stays in the β -Sequence is a candidate for decomposition which is the same as the β -test pass condition with strict inequality for that job in the β -Sequence. The same logic also applies to the job in the last position. If the last job in the β -Sequence fails to pass the β -test then the job with the largest process time can go to the last place. Condition (ii) needs some more attention. The left inequality ($\beta_l \leq \sum_{i=1}^{l-1} p_i$) is the same as the β -test either failing ($\beta_l < \sum_{i=1}^{l-1} p_i$) or passing in the form of equality ($\beta_l = \sum_{i=1}^{l-1} p_i$). Similarly, the right inequality ($\sum_{i=1}^l p_i < \beta_{l+1}$) is the same as β -test passing with strict inequality.

It follows that an equivalent statement of Potts & Wassenhove decomposition theorem is as follows :

Decomposition Theorem(Potts & Wassenhove(1982)) Assume SPT indexing and also assume job n is in position k in the β -Sequence. The problem decomposes with job n in position l for some l satisfying one of the below conditions:

- (i) $l = k$ and $(k + 1)^{st}$ job in the β -Sequence passes with strict inequality
- (ii) $l \in \{k + 1, \dots, n - 1\}$ and $(l)^{th}$ job either fails the β -test or passes with equality while the $(l + 1)^{st}$ passes with strict inequality
- (iii) $l = n$ and $(n)^{th}$ job either fails or passes with equality

Let us look at the same example (ex.2) to illustrate the decomposition theorem.

Ex 3 The data with the β values are in the following table :

Job No	Proc. time	DueDate	Early comp time	β value
1	85	241	85	241
2	15	246	15	246
3	77	292	92	292
4	23	325	38	325
5	15	385	53	385
6	41	390	94	390
7	43	417	137	417
8	35	418	172	418
9	14	432	186	432
10	66	432	252	432
11	42	440	228	440
12	1	456	229	456
13	49	475	278	475
14	15	490	244	490
15	59	506	352	506
16	26	519	270	519

The β -Sequence happens to be the same as the EDD sequence. The pass / fail checks are below

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P	P	P	P	P	P	P	P	P	P	P	E	P	F	F	F

where P denotes a pass with strict inequality, F denotes a fail and E denotes a pass with equality.

So from here we find the same places for job 1. Since the first check in the β -test is a pass, the first place is an alternative for job 1. The last check in the β -test is a fail, so the last place is another alternative for job 1. A change from E to P occurs at place 12, so place 12 is also an alternative for job 1. We find the same places as in the original type of decomposition, but you see that this way is much easier.

The problem of the existing decomposition theorems is that they may (and usually do) find many possible places for the job in consideration, which calls for branching out on each possibility to arrive at an optimal solution. If a more powerful decomposition can be found, it would help a lot in solving large problems. Then comes the decomposition theorem of Tansel & Sabuncuoğlu [34]. Their decomposition theorem finds an exact place and applies to any job (not to the largest job only), but this time it is not guaranteed that the problem decomposes all the time. That is, the theorem may not apply for some cases, but it gives exact positions for the cases that it applies.

This decomposition theorem is mainly based on the left-down, right-down, left-up and right-up sets defined before. Recall that L_i denotes the latest completion time of job i and $L_i \leq p^* - P(RU_i) \forall i$ since job i precedes all the jobs in RU_i as it is in the enclosing rectangle of the jobs in RU_i . L_i 's are formed during the computation of the β values.

According to the definitions of the four sets, the decomposition theorem of Tansel & Sabuncuoğlu is :

Let $q \in J$.

if 1) either $RD_q = \emptyset$ or $p_q + \beta_q \geq L_i \quad \forall i \in RD_q$

and 2) either $LU_q = \emptyset$ or $p_j + \beta_j \geq L_q \quad \forall j \in LU_q$

then there is an optimal sequence such that all jobs in the down set of job q precede q , and all jobs in the up set of job q succeed job q with job q 's position fixed at place $|D_q| + 1$. That is the problem decomposes in the form (D_q, q, U_q) where D_q is the union of $LD(q)$ and $RD(q)$ and U_q is the union of $LU(q)$ and $RU(q)$. The subproblem consisting of jobs in D_q can be solved independent of the one consisting of jobs in U_q . Only the U_q will have a positive ready time, which is equal to total processing times of the jobs in the down set and the job q .

An example will be helpful in understanding the decomposition theorem.

Ex 4 Consider the data given in figure 3.4. For $q = 7$ right-down set is empty as seen from the graph. So the first condition is satisfied. For this job, the left-up set is not empty. So we need to check the second condition for all the jobs in the left-up set. The left-up set is $\{2, 3, 4, 6\}$. The condition to check is if $p_i + \beta_i \geq L_7 \forall i \in LU_7$. And we also now that $L_7 \leq p^* - P(RU_q)$ The right up set is composed of the jobs $\{8, 9, 10\}$ so the process times for the right up set total is 232. So it will be okay if $p_i + \beta_i \geq 544 - 232 = 312 \forall i \in LU_7$ then the problem will decompose with job 7.

$$p_i + d_i \geq 544 - 232 = 312 \forall j \in LU_7$$

$$j = 2 \quad p_2 + d_2 = 29 + 284 = 313 \geq 312 \quad \text{PASS}$$

$$j = 3 \quad p_3 + d_3 = 43 + 271 = 314 \geq 312 \quad \text{PASS}$$

$$j = 4 \quad p_4 + d_4 = 44 + 272 = 316 \geq 312 \quad \text{PASS}$$

$$j = 6 \quad p_6 + d_6 = 63 + 336 = 399 \geq 312 \quad \text{PASS}$$

So the problem decomposes with job 7 at position 3 since there are two jobs in the down set. The optimal sequence will be in the form

$$\{1, 5\}7\{2, 3, 4, 6, 8, 9, 10\}$$

The above theorem decomposes the problem by giving exact places for the cases it works. Of course there may be many cases for which the theorem does not apply. The example below shows such an instance.

Ex 5 Let us look at the same example again. If we add one more job to the data with $p_{11} = 94$ and $d_{11} = 101$ the graph will be

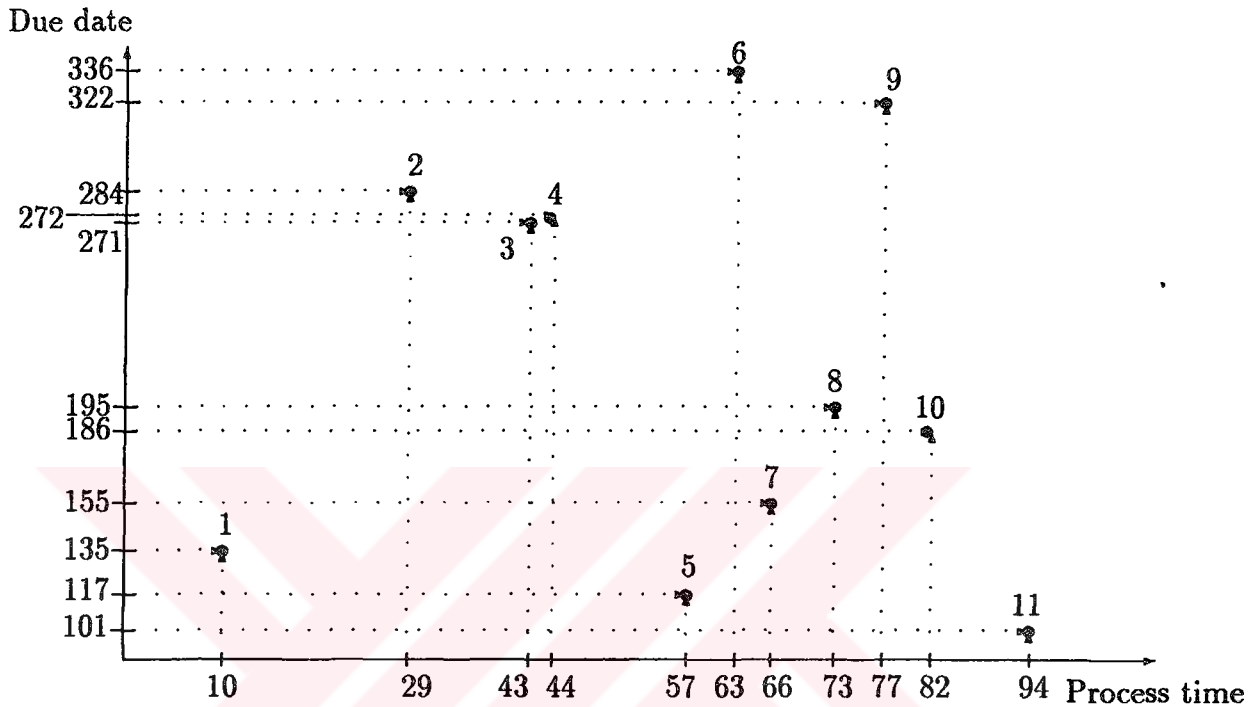


Figure 3.8: Data plot of example 5

It can be seen that no job satisfies $p_i + \beta_i \geq L_{11}$, so we need to consider only the jobs with right-down set empty. Only job 11 satisfies that condition. That is, only the new job has its right down-set empty for the decomposition condition. Since the left-up set for that job is not empty we should check if for all the jobs in the left-up set $p_i + \beta_i \geq L_{11}$ is satisfied. But it is not true for any of the jobs. So, the problem does not decompose for any of the jobs. Adding only one job changed everything about the problem structure.

But still the decomposition theorem is powerful since once it applies, it finds an exact decomposition.

Chapter 4

NEW EXACT ALGORITHMS

As I have mentioned before, existing optimizing algorithms for the single machine total tardiness problem is restricted to 100 jobs. We wanted to enlarge the size to more than 100 by using the concept of β -Sequence and the TS(Tansel and Sabuncuoğlu) decomposition theorem of [34] . We designed 3 different algorithms, each one being built on the previous. The first section describes our first algorithm which is based on the usage of β -Sequence and the TS-decomposition. In the second section, we give an algorithm which is based on the PW(Potts & Wassenhove) decomposition. Finally, we have a hybrid algorithm which we give in the third section. These are followed by some computational results in the last section.

4.1 Algorithm Beta(TS)

The first algorithm that we developed uses the β -Sequence together with the TS-decomposition in a branch and bound algorithm. We need the branching part since both the decomposition and β -test may fail for a given instance in which case we need some other method to continue. We first apply the β -test and if it fails then we try to decompose the problem. If we can, then we handle each part separately, but if the decomposition fails we need to branch

according to some criterion. This algorithm is called Beta(TS). The flow chart of the algorithm is in figure 4.1.

