Dominance Rules for Three-Machine Flow-shop Scheduling Problem With Unit Processing Times, Release Times and Chain Precedence Relationships

by

Dogan Corus

A Thesis Submitted to the

Graduate School of Engineering

in Partial Fulfillment of the Requirements for

the Degree of

Master of Science

in

Industrial Engineering

Koç University

August, 2012

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Dogan Corus

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by the final

examining committee have been made.

Committee Members:

[illegible]

Prof. Ceyda Oğuz

[illegible]

Assoc. Prof. Emre Alper Yıldırım

[illegible]

Asst. Prof. Alptekin Küpçü

ÖZETÇE

Akış tipi işlik problemleri, işlerin belirli bir sırayla makineleri ziyaret etmeleri gereken çizelgeleme problemleridir. Bu tez üç makineli, birim işlem süreli, başlangıç zamanlı, zincir yapılı öncüllük kısıtları olan akış tipi işliklerin en büyük geç kalma süresini enküçültmeyi amaçlayan problemle ilgilidir. Her iş her makineda bir birim işlem görmektedir. şler verilen başlangıç zamanlarından önce işleme başlayamaz. Her iş en fazla bir öncül ve bir ardıl işe sahip olabilir ve işler öncüllerinin bütün işlemleri bitmeden işleme başlayamaz. Problemi çözen kişinin amacı ise bütün işleri bitiş tarihlerine kadar tamamlamak, eğer gecikme olursa da bütün işler içerisinde en fazla gecikenin gecikmesini en az hale getirmektir. Bu problem hesaplama karmaşıklığı bakımından henüz sınıflandırılmamıştır.

Bu tezde yukarıdaki akış tipi işlik çizelgeleme problemi için polinom zamanlı bir algoritma geliştirilmesi amaçlanmıştır. Problemin iki makineli benzeri ile ilişki kurularak olurlu çizelgelerin bazı özellikleri belirlenmiştir. Yoğun zaman aralıkları incelenerek problemin başlangıç zamanları ve bitiş zamanları olurlu çizelge dışlamayacak şekilde daraltılmıştır ve bahsi geçen olurlu çizelge özellikleri polinom zamanlı algoritmalarla sağlanmıştır. özüm için yeterli şartlar ispatlanmış ve bu şartları sağlayacak değişiklikleri yapması için polinom zamanlı bir algoritma ispatsız bir şekilde önerilmiştir.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my primary advisor, Prof. Dr. Ceyda Oğuz whose sincerity and encouragement I will never forget. Her expertise did not only accurately guide me to obtain the necessary knowledge of the field, but also opened me various doors to extend the scope of this research. I also owe gratitude to Dr. Emre Alper Yıldırım and Dr. Alptekin Küpçü, for their constructive and insightful comments and for their time spent to read the thesis and give valuable comments for further improvement. I would also like to thank to the kind people around me by stressing that it would not have been possible to write this thesis without their help and support. Above all, I would like to thank my family and friends. Their patience and encouragement have provided me with the motivation I need to complete this thesis. And last but not the least, I would like to deeply thank to Birce for always being there to drag me out of my misery.

ABSTRACT

A flow-shop problem is a scheduling problem where every job consists of a fixed number of operations each of which to be processed on a different machine. The operations of each job have to be processed in a fixed order, in other words every job visits the machines in the same order. The problem of concern in this thesis is the three-machine flow-shop scheduling problem with unit processing times, release times and chain precedence relationships. All jobs have to spend unit time on each machine. The jobs are available to be initiated at their release times. A partial order of jobs that allows at most one successor and one predecessor for each job restricts the starting times so that every job can be initiated after its predecessor is completed. Our objective is to construct a schedule that will minimize the maximum lateness. The problem is currently open with respect to complexity classification.

In this thesis a polynomial algorithm to minimize the maximum lateness is sought for the above flow-shop scheduling problem with unit processing times, release times and chain precedence relationships. Several dominance rules for a feasible schedule are established through the correspondence with the two-machine version of the problem. By considering overloaded time intervals, given set of deadlines and release times are modified to find an equivalent problem with the same set of solutions. Further dominance rules are presented and shown to be achievable in polynomial time. Sufficient conditions for the solution are proved and a polynomial time limited backtracking algorithm exploiting the above mentioned dominance rules is suggested for the problem without precise proof of completeness.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

$\prec$	Partial order of jobs
n	Number of jobs
m	Number of machines
s	Number of stages in a pipelined processor
i, j, k, h	Indices for jobs
x_i	ith variable in a combinatorial optimization problem
D_i	Domain of variable x_i
C	Set of constraints
c	A single constraint
G	Graphical representation of $\prec$
$\alpha\|\beta\|\gamma$	Three-field notation that specifies scheduling problems
F	Flow-shop problems
$prec$	Arbitrary precedence constraints
$chains$	Chain structured precedence constraints
r_i	Release time of job i
p_{ij}	Processing time of job i on machine j
L_{max}	Maximum lateness
J_i	Job i
$\Im$	Set of jobs
S_i	Starting time of job i
d_i	Deadline of job i
$intree$	Intree structured precedence relations
C_{max}	Makespan

$outtree$	Outtree structured precedence relations
Fm	Flow-shop problem with m machines
$\sum C_i$	Sum of completion times
$\sum U_i$	Number of late jobs
$\sum C_i * w_i$	Sum of weighted completion times
$\sum T_i$	Sum of tardiness
r	Beginning of a time interval
d	End of a time interval
$\{r_i\}$	Set of release times
$\{d_i\}$	Set of deadlines
$(\Im, \prec)$	A job system
$S(i,r,d)$	The set of all jobs J_j $(j \neq i)$ which have $d_j \leq d$ and either $r \leq r_j$ or $J_i \prec J_j$
$N(i,r,d)$	Number of jobs in $S(i,r,d)$
$S'(i,r,d)$	The set of all jobs J_j $(j \neq i)$ which have $r \leq r_j$ and either $d_j \leq d$ or $J_j \prec J_i$
$N'(i,r,d)$	Number of jobs in $S'(i,r,d)$
$S(i,j,r,d)$	The set of all jobs J_k $(k \neq i \wedge k \neq j)$ which have $d_k \leq d$ and either $r \leq r_k$ or $J_i \prec J_k$ or $J_j \prec J_k$
$N(i,j,r,d)$	Number of jobs in $S(i,j,r,d)$
$\{d_i\}_k$	Set of consistent deadlines obtained after modifying d_k
$\{r_i\}_k$	Set of consistent release times obtained after modifying d_k
r_{hk}	Release time of hth job in release time set $\{r_i\}_k$
d_{hk}	Deadline of hth job in deadline set $\{r_i\}_k$
L	An increasing list with respect to $\{d_i\}$
σ	A schedule obtained by using list scheduling with respect to L
$\sigma(J_i)$	Starting time of job i in schedule σ
t_0	Largest positive integer less than $\sigma(J_i)$ such that either no job is initiated at t_0 or the deadline of the job initiated at t_0 is greater than d_i
U	Set of jobs initiated at times $t_0+1, t_0+2, \ldots, \sigma(J_i)$

Chapter 1

INTRODUCTION

In this thesis a polynomial algorithm that minimizes the maximum lateness is sought for the three-machine flow-shop scheduling problem with release times, unit processing times and chain precedence constraints. A flow-shop problem in general is a scheduling problem where every job consists of a fixed number of operations that each should be processed on a different machine. The operations of each job have to be processed in a fixed order, in other words every job visits the machines in the same order. Every operation must be processed for a given processing time that might depend on the job and/or the machine however for the problem of concern in this thesis the processing times are all equal to one. The jobs are available from their distinct release times. A partial order $\prec$ of jobs that allows at most one successor and one predecessor for each job restricts the starting times so that every job can be initiated after its predecessor is completed. Our objective is to construct a schedule that will minimize the maximum lateness.

1.1 Computational Complexity

Computational complexity is the quantitative measure of how much resource an algorithm needs before completion. The complexity is measured, most commonly, in two different dimensions, time and space. The basic notion behind computational complexity is best illustrated via the Turing machine [11]. A Turing machine is the theoretical model of a computer which has an infinite length of tape divided into cells one next to another and a head which can read and alter the symbols on the tape one cell at a time. With a finite set of symbols, once initiated with a blank tape, the Turing machine's head reads the symbol on the current cell, checks its internal state and, according to a fixed set of rules, alters the internal state together with the symbol on the current cell and moves one cell to the right or left. The machine halts when a predefined internal state and a current cell symbol couple is reached. Since the rules can be set arbitrarily, it is possible to create a machine that operates forever without halting but, even if the operation ever halts, two measurable quantities can be observed. First is the number of moves made by the head, which is called the time complexity since every move/read/write sequence is assumed to last a unit time. The other is the number of different cells visited by the head, which corresponds to the space complexity representing the amount of information stored during the operation. Apparently, the time complexity is always at least as much as the space complexity since at each step only one cell can be visited.

In a more practical sense, computational complexity is the number of simple arithmetic operations (addition, multiplication, comparison, etc.) made and the size of the memory needed for an algorithm to solve a given problem. The computational complexity of an algorithm can be constant, which means that it completes in a constant number of the steps regardless of the input. A simple and common example given for such an algorithm is the one that determines whether a given number is divisible by two. Since it is only the last digit of the number that is checked, the number of operations does not depend on how large the given number is, therefore its time complexity is constant. However if we consider the similar problem of determining whether a number is divisible by three, we see that the complexity of the algorithm depends on the given number. That is because when deciding on divisibility by three, the digits are added up and the number of addition operations performed depends on the number of digits. Most interesting algorithms are of at least linear complexity, since we generally at least want to read the whole input. If the computational complexity is a function of the input size, conventionally only the term that grows the fastest is used without any constant coefficient. For example if an algorithm completes in $5n^3 + 4n^2$ steps where n is the input size, it is said that the algorithm runs in $O(n^3)$ time [5].

Since the number of necessary operations might depend on the order of certain possibilities being checked, the computational complexity of an algorithm is based either on the expected number of operations or the worst-case scenario where the right solution is always at the last possible alternative checked.

1.2 Complexity Classes

With a quantitative measure on the performance of algorithms, a comparison between the problems they solve is also possible and a measurable hardness for problems can be established. Since an algorithm that requires less time to compute a given problem is preferable, the problems can be classified according to the algorithm with least computational complexity that can solve it. The problems that can be solved with algorithms of similar complexity is said to form a“complexity class”.

One such complexity class is P , consisting of decision problems that can be solved by polynomial time algorithms. Another class of problems that encompasses P is the class NP which stands for“non-deterministic polynomial” and it includes all problems that can be solved by a“non-deterministic Turing machine” in polynomial time. A non-deterministic Turing machine is similar to a (deterministic) Turing machine except that it has multiple alternative rules for a given symbol and an internal state tuple. The“non-deterministic Turing machine” does not choose any of the alternatives but conducts all of them simultaneously, creating branches in the algorithm. Since it conducts all of the possible moves simultaneously, the time complexity does not increase with each branch. In other words, given a“certificate”, a set of choices among the alternative moves,“non-deterministic Turing machine” can decide in polynomial time whether that certificate corresponds to a solution for a problem in NP. It is apparent that any problem in P is also in NP but the more challenging question is whether there are problems in NP that are not in P . To this day, a proof for any problem in NP stating that it can never be solved by a polynomial algorithm was not provided and this lack of proof is still one of the most prized mathematical challenges in the world. However,“Cook's Theorem” stated that any problem in NP can be reduced to the “Satisfiability problem”, which is in NP itself, in polynomial time. This means that if“Satisfiability problem” can be solved in polynomial time then all the problems in NP can be solved in polynomial time and P =NP. By showing that Satisfiability problem can be polynomially reduced to some other problems in NP, a complexity sub-class called NP-complete problems consisting of the most difficult problems in NP is established.

Since the polynomial reduction is transitive, any problem that can be reduced in polynomial time to an NP-complete problem is also NP-complete. This allowed an expansive categorization of many combinatorial decision problems into NP-complete category which in turn created the complexity class of NP-hard consisting of optimization versions of decision problems which are NP-complete.

1.3 Dominance Rules

Every problem is characterized by a general description of its parameters and a statement of what constitutes a solution for the problem. For example a"traveling salesman problem" is characterized by the rules of the distance matrix and the definition of a Hamiltonian path. Combinatorial optimization problems are defined by an n-tuple of variables $(x_1, x_2, \ldots, x_n)$ where each variable's value belongs to a set D_i, called the domain of x_i, a set C of constraints on variables and an objective function $z : D_1 \times D_2 \times \ldots D_n \to \Re$ which relates each assignment to a real value. A combinatorial decision problem is almost always in class NP since a non-deterministic Turing machine can represent every assignment of a combinatorial optimization problem as a choice of moves that select certain values from a finite set.

