Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

COMPUTATIONALLY TRACTABLE CLASSES OF ORDERED SETS

ROLF H. MÖHRING

No. 181/1987

appeared in I. Rival (ed.) Algorithms and Order,
Kluwer Acad. Publ., Dordrecht, 1989, p. 105-194

COMPUTATIONALLY TRACTABLE CLASSES OF ORDERED SETS

Rolf H. Möhring
Technical University of Berlin
Department of Mathematics
Straße des 17. Juni 135
1000 Berlin 12

ABSTRACT

Ordered sets have recently gained much importance in many applied and theoretical problems in computer science and operations research ranging from project planning via processor scheduling to sorting and retrieval problems. These problems involve partial orders as their basic structure, e.g. as precedence constraints in scheduling problems, or as comparability relation among the objects to be sorted or retrieved.

Since many of the involved problems are NP–hard in general, much attention has recently been given to special classes of partial orders with "nice" structural properties that lend themselves the design of efficient methods, and for obtaining bounds by structural relaxation in more general situations. Typical such classes are: series parallel partial orders, N–free partial orders, interval orders, two–dimensional partial orders, and partial orders with special decomposition properties.

This area of "computationally tractable" classes of partial orders shows many similarities and interactions with algorithmic graph theory and certain classes of perfect graphs.

We will present the structural properties of the mentioned classes, discuss their mutual relationship, and the algorithmic complexity of their recognition. In addition, we present the tractibility of these different classes on several applications dealing with scheduling and sorting.

Contents

1. Introduction

Due to the proverbial intractability (i.e. NP–completeness) of the majority of computational problems occuring in the theory of ordered sets and its applications to computer science and operations research, much interest has been paid to classes of ordered sets that still admit efficient algorithms for otherwise intractable problems. The tractability of these classes is in most cases a consequence of rather strong structural properties not shared by arbitrary partial orders.

These properties can have different aspects such as *identification with other mathematical objects* (e.g. 2–dimensional partial orders with permutations, interval orders with a collection of intervals on the real line etc.); *concise encodings* (e.g. by labeled trees for series–parallel partial orders, by two listings of the elements for 2–dimensional partial orders); *characterization by forbidden substructures* (e.g. by the "N" for series–parallel partial orders); and *recursive construction principles* or *decomposition properties.*

These aspects are then used to obtain solution methods for a great variety of practical problems. These applications motivate, in turn, the development of efficient methods for *recognizing* these partial orders and for *constructing* the associated representation or object required in the solution of the original problem.

While the structural theory is by now quite well understood, there has been much progress during the last few years in the design of highly efficient recognition algorithms and suitable data structures for their representation.

The presentation of these algorithmic aspects and their theoretical foundation is the main concern of this paper. In addition, a selection of applications (mostly from scheduling and sorting) is given in order to show the "algorithmic power" of these aspects. Due to the rapidly growing research in this area, it was impossible to cover every recent development. The choice on the classes of ordered sets presented here was (with a few exeptions) motivated by two properties, *concise representation* and/or *polynomial isomorphism testing.* Therefore, bipartite partial orders, or partial orders of bounded height are not covered. Also, the material chosen is largely complementary to already existing monographs or survey articles on some of the classes (e.g. interval orders and 2–dimensional orders).

For each of the classes covered, we will first discuss the main structural properties of this class and, where necessary, of related graph classes (e.g. series–parallel partial orders and cographs, interval orders and interval graphs). These properties are used to derive the recognition methods and data structures. Each section concludes with selected applications and, if relevant, with a discussion of important subclasses.

1.1 An Overview

Section 2 deals with *series–parallel partial orders.* They may be identified with *arithmetic expressions* with the arithmetic operations $+$ and $*$. This identification leads to a natural *tree representation* as a labeled tree in $O(\#$ elements) space (i.e. *sublinear* in the number of elements and ordered pairs that is necessary to encode arbitrary partial orders). The tree, in turn, is used in most of the applications and leads e.g. directly to a polynomial

algorithm for *isomorphism testing.*

Section 2.2 gives a considerably simplified proof of the characterization via the forbidden "N" that uses the relationship to *cographs* (the comparability graphs of series–parallel partial orders) and also unifies the theory for related structures. Section 2.3 presents the most efficient (linear) recognition algorithms by [CPS] and [VTL]. The first uses cographs and works on–line, while the second can also work with the transitive reduction and uses essentially the fact that series–parallel partial orders are *N–free* and *2–dimensional* by exploiting the associated representations for these classes.

Section 3 deals with *N–free* or *quasi–series–parallel* partial orders. They generalize series–parallel partial orders and may be identified with the ordering of the set of *edges* in a *directed acyclic graph.* They still have a *sublinear encoding* by a decomposition tree, but are not *hereditary* and thus only of limited tractibility. Their main area of applications is the use as PERT–networks in project analysis, cf. Section 3.3. The characterization of series–parallel partial orders by the forbidden "Wheatstone bridge" [Du] turns out to be just a characterization of series–parallel partial orders within this class of N–free partial orders.

Section 4 considers another generalization of series–parallel partial orders that arises in connection with the *substitution decomposition* when the *size* or the *height* of the building blocks (factors) are bounded. The appropriate data structure for their representation is the *canonical decomposition tree* associated with the substitution decomposition. It reflects the *recursive structure* of these partial orders and leads to dynamic programming algorithms for many applications, in particular in connection with linear extensions (*scheduling*) and order ideals (*searching and sorting*), cf. Section 4.2.

Section 5 deals with *2–dimensional partial orders*, yet another generalization of series–parallel partial orders. These partial orders can be identified with *permutations* and have *concise encodings* with sublinear space in the form of two listings of the elements. Different to the previous classes, they do not have a characterization by finitely many forbidden substructures.

Section 5.2 presents the fastest know recognition algorithm recently developed by [Sp1,SV]. They are based on the *substitiution decomposition* and can also be used to obtain a concise encoding and polynomial algorithms for *isomorphism testing.*

The last section deals with *interval orders.* They can be identified with a *collection of intervals* along the real line, and any such collection defines a *concise encoding* with sublinear space.

Due to the many already existing surveys and monographs on interval orders and interval graphs, the main emphasis is put on *recognition*, *seriation*, and applications in *scheduling.*

Concerning recognition and seriation, the recently introduced *MPQ–trees* provide a powerful and yet simple data structure. Intuitively, it represents all interval orders with the same interval graph on sublinear space. It is closely related to the decomposition tree for the substitution decomposition and can be constructed in linear time (Section 6.2).

Concerning scheduling, special attention is given to the description of the solution space of scheduling problems by interval orders. In this interpretation, solving a precendence and resource constrained scheduling problem means to find the "right" interval extension of the partial order P of precedence constraints in the lattice $\mathcal{E}(P)$ of all extensions of P.

1.2 Notation

A *partial order* will be denoted by $P = (V, <_P)$, where V is the (finite) *ground set* of *elements* or *vertices* and $<_P$ (or $<$ if no confusion is possible) is the *order relation*, i.e. an irreflexive and transitive relation whose pairs $(a,b) \in <_P$ are written as $a <_P b$ $(a,b \in V)$ with the usual interpretation. We write $a \leq_P b$ or $a \leq b$ to mean $a <_P b$ or $a = b$.

Two elements $a, b \in V$ are *comparable* in P (denoted by $a \sim_P b$) if $a < b$ or $b > a$. Otherwise they are said to be *incomparable* (denoted by $a \|_P b$). A set of pairwise comparable elements is called a *chain*, and a set of pairwise incomparable elements is called an *antichain*. (Singletons $\{v\}$ and the empty set $\emptyset$ are both chains and antichains.) The *height* $h(P)$ of P is the largest size of a chain of P minus 1. The *width* $w(P)$ of P is the largest size of an antichain of P.

Given $v \in V, Suc(v) := \{u \in V | v <_P u\}$ denotes the set of all *successors* of v, while $Pred(v) := \{u \in V | u <_P v\}$ denotes the set of all *predecessors* of v with respect to $P = (V, <_P)$. $Min(P)$ and $Max(P)$ denote the sets of all *minimal* and *maximal* elements of P, respectively.

If $u < v$ and there is no $w \in V$ with $u < w < v$ then v is said to *cover* u (denoted by $u \lessdot v$). We also say that v is an *immediate successor* of u (or u is an *immediate predecessor* of v). $ImSuc(u)$ and $ImPred(u)$ denote the sets of all immediate successors and immediate predecessors of u.

The *suborder* of $P = (V, <)$ *induced* by $U \subseteq V$ is denoted by $P|U$ with the same symbol $<$ for the order relation. P^{-1} denotes the *dual* or *inverse* order of P.

Graphs (both *directed* and *undirected*) are denoted by $G = (V, E)$, where V is the set of *vertices* or *points* and E is the set of edges.

Undirected graphs are always *simple* (i.e. without loops and multiple edges) and the edges e are denoted by the pair (u,v) of the vertices they connect. $Adj(v) = \{u \in V | (u,v) \in E\}$ denotes the set of all vertices that are *adjacent* to v. $G^c = (V, E^c)$ with $E^c = \{(u,v) | (u,v) \notin E\}$ denotes the *complementary* graph of $G = (V, E)$. If $U \subseteq V$, then $G|U$ denotes the *subgraph* of G *induced* by U.

If $P = (V, <)$ is a partial order, then $G(P) = (V, E)$ with $E = \{(u,v) | u \sim v\}$ denotes the *comparability graph* of P. So two vertices are connected by an edge in $G(P)$ iff they are comparable in P. The complementary graph $G(P)^c$ is called the *incomparability graph* of P. If G is a comparability graph, then any partial order P with $G = G(P)$ is said to be *associated* with G. Every partial order associated with G corresponds to a *transitive orientation* of the vertex set of G and vice versa [Go1].

Directed graphs may have parallel edges but no loops. A directed edge $e \in E$ is directed from its *tail* (initial point) u to its *head* (end point) v. If e has no parallel edges, we will simply write $e = (u,v)$. A *sink* is a vertex that is not a head of any edge, and a *source* is a vertex that is not a tail of any edge.

A *dag* is a directed acyclic graph (possibly with parallel edges). It is *transitively closed* if $(u,v), (v,w) \in E$ implies that $(u,w) \in E$, and *transitively reduced* if $(u,w) \in E$ implies that there is no $v \in V$ with $(u,v) \in E$ and $(v,w) \in E$. Edges (u,w) violating this condition are called *transitive* or *redundant*.

Since every partial order $P = (V, <)$ may be interpreted as a dag with vertex set V and edge set $<$, the terms "transitively closed" and "transitively reduced" carry over to partial orders. Obviousely, the transitively reduced form corresponds to the covering relation $\lessdot$.

For all other notions and definitions not explicitly stated here, we refer to [AHU] or [Go1].

1.3 Selected Applications

The applications dealt with in the next sections are mainly from the area of sorting, sequencing and scheduling. This section is meant as a brief introduction to these problems and the main terminology.

Many of these applications deal with linear extensions of a given partial order. A *linear order* is a partial order without incomparable elements. A *linear extension* of a partial order $P = (V, <_P)$ is a linear order $L = (V, <_L)$ on the same ground set V that *extends* P, i.e.

$$a <_P b \Rightarrow a <_L b \quad \text{for all} \quad a, b \in V. \tag{1.1}$$

We will write linear orders as $L = x_1 x_2 \dots x_n$, where the sequence from left to right defines the order relation $<_L$, i.e. $x_1 <_L x_2 <_L \dots <_L x_n$.

Linear extension can be used to represent a partial order in the following sense. To every partial order $P = (V, <)$, there exist linear extensions $L_i = (V, <_i)$ $(i = 1, \dots, k)$ of P whose intersection is P, i.e.

$$a < b \Leftrightarrow a <_i b \quad \text{for} \quad i = 1, \dots, k. \tag{1.2}$$

The smallest number k of linear extension in such a representation is called the *dimension* of P and is denoted by $dim(P)$. P is called *k-dimensional* if $dim(P) \leq k$. See [KT] for a survey on dimension theory.

In scheduling theory, linear extensions of $P = (V, <)$ can be seen as *1-machine schedules* that schedule the "jobs" from V subject to the *precedence constraints* represented by P. The problem then is to choose a linear extension that minimizes a certain objective. The usual scheduling objectives depend on the *completion times* of the jobs in the schedule.

More specifically, let V be numbered as $V = \{v_1, \dots, v_n\}$. Each job v_i has a *processing time* x_i (with $x = (x_1, \dots, x_n)$ denoting the vector of processing times). A (non-preemptive) *schedule* for V is a vector $S = (S_1, \dots, S_n)$ of starting times where S_i denotes the starting time of job v_i. The *completion times* of schedule S are (due to the non-preemption) obtained as $C_i = S_i + x_i$, or $C = S + x$, where $C = (C_1, \dots, C_n)$.

For the 1-machine schedule represented by the linear order $L = v_{i_1} v_{i_2} \dots v_{i_n}$, one then has $S_k = \sum_{j=1}^{k-1} x_{i_j}$ and $C_k = \sum_{j=1}^{k} x_{i_j}$.

The cost of a schedule S are usually given by a non-decreasing function $\tau : \mathbb{R}^n \to \mathbb{R}^1$ of the completion times, i.e. $\tau(C_1, \dots, C_n) = \tau(S + x)$ denotes the cost for scheduling V according to S. Typical such *cost functions* or *performance measures* are the *maximum completion time* or *makespan* $C_{max} = max\{C_1, \dots, C_n\}$, the *flowtime* $\sum_j C_j$ and the *weighted flowtime* $\sum_j w_j C_j$ (where w_j is a weight associated with job v_j), criteria involving due-dates, etc. [LLR].

In particular, the $1|prec|\sum w_j C_j$ problem then means to find a linear extension of a partial order that minimizes the weighted flowtime (in the sense that the associated

completion times C_j minimize the function $\tau(C_1, \ldots, C_n) = \sum w_j C_j$). For more information on machine scheduling problems, and in particular for the classification used above, see [LLR].

Other problems dealing with linear extensions are *counting* their number, and the *bump* and *jump number* problem.

Let $L = v_1 v_2 \ldots v_n$ be a linear extension of $P = (V, <_P)$. Two consecutive elements v_i, v_{i+1} of L are separated by a *bump* (resp. a *jump* or *setup*) if $v_i <_P v_{i+1}$ (resp. $v_i \|_P v_{i+1}$). The total number of bumps (resp. jumps) of L is denoted by $b(L,P)$ (resp. $j(L,P)$). Clearly, if $|V| = n$, then

$$b(L,P) + j(L,P) = n - 1. \tag{1.3}$$

The *bump number* $b(P)$ of P is the minimum number of bumps in some linear extension, i.e.

$$b(P) = min\{b(L,P) | L \text{ is a linear extension of } P\}. \tag{1.4}$$

A linear extension L of P with $b(L,P) = b(P)$ is called *(bump-)optimal.*

The *jump number* $j(P)$ of P and the notion of jump-optimal linear extension are defined completely analogously. See [BH] for a recent overview.

The 1-machine schedules introduced above may be viewed as best schedules respecting a linear order. In general, we say that a schedule S respects the partial order $P = (V, <_P)$ if $C_i \leq S_j$ whenever $v_i <_P v_j$. So the *componentwise best* schedule respecting P is the *earliest start schedule* $ES_P[x] = (ES_P[x](v_1), \ldots, ES_P[x](v_n))$ associated with P. It is recursively obtained as

$$ES_P[x](v_i) = \begin{cases} 0, & \text{if } Pred(v_i) = \emptyset \\ max_{v_j <_P v_i}[ES_P[x](v_j) + x_j], & \text{otherwise} \end{cases} \tag{1.5}$$

Intuitively speaking, $ES_P[x]$ starts each job as early as possible with respect to the precedence constraints P and the processing times x, i.e. either at time zero or when all predecessors are completed.

Then given any scheduling cost function $\tau : \mathbf{R}^n \to \mathbf{R}^1$, we can associate a cost $\tau(P,x)$ with any partial order P on $V = \{v_1, \ldots, v_n\}$ by putting

$$\tau(P,x) = \tau(ES_P[x] + x), \tag{1.6}$$

i.e. the cost of the completion times arising from the earliest start schedule.

These notions will be essential for applications in *project scheduling* in Sections 3,4 and 6. For instance, $C_{max}(P,x)$ denotes the *project duration* (i.e. the earliest overall completion time) of a project with jobs $V = \{v_1, \ldots, v_n\}$, technological precedence constraints $P = (V, <)$, and processing times $x = (x_1, \ldots, x_n)$.

Searching and sorting problems deal with retrieving or sorting data by exploiting only the order structure of the stored data, cf. [Sa,BHe] for overvies on these topics. From the many problems in this area, we will deal only with the so-called *general search problem*: Let $P = (V, <)$ be a partial order and $f : V \to \mathbf{N}$ be an (unknown) order-preserving function.

The search problem associated with P and f then consists in deciding whether a given number $\alpha \in \mathbb{N}$ (considered as *key* of the data to be retrieved) belongs to $f(V)$ (the *stored* data) by *evaluating* f at elements of P.

Searching and sorting is closely related to *ideals* and *antichains* of a partial order. A subset I of the ground set V of a partial order $P = (V, <)$ is called an *ideal* or *order ideal* if

$$v \in I, u < v \Rightarrow u \in I. \tag{1.7}$$

It is called a *filter* or *dual ideal* if it is an ideal of P^{-1}. Ideals are in a $1-1$ correspondence with the antichains of P, since any ideal I is uniquely determined by the antichain $Max(I)$ of its maximal elements.