Let me explain each step in detail.

- **Start** : At the beginning we first preprocess the data, where possible, to decrease the number of jobs under consideration. That is, we try to select a job for which we can say that there exists an optimal schedule in which that job is first or last among the ones in hand. If we can find such jobs the problem size will decrease. For the preprocessing we have two basic approaches.

The first one is named **Upsearch**. Here we try to select jobs which will be last in some optimal schedule. Once we find one we take it out (i.e. assign it to the last position) and solve the problem for the remaining jobs. This preprocess is in fact, based on the use of Emmons' theorems but in a different manner. We identify the largest due dated job, say job k , (so that its up-set is empty). Recall that we define $p^* = \sum_{i=1}^n p_i$. If job k satisfies $p_k + d_k \geq p^*$, then there exists an optimal schedule in which job k is last.

To see why, let $d_k = \max_{i \in J} d_i$

- For $k < j$, we look at the second theorem of Emmons.

Since

$$\max(E_k, d_k) \geq d_k \geq d_j \text{ then } \max(E_k, d_k) \geq d_j$$

and

$$p_k + d_k \geq p^* \geq L_j$$

then $j \leftarrow k$.

-For $j < k$, the first theorem of Emmons works. Since

$\max(E_k, d_k) \geq d_k \geq d_j$, job j is in the enclosing rectangle of job k . This implies $j \leftarrow k$.

So, if the largest due dated job satisfies $p_k + d_k \geq p^*$ then that job is scheduled for the last position in hand without loss of optimality.

If the largest due dated job also has the largest process time then again there exist an optimal schedule in which that job is last.

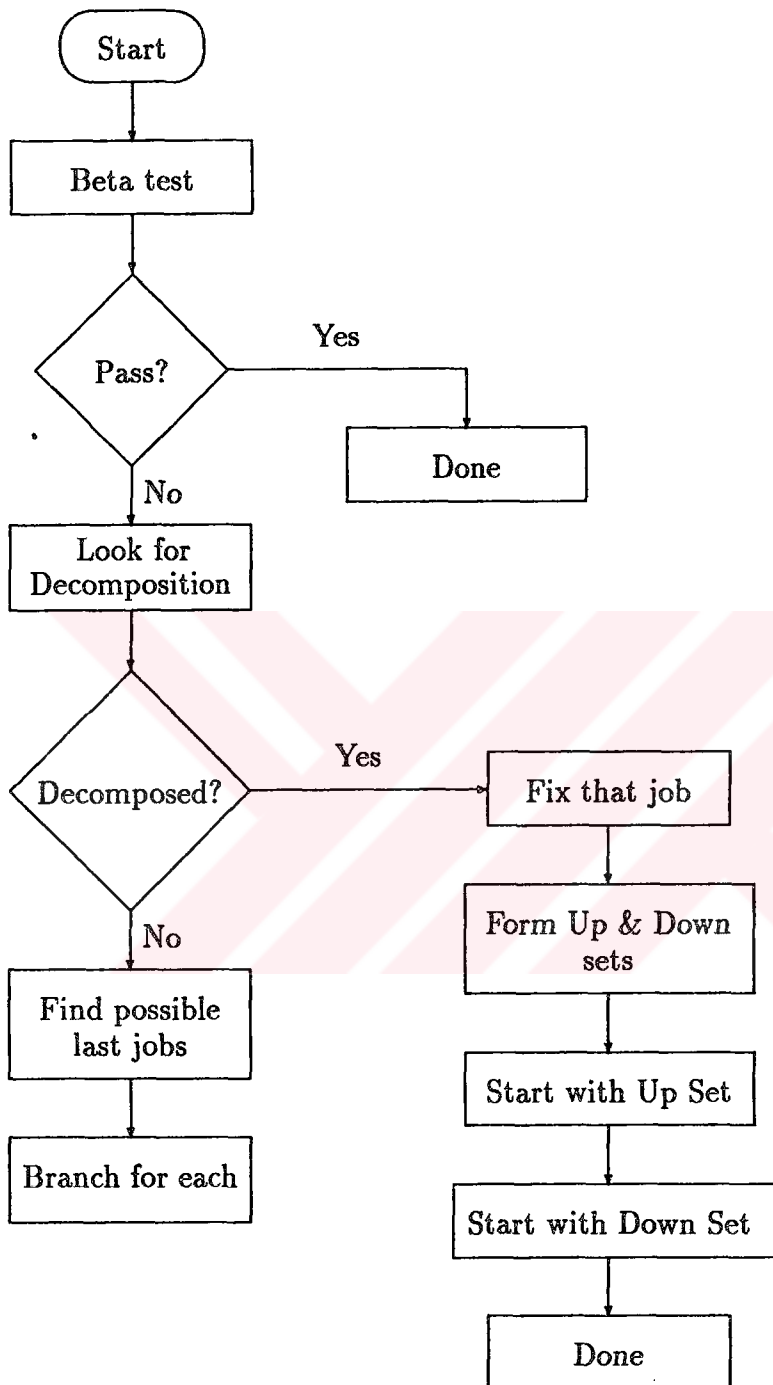


Figure 4.1: Flow chart for the algorithm Beta(TS)

The second approach for preprocessing is called the **Downsearch**. This time we try to identify a job which will be the first in some optimal sequence. This approach is based on the application of theorem 1 of Emmons. If the smallest due dated job also has the smallest processing time it will be the first one in some optimal schedule. So schedule that job to the first place and take its process time to be the ready time for the rest of the jobs. This positive ready time can be transformed to a ready time of zero by decreasing the due dates of the rest of the jobs by the ready time. Applying this method repeatedly with the remaining set each time, the problem size is decreased as much as possible.

After the preprocessing, the problem size is possibly reduced but we now have a set of unscheduled jobs and we cannot fix any of them to the first or last locations at the first sight. Then the algorithm begins with these jobs. In fact it is a recursive algorithm. That is, for each resulting subset of jobs during the algorithm, the whole algorithm is called for that set.

β -test : The first thing is to search if the optimal sequence of the jobs in hand can be found by the β -Sequence. If the β -Sequence passes the β -test, then the optimum is found for this subset of jobs. Otherwise, we could not find the optimal sequence of the set in hand immediately, then we look for a job for which TS-decomposition works.

TS-Decomposition : For all of the jobs in hand the decomposition theorem of Tansel and Sabuncuoğlu is checked. If a decomposition can be found then form the before and after sets according to the decomposition and now the whole recursive algorithm will be applied independently for the smaller sets, both for the after-set and the before-set, with the after-set having a positive ready time which is equal to the total process time of the jobs in the before-set plus the process time of the decomposing job. To incorporate the ready time, we have to initialize the β -Test accordingly. That is, the earliest possible completion time of each job is initialized to the ready time plus the process time of that job. Alternately, we may decrease the due dates of the jobs in the after set by the common ready time and then take the ready time to be zero.

If the decomposition also fails then we need to go by branching for the set in our hand. After some branches the β -test or decomposition may apply but they will be conditional on the branch of this stage.

Branch : Since both of the tools failed, we need to branch for the set in hand . Now some kind of a branching criterion is needed. During the β -test, earliest and latest completion times of each job are found. Branching will be done according to these times. The branching criterion is also important. We first defined the branching rule as that of branching from the last position. That is, the possible last jobs are identified and one branching occurred with each possible last job. Those jobs are the ones whose after sets are found to be null. One reason for that type of branching is that many branch and bound algorithms existing in the literature are based on the same idea (backwards scheduling). Later, we also analyzed other type of branching criteria like forward branching. For each branch the recursive algorithm begins from the beginning with the new set of unscheduled jobs, conditioning on one of the jobs being last.

The algorithm continues until all branches are evaluated. A detailed example may be helpful in here.

Ex 6 Suppose we have 12 jobs to schedule with the below data :

Job No	Proc. time	Due date
1	10	145
2	29	294
3	43	281
4	44	282
5	57	127
6	63	346
7	66	165
8	73	205
9	77	332
10	82	196
11	10	101
12	90	350

In the preprocess part, since the job with the largest process time, job 12, also has the largest due date, job 12 will come last in some optimal schedule. So we just take it away from the unscheduled set and we fix it to the last position.

Again in the preprocess part, since the job with the smallest process time, job 11, also has the smallest due date, job 11 comes first in some optimal schedule. So that job is taken away and scheduled to the first position. Since it is fixed to the first position we need to take this into account for the rest of the jobs. We can either take it as a ready time or we can decrease the due dates of the rest of the jobs by the process time of job 1. For this problem we decreased the due dates.

So we have 10 jobs left and the due dates are decreased by $p_{11} = 10$ units. With the 10 jobs in hand we check if the β -test gives optimum, but it fails. Let us observe the earliest completion times and β values of each job in a table below. Recall that $\beta_i = \max(d_i, E_i) \forall i$

Job No	Early pos. comp. time	Due date	β value
1	67	135	135
2	205	284	284
3	176	271	271
4	220	272	272
5	57	117	117
6	385	336	385
7	133	155	155
8	206	195	206
9	462	322	462
10	544	186	544

The β -Sequence is 5, 1, 7, 8, 3, 4, 2, 6, 9, 10

Jobs 5 and 1 automatically pass the β -test since their right-down sets are empty. Jobs 7, 8, 3, and 4 also pass the β -test. Job 2 fails. Since we find at least one job failing, β -Sequence may or may not be optimum.

Then we look at the decomposition and try to decompose if possible. This problem is the same as the example given for decomposition, ex 3. So we know that the problem decomposes with job 7 in position 3. Then we have two subproblems now. One problem has only two jobs {1, 5} and the other problem has 7 jobs {2, 3, 4, 6, 8, 9, 10} For the two job case we can easily find the optimum. And it happens to be : (5, 1). For the second problem we apply the algorithm once more. We again try the β -test first. The resultant early times and β values are in the table below. Remember that since we are in the after set now, we need to account for the process times of the jobs in the before set, either by taking their total process time as a ready time or by decreasing the due dates by the total process time. Here we decrease the due dates by 133 which is the total process time of the jobs in the before set.

Job No	Early comp. time	Due date	β value
2	72	151	151
3	43	138	138
4	87	139	139
6	252	203	252
8	73	62	73
9	329	189	329
10	411	53	411

The β -Sequence is (8, 3, 4, 2, 6, 9, 10). Jobs 8, 3, and 4 passes the β -test but job 2 fails. So we try decomposition again. Since graphs help in detecting decomposition let us draw it with respect to the β values.