A constraint belonging to C is a function $c : \omega = D_1 \times D_2 \times \ldots D_n \to P(\omega)$ which associates with ω a subset $c(\omega) \subseteq \omega$ containing assignments satisfying the constraint. The elements in $c(\omega)$ is said to be feasible with respect to constraint c and the complementary set $\omega - c(\omega)$ is said to be infeasible. The intersection of all $c(\omega)$ for all $c \in C$ constitutes the set of feasible solutions for the combinatorial optimization problem with constraint set C. Each constraint in this sense provides a subset of solutions which allows the solver to focus its attention in a smaller search space.

Dominance rules are an extension of constraints of a problem. A dominance rule provides conditions under which certain potential solutions can be ignored. They can be considered as inferred constraints that are not in the original description of the problem but must be satisfied in a feasible solution. Similar to the constraints, dominance rules reduce the search space and lead to an easier problem for a search algorithm that seeks a feasible solution.

There are small differences between various definitions of dominance rules in the literature. While in some cases a dominance rule is a rule which allows a set of infeasible solutions to be omitted, in other cases it is a rule where a set of isomorphic solutions is rejected or a subset of solutions which contains a subset of feasible solutions is chosen. In all these cases, a dominance rule is used to eliminate uninteresting solutions or selecting interesting solutions.

1.4 Scheduling Problems

Scheduling problems are combinatorial optimization problems which are characterized by a machine environment, a set of jobs and an objective function. The objective is to find a“schedule” with a set of start times that do not overlap, for each job and machine couple that produces the best objective function value. The start times are restricted by the job characteristics such as release times, setup times and precedence relationships. Release time indicates the earliest time a job can be assigned to a machine. Precedence relationship is an acyclic digraph G that has a vertex for every job and restricts a job j to start after the completion of job i if the edge $(i, j) \in G$. The precedence relationships are further categorized according to the characteristics of its representing graph G. The classes of precedence commonly investigated in the literature are *outtree*, which allows a single incoming edge for every vertex, *intree* which allows a single outgoing edge for every vertex, and *chain* which allows at most one incoming and one outgoing edge for every vertex.

Machine environment defines the characteristics of the resources used for processing the given jobs. Parallel machine problems involve multiple machines that can process any job. Flow-shop problems consist of a set of machines that each job must visit in the same order. Job shop problem generalizes the flow-shop and allows distinct routes for different jobs. A relaxed version of job shop problems is the open shop problem where each job has a set of machines to be visited but it is free to do it in any order.

A scheduling problem is further specified by the objective functions. One of the simplest objective function is the completion time of the last job, also called the “makespan”.

With the addition of a new set of parameters called"deadlines" by which the jobs must be completed, the lateness of a job is defined as the difference between the completion time and deadline of a job. While minimizing the maximum "lateness" can be an objective function, "tardiness" which is the non-negative counterpart of lateness is also commonly used. Minimizing the maximum lateness problem is always polynomially reducible to the problem of finding a feasible schedule for a given deadline. More complicated objective functions include minimizing weighted tardiness, sum of completion times and the number of late jobs. A hierarchy of hardness among objective functions that is independent from the machine environment and job characteristics given given in Figure 1.1. In that hierarchy the objective function in the bottom is the easiest and complexity increases as we go up the hierarchy. The notation most commonly used for representing different scheduling problems is the"'three-field notation" which is written in the form $\alpha|\beta|\gamma$. The α field gives information about the machine environment, β field contains the characteristics of the jobs and γ field is for indicating the objective function. The problem of minimizing the maximum lateness for the three-machine flow-shop scheduling problem with unit processing times, release times and chain precedence relationships is expressed as $F3|chains, r_i, p_{ij} = 1|L_{max}$ in three-field notation.

For a $F3|chains, r_i, p_{ij} = 1|L_{max}$ problem with job set $\Im = \{J_1, J_2, \ldots, J_n\}$, a feasible schedule consists of start times S_i for job set where $S_i \geq r_i$, $S_i + 3 \leq d_i$ and $\forall(i, j) \in \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$ if $J_i \prec J_j$ then $S_j \geq S_i + 3$ which means that every job must wait for its predecessor to exit the third machine in the system. Even though a job that completes its operation on a machine can initiate its operation on next machine in any later time, every feasible schedule has another corresponding feasible schedule with no pause between operations of the same job and such that no job completes at a later time than it does in the original schedule. Because if we consider the first job in schedule that does not initiate its operations right after another, we can see that there are no other available jobs to be initiated at the omitted machine and the rest of the schedule will be no better than the case where every operation is scheduled right after the completion of the previous operation of the same job. Hence this problem and other flow-shop problems with unit processing

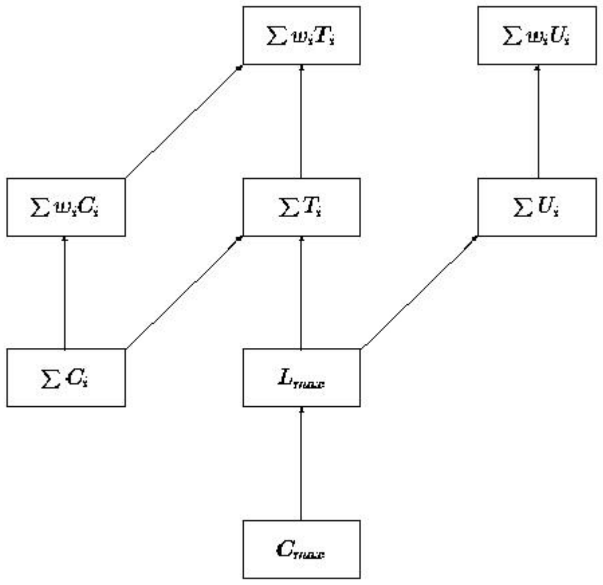

Figure 1.1: Hierarchy of objective functions minimizing makespan C_{max}, maximum lateness L_{max}, sum of completion times $\sum C_i$, sum of tardiness $\sum T_i$, number of late jobs $\sum U_i$, sum of weighted completion times $\sum w_i C_i$, sum of weighted tardiness $\sum w_i T_i$ and sum of weighted number of tardy jobs $\sum w_i U_i$

times can be analyzed as single pipelined (or multi-stage) processor problems with $s = m$ stages where m is the number of machines. In a pipelined processor setting, one job with processing time s can be initiated at every time slot and a second job can start just after the first one [2].

1.5 Outline of Thesis

This thesis contributes a set of dominance rules that are necessary or sufficient for a feasible solution to the $F3|chains, r_i, p_{ij} = 1|L_{max}$ problem. These dominance rules are then exploited by a proposed algorithm which outputs a set of deadlines which provides a feasible schedule when a list scheduling algorithm is applied to them. The next chapter will consider the complexity classification of flow-shop problems in the literature. In Chapter 3 a formal definition of the dominance rules that are adjusted to three-machine environment will be provided. Chapter 4 will cover other dominance rules. In Chapter 5 the reason why this simple dominance rule yields a feasible schedule for two-machine problem while it is not sufficient for the three-machine is discussed and an algorithm that exploits the dominance rules to find a feasible schedule for $F3|chains, r_i, p_{ij} = 1|L_{max}$ will be suggested.

Chapter 2

LITERATURE REVIEW

2.1 *Complexity Classes of Flow-shop Problems*

If we consider multi-stage single machine problems in the literature which are proved to be NP-hard, we see that most of the reductions are dependent on the number of machines, transportation or setup times which can take arbitrary integer values. In other problems the objective functions are either weighted or hierarchically harder than L_{max}. The easiest flow-shop problems which are proved to be NP-hard can be seen in Table 2.1 with corresponding works that establish their classification.

Table 2.1: List of NP-hard Flow Shop Problems

Problem	**Reference**
$F\|intree, r_i, p_{ij} = 1\|C_{max}$	[1]
$F\|prec, p_{ij} = 1\|C_{max}$	[8],[10]
$F2\|chains\|C_{max}$	[7],
$F2\|r_i\|C_{max}$	[7]
$F3\|\|C_{max}$	[6]
$F\|outtree, p_{ij} = 1\|L_{max}$	[1]
$F2\|\|L_{max}$	[7]
$F2\|\|\sum C_i$	[6]
$F2\|chains, p_{ij} = 1\|\sum C_i * w_i$	[9]
$F2\|chains, p_{ij} = 1\|\sum U_i$	[1]
$F2\|chains, p_{ij} = 1\|\sum J_i$	[1]

A polynomial time algorithm for $F3|chains, r_i, p_{ij} = 1|L_{max}$ problem is noteworthy since the complexity class of the problem is not determined and such an algorithm would establish that the problem is in the class P . The problem $F2|prec, r_i, p_{ij} = 1|L_{max}$ is in P [2] while $F|prec, r_i, p_{ij} = 1|L_{max}$ is NP-hard [10]. This means that $F3|prec, r_i, p_{ij} = 1|L_{max}$ sits at a critical point in problem hierarchy where a constant number of machines might still be the breaking point if proved to be NP-hard. Considering that it is in a sense "closer"' to the problems that are known to be in P , seeking a polynomial algorithm for $F3|prec, r_i, p_{ij} = 1|L_{max}$ is a better starting point for finding such a breaking point rather than seeking a polynomial reduction that will yield an NP-hardness result. At this point, the problem $F3|chains, r_i, p_{ij} = 1|L_{max}$ which is still open and hierarchically easier than $F3|prec, r_i, p_{ij} = 1|L_{max}$ is the initial interest.

It should be noted that minimizing maximum lateness in polynomial time is possible by using an algorithm that either provides a feasible schedule or proves that no such schedule exists in polynomial time. With an algorithm that solves the decision problem in polynomial time we can check whether there exists a schedule with zero maximum lateness. If there exists no such schedule then we can add a constant c to all deadlines and check whether there exists a feasible schedule which means that there exists a schedule for the original problem such that the maximum lateness is less than c. Using a binary search to find the smallest positive integer c that should be added to each deadline in order to achieve a feasible schedule actually provides the minimized maximum lateness [4]. Because of this property, in the remaining part of the thesis, our main concern will be finding a feasible schedule for a given set of deadlines. Moreover, most (if not all) necessary rules of a feasible schedule that will be discussed are valid for the arbitrary precedence constraints and for problems with a fixed number of machines m since the main inspiration is that solution of $F2|prec, r_i, p_{ij} = 1|L_{max}$ that is given in [2] can be extended to problems with more machines and the long term objective is to find a solution for $F3|prec, r_i, p_{ij} = 1|L_{max}$ and $Fm|prec, r_i, p_{ij} = 1|L_{max}$.

2.2 $F2|prec, r_i, p_{ij} = 1|L_{max}$

In [2], the deadlines given in the problem $F2|prec, r_i, p_{ij}|L_{max}$ are modified through the dominance rules that restrict the deadline of jobs around congested time frames. A congested time frame is a time interval $[r, d]$ in the scheduling horizon where the number of jobs that are both released and must be completed during that interval is so large that other jobs which are released before that interval and due in that interval must start before that inter-

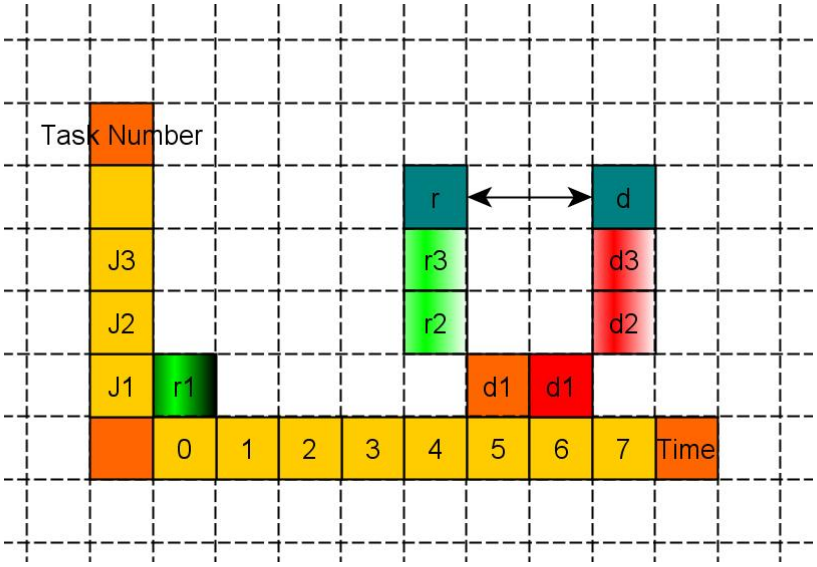

Figure 2.1: Type 1 congestion for two-machine problem

val begins. Figure 2.1 demonstrates a congestion for an instance of the $F2|prec, r_i, p_{ij}|L_{max}$ problem defined by the parameters:

$\{r_i\} = \{0, 4, 4\}$

$\{d_i\} = \{6, 7, 7\}$

$\prec = \{\}$

If we consider the interval $[4, 7]$ we can see that for a feasible schedule J_2 and J_3 must begin processing at 4th and 5th time slots in any order. In the figure, the white shaded jobs are the ones that creates the congestion in the interval. Since this means that J_1 cannot start in times slots 4 and 5, there cannot be a feasible schedule where J_1 completes in 6. This allows us to reduce the deadline of J_1 to 5 and obtaining an equivalent problem with a more restricted solution space. Extension of this rule [2] proposes a second dominance rule to modify deadlines. To illustrate this second rule, consider the problem in Figure 2.2 where the problem is characterized by the following parameters.

