NEW DOMINANCE ORDERS FOR SCHEDULING PROBLEMS

By

PAUL A. STEPHENSON, M.B.A., M.Sc., B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree

Doctor of Philosophy

McMaster University

NEW DOMINANCE ORDERS FOR SCHEDULING PROBLEMS

DOCTOR OF PHILOSOPHY (1999) (Business Administration)	McMaster University Hamilton, Ontario
TITLE:	New Dominance Orders for Scheduling Problems
AUTHOR:	Paul A. Stephenson M.B.A., M.Sc., B.Sc. (McMaster)
SUPERVISOR:	Dr. George Steiner
NUMBER OF PAGES:	ix, 95

Abstract

A sequencing problem involves finding a sequence in which to process a set of jobs that minimizes a given cost function. The primary difficulty with such problems is that as the number of jobs increases the number of sequences grows astronomically large, and the problems become intractable.

Pairwise job interchange is one of the most commonly used solution techniques for sequencing problems. It compares the cost of sequences that differ only in the interchange of two jobs. In this way the cost function indicates a preference for the ordering of certain pairs of jobs such that if a pair of jobs is not in preference order, then the jobs can be interchanged with no increase in cost. The traditional method of pairwise job interchange assumes that either there are no intermediate jobs (*adjacent pairwise interchange*) or that an interchange can be performed no matter what the intermediate jobs are (*nonadjacent pairwise interchange*). In this thesis we introduce a generalization that permits the pairwise interchange of a pair of jobs provided that the intermediate jobs belong to a restricted subset of jobs (*subset-restricted pairwise interchange*).

We use subset-restricted pairwise interchange to derive a dominance order on the jobs. This is a partial ordering of the jobs that consists of pairs whose relative order can be fixed in an optimal sequence. The search space can then be reduced to consist of only those sequences that satisfy the relative job orderings in the dominance order. We apply this technique to certain one- and two-machine sequencing problems, and show that the use of our dominance orders significantly reduces the computation time necessary to solve these problems.

Acknowledgements

I would like to gratefully acknowledge the tremendous assistance I have received from my supervisor Dr. George Steiner. This assistance has ranged from, correcting my grammar on numerous occasions, to 'stretching the truth' on even more numerous progress reports. I thank you for introducing me to the subject of Scheduling, your extensive knowledge, and your (almost) infinite patience.

I would like to acknowledge the many friends who have made my experience as a graduate student at McMaster memorable. I want to thank John Harding for his early programming assistance and for the many trips to Brandon University to work on this material. Thanks are also due to Sam and Mary for allowing us to get in some much needed table-hockey on these trips. I would also like to thank my very good friends from the Mathematics Department: Spiro Daoussis, Bob Stamičar, Mike Pfau, Tarit Saha, and Eric Derbez. You guys belong in the circus...right next to the dog-faced-boy (Franco Guerriero). From the Business School I want to thank: Yavuz Günalay, Kalyan Gupta, Brian Denton, Halit Üster, Laurent Lapierre, Kaywan Afkhamie, Mustafa Karakul, and Erdem Gündüz. The worst softball team deserves the worst captain, thanks guys. Thanks also to some of the friends who helped to make the Phoenix my second home: Russ Riddel, Tammy Ivanco, Jonathan Wilson, Marcy Lumsden, and Chris O'Donovan.

I would especially like to express my gratitude to Susan Larson. Her patience and support during the difficult times will never be forgotten.

I also want to acknowledge Karmen Bleile for her continued companionship.

Finally, I am very grateful to my family for their many years of support and encouragement. I'm glad that we're all around to see this day.

Contents

List of figures

List of tables

Chapter 1 Introduction

1.1 Methods of sequencing and scheduling

Sequencing and scheduling problems are of great practical importance in many areas, including production planning and computer systems. Broadly defined, these problems are concerned with the optimal allocation of scarce resources to activities over time [26]. A sequencing problem involves finding a sequence in which to process a set of tasks, that minimizes a given cost function. A scheduling problem involves determining a detailed assignment of jobs to machines over a period of time, that minimizes a given cost function. The assignment of jobs to machines is subject to different technological constraints imposed by the jobs themselves and the machine environment. Given the variety of technological constraints and cost functions, the number of different types of scheduling problems is practically unlimited. The job shop scheduling model describes a very important class of scheduling problems. In this model, there is a set of jobs that must be processed by a set of machines, in order to minimize a single cost function that is usually a function of the job completion times. Each job is broken down into separate operations that need to be performed on different machines, in a prespecified order depending on the job. Each machine can process only one job at a time, and each job can be processed on only one machine at a time. Problems in which the machine order is the same over all jobs are called flow-shop problems. We will only be concerned with scheduling problems that are sequencing problems; problems for which a schedule is completely specified by the sequence in which the jobs are processed. We will focus on certain single-machine scheduling problems and two-machine flow-shop problems.

The theory of computational complexity provides a mathematical framework in which to study and compare different computational problems. (For an introduction to the concepts involved the interested reader is referred to Lawler et al. [26] and the book by Garey and Johnson [13].) According to this theory, problems are classified as either 'easy' or 'hard' depending on the theoretical worst-case behavior of their solution algorithms. The easy, or well-solved problems, are those for which there exist solution algorithms whose running times are bounded by a polynomial in the size of the problem input. The hard problems are those for which it is provably doubtful that such polynomial time algorithms exist. For our sequencing problems this is the class of "NP-hard" problems. (Following the scheduling literature, we use "NP-hard" to refer to the optimization version of an "NP-complete" decision problem.) For the class "NP-hard" it has been proven that, a polynomial algorithm exists for a given problem if and only if polynomial algorithms exist for all problems in the class. Since the class "NP-hard" contains many known hard problems, this provides strong evidence that a polynomial algorithm probably does not exist for any of these problems. This complexity result suggests that we must resort to solution methods that are essentially enumerative in nature to solve such problems. This is the case for the one- and two-machine problems that we study in this thesis. For these problems we employ a well-known enumerative method called branch and bound.

Branch and bound attempts to solve a sequencing problem by building a search tree that consists of fixed partial sequences. The branching scheme determines how to branch between fixed partial sequences, in order to traverse the tree in search of an optimal sequence. The bounding scheme specifies upper and lower bounds on the optimal completion of a given fixed partial sequence. A partial sequence and all its descendants can be eliminated from further consideration if its lower bound is greater than the best solution identified in the search tree up to this point. A stopping rule, which is usually expressed in terms of the total computation time or the total number of partial sequences examined, curtails the search. If the algorithm terminates by examining or eliminating all partial sequences, then the best solution identified is necessarily optimal. In this thesis, we examine new ways to further reduce the search space for such algorithms, in order to reduce both storage requirements and computation time. We derive a dominance order, which is a partial ordering of the jobs that is obeyed by at least one optimal sequence. Partial sequences that do not obey the dominance order can be eliminated from consideration, reducing the size of the search tree. One technique we use to derive our dominance order is called pairwise job interchange.

Pairwise job interchange is one of the most commonly used techniques for sequencing and scheduling problems. Pairwise job interchange compares the cost of sequences that differ only in the interchange of a pair of jobs. It tries to improve a sequence by performing interchanges that result in no increase in cost. The different types of pairwise interchanges can be considered as interchange operators that perform the specific transformation.

The interchange operators are classified following Monma [30], both by the type of interchange they perform and the relative position of the jobs. The jobs (before interchange) are classified as either *adjacent* or *nonadjacent*, and the type of interchange is either a direct swap of a pair of jobs or the *insertion* of a job either *forward* or *backward* in the sequence. The most commonly used interchange operators in the scheduling literature are: *Adjacent Pairwise Interchange (API), Pairwise Interchange (PI), Backward Insertion (BI),* and *Forward Insertion (FI).* To illustrate these operators consider the sequence $s_1 = (XmYkZ)$ where m and k are individual jobs and X, Y, and Z are (possibly empty) sequences of jobs. If $Y = \varnothing$, then $s_1 = (XmkZ)$, i.e., m and k are *adjacent* and API applied to s_1 gives $s_2 = (XkmZ)$. If $Y \neq \varnothing$, then m and k are *nonadjacent* and PI, BI, and FI applied to s_1 give respectively $s_3 = (XkYmZ)$, $s_4 = (XYkmZ)$, and $s_5 = (XkmYZ)$. Note that BI involves removing job m and inserting it backward in the sequence just after job k, while FI is the dual operation of removing job k and inserting it forward just before job m. Also note that if $Y = \varnothing$, then PI, BI, and FI all reduce to AI.

Pairwise job interchange has been used in various areas of scheduling theory. Among these:

- Proving the optimality of sequencing rules;
- Proving precedence relations and dominance criteria;
- Computing lower bounds for enumerative algorithms like branch and bound.

What all of these have in common is the property that there exists a *preference order* for the pairwise interchange of jobs for a given interchange operator, such that, if a pair of jobs is not in preference order then applying the interchange operator to the pair to make their order consistent with the preference order results in no increase in cost. In this way the sequencing function indicates a preference for the ordering of certain pairs of jobs. This ordering can be defined a priori, or it can be sequence- or time-dependent (i.e., dependent on the start time of the pair of jobs). In general, these preference orders can be very poorly behaved, they need not even be transitive.

The sequencing rules of Johnson [21], Smith [37], and Jackson [20], are examples of preference orders for the adjacent interchange of jobs that are complete and transitive orders. This order is complete because every pair of jobs k and m is ordered by the sequence, and it is transitive because of the transitivity of the sequence. Recall that in this case if k is preferred to m, then k and m can be interchanged if k and m are adjacent and m precedes k. If there are n jobs, then any optimal sequence can be transformed into preference order by applying at most $\frac{n(n-1)}{2}$ adjacent interchanges, proving the optimality of the sequence. Johnson's rule establishes such a sequence for the two-machine maximum completion time flow-shop problem $F2//C_{\max}$. (We use the standard notation to describe scheduling problems, and refer the reader to [26] and [32] for any terminology not defined here.) The rules of Smith and Jackson state the optimality of the weighted shortest processing time and earliest due date sequences for the total weighted completion time and maximum lateness problems, $1//\sum w_jC_j$ and $1//L_{\max}$, respectively.

Dominance criteria and precedence relations are partial orderings of the jobs that are satisfied by at least one optimal sequence. These orderings can be used to limit the search space, or provide branching rules for enumerative techniques such as branch and bound. They are most often derived using interchange operators where the conditions that define the preference ordering for interchange are sequence- or time-dependent. Next we consider some examples to illustrate the different properties of these preference orders. The precedence relation for total tardiness $1//\sum T_j$ due to Emmons [7], is an example where the preference order for nonadjacent pairwise interchange (i.e., for the PI operator) is an incomplete and transitive order, or partial order. This order does not depend on time, and is defined a priori by the jobs' parameters. By applying nonadjacent job interchanges any optimal sequence can be transformed into an optimal sequence which extends or satisfies the preference order. Thus the preference order for the PI operator is a precedence relation. Rinnooy Kan et al. [36] extend this to general nondecreasing costs $f_j(C_j)$, to minimize total cost for the single-machine scheduling problem $1//\sum f_j(C_j)$. They derive dominance criteria for an enumeration scheme that fills a schedule from back to front. They use a number of interchange operators including (PI) and (BI) to demonstrate their dominance criteria. Here the conditions for the preference orders for interchange are sequence-dependent, depending on the predecessor and successor sets as well as the job parameters. For multiple machines Della Croce [5] derives dominance criteria for partial sequences using (PI) and (FI). He does so for the two-machine total completion time problem, $F2//\sum C_j$, and the maximum completion time problem with release times, $F2/r_j/C_{\max}$, in flow-shops. For the API operator two different approaches have been used to derive precedence relations. The first is a sequence-dependent approach, and the second is a time-dependent approach. In the first case we have the pyramid precedence orders of Erschler et al. [8] and [9], for maximum lateness with release times and maximum completion time with release times and deadlines, $1/r_j/L_{\max}$ and $1/r_j,\overline{d_j}/C_{\max}$, respectively. The pyramid order, is a precedence relation that is a subset of the preference order for adjacent interchange. This subset defines precedence relations that are true for nonadjacent jobs, not just adjacent jobs. They find conditions on intermediate jobs that determine when adjacent preferences extend to nonadjacent preferences. Given a sequence $(XmYkZ)$ with k preferred to m for adjacent interchange, they find conditions on the intermediate sequence Y that permit k and m to be interchanged using PI to obtain the dominant sequence $(XkYmZ)$. Thus, this approach can be considered as sequence-dependent, depending on the intermediate sequence. The parametric precedence relations of Szwarc et al. [39], [38], and [40] are examples where the preference order for the adjacent interchange of jobs is an incomplete and intransitive order that depends on the start time of the pair. Pairs that are ordered independently of start time are called globally ordered while the remaining pairs are said to be locally ordered. Assuming that the global order is decomposable, they split the problem into smaller subproblems and look at the mix of global and local orderings on each part. It is the mixture of these two types of orderings that is not transitive, and they try to find a subset of this mixture that is a precedence relation.

Lower bounds can be computed for certain single machine scheduling problems using the technique of Della Croce [5]. This technique is to repeatedly apply the time-dependent precedence relations of Szwarc to compute a lower bound for a partial sequencing of the jobs. He derives lower bounds for the total quadratic lateness problem $1//\sum L_j^2$, the total tardiness problem $1//\sum T_j$, and certain multicriteria problems.

In this thesis, we consider a generalization of pairwise interchange, that is a new sequence-dependent approach which incorporates subset restrictions on the intermediate sequence. We call this technique subset-restricted pairwise interchange. We use this technique to derive new dominance orders for certain one-and two-machine sequencing problems. These dominance orders are used to reduce branching in an efficient branch and bound algorithm capable of solving large problem instances.

1.2 Preview of the thesis

For the remainder of this Chapter, we present the preliminary definitions and notation that will be used throughout the thesis. In the next two sections we give the basic definitions and notation for sequencing and scheduling problems and partial order theory, respectively. This is followed by a discussion of the fundamental concepts and definitions, of subset-restricted pairwise interchange, which are formulated for a general sequencing problem.

In Chapter 2, we derive new dominance results for the single machine scheduling problem $1/r_j/f_{\max}$, using subset-restricted pairwise interchange. We derive a dominance order that is a suborder of the adjacent interchange order for these problems. This extends the pyramid precedence orders of Erschler et al. ([8] and [9]) for $1/r_j/L_{\max}$ and $1/r_j,\overline{d_j}/C_{\max}$, to the general problem $1/r_j/f_{\max}$, which includes $1/r_j/\max w_jC_j$ and $1/r_j/\max w_jL_j$ as special cases. We do so by applying different interchange operators together with a new representation of the dominance order. This new diagonal representation is not limited by the dimension of the adjacent interchange order like the pyramid representation, which was restricted to two-dimensional orders. We tested the effectiveness of the dominance order in a branch and bound algorithm for the problem $1/r_j/\max w_jC_j$. The algorithm performed very well solving 2381 of the 2400 test problems, while reducing the total computation time by 58 percent.

In Chapter 3, we apply subset-restricted interchange to derive a new dominance order for the two-machine flow-shop problem $F2/r_j/C_{\max}$, and incorporate it into a very fast branch and bound algorithm. We tested the algorithm in a large-scale computational experiment. The algorithm solved, within a few seconds, over 95 percent of the test problems, some with up to 500 jobs. Even for the unsolved problems, we were within 3 percent of the optimal solution in the worst case. This means that the algorithm has the potential of being used as a subroutine for $F2/r_j/C_{\max}$ type subproblems generated during the solution of more complicated problems. We also found that the speed of the algorithm is largely due to the use of the dominance order, which reduced the total computation time by around 80 percent. The experiment also helped in classifying 'easy' and 'hard' instances of the problem, it indicated that the hardest problems were those with range of release times approximately 1/2 the expected total processing time of the jobs.

In Chapter 4, we derive new dominance results for the well-known two-machine permutation flow-shop problem $F2\,/r_j, perm/\,L_{\max}$. This time, we apply subset-restricted interchange with a new interchange operator, which we call shuffle interchange. With this operator we derive a new dominance order, which we use in a branch and bound algorithm for the problem. The algorithm also features a new decomposition procedure which fixes jobs at the beginning of the schedule, and was quite effective at reducing the size of the problem. The algorithm performed well, very quickly solving 4,384 of the 4500 test problems which ranged in size from 20 to 200 jobs. Because of this, the algorithm also has great potential as a subroutine for the solution of more complicated scheduling problems. We also found that the dominance order reduced the total computation time by 15 to 20 percent.

Chapter 5 gives a brief summary of the main results of the thesis, as well as some directions for future research.

1.3 Preliminary definitions and notation

1.3.1 Sequencing notation

Recall, we call a scheduling problem a *sequencing problem* and its objective a *sequencing function* if any schedule can be completely specified by the sequence in which the jobs are performed. For the problems $1\,/r_j/\,f_{\max}$ and $F2\,/r_j/\,C_{\max}$, it is well known that such optimal permutation schedules exist. For problem $F2\,/r_j/\,L_{\max}$, we will restrict our attention to such permutation schedules and denote the problem by $F2\,/r_j, perm/\,L_{\max}$. For each of these problems we let $J = \{1, 2, \ldots, n\}$ be the set of jobs to be sequenced. A sequence s on J is a function from $\{1, 2, \ldots, n\}$ to J represented by the n-tuple $(s(1), s(2), \ldots, s(n))$, where $s(i)$ is the i^{th} job in sequence s. For these problems the objective is to find a sequence s that minimizes the maximum cost taken over all the jobs, where the cost of a job is a function of its completion time.

The single-machine scheduling problem $1\,/r_j/\,f_{\max}$ has the following defining properties. Each job j requires p_j units of processing time on the machine, it can start processing on the machine at any time after its release time r_j, however, once its processing has started it cannot be interrupted. The cost of job j is given by a nondecreasing real valued function $f_j(t)$ that gives the cost of completing job j at time t. If we let C_j be the completion time of job j, then its cost is given by $f_j(C_j)$. An important example is the lateness objective with $L_j = C_j - d_j$, where d_j is the due date for job j. Here the objective is to find a sequence s that minimizes $\max\limits_{1\le i\le n} L_{s(i)}$.

For the two machine flow-shop problems $F2\,/r_j/\,C_{\max}$ and $F2\,/r_j, perm/\,L_{\max}$, the jobs have release times r_j and they must be processed in the same sequence without interruption first on machine 1 and then on machine 2. For a given job j its processing on machine 1 must be finished before its processing on machine 2 can start. For job j, its processing times on machines 1 and 2, are given by a_j and b_j, respectively. The completion time of job j on machines 1 and 2 will be denoted by C_j^1 and C_j^2, respectively. The completion time of job j is C_j where now $C_j = C_j^2$. For the problem $F2\,/r_j/\,C_{\max}$, the cost of job j is C_j and the objective is to find a sequence s that minimizes the completion time of the last job in the sequence, $C_{s(n)}$. For $F2\,/r_j, perm/\,L_{\max}$, the cost of job j is given by $L_j = C_j - d_j = C_j^2 - d_j$, and the objective is the same as above for $1\,/r_j/\,L_{\max}$. We will choose to consider these problems in their equivalent delivery-time form, denoted by $1\,/r_j/\,L'_{\max}$ and $F2\,/r_j, perm/\,L'_{\max}$, where $q_j = T - d_j$ is the delivery time for job j and T is a constant with $T \ge \max\{d_j \,|j \in J\}$. If we define $L'_j = C_j + q_j$, then $L'_j = C_j + T - d_j = L_j + T$, and we see that the two objectives $L'_{\max}$ and $L_{\max}$ are equivalent.

For our problems, the dominance order will be a partial order defined by the parameters of the jobs. Thus we introduce certain definitions for partially ordered sets (posets).

1.3.2 Partial orders

By a partially ordered set we mean a pair $P = (X, \leq_P)$ consisting of a set X together with a binary relation $\leq_P$ on $X \times X$, which is reflexive, antisymmetric, and transitive. For $u, v \in X$, $u \leq_P v$ is interpreted as u is less than or equal to v in P. Similarly, $u <_P v$ means that $u \leq_P v$ and $u \neq v$. The usual symbols $\leq$ and $<$ will be reserved for relations between real numbers. A partial order $P = (X, \leq_P)$ is a *linear order* (or *complete order*) if for every pair $(u, v) \in X \times X$ either $u \leq_P v$ or $v \leq_P u$. Given a pair of partial orders $P = (X, \leq_P)$ and $Q = (X, \leq_Q)$ on the same set X, we call Q an *extension* of P (P a *suborder* of Q) if $u \leq_P v$ implies $u \leq_Q v$ for all $u, v \in X$. A partial order $Q = (X, \leq_Q)$ is a *linear extension* of a partial order $P = (X, \leq_P)$, if Q is a linear order that extends P. Given two partial orders $P_1 = (X, \leq_{P_1})$ and $P_2 = (X, \leq_{P_2})$, we can define the partial order $P_1 \cap P_2 = (X, \leq_{P_1 \cap P_2})$, the *intersection* of P_1 and P_2, where $u \leq_{P_1 \cap P_2} v$ if and only if $u \leq_{P_1} v$ and $u \leq_{P_2} v$ for all $u, v \in X$. The *dimension* of a partial order $P = (X, \leq_P)$, denoted by $\dim(P)$, is the smallest l such that there exists a set $\{Q_1, Q_2, \ldots, Q_l\}$ of linear extensions of P such that $P = \cap_{i=1}^{l} Q_i$. A subset $I \subseteq X$ is an *ideal* of P if for every $v \in I$ and $u \in X$ such that $u \leq_P v$ we have $u \in I$. Similarly, $F \subseteq X$ is a *filter* of P if for every $u \in F$ and $v \in X$ such that $u \leq_P v$ we have $v \in F$. For every $v \in X$ the *principal ideal* $I(v)$ is defined by $I(v) = \{u \in X \mid u \leq_P v\}$ and the *principal filter* $F(v)$ is defined by $F(v) = \{u \in X \mid v \leq_P u\}$. A partial order P on the job set of a sequencing problem is called a *dominance order* if there is an optimal sequence that is a linear extension of P.

1.3.3 Subset-restricted pairwise interchange

We follow Monma [30] in defining our interchange operators. Consider a sequence with job m preceding job k, of the form $(XmYkZ)$ where X, Y, and Z are subsequences of J. We define three types of interchanges of jobs k and m that leave k preceding m in the resulting sequence.

1. *Backward Insertion(BI)*	$(XmYkZ) \rightarrow (XYkmZ)$
2. *Forward Insertion(FI)*	$(XmYkZ) \rightarrow (XkmYZ)$
3. *Pairwise Interchange(PI)*	$(XmYkZ) \rightarrow (XkYmZ)$

If we let Y be the set of jobs in sequence Y (we do not distinguish between these), then each of these interchanges reduces to adjacent pairwise interchange in the case when $Y = \varnothing$. This leads to the definition of the *adjacent interchange order.*

Definition 1.1 *A partial order $\lessdot$ is an adjacent interchange order for a sequencing function f if for all jobs k, m and sequences X, Z $k \lessdot m$ implies $f(XkmZ) \leq f(XmkZ)$.*

Note that all of the above interchanges involve interchanging k, m or both k and m around sequence Y. Intuitively, whether or not an interchange of jobs k and m with $k \lessdot m$ leads to a reduction in cost (for a given sequencing function f and adjacent interchange order $\lessdot$), should depend on the composition of Y. This involves placing restrictions on certain of the parameters of the jobs in Y. In the case when interchangeability does not depend on the composition of Y, then $\lessdot$ is a dominance order for the sequencing problem, i.e., there exists an optimal sequence that is a linear extension of $\lessdot$. Such an example is the dominance order $\lessdot$ defined by

$$k \lessdot m \Leftrightarrow p_k \leq p_m \text{ and } d_k \leq d_m$$

for the total tardiness problem on a single-machine, $1//\sum T_i$ [7]. Note that here $\lessdot$ is the intersection of the $\leq_p$ and $\leq_d$ orders. In general, an adjacent interchange order $\lessdot$ is not necessarily a dominance order, as we shall see later. We consider interchanges that are restricted by conditions on Y and define the subset-restricted interchange conditions as follows.

Definition 1.2 *An adjacent interchange order $\lessdot$ together with the collection of subsets $R^{PI} = \left\{R^{PI}_{k \lessdot m} \mid k \lessdot m\right\}$ satisfies the Restricted Pairwise Interchange Condition for a sequencing function f if for all jobs k, m and sequences X, Y, Z $k \lessdot m$ and $Y \subseteq R^{PI}_{k \lessdot m}$ imply $f(XkYmZ) \leq f(XmYkZ)$.*

Definition 1.3 *An adjacent interchange order $\lessdot$ together with the collection of subsets* $R^{BI} = \left\{ R^{BI}_{k\lessdot m} \,|k\lessdot m \right\}$ *satisfies the Restricted Backward Insertion Condition for a sequencing function f if for all jobs k, m and sequences X, Y, Z $k\lessdot m$ and $Y \subseteq R^{BI}_{k\lessdot m}$ imply $f\,(XYkmZ) \leq f\,(XmYkZ)$.*

Definition 1.4 *An adjacent interchange order $\lessdot$ together with the collection of subsets* $R^{FI} = \left\{ R^{FI}_{k\lessdot m} \,|k\lessdot m \right\}$ *satisfies the Restricted Forward Insertion Condition for a sequencing function f if for all jobs k, m and sequences X, Y, Z $k\lessdot m$ and $Y \subseteq R^{FI}_{k\lessdot m}$ imply $f\,(XkmYZ) \leq f\,(XmYkZ)$.*

The *interchange regions* for the pair $k\lessdot m$ are the sets of jobs that k and m can be interchanged around without increasing cost. For PI, BI, and FI these are given by $R^{PI}_{k\lessdot m}$, $R^{BI}_{k\lessdot m}$, and $R^{FI}_{k\lessdot m}$, respectively. In the following Chapters, we will use subset-restricted interchange to derive a dominance order $\prec$ on the jobs, that is a suborder of the adjacent interchange order $\lessdot$.