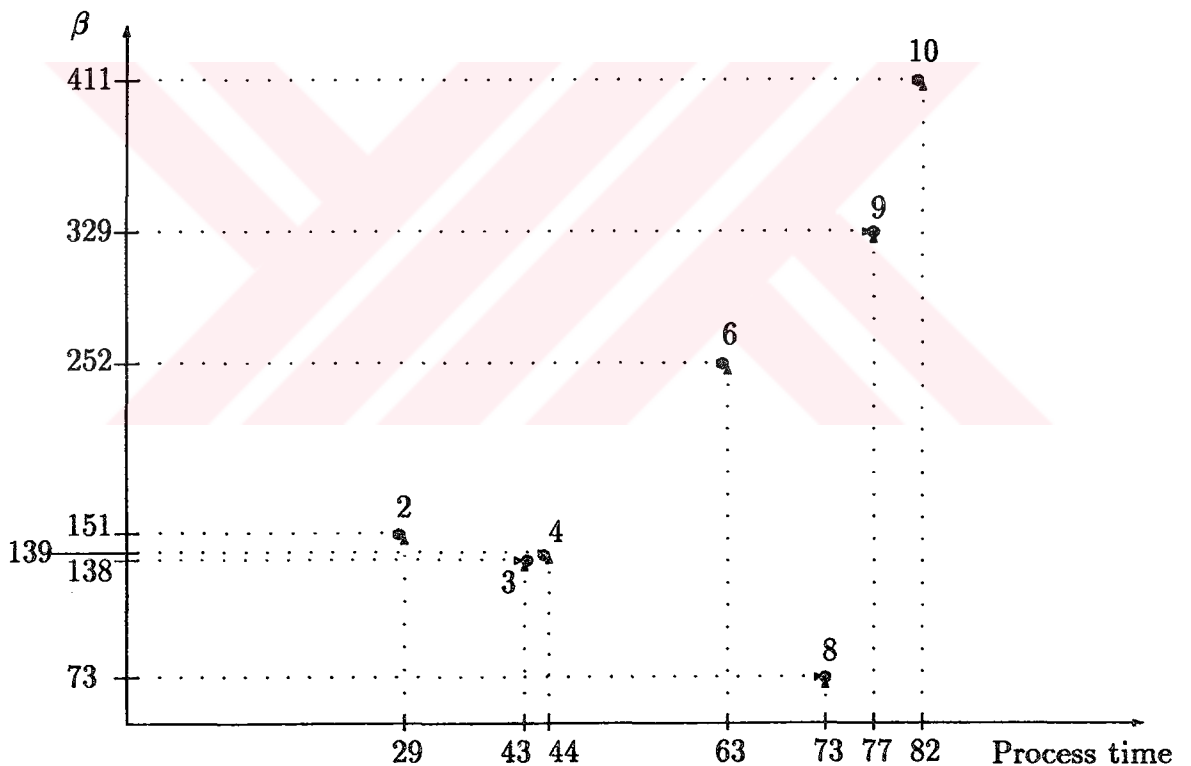


Figure 4.2: Data plot for example 6

From the graph it is seen that the right-down and left-up set of job 9 is empty. So the problem decomposes with job 9 at position 5, with job 10

in the after set and the others in the before set. After the decomposition at job 9, the before set will be solved. Now the p^* decreases to 252 so the job set $\{2, 3, 4, 6, 8\}$ also decomposes with job 6 in the last position (because its left up set is empty and $p_6 + d_6 \geq p^* \geq L_i \forall i$). So the problem is now

$$5 \ 1 \ 7 \ \{2, 3, 4, 8\} \ 6 \ 9 \ 10$$

Now, we are left with 4 jobs to schedule. But for these jobs both the β -Test and decomposition fail. So we need to branch now. According to the earliest and latest completion times given in the table below the branches are formed.

Job no	Early comp time	Latest comp time
2	72	189
3	43	116
4	87	189
8	73	189

It is seen that jobs 2, 4, and 8 are available for the last place because their latest possible completion times are equal to p^* of this set which means no job is found to follow those jobs. So we have to continue in three branches with each branch corresponding to fixing exactly one of $\{2, 4, 8\}$ to the forth position. In each branch, the problem with the remaining three jobs is solved. Finally, we select the solution which gives the minimum tardiness for these 4 jobs. The selected sequence will be the optimum schedule for job population $\{2, 3, 4, 8\}$.

In Beta(TS) we had two different approaches for handling the problem, so we expected to solve larger problems. But the results were not as good as we expected. For some problem instances, the β -Sequence gave the optimum at the very beginning, but for many of the problems, at least a core set is left for which the β -test and decomposition both failed. Then branching is needed which blows up the time of solving. In fact, that is the reason why the literature has at most 100 jobs solved. We first tried to improve the branching criterion. As I mentioned before, we were applying backwards branching which is fixing a job to the last place and continuing with the restricted problem. For a second approach we tried forward branching. That is, we tried to select possible first

jobs among the ones in hand. Those jobs are the ones whose before sets are empty. This type of branching also did not help. In fact it was not clear if one is an improvement over the other. As a final branching criterion we tried both backwards and forwards scheduling at the same time. That is we select the possible last and possible first jobs and then we branch with the one which has less number of alternatives. At the first sight this idea seemed to be promising but it was not. In fact if the problem has some core part, no matter what the branching criterion is, it would take a large amount of time to solve it due to many branches that occur. The solution time of the algorithm depends heavily on the number of branches required to solve it.

Then we looked at another feature to make the algorithm work faster. Since the bottleneck part is the branching, while analyzing the branches, we saw that many problem instances may come into sight more than one time. This occurred when the same jobs happen to be available for the same places in different branches. Suppose we try to schedule 5 jobs and we need to branch for these five jobs. Let us say that jobs 2, 3, 4 are available for the last place. As one alternative take the first available, job 2, put it to last place and continue. Say we need to branch again. That is both β -Test and decomposition failed again. Suppose the available jobs for the last place are now 3 and 4. The available jobs for the last place when 3 is put to last position are {2, 5}. The tree below illustrates the idea:

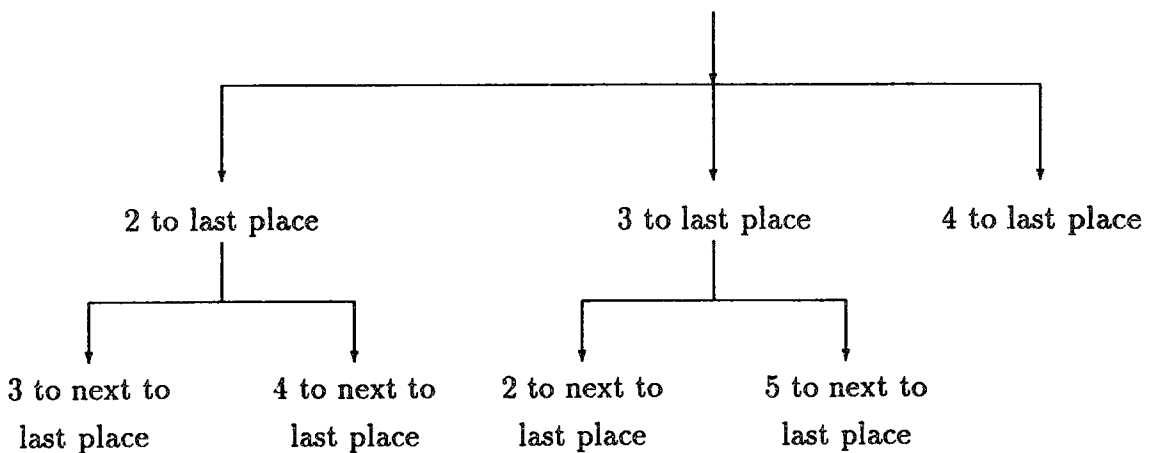


Figure 4.3: Tree structure

So the jobs left to schedule for the first branch, the one fixing 2 to the last place and 3 to the one before last, is the same as the ones for the third branch, which is the one fixing job 3 to the last place and 2 to the one before last. This means we waste time by solving the same problem from scratch each time we encounter it. So we identify those sets and avoid solving them more than once. That is, when we enter the recursive part, before going any further, we first search if the set in hand was solved before or not. The ready time for the set is also important. If we find the same set, the ready time should also be checked to match with that of the one that has already been solved. Otherwise the problems will not be identical. This idea really helped a lot. This way, the solution time of the algorithm Beta(TS) decreased substantially. But it was still not possible to go further than 100 jobs and even those 100-job cases were solved in higher time than the algorithm of Potts and Wassenhove.

4.2 The Algorithm Beta(PW)

Then we have another algorithm which combines the β -Sequence approach with the decomposition idea of Potts and Wassenhove, and we call it Beta(PW). That is the problem is decomposed using the decomposition theorem of Potts & Wassenhove until the β -test passed for the subproblems in hand. This algorithm is again a branch and bound one since the decomposition used here is of a branching type. The flow chart is given on the next page.

The main steps of the algorithm are explained next.

We again have a Preprocess part which is completely the same as that of Beta(TS); try to decrease the number of jobs to schedule, where possible.

The β -test is also the same except that this time the needed strict passes (P), equality passes (E) and fails (F) are also formed during the β -test. This allows an easy identification of positions to which the largest job can be assigned.