$\{r_i\} = \{0, 2, 4, 4\}$

$\{d_i\} = \{5, 7, 7, 7\}$

$\prec = \{(1, 2)\}$

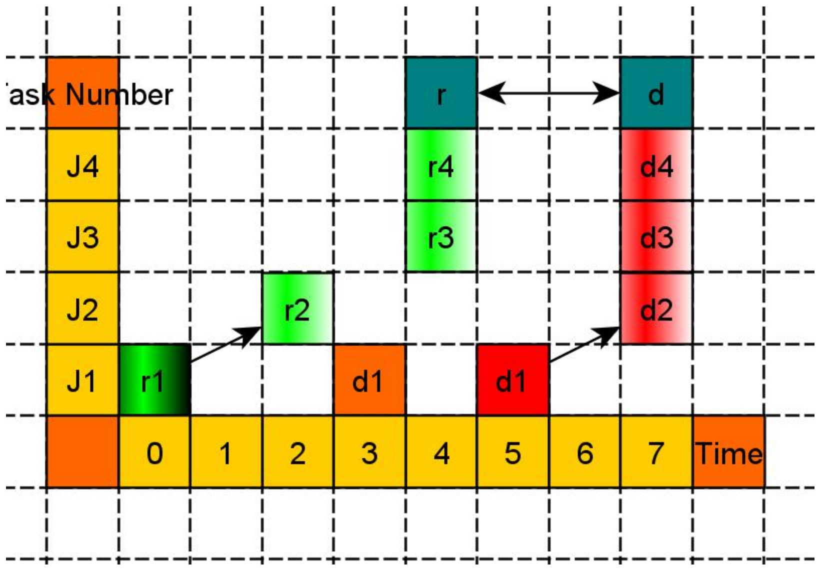

Figure 2.2: Type 2 congestion for two-machine problem

In this second example we see that the interval $[4, 7]$ is not only occupied by the jobs that are released and due in that interval but also by the jobs that succeed J_1. With regards to the previous dominance rule we discussed, since J_3 and J_4 must start in time slots 4 and 5, J_1 can start at 3. But if we consider the implications of its successor J_2, which will by definition start after J_1's completion, J_1 cannot complete at 5 or 4. Since if it does, not only J_3 and J_4 but also J_2 will be in the interval $[4, 7]$, and obtaining a feasible schedule will be impossible. The dominance rule inferred by this restriction, since we have to choose a job to decide which other jobs create a congestion, requires a deadline modification algorithm that searches the scheduling horizon and the job list to find triples (i, r, d) that allow deadline modifications. In Figure 2.2 the chosen job is shaded with black so that it is distinguishable.

Chapter 3

DEADLINE CONGESTION

3.1 Dominance Rule

In this section the known results for $m = 1$ and $s = 2$ [2] are extended to the case $m = 1$ and $s = 3$. This mainly consists of a deadline inference scheme where a job's given deadline is decreased if any period of time between its release time and its current deadline has just enough or more jobs that must start in that period. This consists of the jobs that are released and must be finished in that period and the job's successor.

The formal development here will follow [2]. Let $(\Im, \prec)$ be a job system with chain precedence relationship, $\{r_i\}$ a set of positive integer release times, and $\{d_i\}$ a set of positive integer deadlines. The goal is to construct a *feasible schedule* for $(\Im, \prec)$ on three-machine flow-shop environment($F3$) or on the corresponding single multi-stage processor with three stages ($m = 1$ and $s = 3$).

For any job J_i and integers r, d satisfying $r_i \leq r \leq d_i \leq d$; $S(i, r, d)$ is defined to be the set of all jobs J_j $(j \neq i)$ which have $d_j \leq d$ and either $r \leq r_j$ or $J_i \prec J_j$. Let $N(i, r, d)$ denote the number of jobs in $S(i, r, d)$. Now an important rule necessary for a feasible schedule which justifies certain deadline modifications can be stated and proved.

Lemma 1. *Let $(\Im, \prec)$ be a job system with $m = 1$, $s = 3$, and σ be a feasible schedule. Then for any job J_i and integers r, d satisfying $r_i \leq r \leq d_i \leq d$ the following holds:*

1) *If $N(i, r, d) = d - r - 2$, then J_i must be completed by $d - N(i, r, d)$.*

2) *If $N(i, r, d) > d - r - 2$, then J_i must be completed by $d - N(i, r, d) - 2$*

Proof. **1)** Suppose $N(i, r, d) = d - r - 2$ and let σ be any feasible schedule. Since all the jobs in $S(i, r, d)$ have deadlines that do not exceed d and $s = 3$, all jobs in $S(i, r, d)$ must start before $d - 2$. If all the jobs in $S(i, r, d)$ start no sooner than r then J_i starts no later than time $r-1 = d - N(i,r,d) - 3$ and finishes no later than $d - N(i,r,d)$. Otherwise some job J_j in $S(i,r,d)$ starts before r. By the definition of $S(i,r,d)$, $J_i \prec J_j$ and thus J_i is completed by $r = d - N(i,r,d) - 2$.

2) Suppose $N(i,r,d) > d-r-2$. Then at least $N(i,r,d) - d + r + 2$ jobs in $S(i,r,d)$ start before r. By the definition of $S(i,r,d)$, J_i precedes each of these jobs and thus J_i is completed by time $r - (N(i,r,d) - d + r + 2) = d - N(i,r,d) - 2$. ■

Lemma 1 can be extended to any fixed number of machines by simply replacing -2 with $-s+1$, s being the number of machines (or number of stages in pipelined processor framework). As a consequence of Lemma 1, if $N(i,r,d) = d-r-2$ and $d_i > d - N(i,r,d)$, then d_i can be set to $d - N(i,r,d)$, or if $N(i,r,d) > d-r-2$ and $d_i > d - N(i,r,d) - 2$, then d_i can be set to $d - N(i,r,d) - 2$; and these changes will not alter the set of feasible schedules. The instance in Figure 3.1 with the following parameters can illustrate the congestion and the subsequent changes.

$\{r_i\} = \{0,3,3\}$

$\{d_i\} = \{6,7,7\}$

$\prec = \{\}$

Much like the case in two-machine problem, there are two jobs which are both released and due in the interval $[3,7]$. Since the operations takes 3 unit times to complete the jobs J_2 and J_3 must start either on time slot 3 or time slot 4 so that they can make their deadlines. J_1 in that scenario cannot initiate in any of those time slots and cannot be completed just at its deadline. Since $N(i,r,d) = 2$ for $i=1$, $r=3$ and $d=7$ which is equal to $d-r-2$, J_1 must be completed by $d - N(i,r,d) = 5$ effectively reducing its deadline by one.

An example corresponding to the second part of Lemma 1 would be as the following and shown in Figure 3.2.

$\{r_i\} = \{0,3,4,4\}$

$\{d_i\} = \{5,8,8,8\}$

$\prec = \{1,2\}$

Like the two-machine counterpart, when the number of jobs in an interval is more than that can be processed in the given time, the deadline modification is performed to ensure

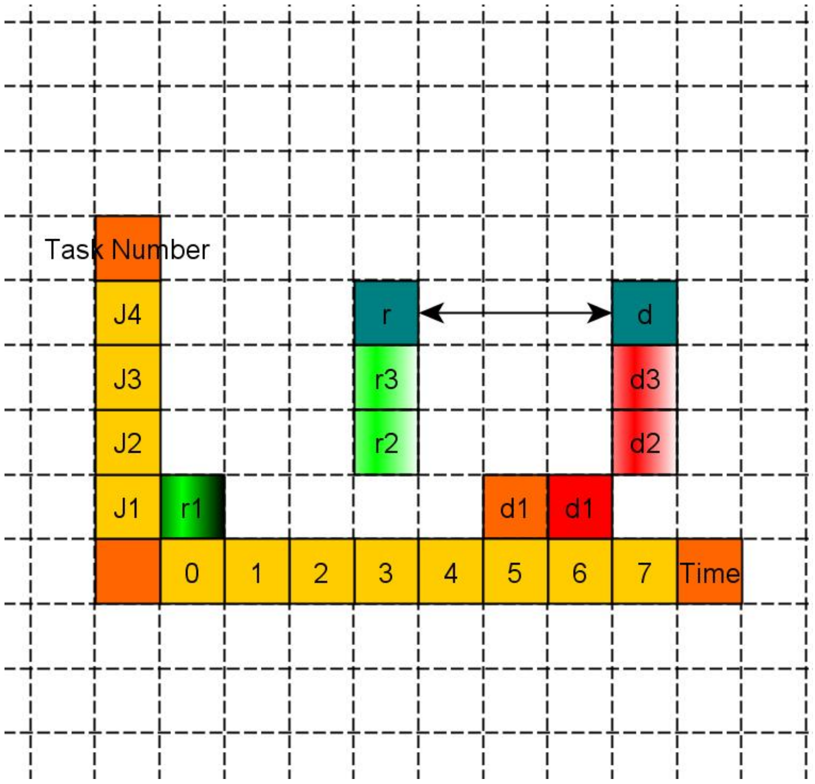

Figure 3.1: Type 1 congestion for three-machine problem

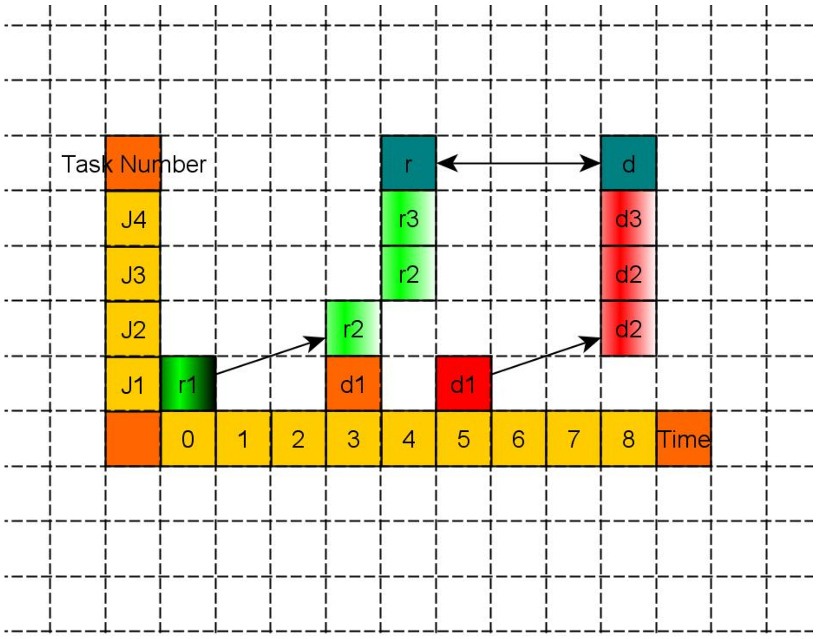

Figure 3.2: Type 2 congestion for three-machine problem

that the successor in the interval can start before the time slot r. While the first part of Lemma 1 pushes the deadline for the same amount in two-machine and three-machine cases, a deadline change originating from the second part is larger for three-machine problem since the successor must wait for the completion of the predecessor, not its initiation.

Following [2] we say that $\{d_i\}$ and $\{r_i\}$ are internally consistent with respect to Lemma 1 whenever the following conditions hold for every job $J_i \in \Im$:

1. $d_i \geq r_i + 3$;

2. for every pair of integers r, d satisfying $r_i \leq r \leq d_i \leq d$

   (a) if $N(i,r,d) = d - r - 2$, then $d_i \leq d - N(i,r,d)$;

   (b) if $N(i,r,d) > d - r - 2$, then $d_i \leq d - N(i,r,d) - 2$

Lemma 1 consistency is a necessary condition for a feasible schedule and its corresponding condition in $F2|prec, r_i, p_{ij} = 1|L_{max}$ is also sufficient for a simple list scheduling with respect to deadlines giving a feasible schedule. The proof for that sufficiency will not be provided here but the algorithm that is used to establish such consistency will be given in detail. The motivation for this algorithm is to find a subset of the possible schedules that does not exclude any feasible solution.