Chapter 2
Minimizing maximum cost for single-machine sequencing problems with release times

2.1 Introduction

In this Chapter, we introduce a new technique, a generalization of pairwise interchange that takes into consideration the composition of the intermediate sequence. This yields a preference order which permits the interchange of a pair of jobs *provided* that the intermediate jobs belong to a restricted subset. The traditional methods of pairwise job interchange can be viewed as special cases of this *subset-restricted interchange,* where the subsets are either uniformly the empty set (adjacent interchange) or the entire job set (nonadjacent interchange). In general, an adjacent interchange order is not a complete order and therefore it is *not* a dominance order, as we shall see for the $1/r_j/L'_{\max}$ problem, using an example due to Lageweg et al. [23]. We prove, however, that using subset-restricted interchange for the class of regular, single machine scheduling problems $1/r_j/f_{\max}$, we can derive a dominance order that is a suborder of the adjacent interchange order. Recall, for these problems, each job j has an associated nondecreasing, real valued cost function $f_j(t)$, the cost of completing j at time t, and the objective is to minimize the maximum cost $f_{\max}$. The dominance orders we derive have the property that they are defined *independently* of the processing times. This makes them especially useful in applications with stochastic or ill-defined processing times. We use only certain extreme values of the other job parameters that display a 'staircase-like' structure. In addition, the dominance orders derived belong to a special class of partial orders, the interval orders [11]. This also leads to the complexity implication that the above problems are strongly NP-hard for interval-ordered tasks. Our results can be viewed as a *unified treatment* of job interchange, that *generalizes* well-known rules for deriving dominance relations and *extends* the pyramid dominance orders of Erschler et al. ([8] and [9]) for $1\,/r_j\,/L_{\max}$ to the general problem $1\,/r_j/\,f_{\max}$. We generalize the pairwise interchange and insertion operations of Monma [30], and introduce 'pyramid-like' structures of higher dimension than two, extending the 2-dimensional staircases of [8] and [9]. In our unified theory, the problem $1\,/r_j\,/L_{\max}$ represents the most special case.

The Chapter is organized as follows. In the next section, we review some important previous results. In section 3, we define the adjacent interchange order $\lessdot$ and the interchange regions for $1\,/r_j\,/f_{\max}$. In section 4, we derive a dominance order $\prec$ for the problem $1\,/r_j\,/f_{\max}$, which includes the problems $1\,/r_j\,/L_{\max}$, $1\,/r_j, \overline{d_j}/\,C_{\max}$, $1\,/r_j\,/\max w_jC_j$, and $1\,/r_j\,/\max w_jL_j$ as special cases. In section 5, we discuss the results of a computational experiment for the problem $1\,/r_j\,/\max w_jC_j$. Finally, in section 6 we summarize the results of the Chapter.

2.2 Previous results

Although the maximum cost objective is quite general, there are still some very important polynomially solvable general cases. The single machine problem with precedence constraints $1\,/prec\,/f_{\max}$, is solved in $O(n^2)$ time by Lawler's algorithm [25]. Lawler's algorithm sequences the jobs from back to front, by repeatedly sequencing last a maximal job in the precedence order which has the smallest cost if put in the last position. An interchange argument using BI shows that it is only necessary to consider such schedules. Baker et al. [2] generalize Lawler's algorithm to cover the preemptive case with release times $1\,/r_j, pmtn, prec\,/f_{\max}$, which can be implemented to run in $O(n^2)$ time. That the nonpreemptive case $1\,/r_j\,/f_{\max}$ is *strongly NP-hard* follows from the result for $1\,/r_j\,/L_{\max}$ due to Garey and Johnson [13]. This complexity result has led researchers to consider various enumeration and approximation algorithms.

Many branch and bound algorithms have been proposed for the maximum lateness problem $1\,/r_j\,/L_{\max}$; exhibiting various branching and lower bounding schemes. Among these are the algorithms of Baker and Su [1], McMahon and Florian [28], and Lageweg et al. [23]. The relatively most effective is an algorithm due to Carlier [3] , for the problem $1\,/r_j\,/L'_{\max}$. The effectiveness of Carlier's algorithm is due largely to the efficient binary branching rule he employs. The branching rule is based on the dominance properties of a critical path in the ready-Jackson sequence; this is the sequence obtained by greedily sequencing the job j with the largest q_j (i.e., earliest due date) from among the ready jobs. Zdralka and Grabowski [43] consider an extension of this branching rule for the problem $1\,/prec, r_j\,/f_{\max}$, which they test for the problems $1\,/r_j\,/L'_{\max}$ and $1\,/r_j\,/\max w_j L_j$. Erschler et al. [8] and [9], derive a dominance order that is a suborder of the adjacent interchange order, which they use to reduce branching in the algorithm of Baker and Su [10].

Potts [33] develops an approximation algorithm A for $1\,/r_j\,/L'_{\max}$, which iteratively tries to improve the 'ready-Jackson' sequence. Algorithm A has a worst case performance guarantee of 3/2, i.e., $L'_{\max}(A)/L'^{*}_{\max} \leq 3/2$. Given the complexity of the problem $1\,/r_j\,/L'_{\max}$, the best possible approximate result that can be obtained is a polynomial approximation scheme. Hall and Shmoys present such a scheme for the problem, in [19].

2.3 Interchange regions

In this section, we derive the interchange regions (subsets) for the general problem $1\,/r_j\,/f_{\max}$. Recall, in this problem, each job j has an associated *nondecreasing,* real-valued cost function f_j, where $f_j\,(t)$ is the cost of completing job j at time t, and the objective is to minimize $f_{\max} = \max_{1\leq j\leq n} f_j(C_j)$ over all sequences. We order the jobs according to $\geq_f$, where $f_i \geq_f f_j \Leftrightarrow f_i\,(t) \geq f_j\,(t)$ *for all* $t \geq 0$. Note that, in the general case, $\geq_f$ does not order every pair i and j, it might be only a partial order. Two special, linearly ordered cases of $\geq_f$ occur for the lateness objective in its delivery form , where $f_j\,(t) = t + q_j$, and the weighted completion time objective, where $f_j(t) = w_j t$. Hall [17] considered the $\geq_f$ order and noticed that the linear ordering property makes it possible to extend Potts' [33] approximation algorithm for the $1/r_j/L'_{\max}$ problem to the $1/r_j/f_{\max}$ problem *when* $\geq_f$ is a linear order.

The adjacent interchange order and the restricted subsets for $1/r_j/f_{\max}$ are defined below. We note that these definitions use *no processing time information.* This means that all of the subsequent results are true *irrespective* of job processing times.

Definition 2.1 *Adjacent interchange order:* $k \lessdot m \Leftrightarrow r_k \leq r_m$ *and* $f_k \geq_f f_m$.

Definition 2.2 *Given* $k \lessdot m$, *define the following subsets of jobs:*
(*i*) $R^{PI}_{k\lessdot m} = \{j \,|r_j \leq r_m, f_k \geq_f f_j\}$
(*ii*) $R^{BI}_{k\lessdot m} = \{j \,|r_j \leq r_m\}$
(*iii*) $R^{FI}_{k\lessdot m} = \{j \,|f_k \geq_f f_j\}$.

In general, an adjacent interchange order $\lessdot$ is not necessarily a dominance order, as it can be demonstrated by the instance of the maximum lateness problem $1/r_j/L'_{\max}$, shown in its equivalent delivery time form in Example 1 [23]. As we shall see in the next section, the adjacent interchange order $\lessdot$ for $1/r_j/L'_{\max}$ is defined by $k \lessdot m \Leftrightarrow r_k \leq r_m$ and $q_k \geq q_m$. For the 5 job example specified below we have $4 \lessdot 2$, however, the unique optimal sequence is $(1, 2, 3, 4, 5)$ with $L'_{\max} = 11$. Thus $\lessdot$ is *not* a dominance order, since the unique optimal sequence is not a linear extension of $\lessdot$.

Example 1 *A 5 job problem to illustrate that* $\lessdot$ *is not necessarily a dominance order.*

j	1	2	3	4	5
r_j	0	2	3	0	7
p_j	2	1	2	2	2
q_j	5	2	6	3	2

Next we prove that the adjacent interchange order $\lessdot$ together with the above collections of subsets satisfy the subset-restricted interchange conditions.

Theorem 2.1 *Partial order* $\lessdot$ *together with the collection of subsets* $R^{PI} = \left\{R^{PI}_{k\lessdot m} \,|k \lessdot m\right\}$ *satisfies the Restricted Pairwise Interchange Condition for* $1/r_j/f_{\max}$.

Proof. Given a sequence s, recall that $f_{\max}(s) = \max_{1\le j\le n} f_{s(j)}\left(C_{s(j)}\right)$, where $C_{s(j)}$ is the completion time of job $s(j)$. We construct a directed graph $G(s)$ to evaluate $f_{\max}(s)$ (see Figure 2.1). Each job $s(j)$ is represented by two nodes: the first one has weight $r_{s(j)}$, followed by the second node with weight $p_{s(j)}$. $G(s)$ has the property that the completion time of job $s(j)$ in sequence s is the length of the longest 'node-weighted' path from 0 to $s(j)$. We represent the paths from 0 to $s(j)$ by pairs $(s(i), s(j))$ $1 \le i \le j \le n$, where $s(i)$ is the first node on the path following the start node 0 and $s(j)$ is the last node of the path. Then by definition

$$f_{\max}(s) = \max_{1\le j\le n}\left[f_{s(j)}\left(\max_{\substack{(s(i),s(j))\\1\le i\le j}}\left(r_{s(i)} + \sum_{l=i}^{j} p_{s(l)}\right)\right)\right] = \max_{\substack{(s(i),s(j))\\1\le i\le j\le n}}\left[f_{s(j)}\left(r_{s(i)} + \sum_{l=i}^{j} p_{s(l)}\right)\right]$$

and we can evaluate $f_{\max}(s)$ as the maximum over all such pairs $(s(i), s(j))$ in $G(s)$.

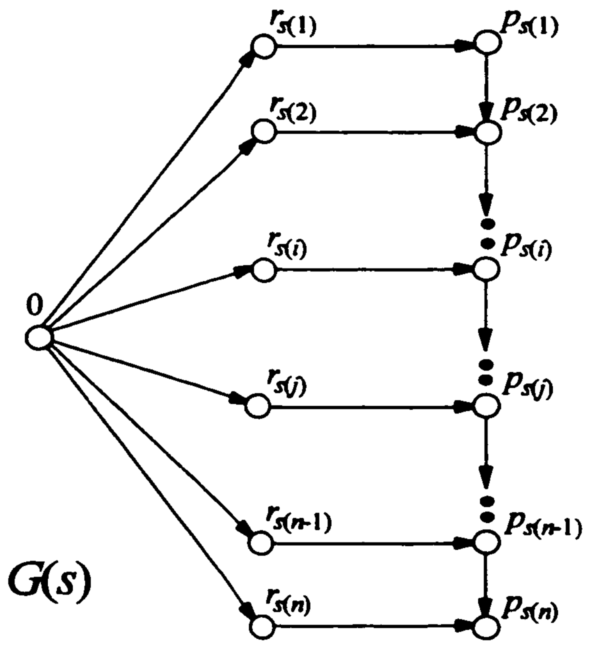

Figure 2.1: Directed graph $G(s)$ for problem $1/r_j/f_{\max}$.

Let $s_1 = (XmYkZ)$ be a sequence with the property that $k{<}m$ and $Y \subseteq R^{PI}_{k<m}$. We apply pairwise interchange to s_1 and obtain sequence $s_2 = (XkYmZ)$.

We demonstrate that s_2 is not worse than s_1 by showing that for *every* pair of jobs in s_2 there exists a ***dominating pair*** in s_1 with a not smaller f value. For example, consider pair (k, m) in s_2, then it has the dominating pair (m, k) in s_1 (see Figure 2.2): That is,

$$f_m\left(r_k + p_k + \sum_{y\in Y} p_y + p_m\right) \le f_k\left(r_m + p_m + \sum_{y\in Y} p_y + p_k\right),$$

which holds since $k<m$ implies $r_k \le r_m$ and $f_k(t) \ge f_m(t)$ *for all* t, from which it follows that $f_m\left(r_k + p_k + \sum_{y\in Y} p_y + p_m\right) \le f_k\left(r_k + p_m + \sum_{y\in Y} p_y + p_k\right)$, and finally f_k is *nondecreasing*.

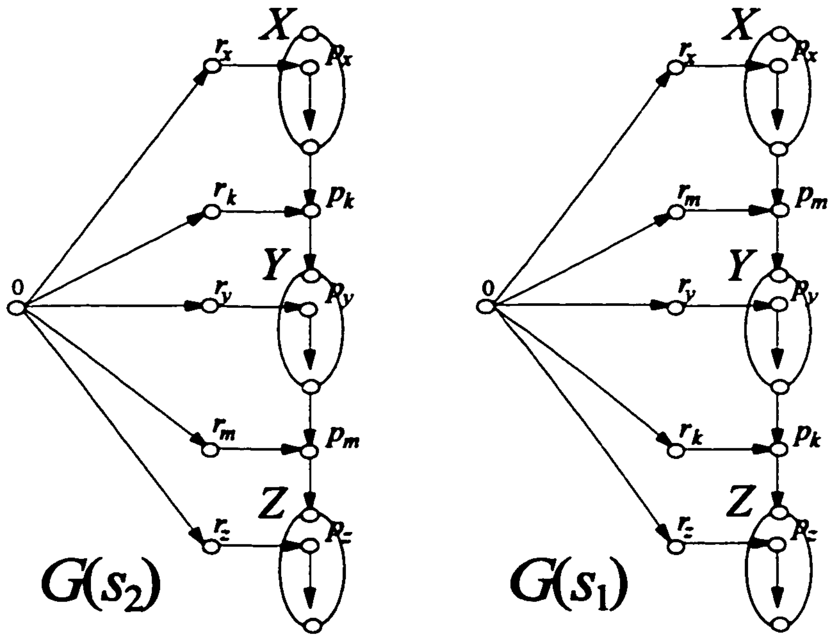

Figure 2.2: Directed graphs $G(s_2)$ and $G(s_1)$ for PI.

Table 2.1 gives a dominating pair in s_1 for each pair in s_2. (We use lower case letters x, y, or z to refer to arbitrary generic elements of the subsequences X,Y or Z, respectively). The last three columns contain the arguments why they are dominating pairs. ■

s_2	s_1			
$(XkYmZ)$	$(XmYkZ)$	$proof$		
(x,k)	(x,k)			f_k *nondecreasing*
(x,y)	(x,k)		$f_k \geq_f f_y$	f_k *nondecreasing*
(x,m)	(x,k)		$f_k \geq_f f_m$	f_k *nondecreasing*
(x,z)	(x,z)			
(k,y)	(m,k)	$r_k \leq r_m$	$f_k \geq_f f_y$	f_k *nondecreasing*
(k,m)	(m,k)	$r_k \leq r_m$	$f_k \geq_f f_m$	f_k *nondecreasing*
(k,z)	(m,z)	$r_k \leq r_m$		f_z *nondecreasing*
(y,m)	(m,k)	$r_y \leq r_m$	$f_k \geq_f f_m$	f_k *nondecreasing*
(y,z)	(m,z)	$r_y \leq r_m$		f_z *nondecreasing*
(m,z)	(m,z)			f_z *nondecreasing*

Table 2.1: Dominating pairs for Theorem 2.1.

s_2	s_1			
$(XYkmZ)$	$(XmYkZ)$	*proof*		
(x,y)	(x,y)			f_y *nondecreasing*
(x,k)	(x,k)			f_k *nondecreasing*
(x,m)	(x,k)		$f_k \geq_f f_m$	f_k *nondecreasing*
(x,z)	(x,z)			
(y,k)	(y,k)			
(y,m)	(m,k)	$r_y \leq r_m$	$f_k \geq_f f_m$	f_k *nondecreasing*
(y,z)	(m,z)	$r_y \leq r_m$		f_z *nondecreasing*
(k,m)	(m,k)	$r_k \leq r_m$	$f_k \geq_f f_m$	f_k *nondecreasing*
(k,z)	(m,z)	$r_k \leq r_m$		f_z *nondecreasing*
(m,z)	(m,z)			f_z *nondecreasing*

Table 2.2: Dominating pairs for Theorem 2.2.

Theorem 2.2 *Partial order* $<$ *together with the collection of subsets* $R^{BI} = \left\{ R^{BI}_{k<m} \,|k<m \right\}$ *satisfies the Restricted Backward Insertion Condition for* $1 / r_j / f_{\max}$.

Proof. The proof is totally analogous to that for pairwise interchange. Table 2.2 gives the corresponding dominating pairs. ■

Theorem 2.3 *Partial order* $<$ *together with the collection of subsets* $R^{FI} = \left\{ R^{FI}_{k<m} \,|k<m \right\}$ *satisfies the Restricted Forward Insertion Condition for* $1 / r_j / f_{\max}$.

Proof. The proof is totally analogous to that for pairwise interchange. Table 2.3 gives the corresponding dominating pairs. ■

Remark 1 *We observe that for any* $k<m$, *we have* $R^{PI}_{k<m} = R^{BI}_{k<m} \cap R^{FI}_{k<m}$. *Thus if* $Y \subseteq R^{PI}_{k<m}$, *then this implies not only* $f(XkYmZ) \leq f(XmYkZ)$, *but also* $f(XYkmZ) \leq f(XmYkZ)$ *and* $f(XkmYZ) \leq f(XmYkZ)$.

The preceding theorems could directly be used in branch and bound algorithms for restricting the search space on sequences. This, however, would require branching on sequences and storing for all pairs $k<m$ the subsets of Definition 2.2 and the testing of membership in these, which would be time consuming and inefficient. In

s_2	s_1			
$(XkmYZ)$	$(XmYkZ)$	$proof$		
(x,k)	(x,k)			f_k *nondecreasing*
(x,m)	(x,k)		$f_k \geq_f f_m$	f_k *nondecreasing*
(x,y)	(x,k)		$f_k \geq_f f_y$	f_k *nondecreasing*
(x,z)	(x,z)			
(k,m)	(m,k)	$r_k \leq r_m$	$f_k \geq_f f_m$	f_k *nondecreasing*
(k,y)	(m,k)	$r_k \leq r_m$	$f_k \geq_f f_y$	f_k *nondecreasing*
(k,z)	(m,z)	$r_k \leq r_m$		f_z *nondecreasing*
(m,y)	(m,y)			
(m,z)	(m,z)			f_z *nondecreasing*
(y,z)	(y,z)			f_z *nondecreasing*

Table 2.3: Dominating pairs for Theorem 2.3.

the following sections, we show that there is a much more effective way to restrict the search space, by proving that there is a dominance order on the jobs.

We also note that by simply modifying release times and processing times, problems in which the jobs have setup times can be handled as well. Two forms of job setups can be considered: either a setup ρ_i is *attached* to job i, i.e., it cannot be performed without the job being present, that is before r_i; or it is *detached,* i.e., it can be performed prior to r_i, if the machine is idle and waiting to process job i. Attached setups can be dealt with by using modified processing times $p_i' = p_i + \rho_i$, while detached setups can be handled using modified release times $r_i' = \max\{0, r_i - \rho_i\}$ and processing times $p_i' = p_i + \rho_i$. Thus, our theory of dominance orders applies to the case with setups too.

2.4 A general dominance order

In this section, first we use subset-restricted interchange to derive a dominance order $\prec$ for the general case of $1/r_j/f_{\max}$, followed by various special cases containing well-known scheduling problems. We can represent the adjacent interchange order $\lessdot$ using a 3-dimensional structure. The axes of the 3-dimensional space correspond to the r, t, and $f(t)$ values (see Figure 2.3). Jobs are represented by their $f_j(t)$ curves in this space resting on planes whose height equals their release times. Recall that $k \lessdot m \iff r_k \leq r_m$ and $f_k \geq f_m$, that is, if k is on a lower r-plane than m and f_k is above f_m, i.e., $f_k(t) \geq f_m(t)$ *for all* t. The precedence order $\prec$ is defined in terms of certain 'extreme jobs' in $\lessdot$, or more precisely, certain extreme curves of the 3-dimensional structure. (Some of these extreme curves might not correspond to real jobs.) To identify these jobs, we introduce the *least upper bound* in $\leq_f$ for a finite set of jobs $I \subseteq J$, as the nondecreasing function $\operatorname{pmax}_{j \in I} f_j$ defined as the *pointwise maximum* of the functions f_j, i.e., with values $(\operatorname{pmax}_{j \in I} f_j(t))(t) = \max_{j \in I} f_j(t)$ for all t. A simple greedy procedure to define the set of *boundary jobs* $M = \{M_i \,|\, i = 1, 2, \ldots, H+1\}$ and the set of *diagonal jobs* $\Delta = \{D_1, D_2, \ldots, D_H\}$, is presented below. Assume that there are K distinct r values denoted by r^i for $i = 1, 2, \ldots, K$, and $r^1 > r^2 > \cdots > r^K$.

Define the function $f^i = \text{pmax}\{f_j \,|r_j = r^i\}$, this is the ***least upper bound*** in $\leq_f$ of the jobs on level r^i.

```
**Algorithm Δ for** $1/r_j/f_{\max}$
let $(r_{M_1}, f_{M_1}) = (r^1, f^1)$, $l = 1$, $i = 2$
while $i \leq K$ do
begin
    if $f_{M_l} \geq_f f^i$ then $i := i + 1$;
    else
    begin
        let $(r_{M_l+1}, f_{M_l+1}) = (r^i, \text{pmax}\{f^i, f_{M_l}\})$ {This defines $M_{l+1}$ and the
        function $f_{M_l+1}$}
        let $(r_{D_l}, f_{D_l}) = (r^i, f_{M_l})$ {This defines $D_l$ and the function $f_{D_l}$}
        $l := l + 1$;
    end;
end;
```

The procedure looks at the r-levels r^i $(i = 1, 2, \ldots, K)$ and compares the largest f on this level (f^i) with the largest f obtained so far (f_{M_l}). If f^i represents a strict increase compared to f_{M_l}, then (r^i, f^i) becomes the new boundary pair (r_{M_l+1}, f_{M_l+1}) and (r_{D_l}, f_{D_l}) is the projection of (r_{M_l}, f_{M_l}) onto level r^i. Note that these extreme jobs are not necessarily real jobs, since they are defined using pointwise maximum. Rather they are 'artificial jobs' used only for ordering purposes. The sets Δ and M define a 'staircase-like' structure that contains all of the curves f_j for $j \in J$ either inside or on its surface (Figure 2.3 shows an example with $|\Delta| = 4$). We augment the partial order $P = (J, \lessdot)$ with the diagonal set Δ and call it $P_\Delta = (J_\Delta, \lessdot_\Delta)$, where $J_\Delta = J \cup \Delta$ and $\lessdot_\Delta$ is the order $\lessdot$ extended to include the diagonal jobs, i.e., $\lessdot_\Delta$ is derived by applying Definition 2.1 to the extended set J_Δ. Jobs $k \lessdot m$ $(k,m \in J)$ are *separated by* Δ (are *Δ-separated)* if there exists a D_i $i \in \{1, 2, \ldots, H\}$ such that k is in its principal ideal and m is in its principal filter, i.e., $k \in I(D_i)$ and $m \in F(D_i)$

in P_Δ. Δ induces a *partition* of P into separable and nonseparable pairs that can be used to define $\prec$.

Definition 2.3 *Dominance order $\prec$: For $k<m$ ($k,m \in J$), we define $k \prec m$ if and only if k and m are separated by Δ.*

It is well known that the main source of difficulty in all $1/r_j/f_{\max}$ problems is the fact that at any time the machine becomes available, it may be better to wait for a yet unreleased job rather than to schedule one of the jobs available. The partition of P by Δ means that the *only* jobs for which it may be worth waiting are the ones which are *not* separated by Δ from the currently available jobs, that is $\prec$ is a dominance order.

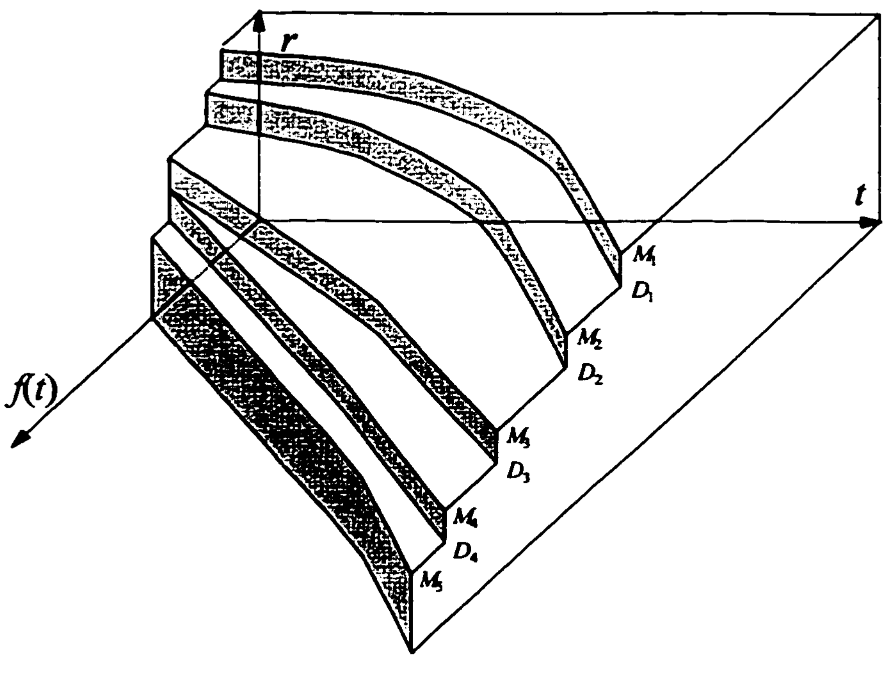

Figure 2.3: Staircase structure for the general case.