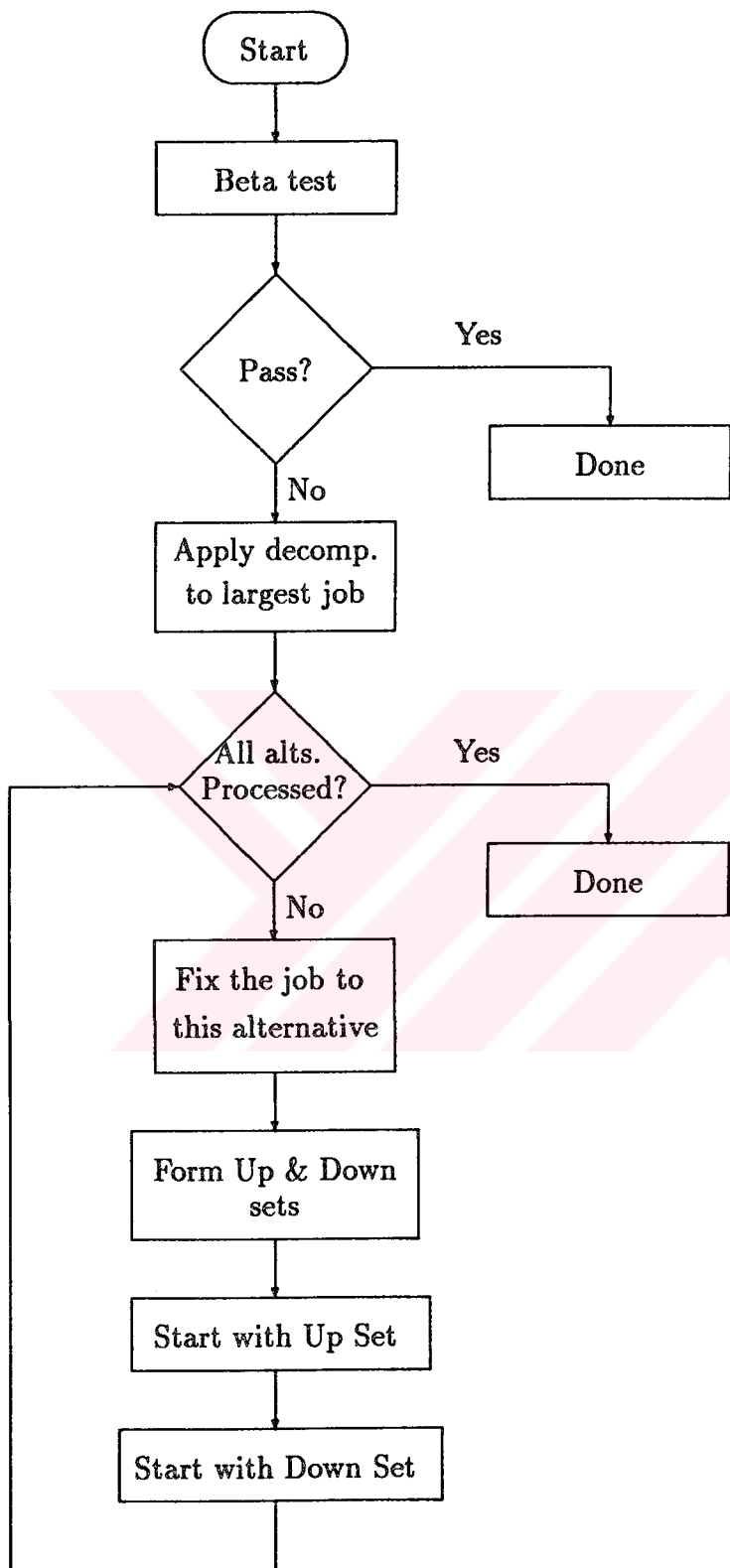


Figure 4.4: Flow chart for the algorithm Beta(PW)

If the β -Sequence is optimal, that is, if it passed the β -test, then for the subset in hand we know the optimal solution and so we continue with the father branch. Otherwise, if we do not know the optimal solution for the subset in hand, we apply the decomposition of Potts & Wassenhove.

PW-Decomposition: The β -Sequence is not optimum. But we have the markers P, E, F for identifying the possible decompositions according to the equivalent statement of Potts & Wassenhove's decomposition given at the end of section 3.3. For the largest processing timed job in hand each possible place defines a branch. Then we go back, for each of the branches, to solve the subproblems (corresponding to before and after sets) resulting from the particular alternative place of the longest job.

This algorithm is completely different from that of Potts and Wassenhove. The only similar part is the decomposition theorem but we modified that theorem (find the alternatives from β -test). With this way of handling, we began to solve the problems faster and we began to handle more than 100 jobs, but the time to solve was not good enough for us.

4.3 Algorithm Beta(TS, PW)

We then decided to merge these two algorithms. That is the branching criterion of the first algorithm is completely changed. When both β -test and TS-decomposition fail, instead of normally branching backwards or forwards, the branching is applied with decomposition of Potts & Wassenhove. Since PW-decomposition surely finds places for the largest job in hand, the algorithm continues. The flow chart is on the next page.

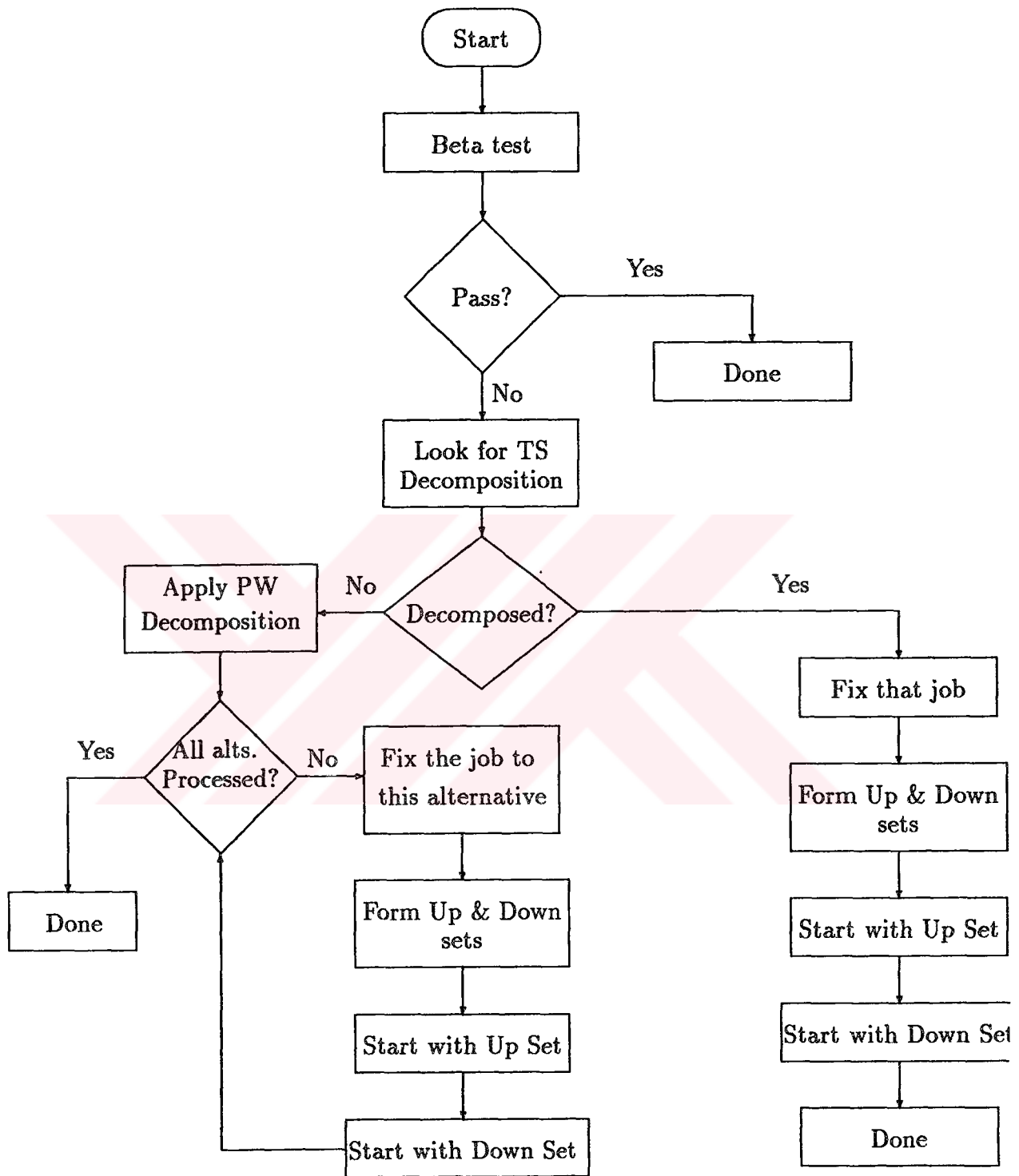


Figure 4.5: Flow chart for the algorithm Beta(TS, PW)

Let me give the basic steps of the merged algorithm.

The first steps are the same as that of Beta(TS). The only change is in the last steps which is the branching case in the first algorithm. This time, instead of branching, we apply the equivalent version of Potts and Wassenhove's decomposition theorem. If for the subset of unscheduled jobs in hand, both β -test and TS-decomposition fail, then we apply the PW-decomposition. There, the job with the largest processing time among the ones in hand is selected and the possible places for that job is found from the β -Sequence and the corresponding strict passes, equality passes and fails. With each of these possible places, the problem decomposes into different subproblems. Then, with branching we analyze all possible decompositions and select the one giving the minimum tardiness at the end.

Merging those two algorithms really helped a lot. This time the program became capable of solving larger problem sizes and in less time.

When we compare these two algorithms, Beta(TS, PW) and Beta(PW), we saw that Beta(TS, PW) solved on the average, three times faster than Beta(PW) and Beta(PW) is better than the original algorithm of Potts & Wassenhove [23] both in computation time and in the maximum number of jobs that can be handled.

When we were making experiments for the 100 and 200-job cases, we saw that the number of branches is still large and the computation time depends heavily on the number of branches required to solve the problem. So we decided to incorporate a lower bound idea. With that way we may be able to decrease the number of branches. Upto this point the only fathom criterion was to find optimum from the β -test. So we were searching all the branches to the lowest level. Observe that we may rewrite the tardiness problem in the following equivalent form:

$$\begin{aligned}
& \min_{S \in \mathcal{S}} \sum_{i \in J} T_i(S) \\
& \text{s.t.} \\
& T_i(S) \geq C_i(S) - d_i \quad \forall i \in J \\
& T_i(S) \geq 0 \quad \forall i \in J
\end{aligned}$$

If we relax the last constraint, the problem turns out to be

$$\begin{aligned}
& \min_{S \in \mathcal{S}} \sum_i T_i(S) \\
& \text{s.t.} \\
& T_i(S) \geq C_i(S) - d_i \quad \forall i \in J
\end{aligned}$$

which is in fact

$$\min_{S \in \mathcal{S}} \sum_i (C_i(S) - d_i) \iff \min_{S \in \mathcal{S}} \sum_i C_i(S) - \sum_i d_i.$$

Since due dates are constant, this turned out to be the minimization of the completion times which is solved by SPT sequence. Because of the relaxation, the value of the objective function is the lateness of the SPT schedule. Let S^* be the optimizing sequence of the original tardiness problem and let $L(S)$ denote the lateness of sequence S whereas $T(S)$ denote the tardiness of it. Then we can say that :

$$L(SPT) \leq L(S^*) \leq T(S^*)$$

where the first inequality follows from the fact that SPT is optimum for the lateness problem. The second inequality is just the lateness being no greater than the tardiness. So the lateness of SPT schedule will give a lower bound for our problem. At any branch we calculate the lateness value of the SPT schedule for the jobs in hand and add it to the calculated tardiness of that branch up to that time. That value will give the lower bound for the branch in hand. If that lower bound is greater than the incumbent solution we fathom that branch and go back to its father to continue. This way, we added another fathom criterion.