3.2 Deadline Modification Algorithm (DMA)

First we describe and discuss the preprocessing that must be done as described in [4]. The first step is to sort and re-index the jobs so that $r_1 \leq r_2 \leq \ldots \leq r_n$. Next we compute the transitive closure of the partial order so that, in constant time, we can determine whether or not $J_i \prec J_j$, $1 \leq (i,j) \leq n$. Then we perform the preliminary deadline and release time modifications to ensure that $d_i \leq d_k - 3$ and $r_k \geq r_i + 3$ whenever $J_i \prec J_k$. After that we initialize variable d to a value that exceeds the largest deadline. The algorithm then proceeds as follows:

1. If any job J_i has $d_i \leq r_i$, halt (no feasible schedule is possible). If no job has a deadline less than d, halt (the current deadlines are internally consistent). Otherwise set d to the largest job deadline less than d and set i to the least job index for which d_i is less than or equal to the new value of d.

2. Scan the job list to compute $N(i, r_i, d)$. Set $COUNT \leftarrow N(i, r_i, d)$, $r \leftarrow r_i$, and $k \leftarrow$ least j such that $r_j = r_i$.

3. If $COUNT = d - r - 2$ and $d_i > d - COUNT$, set $d_i \leftarrow d - COUNT$ and for each $J_j \prec J_i$ whose deadline exceeds $d_i - 3$, set $d_j \leftarrow d_i - 3$.

4. If $COUNT > d - r - 2$ and $d_i > d - COUNT - 2$, set $d_i \leftarrow d - COUNT - 2$ and for each $J_j \prec J_i$ whose deadline exceeds $d_i - 3$, set $d_j \leftarrow d_i - 3$.

5. If $r \geq d_i$, go to Step 6. Otherwise increment k by 1 until either $k > n$ or $r_k > r$. During this scan, decrease $COUNT$ by one for each J_j (original $k \leq j <$ new k) which is not a successor of J_i and which satisfies $(d_j \leq d \wedge r_j = r \wedge j \neq i)$. If $k = n + 1$ or $r_k > d_i$, set $r \leftarrow d_i$ and go to Step 4. Otherwise set $r = r_k$ and go to Step 4.

6. Find the least $j > i$ such that $d_j \leq d$. If such a j exists, set $i \leftarrow j$ and go to Step 2. If no such j exists and some job has current deadline d, go to Step 1. Otherwise halt (no feasible schedule is possible).

The complexity of the above algorithm is bounded by $O(n^3)$. The full proof can be found in [4] but a partial proof can be insightful. We must first denote that every triple $\{i, r, d\}$ is only considered once. Secondly to ensure that we do not miss any deadline modification, we consider the loop structure where the values for d is selected in the outermost loop. We first observe that the value of $N(i, r, d)$ and the new deadline that might be imposed on job J_i if $N(i, r, d) \geq d - r - 2$ do not depend on the value of d_i itself. What concerns that specific deadline modification is the value of $N(i, r, d)$. Extra considerations of the triple (i, r, d) cannot lead to modifications that were not made the first time that triple was considered

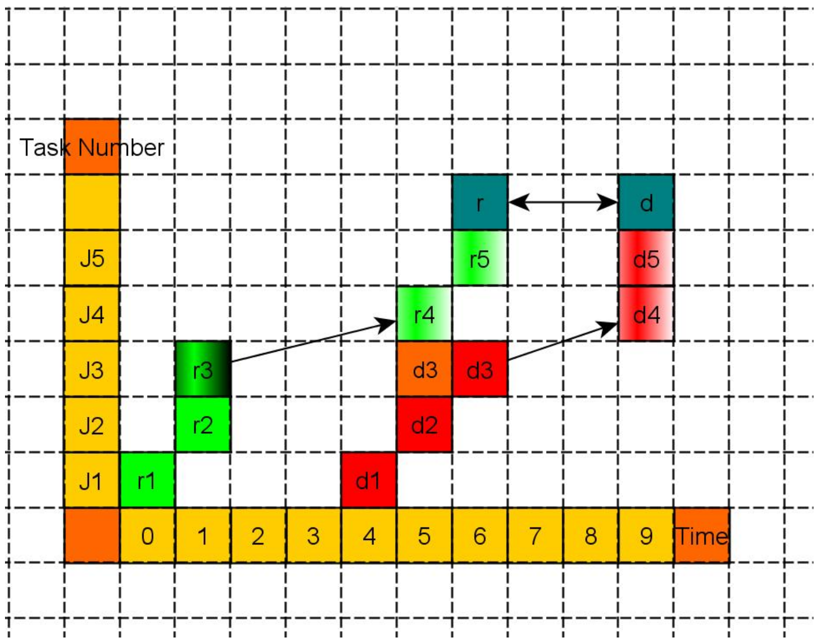

Figure 3.3: First deadline modification

as long as $N(i, r, d)$ remains the same. Next, we observe that the value of $N(i, r, d)$ changes only when a job J_j with deadline $d_j > d$ has its deadline modified to be less than or equal to d. However, d_j can be so modified only when examining $N(j, r, d)$ for some $d \geq d_j > d$. As long as we consider values of d in decreasing order, all such modifications affecting the value of $N(i, r, d)$ are made before $N(i, r, d)$ is examined. Thus we never need to consider a triple $\{i, r, d\}$ more than once when using our loop structure [4].

Now to illustrate the mechanics of the algorithm, let's consider the following problem instance:

$\{r_i\} = \{0, 1, 1, 5, 6\}$

$\{d_i\} = \{4, 5, 6, 9, 9\}$

$\prec = \{3, 4\}$

Since we start from the largest d and wit the job having smallest r_i, the first congestion that the algorithm encounters is the triple $(3, 6, 9)$ (Figure 3.3). With the J_3's predecessor in the interval, the value $N(i, r, d) = 2$ and $N(i, r, d) > d - r - 2 = 1$. So J_3 is pushed to 5 so that J_4 can start processing before 6. The second congestion (Figure 3.4)is $(4, 6, 9)$ where

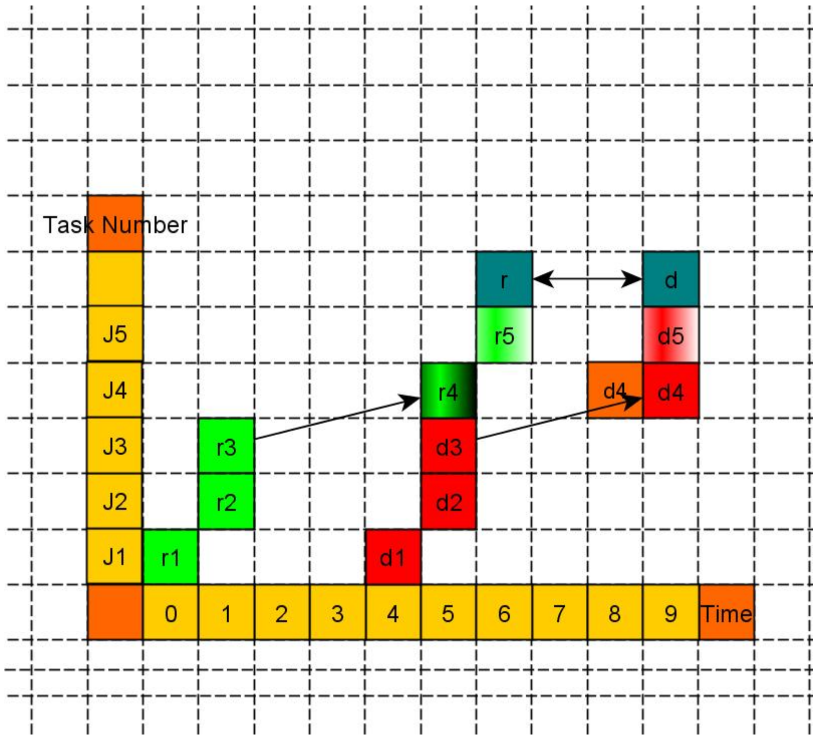

Figure 3.4: Second deadline modification

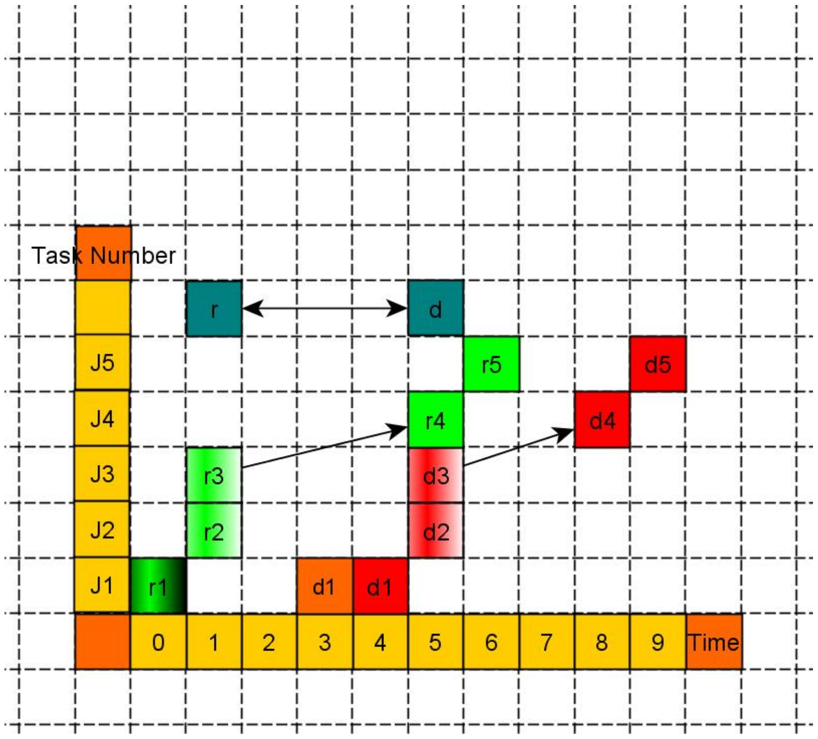

Figure 3.5: Third deadline modification

J_4 is pushed back to allow J_5 to begin processing at time slot 6. The last modification is (Figure 3.5) for the triple $(1, 1, 5)$ which was not visible with the initial deadlines of the instance and emerged after the deadline of J_3 is decreased. The deadline of J_1 is set to 3 so that J_2 and J_3 can be completed in time.

Chapter 4

RELEASE TIME CONGESTION

4.1 Dominance Rule

Since Lemma 1 consistency is not sufficient to provide a feasible schedule, we will continue to find other such rules that might allow us to limit the solution space. We can see that a similar condition that restricts the release times can also be devised if we were to implement our algorithm in a slightly modified manner. The lemma that establishes the second necessary now can be established.

For any job J_i and integers r, d satisfying $r \leq r_i \leq d \leq d_i$; $S'(i, r, d)$ is defined to be the set of all jobs J_j $(j \neq i)$ which have $r \leq r_j$ and either $d_j \leq d$ or $J_j \prec J_i$. Let $N'(i, r, d)$ denote the number of jobs in $S'(i, r, d)$. Now Lemma 2 which justifies certain release time modifications can be stated and proved.

Lemma 2. *Let $(\Im, \prec)$ be a job system with $m = 1$, $s = 3$, and σ be a feasible schedule. Then for any job J_i and integers r, d satisfying $r \leq r_i \leq d \leq d_i$ the following holds:*

If $N'(i, r, d) = d - r - 2$, then J_i must start after $r + N'(i, r, d)$.

If $N'(i, r, d) > d - r - 2$, then J_i must start after $r + N'(i, r, d) + 2$

Proof. **1)** Suppose $N'(i, r, d) = d - r - 2$ and let σ be any feasible schedule. Since all the jobs in $S'(i, r, d)$ have release times that are larger than $r - 1$ and $s = 3$, all jobs in $S'(i, r, d)$ will complete after $r + 2$. If all the jobs in $S'(i, r, d)$ finish no later than d then J_i finishes no sooner than time $d + 1 = N'(i, r, d) + r + 3$ and starts no later than $N'(i, r, d) + r$. Otherwise some job J_j in $S'(i, r, d)$ finishes after d. By the definition of $S'(i, r, d)$, $J_j \prec J_i$ and thus J_i can only start after $d = r + 2 + N(i, r, d)$.

2) Suppose $N'(i, r, d) > d - r - 2$. Then at least $N'(i, r, d) - d + r + 2$ jobs in $S'(i, r, d)$ ends after d. By the definition of $S'(i, r, d)$, J_i succeeds each of these jobs and thus J_i must start after $d + (N'(i, r, d) - d + r + 2) = N'(i, r, d) + r + 2$. ■

This lemma can be extended to m machine case as follows:

Let $(\Im, \prec)$ be a job system with $m = 1$, $s = x$, and σ be a *feasible schedule.* Then for any job J_i and integers r, d satisfying $r \leq r_i \leq d \leq d_i$ the following holds:

1) If $N'(i, r, d) = d - r - x + 1$, then J_i must must start after $r + N'(i, r, d)$.