Theorem 2.4 *Partial order $\prec$ is a dominance order for $1/r_j/f_{\max}$.*

Proof. We use subset-restricted interchange in the proof. The following observations immediately follow from the construction of Δ:

$$\begin{array}{ccccccc} D_H & <_\Delta & D_{H-1} & <_\Delta & \cdots & <_\Delta & D_1 \\ F(D_H) & \supset & F(D_{H-1}) & \supset & \cdots & \supset & F(D_1) \\ I(D_H) & \subset & I(D_{H-1}) & \subset & \cdots & \subset & I(D_1). \end{array}$$

Furthermore, for all $y \in J \setminus F(D_i)$ and $m \in F(D_i)$ we have $r_y \leq r_{D_i} \leq r_m$ for any i. This is the *crucial* property used throughout our proof. Let s be any optimal sequence. If every job in $I(D_1)$ is before every job in $F(D_1)$, then all jobs separated by D_1 are already in $\prec$ order, and consider $I(D_2)$ and $F(D_2)$. Otherwise, let $k_1 \in I(D_1)$ be the last job in s that is after some job from $F(D_1)$, and let m_1 be the last such job from $F(D_1)$ before k_1. That is $s = (X_1 m_1 Y_1 k_1 Z_1)$, where $Z_1 \cap I(D_1) = \varnothing$ and $Y_1 \subset J \setminus F(D_1)$. By the above property, we have $r_{y_1} \leq r_{D_1} \leq r_{m_1}$ for all $y_1 \in Y_1$, which implies that $Y_1 \subseteq R^{BI}_{k_1 < m_1}$. Thus, by subset-restricted interchange, we can insert m_1 *backward* just after k_1 to obtain the alternative optimal sequence $(X_1 Y_1 k_1 m_1 Z_1)$. Following in this way, inserting the last job in $F(D_1)$ *backward* after k_1 until there are no such jobs, we obtain sequence s_1 which is an optimal sequence with the property that $I(D_1)$ is before $F(D_1)$. Continuing similarly, if $I(D_2)$ is before $F(D_2)$ in s_1, then all jobs separated by D_2 are already in $\prec$ order, and consider $I(D_3)$ and $F(D_3)$. Otherwise, let $k_2 \in I(D_2)$ be the last job in s_1 after some job from $F(D_2) \setminus F(D_1)$ (since $I(D_2) \subset I(D_1)$ and $I(D_1)$ is before $F(D_1)$ in s_1, k_2 cannot be after any job from $F(D_1)$), and let m_2 be the last such job from $F(D_2) \setminus F(D_1)$. Similarly, we have $s_1 = (X_2 m_2 Y_2 k_2 Z_2)$, where $Z_2 \cap I(D_2) = \varnothing$ and $Y_2 \subseteq J \setminus F(D_2)$. As above, we have that $r_{y_2} \leq r_{D_2} \leq r_{m_2}$ for all $y_2 \in Y_2$, which implies that $Y_2 \subseteq R^{BI}_{k_2 < m_2}$. Thus we can insert m_2 *backward* just after k_2 to obtain the sequence $(X_2 Y_2 k_2 m_2 Z_2)$. When all such jobs in $F(D_2) \setminus F(D_1)$ have been inserted after k_2, we obtain sequence s_2, an optimal sequence with the property that $I(D_2)$ is before $F(D_2)$ and $I(D_1)$ is before $F(D_1)$. Continuing similarly, we obtain s_i for $i = 3, 4, \ldots, H$. Then s_H is an optimal sequence with the property that $I(D_i)$ is before $F(D_i)$ for $i = 1, 2, \ldots, H$. Thus s_H is a linear extension of $\prec$, and we have that $\prec$ is a dominance order indeed. ■

Theorem 2.4 also has interesting complexity implications. Let us further augment P_Δ by adding new least and greatest elements 0 and 1, and call it $P_{0,1} = (J_{0,1}, \lessdot_{0,1})$, where $J_{0,1} = J_\Delta \cup \{0,1\}$ and $\lessdot_{0,1} = \lessdot_\Delta \cup (\{0\} \times J_\Delta) \cup (J_\Delta \times \{1\})$. Then $\prec$ admits an interval representation using intervals $I_j = [x(j), y(j)]$ $(j \in J)$ on the set $S = \Delta \cup \{0,1\}$ linearly ordered by $\lessdot_{0,1}$, where for $j \in J$, $x(j) = \max\{l \in S \,|\, l \lessdot_{0,1} j\}$ and $y(j) = \min\{l \in S \,|\, j \lessdot_{0,1} l\}$. Partial orders $(P, \leq_P)$ that possess such an interval representation on a linearly ordered set (where $k \leq_P m \Leftrightarrow I_k$ lies to the left of I_m) are called *interval orders* [11]. This leads to the following corollary for $1/r_j/f_{\max}$.

Corollary 2.1 *Problem $1/r_j, prec/f_{\max}$ remains NP-hard in the strong sense even with interval-order precedence constraints.*

2.4.1 Linearly ordered $\geq_f$

In this section, we consider separately the special case where $\geq_f$ is a linear order. This means that for *each* pair of jobs i and j either $f_i(t) \geq f_j(t)$ *for all* t, or $f_j(t) \geq f_i(t)$ *for all* t. That is, we can *completely* arrange the jobs in nonincreasing f order according to $\geq_f$. We examine an equivalent pyramid representation for $\prec$ ([8] ,[9],[29] and [12]), and show how this representation does *not* extend to the nonlinearly ordered general case.

For the linearly ordered case, we are able to represent the adjacent interchange order in the plane using the r and f orders as the x and y axes respectively. Here jobs are represented by points with preferences toward the origin, i.e., $k \lessdot m$ if k is closer to the origin than m in both the r and f orders. The principal ideals and filters in $\lessdot$ are represented by quadrants through these points. That is, let job i be represented by the point (r_i, f_i). If we divide the plane into quadrants using the lines $r = r_i$ and $f = f_i$, then the SW and NE quadrants correspond to $I(i)$ and $F(i)$, the principal ideal and filter in $\lessdot$ for job i (see Figure 2.4).

This planar representation was used by Merce [29] to derive a dominance order for the problem of minimizing the makespan in the presence of release times and deadlines $(1/r_j, \overline{d_j}/C_{\max})$. Fontan [12] noted that if we consider due dates instead of

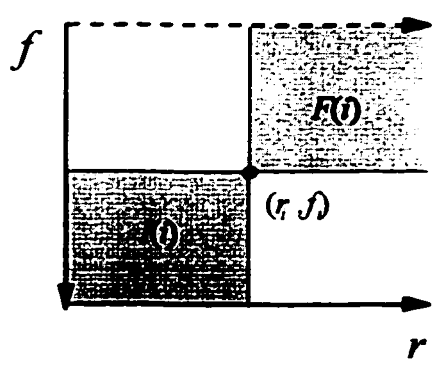

Figure 2.4: Ideals and filters for $\lessdot$.

deadlines then the same order is a precedence order for the lateness model $1\,/r_j\,/L_{\max}$. They did not consider the adjacent interchange order explicitly ([8] and [9]), rather they defined their order using certain extreme points in the plane, called summits. These *summits*, represented by S_i (i = 1 to N), are the jobs that form a staircase boundary in the plane and satisfy the property that their SE quadrant minus the boundary is empty. The summits are completely ordered $S_1 \lessdot S_2 \lessdot \ldots \lessdot S_N$. For each summit S_i, Merce and Fontan define a *pyramid* $P(S_i)$, which is its NW quadrant without its boundary lines but including S_i. Jobs are classified using pairs that represent their membership in pyramids. For $j \in J$, they define $u(j) = \min\{i \,|j \in P(S_i)\}$, and $v(j) = \max\{i\,|j \in P(S_i)\}$. With these, they define their partial order $\prec'$ by $k \prec' m \Leftrightarrow v\,(k) < u\,(m)$. The planar representation for the 5-job problem of Example 1 is shown in Figure 2.5. Jobs 3 and 5 are the summits S_1 and S_2, respectively. $P(S_1) = \{1,2,3,4\}$ and $P(S_2) = \{5\}$. We have $v(i) = 1$ for $i = 1,2,3,4$ and $u(5) = 2$, which implies by definition of $\prec'$ that jobs 1,2,3 and 4 must precede job 5. Notice that the unique optimal sequence $(1,2,3,4,5)$ is a linear extension of $\prec'$ (as we would expect), but not a linear extension of $\lessdot$ (as we saw earlier). Example 2 is another instance of $1/r_j/L'_{\max}$, this time with $N = 9$ summits. For this instance, the optimal $L'_{\max} = 114$ and the sequence $(1,S_1,k,S_2,S_3,S_4,S_5,3,S_6,S_7,m,2,S_8,S_9,4)$ is an optimal sequence, which is also a linear extension of $\prec'$. To further illustrate how $\prec'$ is defined, consider jobs k and m in Figure 2.6. Here $k \prec' m$, since $v(k) = 5 < 6 = u(m)$.

Example 2 *A 15 job instance of* $1/r_j/L'_{\max}$ *with* $N = 9$ *summits.*

j	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	k	m	1	2	3	4
r_j	15	26	34	40	48	57	65	65	73	19	51	8	22	30	38
p_j	6	8	8	7	6	10	2	5	7	5	3	5	8	8	10
q_j	53	46	40	36	36	27	19	13	9	32	17	43	16	22	6

Let us now apply our general dominance order $\prec$ to this special linearly ordered case: we consider a diagonal representation for $\prec$, using the set of corner point boundary jobs $M = \{M_i \,|i = 1, 2, \ldots, H + 1\}$, and the set of artificial jobs on their inscribed diagonal $\Delta = \{D_1, D_2, \ldots, D_H\}$ (see Figure 2.6). The set $M \subseteq S$ is the subset of boundary jobs with empty SE quadrants, and again we call Δ the set of diagonal jobs. We augment the partial order $P = (J, <)$ by these diagonal jobs and call it $P_\Delta = (J_\Delta, <_\Delta)$, where $J_\Delta = J \cup \Delta$ and $<_\Delta$ is the planar order with these diagonal jobs included. Jobs $k<m$ ($k,m \in J$) are separated by Δ (are Δ-separated) if there exists a D_i ($i = 1, 2, \ldots, H$) such that $k \in I(D_i)$ and $m \in F(D_i)$. Notice that $v(k) < u(m)$ when k and m are Δ-separated. It can be easily verified for $k<m$ ($k,m \in J$) that $k \prec' m$ if and only if k and m are separated by Δ, i.e. $k \prec m$. As an example of this equivalence, consider again Example 1 in Figure 2.5: We see that diagonal job D_1 separates job 5 and jobs 1,2,3 and 4, and this is the *only* separation present. Thus, by the diagonal representation of $\prec$, jobs 1,2,3 and 4 must precede job 5, and $\prec$ and $\prec'$ define the same precedence orders indeed.

The pyramid representation used for $\prec'$ can be reinterpreted in partial order terminology to explain *why it does not* extend to the nonlinearly ordered higher dimensional case. The summits S_i ($i = 1, 2, \ldots, N$) are *maximal elements* of a related partial order $<^c$, which is defined by $k<^c m \Leftrightarrow r_k < r_m$ and $f_k <_f f_m$, the conjugate of $<$. Two partial orders on the same set are *conjugate* if every pair of distinct elements

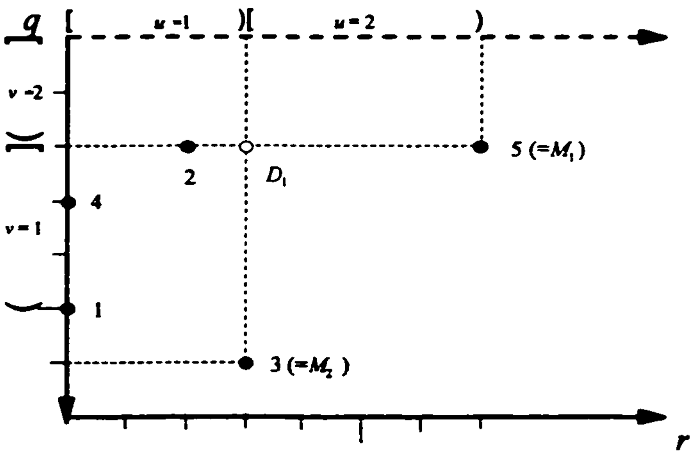

Figure 2.5: Planar representation for Example 1.

is comparable in exactly one of these partial orders. This is clearly the case if we compare the principal ideals and filters for $\lessdot$ and $\lessdot^c$, using the planar representation. For $\lessdot$, these are the *SW* and *NE* quadrants, respectively, with the boundary lines included. For $\lessdot^c$, these are the *NW* and *SE* quadrants minus the boundary lines (compare Figures 2.4 and 2.7). By a well-known theorem of Dushnik and Miller [6] from partial order theory, $\lessdot^c$ exists if and only if $\dim(\lessdot) \leq 2$. In this context, the pyramids are just the ***principal ideals*** of $\lessdot^c$. The $u(j)$ and $v(j)$ defined in [9] implicitly use the fact that $\lessdot^c$ has an ***interval containment order*** representation, i.e., there exist intervals $\{I_j | j \in J\}$ such that $i \lessdot^c j$ iff $I_i \subset I_j$ for $i, j \in J$. In the pyramid representation of $1/r_j/L_{\max}$, these intervals are just $I_j = [r_j, d_j]$. On the other hand, by the same theorem of Dushnik and Miller [6], a poset has an interval containment representation iff its dimension is 2. Thus, the $u(j)$ and $v(j)$ can be defined for *any* problem for which $\dim(\lessdot) \leq 2$, but it can be defined *only* for such problems. Of course, $\dim(\lessdot) \leq 2$ is equivalent to $\geq_f$ being a linear order, so the $u(j)$ and $v(j)$ can be defined *only* in this case. Thus we see that the diagonal representation for $\prec$ has

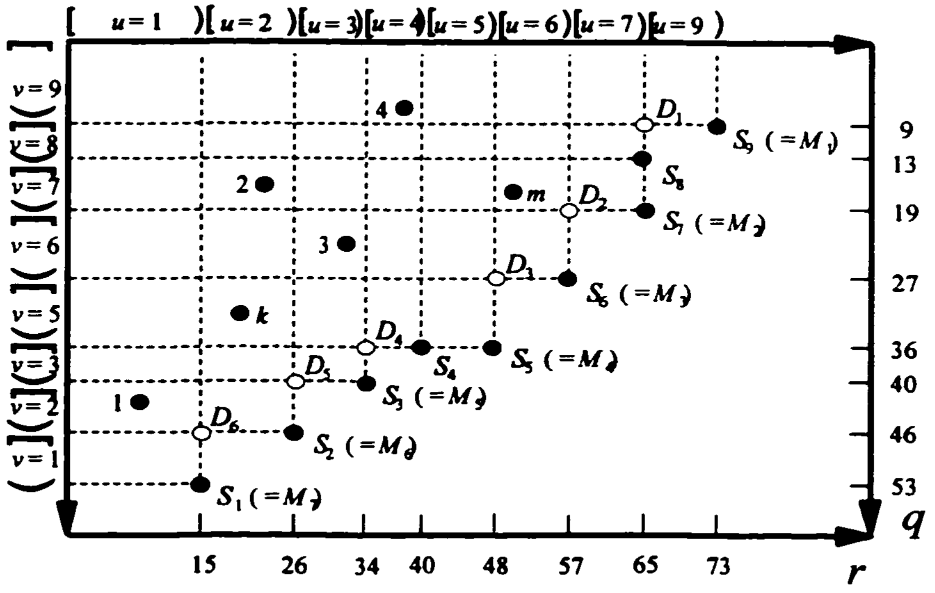

Figure 2.6: Planar representation for Example 2.

the advantage that it does *not* require that $\lessdot$ possess a conjugate $\lessdot^c$, and thus is *not restricted* by the dimension of $\geq_f$.

It is interesting to note that the original proof due to Erschler et al. [9], for $1\,/r_j\,/L_{\max}$, used ***pairwise interchange.*** This proof can also be modified to carry over to other cost functions f *when* $\geq_f$ is a linear order. We chose to present a proof using backward insertion, however, because this extends to the (nonlinearly ordered) general case. A proof using forward insertion can also be obtained by proceeding in the opposite direction. This is due to the duality of the operations and regions for the linearly ordered case. However, the proof based on forward insertion is not extendable to the general case either, as the duality of regions no longer holds.

Although this section deals only with linearly ordered $\geq_f$, it covers a number of well-studied scheduling problems. In addition to the ones studied in ([8] and [9]), we mention one further example.

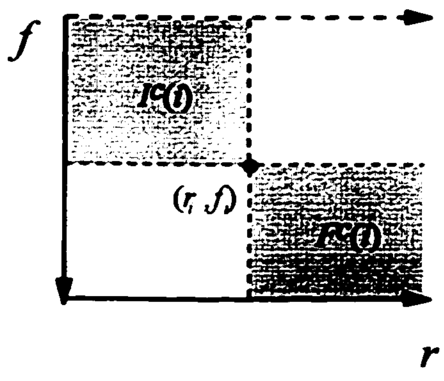

Figure 2.7: Ideals and filters for $<^c$.

Corollary 2.2 *Partial order $\prec$ is a dominance order for $1\,/r_j/\max w_jC_j$ and, in this case, $\geq_f$ is the order which orders the jobs in nonincreasing w_j order.*

Theorem 2.4 and Corollary 2.1 also have interesting complexity implications, they yield the following tightening of previously known complexity results ([26] and [17], respectively).

Corollary 2.3 *Problems $1\,/r_j, prec/\,L_{\max}$ and $1\,/r_j, prec/\max w_jC_j$ remain NP-hard in the strong sense even with interval-order precedence constraints.*

Corollary 2.3 is interesting, as interval orders have a very special restricted structure [11], which does not seem to help in reducing the complexity of the scheduling problems mentioned. This is in contrast with the result of [31] which shows that $Pm\,/p_j = 1, prec/\,C_{\max}$ is polynomially solvable *for* interval-ordered precedence constraints.

Recall that when the adjacent interchange order $<$ *itself* is a linear order, it defines an optimal sequence. It can easily be seen, that $\prec$ is equivalent to $<$ in this case, and so it also defines an optimal sequence. This means that $\prec$ also solves some well-known special cases solved by Jackson's rule [20] or Lawler's method [25]: For $1/r_j/L'_{\max}$, $f_i(t) = t + q_i$ and the jobs are linearly ordered by $f_i \geq_f f_j \Leftrightarrow q_i \geq q_j$. So both $\lessdot$ and $\prec$ are linear orders and define an optimal sequence, for example, *when* the f order is the same as the r order (the agreeably ordered case of $1/r_j/L'_{\max}$ in which $r_i \leq r_j \Leftrightarrow q_i \geq q_j$); or one of the orders is trivial (i.e., the f order is linear and *all* the jobs have the same release time $1//L_{\max}$); or *all* the jobs have the same cost function $(1/r_j, d_j = d/L_{\max})$. Similar comments apply to the corresponding special cases of $1/r_j/\max w_jC_j$ and $1/r_j/f_{\max}$.

2.4.2 Weighted maximum lateness

For this problem the $\geq_f$ order is *no longer linear,* thus we proceed as in the general case. For the weighted maximum lateness problem, $f_j(t) = w_j(t-d_j)$ for all $j \in J$. Let $g_j(t) = f_j(t)+L$, where the constant L is chosen so that $L \geq \max\{w_jd_j \mid j \in J\}$. That is, $g_j(t) = w_jt + q_j$, where $q_j = L - w_jd_j \geq 0$. We see that $f_k \geq_f f_m \iff (w_k \geq w_m)$ and $(q_k \geq q_m)$. Thus, we can represent the adjacent interchange order $\lessdot$ and the interchange regions using this *2-dimensional* representation of $\geq_f$. This means that our general definitions from before reduce to the following for $1/r_j/\max w_jL_j$.

$$\begin{aligned}
k \lessdot m &\Leftrightarrow (r_k \leq r_m) \text{ and } (w_k \geq w_m) \text{ and } (q_k \geq q_m)\\
R^{PI}_{k\lessdot m} &= \{j \mid r_j \leq r_m, w_j \leq w_k, q_j \leq q_k\}\\
R^{FI}_{k\lessdot m} &= \{j \mid w_j \leq w_k, q_j \leq q_k\}\\
R^{BI}_{k\lessdot m} &= \{j \mid r_j \leq r_m\}
\end{aligned}$$

We derive the Δ boundary by modifying the greedy procedure presented earlier. Once again, we assume that there are K distinct r values $r^1 > r^2 > \cdots > r^K$ and let $w^i = \max\{w_j \mid r_j = r^i\}$ and $q^i = \max\{q_j \mid r_j = r^i\}$ for $i = 1, 2, \ldots, K$. We use the same notation as before for the boundary jobs M_i and the diagonal jobs D_i.

```
**Algorithm Δ for** $1/r_j/\max w_jL_j$
let $(r_{M_1}, q_{M_1}, w_{M_1}) = (r^1, q^1, w^1)$, $l = 1$, $i = 2$
while $i \leq K$ do
begin
```

```
        if $(q_{M_l} \geq q^i)$ and $(w_{M_l} \geq w^i)$ then $i := i + 1$; {i.e. $(q_{M_l}, w_{M_l}) \geq_f (q^i, w^i)$}
        else
        begin
            let $(r_{M_{l+1}}, q_{M_{l+1}}, w_{M_{l+1}}) = (r^i, \max\{q^i, q_{M_l}\}, \max\{w^i, w_{M_l}\})$ {This defines
            $M_{l+1}$}
            let $(r_{D_l}, q_{D_l}, w_{D_l}) = (r^i, q_{M_l}, w_{M_l})$ {This defines $D_l$}
            $l := l + 1$;
        end;
    end;
```

If we represent $\lessdot$ in 3-dimensions with q, w, and r as the x, y, and z axes, respectively, then (q^i, w^i) is the least upper bound according to $\leq_f$ of the jobs on the plane $r = r^i$. (This is well defined by the finiteness of J.) Symbolically, $(q^i, w^i) = \max_2 \{(q_j, w_j) \, | r_j = r^i\}$, where we define $\max_2 \{(q_j, w_j) \, | j \in I\} = (\max_{j \in I} q_j, \max_{j \in I} w_j)$. Taking least upper bounds, we greedily construct the sets of possibly artificial jobs Δ and M. These jobs are on the boundary of a *step pyramid,* which contains all of the original jobs inside or on its surface (see Figure 2.8 for an example with $H = 6$). The definition of $\prec$ is analogous to the general case: $k \prec m$ if k and m are Δ-separated. From Theorem 2.4 we then have the following corollary for the problem $1 / r_j / \max w_j L_j$.

Corollary 2.4 *Partial order $\prec$ is a dominance order for $1 / r_j / \max w_j L_j$.*

2.5 Computational experiment

We consider a branch and bound algorithm for $1/r_j/\max w_j C_j$ that follows Potts' implementation [34]. Potts uses an 'adaptive' branching technique to *fix* jobs at both ends of the schedule. More precisely, each node of the search tree is represented by a pair (σ_1, σ_2), where σ_1 and σ_2 are the *initial* and *final* partial sequences, respectively. The immediate successor of (σ_1, σ_2) is either of the form: $(\sigma_1 i, \sigma_2)$ for a *type 1* branching, or $(\sigma_1, i\sigma_2)$ for a *type 2* branching, where i is an *unfixed* job $i \in J \setminus (\sigma_1 \cup \sigma_2)$.

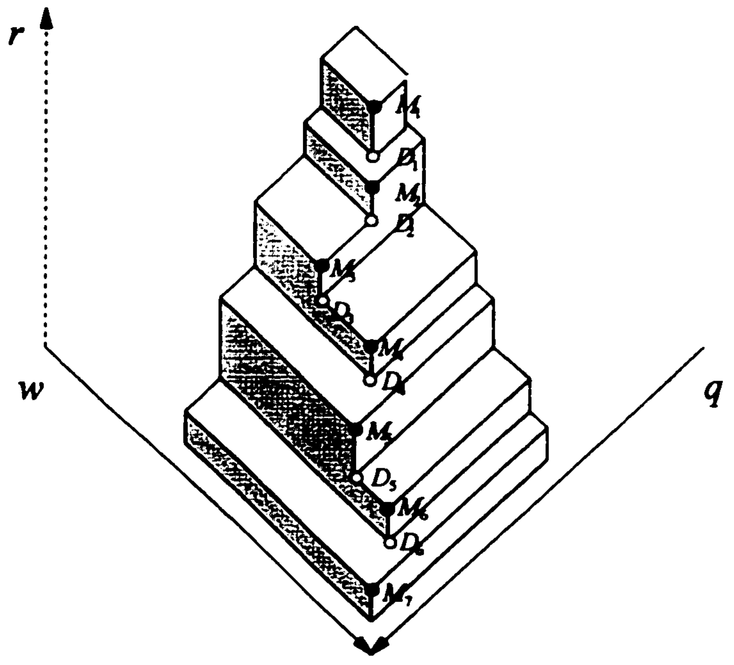

Figure 2.8: Staircase structure for problem $1\,/r_j\,/\max w_jL_j$.

The types of the branchings of the tree, are all the same within a level, but in general, they will differ between levels. The type for a given level k is fixed on the very first visit to level k according to the following rule: branch in the direction of the fewest number of ties at the minimum lower bound. Let n_1 and n_2 be the number of ties at the minimum lower bound for type 1 and type 2 branchings, at level k. If $n_1 < n_2$ the next branching is of type 1, while if $n_2 < n_1$ then the branching is of type 2. If $n_1 = n_2$ then the branching is the same as the previous level. Finally, the search strategy is to branch to the newest active node with the smallest lower bound.