For another lower bound we used the β -Sequence approach. Suppose, at some instance we need to branch. We first check the lower bound before going any further and if the bound is smaller than the best known value we

will continue. We have the β -Sequence in hand so a lower bound from here will not cause much time per branch. For the β -Sequence if we increased the due dates of the failed jobs till their completion times in the β -Sequence (i.e. assign $d_j^{new} = \max\{\beta_j, P(D_j(\beta))\}$ for all j for which $RD_j(\beta) \neq \emptyset$ and assign $d_j^{new} = \beta_j$ for all remaining j) then the β -Sequence in hand will pass the β -test for the new due dates. Since the β -test passes, the β -Sequence is optimal for the perturbed problem. Let S^* be an optimal sequence for the original problem and let $T(S)$ be the tardiness of the sequence S . We have

$$T(\beta\text{-Sequence with new } d_i) = T(S^* \text{ with new } d_i) \leq T(S^* \text{ with old } d_i).$$

The equality is from the fact that β -Sequence and S^* are both optimal for the problem with new due dates. The inequality follows from the fact that the new due dates are greater than or equal to the old due dates. So the tardiness of the β -Sequence with new due dates becomes a lower bound for the original problem. The value is easily determined. It is the total tardiness of the jobs which passed the β -test for the original due dates.

So we have two different ways for lower bounding. The bound coming from the β -Sequence happens to be larger than that of the SPT schedule most of the time. We use the larger of these in our algorithm. That is, we calculate both and select the larger one as the lower bound. Then at any branch, the value of the lower bound calculated for that node with the tardiness incurred up to that node makes up the total lower bound for that node, and if that lower bound is greater than the incumbent solution, we fathom that node.

With the help of the lower bound we decreased the time to solve the problems. So we got the final algorithm, which we still call Beta(TS, PW). And with Beta(TS, PW), we manage to handle problems with 500 jobs, even though we cannot solve their hardest cases. We will give the computational results of Beta(TS, PW) in the next section.

4.4 Computational Results For Beta(TS, PW)

For the computational study, we wanted to use the traditional approach proposed by Fisher [12] which tests an algorithm with different types of instances, corresponding to hard or easy cases. Instances of varying degrees of difficulty are generated by means of two factors : the tardiness factor and range of due dates. For each problem, first the process times are generated from uniform density with parameters (1, 100). Once the process times for the problem have been generated, then $p^* = \sum_{i=1}^n p_i$ is computed. The due dates are computed from a uniform distribution which depends on p^* and two parameters; R (due date range) and T (tardiness factor). The due date distribution is uniform over

$$[p^*(1 - T - R/2), p^*(1 - T + R/2)].$$

Both the tardiness factor T and the range of due dates R is selected from the set $\{0.2, 0.4, 0.6, 0.8\}$. So they give $4 * 4 = 16$ combinations for each problem size. We took 10 different runs for each set giving a total of 160 instances for each choice of n . n is taken to be 100, 200, 300, 400 and 500.

The best known exact algorithm in the literature is the algorithm of Potts & Wassenhove [23, 25]. That algorithm solves 100-job problems on the average in 27.07 CPU secs and does not handle problems of size more than 100. The algorithm is based on their decomposition theorem and the dynamic programming algorithm of Schrage & Baker [29]. For the subproblems resulting from the decomposition, the authors use the Schrage & Baker dynamic programming approach which can only solve the problem in consideration if the total sum of labels needed is less than 48000. (The authors give an algorithm for uniquely labeling each job according to the precedence relations. These labels are used in identifying the subsets to be evaluated during the dynamic programming). The algorithm of Potts & Wassenhove first decomposes the problem, if the number of jobs to be solved in the subproblems is less than 30 (this time number of labels will be less than 48000) then they apply the dynamic programming algorithm of Schrage & Baker and solve the subproblem. If the number of jobs is greater than 30, the authors reapply the labeling

algorithm after once applying precedence relations giving theorems for the subset in hand. If the sum of labels is less than the desired number they apply dynamic programming method, otherwise they decompose the problem again. This procedure is repeated until the full set is solved.

In this algorithm, the fathoming criterion that the authors used is only the criterion of dynamic programming. That is, they decompose the problem (they open branches for each decomposition) until it can be solved with the dynamic programming method. In contrast, in our algorithm, Beta(TS, PW), we have the β -Sequence which fathoms some of the branches even at the beginning, and we also have the TS-decomposition which decomposes the problem without any branches.

In order to compare the behavior of the solution times for both of the algorithms (of course we can make this comparison for $n \leq 100$ since the algorithm of Potts & Wassenhove cannot handle more than 100 jobs) it would be better if we had the code of their algorithm. But the code was not available and the algorithm given in the paper is not clear enough. But they give the solution times of their algorithm for $n = 50, n = 60, n = 70, n = 80, n = 90$ and $n = 100$ in their paper [25]. Trying to attain the same experimental conditions (using the same R, T pairs, having the same number of replications for each n) we also make experiments for those n values with our algorithm. The table below gives the average CPU times in seconds for each of the algorithms.

	$n = 50$	$n = 60$	$n = 70$	$n = 80$	$n = 90$	$n = 100$
Alg. of P & W	0.760	1.50	2.88	7.200	12.750	27.070
Beta(TS, PW)	0.095	0.32	0.57	1.224	1.971	4.028

Table 4.1: Table for comparison of the CPU times

The solution times of both of the algorithms are seen from the table above. But a graph will give more insight here. The plot of the solution times versus number of jobs is given in figure 4.6 .

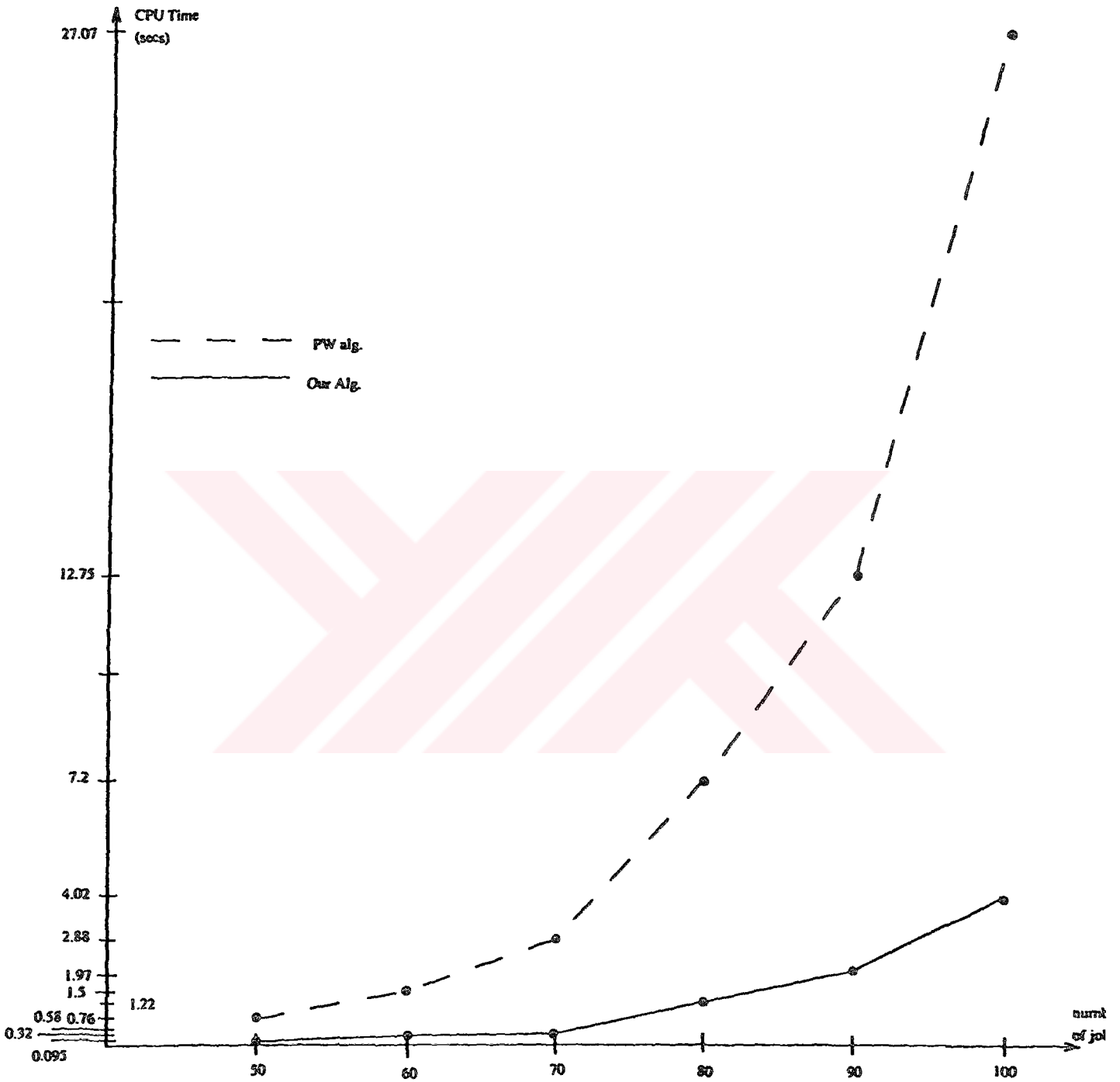


Figure 4.6: Graph of the solution times of both of the algorithms

As can be seen from the graph, the algorithm of Potts & Wassenhove becomes steeper after $n = 70$ and for $n = 100$ the slope makes a sharp increase. For our algorithm we do not see sharp increases for those values of n . Of course the graph will become steeper after $n = 100$ but the other algorithms cannot even handle problems with $n > 100$.

The algorithm of Potts & Wassenhove is coded in FORTRAN IV and CDC7600 machine is used. With our algorithm, Beta(TS, PW), the average time to solve 100-job problems is 4.028 secs and the time for 200-job problems is 4.329 minutes. The algorithm is coded in C language and Sun4 workstations are used. With this algorithm, we can handle 500-job problems, even though we cannot solve all of the instances to optimality.