2) If $N(i, r, d) > d - r - x + 1$, then J_i must start after $r + N'(i, r, d) + x - 1$

As a consequence of Lemma 2, if $N'(i, r, d) = d - r - 2$ and $r_i < N'(i, r, d) + r$, then r_i can be set to $N'(i, r, d) + r$, or if $N'(i, r, d) > d - r - 2$ and $r_i > N'(i, r, d) + r + 2$,then r_i can be set to $N(i, r, d) + r + 2$; and these changes will not alter the set of feasible schedules.

4.2 Release Time Modification Algorithm (RMA)

The algorithm that establishes the Lemma 2 consistency is fairly similar to our deadline modification algorithm. It is practically the application of the deadline modification algorithm to the corresponding problem where all the deadlines and release times are interchanged, in the sense that $r'_i = max(\{d_i\}) - d_i$, $d'_i = max(\{d_i\}) - r_i$ and $T'_i \prec T'k$ if $J_k \prec Ti$. The full algorithm is provided in the appendix. Following problem instance will demonstrate how the algorithm works.

$\{r_i\} = \{0, 0, 3, 4, 4\}$

$\{d_i\} = \{3, 4, 8, 8, 9\}$

$\prec = \{3, 4\}$

Similar to DMA, release time modification algorithm considers triples (i, r, d) but since the modifications will be made to release times, we start by the minimum release time among all jobs in our outermost loop. Considering the jobs in decreasing order with respect to their deadline, we start with J_2 and move to J_4 or J_5 since their deadlines are the same. Our first congestion is at the triple $(4, 0, 4)$. Figure 4.1, Figure 4.2 and Figure 4.3 shows the jobs in the congestions and release time modifications made in consecutive steps of the algorithm in the order they are handled by the algorithm. Like the deadline modification counterpart, congestions that are not initially apparent can arise as the algorithm runs its course.

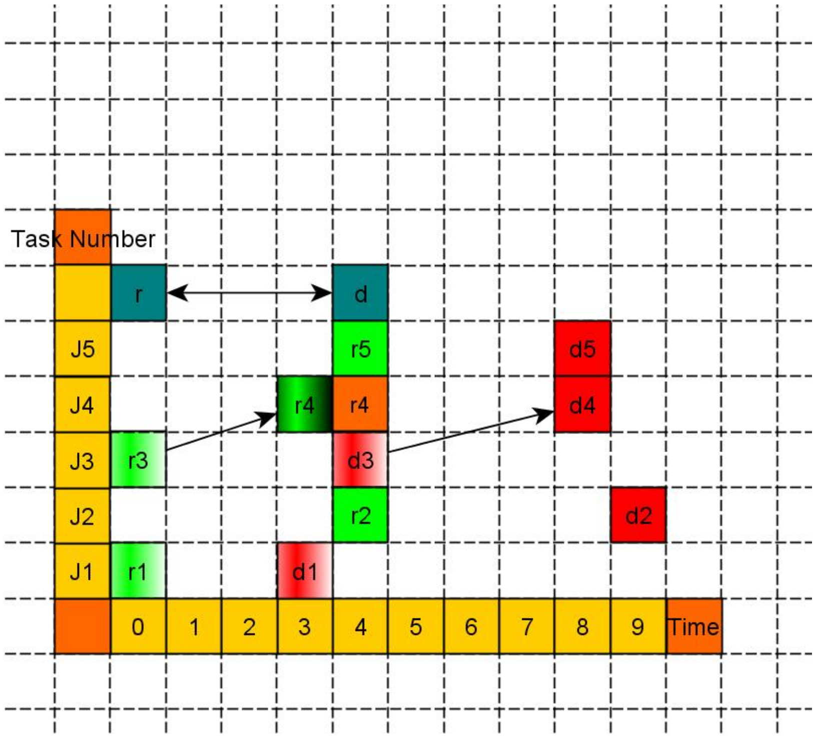

Figure 4.1: First release time modification

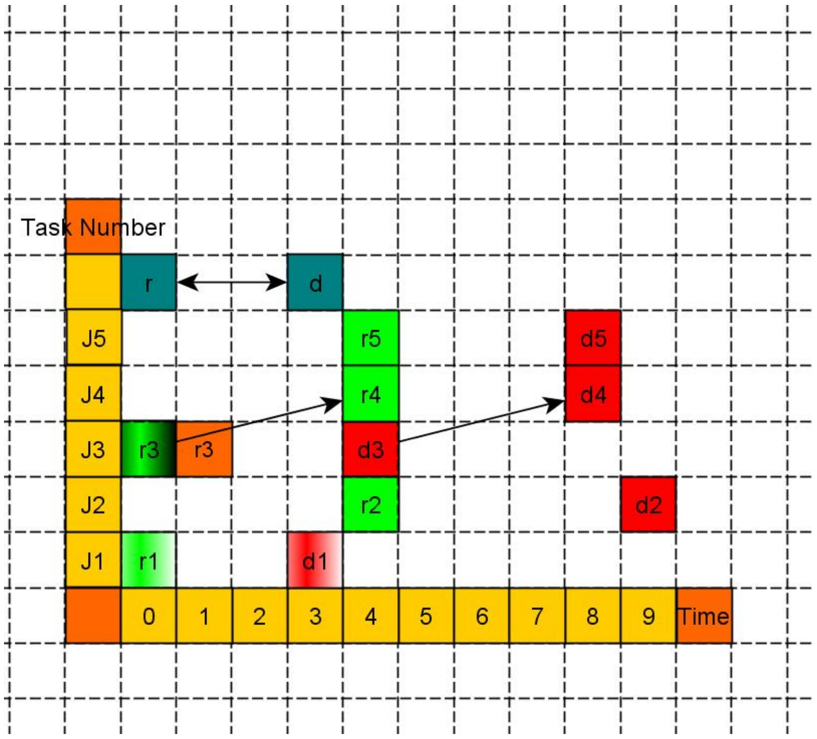

Figure 4.2: Second release time modification

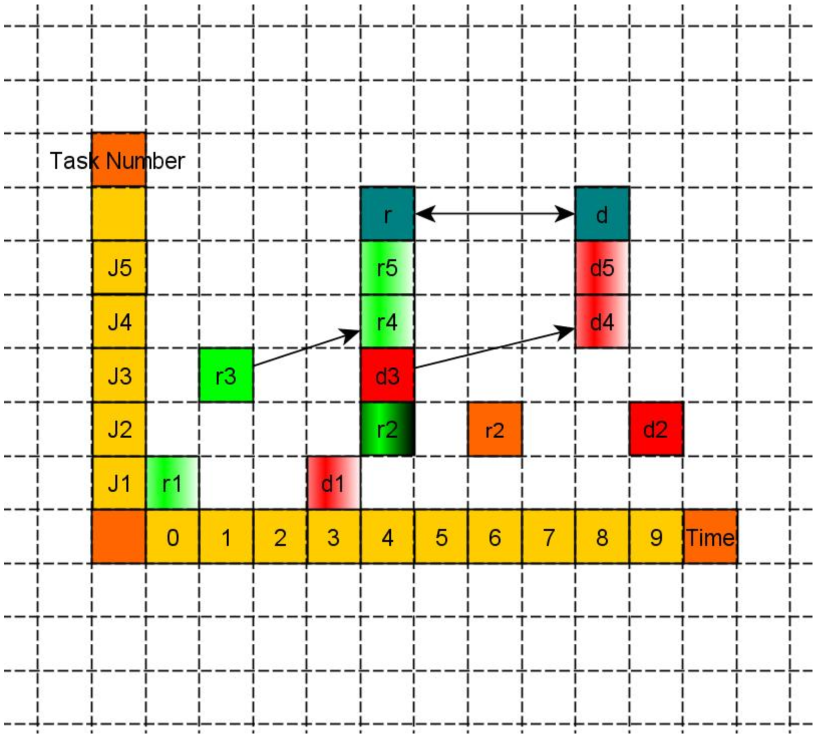

Figure 4.3: Third release time modification

4.3 Combining Deadline and Release Time Consistency

For any job system $(\Im, \prec)$, deadline and release time modification algorithms are shown to run in polynomial time. The algorithms' main idea is to ensure that the Lemmas presented hold for all deadline and release time values. Each algorithm makes the corresponding set of parameters consistent without ruling out any feasible schedule. The question is whether the application of release time modification algorithm can end with inconsistent deadlines and vice versa. The doubt stems from the fact that a once consistent deadline value can become inconsistent if a job which starts earlier is postponed by the release time modification algorithm so that it becomes mandatory that it is initiated after our job. A similar problem also arises for the once consistent release times after the deadlines force a job to be initiated before another. A more elaborate example might be necessary for the illustration of this issue.

Consider the instance with 13 jobs and following $\{r_i\}$,$\{d_i\}$ and $\prec$.

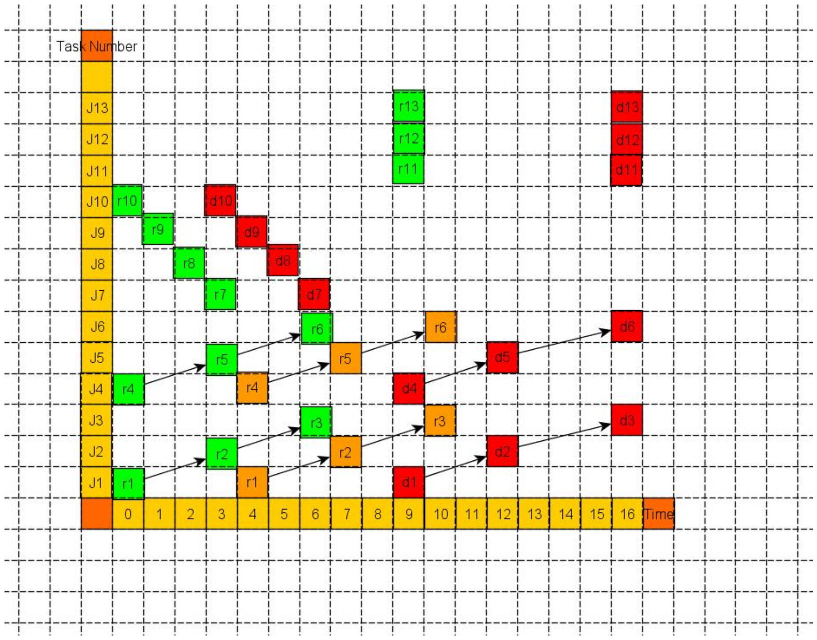

Figure 4.4: Release time modifications after RMA

$\{r_i\} = \{0, 3, 6, 0, 3, 6, 3, 2, 1, 0, 9, 9, 9\}$

$\{d_i\} = \{9, 12, 16, 9, 12, 16, 6, 5, 4, 3, 16, 16, 16\}$

$\prec = \{\{1, 2\}, \{2, 3\}, \{4, 5\}, \{5, 6\}\}$

We can check that the deadlines are initially consistent but the release times are not. During the release time algorithm first congestion is for triple $(5, 0, 6)$ or $(2, 0, 6)$ since their deadlines are equal. This congestion moves r_2 and r_5 to the value of 4 since the time slots 0, 1, 2 and 3 are reserved for J_{10}, J_9, J_8 and J_7. A similar modification then occurs at the next congestions for the triples $(1, 0, 6)$ and $(4, 0, 6)$. Since r_1 and r_4 are also moved to 4 their predecessors must start after 7. After the necessary release time changes are made to comply with precedence relations, the new release times at the end of the release time modification algorithm are (Figure 4.4):

$\{r_i\} = \{4, 7, 10, 4, 7, 10, 3, 2, 1, 0, 9, 9, 9\}$

With the modifications in the release time now for the integers $r = 9$ and $d = 16$, $N(i, r, d) = 6\ \forall i = \{1, 4\}$ since $d - r - 2 = 5$, deadlines of J_1 and J_4 have to be reduced to 8 (Figure 4.5). This modification was not visible before the release time modification

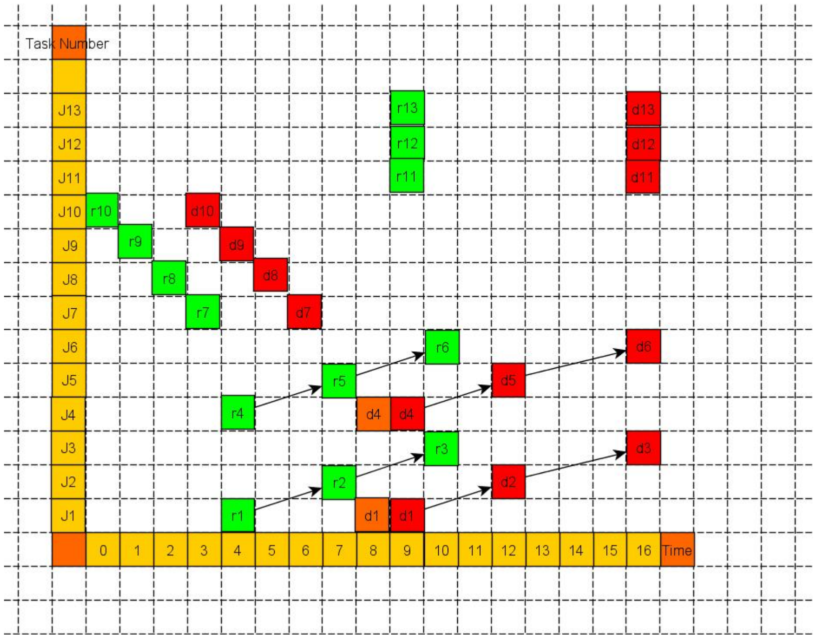

Figure 4.5: Deadline modifications after DMA

algorithm ran its course.