The branch and bound algorithm was coded in Pascal and run on a Sun Sparc5 workstation. The experiment consisted of 24 groups with 100 problems in each group. For each problem, with $n = 100$ or 500 jobs, $3n$ integer data (r_i, p_i, w_i), were generated. Processing times p_i and weights w_i were uniformly distributed between $[1, 100]$ and $[1, W]$ for $W = 10$ and 100 respectively, while release times were uniformly distributed in $[0, n \cdot 50.5 \cdot R]$ for $R \in \{0.2, 0.4, 0.6, 0.8, 1.0, 2.0\}$, following the data generation technique used by Hariri and Potts [16]. Several versions of the algorithm were tested both with and without the dominance order $\prec$, for each group we present only the best results. An upper limit of 1,000,000 was used for the maximum number of nodes in the branch and bound tree. The data in Table 2.4 shows that this limit was exceeded only for a few problems, i.e., very few problems remained unsolved. Most of these seem to be concentrated in the groups defined by $n = 100$, $W = 100$ and $R = 0.4$ or 0.8. Inspection of the results reveals that the number of problems solved is roughly the same with or without using the dominance order $\prec$. Nonetheless, the dominance order seems to ***significantly improve*** the running time of the algorithm. The times shown in Table 2.4 are the *total* times used for solving the 100 problems in the group, using the format (min:sec) or (hrs:min:sec). The average CPU time for the version with the dominance order is 42 percent of that for its counterpart without it, resulting in a savings of 58 percent. This saving from using $\prec$ appears to be due to the reduction in the number of branchings and lower bounds that need to be computed at each node. Note that if the precedence order $\prec$ is used, then lower bounds for type 1 and type 2 branchings, need not be calculated for all the unfixed jobs. For type 1 branchings lower bounds only need to be calculated for those jobs with no unfixed predecessors in $\prec$, i.e., the minimal jobs. Similarly, for type 2 branchings lower bounds only need to be calculated for those jobs with no unfixed successors in $\prec$, i.e., the maximal jobs. In summary, the branch and bound algorithm with the dominance order $\prec$ appears to be a very fast and effective solution method for large problems.

2.6 Summary

In this Chapter, we introduced a new method of pairwise job interchange, which incorporates subset-restrictions on the intermediate sequence. We applied this technique to derive new dominance results for the problem $1/r_j/f_{\max}$, which includes the problems $1/r_j/L_{\max}$, $1/r_j,\overline{d_j}/C_{\max}$, $1/r_j/\max w_jC_j$, and $1/r_j/\max w_jL_j$ as special cases. In particular, we derive a new dominance order $\prec$ that is a suborder of the adjacent interchange order. We extend the pyramid dominance order of Erschler et al. ([8] and [9]) for $1\,/r_j\,/L_{\max}$ to the general problem $1\,/r_j/\,f_{\max}$. The order $\prec$ is shown to belong to a special class of partial orders, the interval orders. This has the complexity implication that these problems are *strongly NP-hard* for interval-ordered jobs. We tested the effectiveness of the dominance order in a branch and bound algorithm for the problem $1\,/r_j\,/\max w_jC_j$. The results indicated that the dominance order was quite effective for this problem, reducing the total computation time by 58 percent.

$n = 100$				
$W = 10$	CPU time		Solved	
R	with $\prec$	without $\prec$	with $\prec$	without $\prec$
0.2	00:01	00:05	100	100
0.4	12:07	43:17	100	98
0.6	00:18	00:34	100	100
0.8	00:19	40:21	100	99
1.0	00:17	00:27	100	100
2.0	00:09	00:14	100	100
$W = 100$	CPU time		Solved	
R	with $\prec$	without $\prec$	with $\prec$	without $\prec$
0.2	0:00:42	0:01:01	100	100
0.4	3:16:27	4:32:02	90	89
0.6	0:02:31	0:07:16	100	99
0.8	0:42:20	0:56:56	92	92
1.0	0:00:22	0:00:32	100	100
2.0	0:00:11	0:00:14	100	100

$n = 500$				
$W = 10$	CPU time		Solved	
R	with $\prec$	without $\prec$	with $\prec$	without $\prec$
0.2	00:44	00:35	100	100
0.4	20:33	35:19	100	100
0.6	12:51	26:59	100	100
0.8	13:08	28:09	100	100
1.0	12:51	33:52	100	100
2.0	09:07	27:02	100	100
$W = 100$	CPU time		Solved	
R	with $\prec$	without $\prec$	with $\prec$	without $\prec$
0.2	0:00:27	0:00:35	100	100
0.4	0:57:54	7:50:25	99	98
0.6	0:49:48	1:26:49	100	99
0.8	0:39:46	0:57:51	100	100
1.0	0:39:44	0:56:32	100	100
2.0	0:22:08	0:31:32	100	100

Table 2.4: **Results of computational experiment for problem $1/r_j/\max w_jC_j$.**

Chapter 3 Minimizing makespan in the two-machine flow-shop with release times

3.1 Introduction

Minimizing the makespan in a flow-shop environment is a classical scheduling problem. The simplest case of this problem, the two-machine flow-shop problem $F2//C_{\max}$ on n jobs, is solved in $O(n \log n)$ time by Johnson's rule [21]. When arbitrary release times are added, the problem $F2/r_j/C_{\max}$ becomes strongly NP-hard [27]. This result has led researchers to consider different approximation and branch and bound algorithms. Potts [35] developed several approximation algorithms and analyzed their worst case performance. The best of these algorithms has a worst case performance ratio of 5/3 with time complexity $O(n^3 \log n)$. Later, polynomial approximation schemes were developed for the problem by Hall [18] and Kovalyov and Werner [22]. Grabowski [14] presented a branch and bound algorithm for the problem $F2/r_j/L_{\max}$, which can also be used for the problem $F2/r_j/C_{\max}$. Grabowski's algorithm used a new type of branching scheme that exploited certain dominance properties of a critical path. Tadei et al. [41] tested several branch and bound algorithms for the problem $F2/r_j/C_{\max}$. They tested the effectiveness of various lower bounds and branching schemes. They have classified instances as "easy" or "hard" according to the distribution of the release times. From the easy class they have solved problems with up to 200 jobs. While from the hard category they were able to solve instances with up to 80 jobs within 300 seconds. They have also found Grabowski's dichotomic branching scheme less effective for the problem $F2\,/r_j/\,C_{\max}$ than their n-ary branching scheme.

In this Chapter, we consider a new branch and bound algorithm for solving the problem $F2\,/r_j/\,C_{\max}$, with the following main features. We use an adaptive n-ary branching rule, that fixes jobs at both ends of the schedule, similar to the one used by Potts [34] for the problem $Fm//C_{\max}$. We present a new dominance order on the job set, which is derived by using subset-restricted interchange. Although the proof of validity for the dominance order is quite elaborate, its application requires only a very fast and simple procedure at the root of the branch and bound tree. We also incorporate a simple decomposition procedure which proved to be especially effective for problems with large job release times. We use four very quickly computable lower bounds at each node of the tree. The algorithm represents an effective and very fast tool for solving large instances of the strongly NP-hard problem $F2\,/r_j/\,C_{\max}$. In a large scale computational experiment, the algorithm has solved in a few seconds 1,714 of the 1,800 randomly generated test problems with up to 500 jobs. We have also gained the insight that the relatively few problems which were left unsolved had their parameter which determines the range of release times concentrated in a very narrow interval. Even for the unsolved problems, the best solution found by the algorithm was, on average, within less than 0.5% of the minimum schedule length.

The Chapter is organized as follows. In the next section we derive several new dominance results for the problem $F2\,/r_j/\,C_{\max}$. In section 3, we present the details of our branch and bound algorithm, a pseudocode for it is found in Appendix A. In section 4, we discuss the results of a large scale computational experiment. In the last section, we summarize our results.

3.2 Dominance results

In this section, we present a new dominance order $\prec$ for problem $F2\,/r_j/\,C_{\max}$. We derive $\prec$ using the new technique of ***subset-restricted interchange***.

3.2.1 Subset-restricted interchange

Recall that the problem $F2//C_{\max}$ is solved by the *Johnson order*: first order the jobs with $a_j \leq b_j$ in *nondecreasing a* order followed by the jobs with $a_j > b_j$ in *nonincreasing b* order [21]. This ordering of the jobs is an adjacent interchange order for the problem $F2//C_{\max}$, and since it also completely orders every pair of jobs, it is an optimal ordering (i.e. an optimal sequence). An adjacent interchange order for the problem $F2/r_j/C_{\max}$ is the *intersection* of the nondecreasing r order and the Johnson order. Note that this order is no longer a complete order, rather it is only a partial order. To emphasize the partitioning of the jobs in the Johnson order we define $J_{a\leq b} = \{j\,|a_j \leq b_j\}$ and $J_{a>b} = \{j\,|a_j > b_j\}$. Next, we formally define the adjacent interchange order $\lessdot$ for the problem $F2/r_j/C_{\max}$.

Definition 3.1 *Adjacent Interchange Order:$k \lessdot m$ if $r_k \leq r_m$ and*

(i) $k, m \in J_{a\leq b}$ and $a_k \leq a_m$ or,
(ii) $k \in J_{a\leq b}$ and $m \in J_{a>b}$ or,
(iii) $k, m \in J_{a>b}$ and $b_k \geq b_m$.

In general, if an adjacent interchange order is only a partial order (and not a complete order), it need not be a dominance order. This is the case for $\lessdot$ defined above, if we consider the instance of $F2/r_j/C_{\max}$ in Example 3. Here we have $3 \lessdot 1$ ($r_3 = r_1 = 10$, $3, 1 \in J_{a>b}$ and $b_3 = 25 > 15 = b_1$), however, the unique optimal sequence is (1,2,3,4) with $C_{\max} = 125$. Thus we see that $\lessdot$ is not a dominance order since there is no optimal sequence that is a linear extension of $\lessdot$. We use subset-restricted interchange to find a suborder of $\lessdot$ that is a dominance order.

Example 3 *A 4 job problem to illustrate that $\lessdot$ is not necessarily a dominance order.*

j	1	2	3	4
r_j	10	20	10	30
a_j	20	20	30	25
b_j	15	30	25	20

Next we distinguish between different types of pairs in $\lessdot$ that have different interchange properties.

Theorem 3.1 *If $k \in J_{a\leq b}$ and $a_k \leq a_m$, then $k{<}m$ together with the set $R^{FI}_{k<m} = J$ satisfies the Restricted Forward Insertion Condition.*

Proof. Given a sequence s we construct a directed graph $G(s)$ to evaluate $C_{\max}(s)$. Each job $s(j)$ is represented by three nodes with weights $r_{s(j)}$, $a_{s(j)}$, and $b_{s(j)}$ respectively. $G(s)$ has the property that $C_{\max}(s)$ is the length of the longest 'node-weighted' path from 0 to $s(n)$. Each of these paths can be identified by the pair $(s(i), s(j))$, for some $i, j \in [1, n]$, which are the endpoints of the horizontal segments of the path (see Figure 3.1). Then by definition

$$C_{\max}(s) = \max_{\substack{(s(i),s(j))\\ 1\leq i\leq j\leq n}} \left(r_{s(i)} + \sum_{l=i}^{j} a_{s(l)} + \sum_{l=j}^{n} b_{s(l)} \right),$$

and we can evaluate $C_{\max}(s)$ as the maximum over all such pairs $(s(i), s(j))$ in $G(s)$.

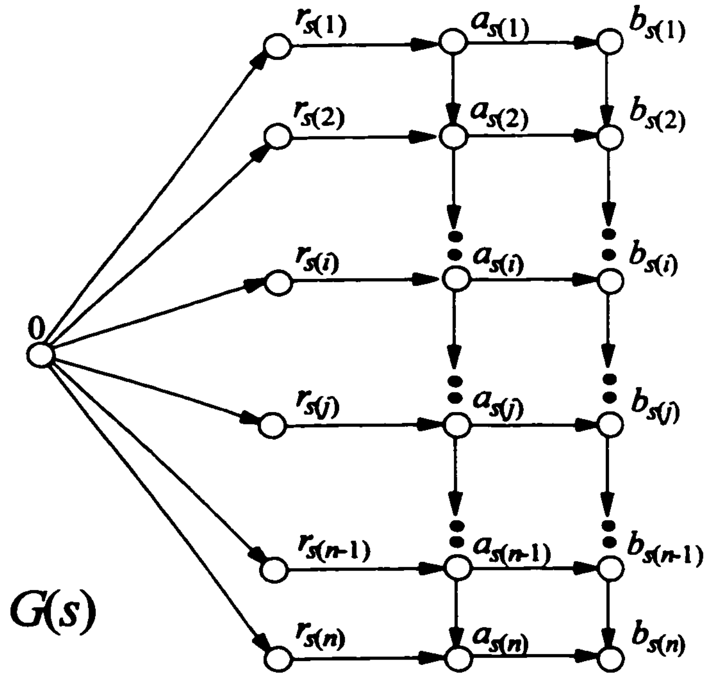

Figure 3.1: Directed graph $G(s)$ for problem $F2/r_j/C_{\max}$.

s_2	s_1			
$(XkmYZ)$	$(XmYkZ)$	*proof*		
(x,k)	(x,m)		$a_k \leq a_m$	
(x,m)	(x,m)			$k \in J_{a\leq b}$
(x,y)	(x,y)			$k \in J_{a\leq b}$
(x,z)	(x,z)			
(k,k)	(m,m)	$r_k \leq r_m$	$a_k \leq a_m$	
(k,m)	(m,m)	$r_k \leq r_m$		$k \in J_{a\leq b}$
(k,y)	(m,y)	$r_k \leq r_m$		$k \in J_{a\leq b}$
(k,z)	(m,z)	$r_k \leq r_m$		
(m,m)	(m,m)			
(m,y)	(m,y)			
(m,z)	(m,z)			
(y,z)	(y,z)			

Table 3.1: Dominating pairs for Theorem 3.1.

Let $s_1 = (XmYkZ)$ be a sequence for pair $k{<}m$ with $k \in J_{a\leq b}$ and $a_k \leq a_m$. We apply forward insertion to s_1 and obtain sequence $s_2 = (XkmYZ)$. We demonstrate that s_2 is not worse than s_1 by showing that for *every* pair of jobs in s_2 there exists a *dominating pair* and corresponding path in s_1 with a not smaller $C_{\max}$ value. Moreover, we show that the choice of the dominating pairs is independent of the intermediate sequence Y, thus we can take the interchange region to be the entire job set, i.e., $R^{FI}_{k<m} = J$. For example, consider pair (k,k) in s_2, then the corresponding dominating pair in s_1 is (m,m). That is,

$$r_k + a_k + b_k + b_m + \sum_{y\in Y} b_y + \sum_{z\in Z} b_z \leq r_m + a_m + b_m + \sum_{y\in Y} b_y + b_k + \sum_{z\in Z} b_z,$$

since $r_k \leq r_m$ and $a_k \leq a_m$.

Table 3.1 gives a dominating pair in s_1 for each pair in s_2. (We use lower case letters x,y or z to refer to arbitrary generic elements of the subsequences X,Y or Z, respectively). The last three columns contain the arguments why they are dominating pairs. ■

Note that $R^{FI}_{k<m} = J$ in Theorem 3.1 means that in fact there are no restrictions on the intermediate set Y, i.e., k can be inserted forward before m around *any*

s_2	s_1			
$(XYkmZ)$	$(XmYkZ)$	$proof$		
(x,y)	(x,y)			$m \in J_{a>b}$
(x,k)	(x,k)			$m \in J_{a>b}$
(x,m)	(x,k)		$b_k \geq b_m$	
(x,z)	(x,z)			
(y_i,y_j)	(m,y_j)	$r_{y_i} \leq r_m$		$m \in J_{a>b}$
(y,k)	(m,k)	$r_y \leq r_m$		$m \in J_{a>b}$
(y,m)	(m,k)	$r_y \leq r_m$	$b_k \geq b_m$	
(y,z)	(m,z)	$r_y \leq r_m$		
(k,k)	(m,k)	$r_k \leq r_m$		$m \in J_{a>b}$
(k,m)	(m,k)	$r_k \leq r_m$	$b_k \geq b_m$	
(k,z)	(m,z)	$r_k \leq r_m$		
(m,z)	(m,z)			

Table 3.2: Dominating pairs for Theorem 3.2.

subsequence $Y \subseteq J$. The next two theorems show cases when the intermediate set Y must satisfy certain real restrictions for the interchange operators to be applicable.

Theorem 3.2 *If $m \in J_{a>b}$ and $b_k \geq b_m$, then $k \lessdot m$ together with the set $R^{BI}_{k \lessdot m} = \{j \,|r_j \leq r_m\}$ satisfies the Restricted Backward Insertion Condition.*

Proof. Similarly, Table 3.2 gives a dominating pair in s_1 for each pair in s_2. ■

Theorem 3.3 *If $k \in J_{a \leq b}$ and $m \in J_{a>b}$ then $k \lessdot m$ together with the set $R^{FI}_{k \lessdot m} = J_{a>b}$ satisfies the Restricted Forward Insertion Condition.*

Proof. Table 3.3 gives the dominating pair in s_1 for each pair in s_2. ■

3.2.2 New dominance order for $F2\,/r_j/\,C_{\max}$

Tadei et al. [41] have established a dominance order for problem $F2\,/r_j/\,C_{\max}$, which can be interpreted as a corollary of Theorem 3.1.

Corollary 3.1 [41] *If $r_k \leq r_m$, $k \in J_{a \leq b}$ and $a_k \leq a_m$ then job k precedes m in an optimal solution.*

s_2	s_1			
$(XkmYZ)$	$(XmYkZ)$	$proof$		
(x,k)	(x,k)		$m \in J_{a>b}$	$Y \subseteq J_{a>b}$
(x,m)	(x,m)		$k \in J_{a\leq b}$	
(x,y)	(x,y)		$k \in J_{a\leq b}$	
(x,z)	(x,z)			
(k,k)	(m,k)	$r_k \leq r_m$	$m \in J_{a>b}$	$Y \subseteq J_{a>b}$
(k,m)	(m,m)	$r_k \leq r_m$	$k \in J_{a\leq b}$	
(k,y)	(m,y)	$r_k \leq r_m$	$k \in J_{a\leq b}$	
(k,z)	(m,z)	$r_k \leq r_m$		
(m,m)	(m,m)			
(m,y)	(m,y)			
(m,z)	(m,z)			
(y,z)	(y,z)			

Table 3.3: Dominating pairs for Theorem 3.3.

Corollary 3.1 is indeed a consequence of Theorem 3.1, since its conditions imply $k \lessdot m$, and so k and m satisfy all the conditions of Theorem 3.1. Therefore, FI can be applied to any sequence of the form $(XmYkZ)$ to insert k before m around any intermediate sequence Y. Since $k \lessdot m$, it is clear that the dominance order of Corollary 3.1 is a suborder of the adjacent interchange order $\lessdot$. Note that this suborder does not contain any pairs with *both* k and $m \in J_{a>b}$. Our dominance order will enrich this suborder by extending it to pairs with $k, m \in J_{a>b}$ too.

Before we derive our dominance order, we introduce a planar representation of the adjacent interchange order $\lessdot$. We represent $\lessdot$ in the plane with the x and y axes replaced by the r and Johnson orders. A Job j is represented by the point (r_j, a_j) if $j \in J_{a \leq b}$ or (r_j, b_j) if $j \in J_{a>b}$. Then $k \lessdot m$ in this representation exactly if k is not to the right or above m. This is demonstrated for the 4-job problem of Example 3 in Figure 3.2. The planar representation here implies $2 \lessdot 4$, $3 \lessdot 1$ and $3 \lessdot 4$.

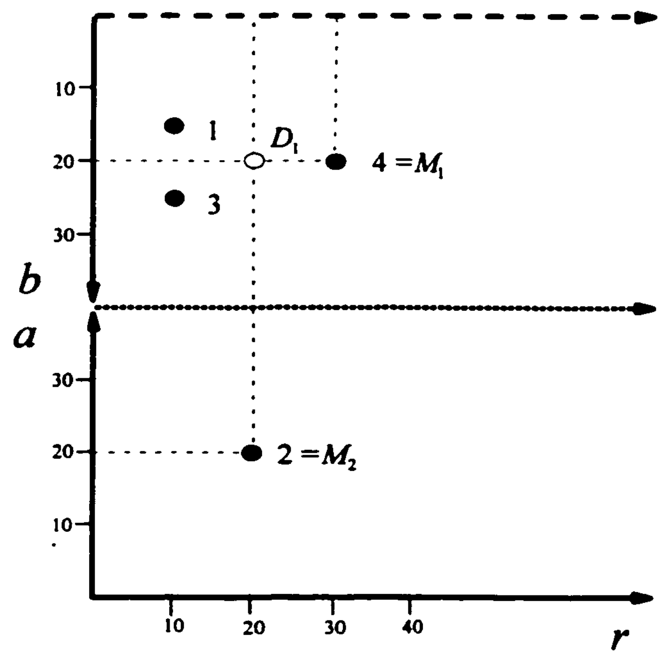

Figure 3.2: Planar representation for Example 3.

The principal ideals and filters in $<$ are obtained by dividing the plane into quadrants using the line $r = r_j$ and the line $a = a_j$ if $j \in J_{a \leq b}$ or $b = b_j$ for $j \in J_{a>b}$. Then the SW and NE quadrants correspond to $I(j)$ and $F(j)$, the principal ideal and filter in $<$ for job j (see Figure 3.3). We add new comparabilities for the jobs in $J_{a>b}$ using subset-restricted interchange. These new comparabilities are defined using certain 'extreme jobs' in the planar representation of $<$. These are the corner-point boundary jobs $M = \{M_i \,|i = 1, 2, \ldots, H + 1\}$, i.e., the jobs with empty SE quadrants that form a descending staircase, and the set of jobs on their inscribed diagonal, the *diagonal jobs* $\Delta = \{D_1, D_2, \ldots, D_H\}$. Note that the diagonal jobs are not necessarily real jobs because of the way they are constructed, rather they may be 'artificial jobs' and are used only for ordering purposes. We augment the partial order $P = (J, <)$ by these diagonal jobs and call it $P_\Delta = (J_\Delta, <_\Delta)$, where $J_\Delta = J \cup \Delta$ and $<_\Delta$ is the planar order with these diagonal jobs included. Jobs $k \lessdot m$ ($k,m \in J$) are ***separated by*** Δ (are Δ-separated) if there exists a D_i ($i = 1, 2, \ldots, H$) such that k is in its principal ideal and m is in its principal filter, i.e., $k \in I(D_i)$ and $m \in F(D_i)$ in P_Δ. Δ induces a ***partition*** of P into separable and nonseparable pairs. Notice that the separable pairs can be of the three types in Definition 3.1, and are not restricted to pairs $k \lessdot m$ with $k \in J_{a \leq b}$. The new pairs we add are precisely the separable pairs in P. We now formally define our dominance order $\prec$.

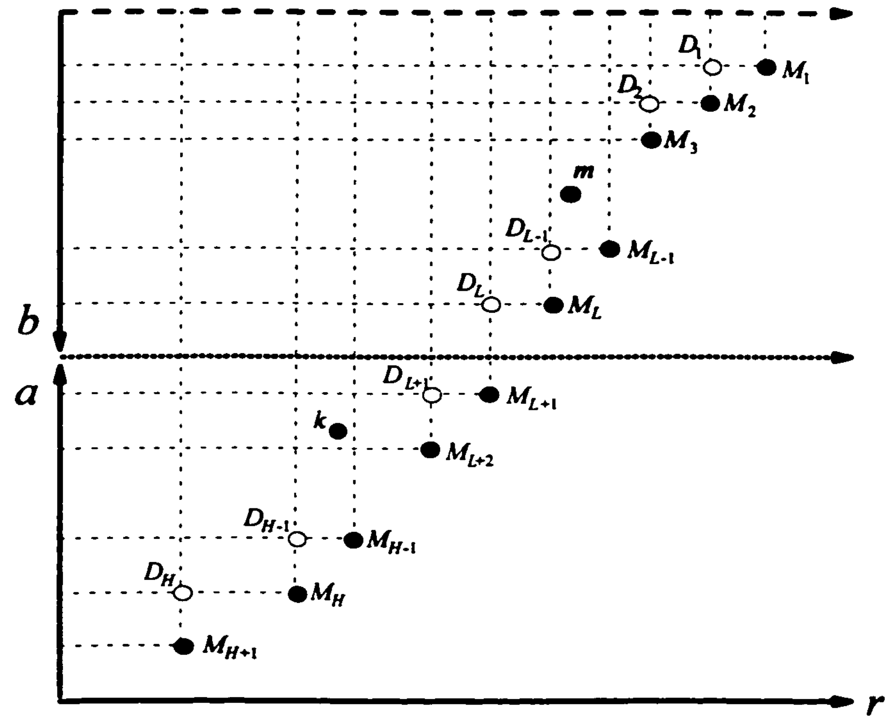

Figure 3.3: Representation of the diagonal jobs Δ.

Definition 3.2 *Dominance order* $\prec$: *For* $k \lessdot m$, *we define* $k \prec m$ *if*

(i) $k \in J_{a \leq b}$ *and* $a_k \leq a_m$ *or,*
(ii) *there exists a* D_i *in* P_Δ *such that* $k \in I(D_i)$ *and* $m \in F(D_i)$.

To illustrate the definition of $\prec$ consider the planar representation for Example 3 in Figure 3.2. Applying condition (i) to pairs $k \lessdot m$ with $k \in J_{a \leq b}$ we see that $2 \lessdot 4$ is the only such pair and since $a_2 = 20 \leq 25 = a_4$ we have $2 \prec 4$. For condition (*ii*) here there are 2 corner-point boundary jobs $M_1 = 4$, $M_2 = 2$, and a single diagonal job D_1 in $J_{a>b}$ with coordinates $(20, 20)$. We see that D_1 separates jobs 2 and 3 from job 4, which gives us (again) $2 \prec 4$ together with the new pair $3 \prec 4$. Finally combining these types of pairs we get that, jobs 2 and 3 precede job 4 in $\prec$.

Theorem 3.4 *Partial order $\prec$ is a dominance order for $F2/r_j/C_{\max}$.*

Proof. Let s be an optimal sequence, we can assume by Theorem 3.1, that s is a linear extension of the suborder for condition (*i*) of the theorem. Thus we need to prove the theorem only for pairs satisfying condition (*ii*).