It is shown in [LS] that the general search problem requires $\Theta(log_2 N(P))$ evaluations of f, where $N(P)$ denotes the number of ideals of P. The upper bound is implied by showing that any partial order P has a so–called *central element*, i.e. an element u with

$$\delta_o \leq \frac{N(u)}{N(P)} \leq 1 - \delta_o, \tag{1.8}$$

where $N(u)$ denotes the number of ideals of P containing u and δ_o is a constant between 0 and 1 that depends on P (in general, 0.228 is the best possible value [San]).

In algorithmic respect, this existence result raises the question of how to determine a central element in general or in special classes, and whether the constant δ_o can be improved for special classes of partial orders.

Finally, the *isomorphism problem* means to test whether or not two given partial orders $P_1 = (V, <_1), P_2 = (V, <_2)$ on the same ground set V are isomorphic. For arbitrary partial orders, this problem is *isomorphism complete*, i.e. as hard as the general graph isomorphism problem. The complexity status of this problem is still open, cf. [Jo] for details.

2. Series–Parallel Partial Orders

Series–parallel partial orders and related structures have been investigated and found computationally attractive in many different applications. These include: electrical networks [Du], several scheduling problems [La3, MS1], problems involving linear extensions of a partial order [CHab,St7], see also [At] in this volume, and applications in VLSI layout (macro placement [GM] and CMOS cell layout [LMü]) and parallel computing (modelling the semantics of concurrency [BC, Gi]).

This has lead to distinct terminlogies and independent (re–) discovery of the main results. We give below a streamlined exposition from the viewpoint of the theory of ordered sets.

2.1 Definition and Related Structures

Let $P_1 = (V_1, <_1)$ and $P_2 = (V_2, <_2)$ be partial orders with disjoint ground sets V_1, V_2. The *parallel composition* $P_P = (V, <_P)$ and the *series composition* $P_S = (V, <_S)$ of P_1 and P_2 are partial orders on $V = V_1 \cup V_2$ whose ordering relations are defined as

$$u <_P v :\Leftrightarrow \begin{cases} (u, v \in V_1 & \text{and } u <_1 v) \text{ or} \\ (u, v \in V_2 & \text{and } u <_2 v) \end{cases} \tag{2.1}$$

(*parallel composition*)

and

$$u <_S v :\Leftrightarrow (u <_P v) \text{ or } (u \in V_1 \text{ and } v \in V_2) \tag{2.2}$$

(*series composition*)

We will use the notation $P_1 + P_2$ for the parallel composition and $P_1 * P_2$ for the series composition. The partial orders P_1 and P_2 are called the *parallel blocks* and *series blocks* of P_P and P_S, respectively. Other names for these composition operations are *direct sum* or *disjoint union* and *ordinal sum*, respectively.

Viewed in the Hasse diagram, the parallel composition of P_1 and P_2 is just the disjoint union of P_1 and P_2, while the series composition is obtained by putting P_2 "on top" of P_1.

The class of series–parallel partial orders is then defined recursively as follows: $P = (V, <)$ is *series–parallel* if

$$|V| = 1, \text{ i.e. } P \text{ is a one–element partial order} \tag{2.3}$$

or

$$\begin{cases} \text{P is obtained by series composition or parallel composition} \\ \text{of two (smaller) series–parallel partial orders.} \end{cases} \tag{2.4}$$

Hence series–parallel partial orders constitute the smallest class of partial orders that contains the one–element partial order and is closed under parallel and series composition.

As a direct consequence of the recursive definition, each series–parallel partial order can be represented in a natural way by a binary tree, the *binary decomposition tree*, as shown in Figure 1. The leaves of this tree represent one–element partial orders, and each internal node corresponds to a partial order obtained by series (label S) or parallel (label P) composition of the partial orders associated with the left and right son of the node.

This tree can also be interpreted as *parse tree* of an arithmetic expression with the operations $*$ (label S) and $+$ (label P) over V. This shows that series–parallel partial orders correspond to *arithmetic expressions with 2 operations*. For the partial order P from Figure 1, the expression is $1 * (2 + 3) * 4 + 5 * 6$.

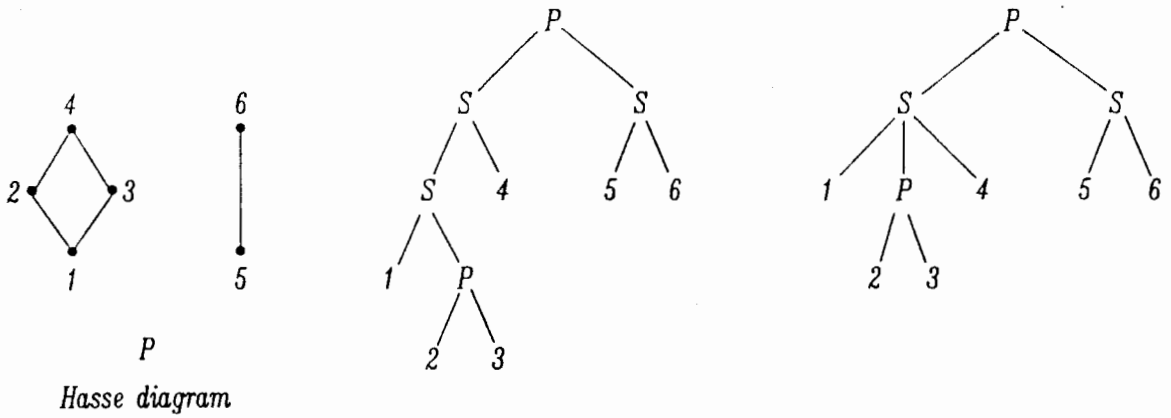


Figure 1: A series–parallel partial order P, its binary decomposition tree, and its (canonical) decomposition tree

There may be several non–isomorphic binary tree representations for the same partial order (e.g. for a linear order). This is due to the fact that a sequence of series or parallel composition can be grouped together in different ways.

A standardization leading to a unique tree representation, called the *(canonical) decomposition tree*, is obtained by taking all minimal series or parallel blocks occurring in a sequence of compositions of the same type as sons of the composition node in the tree (cf. Figure 1). The thus obtained tree is a special case of the decomposition tree for the substitution decomposition (cf. Section 4).

The (binary or canonical) tree representation is also the basic structure in most of the algorithms for problems on series–parallel partial orders, as will be demonstrated in Section 2.4.

In fact, the tree represents the structure of a series–parallel partial order $P = (V, <)$ with $|V| = n$ in $O(n)$ space. This follows from the following proposition and the simple fact that any (binary or canonical) tree with n leaves has at most $n - 1$ internal nodes.

2.1 Proposition: *Let $P = (V, <)$ be a series–parallel partial order with (binary or canonical) decomposition tree T. Then:*

(a) *$u \sim v$ ($u \parallel v$) iff the lowest anchestor of u and v in T has label S (label P).*

(b) *Any induced suborder $P \mid U$, where $U \subseteq V$, of P is series–parallel, and its decomposition tree is obtained from T by taking the subtree T' of T consisting of all paths of T from the root to a leaf $u \in U$, and by replacing in T' subpaths with all nodes of outdegree 1 by a single edge.*

Statement (a) follows from the fact that order relations $u < v$ can only be created by series composition. (b) is then a direct consequence of (a).

Note that (a) implies that testing whether $u < v$ in the tree has the same time–complexity as finding the *lowest common ancestor* of 2 leaves. This can be done in $O(1)$

time per test (on a random access machine model) with an additional $O(n)$ preprocessing time [HT], which hence means no disadvantage in access time over the representation by an adjacency matrix. Also (immediate) successors and predecessors can easily be found in the tree representation.

As a further example of the usefullness of the tree representation, we give a simple proof that series–parallel partial orders are 2–dimensional. This was first shown in [VTL] by constructing an embedding in $\mathbf{R}^2$ from the tree.

So let P be series–parallel and T be a (binary for simplicity) decomposition tree of P. Starting from the root, we traverse T twice in a depth first search. In the first search, we always visit the left son of a node before its right son, while in the second search, we change the order on the sons of P–nodes (nodes corresponding to a parallel composition). Then the two induced orders on the leaves are linear extensions $L_1 = (V, <_1), L_2 = (V, <_2)$ of $P = (V, <)$, and using Proposition 2.1(a), one obtains $u < v$ iff $u <_1 v$ and $u <_2 v$. Hence $\dim(P) \leq 2$.

If it shown in [DM] that, for any partial order P, $\dim(P) \leq 2$ iff there exists a partial order P^* on the same ground set V such that, for all $u, v \in V$,

$$(u <_P v \text{ or } v <_P u) \Leftrightarrow u \|_{P^*} v. \tag{2.5}$$

Such a partial order P^* is called a *conjugate* of P.

Obviously, if P is series–parallel, then Proposition 2.1(a) implies that a conjugate P^* is obtained by interchanging the labels "P" and "S" of the internal nodes of a tree representation of P. So we have:

2.2 Proposition: *Let P be series–parallel. Then $\dim(P) \leq 2$, and every conjugate P^* of P is also series–parallel.*

Moreover, P and P^* have isomorphic decomposition trees with labels "P" and "S" interchanged.

The idea underlying the definition of series–parallel partial orders is a common principle in many structural theories. There are two (usually natural) operations, often called (direct) sum and product, that are used to construct larger structures recursively from "atoms" (the singletons). Interchanging the operations leads to the definition of "dual" structures, which usually can be seen in a wider "duality" context. In discrete mathematics, examples can be found in matroid theory (series–parallel matroids, dual matroids), in the theory of Boolean functions (sum and product, dual functions), for clutters (sum and product, blocking clutter) and for relational systems; see [MR1] for more details.

For graphs and digraphs, some of these definitions are very close to the constructions of series–parallel partial orders.

For instance [Du,La2,VTL] define the class of *TTSP* (*Two–Terminal–Series–Parallel*) *digraphs* or *edge series–parallel* digraphs recursively as follows:

(2.6) The digraph $q\bullet \longrightarrow \bullet s$ with source q and sink s is $TTSP$.

(2.7) If G_1 and G_2 are $TTSP$, then so are the graphs obtained from G_1 and G_2 by

a) *parallel composition*: Identify by sources of G_1 and G_2 and the sinks of G_1 and G_2.

b) *series composition*: Identify the source of G_1 with the sink of G_2.

An alternative definition of series–parallel digraphs takes the vertex set (instead of the edge set as in $TTSP$–digraphs) as ground set of the structure to be defined. This leads to the following recursive definition of *minimal series–parallel (msp) digraphs* [VTL].

(2.8) The one–point digraph (without edges) is msp.

(2.9) If G_1 and G_2 are msp, then so are the graphs G obtained from G_1 and G_2 by

a) *parallel composition*: Take the disjoint union of G_1 and G_2.

b) *series composition*: Join every sink of G_1 and every source of G_2 by an edge.

A digraph G is called *series parallel* or *vertex series–parallel* (*vsp*), if its *transitive reduction* (obtained from G by deleting all edges (u,v) for which there is a directed path from u to v with more than one edge) is msp.

In other words, series–parallel digraphs are digraphs including a msp digraph plus possibly some (up to all) edges implied by transitivity (*transitive edges*).

These seemingly different classes of digraphs turn out to be just slightly different representations of series–parallel partial orders. This difference comes from the fact that a directed aycylic graph G (*dag* for short) induces a partial order in two different ways.

Let V be the set of vertices of G. Then

$$u <_1 v :\Leftrightarrow G \text{ contains a directed path from } u \text{ to } v. \tag{2.10}$$

defines a partial order $P_1 = (V, <_1)$ on V. Obviously, any partial order P can be obtained in this way, and the transitive reduction of a digraph G inducing P in the sense of (2.10) is called a *node diagram* of P. (The Hasse diagram is just a special drawing of the node diagram.)

Let E be the set of edges of G, then

$$a <_2 a' :\Leftrightarrow \left\{ \begin{array}{l} \text{G contains a directed path from the head } h_a \text{ of edge} \\ a \text{ to the tail } t_{a'} \text{ of edge } a' \ (h_a = t_{a'} \text{ is possible}) \end{array} \right. \tag{2.11}$$

defines a partial order $P_2 = (E, <_2)$ on E. A partial order obtained in this way is called *edge–induced* and the inducing graph G is called an *edge diagram* or *arc diagram* of P. [G is unique up to the number of sources, sinks and isolated vertices.]

Note that not all partial orders are edge–induced. We will see in Section 3 that the edge–induced partial orders are exactly the N–free partial orders.

It is clear from these definitions that the class of partial orders induced by $TTSP$–digraphs and series–parallel digraphs, respectively, is just the class of series–parallel partial orders. So from a structural viewpoint, all these classes are equivalent and differ only in the chosen representation of partial orders by edges or vertices of a digraph.

These differences have, however, led to different characterization theorems and recognition algorithms, cf. Sections 2.2 and 2.3.

The construction principle for series–parallel partial orders P carries over to their comparability graphs $G(P)$. Obviously, the parallel composition $P_1 + P_2$ corresponds to the disjoint sum $G(P_1) + G(P_2)$ of $G(P_1)$ and $G(P_2)$, whereas the series composition $P_1 * P_2$ corresponds to joining any two vertices v_1 of $G(P_1)$ and v_2 of $G(P_2)$ by an edge.

These two graph operations will also be denoted by " $+$ " and " $*$ ", respectively. They apply of course also to arbitrary graphs.

Obviously, the class of graphs obtained from the one–vertex graph by these operations gives just the class of comparability graphs of series–parallel partial orders.

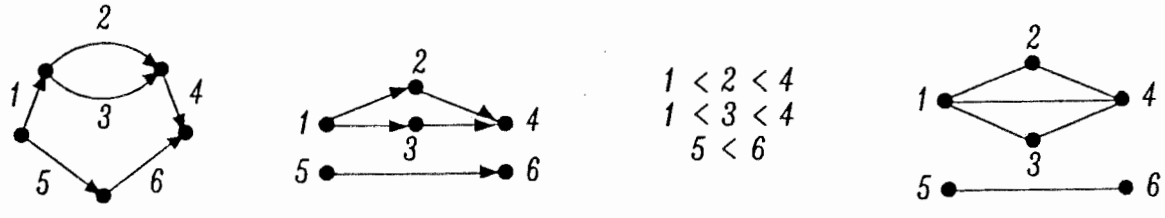
The operations " $+$ " and " $*$ " may equivalently be expressed by " $+$ " and graph complementation "c", where G^c denotes the *complementary graph* of G. This follows from

$$G_1 * G_2 = [G_1^c + G_2^c]^c. \tag{2.12}$$

The class of graphs constructed from the one–vertex graph recursively by disjoint union " $+$ " and complementation "c" has been studied (largely independently) under the name *complement reducible graphs* or *cographs*, cf. [CLS] for an overview. From the above remarks, we have:

2.3 Proposition: *The class of cographs and the class of comparability graphs of series–parallel partial orders are identical.*

An example of the structures defined so far is given in Figure 2.



a TTSP–digraph G | an equivalent series–parallel digraph | the associated series–parallel partial order | the associated cograph

Figure 2: Related series–parallel structures of the series–parallel partial order from Figure 1

The recursive definition of cographs by means of " $+$ " and "c" leads to a related tree

representation, the so–called *cotree* [CPS].

Instead of 2 operations at internal nodes, it uses only the operation *"complemented sum"* $G = (G_1 + \ldots + G_k)^c$ of the graphs $G_1, \ldots, G_k$ associated with the sons of the considered internal tree node (where $k = 1$, i.e. simple complementation, is included). It follows from (2.12) and

$$G_1 + G_2 = [G_1 + G_2]^{cc} \tag{2.13}$$

that every cograph can be represented in such a way, though several different representations are possible.

The cotree T refers to a unique such representation in which each internal vertex different from the root has at least two sons. The internal nodes of T may then be labeled by "0" and "1" such that the root obtains label "1" and that labels alternate along every path from the root. An example is given in Figure 3.

The labels "0" and "1" express the parity of the number of complementations along a path from a leaf to the root. In particular, label "0" then means that vertices of the cograph G belonging to distinct subtrees are in different components of G and thus not joined by an edge. This implies that the root has only one son iff G is disconnected. Taking into account Proposition 2.1*a*), one obtains the following relationship to the canonical decomposition tree of series–parallel partial orders.

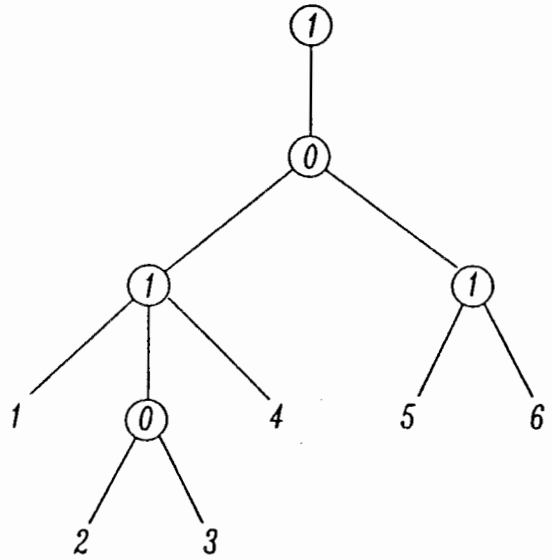


Figure 3: The cotree of the cograph $G(P)$ of P from Figure 1

2.4 Proposition: *Let G be a cograph and P be an associated partial order, i.e. $G = G(P)$. Then*

a) *u, v are joined by an edge in G (i.e. comparable in P) iff the lowest common acestor of u and v in the cotree has label* 1.

b) The cotree T of G (minus the root of the root has only one son) is isomorphic to the canonical decomposition tree of P. Moreover, labels "O" correspond to labels "P" and labels "1" to labels "S".

Cotrees will be used in Section 2.3 as the basic structure for fast and yet relatively simple recognition algorithms for series–parallel partial orders.