As we can see in the example, the infeasibility which is not visible in the first application of deadline and release time algorithms (in that order) becomes visible when we apply them consecutively. However, unless a bound is set for how many times these algorithms must be applied consecutively we cannot establish a bound on getting a problem consistent for both Lemma 1 and Lemma 2.

Fortunately such a bound can be devised. If we consider any deadline or release time modification move, we see that every time a deadline is modified, it means that all the jobs in corresponding $S(i, r, d)$ are to be initiated after job i. Now consider the changes done to these jobs that constitute $S(i, r, d)$, since we always decrease deadlines and increase release times, we can say that none of those jobs (except for those that succeed job i which are bound to be initiated after job i) will leave the set $S(i, r, d)$ because of a deadline or release time modification. So a new modification to job i will only happen when a job that is not in $S(i, r, d)$ or any other sets that caused a deadline modification to job i is set to be initiated after i. Establishing a loose but sufficient bound is then possible since every job can have its deadline modified in at most n runs of the deadline modification algorithm. Since the same argument is valid for release time modification, release time and deadline modification can be run at most $O(n^2)$ times before one of them completes without making any changes, thus leaving a set of deadlines and release times that are consistent for Lemma 1 and Lemma 2. For the rest of the thesis, the consecutive applications of release and deadline modification algorithms until no change occurs will be referred to as 'Consistency Algorithm' (CA). Apparently Lemma 1 and Lemma 2 consistency, even when established simultaneously is not enough for a feasible schedule. Example in the following chapter illustrates a job system which is consistent but has no feasible solution.

Chapter 5

DISJUNCTIVE AND CONJUNCTIVE CONGESTIONS

Before contemplating on what is sufficient for a feasible solution, it is beneficial to consider the similar problem $F2|prec, r_i, p_{ij} = 1|L_{max}$ once again. We previously mentioned that Lemma 1 consistency is sufficient for this problem but did not discuss why. Now consider a problem $F2|prec, r_i, p_{ij} = 1|L_{max}$ whose deadlines are Lemma 1 consistent and suppose that there is a job that is late after we initiate them according to the earliest due date rule. If there were no open time slots before the first late job it is obvious that the first job initiated had to have its deadline reduced to a number less than the minimum release time among all jobs. If there were an open time slot before our first late job, now we can consider the job that is initiated just before that opening. The time slot remains open in a list schedule either if there are no released jobs or the predecessor of the released jobs is not complete. In a two-machine environment, we can clearly say that the job that is initiated just before the open time slot must be the predecessor of such jobs that are already released. So all the jobs initiated up until the late job create a congestion that should push that predecessor job to an earlier deadline which means that it is late and creates a contradiction by making it the first late job. Now the complexity of the three-machine setup might be more apparent. In a similar case with three-machines, we cannot say that the job that starts before the empty slot is the predecessor of all the released but not initiated jobs since they can be also preceded by the job that is initiated two time slots before the opening. However, another necessary condition can be stated that would solve our problem and actually is sufficient for a feasible schedule by using the list scheduling.

5.1 Dominance Rules

In this dominance rule we will determine what a more inclusive version of set $S(i, r, d)$ would imply about the d_i. By more inclusive, we consider building a new set of solution with respect to a time interval that combines successors of two jobs instead of job. So our notation will transform into $S(i, j, r, d)$, and the implications of a congestion will affect either J_i or J_j For any couple of jobs (J_i, J_j) and integers r, d satisfying $r_i \leq r \leq d_i \leq d$ and $r_j \leq r \leq d_j \leq d$; $S(i, j, r, d)$ is defined to be the set of all jobs J_k $(k \neq i \wedge k \neq j)$ which have $d_k \leq d$ and either $r \leq r_k$ or $J_i \prec J_k$ or $J_j \prec J_k$. Let $N(i, j, r, d)$ denote the number of jobs in $S(i, j, r, d)$. Now a lemma which justifies certain deadline modifications can be stated and proved.

Lemma 3. *Let $(\Im, \prec)$ be a job system with $m = 1$, $s = 3$, and σ be a feasible schedule. Then for any couple of jobs (J_i, J_j) and integers r, d satisfying $r_i \leq r \leq d_i \leq d$ and $r_j \leq r \leq d_j \leq d$, the following holds:*

1) If $N(i, j, r, d) = d - r - 2$, then either J_i or J_j must be completed by $d - N(i, j, r, d)$.

2) If $N(i, j, r, d) > d - r - 2$, then either J_i or J_j must be completed by $d - N(i, r, d) - 2$

Proof. **1)** Suppose $N(i, j, r, d) = d - r - 2$ and let σ be any feasible schedule. Since all the jobs in $S(i, j, r, d)$ have deadlines that do not exceed d and $s = 3$, all jobs in $S(i, j, r, d)$ must start before $d - 2$. If all the jobs in $S(i, j, r, d)$ start no sooner than r then neither J_i nor J_j starts no later than time $r - 1 = d - N(i, r, d) - 3$. Otherwise some job J_k in $S(i, j, r, d)$ starts before r. By the definition of $S(i, j, r, d)$, either $J_i \prec J_k$ or $J_j \prec J_k$ and thus either J_i or J_j is completed by $d - N(i, r, d) = r + 2$.

2) Suppose $N(i, j, r, d) > d - r - 2$. Then at least $N(i, j, r, d) - d + r + 2$ jobs in $S(i, j, r, d)$ start before r. By the definition of $S(i, j, r, d)$, either J_i or J_j precedes each of these jobs and thus either J_i or J_j must be completed by time $r - (N(i, j, r, d) - d + r + 2) = d - N(i, j, r, d) - 2$.

■

Before establishing the concept of Lemma 3 consistency, a corollary must be added for future use.

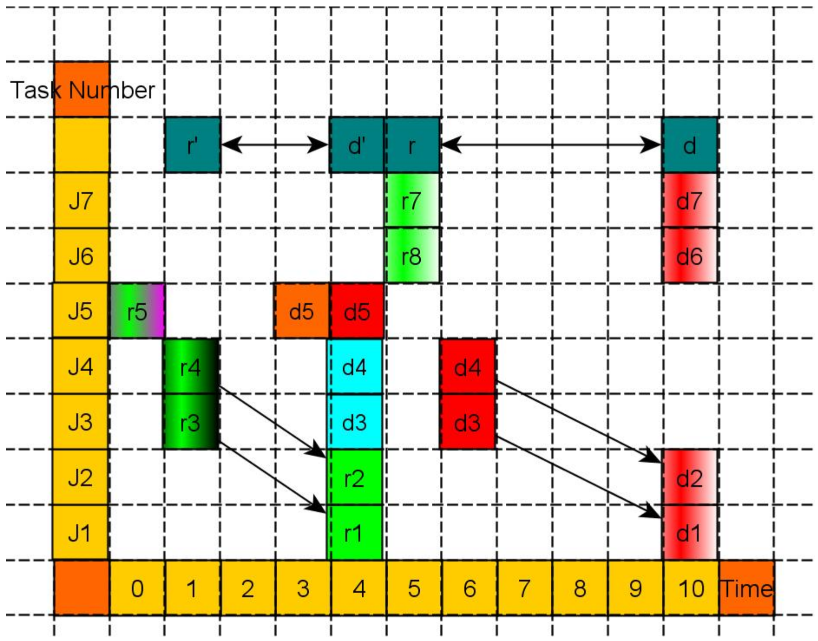

Figure 5.1: A conjunctive congestion

Corollary 1. *Let $(\Im, \prec)$ be a job system with $m = 1$, $s = 3$, and σ be a feasible schedule. Then for any couple of jobs (J_k, J_j) and integers r, d satisfying $r_k \leq r \leq d_k \leq d$, $r_j \leq r \leq d_j \leq d$ and $N(k, j, r, d) \leq d - r - 2$, consider the two set of deadlines and release times $\{d_i\}_k$ $\{r_i\}_k$ and $\{d_i\}_j$ $\{r_i\}_j$, obtained by modifying the deadline of J_k and J_j to comply with Lemma 3 respectively and then using CA to establish Lemma 1 and Lemma 2 consistency. Then the following holds $\forall\ h \in \{1 \ldots n\}$:*

1. $r_h = \min\{r_{hk}, r_{hj}\}$,
2. $d_h = \max\{d_{hk}, d_{hi}\}$.

The above corollary refers to the fact that since either J_k or J_j will have to have their deadlines reduced, coinciding deadline and release time changes can be implemented permanently. In Figure 5.1, disjunctive modifications are shown in blue while the conjunctive modification which is the deadline change that occurs no matter which alternative is chosen in disjunctive congestion, is shown in purple shaded green. Even if the actual values of the new deadlines and release times are different, the modification that caused the smaller change holds for any feasible schedule. Moreover, all the deadlines can be modified in polynomial time to comply with this corollary. The following algorithm only considers deadline changes that can be made permanently and propagates them as it visits different congested pairs sequentially.

5.2 Conjunctive Deadline Modification Algorithm (CDMA)

1. If any job J_i has $d_i \leq r_i$, halt (no feasible schedule is possible). If no job has a deadline less than d, halt (the current deadlines are internally consistent). Otherwise set d to the largest deadline less than d and set i and j to the least job index for which d_i is less than or equal to the new value of d.

2. Scan the job list to compute $N(i, j, r_j, d)$. Set $COUNT \leftarrow N(i, j, r_j, d)$ $r \leftarrow r_j$, and set $k \leftarrow$ least index k such that $r_k = r_j$.

3. If $COUNT = d - r - 2$ and $d_i > d - COUNT$ and $d_j > d - COUNT$:

   (a) If$(r_i \geq d - COUNT - 2) \wedge (r_j \geq d - COUNT - 2)$ then halt, no feasible schedule is possible.

   (b) If$(r_i \geq d - COUNT - 2)$ XOR $(r_j \geq d - COUNT - 2)$ set $d_k = d - COUNT$ such that $k \in \{i, j\} \wedge r_k < d - COUNT - 2$. Run DMA to achieve Lemma 1 consistency and perpetuate the new deadlines.

   (c) If $(r_i < d - COUNT - 2) \wedge (r_j < d - COUNT - 2)$ then store current deadlines, set $d_i = d - COUNT$ and run DMA to obtain $\{d_k\}_i$. Set all deadlines to their stored values so that the other alternative can be evaluated, set $d_j = d - COUNT$ and run DMA to obtain $\{d_k\}_j$. Once these two set of deadlines are obtained, set all $d_k = \max\{d_{ki}, d_{kj}\}$

4. If $COUNT \geq d - r - 2$ and $d_i > d - COUNT - 2$ and $d_j > d - COUNT - 2$:

   (a) If $(r_i \geq d - COUNT - 4) \wedge (r_j \geq d - COUNT - 4)$ then halt, no feasible schedule is possible

(b) If $(r_i \geq d - COUNT - 4)$ XOR $(r_j \geq d - COUNT - 4)$ set $d_k = d - COUNT - 2$ such that $k \in \{i, j\} \wedge r_k < d - COUNT - 4$. Run DMA to achieve Lemma 1 consistency and perpetuate the new deadlines. If DMA returns infeasible, the problem is infeasible.

(c) If $(r_i < d - COUNT - 4) \wedge (r_j < d - COUNT - 4)$ then store current deadlines, set $d_i = d - COUNT - 2$ and run DMA to obtain $\{d_k\}_i$. Set all deadlines to their stored values so that the other alternative can be evaluated, set $d_j = d - COUNT - 2$ and run DMA to obtain $\{d_k\}_j$. Once these two set of deadlines are obtained, set all $d_k = \max\{d_{ki}, d_{kj}\}$

5. If $r \geq d_i$ or $r \geq d_j$ then go to Step 6. Otherwise increment k by 1 until either $k > n$ or $r_k > r$. During this scan, decrease $COUNT$ by one for each J_h (original $k \leq h <$ new k) which is not a successor of J_i or J_j and which satisfies $(d_h \leq d \wedge r_h = r \wedge h \neq i \wedge h \neq j)$. If $k = n + 1$ or $r_k > d_i$ or $r_k > d_j$, set $r \leftarrow min(d_i, d_j)$ and go to Step 4. Otherwise set $r = r_k$ and go to Step 4.