The following observations immediately follow from the construction of Δ:

$$\begin{array}{ccccccc} D_H & <_\Delta & D_{H-1} & <_\Delta & \cdots & <_\Delta & D_1 \\ F(D_H) & \supset & F(D_{H-1}) & \supset & \cdots & \supset & F(D_1) \\ I(D_H) & \subset & I(D_{H-1}) & \subset & \cdots & \subset & I(D_1). \end{array}$$

Furthermore, we have the following crucial property: for all $y \in J \setminus F(D_i)$ and $m \in F(D_i)$ we have $r_y \leq r_{D_i} \leq r_m$ for any i. Consider a pair k and m satisfying condition (*ii*). If both $k, m \in J_{a \leq b}$, then they also satisfy condition (*i*), and k is already before m in s.

Next we deal with pairs with k and m satisfying condition (*ii*) and for which both $k, m \in J_{a>b}$. In the following, the notation $S|_{J_{a>b}}$ is used to refer to $S \cap J_{a>b}$ for any $S \subseteq J$. If every job in $I(D_1)|_{J_{a>b}}$ is before every job in $F(D_1)$, then all jobs separated by D_1 in $J_{a>b}$ are already in $\prec$ order, and we consider $I(D_2)|_{J_{a>b}}$ and $F(D_2)$. Otherwise, let $k_1 \in I(D_1)|_{J_{a>b}}$ be the last job in s that is after some job from $F(D_1)$, and let m_1 be the last such job from $F(D_1)$ before k_1. That is $s = (X_1 m_1 Y_1 k_1 Z_1)$, where $Z_1 \cap I(D_1)|_{J_{a>b}} = \varnothing$ and $Y_1 \subset J \setminus F(D_1)$. By the above property, we have $r_{y_1} \leq r_{D_1} \leq r_{m_1}$ for all $y_1 \in Y_1$, which implies that $Y_1 \subseteq R^{BI}_{k_1 \lessdot m_1}$. Thus, by Theorem 3.2, we can insert m_1 *backward* just after k_1 to obtain the alternative optimal sequence $(X_1 Y_1 k_1 m_1 Z_1)$. Note that this interchange cannot violate condition (*i*), because $m_1 \in J_{a>b}$ and condition (*i*) could never require it to precede any job. Following in this way, inserting the last job in $F(D_1)$ *backward* after k_1 until there are no such jobs, we obtain sequence s_1 which is an optimal sequence with the property that $I(D_1)|_{J_{a>b}}$ is before $F(D_1)$. Continuing with a similar argument, we obtain s_i for $i = 2, 3, \ldots, L$, where D_L is the last diagonal job in $J_{a>b}$, and s_L is an optimal sequence satisfying condition (i) with the additional property that $I(D_i)|_{J_{a>b}}$ is before $F(D_i)$ for $i = 1, 2, \ldots, L$. This takes care of the pairs with k, $m \in J_{a>b}$.

Now we prove $\prec$ is a dominance order for the remaining pairs with $k \in J_{a\leq b}$ and $m \in J_{a>b}$. Let m_L be the first job in s_L from $F(D_L)$. If there are no jobs from $J_{a\leq b}$ after m_L, then s_L also satisfies the property that $I(D_i)$ is before $F(D_i)$ for $i = 1, 2, \ldots, L$. Otherwise, let $k_L \in J_{a\leq b}$ be the first such job after m_L. By the definition of D_L we have $k_L \lessdot m_L$. Then $s_L = (X_L m_L Y_L k_L Z_L)$ and by the choice of k_L we have that $Y_L \subset J_{a>b} = R^{FI}_{k_L \lessdot m_L}$. Thus, by Theorem 3.3, we can insert k_L *forward* just before m_L to obtain the sequence $(X_L k_L m_L Y_L Z_L)$. Note that this interchange does not violate condition (i), because $Y_L \subset J_{a>b}$ and condition (i) would never require that k_L follow any job from Y_L. Repeating, until there is no such job k_L, we obtain the sequence $s_L^{'}$ satisfying condition (i) with the property that $I(D_i)$ is before $F(D_i)$ for $i = 1, 2, \ldots, L$. We want to demonstrate that $I(D_i)$ is before $F(D_i)$ for $i = L+1, \ldots, H$. Let k_{L+1} be the last job in $I(D_{L+1})$ that is after some job in $F(D_{L+1})|_{J_{a>b}}$, and let m_{L+1} be the last such job before k_{L+1}. Then $s_L^{'} = (X_{L+1} m_{L+1} Y_{L+1} k_{L+1} Z_{L+1})$ where $Z_{L+1} \cap I(D_{L+1}) = \varnothing$ and $Y_{L+1} \subset J \setminus F(D_{L+1})$. Note that Y_{L+1} may contain jobs from both parts of the Johnson partition. Let y_i be the first job in $J_{a\leq b} \cap Y_{L+1}$ after m_{L+1}. Then we have by our crucial property that $r_{y_i} \leq r_{D_{L+1}} \leq r_{m_{L+1}}$, and since $y_i \in J_{a\leq b}$ and $m_{L+1} \in J_{a>b}$ that $y_i \lessdot m_{L+1}$. Furthermore, all the jobs between m_{L+1} and y_i are in $J_{a>b} = R^{FI}_{y_i \lessdot m_{L+1}}$, and thus by Theorem 3.3, we can insert y_i *forward* before m_{L+1}. By repeatedly doing this, we end up with a sequence where there are only jobs from $J_{a>b}$ between m_{L+1} and k_{L+1}. Finally, again using Theorem 3.3, we can insert k_{L+1} *forward* before m_{L+1} to obtain a sequence with k_{L+1} before m_{L+1}. As for the previous case, these interchanges do not violate condition (i). We continue until there is no such k_{L+1} and m_{L+1} and then we obtain the sequence s'_{L+1} that satisfies the property that $I(D_i)$ is before $F(D_i)$ for $i = 1, 2, \ldots, L+1$. We proceed in exactly the same way for the remaining cases for $L+1 < j \leq H$, and we end up with a sequence s'_H satisfying both conditions (i) and (ii). ■

3.3 Branch and bound

In this section, we outline the basic components of the branch and bound algorithm used to solve the problem $F2/r_j/C_{\max}$. A pseudocode for it is given in Appendix A.

3.3.1 Branching rule

We consider a variant of Potts' adaptive branching rule that fixes jobs at both ends of the schedule [34]. More precisely, each node of the search tree is represented by a pair (σ_1, σ_2), where σ_1 and σ_2 are the initial and final partial sequences, respectively. Let S_i denote the set of jobs in σ_i for $i = 1$, 2 and let S be the set of unfixed jobs i.e., $S = J \backslash (S_1 \cup S_2)$. We use $\prec|_S$ to refer to the restriction of $\prec$ to the set S. An immediate successor of (σ_1, σ_2) in the tree is either of the form $(\sigma_1 i, \sigma_2)$ for a *type 1* branching; or $(\sigma_1, i\sigma_2)$ for a *type 2* branching, where i is a minimal or maximal job in $\prec|_S$, respectively. The types of the branchings are all the same within a level of the tree. The type for a given level k is fixed on the very first visit to level k according to the following rule: branch in the direction of the fewest number of ties at the minimum lower bound. Let n_1 and n_2 be the number of ties at the minimum lower bound for potential type 1 and type 2 branchings, at level k. If $n_1 < n_2$ the next branching is of type 1, while if $n_2 < n_1$ then the branching is of type 2. If $n_1 = n_2$ then the branching is the same type as at the previous level.

The search strategy is to branch to the newest active node with the smallest lower bound. We consider two rules to break ties between nodes with the same lower bound. We assume that the jobs are indexed in increasing Johnson order. The rule T_2 breaks ties for type 1 or type 2 branchings by choosing the job with the smallest or largest index, respectively. The other rule T_1 breaks ties for type 1 branchings by choosing the job with the largest principal filter in $\prec|_S$, the smallest index is used as a further tie-breaker. Similarly, for type 2 branchings the rule is to break ties by choosing the job with the largest principal ideal in $\prec|_S$, with the largest index as a further tie-breaker.

3.3.2 Bounds

Upper bounds are calculated for the root node and for the first n nodes, afterward the upper bound is evaluated only at leaf nodes. At the root, the upper bound is the result of the improved ready-Johnson heuristic due to Potts [35]. This heuristic has time complexity $O(n^3 \log n)$ and a worst case performance ratio of 5/3. For the first n nodes, the upper bound is the length of the sequence obtained by concatenating the ready-Johnson sequence between σ_1 and σ_2, which requires $O(n \log n)$ time. For leaf nodes, there are no unfixed jobs and the upper bound is just the length of the sequence obtained by concatenating σ_1 and σ_2.

Since the branch and bound tree may require the computation of lower bounds for a huge number of nodes, it is very important that we use lower bounds whose calculation requires only $O(n)$ time per bound. (We have also experimented with some potentially better lower bounds, requiring $O(n \log n)$ time, but they have noticeably slowed down the algorithm without any substantial increase in the number of problems solved.) We consider four lower bounds for each node (σ_1, σ_2). We calculate lower bounds on the lengths of different paths for the unfixed jobs S, and we combine these in various ways with the actual lengths of the fixed sequences σ_1 and σ_2. For σ_1 we look at the forward problem (with release times included) and we let $C^1(\sigma_1)$ and $C^2(\sigma_1)$ be the completion times on machines 1 and 2, respectively. For σ_2 we consider the reverse problem, in which each job j must be processed first by machine 2 for b_j time, followed by processing on machine 1 for a_j and a 'delivery time' of r_j. We define the completion times $C^1(\sigma_2)$ and $C^2(\sigma_2)$ analogously, and we define $C_{\max}(\sigma_2)$ to be the 'delivery completion time' for σ_2. For S we consider the forward problem, and assume that these jobs cannot start on machines 1 and 2 before $C^1(\sigma_1)$ and $C^2(\sigma_1)$, respectively. Ignoring release times for the jobs in S, let $L_1(S)$ be the completion time on machine 2 of the Johnson sequence on S. Let $L_2(S)$ be the completion time on machine 1 of the jobs in S sequenced in nondecreasing r order, that is the earliest time at which the jobs in S can be completed on machine 1. Similarly, if we relax the capacity constraint on machine 1 and sequence the jobs in nondecreasing $r+a$ order, then the resulting completion time of this sequence on machine 2, $L_3(S)$ is a lower bound on when the jobs in S can complete on machine 2. Our lower bounds are the following:

$$\begin{aligned} LB_1 &= L_1(S) + C^2(\sigma_2) \\ LB_2 &= L_2(S) + C^1(\sigma_2) \\ LB_3 &= L_2(S) + \min_{i \in S} b_i + C^2(\sigma_2) \\ LB_4 &= L_3(S) + C^2(\sigma_2). \end{aligned}$$

Finally, the best lower bound is the maximum of the above four lower bounds and $C_{\max}(\sigma_2)$.

3.3.3 Decomposition and dominance

We find a starting sequence σ_1 by applying the following simple decomposition procedure, which was also used in [41]. Given a sequence s, if there exists a $2 \leq k \leq n$ such that $\min_{k \leq i \leq n} r_{s(i)} \geq C^1_{s(k-1)}$ and $\min_{k \leq i \leq n} [r_{s(i)} + a_{s(i)}] \geq C^2_{s(k-1)}$, then sequence $(s(1), \ldots, s(k-1))$ is an optimal initial sequence. We apply the decomposition procedure for the jobs sequenced in nondecreasing r order, then $\sigma_1 = (s(1), \ldots, s(k-1))$ for the largest k value found. After we have fixed σ_1, we determine the dominance order $\prec$ on the remaining jobs in $S = J \backslash S_1$. Note that the dominance order is calculated only once at the root node. This eliminates the potentially large overhead of having to calculate and update dynamically changing dominance conditions at every node of the search tree.

3.4 Computational experiment

3.4.1 Test problems

For each problem with n jobs $3n$ integer data (r_i, a_i, b_i) were generated. The processing times a_i and b_i were both uniformly distributed between $[1, 100]$. Release times r_i were uniformly distributed in the range $[0, n \cdot 101 \cdot R]$ for $R \in \{0.2, 0.4, 0.5, 0.6, 0.8, 1.0\}$, following the technique used by Hariri and Potts [16]. For each R and n combination, 50 problems were generated. We used the random number generator of Taillard [42] to generate these problems. This means that all our test problems are reproducible by running the problem generation procedure from the same seeds. (The seeds used have been saved and are available from the author on request.)

3.4.2 Results

The branch and bound algorithm was coded in Sun Pascal 4.2 and run on a Sun Sparc5 workstation. Three separate versions of the algorithm were run, testing the effectiveness of the dominance order and the different tie-breakers T_1 and T_2. Algorithm A_1 has the dominance order $\prec$ 'turned on' and uses tie-breaker T_1. Algorithm A_2 also has the dominance order 'turned on', but it uses T_2 to break ties. Finally, algorithm A_3 has the dominance order 'turned off' with tie-breaker T_2. Each version of the algorithm was run until either it obtained the optimal solution or the number of nodes branched from in the tree reached one million for the problem. In the latter case, the problem was declared unsolved.

Table 3.4 contains the results of the computational experiment for the different R values. For each group of 50 problems we report: the total CPU time required for the 50 problems (denoted by *total CPU*); the average CPU time for the solved problems in the group (denoted by *avg CPU*); the total number of nodes in all of the trees (denoted by *total nodes*); the number of problems solved (denoted by *solved*); and for each of the unsolved problems we calculate the gap between the best solution obtained and the smallest lower bound among the left-over nodes in the tree, the maximum gap (denoted *max gap*) is the largest of these over all the problems in the group.

Our results indicate that the difficulty in solving a problem depends *much more* on the value of R than on the number of jobs. The actual size of the problem for the branch and bound algorithm is determined by how many jobs are fixed by the decomposition procedure. In general, as R increases the percentage of the jobs fixed increases. Recall, in the model $n \cdot 101$ is the expected total processing time of the jobs, and the range of the release times is $[0, n \cdot 101 \cdot R]$. For small R values ($R < 0.4$), the decomposition procedure is ineffective and fixes less than 2 percent of the jobs. However, since all the jobs are ready relatively early, release times play less of a role and the problem is easily solved in just a few nodes. For large R values ($R > 0.6$), the opposite is true, the jobs do not become ready simultaneously, there may be gaps in the schedule, and the decomposition procedure fixes over 90 percent of the jobs on average. The difficult problems are those with R values concentrated close to $R = 0.5$, in the range $R \in [0.4, 0.6]$, with the most difficult being $R = 0.5$ itself. For $R = 0.4$ the decomposition procedure fixes only 3 percent of the jobs, while for $R = 0.6$ it fixes a much larger 70 percent of the jobs on average. In spite of this large difference in the effectiveness of the decomposition procedure for $R = 0.4$ and 0.6, we are still able to solve at least 90 percent of these problems, even those with 500 jobs. Whereas, for $R = 0.5$ the decomposition procedure fixes around 20 percent of the jobs, and the problems appear to be more difficult with a clear decreasing trend in the number of problems solved as n increases.

Recall, a problem was 'unsolved' when the number of nodes branched from reached a million. Thus the total number of nodes for a group of 50 problems, contains a million nodes for each unsolved problem. As can be seen, the algorithm typically generated much fewer nodes for the solved problems. In those groups where there were unsolved problems, the solution obtained was nearly optimal as measured by the maximum gap. The largest such gap was on the order of 1 to 2 percent, meaning that in the worst case we were within this range of the optimal solution. The average gap was much smaller, less than 0.5 percent.

Comparing the different versions of the algorithm, we found that versions A_1 and A_2 (with the dominance order included) together always solved as many problems as version A_3 (without the dominance order included), and in some cases a few more. The main benefit of using the dominance order is the substantial reduction in the CPU time required to solve a group of problems. This reduction in CPU time is approximately 80 percent over all the groups, although sometimes it is more than this. Consider for example the group of problems with $R = 0.4$ and $n = 200$: here A_1 and A_2 both took approximately 37 minutes to complete, while A_3 took over 12.5 hours to do so. We also found that this speed-up factor tends to increase as n increases. This explains why we did not run algorithm A_3 for groups with $n = 500$ jobs, as it would have required excessively long times. The use of the different tie-breaking rules in A_1 and A_2 did not result in any clearly identifiable differences in the relative performance.

3.5 Summary

This Chapter considered a new branch and bound algorithm for the problem $F2\,/r_j/\,C_{\max}$. The main features of the proposed algorithm are: an adaptive branching rule that fixes jobs at both ends of the schedule, and a new dominance order that substantially reduces branching. In general, computational results indicated that the algorithm performed well in solving a large percentage of test problems, some with up to 500 jobs. We found that the difficult problems were those where the range of release times was approximately 1/2 the expected total processing time of the jobs (i.e., $R = 0.5$ in our model). Even when we failed to solve a problem, we were always very close to the optimal solution. We also found that the use of the dominance order resulted in a reduction in computation time of roughly 80 percent. In summary, we feel that the proposed algorithm is relatively more effective than previous solution methods for the problem $F2\,/r_j/\,C_{\max}$, in that it found fewer hard problems and it was also able to solve much larger problems than previous algorithms. The fact that most problems were solved to optimum within a few seconds, means that the algorithm has the potential of being used as a callable subroutine for $F2\,/r_j/\,C_{\max}$-type subproblems generated during the solution of more complex scheduling problems, for example problem $Fm\,/r_j/\,C_{\max}$.

Table 3.4: **Results of computational experiment for problem** $F2\,/r_j/\,C_{\max}$.

n	*alg*	*total CPU* (hrs:min:sec)	*avg CPU* (min:sec)	*total nodes*	*solved*	*max gap* (%)
$R = 0.2$						
40	A_1	0.28	0.0056	107	50	-
	A_2	0.24	0.0048	107	50	-
	A_3	0.24	0.0048	107	50	-
60	A_1	1.28	0.0256	1787	50	-
	A_2	1.25	0.0250	1787	50	-
	A_3	2.76	0.0552	1789	50	-
80	A_1	1.08	0.0216	89	50	-
	A_2	1.09	0.0218	89	50	-
	A_3	0.61	0.0122	89	50	-
100	A_1	1.86	0.0372	111	50	-
	A_2	1.83	0.0366	111	50	-
	A_3	1.02	0.0204	129	50	-
200	A_1	11.95	0.2390	218	50	-
	A_2	12.13	0.2426	218	50	-
	A_3	5.37	0.1074	218	50	-
500	A_1	5:54.83	7.0966	1384	50	-
	A_2	4:41.84	5.6368	1384	50	-
$R = 0.4$						
40	A_1	1:28.11	0.0050	1,001,861	49	0.14
	A_2	1:04.16	0.0046	1,001,861	49	0.38
	A_3	3:21.83	0.0223	1,002,050	49	0.24
60	A_1	6:36.66	0.4769	2,087,118	48	0.74
	A_2	6:32.20	0.4757	2,087,118	48	0.74
	A_3	43:24.47	21.8289	2,857,484	48	0.74
80	A_1	24:25.99	0.7544	3,098,632	47	0.70
	A_2	23:09.80	0.7507	3,098,632	47	0.70
	A_3	1:51:02.56	6.1055	3,118,415	47	0.70
100	A_1	5:44.35	0.0888	1,005,725	49	0.35
	A_2	5:37.48	0.0910	1,005,725	49	0.29
	A_3	38:30.42	0.6191	1,007,805	49	0.29
200	A_1	37:17.79	0.7253	3,018,471	47	0.39
	A_2	37:01.58	1.0868	3,018,471	47	0.39
	A_3	12:31:32.71	17.2627	3,050,546	47	0.39
500	A_1	11:23:54.14	7.0742	5,005,152	45	0.30
	A_2	10:07:29.49	5.4205	5,005,152	45	0.30

Table 3.4: (continued).

n	*alg*	*total CPU* (hrs:min:sec)	*avg CPU* (min:sec)	*total nodes*	*solved*	*max gap* (%)
$R = 0.5$						
	A_1	3:30.76	0.0853	2,027,081	48	0.62
40	A_2	5:59.66	0.0926	2,029,231	48	0.62
	A_3	10:08.70	0.4476	2,051,022	48	0.62
	A_1	18:15.17	0.7275	5,189,188	45	1.37
60	A_2	20:24.29	0.7349	5,189,000	45	1.55
	A_3	38:43.69	13.2302	5,733,770	45	1.37
	A_1	1:14:10.25	1.5020	8,211,107	42	0.79
80	A_2	1:18:09.24	1.4863	8,210,998	42	0.79
	A_3	2:36:10.73	8.3622	8,438,701	42	0.79
	A_1	1:59:06.36	5.3600	9,540,374	41	1.26
100	A_2	1:52:27.48	5.3471	9,540,357	41	1.26
	A_3	4:19:29.52	8.3293	10,143,663	40	1.26
	A_1	3:33:16.72	1.3308	13,019,825	37	0.97
200	A_2	6:15:50.78	1.2841	13,020,073	37	0.97
	A_3	26:17:52.63	4:46.57	13,639,751	37	0.97
	A_1	17:42:05.92	7.0524	14,019,667	36	0.24
500	A_2	15:52:55.39	5.9951	14,019,667	36	0.24
$R = 0.6$						
	A_1	2:59.01	0.0608	1,017,391	49	2.71
40	A_2	2:51.86	0.0594	1,017,379	49	2.31
	A_3	7:38.57	0.5191	1,283,222	49	2.71
	A_1	13:23.16	0.0030	2,001,035	48	0.26
60	A_2	13:19.02	0.0022	2,000,982	48	0.26
	A_3	23:51.37	0.1535	2,001,383	48	0.24
	A_1	16:06.71	1.6158	3,548,901	47	1.30
80	A_2	19:42.08	1.5206	4,496,662	46	1.13
	A_3	42:25.34	5.6657	5,154,014	45	1.47
	A_1	24:59.50	0.1235	4,029,291	46	0.50
100	A_2	24:48.82	0.1193	4,029,291	46	0.50
	A_3	58:36.76	0.4002	5,071,945	45	0.50
	A_1	11:26.63	0.3044	2,082,825	48	0.49
200	A_2	13:44.03	0.0985	3,028,215	47	0.49
	A_3	21:10.28	0.8350	2,139,389	48	0.49
	A_1	3:16:36.69	4.1353	5,066,898	45	0.19
500	A_2	3:14:37.67	7.1349	5,066,898	45	0.19

Table 3.4: (continued).

n	*alg*	*total CPU* (hrs:min:sec)	*avg CPU* (min:sec)	*total nodes*	*solved*	*max gap* (%)
$R = 0.8$						
	A_1	0.12	0.0024	216	50	-
40	A_2	0.10	0.0020	216	50	-
	A_3	0.13	0.0026	216	50	-
	A_1	0.13	0.0026	284	50	-
60	A_2	0.15	0.0030	284	50	-
	A_3	0.17	0.0034	342	50	-
	A_1	0.19	0.0038	218	50	-
80	A_2	0.20	0.0040	218	50	-
	A_3	0.22	0.0044	232	50	-
	A_1	0.25	0.0050	502	50	-
100	A_2	0.29	0.0058	502	50	-
	A_3	0.33	0.0060	746	50	-
	A_1	4:09.77	0.0086	1,000,613	49	0.22
200	A_2	3:58.20	0.0119	1,000,607	49	0.22
	A_3	6:49.31	0.0138	1,001,398	49	0.22
	A_1	6.34	0.1268	11,683	50	-
500	A_2	4.46	0.0892	6,916	50	-
$R = 1.0$						
	A_1	0.07	0.0014	91	50	-
40	A_2	0.10	0.0020	91	50	-
	A_3	0.07	0.0014	91	50	-
	A_1	0.12	0.0024	102	50	-
60	A_2	0.11	0.0022	102	50	-
	A_3	0.12	0.0024	102	50	-
	A_1	0.17	0.0034	169	50	-
80	A_2	0.17	0.0034	169	50	-
	A_3	0.20	0.0040	189	50	-
	A_1	0.19	0.0038	88	50	-
100	A_2	0.21	0.0042	87	50	-
	A_3	0.22	0.0044	101	50	-
	A_1	0.55	0.0110	82	50	-
200	A_2	0.55	0.0110	82	50	-
	A_3	0.56	0.0112	89	50	-
	A_1	2.35	0.0470	81	50	-
500	A_2	2.32	0.0464	81	50	-

Chapter 4
Minimizing maximum lateness in the two-machine flow-shop with release times

4.1 Introduction

We consider the two-machine flow-shop problem with release times where the objective is to minimize the maximum lateness for permutation schedules. This problem denoted by $F2\,/r_j, perm/\,L'_{\max}$, is a well known strongly NP-hard scheduling problem [27]. Much of the underlying interest in the problem $F2\,/r_j, perm/\,L'_{\max}$ stems from the fact that it arises naturally in the solution of more complicated scheduling problems, such as the problem $Fm//C_{\max}$ [24]. The complexity result above suggests that in order to optimally solve the problem $F2\,/r_j, perm/\,L'_{\max}$, it is necessary to resort to some efficient enumerative technique, such as branch and bound.

Branch and bound algorithms are characterized by various components: the branching scheme employed, the different types of upper and lower bounds used, along with other features. Grabowski [14] and Grabowski et al. [15] presented branch and bound algorithms for the problems $F2\,/r_j, perm/\,L'_{\max}$ and $Fm\,/r_j, perm/\,L'_{\max}$, respectively. These algorithms used a branching scheme that exploited certain dominance properties of a critical path. Tadei et al. [41] tried this type of branching scheme for the problem $F2\,/r_j/\,C_{\max}$, but found it to be less effective than their traditional n-ary branching scheme, that fixes jobs only at the beginning of the schedule. Potts [34] considered an adaptive branching scheme that fixes jobs at both ends of the schedule, and found this branching scheme to be quite effective for the problem $Fm//C_{\max}$. Dominance rules are another common feature of branch and bound algorithms, they are typically used to eliminate nodes (before their bounds are calculated) in order to reduce computation time and storage requirements. Cheng et al. [4] apply approximate dominance rules, in the context of fuzzy inference, for scheduling and searching in flow-shop problems such as $F3//C_{\max}$ and $Fm\,/perm/\,C_{\max}$. The authors were able to find a nearly optimal initial schedule by repeatedly applying the fuzzy dominance rule to schedule the jobs. This fuzzy schedule also proved very useful for tie-breaking purposes, to decide which way to branch among several nodes with the same lower bound.