There is another, related class of undirected graphs, called *series–parallel* graphs, see e.g. [Sy3, SKOM, TNS]. These are undirected graphs with multiple edges constructed recursively from the one–point graph with a loop by subdividing an edge by a new vertex (*series operation*) and by creating a parallel edge for a non–loop edge (*parallel operation*).

Let G be a series–parallel in this sense. Then deleting any edge $(a,b), a \neq b$, and directing all other edges from a to b along the (unique!) path from a to b gives a TTSP–digraph with source a and sink b. Conversely, every TTSP–digraph gives a series–parallel graph by joining its source and sink by an edge and making all edges undirected.

So series–parallel graphs are closely related to TTSP–digraphs, and thus also to series–parallel partial orders.

2.2 Structural Properties

Series–parallel partial orders and the related structures defined in Section 2.1 have many different equivalent characterizations by structural properties. The most interesting of these are perhaps the characterizations by forbidden substructures, which have been (re)discovered many times with slight variations depending on the type of structure considered (i.e. partial order, $TTSP$–digraph, digraph, or graph).

The earliest of these characterizations is due to Duffin [1965] and considers the successive reduction of an undirected graph G by two operations:

(2.14) delete one of 2 parallel edges *(parallel reduction)*

(2.15) { If u has only two neighbors a, b, delete u and join a, b by an edge *(series reduction)*

Then the theorem of Duffin states that G is series–parallel iff no sequence of parallel and series reductions leads to a K_4 (the complete graph with 4 vertices).

Because of the relationship between series–parallel graphs and TTSP–digraphs, it follows that a digraph (with exactly one source and one sink) is TTSP iff it cannot be reduced to the "*Wheatstone bridge*" of Figure 4.

In the case of dags, it was shown in [La2, VTL] that a dag is series–parallel iff its transitive closure does not contain an induced subgraph isomorphic to the "N" in Figure

4. The equivalent statement for partial orders appeared independently in [KM], see also [Ka2].

Finally, for cographs, it is shown in [CLS], partially based on previous results of [Ju] and [Se], that a graph is a cograph iff it does not contain an induced subgraph isomorphic to P_4, the path with 4 vertices. The earliest reference for this result seems to be the unpublished manuscript [Fo].

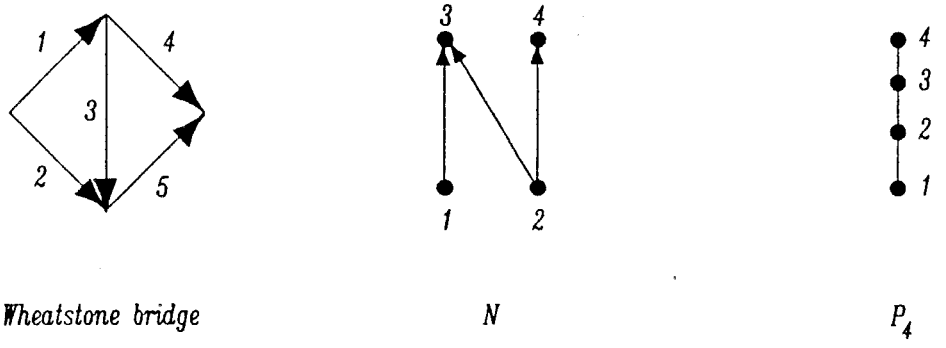


Figure 4: Forbidden subconfigurations for series–parallel structures

All these characterizations turn out to be equivalent [VTL, CLS, Mö2]. A very elegant way to these results may be based on an elementary lemma due to Seinsche [Se].

2.5 Lemma: *If G is a finite graph and both G and G^c are connected, then G contains an induced P_4.*

In the infinite case, one needs a stronger assumption for the same result, viz. that G contains no proper autonomous set (i.e. is indecomposable with respect to the substitution decomposition; cf. Section 4 for these notions). An elegant proof in this case is given in [Ke, p.23–24].

The rest of the characterization is now very easy. First it follows from Proposition 2.3 (cographs are the comparability graphs of series–parallel partial orders) and Proposition 2.1 b) (being series–parallel is hereditary) that any induced subgraph of a cograph is again a cograph. Since P_4 is not a cograph, no cograph contains a P_4 as induced subgraph.

Conversely, if G is not a cograph then the recursive decomposition by disjoint union and complementation must fail at a certain stage, because some induced subgraph H of G or G^c has been obtained for which both H and H^c are connected. Hence H contains a P_4 and, since $P_4^c = P_4$, also G. This gives:

2.6 Theorem: *G is a cograph iff G contains no P_4 as induced subgraph.*

The transitive closure of the digraph N in Figure 4 is a partial order that is also denoted by N. Obviousely, N and its dual $N^{-1} \cong N$ are the only partial orders P with P_4 as comparability graph. Hence Theorem 2.6 and Proposition 2.3 give:

2.7 Corollary: *a) A partial order is series–parallel iff it contains no induced N.*
b) A dag is series–parallel iff its transitive closure contains no induced N.

The characterization by Duffin via the embedded Wheatstone bridge will follow in Section 3 as a characterization of series–parallel partial orders within the class of N–free partial orders (Theorem 3.5).

There are several other equivalent characterizations of series–parallel partial orders which are sometimes most elegantly formulated in terms of their comparability graphs, i.e. via cographs. Some of these are formulated in the next theorem. For more details we refer to [CLS].

We need the following notation. A partial order $P = (V, <)$ is called a *multitree* [Ju] if, for all $u, v \in V, u < v$ or $a < b$ for every $a \in Suc(u) - Suc(v)$ and every $b \in Suc(u) \cap Suc(v)$. Two vertices $u, v \in V$ are called *twins*, if $Pred(u) - \{v\} = Pred(v) - \{u\}$ and $Suc(u) - \{v\} = Suc(v) - \{u\}$. Finally, P has the *CAC–property* (chain–antichain–complete [Gr, LM]) if every maximal chain of P meets every maximal antichain of P.

2.8 Theorem: *Given a partial order P, the following statements are equivalent:*

(1) *P is series–parallel.*

(2) *Any non–trivial induced suborder contains a pair of twins.*

(3) *Any induced suborder has the CAC–property.*

(4) *The complement of every non–trivial connected induced subgraph of $G(P)$ is disconnected.*

(5) *Every connected induced subgraph of $G(P)$ has path length ≤ 2 between any two vertices.*

(6) *P is a multitree.*

Note that twins correspond exactly to parallel– and series–reducible edges in (2.14) and (2.15). So taking the directed versions of (2.14) and (2.15), one obtains [Du]:

2.9 Corollary: *P is series–parallel iff it can be reduced to the one–point partial order by a sequence of directed series and parallel reductions.*

2.3 Recognition Algorithms

Because of the given characterization by forbidden sub–configurations, the recognition of series–parallel partial orders P and cographs G is polynomial. (Check all 4–element induced suborders whether they are isomorphic to N.) This algorithm provides a proof if P is not series–parallel (exhibiting the N), but does not give any information in the positive

case. A natural requirement would be to construct the associated decomposition tree or cotree in the positive case, and to exibit a forbidden substructure in the negative case.

There exist different algorithms with these properties, the fastest of which require only linear time, i.e. $O(n+m)$, where $n = |V|$ and m denotes the number of comparable pairs [VTL, CLS]. Before dealing with these more sophisticated algorithms, we will discuss a very simple $O(n^3)$ algorithm of this type.

2.10 Algorithm: *(Recognition of series-parallel partial orders and cographs):*
Let $D = (V, E)$ be a graph or a partial order. Put $G = D$ if G is a graph and $G = G(D)$ if D is a partial order.

Step 1: Put $N := V$ (the root node N of a tree to be constructed)

Step 2: Compute the connected components of the induced subgraph of $G \mid N$. If $G \mid N$ is disconnected, then label node N with "P", assign the connected components of $G \mid N$ as sons to N, and go to Step 5.

Step 3: Compute the connected components of $(G \mid N)^c$. If $(G \mid N)^c$ is disconnected, then label node N with "S", assign the connected components of $(G \mid N)^c$ as sons to N and go to Step 5.

Step 4: ($G \mid N$ and $(G \mid N)^c$ are connected). Stop. The graph G is not a cograph and $G \mid N$ is a forbidden subgraph in the sense of Theorem 2.8(4).

Step 5: If there is an unlabeled node N' with $|N'| \geq 2$, put $N := N'$ and return to Step 2. Otherwise stop; the constructed tree is the canoncial decomposition tree T of G.

Correctness of this algorithm follows from Proposition 2.3 and the correspondence between the graph operations "$+$" and "$*$" and the parallel and series composition. So the constructed tree T is the canonical decomposition tree if D is a partial order. If D is a graph, its cotree is isomorphic to T and can be obtained from T by the rules specified in Proposition 2.4 b) in $O(n)$ time.

Furthermore, since T contains at most $|V|-1$ inner nodes, Steps 2,3 and 5 are executed at most $|V|-1$ times. At each time, the connected components of $G \mid N$ and $(G \mid N)^c$ are computed, which requires $O(|N|^2)$ time together. So we obtain:

2.11 Proposition: *Algorithm 2.10 recognizes a series-parallel partial order or a cograph with n vertices in $O(n^3)$ time.*

A much faster recognition algorithm that is still relatively simple has been obtained in [CPS]. It also works with cographs and either constructs the cotree of a given graph or exhibits an embedded P_4 (instead of just some subgraph H for which H and H^c are connected as Algorithm 2.10 does).

Its increased efficiency is obtained by working *incremental* or *on-line* (instead of top-down). This means that vertices are processed one at a time in arbitrary order and that the cotree is grown as the vertices are being processed. More specifically, if G is a cograph

with cotree T then the algorithm determines from T whether $G+v$ (G plus a new vertex v with arbitrary adjacencies to vertices of G) is a cograph and, if so, modifies T to represent $G+v$.

The algorithm uses a new characterization of cographs that is based on the adjacency information about v with respect to the tree T. This information is obtained by a marking procedure that initially marks all leaves of T that (as elements of V) are adjacent to the new vertex v, and perculates this information up the tree to obtain a final marking of the inner nodes of the tree T.

This marking can be produced as follows. The marking of a node is chosen from the set $\{O, *, \infty\}$. Initially, all leaves $u \in V$ are marked with "∞" if u is adjacent to the new vertex v, and by "O" if it is not. The marking of an inner node is the "sum" of the markings of it sons according to the following rules

$$\begin{cases} \infty + \infty = \infty, & \infty + * = *, \quad \infty + O = * \\ * + * = *, & * + O = *, \quad O + O = O \end{cases} \tag{2.16}$$

where all sums are commutative and associative.

So in the final marking, a node N is marked with "∞" iff all leaves below it are adjacent to v, with "O" iff there are no leaves below it that are adjacent to v, and with "$*$" iff there are leaves below it that are adjacent to v and others that are not.

An example of this marking is given in Figure 5 for the cotree of the partial order P from Figure 1. The first marking corresponds to adding vertex v_1 to $G(P)$ with $Adj(v_1) = \{1,4\}$, while the second corresponds to adding v_2 with $Adj(v_2) = \{4,6\}$.

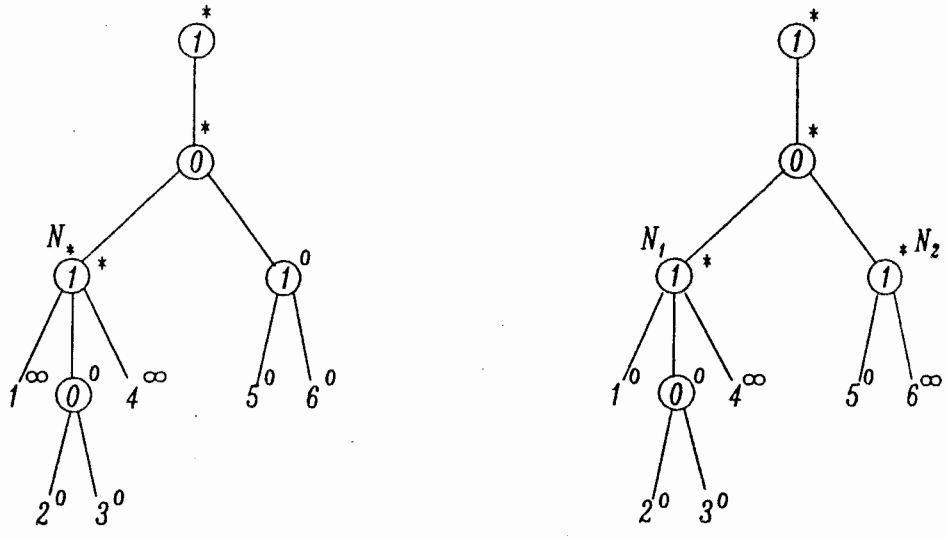


Figure 5: Two markings of the cotree of $G(P)$ of P from Figure 1

The algorithmic characterization of cographs derived in [CPS] is based on the set M of all nodes marked with "$*$".

2.12 Theorem: *If G is a cograph with cotree T then $G+v$ is a cograph iff*

1. *M is empty or*
2. *M forms a path P in T from the root to some node N_* such that*

   (i) if $N \in P - \{N_*\}$ has label "1" then all sons of N not on P are marked with "∞"

   (ii) if $N \in P - \{N_*\}$ has label "0" then all sons of N not on P are marked with "0".

In other words, the new vertex v must have rather regular adjacencies in T in order to make $G + v$ again a cograph.

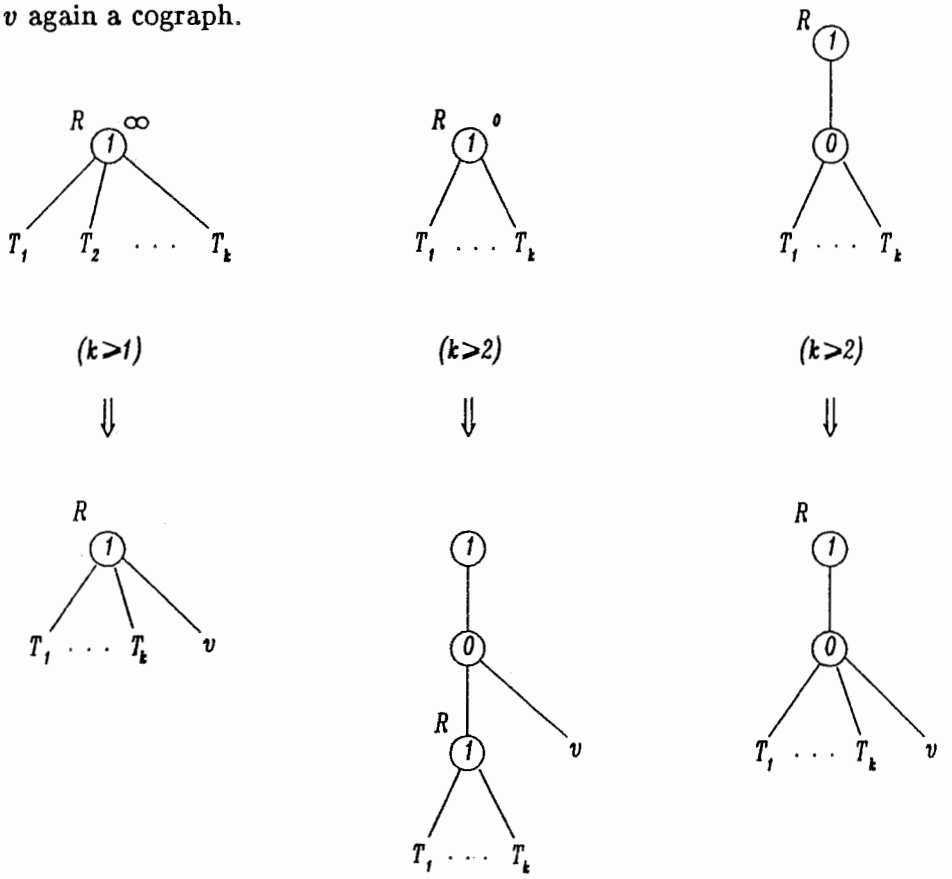


Figure 6: Tree modifications rules for case 1 of Theorem 2.12

It is not too difficult to see that $G + v$ contains a P_4 if the conditions of the theorem are not satisfied. For instance, the marking for adding v_2 in Figure 5 violates condition 2 since M is not a path. Then there are two incomparable lowest nodes N_1, N_2 in M (see Figure 5) which thus have distinct leaves a_1, b_1 and a_2, b_2 below themselves such that a_1, a_2 are adjacent to v and b_1, b_2 are not. Let N be the lowest common ancestor of N_1 and N_2. Then there are different cases depending on the labels of N, N_1, N_2. If, as is the case in the example, N has label "0", and N_1 and N_2 have label "1", then both $b_1 - a_1 - v - a_2$ and $a_1 - v - a_2 - b_2$ form a P_4. The other cases follow by similar but more tedious arguments.

The other direction of the theorem is proved by constructing the cotree of $G + v_2$ from the given cotree by local modifications that depend on the different cases in the conditions of Theorem 2.12.

Case 1 is easy since the new vertex v is either adjacent to all vertices of G (root is marked with "∞") or to none (root is marked with "O"). For instance, in the first case, v is just added as a new son to the root (which, by definition of a cotree, has label "1"). The set of all modification rules is illustrated in Figure 6. There, R denotes the root, $T_1, \ldots, T_k$ denote subtrees and v denotes the vertex to be added.

In case 2 of Theorem 2.12., the modification rules modify only the subtree with root N_*. These rules are illustrated in Figure 7 for the case that N_* has label "O". The rules depend on the markings of the subtrees of N_* (which, by definition of N_*, are either "O" or "∞"). The subtrees marked with "O" are denoted by $S_1, \ldots, S_l$, while those marked with "∞" are denoted by $T_1, \ldots, T_k$. Since N_* is marked with "$*$", there are subtrees of both kinds. If $l = 1$ or $k = 1$ and S_l or T_k is a leaf, it will be denoted by s or t, respectively.