Our aim was to increase the limits on solvability of this NP-Hard problem significantly. Handling 500-job problems with the existing algorithms in the literature seems to be out of question. Dynamic programming ones cannot solve because of the core storage requirements. Among the branch and bound algorithms best of the reported times is 7.9 sec for 50-job problems, so it will not be possible to solve 500-jobs with those algorithms in reasonable computation times. We managed to handle many 500-job problems, but we could not solve all of them to optimality because of time limitations. Table 4.2 will provide the details.

Problem hardness is defined by the R, T factors. In the table, the first column is the R, T factors and so defines the problem type. The following columns show the computational results for corresponding n values. The number in each cell is the average of the solution times in CPU seconds of the 10 problems corresponding to that R, T pair. The last row gives the overall average. For $n = 500$, as can be seen from the table, we could not solve 4 of the 16 cases to optimality since the time needed to solve these cases were more than 72 hours. For $n = 400$ case we first do not put that time bound and so some of the cases are solved in more than 72 hrs. One set of the problem instances, which is the hardest one, cannot be solved in 90 hrs and so we stayed with the premature solutions for that case also, like the 4 cases of $n = 500$.

R	T	n = 100	n = 200	n = 300	n = 400	n = 500
0.2	0.2	0.384	5.750	35.638	193.410	535.520
0.2	0.4	11.300	623.460	7845.690	34.4 hrs	-
0.2	0.6	21.690	1703.350	20.961 hrs	-	-
0.2	0.8	5.926	895.790	11016.820	75.61 hrs	-
0.4	0.2	0.000	0.000	0.000	0.000	0.000
0.4	0.4	2.400	95.870	385.100	3866.239	23137.400
0.4	0.6	11.190	698.011	13901.150	23.84 hrs	-
0.4	0.8	0.323	3.531	47.310	93.585	300.900
0.6	0.2	0.000	0.000	0.000	0.000	0.000
0.6	0.4	0.888	5.060	33.062	120.490	312.480
0.6	0.6	7.985	200.580	1788.850	15104.138	35.82 hrs
0.6	0.8	0.175	0.414	1.896	2.119	5.612
0.8	0.2	0.000	0.000	0.000	0.000	0.000
0.8	0.4	0.079	0.133	0.211	0.328	0.571
0.8	0.6	1.981	5.800	35.840	104.390	562.880
0.8	0.8	0.078	0.433	0.884	1.975	4.0780
Avg.	Comp.	4.0280 sec.	4.3290 min	1.9 hrs	11.01 hrs for 15 sets	-

Table 4.2: Table for the computational results of Beta(TS, PW)

Our algorithm handles problems of size 200 quite successfully with an average of about 4.5 minutes. The results of size 300 indicate that those problems will be solved in the average of 1.9 hours. When we analyze the individual CPU times, we saw that the maximum time for 200-job problems is 58 minutes. So within time limit of 1 hour, we expect to solve any instance of 200-job problems to optimality, whereas this limit is only 1 minute for 100-job problems.

The analysis of the table above shows that for three cases of R, T pair we can find the optimum solution immediately. Those are the cases for $T = 0.2$ with $R = 0.4, 0.6$ and 0.8 respectively. For these problems the optimum is found during our preprocessing or the β -Sequence happens to be the optimal one. So, if the problem is in those types the optimum solution can be found very easily for any n .

The hardest case happened to be the one for $R = 0.2, T = 0.6$.

In this case, the data is in a bad shape which causes β -Sequence and TS-decomposition to fail. Since both of them failed, we need to continue by branching and this causes the time to solve the problem to increase. For example for that R, T pair the number of branches is 585533 for $n = 300$ and the CPU time for that problem is about 17 hours, whereas for another problem with $n = 300$ from $R = 0.4, T = 0.8$ pair, the number of branches is 3468 and the solution time is about 0.5 minutes. This comparison is an example for the CPU time dependence on the number of branches.

It is seen from the results in the table that the problem is harder for the cases with small range of due dates. The three of the unsolved four cases of $n = 500$ are from $R = 0.2$ combination. It seems that, with that value of R it is difficult to find precedence relations and so one seldom gets the optimum from the β -Sequence or the application of the TS-decomposition. The $T = 0.6$ case is the one having the largest CPU time for each value of R . This is intuitive since increasing the tardiness factor increases the number of tardy jobs but when it increases too much, like for the case of $T = 0.8$, then most of the jobs become tardy and then it becomes relatively easy to find the optimum.

As a result, with our algorithm, Beta(TS, PW), we can handle 500-job cases, even though we were able to solve only %75 of the test problems of this size. We solved 200-job problems in 4.329 minutes in the average while we solved 100-job problems within a matter of few seconds in most instances.

Chapter 5

A NEW HEURISTIC : Beta

Since we had problems in solving large problems we wanted to find a good heuristic. This chapter will be for the heuristic that we developed. We will give the new heuristic and compare our heuristic with the best known heuristic of the literature which is the PSK of Panwalkar et al. [22]. So in the first section we give the details of their heuristic. The second section explains our heuristic. We give the comparisons in the third section.

5.1 PSK Heuristic

PSK is the best heuristic in the literature both in terms of solution quality (percent deviation from optimality) and of running time. Since we compared our heuristic with this one, it will be better to give the details of that heuristic.

The PSK heuristic starts with the SPT sequence and makes n passes to fix the jobs. Each pass starts with the smallest unscheduled job, which is called the active job, and according to some criterion, either fixes the active job or changes the active job in hand. The logic for the heuristic is not very well defined. What I understand from their heuristic is that if the smallest unscheduled job will be tardy in any sequence, schedule it to the first available

place. But this argument is not given in the paper.

Let me give the steps of the heuristic here.

Consider a set of jobs labeled in the SPT order. All unscheduled jobs are in the set U so the leftmost job in the set U is the smallest unscheduled job. Set S contains the scheduled jobs up to that time in the found order. Let C be the sum of the processing times of the jobs in set S . Initially $C = 0$, $S = \emptyset$ and set U is the whole set.

Step 1 If U contains only one job, schedule it in the last position in S and terminate, else label the leftmost job in U as the active job.

Step 2 If $C + p_i \geq d_i$ then schedule job i to the current place in set S , remove the job from the set U , add p_i to C and return to step 1.

Step 3 Select the next job in U as job j

Step 4 If $C + p_j \geq d_j$ then schedule job j to the current place in set S , remove the job from the set U , add p_j to C and return to step 1.

Step 5 if $d_i \leq d_j$ then return to step 3

Step 6 Now job j becomes the active job. If this is the last job, just schedule it else go to step 2

It will be better to give an example here to clarify the algorithm. Suppose we have 5 jobs with the following data.

Job No	Processing time	Due Date
1	2	8
2	2	17
3	3	6
4	4	5
5	7	8

Start with $U = \{1, 2, 3, 4, 5\}$

Step 1 $i = 1$

Step 2 $C + p_i < d_i$

Step 3 $j = 2$

Step 4 $C + p_j < d_i$

Step 5 $d_i = 8 \leq d_j = 17$ so return back to step 3

Step 3 $j = 3$

Step 4 $C + p_j < d_i$

Step 5 $d_i = 8 > d_j = 6$

Step 6 $j = 3$ becomes the active job

Step 2 $i = 3$ $C + p_i < d_i$

Step 3 $j = 4$

Step 4 $C + p_j = 5 < d_i = 6$

Step 5 $d_i > d_j$

Step 6 $j = 4$ becomes the active job

Step 2 $i = 4$ $C + p_i = 4 < d_i$

Step 3 $j = 5$

Step 4 $C + p_j = 7 \geq d_i = 5$ so schedule job 1 to the current place in set S

Now $U = \{1, 2, 3, 5\}$, $C = 4$, $S = \{4\}$ and continue.

With this algorithm Panwalkar et al. get good results both in solution time and solution quality.

5.2 Our Heuristic: Beta

During developing our heuristic, we used our algorithm Beta(TS, PW). The only part that can be played with is the branching part of the algorithm which is in fact based on the decomposition of Potts & Wassenhove. We decided to decrease the number of branches, and for that we first tried to take the first and last places for branching. That is, we ignored the rest of the alternatives that are developed and we just used the first and last places as alternatives and continue branching. We call this heuristic Two-Branch. What we found out from Two-Branch is that, the resultant sequence is really very good in solution quality, (much better than that of PSK and in fact, the resultant sequence was very similar to or, in most of the cases, the same with the optimum one) but the solution time was very high when compared with that of PSK. In fact, this is expected since the order of Two-Branch is $O(2^n)$ which is not good for being a heuristic.

Then we decided to decrease the number of branches to one in the Two-Branch, according to a criterion. That is, we select one of the alternatives according to some criterion that we developed. Then the resultant sequence should come up quickly since we do not branch any more.

In finding the criterion for evaluating the branches we use the interchange idea. Recall that the decomposition of Potts & Wassenhove is to decompose the problem with respect to the longest job in hand, say k , and so the branching is for the place of job k . Our aim is to find a condition which will avoid job k 's staying at its original position, or a condition to avoid job k 's being in last position, so that we will select one of the two as the position of job k . Since we have the β -Sequence in hand, it will be more meaningful to derive a condition which will avoid job k 's staying at its original position. That is, we assumed that job k is in its original place and we wanted to find a favorable interchange with this job and the ones following it in the β -Sequence. If we can find a favorable interchange then job k will not stay at its original place.

The figure 5.1 will be used in the tardiness calculations.

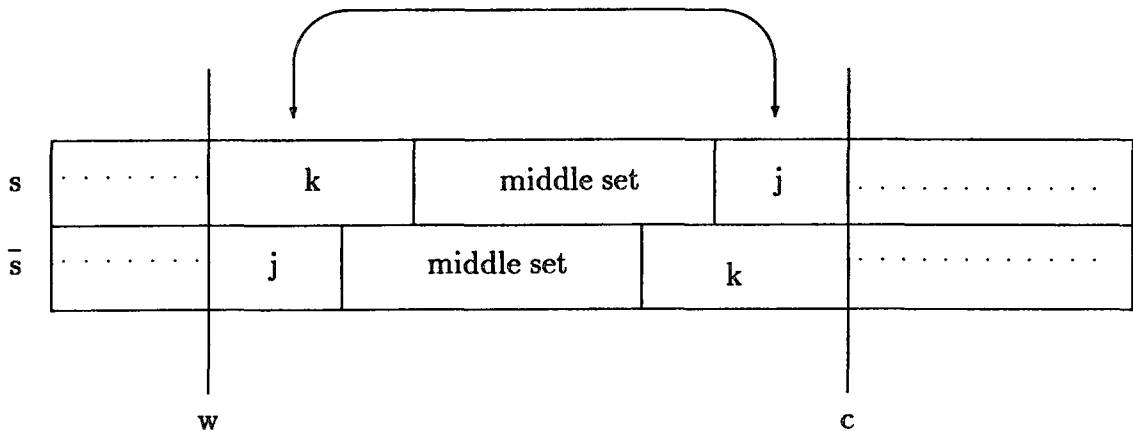


Figure 5.1: Figure for illustration of a swap

Since $p_k \geq p_j$ the tardiness in the middle set will decrease. So we can only try to show that the tardiness after interchange will be less than or equal to tardiness of the sequence before interchange. Then we will only need to consider the tardiness of jobs k and j . We have

$$T(S) = \max\{W + p_k - d_k, 0\} + \max\{C - d_j, 0\} + T(\text{middle set}) + K$$

$$T(\bar{S}) = \max\{C - d_k, 0\} + \max\{W + p_j - d_j, 0\} + \bar{T}(\text{middle set}) + K$$

We know that $p_k \geq p_j$ so if $d_k \geq d_j$ then the relation $j \leftarrow k$ will be found which contradicts with the generation of the β -Sequence. So we also know that $d_k < d_j$.