In this Chapter, we consider a new branch and bound algorithm for solving the problem $F2\,/r_j, perm/\,L'_{\max}$, with the following main features. We use an adaptive n-ary branching rule, a variant of the one used by Potts. We derive a new dominance order on the job set, using the proof technique of subset-restricted interchange that employs a new interchange operator which we introduce. In addition, we make use of fuzzy dominance properties for initial scheduling and tie-breaking. We also incorporate a simple decomposition procedure that reduces the problem size by fixing jobs at the beginning of the schedule. We use six very quickly computable lower bounds at each node of the tree. The algorithm represents an effective tool for solving large instances of the strongly NP-hard problem $F2\,/r_j, perm/\,L'_{\max}$. In a large scale computational experiment, the algorithm has solved in a matter of a few seconds 4,384 of the 4,500 randomly generated test problems with up to 200 jobs. Even for the unsolved problems, the best solution found by the algorithm was on average within less than 0.5% of the optimal value.

The rest of the Chapter is organized as follows. In the next section, we derive several new dominance results for the problem $F2\,/r_j, perm/\,L'_{\max}$. In section 3, we present the details of our branch and bound algorithm, a pseudocode for it is found in Appendix B. In section 4, we discuss the results of a large scale computational experiment. In the last section, we summarize our results.

4.2 Dominance results

In this section, we present dominance results for the problem $F2/r_i, perm/L'_{\max}$. First we derive a new dominance order $\prec$, using ***subset-restricted interchange*** for a new interchange operator which we introduce. Secondly, we consider a fuzzy approximation of dominance that is used in both initial scheduling and searching.

4.2.1 Subset-restricted interchange

Recall that we follow Monma [30] in defining our interchange operators. Let s_1 be a sequence with job m preceding job k. In general, s_1 is of the form $s_1 = (XmYkZ)$, where X, Y and Z are subsequences of J, and let U and V be disjoint subsequences of Y. Three types of interchanges of jobs k and m that leave k preceding m in the resulting sequence s_2, are the following:

1. *Backward Insertion*(BI)	$s_2 = (XYkmZ)$
2. *Forward Insertion*(FI)	$s_2 = (XkmYZ)$
3. *Shuffle Interchange(SI)*	$s_2 = (XUkmVZ)$.

To define more precisely the new SI operator, consider again sequence $s_1 = (XmYkZ)$ and a partition of Y into $Y = U \cup V$, where $U = (u_1u_2u_3)$ and $V = (v_1v_2v_3v_4)$, and Y is of the form $Y = (u_1u_2v_1v_2u_3v_3v_4)$. Then SI applied to sequence s_1, for this particular choice of U and V, gives sequence $s_2 = (Xu_1u_2u_3kmv_1v_2v_3v_4Z)$. We call it shuffle interchange because the interchange of jobs has the resulting net effect of 'shuffling' sequence Y into subsequences U and V, and placing jobs k and m between them. Notice that SI may change the relative order of some u's and v's after interchange, but it never changes the relative order of two u's or two v's. Thus SI depends on the choice of subsequences U and V. SI generalizes BI and FI: If we let V and U be the sets of jobs in sequences V and U (for the sake of brevity, we do not distinguish between sets and sequences) when $V = \varnothing$ or $U = \varnothing$, then SI reduces to BI or FI, respectively. Further, these interchanges all reduce to adjacent pairwise interchange in the case when $Y = \varnothing$.

Recall that the problem $F2//C_{\max}$ is solved by the *Johnson order*: first order the jobs with $a_j \leq b_j$ in *nondecreasing* a order followed by the jobs with $a_j > b_j$ in *nonincreasing* b order [21]. To emphasize the partitioning of the jobs in the Johnson order we define $J_{a\le b} = \{j \,|a_j \le b_j\}$ and $J_{a>b} = \{j \,|a_j > b_j\}$. This ordering of the jobs is an adjacent interchange order for the problem $F2//C_{\max}$, and since it also completely orders every pair of jobs, it is an optimal ordering (i.e., an optimal sequence). An adjacent interchange order for the problem $F2\,/r_j, perm/\, L'_{\max}$ is the *intersection* of the nondecreasing r order, the nonincreasing q order and the Johnson order, as defined formally below. Note that this order is no longer a complete order, rather it is only a partial order.

Definition 4.1 *Adjacent Interchange Order* $\lessdot$*:* $k \lessdot m$ *if* $r_k \le r_m$*,* $q_k \ge q_m$ *and*

(i) $k,\, m \in J_{a\le b}$ *and* $a_k \le a_m$ *or,*
(ii) $k \in J_{a\le b}$ *and* $m \in J_{a>b}$ *or,*
(iii) $k,\, m \in J_{a>b}$ *and* $b_k \ge b_m$.

In general, if an adjacent interchange order is only a partial order (and not a complete order), it need not be a dominance order. This is the case for $\lessdot$ defined above, if we consider the instance of $F2\,/r_j, perm/\, L'_{\max}$ in Example 4. Here we have $3 \lessdot 1$ (since $r_3 = r_1 = 10$, $q_3 = 10 \ge 0 = q_1$, and $3, 1 \in J_{a>b}$ with $b_3 = 25 > 15 = b_1$), however, the unique optimal sequence is $(1, 2, 3, 4)$ with $L'_{\max} = 125$. Thus we see that $\lessdot$ is not a dominance order since there is no optimal sequence that is a linear extension of $\lessdot$. We will use subset-restricted interchange to find a suborder of $\lessdot$ that is a dominance order.

Example 4 *A 4 job problem to illustrate that* $\lessdot$ *is not necessarily a dominance order.*

j	1	2	3	4
r_j	10	20	10	30
a_j	20	20	30	25
b_j	15	30	25	20
q_j	0	10	10	0

Recall, that the SI operator above interchanges k and m 'around' sequence Y, for some particular choice of U and V. Intuitively, whether or not such an interchange leads to a reduction in cost (for a given sequencing function f and adjacent interchange order $\lessdot$), should depend on the composition of Y, as well as on the choice of U and

V. We consider interchanges that are restricted by conditions on Y and define the subset-restricted interchange condition for SI.

Definition 4.2 *An adjacent interchange order $\lessdot$ together with the collection of subsets $\{U_{k\lessdot m} \cup V_{k\lessdot m} \,|k\lessdot m\}$ satisfies the Shuffle Interchange Condition for a sequencing function f if for all jobs k, m and sequences X, Y, Z there exist a partition U and V into disjoint subsequences of Y such that $k\lessdot m$, $U \subseteq U_{k\lessdot m}$, and $V \subseteq V_{k\lessdot m}$ imply that $f\,(XUkmVZ) \le f\,(XmYkZ)$.*

Next we examine the structure of $U_{k\lessdot m}$ and $V_{k\lessdot m}$ for the different types of pairs $k\lessdot m$ for the problem $F2\,/r_j, perm/\,L'_{\max}$.

Theorem 4.1 *If $k \in J_{a\le b}$ and $a_k \le a_m$, then $k\lessdot m$ together with the sets $U_{k\lessdot m} = \{u\,|u \in J_{a\le b}, r_u \le r_m, a_u \le a_m\}$ and $V_{k\lessdot m} = \{v\,|q_v \le q_k\}$ satisfy the Shuffle Interchange Condition.*

Proof. Given a sequence s we construct the directed graph $G(s)$ to evaluate $L'_{\max}(s)$. Each job $s(j)$ is represented by four nodes with weights $r_{s(j)}$, $a_{s(j)}$, $b_{s(j)}$, and $q_{s(j)}$, respectively. $L'_{\max}(s)$ is the length of the longest 'node-weighted' path from the start node to the finish node. These paths can be uniquely identified by the triples $(s(i), s(j), s(k))$, for $1 \le i \le j \le k \le n$, representing the end points of the three horizontal segments on the path (see Figure 4.1). By definition,

$$L'_{\max}(s) = \max_{\substack{(s(i),s(j),s(k))\\ 1\le i\le j\le k\le n}} \left(r_{s(i)} + \sum_{l=i}^{j} a_{s(l)} + \sum_{l=j}^{k} b_{s(l)} + q_{s(k)} \right),$$

and we can evaluate $L'_{\max}(s)$ as the maximum over all such paths in $G\,(s)$.

Let $s_1 = (XmYkZ)$ be a sequence for pair $k\lessdot m$ with $k \in J_{a\le b}$ and $a_k \le a_m$. Let $U \subseteq U_{k\lessdot m}$ and $V \subseteq V_{k\lessdot m}$ be disjoint subsequences of Y, with $Y = U \cup V$. We apply shuffle interchange to s_1, with this U and V, and obtain sequence $s_2 = (XUkmVZ)$. (We use lower case letters x, u, v, and z to refer to arbitrary generic elements of subsequences X, U, V, or Z, respectively.) We can demonstrate that s_2 is not worse than s_1, by exhibiting for every path in s_2 a dominating path in s_1 (see Figure 4.2). For example, consider the path (u, k, v) in s_2, then the corresponding

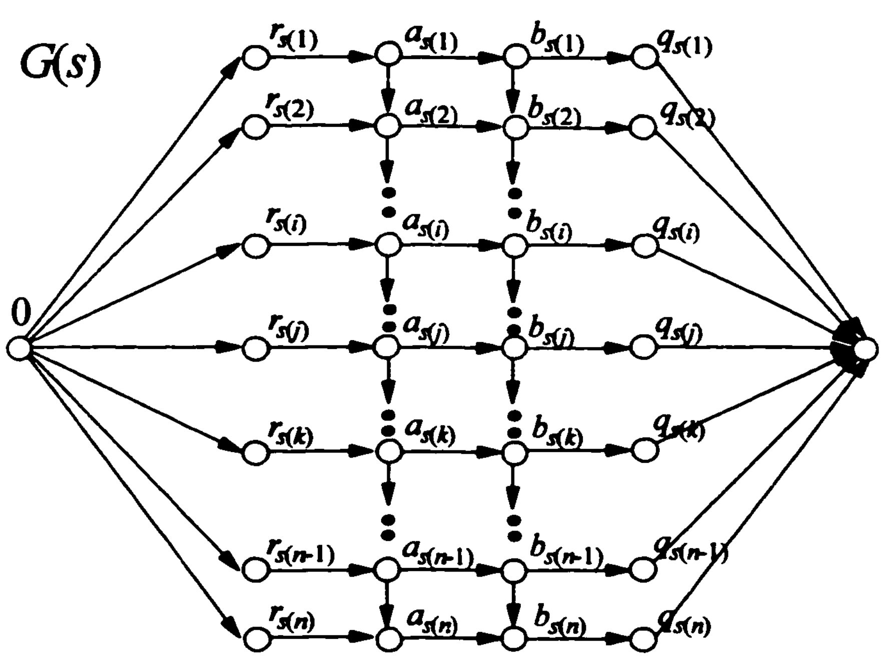

Figure 4.1: Directed graph $G(s)$ for problem $F2\,/r_j, perm/\,L'_{\max}$.

dominating path in s_1 is (m, m, k), which can be proved as follows. (Note that we use $U_{[u,\cdot]}$ to represent the subsequence of U starting with u and ending in the last job in U; and $V_{[\cdot,v]}$ to represent the subsequence of V from its beginning to job v.)

$$r_u + \sum_{i\in U_{[u,\cdot]}} a_i + a_k + b_k + b_m + \sum_{i\in V_{[\cdot,v]}} b_i + q_v \le r_m + a_m + b_m + \sum_{i\in U} b_i + \sum_{i\in V} b_i + b_k + q_k,$$

since $r_u \le r_m$, $U \subseteq J_{a\le b}$, $a_k \le a_m$, and $q_k \ge q_v$.

To complete the proof, we present Table 4.1 which gives the dominating paths in s_1 for each path in s_2. The last column contains the argument why each path is a dominating path. ■

Theorem 4.2 *If $k \in J_{a\le b}$ and $m \in J_{a>b}$, then $k{<}m$ together with the sets $U_{k<m} = \{u\,|u \in J_{a\le b}, r_u \le r_m\}$ and $V_{k<m} = \{v\,|v \in J_{a>b}, q_v \le q_k\}$ satisfies the Shuffle Interchange Condition.*

Proof. Similarly, Table 4.2 contains the appropriate dominating paths. ■

Table 4.1: Dominating paths for Theorem 4.1.

s_2	s_1	
$(XUkmVZ)$	$(XmYkZ)$	*proof*
(x_1, x_2, x_3)	(x_1, x_2, x_3)	
(x_1, x_2, u)	(x_1, x_2, u)	
(x_1, x_2, k)	(x_1, x_2, k)	
(x_1, x_2, m)	(x_1, x_2, k)	$q_k \geq q_m$
(x_1, x_2, v)	(x_1, x_2, k)	$q_k \geq q_v$
(x_1, x_2, z)	(x_1, x_2, z)	
(x, u_1, u_2)	(x, u_1, u_2)	
(x, u, k)	(x, u, k)	
(x, u, m)	(x, m, k)	$U \subseteq J_{a \leq b}, a_u \leq a_m, q_k \geq q_m$
(x, u, v)	(x, m, k)	$U \subseteq J_{a \leq b}, a_u \leq a_m, q_k \geq q_v$
(x, u, z)	(x, m, z)	$U \subseteq J_{a \leq b}, a_u \leq a_m$
(x, k, k)	(x, k, k)	
(x, k, m)	(x, m, k)	$U \subseteq J_{a \leq b}, k \in J_{a \leq b}, q_k \geq q_m$
(x, k, v)	(x, m, k)	$U \subseteq J_{a \leq b}, a_k \leq a_m, q_k \geq q_v$
(x, k, z)	(x, m, z)	$U \subseteq J_{a \leq b}, a_k \leq a_m,$
(x, m, m)	(x, m, k)	$U \subseteq J_{a \leq b}, k \in J_{a \leq b}, q_k \geq q_m$
(x, m, v)	(x, m, k)	$U \subseteq J_{a \leq b}, k \in J_{a \leq b}, q_k \geq q_v$
(x, m, z)	(x, m, z)	$U \subseteq J_{a \leq b}, k \in J_{a \leq b}$
(x, v_1, v_2)	(x, v_1, k)	$U \subseteq J_{a \leq b}, k \in J_{a \leq b}, q_k \geq q_{v_2}$
(x, v, z)	(x, v, z)	$U \subseteq J_{a \leq b}, k \in J_{a \leq b}$
(x, z_1, z_2)	(x, z_1, z_2)	
(u_1, u_2, u_3)	(u_1, u_2, u_3)	
(u_1, u_2, k)	(u_1, u_2, k)	
(u_1, u_2, m)	(m, m, k)	$r_{u_1} \leq r_m, U \subseteq J_{a \leq b}, a_{u_2} \leq a_m, q_k \geq q_m$
(u_1, u_2, v)	(m, m, k)	$r_{u_1} \leq r_m, U \subseteq J_{a \leq b}, a_{u_2} \leq a_m, q_k \geq q_v$
(u_1, u_2, z)	(m, m, z)	$r_{u_1} \leq r_m, U \subseteq J_{a \leq b}, a_{u_2} \leq a_m$
(u, k, k)	(m, k, k)	$r_u \leq r_m$
(u, k, m)	(m, m, k)	$r_u \leq r_m, U \subseteq J_{a \leq b}, a_k \leq a_m, q_k \geq q_m$

Table 4.1: (continued).

s_2	s_1	
$(XUkmVZ)$	$(XmYkZ)$	*proof*
(u,k,v)	(m,m,k)	$r_u \le r_m, U \subseteq J_{a\le b}, a_k \le a_m, q_k \ge q_v$
(u,k,z)	(m,m,z)	$r_u \le r_m, U \subseteq J_{a\le b}, a_k \le a_m$
(u,m,m)	(m,m,k)	$r_u \le r_m, U \subseteq J_{a\le b}, k \in J_{a\le b}, q_k \ge q_m$
(u,m,v)	(m,m,k)	$r_u \le r_m, U \subseteq J_{a\le b}, k \in J_{a\le b}, q_k \ge q_v$
(u,m,z)	(m,m,z)	$r_u \le r_m, U \subseteq J_{a\le b}, k \in J_{a\le b}$
(u,v_1,v_2)	(m,v_1,k)	$r_u \le r_m, U \subseteq J_{a\le b}, k \in J_{a\le b}, q_k \ge q_{v_2}$
(u,v,z)	(m,v,z)	$r_u \le r_m, U \subseteq J_{a\le b}, k \in J_{a\le b}$
(u,z_1,z_2)	(m,z_1,z_2)	$r_u \le r_m$
(k,k,k)	(k,k,k)	
(k,k,m)	(m,m,k)	$r_k \le r_m, a_k \le a_m, q_k \ge q_m$
(k,k,v)	(m,m,k)	$r_k \le r_m, a_k \le a_m, q_k \ge q_v$
(k,k,z)	(m,m,z)	$r_k \le r_m, a_k \le a_m$
(k,m,m)	(m,m,k)	$r_k \le r_m, k \in J_{a\le b}, q_k \ge q_m$
(k,m,v)	(m,m,k)	$r_k \le r_m, k \in J_{a\le b}, q_k \ge q_v$
(k,m,z)	(m,m,z)	$r_k \le r_m, k \in J_{a\le b}$
(k,v_1,v_2)	(m,v_1,k)	$r_k \le r_m, k \in J_{a\le b}, q_k \ge q_{v_2}$
(k,v,z)	(m,v,z)	$r_k \le r_m, k \in J_{a\le b}$
(k,z_1,z_2)	(m,z_1,z_2)	$r_k \le r_m$
(m,m,m)	(m,m,m)	
(m,m,v)	(m,m,v)	
(m,m,z)	(m,m,z)	
(m,v_1,v_2)	(m,v_1,v_2)	
(m,v,z)	(m,v,z)	
(m,z_1,z_2)	(m,z_1,z_2)	
(v_1,v_2,v_3)	(v_1,v_2,v_3)	
(v_1,v_2,z)	(v_1,v_2,z)	
(v,z_1,z_2)	(v,z_1,z_2)	
(z_1,z_2,z_3)	(z_1,z_2,z_3)	

Table 4.2: Dominating paths for Theorem 4.2.

s_2	s_1	
$(XUkmVZ)$	$(XmYkZ)$	*proof*
(x_1, x_2, x_3)	(x_1, x_2, x_3)	
(x_1, x_2, u)	(x_1, x_2, u)	
(x_1, x_2, k)	(x_1, x_2, k)	
(x_1, x_2, m)	(x_1, x_2, k)	$q_k \geq q_m$
(x_1, x_2, v)	(x_1, x_2, k)	$q_k \geq q_v$
(x_1, x_2, z)	(x_1, x_2, z)	
(x, u_1, u_2)	(x, u_1, u_2)	
(x, u, k)	(x, u, k)	
(x, u, m)	(x, u, k)	$m \in J_{a>b}, q_k \geq q_m$
(x, u, v)	(x, u, k)	$m \in J_{a>b}, V \subseteq J_{a>b}, q_k \geq q_v$
(x, u, z)	(x, u, z)	$m \in J_{a>b}, V \subseteq J_{a>b}$
(x, k, k)	(x, k, k)	
(x, k, m)	(x, k, k)	$m \in J_{a>b}, V \subseteq J_{a>b}, q_k \geq q_m$
(x, k, v)	(x, k, k)	$m \in J_{a>b}, V \subseteq J_{a>b}, q_k \geq q_v$
(x, k, z)	(x, k, z)	$m \in J_{a>b}, V \subseteq J_{a>b}$
(x, m, m)	(x, m, k)	$U \subseteq J_{a\leq b}, k \in J_{a\leq b}, q_k \geq q_m$
(x, m, v)	(x, m, k)	$U \subseteq J_{a\leq b}, k \in J_{a\leq b}, q_k \geq q_v$
(x, m, z)	(x, m, z)	$U \subseteq J_{a\leq b}, k \in J_{a\leq b}$
(x, v_1, v_2)	(x, v_1, k)	$U \subseteq J_{a\leq b}, k \in J_{a\leq b}, q_k \geq q_{v_2}$
(x, v, z)	(x, v, z)	$U \subseteq J_{a\leq b}, k \in J_{a\leq b}$
(x, z_1, z_2)	(x, z_1, z_2)	
(u_1, u_2, u_3)	(u_1, u_2, u_3)	
(u_1, u_2, k)	(u_1, u_2, k)	
(u_1, u_2, m)	(m, u_2, k)	$r_{u_1} \leq r_m, m \in J_{a>b}, q_k \geq q_m$
(u_1, u_2, v)	(m, u_2, k)	$r_{u_1} \leq r_m, m \in J_{a>b}, V \subseteq J_{a>b}, q_k \geq q_v$
(u_1, u_2, z)	(m, u_2, z)	$r_{u_1} \leq r_m, m \in J_{a>b}, V \subseteq J_{a>b}$
(u, k, k)	(u, k, k)	
(u, k, m)	(m, k, k)	$r_u \leq r_m, m \in J_{a>b}, q_k \geq q_m$

Table 4.2: (continued).

s_2	s_1	
$(XUkmVZ)$	$(XmYkZ)$	*proof*
(u,k,v)	(m,k,k)	$r_u \leq r_m, m \in J_{a>b}, V \subseteq J_{a>b}, q_k \geq q_v$
(u,k,z)	(m,k,z)	$r_u \leq r_m, m \in J_{a>b}, V \subseteq J_{a>b}$
(u,m,m)	(m,m,k)	$r_u \leq r_m, U \subseteq J_{a\leq b}, k \in J_{a\leq b}, q_k \geq q_m$
(u,m,v)	(m,m,k)	$r_u \leq r_m, U \subseteq J_{a\leq b}, k \in J_{a\leq b}, q_k \geq q_v$
(u,m,z)	(m,m,z)	$r_u \leq r_m, U \subseteq J_{a\leq b}, k \in J_{a\leq b}$
(u,v_1,v_2)	(m,v_1,k)	$r_u \leq r_m, U \subseteq J_{a\leq b}, k \in J_{a\leq b}, q_k \geq q_{v_2}$
(u,v,z)	(m,v,z)	$r_u \leq r_m, U \subseteq J_{a\leq b}, k \in J_{a\leq b}$
(u,z_1,z_2)	(m,z_1,z_2)	$r_u \leq r_m$
(k,k,k)	(k,k,k)	
(k,k,m)	(m,k,k)	$r_k \leq r_m, m \in J_{a>b}, q_k \geq q_m$
(k,k,v)	(m,k,k)	$r_k \leq r_m, m \in J_{a>b}, V \subseteq J_{a>b}, q_k \geq q_v$
(k,k,z)	(m,k,z)	$r_k \leq r_m, m \in J_{a>b}, V \subseteq J_{a>b}$
(k,m,m)	(m,m,k)	$r_k \leq r_m, k \in J_{a\leq b}, q_k \geq q_m$
(k,m,v)	(m,m,k)	$r_k \leq r_m, k \in J_{a\leq b}, q_k \geq q_v$
(k,m,z)	(m,m,z)	$r_k \leq r_m, k \in J_{a\leq b}$
(k,v_1,v_2)	(m,v_1,k)	$r_k \leq r_m, k \in J_{a\leq b}, q_k \geq q_{v_2}$
(k,v,z)	(m,v,z)	$r_k \leq r_m, k \in J_{a\leq b}$
(k,z_1,z_2)	(m,z_1,z_2)	$r_k \leq r_m$
(m,m,m)	(m,m,m)	
(m,m,v)	(m,m,v)	
(m,m,z)	(m,m,z)	
(m,v_1,v_2)	(m,v_1,v_2)	
(m,v,z)	(m,v,z)	
(m,z_1,z_2)	(m,z_1,z_2)	
(v_1,v_2,v_3)	(v_1,v_2,v_3)	
(v_1,v_2,z)	(v_1,v_2,z)	
(v,z_1,z_2)	(v,z_1,z_2)	
(z_1,z_2,z_3)	(z_1,z_2,z_3)	

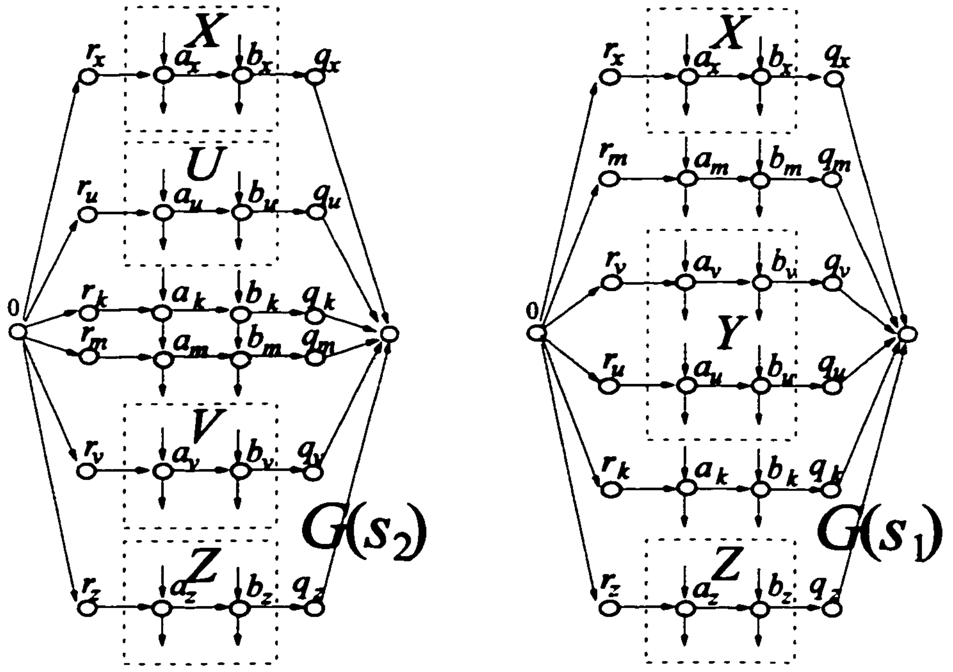

Figure 4.2: Directed graphs $G(s_2)$ and $G(s_1)$ for SI.