The rules when N_* has label "1" are completely symmetric to those given in Figure 7, only with interchanged roles of subtrees T_i and S_i.

It is clear that these rules represent the adjacencies of v with respect to the leaves below N_* correctly. Correctness for the other leaves then follows from the conditions on the path P in Theorem 2.12.

In Figure 5, the first marking is correct and produces the cotree of $G + v_1$ by applying the analogue of rule b) from Figure 7.

The computational complexity of this algorithms depends on how the marking is done and how the tree rearrangements are carried out. In these rearrangements, the only problem is the grouping of the subtrees $S_1, \ldots, S_l$ since these are not adjacent to v and thus may blow up the wanted time complexity of $O(|Adj(v)|)$ for the insertion of v. It is shown in [CPS] that these S_j can be arranged into a linked list while the marking is done (by removing the T_i from an initial list consisting of all S_j and T_i). For these and further implementation details, we refer to [CPS]. In fact, the marking process, and thus also the conditions in Theorem 2.10 are expressed differently in [CPS] in order to carry out the marking efficiently. They are equivalent to the presentation here (also in computational respect if the marking process of [CPS] is adapted). So we have:

2.13 Theorem: *Given a graph $G = (V, E)$, the incremental tree modification algorithm checks in $O(|V| + |E|)$ time whether G is a cograph, and produces either an embedded P_4 or the cotree of G.*

As a consequence, also the one–line recognition of series–parallel partial orders and the construction of the canonical decomposition tree is possible in $O(|V| + |E|)$, where E is the set of all comparable pairs of P.

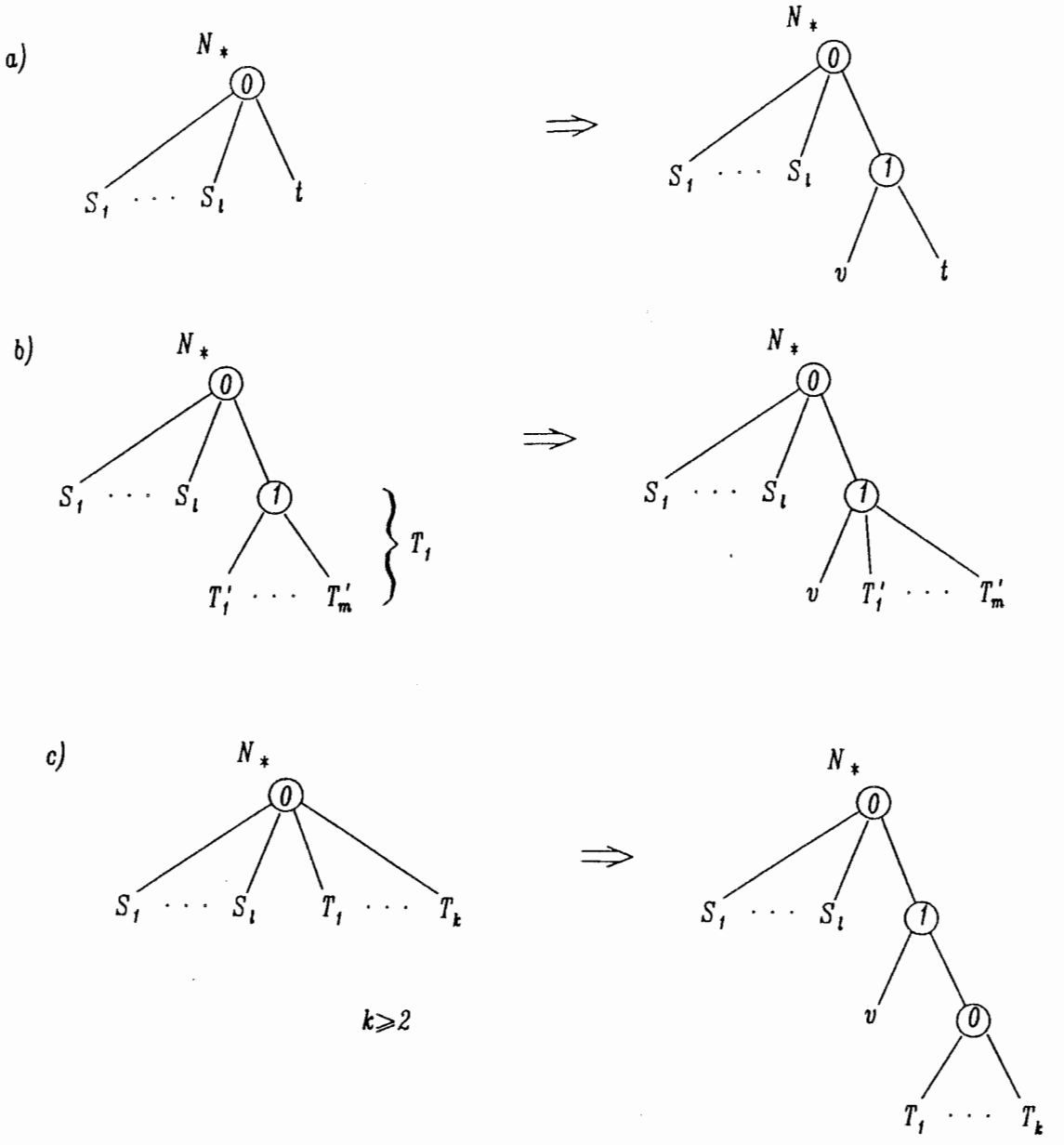


Figure 7: Tree modification rules for case 2 and "O"–node N_*

We close this section with some remarks on the recognition algorithm developed in [VTL]. It works with digraphs instead of graphs and tests whether a digraph $G = (V, E)$ is series–parallel and, if so, constructs a binary decomposition tree of G.

It also works in $O(|V| + |E|)$ time, but has the advantage above the cograph algorithm that it accepts as input *any* digraph $G = (V, E)$, i.e. not necessarily transitively closed ones. So for testing a partial order P for being series–parallel, any digraph G whose transitive closure is P will lead to the same result. Hence, this algorithm would also work on the Hasse diagram of P without constructing the transitive closure (which is unlikely to be possible

in linear time [AHU]) as the cograph algorithm must do.

The details of the algorithm are as follows [VTL]:

Step 1: (*Compute the pseudo-transitive reduction of* G) Given $G = (V, E)$, E is partitioned into E_T and E_M such that, if G is series–parallel, then $G_M = (V, E_M)$ is the transitive reduction of G. (If G is not series–parallel, G_M need not be the transitive reduction of G, but may still be series–parallel). E_M is computed via height differences. If $h(u)$ denotes the *height* of u in G (i.e. the lenght of the longest path in G from a source to u), then

$$E_M = \{(u,v) \in E \mid h(v) - h(u) = \min_{(u,x)\in E}[h(x) - h(u)]\}. \tag{2.17}$$

Obviously, all edges redundant under transitive closure are then contained in $E - E_M$, but in general dags, there may be non–redundant edges (u, v) not fulfilling the condition in (2.17).

Step 2: (*Compute an edge diagram of* G_M) If G is series–parallel and so, by Step 1, G_M is its transitive reduction, then there is a TTSP–digraph $\overline{G_M}$ equivalent to G_M (see Figure 2 and the definitions of $TTSP$ and series–parallel digraph). $\overline{G_M}$ is an edge diagram of the partial order P_M induced by G_M in the sense of (2.11). $\overline{G_M}$ can be obtained in linear time by any fast algorithm for recognizing whether P_M is N–free, e.g. by Algorithm 3.6 in Section 3. So if this algorithm detects that the partial order P_M represented by G_M does not have an edge diagram (i.e. is not N–free), then G is not series–parallel and we can stop at this step.

Step 3: (*Test whether* $\overline{G_M}$ *is* $TTSP$) This can be done by finding a sequence of parallel and series reductions (in the sense of (2.14), (2.15) and Corollary 2.9) that reduce $\overline{G_M}$ to the one–edge graph. Such a reduction can, if it exists, be obtained in linear time [VTL, Section 3.3.] The sequence of reductions also determines an associated binary decomposition tree T. [In fact, this sequence of reductions can be obtained in such a way that it corresponds to a postorder traversal of T and thus defines T uniquely]. So Step 3 either produces a binary decomposition tree of $\overline{G_M}$, and thus also of G_M or detects that $\overline{G_M}$ is not $TTSP$ and thus G is not series–parallel and the algorithm stops.

Step 4: (*Test whether* G_M *is the transitive reduction of* G) As a result of Step 3, $\overline{G_M}$ is $TTSP$ and G_M is series–parallel. Hence the partial order P_M induced by G_M is also series–parallel and thus 2–dimensional (Proposition 2.2). Then the binary decomposition tree of Step 3 is used as in the proof of Proposition 2.2 to compute two linear extensions L_1, L_2 of P_M whose intersection is P_M. $L_1 = (V, <_1)$ and $L_2 = (V, <_2)$ are then used to test the edges $(u, v) \in E - E_M$ for redundancy. Such an edge (u, v) is redundant iff $u <_1 v$ and $u <_2 v$. [Otherwise $u \parallel_{P_M} v$ and thus G_M is not the transitive reduction of G.] Each such query can be tested in constant time by associating with each $v \in V$ its positions in L_1 and L_2.

This can be done while the L_i are constructed and corresponds essentially to the specification of coordinates $(l_1(v), l_2(v))_{v \in V}$ of an embedding of P_M into $\mathbb{R}^2$. If all edges in $E - E_M$ are redundant, then G is series–parallel and the algorithm outputs the binary decomposition tree T and stops. Otherwise it answers "NO" and stops.

The different steps show that the algorithm exploits many properties of series–parallel digraphs and the induced partial orders in a very sophisticated way. Step 1 uses a property of the *transitive reduction* of series–parallel digraphs, Step 2 uses the existence of an edge diagram (i.e. the property of *being N–free*, cf. Section 3), Step 3 uses the computationally simple series and parallel reductions in the arc diagram (i.e. the existence of *twins* and the *hereditary property*), and, finally, Step 4 uses the *two–dimensionality*.

As a side–effect, the algorithm also computes in linear time the *transitive reduction* (G_M) and the *transitive closure* (in the form of the realizing linear extensions L_1, L_2) of any series–parallel digraph G.

2.4 Some Applications

In this section we will present some selected applications of series–parallel partial orders with main emphasis on sorting, sequencing and scheduling.

The special structural properties of series–parallel partial orders and (di–)graphs have led to fast solution methods for nearly all problems that are known to be intractable on larger classes of partial orders or graphs, cf. for instance [CLS] for co-graphs, [La2] for series–parallel digraphs and [TNS] for series–parallel graphs.

Sequencing problems usually involve the construction of special linear extensions of a given partial order. For a series–parallel partial order, all its linear extensions can be constructed in a natural fashion along the (canonical or binary) decomposition tree.

For a series composition $P = P_1 * P_2$, any linear extension L of P is obtained by a series composition $L_1 * L_2$ of linear extensions L_i of P_i, while for a parallel composition $P = P_1 + P_2$, any linear extension L of P is obtained by merging linear extensions L_i of P_i in all possible ways.

This leads e.g. to easy counting formulas for the number $l(P)$ of linear extensions of P, while for arbitrary partial orders, the complexity status of this counting problem is still open (but conjectured to be $\#P$–complete [Li]). These counting formulas are based on

$$l(P_1 * P_2) = l(P_1) \cdot l(P_2) \tag{2.18}$$

and

$$l(P_1 + P_2) = \frac{(n_1 + n_2)!}{n_1! \cdot n_2!} l(P_1) \cdot l(P_2), \tag{2.19}$$

where $n_i = |V_i|, i = 1,2$. So given the decomposition tree of P, $l(P)$ may be computed in a bottom up fashion in the tree by assigning 1 to the leaves and applying (2.18) and (2.19) to internal nodes representing series or parallel compositions, respectively. These counting arguments can be generalized to obtain formulas for all coefficients e_i of the order polynomial [FS2]. The coefficient e_i denotes the number of surjective order preserving maps of P into an i–element chain. So in particular, $e_{|P|}$ is the number of linear extensions of P, and e_2 is the number of ideals of P (minus 2). The determination of $e_2, \ldots, e_{|P|-1}$ is known to be $\#P$–complete for arbitrary partial orders [PB, FS2].

Also constructing special linear extensions (such as minimizing the number of jumps [CHab] or bumps [St6], or the $\sum_i w_i C_i$ one–machine scheduling problem [Si,La3]) has turned out to be easy for series–parallel partial orders. The usual way to solve these problems consists in constructing an optimal linear extension along the decomposition tree. Since there is no choice for a series composition, the only problem is to find the right rule for a parallel composition, i.e. for an *optimal merging of two parallel chains.* For instance, for the $\sum_i w_i C_i$ scheduling problem, the rule consists in merging the chains according to a "mergesort" rule that prefers jobs v_i with smaller x_i/w_i value, where x_i denotes the processing time of v_i. This merging rule is combined with a *contraction rule* that is applied after each series composition. It repeatedly contracts adjacent jobs v_i, v_j with $v_i < v_j$ and $x_i/w_i > x_j/w_j$ to form a new job $\overline{v_l}$ with $\overline{x_l} = x_i + x_j$ and $\overline{w_l} = w_i + w_j$. For details, we refer to [La3].

This construction process along the decomposition tree generalizes to partial orders decomposable by substitution decomposition and leads to still polynomial algorithms for partial orders with bounded decomposition diameter and bounded decomposition width (cf. Section 4).

Searching and sorting problems for series–parallel partial orders are treated in [FLRT]. They first give recursive formulas for determining the number N of all ideals and the number $N(u)$ of all ideals containing u. These formulas are closely related to a compact labeling scheme for series–parallel partial orders [St2] that can also be used for dynamic programming solutions to several scheduling problems, cf. also Section 5.3.

These formulas imply that a "good" central element can be found by calculating $N(u)/N$ for each u and by choosing an u for which this value is close to $1/2$. The constant δ_o is shown to be $1/4$, i.e. each series–parallel partial order contains a point u with $1/4 \leq N(u)/N \leq 3/4$. This bound is sharp.

The *isomorphism problem* for series–parallel partial orders can be solved in polynomial time [La2] by applying polynomial–time isomorphism algorithms for labelled trees [AHU] to the canonical decomposition tree. Similar methods for cographs are presented in [CLS]. For series–parallel digraphs, however, the isomorphism problem is isomorphism complete [Va]. This is due to the fact that arbitrary graphs can be encoded into the redundant edges of a series–parallel digraph.

We close with some problems that are NP–hard even on series–parallel partial orders or digraphs. These are the *subgraph isomorphism problem* [VTL], the *unit–time machine scheduling problem* $P\,|prec, x_j = 1|\,C_{max}$ (which is already NP–complete on parallel compositions of trees, i.e. on *opposing forests* [GJTY, Ma], and the *quadratic assignment problem* [Re].

2.5 Some Subclasses

In this section we consider two subclasses of series–parallel partial orders, viz. weak orders and threshold orders. They arise in certain scheduling and sequencing problems and are interesting in their own right.

A *weak order* is a partial order P whose incomparability relation is transitive, i.e. $a \,||_P\, b$ and $b \,||_P\, c$ implies $a \,||_P\, c$. These orders arise naturally as generalization of linear orders in utility and measurement theory, see e.g. [Bo]. From these definition, one immediately obtains

2.14 Proposition: *Weak orders are exactly the partial orders obtained by series–compositions of antichains, i.e. every weak order is of the form*

$$P = A_1 * A_2 * \ldots * A_k, \tag{2.20}$$

where each A_i is an antichain.

Now consider the m–machine unit–time scheduling problem $m \mid prec, x_j = 1 \mid C_{max}$. This problem is polynomially solvable for $m = 2$ but its complexity status is open for any fixed $m \geq 3$. In fact, it is one of the few still unsolved problems on the list of open complexity problems in the book of Garey and Johnson [GJ].

In order to show tractability of this problem, a weaker approach is to show that it is in $\mathcal{NP} \cap co\mathcal{NP}$. Since it is very unlikely that $\mathcal{NP} \cap co\mathcal{NP} \neq \mathcal{P}$, such a result would be a strong indication for the existence of a polynomial algorithm. Loosely speaking, membership in $\mathcal{NP} \cap co\mathcal{NP}$ means that one should be able to convince somebody in polynomial time that $C^m_{max}(P) \leq k$ and $C^m_{max}(P) \geq k$, where $C^m_{max}(P)$ denotes the optimal schedule length of an m–machine schedule. The former can be done by specifying a 2–machine schedule for P with length at most k. The latter can be done by specifying a suborder P' (i.e. with less relations) for which it can be shown in polynomial time that $C^m_{max}(P') \geq k$.

More precisely, let P and P' be partial orders on the same ground set V. Then P' is called a *suborder* of P and P is called an *extension* of P' (denoted by $P' \subseteq P$) if

$$u <_{P'} v \Rightarrow u <_P v \text{ for all } u, v \in V. \tag{2.21}$$

It is clear that $C^m_{max}(P') \leq C^m_{max}(P)$ if $P' \subseteq P$. A partial order P is called *m–critical*

(or critical with respect to the $m|prec, x_j = 1|C_{max}$ problem) if $C^m_{max}(P') < C^m_{max}(P)$ for every proper suborder P' of P.

It follows that, in order to show membership of the m–machine problem in $co\mathcal{NP}$, it suffices to give a polynomial characterization of the m–critical partial orders. For $m = 2$ this problem has been solved in [Ra4], cf. also [BMR1] for a simpler proof, and [CG,Gab] for implicit characterizations.