The table 5.1 illustrates $\Delta = T(S) - T(\bar{S})$

Aft. Move → Bef Move ↓	k=Tardy j=Tardy	k=Tardy j=NotTardy	k=NotTardy j=Tardy	k=NotTardy j=NotTardy
k=Tardy j=Tardy	(1) $p_k - p_j \geq 0$	(2) $W + p_k - d_j$	X	X
k=Tardy j=NotTardy	X	(3) $W + p_k - C < 0$	X	X
k=NotTardy j=Tardy	(4) X	(5) $d_k - d_j < 0$	(6) X	(7) X
k=NotTardy j=NotTardy	X	(8) < 0	X	0

Table 5.1: Tardiness changes caused by swap

Since we are trying to cause the movement to yield a nonnegative Δ we eliminate the ones for which $\Delta < 0$. We need to define conditions to avoid cells 3, 5, & 8 and cause $\Delta \geq 0$ for cell 2. Let us try the condition $W + p_k - d_j \geq 0$. Then we have $W + p_k \geq d_j > d_k$, so k is tardy in S which eliminates cells 5 and 8. And since $d_j \leq W + p_k \leq C$ then j is also tardy in S which eliminates cell 3. So if $W + p_k - d_j \geq 0$ then interchange of jobs j and k will not increase the total tardiness. So if the inequality is satisfied, then we can say that job k need not stay at its original position.

In our algorithm, for the time being, instead of searching for any j between k and n we just look at the condition for the last job in hand, say job n . That is if $W + p_k \geq d_n$ we select the last place for job k .

The above criterion is the one that we use in our heuristic. That is, when we encounter the branching case we apply the above inequality and if it is satisfied then we set the decomposition by assigning the largest job to the last place otherwise decompose with job k at its current position. We call this heuristic as Beta.

In fact, if $W + p_k < d_j \leq d_n$, the first place is an alternative to decompose in which case our heuristic, Beta, continues with decomposing at the first place.

Let us give an example here to clarify the heuristic.

Ex 7 Suppose we have a problem of 10 jobs. The process times and due dates are given below.

Job	Process time	DueDate	NewDueDate
1	1	271	229
2	36	239	197
3	42	246	204
4	49	198	156
5	47	231	189
6	43	257	215
7	31	235	193
8	42	150	150
9	42	270	228
10	8	255	213

Job 8 is fixed to the

first place during the preprocess. So the problem is reduced to $n = 9$ jobs and the due dates of these should be decreased by $p_8 = 42$. Those due dates are given in the third column. When we apply Beta for this problem:

First see if the β -Sequence is optimal or not. The β -Sequence is 4 5 7 2 3 10 6 9 1 and we found that it does not pass the β -test.

Then we try to decompose the problem by using the TS-decomposition, but the instance is such that the decomposition also fails.

So we would use the PW-decomposition now. The job with the largest process time is job 4. For this sequence $W = 0$ and we check if $W + p_4 = 49 < d_1 = 229$ and it fails, so we select the place that job 4 occupies as the alternative and continue with that assumption. (i.e. we fixed job 4 to the second place of the final sequence). So now we have 8 jobs, with ready times 49. We apply the same procedure again. That is we construct the BetaSequence which happens to be 5 7 2 3 10 6 9 1, but the β -test fails again. We try to decompose the problem, but it also fails to apply. So we decide on the place of the largest job in hand by our criterion again. The largest job is 5 and $W = 49$. We need to check if $W + p_5 = 96 < d_1 = 229$ and it fails. So we fix job 5 to the third place and continue with the rest of the jobs. So we are left with 7 jobs and they have a ready time of 96. When we construct the β -Sequence and

apply the β -test we see that the β -Sequence is optimal for this set of jobs. So we have the final sequence which is 8 4 5 7 2 3 10 1 9 6 which happens to be the optimal sequence also.

5.3 Comparisons

In order to compare Beta with PSK we should give the same conditions for both of the heuristics. To be fair, we add our preprocess part to PSK heuristic also, eventhough PSK does not have such a preprocesses before starting. So we mean preprocess followed by the original PSK when we say PSK only. We used the same sets of data that we have generated for the computational studies of our exact algorithms. So we have 16 sets of 10 problems for each n , and we took $n = 100, n = 200$ and $n = 500$ as before. But since we do not know the exact solutions of the 4 sets for $n = 500$ we omitted those cases. Also we do not need to make comparisons for 3 sets for all n cases, since in those cases the optimal is found during the preprocess. So we are left with $13 * 10 = 130$ problem instances yielding 13 deviations from optimum averages for $n = 100$ and $n = 200$ and we have $13 - 4 = 9 * 10 = 90$ different problem instances for $n = 500$.

The results were interesting since we encounter the same solution of PSK from Beta most of the time. That is, for $n = 100$ in 10 of the 130 problems the resultant tardiness of Beta is strictly better than that of PSK, and the solution values are equal for the remaining 120 instances. For $n = 200$ our heuristic gives better tardiness in 12 problem instances and the results were equal in the other 118 problems. For $n = 500$ the number of times we get better solution decreased to 3 among 90 and we still get the same solution for the remaining 87 problems. The deviations from the optimum varies with n . Since it was hard to understand the logic behind the PSK heuristic, this result was interesting. It will be better to tabulate the deviations from the optimum and their averages. The table given in the last page of this section illustrates the average deviations from optimum for both of the algorithms. The number in each cell is the average of 10 ratios of the form

$$(T(\text{Heuristic}) - T(\text{Optimum}))/T(\text{Optimum}).$$

The table below gives the average deviations of both of the heuristics from optimum.

n	Avg. Deviation of PSK	Avg. deviation Beta
100	0.02410	0.02250
200	0.03170	0.02540
500	0.02348	0.01460

Table 5.2: Average Deviations of the heuristics from optimum

As, can be seen from the table 5.2 the average deviations for PSK and Beta are very close to each other. But there are some cases in which the deviations differ. But, since we are taking averages, the affect of the values for the instances having different tardiness diminish. So it is better to identify the cases where the tardiness differ (i.e. the 10 instances of $n = 100$, 12 instances of $n = 200$ and 3 instances of $n = 500$). The table below gives the average deviations of those cases.

n	Avg. Deviation of PSK	Avg. deviation of Beta
100	0.02826	0.021100
200	0.09123	0.023870
500	0.24340	0.004943

Table 5.3: Average Deviations of the heuristics for cases they differ

As can be seen from the table 5.3, the deviation differs by a considerable amount for $n = 200$ and especially $n = 500$. In fact, for those cases there are some problem instances in which PSK resulted in a bad solution whereas Beta gives really good solutions(i.e. the tardiness of the resultant sequence of our algorithm is very near to the optimal value whereas that of PSK is really far.) For example, for $n = 200$ there are 2 such instances. One has tardiness of 1683 at optimum, PSK resulted in 2629 (has a deviation of 0.5621) and Beta gives 1747 (the deviation is only 0.038). The other one has tardiness of 1162 at optimum, PSK found a sequence with tardiness 1483 and our heuristic gives 1170 (which is very close to the optimum). Another extreme case occurs for $n = 500$. The optimum tardiness is 2581, PSK gives a tardiness of 4430 (the

deviation is 0.7164) whereas Beta found a sequence with tardiness 2587, which is very close to the optimum one. When we do not consider such extreme cases the average deviation table becomes :

n	Avg. Deviation of PSK	Avg. deviation of OUR Alg.
100	0.028260	0.02110
200	0.025640	0.02416
500	0.006877	0.00625

Table 5.4: Average deviations without the extreme cases

This time, the differences are small. This leads us to think more about the cases in which there was a tremendous change. Loosely speaking, we feel that the resultant optimal sequence for those cases might be similar to the EDD sequence and so the PSK failed since it starts with the SPT sequence. When we explored the extreme cases, in fact those cases were for $R = 0.8$ and $T = 0.4$ for both $n = 200$ and $n = 500$. We found such extreme results for $n = 200$ case (2 out of 10 runs), but we got the same results for $n = 500$ (both of the algorithms resulted in the same solution for this case). These results need some more study and this part is still ongoing.

We need to compare the solution times also, but the CPU times are not comparable yet, since our code of the heuristic is in a premature form. We will make some important changes and the solution times will change then. However we can say that, in this premature form, the solution time of our algorithm is less than 0.011 secs in the average for $n = 100$ case.