Theorem 4.3 *If $m \in J_{a>b}$ and $b_k \geq b_m$, then $k{<}m$ together with the sets $U_{k<m} = \{u \,|\, r_u \leq r_m\}$ and $V_{k<m} = \{v \,|\, v \in J_{a>b}, q_v \leq q_k, b_v \leq b_k\}$ satisfies the Shuffle Interchange Condition.*

Proof. Symmetric to Theorem 4.1. ■

4.2.2 New dominance order for $F2/r_j, perm/L'_{\max}$

In this section, we use subset-restricted interchange to derive a new dominance order on the jobs, denoted by $\prec$, for the problem $F2\,/r_i, perm/\,L'_{\max}$. We define the following sets of jobs for every pair $k{<}m$. (Note that a pair $k{<}m$ may satisfy more than one of the following three conditions, so that more than one may be applicable to a pair.)

1. If $a_k \leq b_k$ and $a_k \leq a_m$, then let $U^1_{k \lessdot m} = \{u \,|\, u \in J_{a \leq b}, r_u \leq r_m, a_u \leq a_m, q_u > q_k\}$
   $V^1_{k \lessdot m} = \{v \,|\, q_v \leq q_k\}$.
2. If $a_k \leq b_k$ and $a_m > b_m$, then let $U^2_{k \lessdot m} = \{u \,|\, u \in J_{a \leq b}, r_u \leq r_m\}$
   $V^2_{k \lessdot m} = \{v \,|\, v \in J_{a > b}, q_v \leq q_k\}$.
3. If $a_m > b_m$ and $b_k \geq b_m$, then let $U^3_{k \lessdot m} = \{u \,|\, r_u \leq r_m\}$
   $V^3_{k \lessdot m} = \{v \,|\, v \in J_{a > b}, q_v \leq q_k, b_v \leq b_k, r_v > r_m\}$.

These are just the sets $U_{k \lessdot m}$ and $V_{k \lessdot m}$ from the previous three Theorems with additional conditions in cases 1 and 3, to ensure that $U^i_{k \lessdot m} \cap V^i_{k \lessdot m} = \varnothing$ and maintain that $v_i \not\lessdot u_i$ for $u_i \in U^i_{k \lessdot m}$ and $v_i \in V^i_{k \lessdot m}$. In case 1, $q_u > q_k$ has been added, and $r_v > r_m$ has been added in case 3. For a given pair $k \lessdot m$, then $k \lessdot m$ is included in the dominance order $\prec$ exactly when any of the applicable sets $U^i_{k \lessdot m} \cup V^i_{k \lessdot m}$ is the *entire* job set. Thus, the dominance order $\prec$ is defined as the suborder of $\lessdot$ consisting of pairs $k \lessdot m$, that can be interchanged around *any* intermediate sequence using the SI operator for the appropriate choice of U and V.

Definition 4.3 *If $k \lessdot m$ and $U^i_{k \lessdot m} \cup V^i_{k \lessdot m} = J$ for some i applicable to k and m, then $k \prec m$.*

To illustrate the definition of $\prec$, consider again Example 4. The adjacent interchange order $\lessdot$ consists of the pairs $2 \lessdot 4$, $3 \lessdot 4$, and (as we saw previously) $3 \lessdot 1$. For pair $2 \lessdot 4$ both Theorems 4.1 and 4.2 apply, and checking the conditions for these we see that both $U^1_{2 \lessdot 4} \cup V^1_{2 \lessdot 4}$ and $U^2_{2 \lessdot 4} \cup V^2_{2 \lessdot 4}$ are equal to J, thus we have that $2 \prec 4$. For pairs $3 \lessdot 4$ and $3 \lessdot 1$ only Theorem 4.3 applies. We also have $U^3_{3 \lessdot 4} \cup V^3_{3 \lessdot 4} = J$, thus $3 \prec 4$. However, $U^3_{3 \lessdot 1} \cup V^3_{3 \lessdot 1} \neq J$ because job $2 \notin V^3_{3 \lessdot 1}$ since $2 \in J_{a \leq b}$, and $2 \notin U^3_{3 \lessdot 1}$ since $r_2 > r_1$, thus $3 \not\prec 1$. Therefore, we have that the dominance order $\prec$ is the suborder of $\lessdot$consisting of the pairs $2 \prec 4$ and $3 \prec 4$, and we see that the unique optimal sequence $(1, 2, 3, 4)$ is indeed a linear extension of $\prec$, as we would expect.

We define the following suborders of $\prec$ for the different types of pairs in $\lessdot$

$$\begin{aligned}\prec_1 &= \prec \bigcap (J_{a\le b} \times J_{a\le b})\\ \prec_2 &= \prec \bigcap (J_{a\le b} \times J_{a>b})\\ \prec_3 &= \prec \bigcap (J_{a>b} \times J_{a>b}),\end{aligned}$$

where $A \times B$ represents all pairs with first element from set A and second element from set B. These suborders will be used first to demonstrate that $\prec$ is indeed a partial order, and then to prove that $\prec$ is a dominance order for problem $F2\,/r_i, perm/\,L'_{\max}$.

Lemma 4.4 *Suborder $\prec$ is a partial order.*

Proof. Since $\prec$ is a suborder of $\lessdot$, it only remains to show that $\prec$ is transitive, i.e., we want to show that for any jobs k, l, and m if $k \prec l$ and $l \prec m$, then $k \prec m$. First we consider the case where all of the jobs k, l, and m are in $J_{a\le b}$. In this case, we have that $k \prec_1 l$ and $l \prec_1 m$, and we want to demonstrate that $k \prec_1 m$. By Definition 4.3 $k \prec_1 m$ if $U^1_{k\lessdot m} \cup V^1_{k\lessdot m} = J$. Since $k\lessdot l\lessdot m$ and k, l, and m are in $J_{a\le b}$, we have that $r_k \le r_l \le r_m$ and $a_k \le a_l \le a_m$. These imply that $U^1_{k\lessdot m} \supseteq U^1_{k\lessdot l}$, and since $V^1_{k\lessdot m} = V^1_{k\lessdot l}$, we have that $U^1_{k\lessdot m} \cup V^1_{k\lessdot m}$ contains $U^1_{k\lessdot l} \cup V^1_{k\lessdot l}$. However, $k \prec_1 l$ implies that $U^1_{k\lessdot l} \cup V^1_{k\lessdot l} = J$, which by the above observation and Definition 4.3 implies that $k \prec_1 m$. Next we examine the case where $k, l \in J_{a\le b}$ and $m \in J_{a>b}$, in which $k \prec_1 l$ and $l \prec_2 m$. We consider separately the different cases of Definition 4.3 that may apply for pair $l \prec_2 m$. If case 1 applies for $l \prec_2 m$, then $(a_k \le)\, a_l \le a_m$ and the previous argument carries over exactly as above. For case 2, we assume that $U^2_{l\lessdot m} \cup V^2_{l\lessdot m} = J$. Since $k\lessdot l\lessdot m$, we have that $r_k \le r_l \le r_m$ and $q_k \ge q_l \ge q_m$. These imply, however, that $U^2_{k\lessdot m} = U^2_{l\lessdot m}$and $V^2_{k\lessdot m} \supseteq V^2_{l\lessdot m}$, so we also have that $U^2_{k\lessdot m} \cup V^2_{k\lessdot m} = J$. Thus, by Definition 4.3 we have that $k \prec_2 m$. Finally, if case 3 applies to $l \prec_2 m$, then $b_l \ge b_m$ and $U^3_{l\lessdot m} \cup V^3_{l\lessdot m} = J$. We notice that $(U^3_{l\lessdot m} \cup V^3_{l\lessdot m}) \cap J_{a\le b}$ equals $U^3_{l\lessdot m} \cap J_{a\le b}$, which equals $U^2_{k\lessdot m}$. Since $V^3_{l\lessdot m} \cap J_{a\le b} = \varnothing$ and $U^3_{l\lessdot m} \cup V^3_{l\lessdot m} = J$, this implies that $U^2_{k\lessdot m}$ contains all of the jobs in $J_{a\le b}$. We also have that $U^1_{k\lessdot l} \cup V^1_{k\lessdot l} = J$ since $k \prec_1 l$. Similarly, $(U^1_{k\lessdot l} \cup V^1_{k\lessdot l}) \cap J_{a>b}$ equals $V^1_{k\lessdot l} \cap J_{a>b}$, which equals $V^2_{k\lessdot m}$. Since $U^1_{k\lessdot l} \cap J_{a>b} = \varnothing$ and $U^1_{k\lessdot l} \cup V^1_{k\lessdot l} = J$, this implies that $V^2_{k\lessdot m}$ contains all of the jobs in $J_{a>b}$. Combining these two results we have that $U^2_{k<m} \cup V^2_{k<m}$ contains all of the jobs in J, from which it follows by Definition 4.3 that $k \prec_2 m$. The remaining cases, where k, l, and m are in $J_{a>b}$ $k \prec_3 l$ and $l \prec_3 m$, or $k \in J_{a\leq b}$ and $l, m \in J_{a>b}$ with $k \prec_2 l$ and $l \prec_3 m$, follow by symmetry. Thus we see that $\prec$ is a transitive order indeed. ■

Theorem 4.5 *Partial order $\prec$ is a dominance order for problem $F2\,/r_i, perm/\,L'_{\max}$.*

Proof. Let s be an optimal sequence. We find an optimal linear extension of $\prec$ by repeatedly applying the SI operator to sequence s without increasing the length of the longest path. We demonstrate the comparabilities in the order $\prec_3$, $\prec_1$ and $\prec_2$, respectively.

Assume that s is not a linear extension of $\prec_3$. Let $m \in J_{a>b}$ be the last job in s with some job $j \in J_{a>b}$ with $j \prec_3 m$, such that j is after m in s. Let k be the first such job j in s, then s is of the form $s = (XmYkZ)$. Since $k \prec m$, k and m can be interchanged around any intermediate sequence using SI, in particular around subsequence Y. Since $k \prec_3 m$, $U_{k<m} \cup V_{k<m} = U^3_{k<m} \cup V^3_{k<m} = J$. Applying SI to s with $U = U^3_{k<m} \cap Y$ and $V = V^3_{k<m} \cap Y$, we obtain sequence $(XUkmVZ)$. This sequence now orders k and m properly, and it does not introduce any new violations of $\prec_3$: To see this, consider any jobs $u \in U$ and $v \in V$. Since $u \in U^3_{k<m}$ and $v \in V^3_{k<m}$, $v \not\lessdot u$, so $v \not\prec u$, which implies that $v \not\prec_3 u$. Also, $m \not\prec_3 u$ follows by the choice of m, and $v \not\prec_3 k$ follows since $r_v > r_m \geq r_k$. By repeatedly applying the above procedure until there is no such job m, we obtain an optimal sequence s' which is a linear extension of $\prec_3$.

For $\prec_1$, we proceed in exactly the same way. Assume that sequence s' is not a linear extension of $\prec_1$. Let $m \in J_{a\leq b}$ be the last job in s' with some job $j \in J_{a\leq b}$ with $j \prec_1 m$, such that j is after m in s. Let k be the first such job j in s', then s' is of the form $s' = (XmYkZ)$. Applying SI to s' with $U = U^1_{k<m} \cap Y$ and $V = V^1_{k<m} \cap Y$, we obtain sequence $(XUkmVZ)$. This sequence now orders k and m properly, and it does not introduce any new violations of $\prec_1$ or $\prec_3$: For $\prec_1$, once again consider any jobs $u \in U$ and $v \in V$. Since, $v \not\lessdot u$, then $v \not\prec_1 u$. Also $m \not\prec_1 u$ since $q_u > q_k \geq q_m$, and $v \not\prec_1 k$ follows by the choice of k. To see that this doesn't introduce any new violations of $\prec_3$ either, note that all jobs in $Y \cap J_{a>b}$ must be in V, and for these jobs their relative order is unchanged. Repeatedly applying SI we obtain an optimal sequence s'' that is a linear extension of $\prec_1$ and $\prec_3$.

Finally, assume that sequence s'' is not a linear extension of $\prec_2$. Let $m \in J_{a>b}$ be the last job in s with some job $j \in J_{a\leq b}$ with $j \prec_2 m$, such that j is after m in s. Let k be the first such job j in s'', then s'' is of the form $s'' = (XmYkZ)$. Since $k \prec_2 m$, we know for some i that $J = U^i_{k<m} \cup V^i_{k<m}$, applying SI for this choice of i with $U = U^i_{k<m} \cap Y$ and $V = V^i_{k<m} \cap Y$, we obtain sequence $(XUkmVZ)$. This sequence now orders k and m properly, and it does not introduce any new violations of $\prec$: As above, for any jobs $u \in U$ and $v \in V$ we have that $v \not\prec u$ which implies that $v \not\prec u$. In addition, we have $m \not\prec_3 u$ ($\Rightarrow m \not\prec u$) and $v \not\prec_1 k$ ($\Rightarrow v \not\prec k$) as above. Thus, we see that this interchange does not introduce any new violations of $\prec$. Continuing to apply SI until there is no such job m, we obtain an optimal sequence s^* that is also a linear extension of $\prec$. ■

4.2.3 Fuzzy approximation of dominance

Dominance rules on sequences are usually used to specify whether a node can be eliminated before its lower bound is calculated in a branch and bound algorithm. However, they also can be used heuristically in finding a good initial solution, or directing the search in case of ties [4].

Theorem 4.6 *Consider a partial sequence σ for the problem $F2/r_j/L'_{\max}$. If there are unsequenced jobs i and j such that*

$$C^l(\sigma ij) \leq C^l(\sigma ji), 1 \leq l \leq 2 \tag{1}$$

$$L'_{\max}(\sigma ij) \leq L'_{\max}(\sigma ji), \tag{2}$$

then the partial sequence σij dominates σji, i.e., any completion of σij has an $L'_{\max}$ which is not larger than the $L'_{\max}$ for the same completion of σji.

Proof. Consider an arbitrary completion sequence w for the remaining unsequenced jobs. If Eq.(1) holds, then every job in w will be able to start no later on both machines, in the sequence (σijw) than in the sequence (σjiw). This implies that no job in w can have a larger L' value in the sequence (σijw) than in the sequence (σjiw). Combining this with Eq.(2) completes the proof. ■

If there exists a job i for which Eq.(1) and Eq.(2) hold for *all* unsequenced jobs j, then sequence σ has an optimal completion with job i sequenced next. Such a job i, very rarely exists, however. This suggests that if they only *approximately* hold, i.e., they hold for job i with *almost* all unsequenced jobs j, then job i may precede another job j in an optimal completion of sequence σ with high probability. We measure the closeness of this approximation by a fuzzy membership function. We use this fuzzy inference to find a good initial sequence, and to break ties between nodes with the same lower bound when branching. These techniques proved very useful for the problem $Fm//C_{\max}$ [4]. Let

$$\begin{aligned} D^l(\sigma ij) &= C^l(\sigma ij) - C^l(\sigma ji), \quad 1 \le l \le 2 \\ D^3(\sigma ij) &= L'_{\max}(\sigma ij) - L'_{\max}(\sigma ji)). \end{aligned}$$

The *fuzzy membership function* that represents the likelihood that job i precedes job j in an optimal completion of σ is given by

$$\mu_\sigma(i,j) = 0.5 - \frac{D(\sigma ij)}{2D_{max}(\sigma)},$$

where $D(\sigma ij) = \sum_{l=1}^{3} \alpha_l D^l(\sigma ij)$, $D_{max}(\sigma) = \max_{i,j} |D(\sigma ij)|$ and $\alpha_1, \alpha_2, \alpha_3$ $(0 \le \alpha_1, \alpha_2, \alpha_3 \le 1$ and $\sum_{l=1}^{3} \alpha_l = 1)$ are real numbers. (Note that this definition ensures that $0 \le \mu_\sigma(i,j) \le 1$.) Let S be the set of jobs not scheduled in σ. Then, the likelihood of job $i \in S$ dominating the remaining jobs after partial sequence σ is measured by

$$\mu^*_\sigma(i) = \min_{j \in S \setminus \{i\}} \mu_\sigma(i,j),$$

and job i^* satisfying

$$\mu^*_\sigma(i^*) = \max_{i \in S} \mu^*_\sigma(i)$$

is identified as the job that immediately follows σ.

The rule determining i^* in this way is referred to as the *fuzzy rule* and the schedule obtained by repeatedly applying the fuzzy rule is referred to as the *fuzzy schedule* (or *fuzzy sequence)*. To obtain our fuzzy sequence we apply the fuzzy rule with $\alpha_l = 1/3$ for $l = 1, 2, 3$. This requires $O(n^2)$ time.

4.3 Branch and bound

In this section, we outline the basic components of our proposed branch and bound algorithm to solve the problem $F2\,/r_j, perm/\,L'_{\max}$. A pseudocode for it is given in Appendix B.

4.3.1 Branching rule

In order to exploit the symmetry of the problem, we consider a variant of Potts' adaptive branching rule that fixes jobs at both ends of the schedule [34]. More precisely, each node of the search tree is represented by a pair (σ_1, σ_2), where σ_1 and σ_2 are the initial and final partial sequences, respectively. Let S_i denote the set of jobs in σ_i for $i = 1$, 2, then the set of unfixed jobs is $S = J\backslash(S_1 \cup S_2)$. We use $\prec|_S$ to refer to the restriction of $\prec$ to the set S. An immediate successor of (σ_1, σ_2) in the tree is either of the form $(\sigma_1 i, \sigma_2)$ for a *type 1* branching; or $(\sigma_1, i\sigma_2)$ for a *type 2* branching, where i is a minimal or maximal job in $\prec|_S$, respectively. The types of the branchings are all the same within a level of the tree. The type for a given level k is fixed on the very first visit to level k according to the following rule: branch in the direction of the fewest number of ties at the minimum lower bound. Let n_1 and n_2 be the number of ties at the minimum lower bound for potential type 1 and type 2 branchings, at level k. If $n_1 < n_2$ the next branching is of type 1, while if $n_2 < n_1$ then the branching is of type 2. If $n_1 = n_2$ then the branching is the same type as at the previous level.

The search strategy is to branch to the newest active node with the smallest lower bound, breaking ties by the appropriate fuzzy rule. For type 1 branchings we use fuzzy sequence 1 to break ties, this is the sequence obtained by repeatedly applying the fuzzy rule to the forward problem. For type 2 branchings we use fuzzy sequence 2 to break ties, this is the analogous sequence for the reverse problem.

4.3.2 Bounds

Upper bounds are calculated only for the first n nodes, afterward the upper bound is evaluated only at leaf nodes. For the first n nodes we calculate four separate upper bounds. The first two upper bounds are obtained by sequencing the unfixed jobs in the fuzzy sequence for the forward and reverse problems, respectively. Each of these requires only $O(n)$ time working from the previously established fuzzy sequences. The two remaining upper bounds are obtained by sequencing the unfixed jobs in ready $q + b$ order for the forward problem, and ready $r + a$ order for the reverse problem. Each of these upper bounds requires $O(n \log n)$ time to compute. For leaf nodes, there are no unfixed jobs and the upper bound is just the length of the sequence obtained by concatenating σ_1 and σ_2.

Since the branch and bound tree may require the computation of lower bounds for a potentially very large number of nodes, it is important that we use lower bounds which require only $O(n)$ time per bound. (We have also experimented with some potentially better lower bounds, requiring $O(n \log n)$ time, but they have noticeably slowed down the algorithm without any substantial increase in the number of problems solved.) We consider six lower bounds for each node (σ_1, σ_2). We calculate lower bounds on the lengths of different paths for the unfixed jobs S, and we combine these in various ways with the actual lengths of the fixed sequences σ_1 and σ_2. For σ_1 we look at the forward problem and we let $C^1(\sigma_1)$ and $C^2(\sigma_1)$ be the completion times on machines 1 and 2, respectively. For S we consider the forward problem, and assume that these jobs cannot start on machines 1 and 2 before $C^1(\sigma_1)$ and $C^2(\sigma_1)$, respectively. Ignoring release times for the jobs in S, let $L_1(S)$ be the completion time on machine 2 of the Johnson sequence on S. Let $L_2(S)$ be the completion time on machine 1 of the jobs in S sequenced in nondecreasing r order, that is, $L_2(S)$ is the earliest time at which the jobs in S can be completed on machine 1. Similarly, if we relax the capacity constraint on machine 1 and sequence the jobs in nondecreasing $r + a$ order, then the length of this schedule on machine 2, $L_3(S)$, is a lower bound on when the jobs in S can complete on machine 2. For σ_2 we define the completion times $C^1(\sigma_2)$ and $C^2(\sigma_2)$ on machines 1 and 2, respectively, for the reverse problem. Once again, we assume that the jobs in S cannot start before these times. We let $L_4(S)$ be the completion time on machine 1 of the reverse Johnson sequence. As for the forward problem, the earliest that the jobs in S can complete on machine 2 in the reverse problem is to sequence them in nondecreasing q order, we denote the length of this schedule on machine 2 by $L_5(S)$. Relaxing the capacity constraint on machine 2, and sequencing the jobs on machine 1 in nondecreasing $q + b$ order, the length of this schedule denoted by $L_6(S)$, is a lower bound on when the jobs in S finish on machine 1 in the reverse problem. We compute the following for the node (σ_1, σ_2):

$$\begin{aligned}
LB_1 &= L_1(S) + C^2(\sigma_2) \\
LB_2 &= L_2(S) + C^1(\sigma_2) \\
LB_3 &= L_3(S) + C^2(\sigma_2) \\
LB_4 &= C^1(\sigma_1) + L_4(S) \\
LB_5 &= C^2(\sigma_1) + L_5(S) \\
LB_6 &= C^1(\sigma_1) + L_6(S).
\end{aligned}$$

Finally, the lower bound is the maximum of the above six lower bounds and the lengths of the fixed sequences σ_1 and σ_2 (in the forward and reverse problems), $L'_{\max}(\sigma_1)$ and $L'_{\max}(\sigma_2)$, respectively.

At the root node we evaluate two additional lower bounds, these are single machine preemptive bounds obtained by relaxing the capacity constraints on each of machines 1 and 2 in the forward problem, respectively. It is well known that these are solved by the preemptive ready Jackson rule, which requires $O(n \log n)$ time to compute.

4.3.3 Decomposition and dominance

We find a starting sequence σ_1 by applying the following simple decomposition procedure, which is a generalization of the one used in [41] for the problem $F2/r_i/C_{max}$. Given a sequence s, then partial sequence $s^{k-1} = (s(1), \ldots, s(k-1))$ is an optimal initial sequence if there exists a $k \in [2, n]$ such that $\min_{k \le i \le n} r_{s(i)} \ge C^1_{s(k-1)}$, $\min_{k \le i \le n} [r_{s(i)} + a_{s(i)}] \ge C^2_{s(k-1)}$, and we have $LB(J \backslash (s^{k-1})) \ge L'_{\max}(s^{k-1})$, where $LB(J \backslash (s^{k-1}))$ is computed as follows. We apply the decomposition procedure for the jobs sequenced in nondecreasing r order, and we use the two preemptive single machine bounds mentioned in the last paragraph of the previous section to determine $LB(J \backslash (s^{k-1}))$. Then the initial partial sequence σ_1 is s^{k-1} for the largest k value found.

After we have fixed σ_1, we determine the dominance order $\prec$ on the remaining jobs in $S = J \backslash S_1$. In addition, it is possible to dynamically update the dominance order $\prec$ at each node (σ_1, σ_2), to see whether fixing jobs in σ_1 and σ_2 can add new comparabilities, thus further reducing the amount of branching.

4.4 Computational experiment

4.4.1 Test problems

For each problem with n jobs $4n$ integer data (r_i, a_i, b_i, q_i) were generated. The processing times a_i and b_i were both uniformly distributed between $[1, 100]$. Release times r_i and delivery times q_i were uniformly distributed in the range $[0, n \cdot 101 \cdot R]$ and $[0, n \cdot 101 \cdot Q]$, respectively, following the technique used by Hariri and Potts [16]. Different R and Q values were tested for values $Q \le R$ and $R \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. For each individual R, Q, and n combination, 50 problems were generated. We used the random number generator of Taillard [42] to generate these problems. This means that all of our test problems are reproducible by running the problem generation procedure from the same seeds. (The seeds used have been saved and are available from the author on request.)

4.4.2 Results

The branch and bound algorithm was coded in Sun Pascal 4.2 and run on a Sun Sparc5. Three separate versions of the algorithm were run. Algorithm A_1 has the dominance order $\prec$ 'turned off'. Algorithm A_2 has the dominance order 'turned on', and it dynamically updates $\prec$ at each node. Finally, algorithm A_3 also has the dominance order 'turned on', but $\prec$ is only calculated once at the root node. Each version of the algorithm was run until either it obtained the optimal solution or the number of nodes branched from in the tree reached one million for the problem. In the latter case, the problem was declared unsolved.

Tables 4.3, 4.4, and 4.5 contain the results of the computational experiment for the different R and Q values. For each group of problems we report: the fraction of problems solved (denoted by *solved*); for each of the unsolved problems we calculate the gap between the best solution obtained and the smallest lower bound among the left-over nodes in the tree, and the maximum gap (denoted *max gap*) is the largest of these over all the problems in the group; the total number of nodes in all of the trees (denoted by *total nodes*); the average CPU time for the solved problems in the group (denoted by *avg CPU*); and the total CPU time required for all of the problems in the group (denoted by *total CPU*).

Recall, a problem was 'unsolved' when the number of nodes branched from reached a million. Thus the total number of nodes for a group of 50 problems, contains a million nodes for each unsolved problem. As can be seen, the algorithms typically generated much fewer nodes for the solved problems. In those groups where there were unsolved problems, the solution obtained was always nearly optimal as measured by the maximum gap. The largest such gap was on the order of 3 percent, meaning that, in the worst case, we were within this range of the optimal solution. The average gap was much smaller, less than 0.5 percent.