2.15 Theorem: *A partial order P is 2–critical iff it is the parallel composition $P = P' + \{u_1\} + \ldots + \{u_l\}$ of a weak order $P' = A_1 * \ldots * A_k$ in which each $|A_i|$ is odd and $0 \leq l \leq k-2$ singletons u_i.*

Obviously, the schedule length for such a partial order P is

$$C^2_{max}(P) = \sum_{i=1}^{k} \lceil |A_i| / 2 \rceil, \tag{2.22}$$

and any deletion of a covering pair (a,b) in P leads to a partial order $P-(a,b)$ with $C^2_{max}(P-(a,b)) < C^2_{max}(P)$. So every such partial order is 2–critical. The converse direction can be shown by examining the points in an optimal schedule at which idle times can occur. These points essentially define the decomposition into the A_i, cf [Ra4] or [BMR1] for details.

The characterization of the 3–critical partial orders is still open. Preliminary results are presented in [BMR1]. So far, it seems that they need not be series–parallel in general, but that their structure is still close to that of series–parallel partial orders.

A related result holds also for the bump number problem [HMS]. Since the bump number $b(P)$ is monotonic with respect to extensions, i.e.

$$P' \subseteq P \Rightarrow b(P') \leq b(P), \tag{2.23}$$

one can define a *bump critical* partial order as a partial order P such that $b(P') < b(P)$ for all proper suborders $P' \subseteq P$. Then:

2.16 Theorem: *A partial order P is bump critical iff it is the parallel composition $P = P' + \{u_1\} + \ldots + \{u_l\}$ of a weak order $P' = A_1 * \ldots * A_k$ and $0 \leq l \leq k-2$ singletons $u_1, \ldots, u_l$.*

In that case, $b(P) = k - 1 - l$.

It is clear that these partial orders are bump critical. The converse is shown in [HMS] by a polynomial algorithm that constructs for any partial order P at the same time a linear extension L of P and a suborder $P' = (A_1 * \ldots * A_k) + \{u_1\} + \ldots + \{u_l\}$ of P with antichains A_i such that $b(P') = b(L,P)$. $P' \subseteq P \subseteq L$ then implies $b(P') \leq b(P) \leq b(L,P)$. Hence L is an optimal linear extension and P contains a suborder of the special kind described in Theorem 2.16. Thus every bump critical partial order must be of this form.

Note that the approach by critical suborders implicitly leads to a *duality theorem* of the form

$$\min_{(P \subseteq P^*)} u(P^*) = \max_{(P_* \subseteq P)} \lambda(P_*), \tag{2.24}$$

where P^* denotes certain extensions of P that define feasible solutions (e.g. a linear extension for the bump number problem and an extension of width $\leq m$ for the m–machine problem), u denotes a derived parameter of P^* (e.g. number of bumps in the bump number problems and height minus 1 = schedule length in the m–machine problem), and where P_* denotes critical suborders with objective value $\lambda(P_*)$ (e.g. bump number or schedule length).

This duality theorem implies an $\mathcal{NP} \cap co\mathcal{NP}$ characterization if there is a "good" characterizaton for the critical partial orders that leads to polynomial recognition algorithms, and it implies membership of $\mathcal{P}$ if a critical suborder giving the maximum value $\lambda(P^*)$ can be constructed in polynomial time.

The other class of special series–parallel partial orders arising in scheduling theory is the class of threshold orders. A *threshold order* is a partial order $P = (V, <)$ with $V = \{v_1, \ldots, v_n\}$ such that there are weights $w_1, \ldots, w_n$ and a *threshold* t with

$$C \subseteq V \text{ is a maximal chain of } P \Leftrightarrow \sum_{v_i \in C} w_i \leq t. \tag{2.25}$$

Since chains of a partial order corrrespond to cliques in its comparability graph, it follows that the property of being a threshold graph is a comparability invariant. In fact, the comparability graphs of threshold orders are exactly the *threshold graphs*, which have been studied extensively in graph theory, cf. [Go1] for an overview. A characterization by forbidden subgraphs is given in [CHam]. It is equivalent to statement (3) in Theorem 2.17 below that was obtained in characterizing so–called "singular solutions" of scheduling problems, see [Mö1] for details of this application.

2.17 Theorem: *The following statement are equivalent:*

(1) P is a threshold order.

(2) P does not contain any of the partial orders of Figure 8 as induced suborder.

(3) P is series–parallel and each node of its canonical decomposition tree has at most one son that is not a leaf.

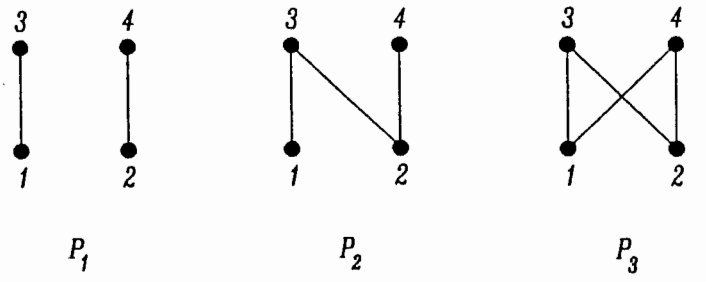


Figure 8: The partial orders of Theorem 2.17

An example of a threshold order P and its decomposition tree T are given in Figure 9. The weights w_i and the threshold t of (2.25) can be constructed along the decomposition tree as follows:

Let N_o be the (unique because of statement (3)) inner node of greatest depth in T. Let $w_i = 1$ for all v_i below N_o, and let t be the number of sons of N_o if N_o is a series node and 1 if N_o is a parallel node.

Adjust the definition of the w_i and t going up the tree as follows. Let N be a node whose (only) interior son N' has already been processed. If N is a parallel node, then put

$$w_i := \begin{cases} w_i & \text{if } v_i \text{ is below } N' \\ t & \text{otherwise} \end{cases}, \text{and } t := t. \tag{2.26}$$

If N is a series node, then let m be the number of sons of N and put

$$w_i := \begin{cases} m \cdot w_i & \text{if } v_i \text{ is below } N' \\ 1 & \text{otherwise;} \end{cases}, \text{and } t := m(t+1) - 1. \tag{2.27}$$

These steps are carried out until the root of T has been processed. For the example of Figure 9, one thus obtains $w_1 = 6, w_2 = w_3 = w_4 = 2, w_5 = 1, w_6 = 7$ and $t = 7$.

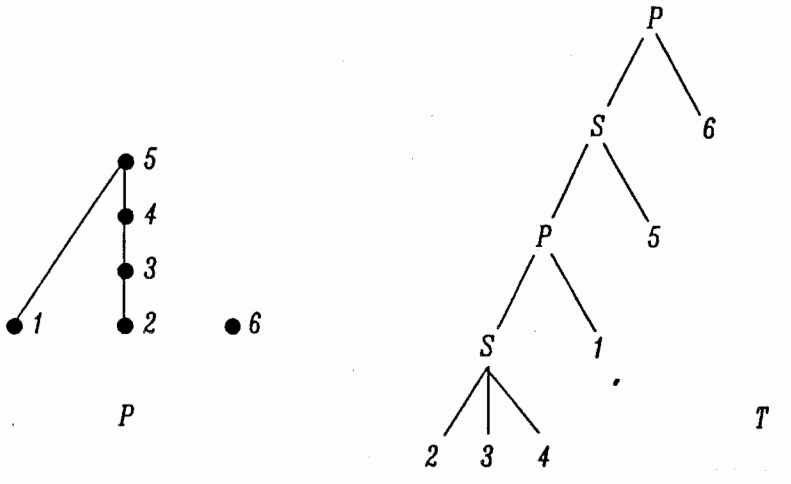


Figure 9: A threshold order P and its decomposition tree T

3. N-free Partial Orders

Like series–parallel–partial orders, also N-free partial orders have been considered independently in different applications. The introduction to this class presented here follows [HJ]. It is based on a very natural generalization of series–parallel partial orders that immediately leads to a tree representation of these partial orders. The other, historically older equivalent characterizations will be dealt with afterwards.

3.1 Definition and Structural Properties

Let $P_1 = (V_1, <_1)$ and $P_2 = (V_2, <_2)$ be partial orders with disjoint ground sets V_1, V_2, and let $M \subseteq Max(P_1), N \subseteq Min(P_2)$ be non–empty subsets of the maximal and minimal elements of P_1 and P_2, respectively.

Then the *quasi–series composition* $P_{QS} = (V, <_{QS})$ of P_1 and P_2 *relative to* M, N is the partial order on $V = V_1 \cup V_2$ defined by

$$u <_{QS} v :\Leftrightarrow \begin{cases} u, v \in V_i & \text{and } u <_i v \ (i = 1, 2) \text{ or} \\ \exists\, a \in M, b \in N & \text{with } u \leq_1 a \text{ and } b \leq_2 v. \end{cases} \tag{3.1}$$

So viewed in the Hasse–diagram, P_{QS} is obtained from P_1 and P_2 by putting all elements of N on top of all elements of M. We will use the notation $P_1 {}_M *_N P_2$ for the quasi–series composition and call P_1 and P_2 the *quasi–series blocks* of P_{QS}.

The class of *quasi–series–parallel* (QSP for short) partial orders is then recursively defined as follows. $P = (V, <)$ is QSP if

$$|V| = 1, \text{ i.e. } P \text{ is a one–element partial order,} \tag{3.2}$$

or

$$\begin{cases} P \text{ is obtained by parallel composition or quasi–series} \\ \text{composition of two smaller QSP partial orders.} \end{cases} \tag{3.3}$$

Obviously, the series composition (2.2) is just a special case of the quasi–series composition since

$$P = P_1 * P_2 \Leftrightarrow \begin{cases} P = P_1 {}_M *_N P_2 \text{ with } M = Max(P_1) \\ \text{and } N = Min(P_2) \end{cases} \tag{3.4}$$

So we have:

3.1 Proposition: *The class of QSP–partial orders contains the class of series–parallel partial orders.*

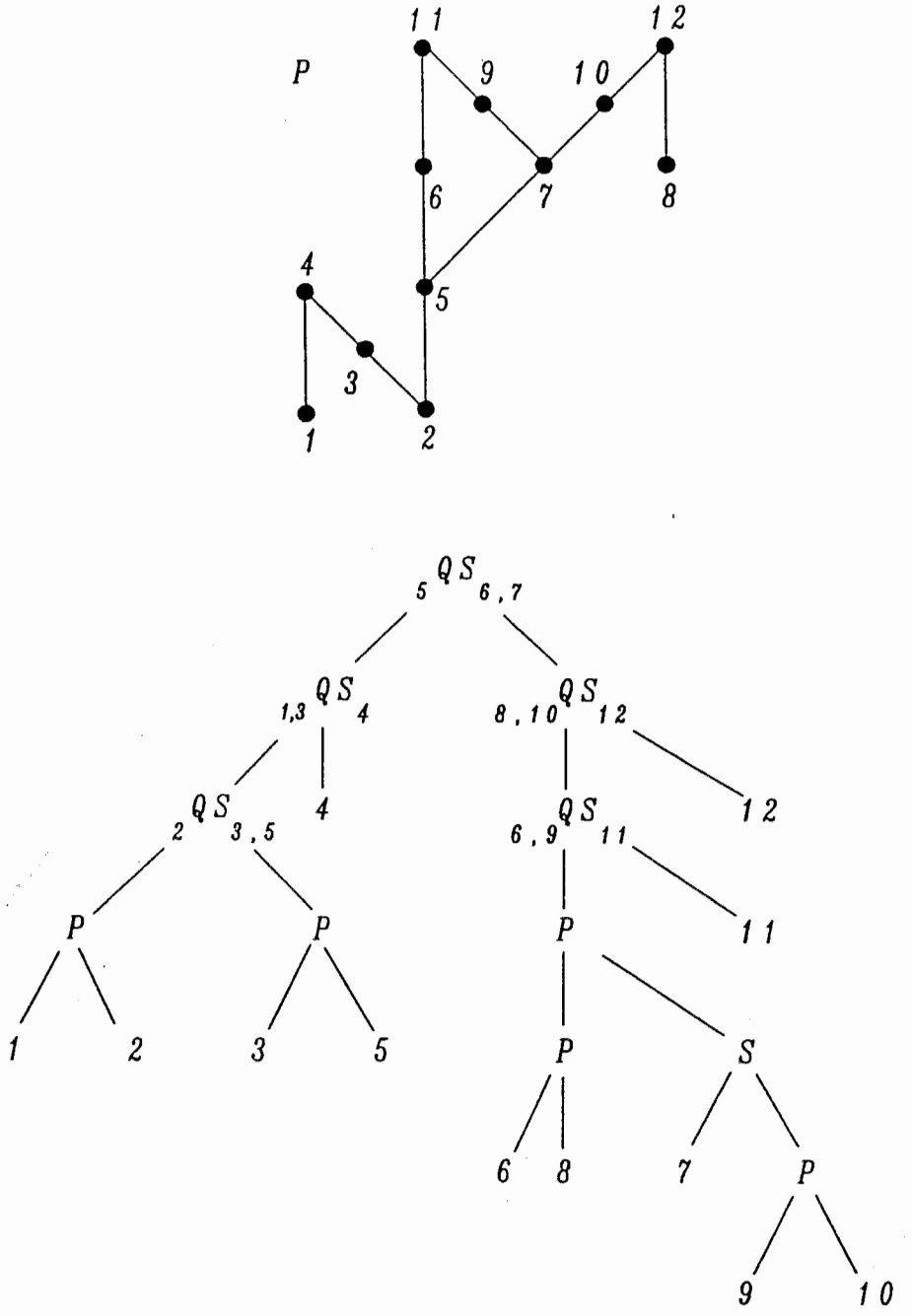


Figure 10: A QSP partial order and a QSP–decomposition tree

As for series–parallel partial orders, the recursive definition yields a *tree representation* of QSP partial orders by a binary tree, the *QSP–decomposition tree*, as shown in Figure 10. The leaves of this tree represent one–element partial orders, and each interior node represents the parallel composition (label P) or the quasi–series–composition (label QS and

sets M, N) of its left and right son.

Usually, the tree representation is not unique. A certain standardization is given by Theorem 3.2 (6) below. It is still not unique, but allows a reduction to quasi-series compositions $P_{1M} *_N P_2$ where $N = Min(P_2)$ and P_2 is an antichain.

The standardized tree leads to a representation of a QSP-partial order $P = (V, <)$ in $O(|V|)$ space, since each set N need not be represented any more, and the sets M must be pairwise disjoint. Then generalizing Proposition 2.1, one obtains

$$(3.5) \qquad u <_P v \Leftrightarrow \begin{cases} \text{the lowest common ancestor of } u \text{ and } v \text{ is a QSP-node} \\ \text{with } u \leq w \text{ for some } w \in M \end{cases}$$

Using the techniques from [HT], the lowest common ancestor can still be found in constant time, but testing wether $u \leq w$ for some $w \in M$ requires some knowledge of the transitive closure of P. Thus the standarized QSP-decomposition tree still represents the *transitive reduction* of QSP partial orders in *linear space*, but no longer the *transitive closure* as does the tree for series-parallel partial orders.

We will now prove the main structural properties of QSP partial orders. We need the following notions.

A partial order P is *CAC* (*chain-antichain-complete*) if $C \cap A \neq \emptyset$ for each maximal chain C and each maximal antichain A of P [Gr,LM]. It is *edge-induced* if it can be obtained from a directed acyclic graph (dag) G in the sense of (2.11). It is called *N-free* [Ri1] if its Hasse-diagram does not contain the N from Figure 4 as induced subgraph.

3.2 Theorem: *Given a partial order $P = (V, <)$, the following statement are equivalent:*

(1) P is QSP.

(2) P is CAC.

(3) P is N-free.

(4) For all $u, v \in V$, $\mathrm{ImPred}(u) = \mathrm{ImPred}(v)$ or $\mathrm{ImPred}(u) \cap \mathrm{ImPred}(v) = \emptyset$.

(5) P is edge-induced.

(6) P is QSP where each quasi-series composition is of the form $P_{1M} *_N P_2$ with $N = Min(P_2)$ and P_2 is an antichain.

We will give an outline of the proof of Theorem 3.2. First, observe that both the parallel composition and the quasi-series composition preserve the CAC-property, which shows "(1) $\Rightarrow$ (2)" by induction along the QSP-decomposition tree.

If P contains the N, i.e. elements a, b, c, d with $a \lessdot b$, $c \lessdot d$, $c \lessdot b$, and $a \parallel c$, $a \parallel d$, $b \parallel d$, then any maximal antichain A containing $\{a, d\}$, and any maximal chain C containing $\{b, c\}$ have $A \cap C = \emptyset$, i.e. the CAC property does not hold. This shows "(2) $\Rightarrow$ (3)".

"(3) ⇒ (4)" is immediate since violation of (4) gives an embedded N.

Now let (4) hold. Then a digraph $G = (V', E')$ inducing $P = (V, <)$ can be defined as follows. The vertices of G are the distinct sets $\mathrm{ImPred}(u), u \in V$, plus the set $\mathrm{Max}(P) = V - \cup_{u \in V} \mathrm{ImPred}(u)$. The edges of G are the elements of P. The *tail* of edge u is the set $\mathrm{ImPred}(u)$, and the *head* of edge u is the (unique because of (4)) set $\mathrm{ImPred}(v)$ containing u. This construction is illustrated in Figure 11 for the QSP partial order of Figure 10.