We can conclude that, for most of the cases, Beta and PSK give similar results, but for some instances, especially for the ones which have the optimum sequence similar to the EDD sequence our heuristic results in much better solutions.

n	R & T	Avg. Deviation of PSK	Avg. deviation Beta
100	0.2 0.2	0.049100	0.049100
	0.2 0.4	0.049610	0.049610
	0.2 0.6	0.054150	0.054150
	0.2 0.8	0.039530	* 0.039230
	0.4 0.4	0.027550	* 0.027500
	0.4 0.6	0.027330	0.027330
	0.4 0.8	0.001252	* 0.001246
	0.6 0.4	0.021880	0.021880
	0.6 0.6	0.020910	* 0.019500
	0.6 0.8	0.000699	0.000699
	0.8 0.4	0.001270	0.001270
	0.8 0.6	0.018990	* 0.001371
0.8 0.8	0.000500	0.000500	
200	0.2 0.2	0.063400	0.063400
	0.2 0.4	0.062990	* 0.062970
	0.2 0.6	0.063750	0.063750
	0.2 0.8	0.030860	* 0.030830
	0.4 0.4	0.025096	* 0.025094
	0.4 0.6	0.000954	0.000954
	0.4 0.8	0.020890	0.020890
	0.6 0.4	0.020442	* 0.019870
	0.6 0.6	0.017310	* 0.017170
	0.6 0.8	0.000163	0.000163
	0.8 0.4	0.099100	* 0.019020
	0.8 0.6	0.004130	0.004130
0.8 0.8	0.002896	* 0.002870	
500	0.2 0.2	0.051660	0.051660
	0.4 0.4	0.022630	0.022630
	0.4 0.8	0.000958	0.000958
	0.6 0.4	0.035960	0.035960
	0.6 0.6	0.012900	0.012900
	0.6 0.8	0.000105	0.000105
	0.8 0.4	0.083480	* 0.003720
	0.8 0.6	0.003457	* 0.003340
0.8 0.8	0.000062	0.000062	

Table 5.5: Deviations of the heuristics from optimum

Chapter 6

CONCLUSIONS and FUTURE RESEARCH

In this thesis, we developed an exact algorithm which solves problems with 100 and 200 jobs in really low times and can handle 500 jobs whereas the literature is limited to 100 jobs only. The algorithm is based on the β -Sequence of Tansel & Sabuncuoğlu [34] and decomposition theorems of Tansel & Sabuncuoğlu and Potts & Wassenhove [23]

The β -Sequence is very good for easy problems, since it gives the optimum immediately. Decomposition of Tansel & Sabuncuoğlu is also a helpful tool in solving large problems, since it exactly decomposes the problem for the cases it applies. The problem is that for "hard instances" described in Tansel & Sabuncuoğlu [34, 35] both β -test and TS-decomposition fail and we have to branch in order to solve the problem. With the normal branching criteria (i.e backwards or forwards branching) the time to solve the problem was really high. It is not possible to solve large problems with that type of branching. Then we used the decomposition theorem of Potts & Wassenhove. That theorem gives alternative decomposition which causes branches. And, since while branching, we also decompose the problem, the time to solve the problems decreased, and we became capable of handling 500-job cases.

In fact our aim was to solve all 500 jobs to optimality, but time required for some instances is very high and we had to terminate with premature

solutions for those cases. But with our algorithm $n = 100$, and $n = 200$ cases are solved to optimality and in really low times. It is 4.028 sec. for $n = 100$ and 4.329 min. for $n = 200$. The algorithm is coded in C language and Sun4 workstations are used. The average times of our algorithm is considerable when we compare with the best known exact algorithm which can only solve problems of size at most 100 and in 27.07 secs (in CDC7600 with FORTRAN IV coding).

We also developed a heuristic basing on our exact algorithm. The best known heuristic is the one developed by Panwalkar et al. [22]. That heuristic results in near optimal solution, and since it is a construction type heuristic, its running time is really low. But the logic behind the heuristic is not very clear. It seems to be the result of many experimental studies. For our heuristic, we used our exact algorithm, Beta(TS, PW) , but with relaxed branch evaluation criterion. That is , when we face the branching case, according to our criterion, we select one of the alternatives. The interesting result is that, in most of our experiments, our heuristic and the one of Panwalkar et al. result in the same sequence. But there are some cases in which our heuristic gives markedly better solutions (nearer to the optimum). We feel that those cases are the ones whose optimal solution is similar to EDD sequence, and so the heuristic of Panwalkar et al. fails.

For future research we first want to improve our heuristic. The form of the heuristic seems to be open for improvements and we will try to develop some different criterion, for evaluating branches, which will result in better solutions.

We also want to analyze the failure of the β -Sequence. The β -Sequence used in this thesis is slightly different than the original version and with this β -Sequence we may also know which pair of jobs causes the β -test to fail. This information might be useful for generating heuristics, or it might be a way for finding the exact solution from the β -Sequence. What we observe is that, the β -Sequence can be transformed to the optimal one with some movements and/or swaps. But, of course, a guidance for those movements are needed, and we also need some criteria which will indicate when we are at the

optimum. Otherwise we will only satisfy local optimality.

As a next step, we will analyze the β -Sequence based movements and swaps in comparison with different seed sequences like SPT, EDD. We expect to find the seed taken as β -Sequence will result nearer to optimal solutions and in less time than the others.



Bibliography

- [1] K.R. BAKER, J. BERTRAND (1982) "A dynamic priority rule for scheduling against due dates" *Journal of Operations Management* **3**, 37 - 42
- [2] K. R. BAKER(1974)*Introduction to Sequencing and Scheduling*, John Wiley, NewYork.
- [3] K.R. BAKER, J.B. MARTIN (1974) "An experimental comparison of solution algorithms for the single machine tardiness problem" *Naval Research Logistics Quarterly* **21**, 187 - 199
- [4] K.R. BAKER, L. SCHRAGE (1978) "Finding an optimal sequence by dynamic programming : An extension to precedence - related tasks" *Operations Research* **26**, 111 - 120
- [5] D.C. CARROLL (1965) "Heuristic sequencing of single and multiple components" Ph.D Dissertation, Massachusetts Institute of Technology.
- [6] R.J. CHAMBERS, R.L. CORRAWAY, T.J. LOWE, T.L. NORIN (1991) "Dominance and decomposition heuristics for single machine scheduling" *Operations Research* **39**, 639 - 647
- [7] S. CHANG, H. MATSUO, G. TANG (1990) "Worst Case Analysis of local search heuristics for the one machine total tardiness problem" *Naval Research Logistics* **37**, 111 - 121
- [8] C. CHU (1992) "A branch & bound algorithm to minimize total tardiness with different release dates" *Naval Research Logistics* **39**, 265 - 283

- [9] J. DU, T. LEUNG (1990) "Minimizing total tardiness on one machine is NP hard" *Operations Research* **15**, 483 - 495
- [10] S. ELMAGHRABY (1968) "The one machine sequencing problem with delay costs" *The journal of Industrial Engineering* **19**, 105 - 108
- [11] H.EMMONS (1969) "One machine sequencing to minimize certain functions of job tardiness" *Operations Research* **17**,701 - 715
- [12] M.L.FISHER (1976) "A Dual Algorithm for the one machine scheduling problem" *Mathematical Programming* **11**, 229 - 251
- [13] M.L. FISHER, A.M. KRIEGER (1984) "Analysis of a linearization heuristic for single machine scheduling to maximize profit" *Mathematical Programming* **28**, 218 - 225
- [14] T.D. FRY, L.VICENS, K. MACLEOD, S. FERNANDEZ (1989) "A heuristic solution procedure to minimize T-bar on a single machine" *Journal of operations Research Society* **40**, 293 - 297
- [15] J.E.HOLSENBACK, R.M. RUSSELL (1992) "A heuristic algorithm for sequencing on one machine to minimize total tardiness" *Journal of Operations Research Society* **43**, 53 - 62
- [16] E.L. LAWLER (1977) "A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness" *Annals of Discrete Math* **1**,331 - 342
- [17] E.L LAWLER (1982) "A fully polynomial approximation scheme for the total tardiness problem" *Operations Research Letters* **1** 207 - 208
- [18] J.K. LENSTRA, A.H.G. RINNOOY KAN (1984) "New directions in scheduling theory" *Operations Research Letters* **2**, 255 - 259
- [19] J.K. LENSTRA, A.H.G. RINNOOY KAN, P. BRUCKER (1977) "Complexity of machine scheduling problems" *Annals of Discrete Math* **1** 343 - 362

- [20] T.E. MORTON, R.M. RACHAMADUGU (1982) "Myopic heuristics for the single machine weighted tardiness problem", Working paper, Carnegie Mellon University
- [21] J.C. PICARD, M. QUEYRANNE (1978) "The time dependent travelling salesman problem and its application to the tardiness problem in one machine scheduling" *Operations Research* **26**, 86 - 110
- [22] S. S. PANWALKAR, M.L. SMITH, C.P. KOULAMAS (1993) "A Heuristic for the Single Machine Tardiness Problem" *European journal of Operations Research* **70**, 304 - 310
- [23] C.N. POTTS, L.N. VAN WASSENHOVE (1982) "A decomposition algorithm for the single machine total tardiness problem" *Operations Research Letters* **11**, 177 - 181
- [24] C.N. POTTS, L.N. VAN WASSENHOVE (1985) "A branch and bound algorithm for the total weighted tardiness problem" *Operations Research* **33**, 363 - 377
- [25] C.N. POTTS, L.N. VAN WASSENHOVE (1987) "Dynamic programming and decomposition approaches for the single machine total tardiness problem" *European journal of Operations Research* **32**, 405 - 414
- [26] C.N. POTTS, L.N. VAN WASSENHOVE (1991) "Single machine tardiness sequencing heuristics" *IIE Transactions* **23**, 346 - 354
- [27] R.M.V RACHAMADUGU (1987) "A note on the weighted tardiness problem" *Operations Research* **35**, 450 - 452
- [28] A.H.G. RINNOOY KAN, B.J. LAGEWEG, J.K. LENSTRA (1975) "Minimizing total costs in one machine scheduling" *Operations Research* **23**, 908 - 927
- [29] L. SCHRAGE, K.R. BAKER (1978) "Dynamic Programming solution of sequencing problems with precedence constraints" *Operations Research* **26**, 444 - 449

- [30] J. SCHWIMER (1972) "On the n job one machine sequence independent scheduling with tardiness penalties : A Branch & Bound solution" *Management Science* **18**, 301 - 313
- [31] T.T. SEN, L.M. AUSTIN, P.GHANDFOROUSH (1983) "An algorithm for the single machine sequencing problem to minimize total tardiness" *IIE Transactions* **15**,363 - 366
- [32] T.T. SEN, B.N. BORAH (1991) "On the single machine scheduling problem with tardiness penalties" *Journal of Operations Research Society* **42**, 695 - 702
- [33] V. SRINIVASAN (1971) "A hybrid algorithm for the one machine sequencing problem to minimize total tardiness" *Naval Research Logistics Quarterly* **18**, 317 - 327
- [34] B. TANSEL, I. SABUNCUOGLU (1994) "Geometry Based Analysis of Single Machine Tardiness Problem and Implications on Solvability" *Research Report* IEOR-9405
- [35] B. TANSEL, I. SABUNCUOGLU (1993) "An Analysis of Single Machine Tardiness Problem and New Insights" *Research Report*
- [36] J.I. WILKERSON, J.D IRWIN (1971) "An improved method for scheduling independent tasks" *AIIE Transactions* **3**, 239 - 245