6. Find the least $h > j$ such that $d_h \leq d$. If such a h exists, set $j \leftarrow h$ and go to Step 2. If no such h exists find the least $h > i$ such that $d_h > d$. If such a h exists set $i \leftarrow h$ $j \leftarrow h$ and go to Step 2. If no such h exists either and some job has current deadline d, go to Step 1. Otherwise halt (no feasible schedule is possible).

Since every quadruple (i, j, r, d) is considered only once and at the worst-case DMA is called for each quadruple, time complexity of CDMA is bounded by $O(n^7)$.

The following problem, which is consistent with respect to Lemma 1 and Lemma 2 is actually infeasible and this infeasibility can be detected by conjunctive deadline modification algorithm. In the first step of the algorithm (Figure 5.2), J_1 and J_2 are determined to be in a disjunctive congestion over the time interval $[10, 14]$. Since either of them will have their deadline to be decreased to 9, considering the interval $[6, 9]$ the deadline of J_{10} must be decreased to 8. In the second step of the algorithm (Figure 5.3) the job couple J_5 and J_7 is in a disjunctive congestion over the interval $[5, 8]$ and their modified deadline will be

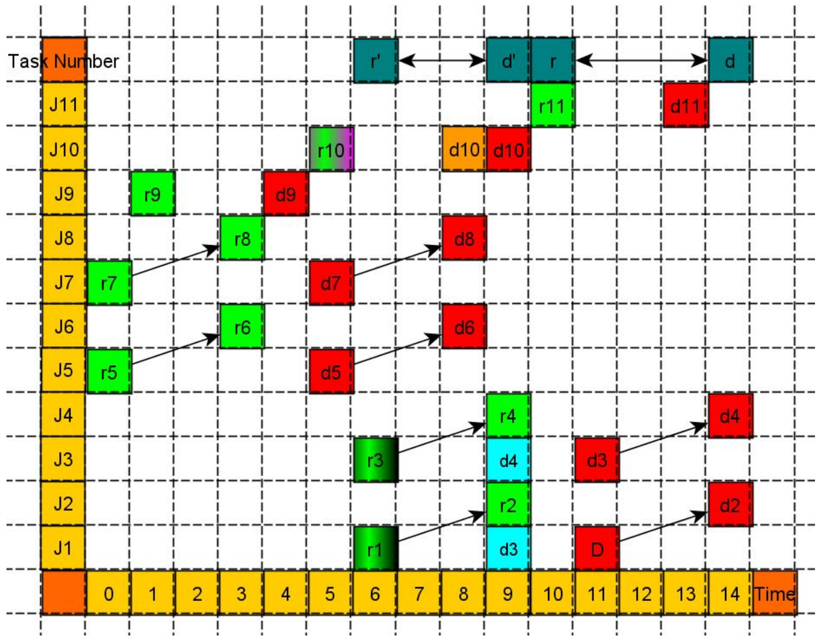

Figure 5.2: First modification of CDMA

7 which in turn pushes back the deadline of their predecessors J_5 and J_7 to 4. Since the interval $[0, 4]$ is now congested no matter which job is selected to relieve the disjunctive congestion, J_9 has its deadline reduced to 2, which makes the problem infeasible.

Conjunctive deadline modification algorithm runs in polynomial time like the original DMA algorithm since the changed deadlines do not affect the previously visited congestions. However, we cannot make such a claim for a version of CDMA which uses CA instead of DMA since the changed release times would effect the consistency of all the jobs. Even though we have not discussed the release time equivalent of Lemma 3, it can be seen that such a necessary condition can also be established by combining the predecessors of two jobs and finding a release time restriction that must be complied by at least one of them. Respectively, a conjunctive algorithm for release times can be implemented with the use of RMA instead of DMA. Similarly these two algorithms can be implemented consecutively until no further change occurs, leaving the problem Lemma 3 consistent for both deadlines and release times. Such an algorithm, as its simple counterpart CA does, finishes in polynomial number of implementation of release and deadline counterparts since the changes

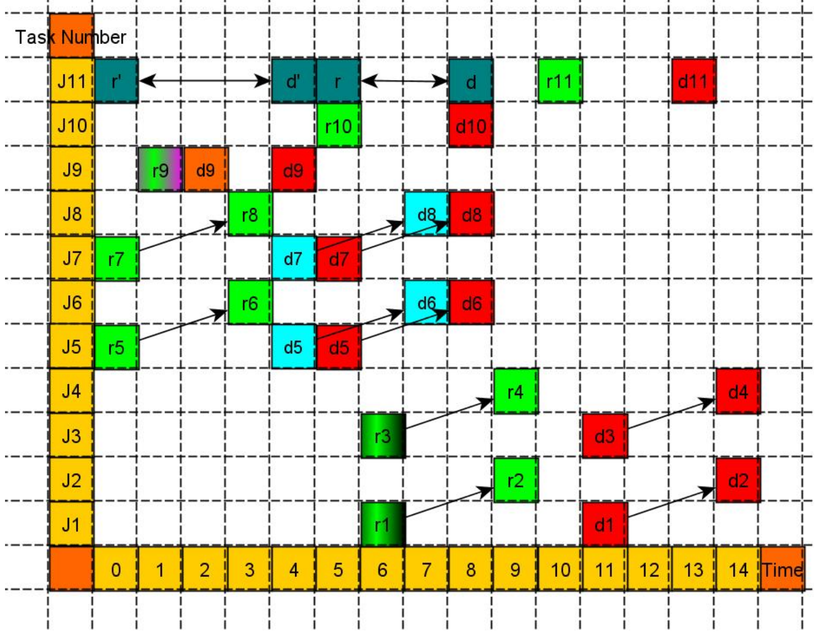

Figure 5.3: Second modification of CDMA

they make are permanent and limited in number by $O(n^2)$. This combined algorithm will henceforth be referred as "Conjunctive Consistency Algorithm" (CCA).

Chapter 6

SUFFICIENT CONDITIONS FOR A FEASIBLE SOLUTION

In this chapter, we will establish that a problem whose jobs are all Lemma 3 consistent can be solved by a list scheduling method. Moreover, an algorithm which uses previous algorithms to establish Lemma 3 consistency will be provided without proof of completeness.

6.1 Proof of Sufficiency

We say that $\{d_i\}$ and $\{r_i\}$ are Lemma 3 consistent whenever the following conditions hold for every job $(J_i, J_j) \in \Im \times \Im$:

For every pair of integers r, d satisfying $r_i \leq r \leq d_i \leq d$ and $r_j \leq r \leq d_j \leq d$

1. $d_i \geq r_i + 3$ and $d_j \geq r_j + 3$;

2. if $N(i, j, r, d) = d - r - 2$, then either $d_i \leq d - N(i, j, r, d)$ or $d_j \leq d - N(i, j, r, d)$;

3. if $N(i, j, r, d) > d-r-2$, then either $d_i \leq d-N(i, j, r, d)-2$ or $d_j \leq d-N(i, j, r, d)-2$.

Theorem 1. *Let $(\Im, \prec)$ be a job system with $m = 1$, $s = 3$. Let $\{d_i\}$ and $\{r_i\}$ be internally consistent with respect to Lemma 3 and L be an increasing list with respect to $\{d_i\}$. Then every schedule obtained by applying list scheduling with respect to L is feasible.*

Proof. Assume $\{d_i\}$ and $\{r_i\}$ are internally consistent and L is an increasing list with respect to $\{d_i\}$. Let σ be a schedule obtained by using list scheduling with respect to L. Choose a job J_i which exceeds its deadline and such that $\sigma(J_i)$ is as small as possible. Let t_0 be the largest positive integer less than $\sigma(J_i)$ such that either no job is initiated at t_0 or the deadline of the job initiated at t_0 is greater than d_i. If t_0 does not exist, set $t_0 = 0$. Let U be the set of jobs initiated at times $t_0 + 1, t_0 + 2, \ldots, \sigma(J_i)$. Set $d = \sigma(J_i) + 2$; then $d_i \leq d$.

If we assume that the release times of all the jobs in U are greater than or equal to $t_0 + 1$ then we can use Lemma 1 to show that $\{d_i\}$ and $\{r_i\}$ could not be internally consistent (simply consider $N(k, t_0 + 1, d)$ where $\sigma(J_k) = t_0 + 1$).) Otherwise, suppose there is a job $J_k \in U$ with a release time less than $t_0 + 1$. Then, $J_j \prec J_k$, where either $\sigma(J_j) = t_0 - 1$ or $\sigma(J_j) = t_0 - 2$. In this step we divide our proof into two cases:

1. There are two jobs J_j and J_m scheduled to start at $t_0 - 1$ and $t_0 - 2$:

   $U \subseteq S(j, m, t_0 + 1, d)$ and thus $N(j, m, t_0 + 1, d) \geq |U| \geq d - t_0 - 2 > d - t_0 - 3$. By Lemma 3 either $d_j \leq d - N(m, j, t_0 + 1, d) - 2 < to + 1$ or $d_m \leq d - N(m, j, t_0 + 1, d) - 2 < to + 1$, which contradicts the assumption that J_i is the earliest job that does not meet its deadline.

2. There is only one job J_j scheduled to start at $t_0 - 1$ or $t_0 - 2$:

   $U \subseteq S(j, t_0 + 1, d)$ and thus $N(j, t_0 + 1, d) \geq |U| \geq d - t_0 - 2 > d - t_0 - 3$. By Lemma 1 $d_j \leq d - N(j, t_0 + 1, d) - 2 < to + 1$, which contradicts the assumption that J_i is the earliest job that does not meet its deadline.

■

This proof of sufficiency is applicable for any fixed number of machines. However, the generalized version requires consistency with respect to the following lemma instead of Lemma 3.

For any x-tuple of jobs $J* = (J'_1, J'_2, \ldots, J'_x)$ and integers r, d satisfying $r_i \leq r \leq d_i \leq d$ $\forall i = 1 \ldots x$; $S(J^*, r, d)$ is defined to be the set of all jobs J_k $J_(k) \notin J^*$ which have $d_k \leq d$ and either $r \leq r_k$ or $\exists J_i \in J^*$ such that $J_i \prec J_k$. Let $N(J^*, r, d)$ denote the number of jobs in $S(i, j, r, d)$.

Lemma 4. *Let $(\Im, \prec)$ be a job system with m = 1, $s = x$, and σ be a feasible schedule. Then for any $(x - 1)$-tuple of jobs $J* = (J'_1, J'_2, \ldots, J'_{x-1})$ such that $J'_i \in \Im$ and integers r,*

d satisfying $r'_i \leq r \leq d'_i \leq d \; \forall i = 1 \dots (x-1)$ where r'_i and d'_i are release time and deadline of job J'_i respectively.:

1) *If $N(J^*, r, d) = d - r - x + 1$, then at least one member J'_i of J^* must be completed by $d - N(J^*, r, d)$.*

2) *If $N(J^*, r, d) > d - r - x + 1$, then at least one member J'_i of J^* must be completed by $d - N(J^*, r, d) - x + 1$.*

6.2 Disjunctive Deadline Modification Algorithm

With this algorithm we seek to find a closure on rule stated in Lemma 3. Our problem now involves only one disjunction for every $N(i, j, r, d)$ and this disjunction is characterized by the choice of which job's deadline will be modified to comply with Lemma 3. The algorithm does not only update the deadlines but also creates choice points whenever we choose any job involved in a congested disjunction to satisfy the deadline requirements. Our assumption is that a constant number of backtracking move along these choice points will be enough to determine whether the problem is infeasible or not. The backtracking algorithm is inspired by the solution of the $2 - SAT$ problem. In [3], a search algorithm that assigns truth values to literals and checks whether the assignment causes any clause to be false is used to come up with a truth assignment literal by literal for the 2-SAT problem. The idea of limiting the backtracking is of importance since if every combination of alternatives is to be checked, the algorithm would be exponential in time. This means that every time the deadline of a job is updated to satisfy the requirements of Lemma 3, we will check whether our choice triggers an inevitable inconsistency by running a CDMA and when that is the case we will reverse our choice at the last choice point created. When we cannot find such an inconsistency, we will skip to the next congested disjunction and create a new choice point and the previous choice will be permanent. Since we will consider all quadruples (i, j, r, d) and at worst-case will run CDMA for each of them, DDMA's time complexity is bounded by $O(n^1 1)$. Same preprocessing done in DMA is used and the main body of the algorithm is assumed to start after the completion of CCA. d is reset to a number larger than largest deadline.

Whenever a choice point is created, the current values for the variables i, j, r, d, $COUNT$, $CONG$ and all deadline values are stored, and when a reverse is invoked all the variables are set to their stored values except for d_j which is set to the value of the stored variable $CONG$ and the variable $reverse$ is set to 1 in order to signify that a back-tracking has occurred and a further inconsistency will mean that the problem is infeasible. After d_j is set to $CONG$, deadlines of the jobs in $\Im$ which precedes J_j are also updated to ensure that all of them are smaller than $d_j - 3$. The algorithm skips to Step 5 regardless of where the reversing occurred in the algorithm.