$u \in V$	1	2	3	4	5	6	7	8	9	10	11	12
ImPred(u)	∅	∅	2	1,3	2	5	5	∅	7	7	6,9	8,10

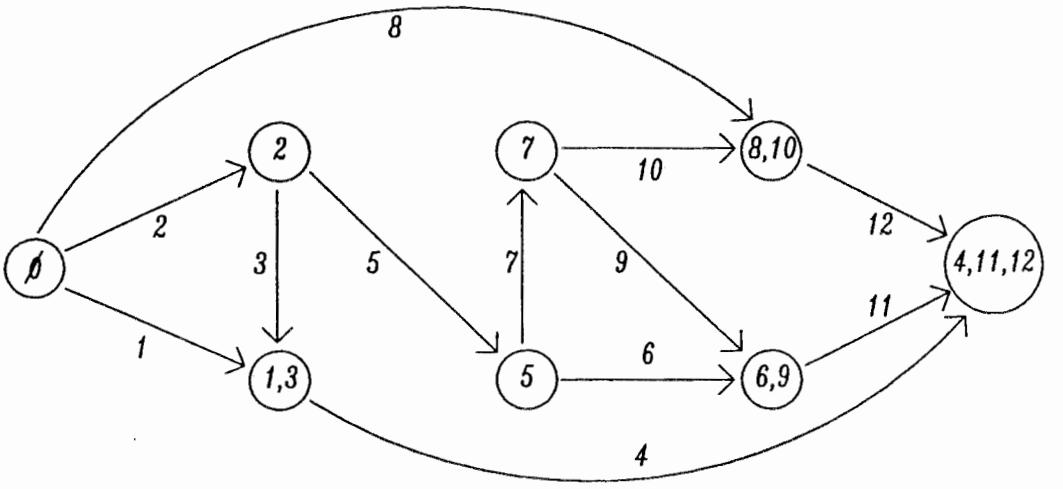


Figure 11: List of immediate predecessor sets and edge diagram of P from Figure 10

Any such digraph G inducing P is called an *edge diagram* of P. From the edge diagramm, one obtains the standardized QSP–decomposition in statement (6) by deleting successively the whole edge set emanating from a node such that all edges have no successor edges. In P, this corresponds to deleting successively a set of all maximal elements with the same immediate predecessors. In the example, such a sequence of deletions is $\{12\}, \{11\}, \{9,10\}, \{6,7\}, \{4\}, \{3,5\}, \{1,2,8\}$. The associated sets of immediate predecessors then give the sets M in the quasi–series composition. This is illustrated in Figure 12 by the standardized decomposition tree obtained from this sequence of deletions.

The last conclusion "(6) ⇒ (1)" is trivial and completes the proof of Theorem 3.2.

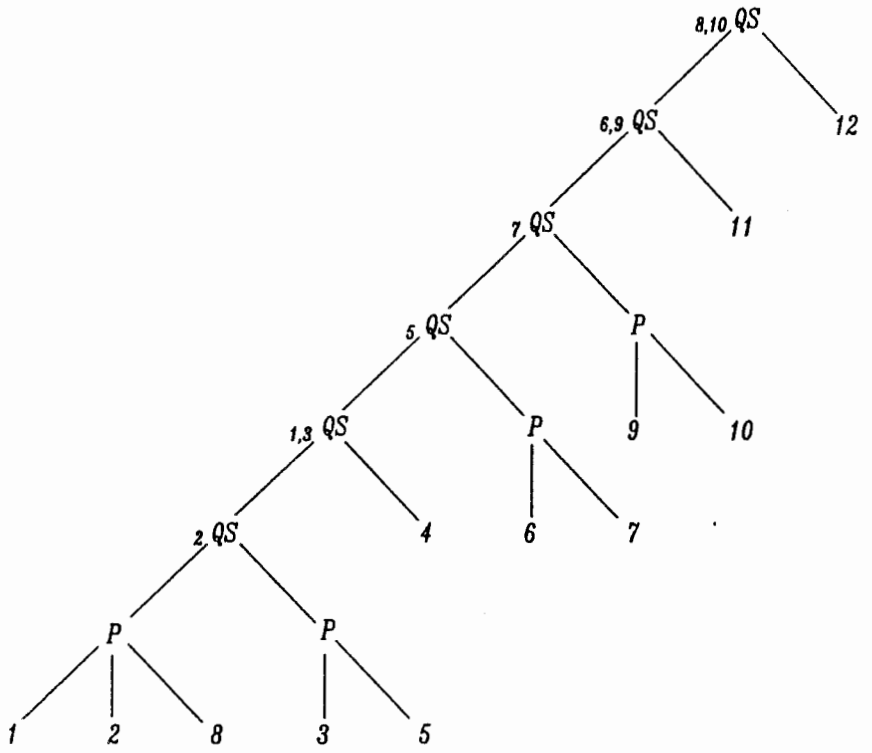


Figure 12: The standardized QSP–decomposition tree of P from Figure 10

Below follow some remarks on the historical development of the different statements of Theorem 3.2.

CAC partial orders were introduced in [Gr]. Grillet also showed that a partial order is CAC iff it does not contain a suborder on 4 elements a, b, c, d with $a < b$, $c \lessdot b$, $c < d$ and $a \parallel c$, $a \parallel d$, $b \parallel d$. This characterization was strengthened in [LM] to $a \lessdot b$ and $c \lessdot d$. The term "N–free" for this condition was introduced in [Ri1].

Property (4) states that the set of immediate predecessors (lower covers) of vertices form a partition. This is a necessary and sufficient condition for a dag to be a line–digraph (see [HN,HB] for a discussion of general digraph to be a line–digraph), which, in the context of N–free partial orders, seems to appear first in [Ka1], cf. also the presentation in [Sy1]. The equivalence between (3) and (4) appears e.g. in [HN, He].

Property (6) is also considered in [FGS,RZ]. The quasi–series composition is expressed in [RZ] as a "gluing operation" motivated from lattice theory, and a very elegant and simple inductive proof of (6) is given.

An equivalent graph–theoretic approach to statements (3)–(5) has been developed in [VTL], cf. also [FGS]. They define the class of *complete bipartite composite dags* (CBC dag for short) as the class of directed acyclic graphs $G = (V, E)$ for which there is a set $\{B_1, \ldots, B_k\}$ of complete bipartite subgraphs of G (called the *bipartite components* of G) such that

(3.6) every edge of G belongs to exactly one bipartite component,

(3.7) for each non–sink vertex v, all edges leaving v belong to the same bipartite component,

(3.8) for each non–source vertex v, all edges entering v belong to the same bipartite component.

It follow easily that every bipartite component B_i joining vertices from S_i with T_i corresponds to a node in the edge diagram with S_i as ingoing and T_i as outgoing edges. Equivalently, each $S_i = \mathrm{ImPred}(u)$ for all $u \in T_i$, and each $T_i = \mathrm{ImSuc}(v)$ for each $v \in S_i$. For the partial order P in Figure 10, the bipartite components are given by $\{1,3\}\times\{4\}, \{2\}\times\{3,5\}, \{5\}\times\{6,7\}, \{6,9\}\times\{11\}, \{7\}\times\{9,10\}$, and $\{8,10\}\times\{12\}$. This gives:

3.3 Theorem: *A dag is CBC iff it is the transitive reduction of an N–free partial order.*

The CBC–formulation contains besides (5) also the dualized version of (5) for the sets of immediate sucessors $ImSuc(u)$, $u \in V$.

There exist several extensions of the notion of N–free that weaken condition (5) [or its dual] by requiring disjointness or *containment* instead of disjointness or *equality* for the sets $\mathrm{ImPred}(u)$, $u \in V$. These classes of partial orders are of interest in connection with the jumb number problem and are considered in more detail in [BH].

The statements of Theorem 3.2 can all be seen as generalizations of properties of series–parallel posets. (1) and (6) generalize the series composition, (2) generalizes statement (3) in Theorem 2.8, (3) Corollary 2.7a), (4) statement (6) in Theorem 2.8, and (5) property (2.11) of series–parallel partial orders.

The main difference is that these properties are inherited by suborders of series–parallel partial orders, but not by suborders of N–free partial orders. In fact, we have:

3.4 Proposition:

(a) None of the properties of Theorem 3.2 is hereditary.

(b) If one of the properties of Theorem 3.2 is required to hold for all ideals of P, then P is already series–parallel.

(c) Every partial order can be embedded as induced suborder into an N–free partial order.

The smallest example showing a) is the partial order N' from Figure 13 below. Property b) was observed by [FS3]. Property c) follows by subdividing in the Hasse diagram of an

arbitrary partial order P each edge by a vertex. The resulting partial order is N–free and contains P as induced suborder.

This embedding property is the reason that some computationally intractable properties of arbitrary orders are inherited by N–free partial orders, cf. Section 3.3.

The embedding property also implies that every partial order $P = (V, <)$ can be assigned an *edge diagram* by taking the edge diagram of an N–free partial order $P^* = (V^*, <^*)$ containing P as induced suborder. The elements of $V^* - V$ are called *dummy edges* or *dummies* of the edge diagram. For example, the *subdivision technique* for proving (c) above creates a dummy for each edge in the Hasse diagram of P.

In finding such a diagram, it is desired to keep the number of dummies small. For a discussion of several such constructions, as well as complexity results (minimizing the number of dummies is NP–hard) we refer to [Sy2, Sp3].

We close this section by showing that the characterization of series–parallel partial orders via the forbidden "Wheatstone bridge" in [Du] is equivalent to characterizing series–parallel partial orders within the class of N–free partial orders.

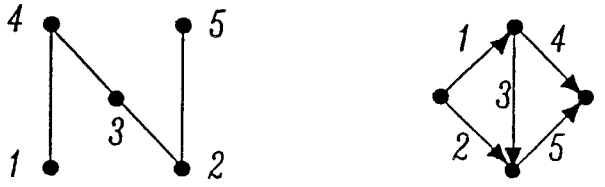


Figure 13: Hasse and edge–diagram of the partial order N'.

3.5 Theorem: *For a partial order P, the following conditions are equivalent*

(1) P is series–parallel.

(2) P is N–free and does not contain the partial order N' of Figure 13 as induced suborder.

If P is series–parallel, then P is N–free by Proposition 3.1 and does not contain the N by Corollary 2.7, and thus also not N' (since N' contains N). So (2) follows from (1).

Let (2) hold and assume that P is not series–parallel. Then P contains the N by Corollary 2.7, i.e. elements a, b, c, d with $a < b$, $c < d$, $c < b$ and $a \parallel c$, $a \parallel d$, $b \parallel d$. Let C be a maximal chain containing $\{b, c\}$ and A be a maximal antichain containing $\{a, d\}$. Since P is N–free because of (2), it is CAC because of Theorem 3.2. So there is $e \in A \cap C, e \neq a, b, c, d$. It follows that the restriction of P to $\{a, b, c, d, e\}$ must be N', a contradiction to (2). Hence (2) implies (1).

Since the edge diagram of N' is just the Wheatstone bridge of Figure 4, it follows that the characterization of series–parallel partial orders in [Du] is equivalent to Corollary 2.7 a). This equivalence was e.g. used in [VTL] to give another proof of Corollary 2.7 b) from

the characterization in [Du].

3.2 Recognition Algorithms

The fastest known recocognition algorithms for N–free partial orders assume that the partial order $P = (V, <)$ is given in transitively reduced form and construct either an edge diagram or a standardized QSP decomposition tree if the partial order is N–free. Their running time is $O(n + m)$, where $n = |V|$ and m is the number of edges in the transitive reduction of P.

The first such algorithm is implicitly contained in the recognition algorithm for series–parallel partial orders in [VTL]; see Step 2 in the description of this algorithm in Section 2.3. It is based on the characterization in Theorem 3.3 and constructs first the bipartite components of the transitive reduction (or states that P is not N–free) and then constructs an edge diagram from them.

The first "explicit" linear recognition algorithm for N–free partial orders appears in [Sy1]. It verifies property (4) in Theorem 3.2 and constructs an edge diagram along the same lines as the proof of "(4) $\Rightarrow$ (5)" in Theorem 3.2.

The most recent algorithm in [HJ] also verifies property (4) but constructs the standardized QSP–decomposition tree instead of an edge diagram.

All algorithm have the common property that they find an embedded N if the condition they try to verify is violated.

We will give a short presentation of the algorithm of Syslo [Sy1].

3.6 Algorithm *(Recognition of N–free partial orders):*

Input: The transitive reduction of $P = (V, <)$ by, for each $v \in V$, a list of its immediate predecessors ImPred(v).

Output: An edge diagram if P is N–free. An embedded N if P is not N–free.

Method: Assign labels head(u), tail(u) to each $u \in V$. These labels define an edge diagram if P is N–free.

Step 0: (*Initialization*) Put tail(u) := 0, head(u) := $n + 1$ for all $u \in V$ (with $n = |V|$). Put $p := 0$ (p denotes the current number of interior nodes in the edge diagram).

Step 1: (*Assigning a tail*) Choose $v \in V$ with ImPred(v) $\neq \emptyset$. Choose any $u \in$ ImPred(v) and consider head(v). If head(v) = $n + 1$ (i.e. no successor so far), then put $p := p + 1$ (create a new node in the edge diagram) and put head(u) := p and tail(v) := p. Otherwise (i.e. head(v) $\neq n + 1$) put tail(v) := head(u).

Step 2: (*Checking for condition (4) in tail(v)*) For every $u' \in$ ImPred(v) $- \{u\}$ do the following: If head(u') = $n+1$ (u' is still pointing to the sink) then put head(u') = tail(v). If head(u') $\neq$ tail(v), then go to Step 4.

Step 3: (*Positive termination condition*) Put $V := V - \{v\}$. If $V \neq \emptyset$ go to Step 1. Otherwise stop. $G = (V', E')$ with $V' = \{0, 1, \ldots, p\} \cup \{n+1\}$ and $E' = \{(v, \text{tail}(v), \text{head}(v)) \mid v \in V\}$ is an edge diagram of P.

Step 4: (*Negative termination condition*) Stop. P is not N–free. The elements v, u, u' and any w with $\text{tail}(w) = \text{head}(u')$ form an embedded N.

It follows easily that this algorithm runs in linear time. So we have

3.7 Theorem: *If $P = (V, <)$ is given in transitively reduced form with n vertices and m edges, then testing whether P is N–free and constructing an edge diagram can be done in $O(n+m)$ time.*

The construction of the standardized QSP–decomposition tree in linear time is only slightly more involved. It uses in addition the height function of P to find the currently last antichain in the proof of "(5) $\Rightarrow$ (6)" in Theorem 3.2. For details, we refer to [HJ].

3.3 Some Applications

One of the main and oldest applications of N–free partial orders is their use in *project analysis*, in particular in the older techniques such as CPM or PERT, see e.g. [El,MP]. These techniques represent a project by a digraph in which the edges correspond to the activities of the project and nodes correspond to events (the completion of all activities entering the node).

In order–theoretic terms, this so–called *activity–on–edge representation* or *PERT–network* is just the edge diagram of an N–free partial order. If the original partial order P describing the technological precedence constraints of the project is not N–free, then *dummy activities* are added to make it N–free and thus obtain an edge diagram of the larger N–free partial order P^* containing P as induced suborder. Many techniques have been proposed for this task, cf. [Sy2, Sy4, Sp3] for further references.

This representation of projects by edge diagrams was chosen in order to apply digraph algorithms that work with edge weights such as *longest path algorithms* (to compute the *project duration* and the *critical paths*) or *flow algorithms* (to compute resource requirements or maximum weighted antichains). In some of these applications it is essential to use the edge diagram. For instance, the flow algorithm mentioned requires the CAC–property of N–free partial orders, see [Mö4, p.64ff] for details.

In general, however, the use of edge diagrams, i.e. the embedding into N–free partial orders is rather artificial and has led to misunderstandings concerning certain results or algorithms. For instance, consider the *free slack* $\text{FS}(u)$ of an activity u. $\text{FS}(u)$ denotes the maximum time by which u may be delayed without increasing the project duration or

delaying the earliest start of any successor of u. If x_v denotes the *duration* of activity v, $PD = C_{max}(P, x)$ denotes the *project duration*, and $\mathrm{ES}(v)$ denotes the earliest start of v (cf. Section 1.3 for details), then $\mathrm{FS}(u)$ is obtained as

$$FS(u) = \begin{cases} PD - ES(u) - x_u & Suc(u) = \emptyset \\ min_{v \in ImSuc(u)}[ES(v) - ES(u) - x_u], & \text{otherwise.} \end{cases} \tag{3.9}$$

If this recursive computation is applied to the edge diagram (with dummies considered as real activities with duration 0) instead of the partial order P of precedence constraints, then dummies may obtain a positive free slack that actually belongs to a real activity. An example is given in Figure 14.

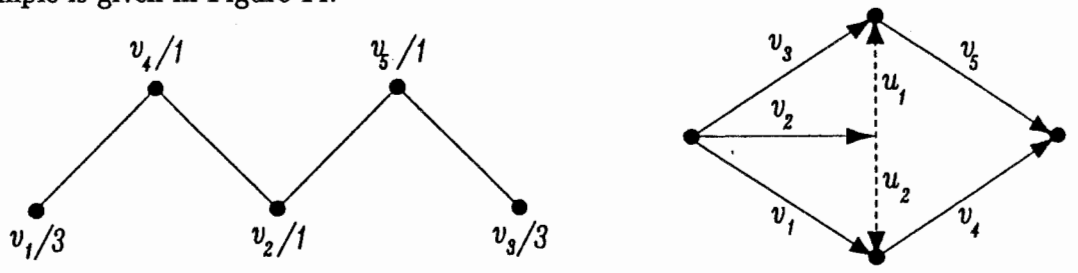


Figure 14: Dummies take free slack

While this wrong procedure for calculating the free slack can easily be corrected, there is another, more fundamental property of free slacks in N–free partial orders that often leads to wrong conclusions when dealing with edge diagrams.

A *path* in the edge diagram is a maximal chain in the associated partial order. The *path slack* $PS(C)$ of a path C is defined as

$$PS(C) = PD - \sum_{v \in C} x_v, \tag{3.10}$$

i.e. the difference between project duration and path length. The following relationship [Ra1] holds for path slacks of N–free partial orders.