As the results indicate, all three versions of the new branch and bound algorithm proved very effective in solving the test problems. In total, each of the three solved 4,384 of the 4500 randomly generated test problems with up to 200 jobs. There were differences in the time-performance of the three versions, however. In general, algorithms A_2 and A_3 (with the dominance order) required less time than version A_1 (without the dominance order). Algorithms A_2 and A_3 needed roughly 5 percent and 15 percent less total CPU time, respectively, than A_1 for the 4500 test problems. For the 4384 solved problems, the savings in total CPU time amounted to 15 percent and 20 percent, respectively, when compared to A_1. Thus, both A_2 and A_3 made the search through the tree faster, but the substantially larger overhead of A_2 (i.e., of updating $\prec$ at every node of the tree) offset a large part of the speed-up. In summary, algorithm A_3 was the most effective method, it was extremely fast on all of the solved problems, never requiring more than a few seconds.

The very extensive computational experiment has led to further insights. Our results suggest that the difficulty in solving a particular problem depends *much more* on the values of R and Q, than it does on the number of jobs. Given this observation, the difficult problems are those with intermediate R values, with the most difficult being the problems with $R = 0.4$. We have also found that for a given R, the problems with small Q are most difficult. Recall, that the actual size of the problem for the branch and bound algorithm is determined by how many jobs are fixed by the decomposition procedure. We noticed some trends regarding the effectiveness of the decomposition procedure. First, and most importantly, we noticed that as R increases the percentage of jobs fixed tends to increase. We also noticed that for a given R value, the percentage of jobs fixed decreases for increasing Q values, where recall that $Q \leq R$. For small R values with $R \leq 0.4$, the decomposition procedure is ineffective and fixes at most 1 and 6 percent of the jobs for $R = 0.2$ and 0.4, respectively. Moreover, for these values the decomposition procedure tends to become less effective as the number of jobs increases. For larger R values with $R \geq 0.6$, the decomposition procedure is quite effective, and it performs even better as the number of jobs increases. For $R = 0.6$, the decomposition procedure fixes around 40 percent of the jobs for the problems with $n = 20$, this increases to around 80 percent for $n = 200$. For $R = 1.0$, the percentage of jobs fixed increases to around 75 percent and 90 percent for problems with $n = 20$ and $n = 200$, respectively.

4.5 Summary

This Chapter considered a new branch and bound algorithm for the problem $F2\,/r_j, perm/\,L'_{\max}$. The main features of the proposed algorithm are: an adaptive branching rule that fixes jobs at both ends of the schedule, a new dominance order to reduce branching together with new fuzzy dominance properties that are used for scheduling and tie-breaking, and a simple decomposition procedure that reduces the problem size by fixing jobs at the beginning of the schedule. In general, computational results indicated that the algorithm performed very well. It has solved more than 97% of the test problems with up to 200 jobs within a few seconds. Even when we failed to solve a problem, we were always very close to the optimal solution. We found that the use of the dominance order resulted in a reduction in computation time of 15 to 20 percent. We also found that the use of fuzzy dominance properties aided the search by breaking ties and by often finding a nearly optimal initial solution. In summary, we feel that the proposed algorithm is more effective than previous solution methods for the problem $F2\,/r_j, perm/\,L'_{\max}$, in that it found fewer hard problems and it was also able to solve much larger problems. The fact that most problems were solved to optimum within a few seconds, means that the algorithm has the potential of being used as a callable subroutine for $F2\,/r_j, perm/\,L'_{\max}$-type subproblems generated during the solution of more complex scheduling problems.

Table 4.3: Results for (0.2, 0.2) and (0.4, 0.2).

n	*alg*	*solved*	*max gap* (%)	*total nodes*	*avg CPU* (sec)	*total CPU* (hrs:min:sec)
$R = 0.2$, $Q = 0.2$						
	A_1	50/50	-	24,666	0.0840	4.20
20	A_2	50/50	-	24,593	0.0958	4.79
	A_3	50/50	-	24,666	0.0828	4.14
	A_1	50/50	-	3,894	0.1146	5.73
40	A_2	50/50	-	3,893	0.1190	5.95
	A_3	50/50	-	3,894	0.1098	5.49
	A_1	48/50	1.1401	2,001,789	0.2225	30:57.50
60	A_2	48/50	1.1401	2,001,791	0.2056	33:17.86
	A_3	48/50	1.1401	2,001,791	0.1954	29:33.53
	A_1	49/50	0.4968	1,009,292	1.0969	20:16.54
80	A_2	49/50	0.4968	1,009,278	1.1967	22:48.84
	A_3	49/50	0.4968	1,009,288	1.0549	19:53.81
	A_1	48/50	0.6178	2,029,119	4.6633	2:40:15.34
100	A_2	48/50	0.6178	2,048,242	6.9856	2:16:42.26
	A_3	48/50	0.6178	2,048,242	5.9825	2:01:24.72
	A_1	49/50	0.0655	1,063,650	38.7327	8:02:25.27
200	A_2	49/50	0.0655	1,063,647	24.1063	7:06:01.35
	A_3	49/50	0.0655	1,063,647	22.0294	6:11:14.99
$R = 0.4$, $Q = 0.2$						
	A_1	48/50	1.6575	2,947,498	2.9265	7:32.65
20	A_2	48/50	1.6575	2,900,048	3.1988	8:11.50
	A_3	48/50	1.6575	2,903,012	2.7614	7:14.97
	A_1	38/50	1.4881	13,365,909	7.3640	49:02.53
40	A_2	38/50	1.4881	13,364,572	7.0429	44:46.62
	A_3	38/50	1.4881	13,365,936	6.7934	42:02.63
	A_1	37/50	1.0747	13,323,626	1.5924	49:22.47
60	A_2	37/50	1.0747	13,323,151	1.7548	43:38.78
	A_3	37/50	1.0747	13,323,293	1.5751	44:37.38
	A_1	41/50	0.8299	9,758,575	3.8098	30:58.02
80	A_2	41/50	0.8299	9,769,357	4.1583	33:56.53
	A_3	41/50	0.8299	9,782,955	3.8634	31:27.20
	A_1	42/50	0.4259	8,232,036	2.9133	2:03:40.23
100	A_2	42/50	0.4259	8,203,264	2.3276	2:20:44.43
	A_3	42/50	0.4259	8,203,030	2.4000	2:00:28.92
	A_1	39/50	0.2801	12,391,165	15.8108	1:07:04.76
200	A_2	39/50	0.2801	12,391,035	16.7367	1:12:45.20
	A_3	39/50	0.2801	12,391,171	15.5379	1:07:01.03

Table 4.4: Results for $(0.4, 0.4)$ and $R = 0.6$.

n	*alg*	*solved*	*max gap* (%)	*total nodes*	*avg CPU* (sec)	*total CPU* (hrs:min:sec)
$R = 0.4$, $Q = 0.4$						
	A_1	45/50	3.2119	5,505,470	1.9667	13:31.30
20	A_2	45/50	3.2119	5,474,584	1.6951	14:21.46
	A_3	45/50	3.2119	5,468,818	1.5591	12:53.13
	A_1	43/50	1.3497	7,000,924	0.0623	30:22.67
40	A_2	43/50	1.3497	7,000,874	0.0642	30:20.04
	A_3	43/50	1.3497	7,000,879	0.0607	27:27.53
	A_1	46/50	0.2151	4,001,078	0.1924	18:50.55
60	A_2	46/50	0.2151	4,001,105	0.1935	18:55.89
	A_3	46/50	0.2151	4,001,111	0.1820	17:25.11
	A_1	47/50	0.6494	3,000,978	0.3975	15:54.83
80	A_2	47/50	0.6494	3,001,023	0.3966	9:51.74
	A_3	47/50	0.6494	3,001,013	0.3753	11:24.47
	A_1	48/50	0.4598	2,001,628	0.8077	8:10.88
100	A_2	48/50	0.4598	2,001,549	0.7938	6:40.89
	A_3	48/50	0.4598	2,001,582	0.7523	8:14.86
	A_1	49/50	0.1693	1,002,837	6.4861	22:23.65
200	A_2	49/50	0.1693	1,002,824	6.4520	22:36.68
	A_3	49/50	0.1693	1,002,841	6.0212	19:50.06
$R = 0.6$, $Q = 0.2, 0.4, 0.6$						
	A_1	147/150	1.3150	3,120,585	0.0995	6:22.79
20	A_2	147/150	1.3150	3,098,357	0.0852	6:37.17
	A_3	147/150	1.3150	3,099,007	0.0767	6:25.34
	A_1	137/150	1.3754	13,100,696	0.1467	36:56.68
40	A_2	137/150	1.3754	13,103,281	0.1450	36:03.00
	A_3	137/150	1.3754	13,103,391	0.1322	31:22.49
	A_1	147/150	0.7059	3,055,775	0.1070	13:34.11
60	A_2	147/150	0.7059	3,036,978	0.0748	12:32.31
	A_3	147/150	0.7059	3,036982	0.0707	11:38.68
	A_1	145/150	0.3267	5,003,273	0.0682	11:43.19
80	A_2	145/150	0.3267	5,003,245	0.0657	24:34.21
	A_3	145/150	0.3267	5,003,245	0.0635	14:08.36
	A_1	148/150	0.4820	2,006,622	0.0818	4:28.17
100	A_2	148/150	0.4820	2,006,722	0.0845	4:40.68
	A_3	148/150	0.4820	2,006,722	0.0810	3:23.59
	A_1	147/150	0.1533	3,000,729	0.2136	17:11.18
200	A_2	147/150	0.1533	3,000,778	0.2206	18:39.67
	A_3	147/150	0.1533	3,000,732	0.2064	15:32.97

Table 4.5: Results for $R = 0.8$ and $R = 1.0$.

n	*alg*	*solved*	*max gap* (%)	*total nodes*	*avg CPU* (sec)	*total CPU* (hrs:min:sec)
$R = 0.8$, $Q = 0.2, 0.4, 0.6, 0.8$						
	A_1	200/200	-	1164	0.0026	0.51
20	A_2	200/200	-	1149	0.0029	0.58
	A_3	200/200	-	1149	0.0027	0.54
	A_1	200/200	-	914	0.0056	1.11
40	A_2	200/200	-	912	0.0056	1.11
	A_3	200/200	-	912	0.0053	1.06
	A_1	199/200	0.2625	1,001,163	0.0098	2:32.97
60	A_2	199/200	0.2625	1,001,167	0.0097	6:53.71
	A_3	199/200	0.2625	1,001,167	0.0095	2:33.83
	A_1	198/200	0.1513	2,000,359	0.0013	6:10.13
80	A_2	198/200	0.1513	2,000,357	0.0012	3:35.69
	A_3	198/200	0.1513	2,000,357	0.0012	3:30.68
	A_1	200/200	-	257	0.0181	3.61
100	A_2	200/200	-	261	0.0183	3.65
	A_3	200/200	-	261	0.0180	3.59
	A_1	200/200	-	259	0.0571	11.42
200	A_2	200/200	-	267	0.0569	11.37
	A_3	200/200	-	267	0.0568	11.36
$R = 1.0$, $Q = 0.2, 0.4, 0.6, 0.8, 1.0$						
	A_1	250/250	-	376	0.0020	0.49
20	A_2	250/250	-	543	0.0022	0.54
	A_3	250/250	-	543	0.0023	0.57
	A_1	250/250	-	332	0.0042	1.04
40	A_2	250/250	-	329	0.0043	1.08
	A_3	250/250	-	329	0.0043	1.08
	A_1	250/250	-	354	0.0076	1.91
60	A_2	250/250	-	354	0.0080	1.99
	A_3	250/250	-	354	0.0082	2.06
	A_1	249/250	0.2736	1,000,309	0.0119	2:36.63
80	A_2	249/250	0.2736	1,000,307	0.0121	3:14.71
	A_3	249/250	0.2736	1,000,307	0.0121	3:13.30
	A_1	250/250	-	273	0.0177	4.43
100	A_2	250/250	-	273	0.0178	4.45
	A_3	250/250	-	273	0.0178	4.45
	A_1	250/250	-	276	0.0619	15.47
200	A_2	250/250	-	276	0.0620	15.49
	A_3	250/250	-	276	0.0620	15.50

Chapter 5
Thesis summary

In this thesis, we have introduced a new technique, subset-restricted pairwise interchange, that generalizes well-known methods used to derive dominance results for scheduling problems. Subset-restricted pairwise interchange is a generalization of pairwise job interchange that incorporates subset-restrictions on intermediate jobs. The traditional methods of pairwise job interchange can be viewed as special cases, where these subsets are all uniformly either the empty set (adjacent pairwise interchange) or the entire job set (nonadjacent pairwise interchange).

We have used this technique to derive dominance orders for the one- and two-machine scheduling problems with release times $1\,/r_j/\,f_{\max}$, $F2\,/r_j/\,C_{\max}$, and $F2\,/r_j, perm/\,L'_{\max}$. The dominance order we derive is a suborder of the adjacent interchange order for the problem. We have shown that in general the adjacent interchange order is not a dominance order, but that there is a subset of this order that is a dominance order. These results generalize the dominance results of Erschler et al. [8], [9], and [10], for the one-machine scheduling problems $1\,/r_j, \overline{d_j}/\,C_{\max}$ and $1\,/r_j/\,L_{\max}$.

We have tested the effectiveness of the dominance orders in efficient branch and bound algorithms. We found that the dominance orders were effective at reducing the number of branches in the search tree, and that this led to a sizeable reduction in the total computation times. Our experiments indicated that the difficulty in solving these problems depends much more on the range of release (and delivery) times than on the number of jobs. We have classified problem instances as either 'easy' or 'hard' on this basis. Our test problems were generated using the random number generator of Taillard [42], thus they are easily reproducible, and can serve as benchmark problems for future researchers.

There are several directions for future research. A natural extension would be to consider problems with more than two machines (i.e. $m > 2$) such as $Fm\,/r_j/\,C_{\max}$, $Fm//C_{\max}$, and $Fm\,/r_j/\,L'_{\max}$. These techniques could be applied directly to derive dominance orders, or the proposed algorithms could be used as subroutines to solve $F2\,/r_j/\,C_{\max}$- or $F2\,/r_j, perm/\,L'_{\max}$-type subproblems generated during the solution process. Another extension would be to apply these techniques to problems where the objective is to minimize the total cost, instead of the maximum cost. Continuing in this vein, still another extension would be to consider multiple criteria combinations of the above two types of objectives. These directions will be the subject of future research.

Appendix A
Algorithm for $F2/r_j/C_{\max}$

Calculations for root node

- apply decomposition procedure with nondecreasing r sequence to fix σ_1;
- let $S := J \backslash S_1$, $\sigma_2 := \varnothing$; $nodes := 0$;
- construct the dominance order $\prec$;
- calculate initial UB and LB, and let $BestMakespan := UB$;
- call recursive procedure Opt that facilitates branching;

```
procedure $Opt(\sigma_1, S, \sigma_2, BestMakespan)$;
begin
  if $nodes \leq 1000000$ then
  begin
     $nodes := nodes + 1$;{increment the number of nodes branched from}
     if ($nodes \leq n$) or ($|S| = 1$) then calculate $UB$ and (possibly) update
     $BestMakespan$
     if ($LB < BestMakespan$) and ($|S| \neq 1$) then
     begin
          find lists of minimal and maximal jobs in $\prec|_S$
          if $type = 1$ then calculate lower bounds for minimal and sort in
          nondecreasing order
          else if $type = 2$ then calculate lower bounds for maximal and sort in
          nondecreasing order
          else ($type$ not set)
          begin
```

```
            must calculate and sort lower bounds for both $minimal$ and
            $maximal$ to find $n_1$ and $n_2$ to determine $type$ for node
            if $n_1 < n_2$ then $type := 1$
            else if $n_1 > n_2$ then $type := 2$
            else $type := previous\_type$;
        end;
        if $type = 1$ then $list := minimal$
        else if $type = 2$ then $list := maximal$;
        while ($list \neq \varnothing$) and ($BestMakespan >$ lower bound from head of $list$)
        do
        begin
            remove head of $list$
            let $job$ be the job that is fixed and let $LB$ be its lower bound
            remove $job$ from $\prec$
            $S := S \setminus \{job\}$;
            if $type = 1$ then add $job$ to $\sigma_1$
            else if $type = 2$ then add $job$ to $\sigma_2$;
            $Opt(\sigma_1, S, \sigma_2, BestMakespan)$;{recursive call}
            if $type = 1$ then remove $job$ from $\sigma_1$
            else if $type = 2$ then remove $job$ from $\sigma_2$;
            $S := S \cup \{job\}$;
            restore $job$ to $\prec$
        end;{while}
    end;{if ($LB < BestMakespan$) and ($|S| \neq 1$)}
  end;{if $nodes \leq 1000000$}
end;
```

Appendix B
Algorithm for $F2/r_j, perm/L'_{\max}$

Calculations for root node

- apply decomposition procedure with nondecreasing r sequence to fix σ_1;
- let $S := J \backslash S_1$, $\sigma_2 := \varnothing$; *nodes* $:= 0$;
- construct the dominance order $\prec$;
- find fuzzy sequence 1and fuzzy sequence 2;
- calculate initial UB and LB, and let $BestUB := UB$;
- call recursive procedure *Opt* that facilitates branching;

```
procedure $Opt(\sigma_1, S, \sigma_2, BestUB)$;
begin
  if $nodes \leq 1000000$ then
  begin
     $nodes := nodes + 1$;{increment the number of nodes branched from}
     if $(nodes \leq n)$ or $(|S| = 1)$ then calculate $UB$ and (possibly) update $BestUB$
     if $(LB < BestUB)$ and $(|S| \neq 1)$ then
     begin
          find lists of minimal and maximal jobs in $\prec|_S$
          if $type = 1$ then calculate lower bounds for minimal and sort in
          nondecreasing lower bound order breaking ties using fuzzy sequence 1
          else if $type = 2$ then calculate lower bounds for maximal and sort in
          nondecreasing lower bound order breaking ties using fuzzy sequence 2
          else (type not set)
          begin
```

```
            calculate and sort lower bounds for both *minimal* and *maximal*
            breaking ties accordingly
            find $n_1$ and $n_2$ to determine *type* for node
            **if** $n_1 < n_2$ **then** $type := 1$
            **else if** $n_1 > n_2$ **then** $type := 2$
            **else** $type := previous\_type$;
        **end**;
        **if** $type = 1$ **then** $list := minimal$
        **else if** $type = 2$ **then** $list := maximal$;
        **while** ($list \neq \varnothing$) **and** ($BestUB >$ lower bound from head of *list*) **do**
        **begin**
            remove head of *list*
            let *job* be the job that is fixed and let $LB$ be its lower bound
            remove *job* from $\prec$
            $S := S \backslash \{job\}$;
            **if** $type = 1$ **then** add *job* to $\sigma_1$
            **else if** $type = 2$ **then** add *job* to $\sigma_2$;
            $Opt(\sigma_1, S, \sigma_2, BestUB)$;{recursive call}
            **if** $type = 1$ **then** remove *job* from $\sigma_1$
            **else if** $type = 2$ **then** remove *job* from $\sigma_2$;
            $S := S \cup \{job\}$;
            restore *job* to $\prec$
        **end**;{while}
    **end**;{if ($LB < BestUB$) and ($|S| \neq 1$)}
  **end**;{if $nodes \leq 1000000$}
**end**;
```

Bibliography

[1] Baker, K.R., and Z.S. Su (1974). Sequencing with due dates and early start times to minimize maximum tardiness. *Naval Res. Logist. Quart.* 21, 171-176.

[2] Baker, K.R., E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (1983). Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Oper. Res.* 31,381-386.

[3] Carlier, J. (1982). The one-machine sequencing problem. *European J. Oper. Res.* 11, 42-47.

[4] Cheng, J., H. Kise, and H. Matsumoto (1997). A branch and bound algorithm with fuzzy inference for a permutation flowshop scheduling problem. *European J. Oper. Res.* 96, 578-590.

[5] Della Croce, F. (1995). Generalized pairwise interchanges and machine scheduling. *European J. Oper. Res.* 83, 310-319.

[6] Dushnik, B., and E.W. Miller (1941). Partially ordered sets. *Amer. J. Math.* 63, 600-610.

[7] Emmons, H. (1969). One machine sequencing to minimize certain functions of job tardiness. *Oper. Res.* 17, 701-715.

[8] Erschler, J., G. Fontan, C. Merce, and F. Roubellat (1982). Applying new dominance concepts to job schedule optimization. *European J. Oper. Res.* 11, 60-66.

[9] Erschler, J., G. Fontan, C. Merce, and F. Roubellat (1983). A new dominance concept in scheduling n jobs on a single machine with ready times and due dates. *Oper. Res.* 31, 114-127.

[10] Erschler, J., G. Fontan, and C. Merce (1985). Un nouveau concept de dominance pour l'ordonnancement de travaux sur une machine. *R.A.I.R.O. Recherche opérationnelle* 19, 15-26.

[11] Fishburn, P.C. (1985) *Interval Orders and Interval Graphs*, J. Wiley, New York, N.Y.

[12] Fontan, G. (1980). Notion de dominance et son application à l'étude de certains problèmes d'ordonnancement, Thèse de Doctorat ès Sciences, Université Paul Sabatier, Toulouse, France.

[13] Garey, M.R., and D.S. Johnson (1979) *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

[14] Grabowski, J. (1980). On two-machine scheduling with release and due dates to minimize maximum lateness. *Opsearch* 17, 133-154.

[15] Grabowski, J., E. Skubalska, and C. Smutnicki (1983). On flow shop scheduling with release and due dates to minimize maximum lateness. *J. Oper. Res. Soc.* 34, 615-620.

[16] Hariri, A.M., and C.N. Potts (1983). An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Appl. Math.* 5, 99-109.

[17] Hall, L.A. (1993). A note on generalizing the maximum lateness criterion for scheduling. *Discrete Appl. Math.* 47, 129-137.

[18] Hall, L.A. (1994). A polynomial approximation scheme for a constrained flow shop scheduling problem. *Math. Opns. Res.* 19, 68-85.

[19] Hall, L.A., and D.B. Shmoys (1992). Jackson's rule for single machine scheduling making a good heuristic better. *Math. Opns. Res.* 17, 22-35.

[20] Jackson, J.R. (1955). Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, Univ. of California, Los Angeles.

[21] Johnson, S.M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Res. Logist. Quart.* 1, 61-68.

[22] Kovalyov, M.Y., and F. Werner (1997). A polynomial approximation scheme for problem $F2/r_j/C_{\max}$. *Oper. Res. Lett.* 20, 75-79.

[23] Lageweg, B.J., J.K. Lenstra, and A.H.G. Rinnooy Kan (1976). Minimizing maximum lateness on one machine: computuational experience and some applications. *Stast. Neerlandica* 30, 25-41.

[24] Lageweg, B.J., J.K. Lenstra, and A.H.G. Rinnooy Kan (1978). A general bounding

scheme for the permutation flowshop problem. *Oper. Res.* 26, 53-67.

[25] Lawler, E.L. (1973) Optimal sequencing of a single machine subject to precedence constraints. *Management Science* 19, 544-546.

[26] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (1989). Sequencing and scheduling: Algorithms and complexity. Report BS-R8909, CWI Amsterdam.

[27] Lenstra, J.K., A.H.G. Rinnooy Kan, and P. Brucker (1977). Complexity of machine scheduling problems. *Annals of Disc. Math.* 1, 343-362.

[28] McMahon, G.B., and M. Florian (1975). On scheduling with ready times and due dates to minimize maximum lateness. *Oper. Res.* 23, 475-482.

[29] Merce, C. (1979). Etude de l'existence de solutions pour certains problèmes d'ordonnancement, Thèse de Doctorat Ingénieur, Université Paul Sabatier, Toulouse, France.

[30] Monma, C.L. (1978). Properties and efficient algorithms for certain classes of sequencing problems, Ph. D. Thesis, Cornell University, Ithaca, New York.

[31] Papadimitriu, C.H., and M. Yannakakis (1979). Scheduling interval-ordered tasks. *SIAM J. Computing* 8, 405-409.

[32] Pinedo, M. (1995) *SCHEDULING: Theory, Algorithms and Systems*, Prentice Hall, Englewood Cliffs, N.J.

[33] Potts, C.N. (1980). Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Oper. Res.* 28, 1436-1441.

[34] Potts, C.N. (1980). An adaptive branching rule for the permutation flow-shop problem. *European J. Oper. Res.* 5, 19-25.

[35] Potts, C.N. (1985). Analysis of heuristics for two-machine flow-shop sequencing subject to release dates. *Math. Opns. Res.* 10, 576-584.

[36] Rinnooy Kan, A.H.G.,B.J. Lageweg, and J.K. Lenstra (1975). Minimizing total costs in one machine scheduling. *Oper. Res.* 23, 908-927.

[37] Smith, W.E. (1956). Various optimizers for single-stage production. *Naval Res. Logist. Quart.* 3, 59-66.

[38] Szwarc, W. (1990). Parametric precedence relations in single machine scheduling. *Oper. Res. Lett.* 9, 133-140.

[39] Szwarc, W., M.E. Posner, and J.J. Liu (1988). The single machine problem with a quadratic cost function of completion times. *Management Sci.* 34, 1480-1488.

[40] Szwarc, W., and J.J. Liu (1993). Weighted tardiness single machine scheduling with proportional weights. *Management Sci.* 39, 626-632.

[41] Tadei, R., J.N.D. Gupta, F. Della Croce, and M. Cortesi (1998). Minimising makespan in the two-machine flow-shop with release times. *J. Oper. Res. Soc.* 49, 77-85.

[42] Taillard, E. (1993). Benchmarks for basic scheduling problems. *European J. Oper. Res.* 64, 278-285.

[43] Zdralka, S., and J. Grabowski (1989). An algorithm for single machine sequencing with release dates to minimize maximum cost. *Discrete Appl. Math.* 23, 73-89.