1. If any job J_i has $d_i \leq r_i$ and $reverse = 1$, halt (no feasible schedule is possible). If any job J_i has $d_i \leq r_i$, and $reverse = 0$, set $reverse \leftarrow 1$ and reverse all the deadline changes made since the last choice point and go to Step 5. If no job has a deadline less than d and $reverse = 0$, set $reverse \leftarrow 1$ and set d to a value greater than the largest deadline and set i and j to the least job index for which d_i is less than or equal to the new value of d. If no job has a deadline less than d and $reverse = 1$, halt (the current deadlines are internally consistent). Otherwise set d to the largest deadline less than d and set i and j to the least job index for which d_i is less than or equal to the new value of d.

2. Scan the job list to compute $N(i, j, r_j, d)$. Set $COUNT \leftarrow N(i, j, r_j, d)$ $r \leftarrow r_j$, and set $k \leftarrow$ least k such that $r_k = r_j$.

3. If $COUNT = d - r - 2$ and $d_i > d - COUNT$ and $d_j > d - COUNT$:

   (a) If $(r_i \geq d - COUNT - 2) \wedge (r_j \geq d - COUNT - 2) \wedge reverse = 1$ then halt, no feasible schedule is possible.

   (b) If $(r_i \geq d - COUNT - 2) \wedge (r_j \geq d - COUNT - 2) \wedge reverse = 0$ then $reverse = 1$ and set $d_{Cj} = CONG$. For each $J_k \prec J_h$ whose deadline exceeds

$d_h - 3$, set $d_k \leftarrow d_h - 3$. Run CDMA. If CDMA returns infeasible, halt the problem is infeasible.

(c) If only one of $(r_i \geq d - COUNT - 2)$ and $(r_j \geq d - COUNT - 2)$ is true, set $d_k = d - COUNT$ such that $k \in \{i, j\} \wedge r_k < d - COUNT - 2$. Run CDMA. If CDMA returns infeasible, the problem is infeasible.

(d) If $(r_i < d - COUNT - 2) \wedge (r_j < d - COUNT - 2) \wedge reverse = 1$ then create a choice point. Set $Cd \leftarrow d$, $Cr \leftarrow r$, $Ci \leftarrow i$, $Cj \leftarrow j$, $Cd_m \leftarrow d_m \; \forall J_m \in \Im$, $d_i \leftarrow d - COUNT$, $reverse \leftarrow 0$, $CCOUNT \leftarrow COUNT$, $CONG = d - COUNT$. For each $J_k \prec J_i$ whose deadline exceeds $d_i - 3$, set $d_k \leftarrow d_i - 3$.

4. If $COUNT > d - r - 2$ and $d_i > d - COUNT - 2$ and $d_j > d - COUNT - 2$:

(a) If $(r_i \geq d - COUNT - 4) \wedge (r_j \geq d - COUNT - 4) \wedge reverse = 1$ then halt, no feasible schedule is possible.

(b) If $(r_i \geq d - COUNT - 4) \wedge (r_j \geq d - COUNT - 4) \wedge reverse = 0$ then $reverse = 1$ and set $d_{Cj} = CONG$ For each $J_k \prec J$ whose deadline exceeds $d_i - 3$, set $d_k \leftarrow d_i - 3$.

(c) If only one of $(r_i \geq d - COUNT - 4)$ and $(r_j \geq d - COUNT - 4)$ is true, set $d_k = d - COUNT - 2$ $s.t.$ $k \in \{i, j\} \wedge r_k < d - COUNT - 4$. Run CDMA. If CDMA returns infeasible, the problem is infeasible.

(d) If $(r_i < d - COUNT - 4) \wedge (r_j < d - COUNT - 4) \wedge reverse = 1$ then create a choice point. Set $Cd \leftarrow d$, $Cr \leftarrow r$, $Ci \leftarrow i$, $Cj \leftarrow j$, $Cd_m \leftarrow d_m \; \forall J_m \in \Im$, $d_i \leftarrow d - COUNT$, $reverse \leftarrow 0$, $CCOUNT \leftarrow COUNT$, $CONG = d - COUNT - 2$. For each $J_k \prec J_i$ whose deadline exceeds $d_i - 3$, set $d_k \leftarrow d_i - 3$.

5. If $r \geq d_i$, go to Step 6. Otherwise increment k by 1 until either $k > n$ or $r_k > r$. During this scan, decrease $COUNT$ by one for each J_h(original $k \leq h <$ new k) which is not a successor of or J_j and which satisfies $(d_h \leq d \wedge r_h = r \wedge h \neq i \wedge h \neq j)$. If $k = n + 1$ or $r_k > d_i$ or $r_k > d_j$, set $r \leftarrow min(d_i, d_j)$ and go to Step 4. Otherwise set $r = r_k$ and go to Step 4.

6. Find the least $h > j$ such that $d_h \leq d$. If such a h exists, set $j \leftarrow h$ and go to Step 2. If no such h exists, find the least $h > i$ such that $d_h > d$. If such a h exists, set $i \leftarrow h$ $j \leftarrow h$ and go to Step 2. If no such h exists either and some job has current deadline d, go to Step 1. Otherwise, if $reverse = 0$, reverse changes since last choice point, if $reverse = 1$, halt (no feasible schedule is possible).

6.3 Computational Evaluations

Due to the lack of theoretical proof for the completeness of DDMA algorithm, the algorithm is run against random instances of the problem. With this experiments we wanted to point out that the cases where CDMA turns out to be feasible and DDMA turns out to be infeasible are very rare or none existent. 5000 instances of 25, 30 and 35 jobs each are used. Since their infeasibility results are binding, DMA and CDMA was implemented before DDMA and DDMA is used only for instances that both DMA and CDMA found to be feasible.

While creating the instances, we first determined the number of chains. Then we determined the release times of the first jobs in the chains according to the number of its successors. Succeeding jobs' release times are created consecutively regarding the number of remaining jobs in the chain. Deadlines then generated according to a jobs release time and its immediate predecessors deadline.

Unfortunately, during the computational experiments that span 15000 instance we have seen that 92% percent of the instances are found to be infeasible in the first stage (DMA check). The remaining instances were found to be feasible by both CDMA and DDMA. This results form computational experiments showed that a random instance generation is not a valid method for assessing the performance of the algorithms since even though DMA feasible instances can be CDMA infeasible, no such instances were observed during experimentation. This absence in return makes the results we obtain from computational experiments unreliable though the results are in favor of the DDMA algorithm.

Chapter 7

CONCLUSION

In this thesis we analyzed the $F3|p_{ij} = 1, chains, r_i|L_{max}$ problem since its classification status with respect to computational complexity is open. Our analysis resulted in several necessary and sufficient conditions for obtaining a feasible schedule for the decision version of the problem. These conditions are then developed to a limited backtracking algorithm whose output is a set of deadlines that can be used to find a feasible schedule.ped to a limited backtracking algorithm whose output is a set of deadlines that can be used to find a feasible schedule.

7.1 Contributions

The contributions of this thesis can be given under the two following heading:

7.1.1 Dominance Rules

In this thesis several dominance rules to construct a feasible solution to the decision version of $F3|chains, r_i, p_{ij} = 1|L_{max}$ are proposed. The concept of congested time intervals that are used in [2] to restrict deadlines is adjusted for the three-machine case. We further observe that release times of the jobs are also subject to similar restrictions. The idea of congestion is then extended to the case where successors of two jobs are combined to find additional deadlines that must comply by at least one of the jobs. The idea of disjunctive congestion has lead to the conjunctive disjunctions where any choice made resolving disjunctive congestions causes the same deadline or release time modifications that can be perpetuated without ignoring possible feasible solutions.

Apart from being a general analysis of the problem, the dominance rules have further significance since $F3|chains, r_i, p_{ij} = 1|L_{max}$ is still an open problem. Given an instance of the problem, it is possible to find in polynomial time, an equivalent instance that complies with all the dominance rules mentioned in the thesis (except for disjunctive congestions). Since that new instance has the same set of feasible solutions, finding a feasible solution or proving that none exists is sufficient for solving the original problem. So, these dominance rules can constitute the basis of a polynomial algorithm that will be devised for $F3|chains, r_i, p_{ij} = 1|L_{max}$.

7.1.2 Disjunctive Algorithm

Among the dominance rules discussed only one is sufficient for finding a polynomial algorithm that provides a feasible solution. Disjunctive congestions, however, are intrinsically harder to satisfy throughout the problem since alternative choices add up and the number of alternatives to check becomes exponential with respect to the number of jobs. We have devised a limited backtracking algorithm that in each step chooses one disjunctive congestion and creates a choice point by arbitrarily choosing one of the jobs involved in the congestion to be modified. If that particular choice satisfies other dominance rules then its new value becomes permanent. If an infeasibility occurs, we simply reverse our choice and go to another disjunction to create a new choice point.

This algorithm completes with a set of deadlines that can be used in a list scheduling algorithm and provides a feasible schedule. Since the number of times a backtracking can occur is restricted by the number of job pairs, this algorithm completes in $O(n^1 1)$. This algorithm if proven to cover all possible feasible solutions places $F3|chains, r_i, p_{ij} = 1|L_{max}$ in P .

However, such a proof cannot be devised during the research phase of this thesis. A counter-example cannot be constructed either. The reason is either because the algorithm works effectively or because finding counter-examples becomes increasingly difficult with the addition of conjunctive deadline modification algorithms to the verification of choices. A possible counter-example now must contain a disjunctive congestion such that modification of any of the job leads to an infeasibility, but it does so through affecting different jobs in the system. If such an example does not exist, it must be noted that the conjunctive deadline modification algorithm is actually enough to assess whether there exists a feasible schedule or not and the disjunctive algorithm is only used to modify the deadlines so that a list schedule is feasible.

7.2 Future Research

The next step for classifying $F3|chains, r_i, p_{ij} = 1|L_{max}$ is to find a counter-example for the disjunctive algorithm or proving that it covers all solutions. However, if such a proof is achieved then it can be extended to $F3|prec, r_i, p_{ij} = 1|L_{max}$ if any restriction of chain precedence relation is not used for the proof. In that case the natural next step will be $F4|chains, r_i, p_{ij} = 1|L_{max}$. Even though all the dominance rules will have a counterpart for the four-machine version, since the sufficient condition will now involve a disjunctive congestion that involves three jobs, the problem will be harder. In that case a limited back-tracking might not be sufficient since limited backtracking, while sufficient for a polynomial solution of $2 - SAT$ problem, fails to find a solution for $3 - SAT$ problem. Moreover since $3 - SAT$ problem is NP-complete, three-machine case might very well be the breaking point for the classification where problems with larger number of machines are intractable for pipelined processor with chain constraints and release times.

In the case that a counter-example is found, some effort might be put in finding an NP-completeness result. A starting point for that research might be a polynomial reduction to the weighted $2 - SAT$ problem. Weighted $2 - SAT$ problem is a relatable problem since to resolve the necessary disjunctions in our problem we bring forward deadlines and there might be a instance structure that only allows a limited number of jobs to be brought forward. If the NP-hardness result for the $F3|chains, r_i, p_{ij} = 1|L_{max}$ problem can be achieved, then the depth of backtracking has also the potential to be used as an approximation scheme since it is actually what causes a complete algorithm to be exponential.

BIBLIOGRAPHY

[1] P. Brucker and S. Knust. Shortest path to nonpreemptive schedules of unit-time jobs on two identical parallel machines with minimum total completion time. *Computing*, 63:299 – 316, 1999.

[2] J. Bruno, J. W. Jones, and K. So. Deterministic scheduling with pipelined processors. *IEEE Transactions on Computers*, C-29(4), 1980.

[3] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4), 1976.

[4] M. R. Garey and D. S. Johnson. Two-processor scheduling wit start-times and deadlines. *SIAM Journal of Computing*, 6(3):416 – 425, 1977.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[6] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.*, 1(2):117 – 129, 1976.

[7] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Ann. of Discrete Math.*, 1:343 – 362, 1977.

[8] J.Y.-T. Leung, J.D. Witthoff, and O. Vornberger. On some variants of the bandwidth minimization problem. *SIAM Journal of Computing*, 13(13):650 – 667, 1984.

[9] V.S. Tanaev, Y.N. Sotskov, and V.A. Strusevich. Scheduling theory. multi-stage systems. *Mathematics and its Applications. Kluwer Academic Publishers Group, Dordrecht*, 1994.

[10] V.G. Timkovsky. Identical parallel machines vs. unit-time shops and preemptions vs. chains in scheduling complexity. *European J. Oper. Res.*, 149(2):355 – 376, 2003.

[11] A.M. Turing. On computable numbers, with an application to the entscheidungs problem. *Proceedings of the London Mathematical Society*, 2(42):230 – 265, 1936.