3.8 Theorem: *The following statements are equivalent*

(1) P is N–free

(2) For each path C of P, $PS(C) = \sum_{v \in C} FS(v)$ for all possible durations x_v of v, i.e., the path slack is the sum of the free slacks of the activities on the path.

Property (2) is usually assumed in network analysis when dealing with free slacks. However, it is actually only a property of N–free partial orders, as the equivalence with (1) shows. This means that, when arbitrary partial orders are embedded into edge diagrams, then necessarily some of the dummies will obtain a positive free slack in order to achieve

equality in (2), thereby possibly taking away (some of) the free float from other activities. An example illustrating Theorem 3.8 given in Figure 15. For further implications of project analysis with N–free partial orders, we refer to [Ka2, Ra1].

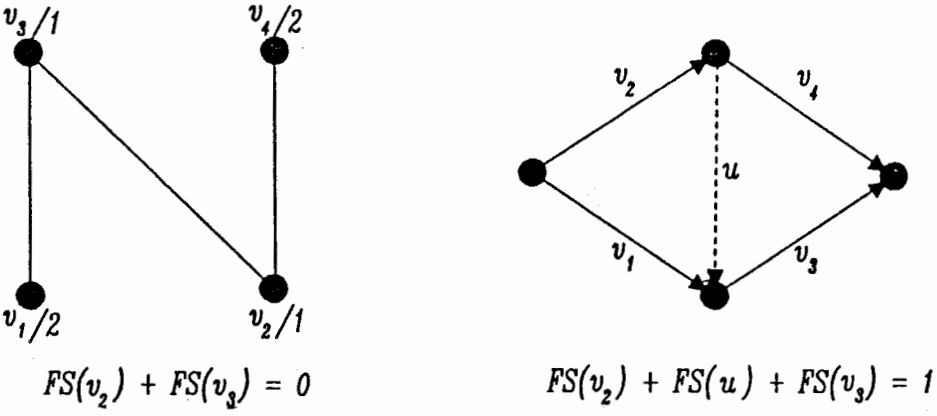


Figure 15: Path slack and free slack

The other, major application of N–free partial orders arose with the investigation of the jump number. This number can be computed by a simple greedy algorithm in N–free partial orders [Ri1], and is also related to several structural properties of N–free partial orders.

3.9 Theorem: *The jump number of an N–free partial order is equal to the cyclomatic number of its edge diagram [Sy4] and to the number of parallel nodes in any binary QSP decomposition tree [HJ].*

These results have led to many further investigations and extensions of the class of N–free partial orders. These are treated in [BH] in this volume.

Apart from the jump number, little is known about the tractibility of other problems on N–free partial orders. Some negative results have occurred in [HMö]. They show that some of the problems that are still polynomially solvable on series–parallel partial orders become on N–free partial orders already as hard as on general partial orders. The proof exploits the fact that every partial order can be embedded into an N–free partial order, and that thus N–free partial orders inherit some of the difficulties of arbitrary partial orders.

3.10 Theorem: *The following holds on the class of N–free partial orders:*

(a) The isomorphism problem is isomorphism complete.

(b) The $1\,|prec|\sum_j w_j C_j$ scheduling problem is NP–complete.

It is still an open problem whether or not this is also true for the dimension.

4. Decomposable Partial Orders

The *substitution decomposition* (also called *modular decomposition* or *lexicographic decomposition*) of partial orders arose in connection with investigating the structure of comparability graphs [Ga]. It is by now a well–understood theory (cf. [Mö4] for an overview) that has many applications in discrete mathematics, also outside the theory of ordered sets and graphs (see [MR1] for a general discussion of the substitution decomposition).

One arives at computationally tractable classes of partial orders if certain parameters of the substitution process (such as size or width) are bounded. This leads to the classes of partial orders with bounded decomposition diameter and bounded decomposition width treated below. Both classes generalize the class of series–parallel partial orders and lead to polynomial algorithms (where the bound on the diameter or the width may enter the degree of the polynomial) for nearly all of the problems solvable on series–parallel partial orders. So in this respect, they constitute a more appropriate generalization of series–parallel partial orders than N–free partial orders.

4.1 Facts about the Substitution Decomposition

We review here some facts about the substitution decomposition. For more extensive overviews, we refer to [Mö4, MR1].

Let $Q = (V', <_Q)$ be a partial order, let $a_1, \ldots, a_h \in$ be pairwise distinct elements from V', and let $P_1 = (V_1, <_1), \ldots, P_h = (V_h, <_h)$ be partial orders with mutually disjoint ground sets $V', V_1, \ldots, V_h$. Then we denote by

$$P = Q_{a_1,\ldots,a_h}^{P_1,\ldots,P_h} \tag{4.1}$$

the partial order $P = (V, <_P)$ resulting from *substituting* the elements a_i of Q by the associated partial order P_i, $i = 1, \ldots, h$. More formally,

$$V = (V' - \{a_1, \ldots, a_h\}) \cup V_1 \cup \ldots \cup V_h \tag{4.2}$$

and

$$a <_P b \Leftrightarrow \begin{cases} \exists\, i & \text{with } a, b \in V_i \text{ and } a <_i b, \text{ or} \\ \exists\, i \neq j & \text{with } a \in V_i, b \in V_j \text{ and } a_i <_Q a_j. \end{cases} \tag{4.3}$$

An example is given in Figure 16.

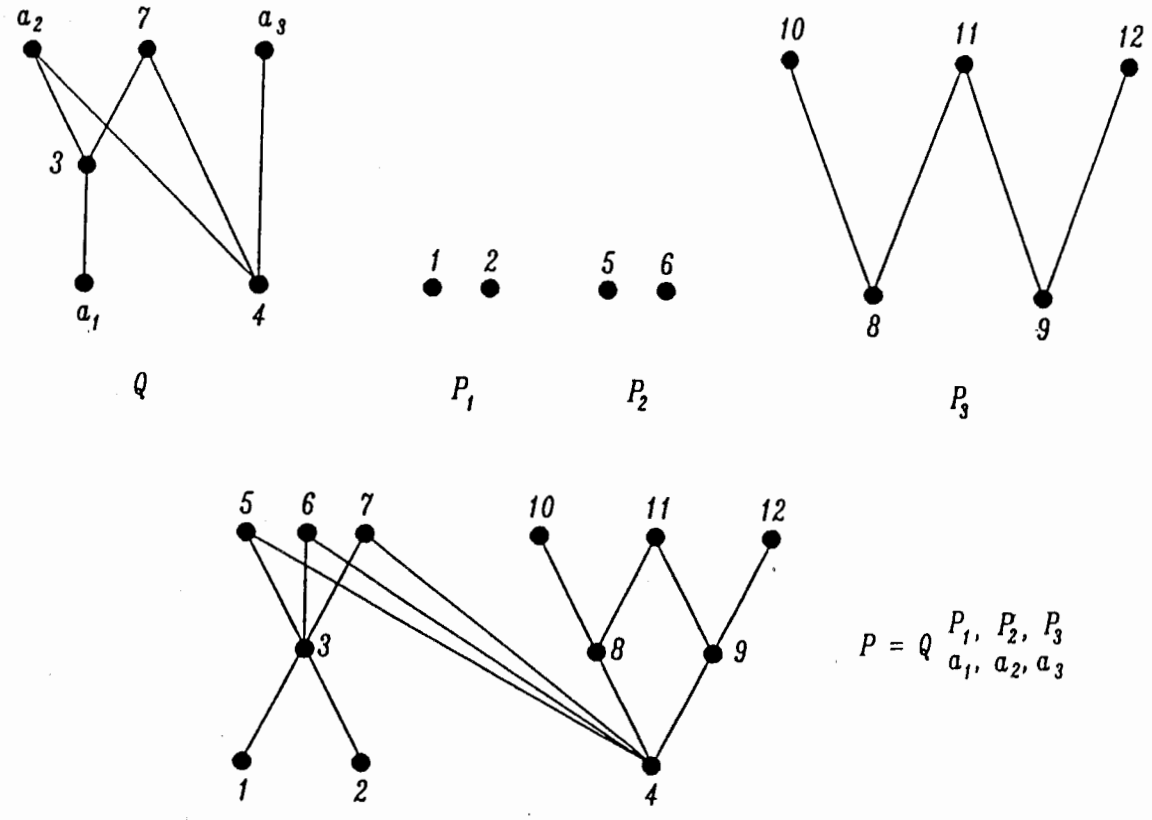


Figure 16: A substitution of partial orders

The substitution is *proper* if $1 < |V_i| < |V|$ for some i. A partial order is *decomposable* if it can be obtained by proper substitution. Otherwise it is said to be *indecomposable* or *prime*. A common alternative notation for (4.1) is $P = Q[P_1, \ldots, P_h]$, with the understanding that every element a_i of Q is substituted by some (possibly one–element) partial order P_i.

Every decomposable partial order P is obtained by a sequence

$$P_1 = Q_1, P_2 = (P_1)^{Q_2}_{a_2}, \ldots, P = (P_{m-1})^{Q_m}_{a_m} \tag{4.4}$$

of "elementary" substitutions in which each Q_j is prime. The reverse of such sequence is also referred to as a *composition series* [MR1]. The prime partial orders $Q_1, \ldots, Q_m$ are called the *factors* of the composition series.

Composition series fulfill both a "Jordan–Hölder" and a "Church–Rosser" property [MR1].

4.1 Theorem: *Any two composition series of a partial order have the same length and the same factors up to rearrangement and isomorphism (Jordan–Hölder) and the same last factor Q_1 up to isomorphism (Church–Rosser).*

The maximum size of the factors is called the *decomposition diameter* of P, and the

maximum width of the factors is called the *decomposition width* of P. They are denoted by $\mathrm{decd}(P)$ and $\mathrm{decw}(P)$, respectively.

Figure 17 presents a composition series with the associated factors for the partial order P from Figure 16. It follows that $\mathrm{decd}(P) = 5$ and $\mathrm{decw}(P) = 3$.

A subset B of the ground set V of $P = (V, <)$ is called *autonomous* if $a < b_0$ (resp. $a > b_0$) for some $b_0 \in B$ and $a \in V - B$ implies that $a < b$ (resp. $a > b$) for all $b \in B$.

A partial order $P = (V, <)$ is decomposable iff it has a non-trivial autonomous set B (i.e. an autonomous set with $1 < |B| < V$). Then $P = Q_a^{P|B}$, where $P|B$ denotes the suborder of P induced by B, and where Q is obtained from P by replacing B by just one vertex a.

If $\pi = \{B_1, \ldots, B_m\}$ is a partition of V into autonomous sets, then we denote by P/π the partial order Q obtained from P by replacing the sets B_i by just one representative vertex $a_i \in B_i$. P/π is called the *quotient* of P *modulo* π.

The following basic decomposition theorem [Ga] shows that the substitution operation generalizes the parallel and the series composition introduced in Section 2.1.

4.2 Theorem (Decomposition Theorem): *For each decomposable partial order P, one of the following cases applies:*

(1) $P = Q_{a_1,\ldots,a_h}^{P_1,\ldots,P_h}$, *where Q is an antichain. Then P is obtained by parallel composition of $P_1, \ldots, P_h$.*

(2) $P = Q_{a_1,\ldots,a_h}^{P_1,\ldots,P_h}$, *where Q is a linear order. Then P is obtained by series composition of $P_1, \ldots, P_h$.*

(3) $P = Q_{a_1,\ldots,a_h}^{P_1,\ldots,P_h}$, *where Q is a (uniquely determined) prime partial order. Then P is said to be of the prime type and Q is called the associated prime quotient.*

These 3 *mutually exclusive* cases are used to represent any partial order in a *decomposition tree* T. The root of T is V, the leaves are the elements of P, and the sons of interior nodes are the ground sets of the partial orders P_i in the respective composition (parallel, series, or prime type) of the partial order associated with the node.

The tree is unique if we require the quotient in (1) and (2) to be as large as possible. We will call this tree the *canonical decomposition* tree. Then the tree nodes correspond exactly to the so-called *strongly autonomous* sets (i.e. those autonomous sets that do not properly overlap with any other autonomous set). The nodes of the tree are labeled with P and S, if case (1) (parallel composition) or (2) (series composition) of Theorem 4.2 applies. So unlabeled nodes correspond to prime type partial orders. They are called *prime type nodes.*

As an immediate consequence of the Decomposition Theorem, one obtains:

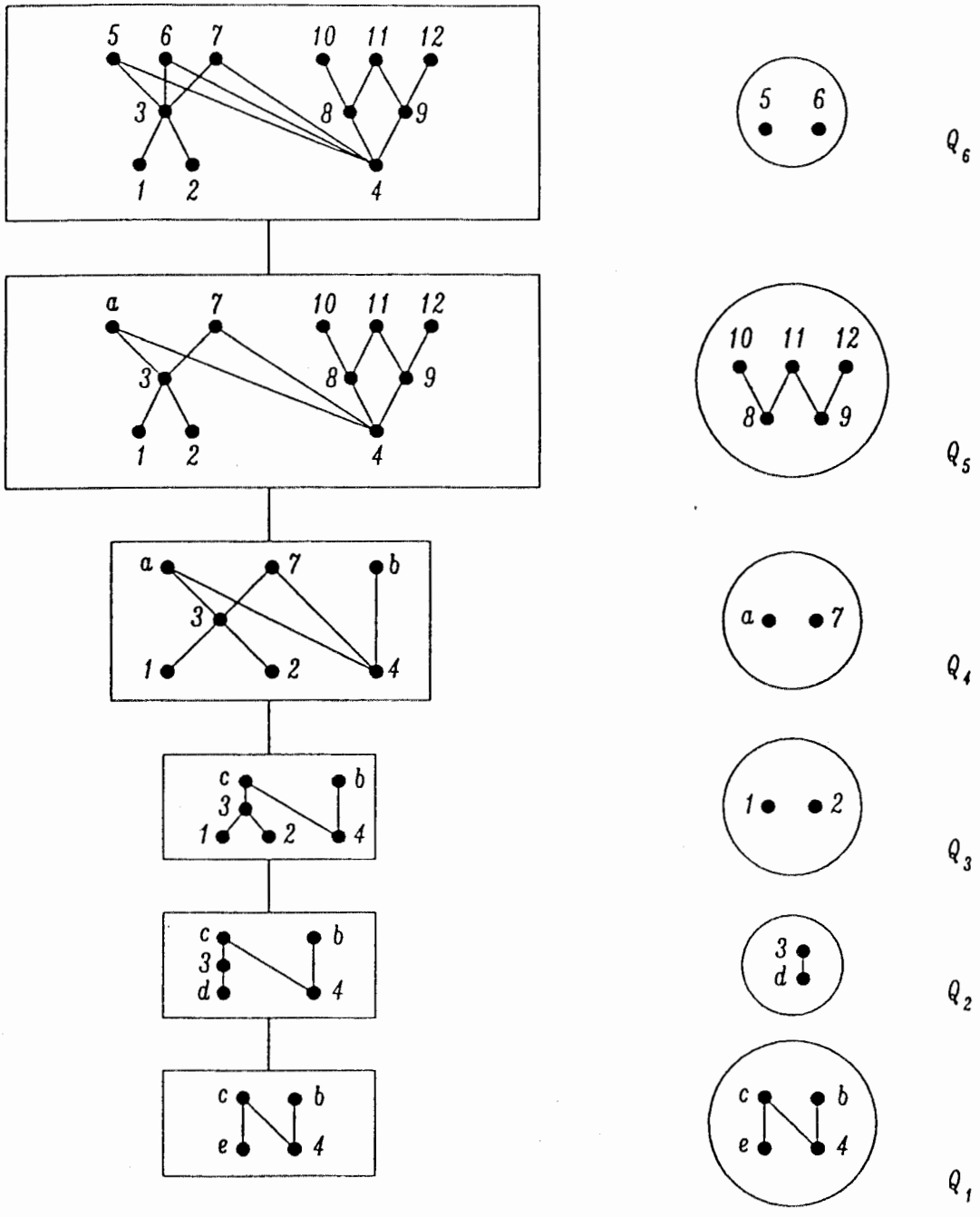


Figure 17: A composition series and the associated factors for the partial order from Figure 16

4.3 Corollary: *A partial order is series–parallel iff its (canonical) decomposition tree contains no prime type node.*

For the partial order P from Figure 16, the canonical decomposition tree T is given in

Figure 18. Note that P is not series–parallel since T contains a prime type node.

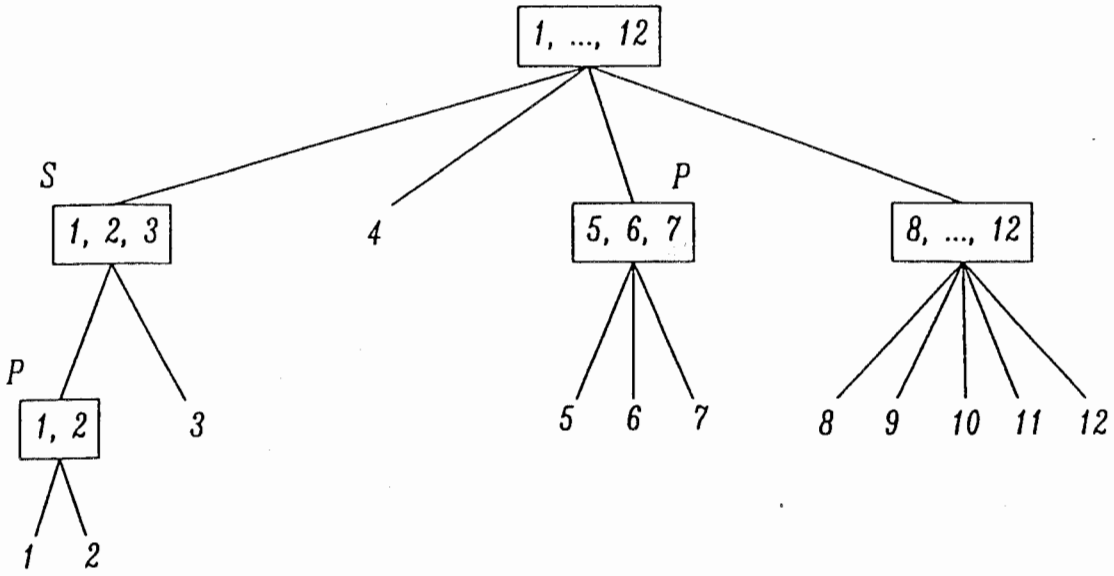


Figure 18: The canonical decomposition tree of the partial order from Figure 16

The factors of a partial order P can be obtained from the (canonical) decomposition tree. A series node with k sons contributes $k-1$ times a 2–element chain as factors, while a parallel node with k sons contributes $k-1$ times a 2–element antichain. Each prime type node N with sons $N_1, \ldots, N_k$ contributes the factor P_N obtained by choosing an arbitrary element from each N_i and taking the induced suborder of P.

These considerations give

$$\text{decd}(P) = \begin{cases} 2 \text{ if } P \text{ is series–parallel,} \\ \text{maximum number of sons of a} \\ \text{prime type node, otherwise} \end{cases} \tag{4.5}$$

and

$$\text{decw}(P) = \begin{cases} 2 \text{ if } P \text{ is series–parallel,} \\ \text{maximum width of the partial orders } P_N \\ \text{associated with prime type nodes, otherwise.} \end{cases} \tag{4.6}$$

The canonical decomposition tree can be constructed efficiently. The fastest methods (there are 3 alternatives) require $O(n^2)$ time for a partial order with n elements [Sp1,MSp,Sp5]. Less involved methods require $O(n^3)$ time [HMa, BM]. An overview on these algorithms (except [Sp5]) is contained in [Mö4].

4.2 Partial Orders with Bounded Decomposition Diameter or Width

It has been observed by many people that he substitution decomposition can be used to factorize many problems on partial orders (and other discrete structures) into smaller parts by exploiting a decomposition $P = Q_{a_1,\dots,a_k}^{P_1,\dots,P_k}$ or by traversing the decomposition tree T of P. An overview on such applications is contained in [Mö4, MR1].

In general, however, this factorization does neither reduce the *worst-case complexity* of a problem (since there exist prime partial orders of arbitrarily large size) nor the *average case complexity* (since almost all partial orders are prime with respect to the uniform distribution on a fixed ground set [Mö3]). But if the prime factors occuring in a composition series are small or have other computationally favourable properties, then the complexity can be reduced.

When the size of the prime factors is bounded by some natural number k, one arrives at the class $\mathcal{P}_{d\le k}$ of partial orders P with decomposition diameter $\operatorname{decd}(P) \le k$.

An equivalent characterization of $\mathcal{P}_{d\le k}$ can be obtained by observing that every partial order contains its factors as induced suborders. This gives:

4.4 Proposition: *$P \in \mathcal{P}_{d\le k}$ iff P does not contain a prime induced suborder with $l > k$ elements.*

Note that the condition "$l > k$" cannot be replaced by "$l = k + 1$" since there exist l–element prime partial orders not containing an induced $(l-1)$–element prime suborder [Su1]. This establishes a major difference between the characterization of series–parallel partial orders (each l–element prime partial order contains an induced N) and $\mathcal{P}_{d\le k}$.

Proposition 4.4 and (4.4) then imply:

4.5 Corollary: *The following holds for $\mathcal{P}_{d\le k}$:*

(1) $\mathcal{P}_{d\le k}$ is the smallest class of partial orders that contains the partial orders with at most k elements and is closed under substitution.

(2) $\mathcal{P}_{d\le k}$ is hereditary, i.e. every induced suborder of $P \in \mathcal{P}_{d\le k}$ belongs also to $\mathcal{P}_{d\le k}$.

(3) $\mathcal{P}_{k\le d}$ contains for each $k \ge 2$ the class of series–parallel partial orders. The containment is proper if $k \ge 4$.

(4) Membership of $P = (V, <)$ in $\mathcal{P}_{d\le k}$ can be tested in $O(|V|^2)$ time.

This class was first considered in [MR2] in connection with the $1\,|\mathrm{prec}|\,\sum_i w_i C_i$–scheduling problem, and considered again in [HMö] in a broader perspective.

By now, the following problems are known to be solvable in polynomial time on $P_{d\le k}$.

The *dimension* $\dim(P)$ can be computed in $O(k! \cdot n^2)$ time by means of Hiragushi's [Hi] formula

$$\dim(Q_{a_1,\dots,a_k}^{P_1,\dots,P_k}) = \max\{\dim(Q), \dim(P_1), \dots, \dim(P_k)\}, \tag{4.7}$$

where the dimension of each prime partial order involved is computed by brute force (giving the constant $k!$ in the bound).

The *jump number* $j(P)$ can be computed by similar arguments in $O((2k)!n^2)$ time because of the fact that

$$(4.8)\qquad j(Q^{P_1,\dots,P_h}_{a_1,\dots,a_h}) = \begin{cases} \sum_{i=1}^h j(P_i) & \text{if } Q \text{ is a linear order,} \\ \sum_{i=1}^h j(P_i) + h - 1 & \text{if } Q \text{ is an antichain,} \\ \sum_{i=1}^h j(P_i) + j(Q^{S_2,\dots,S_2}_{a_1,\dots,a_h}) - h, & \text{otherwise.} \end{cases}$$

Here S_2 is the 2–element antichain that is substituted for each of the a_i in the case that Q is of the prime type in the decomposition theorem. This formula can be derived from results in [Hab,FGS,St3]. The jumb number of the at most $2k$–element partial orders arising in the process is again computed by brute force.

The *number of linear extensions* $l(P)$ can be computed along the canonical decomposition tree in $O(n^{k^2})$ time by means of the formulas (2.18) and (2.19) in the cases (1) and (2) of the Decomposition Theorem, and by

$$(4.9)\qquad l(Q^{P_1,\dots,P_h}_{a_1,\dots,a_h}) = l(Q^{L_1,\dots,L_h}_{a_1,\dots,a_h}) \cdot \Pi_{i=1}^h l(P_i),$$

where L_i is an arbitrary linear extension of P_i. The computation of $l(Q^{L_1,\dots,L_h}_{a_1,\dots,a_h})$ can be carried out in $O(n^{k^2-2})$ time [HMö] by looking at all possible positions $r_1,\dots,r_h$ at which the last elements of the linear extensions $L_1,\dots,L_h$ may occur in a linear extension L of Q, and, for each fixed such choice of positions $r_1,\dots,r_h$, by looking at all possible combinations of subsets of $L_1,\dots,L_r$ that may be scheduled between successive positions $r_{i_j}, r_{i_{j+1}}$. For each such choice, the counting problem then reduces to *merging parallel chains*. Since $h \le k$, at most n^{k^2-2} cases have to be considered.

Essentially the same arguments combined with Sidney's [Si] result that the one–machine scheduling problem $1\,|\mathrm{prec}|\,\sum_j w_jC_j$ can be solved by decomposition (i.e. given an optimal solution L_A of an autonomous suborder $P \mid A$ of P, there exists an optimal solution L of P preserving L_A, i.e. $L \mid A = L_A$) has led to an $O(n^{k^2})$ algorithm for the class $\mathcal{P}_{d\le k}$ in [MR2]. Instead of counting the mergings of parallel chains as in the counting problem, the chains are merged optimally by the same rules as for the series–parallel case in Section 2.4.

Other well–solved problems on the class $\mathcal{P}_{d\le k}$ are the *isomorphism problem* [HMö], and counting *the number of ideals* [FS2].

These results show that virtually every problem that can be solved efficiently for series–parallel partial orders can also be solved in polynomial time in $\mathcal{P}_{d\le k}$, but with k entering the coefficient and/or the degree of the bounding polynomial.

A larger class of partial orders with similar algorithmic properties as $\mathcal{P}_{d\le k}$ has been introduced in [SiS] in connection with the $1\,|\mathrm{prec}|\,\sum w_jC_j$–scheduling problem.

This class is obtained by bounding the *width* of the prime factors instead of the *size*. So given $k \in \mathbb{N}$, $\mathcal{P}_{w\le k}$ denotes the class of all partial orders P with $\mathrm{decw}(P) \le k$.

Similarly to Proposition 4.4 and Corollary 4.5, one obtains (with $w(P)$ denoting the width of P):

4.6 Proposition: *The following holds for* $\mathcal{P}_{w\leq k}$:

(1) $P \in \mathcal{P}_{w\leq k}$ *iff* P *does not contain a prime induced suborder* P' *with* $w(P') > k$.

(2) $\mathcal{P}_{w\leq k}$ *is the smallest class of partial orders that contains the partial orders with width* $\leq k$ *and is closed under substitution.*

(3) $\mathcal{P}_{w\leq k}$ *is hereditary.*

(4) For each k, $\mathcal{P}_{d\leq k} \subseteq \mathcal{P}_{w\leq k}$

(5) Membership of $P = (V, <)$ *in* $\mathcal{P}_{w\leq k}$ *can be tested in* $O(|V|^3)$ *time.*

The membership test in (5) can be done by first computing the canonical decomposition tree and then testing each factor P_N associated with a prime type node N for $w(P_N) \leq k$. The latter can be done by determining a matching in a bipartite graph [FF] or by flow methods (see, e.g. [Mö4, p65ff]), thus determining the complexity of $O(|V|^3)$.

The problems that can be solved efficiently on $\mathcal{P}_{w\leq k}$ have the common property that (1) they can be solved by decomposition and that (2) they can be solved in polynomial time if the width is bounded.

We illustrate this on several scheduling and sequencing problems following [CP,SiS]. Because of Dilworth's Theorem, every (finite) partial order P with width w and size n can be partitioned into w chains $C_1, \ldots, C_w$ having $n_1, \ldots, n_w$ elements, respectively. Such a partitioning can be obtained in $O(n^3)$ time by network flow techniques (see e.g. [Mö4; p65ff]). Then each order ideal can be characterized by how many elements it contains from each of the w chains. Consequently, the number N of ideals of P is bounded by

$$N \leq (1+n_1)\cdot(1+n_2)\cdot\ldots\cdot(1+n_w) \leq \left(\frac{n+w}{w}\right)^w = O(n^w). \tag{4.10}$$

This bound is sharp if P consists of w parallel chains.

Now every initial part of a feasible schedule for any scheduling problem corresponds to an order ideal, and at any completion of a job, there is a *transition* into other order ideals by starting new jobs.

This implies that the construction of a feasible schedule can be viewed as following a path in the *digraph of order ideals*. The vertices of this digraph are the order ideals, and two order ideals I_1, I_2 are connected by an edge if I_2 is obtained from I_1 at a job completion. An illustrating example is given in Figure 19 for the case of a 1–machine schedule for P_1 and a 2–machine schedule for P_2 for the case of unit processing times.

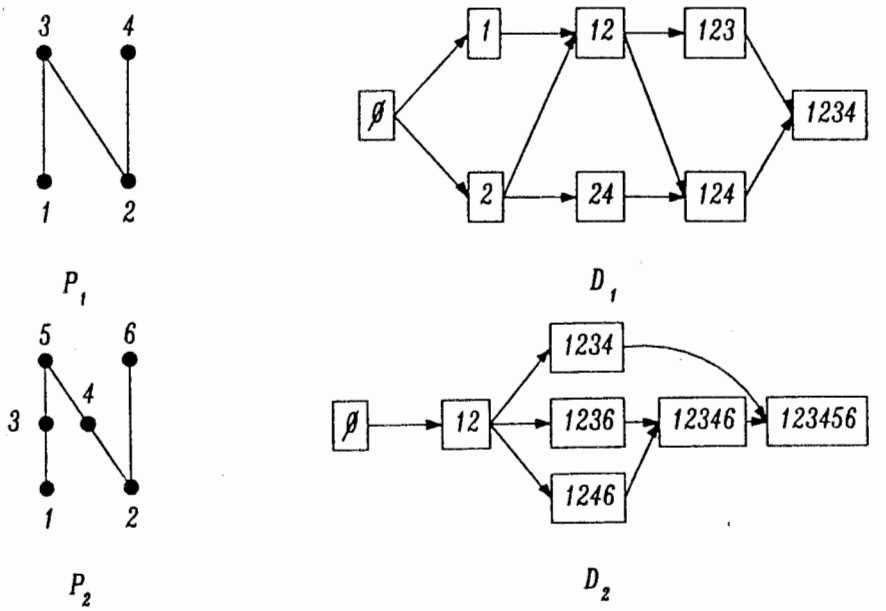


Figure 19: Digraphs of order ideals for two scheduling problems.

For instance, the path $\emptyset \to 2 \to 24 \to 124 \to 1234$ in D_1 corresponds to the 1–machine schedule (linear extension) $2 - 4 - 1 - 3$ of P_1. Similarly, the path $\emptyset \to 12 \to 1236 \to 12346 \to 123456$ in D_2 corresponds to the 2–machine schedule $\begin{smallmatrix} 1 & 3 & 4 & 5 \\ 2 & 6 \end{smallmatrix}$ of length 4 for P_2.

More generally, any path from $\emptyset$ to any node I (identified with the ideal I) corresponds to a schedule for the induced suborder $P \mid I$. Thus the digraph can be used for a recursive computation of an optimal schedule if the objective fulfills a *Recursion Property* of the form

$$f(I) = \min\{f(I') + c(I', I) \mid (I', I) \text{ is an edge of } D\}. \tag{4.11}$$

Here $f(I)$ denotes the optimum value associated with $P \mid I$ (where $f(\emptyset) = 0$), and $c(I', I)$ denotes a cost value assigned to the edge (I', I) of D. This value expresses a minimal *additive increase* in cost when the schedule for $P \mid I$ is constructed from any optimal schedule for $P \mid I'$.

This procedure is known as *dynamic programming*. Equation (4.10) expresses the *optimality principle* of dynamic programming, viz. that any "initial" part of an optimal solution must be optimal for this part. Thus the global solution can be computed along the acyclic digraph by successive evaluations of (4.10) in a fashion very much similar to shortest path computations.

The Recursion Property holds e.g. for the *makespan* $C_{\max}$ in the m–machine problem with unit processing times, for the *weighted flowtime* $\sum_j w_j C_j$ in the 1–machine problem, and for the *jump number*. For more objectives, see [SiS,MS2].

For the makespan, the edge cost $c(I', I)$ is 1 for each edge and does not depend on the optimal solution for $P \mid I'$. For the weighted flowtime on 1 machine, the edge cost $c(I', I)$

is given by

$$c(I', I) = w_i \sum_{j \in I} x_j, \text{ where } \{v_i\} = I - I'. \tag{4.12}$$

So it depends on the edge, but not on the optimal solution for $P \mid I'$. Finally, for the jump number, the edge cost is given by

$$c(I', I) = \begin{cases} 0 & \text{if } v_i \in I - I' \text{ is comparable to the last} \\ & \text{element of the optimal solution for } I', \\ 1 & \text{otherwise.} \end{cases} \tag{4.13}$$

In this case, $c(I', I)$ depends both on the edge and on the optimal solutions associated with I'.

So in D_2 from Figure 19, the path $\emptyset \to 12 \to 1234 \to 123456$ corresponds to an optimal schedule $\begin{smallmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{smallmatrix}$ of length 3. If P_1 from Figure 19 has processing times $x_1 = 2, x_2 = x_3 = 1, x_4 = 3$ and weights $w_1 = 1, w_2 = w_3 = 2, w_4 = 1$, then the resulting edge cost together with a *tree* of optimal partial schedules and associated values is represented in Figure 20. For the jump number, a tree of optimal partial schedules with the associated optimum values is presented in Figure 21.

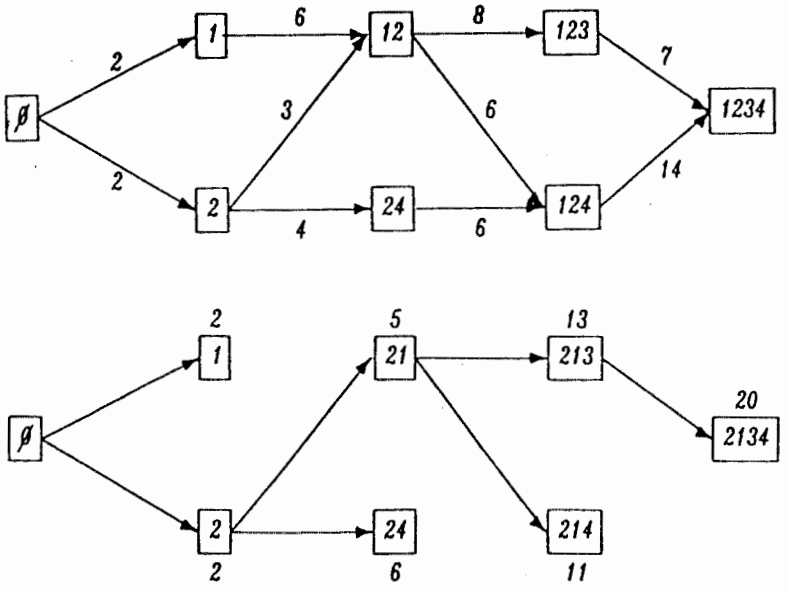


Figure 20: Edge weights and tree of optimal partial schedules for the $1 \mid prec \mid \sum w_j C_j$ problem.

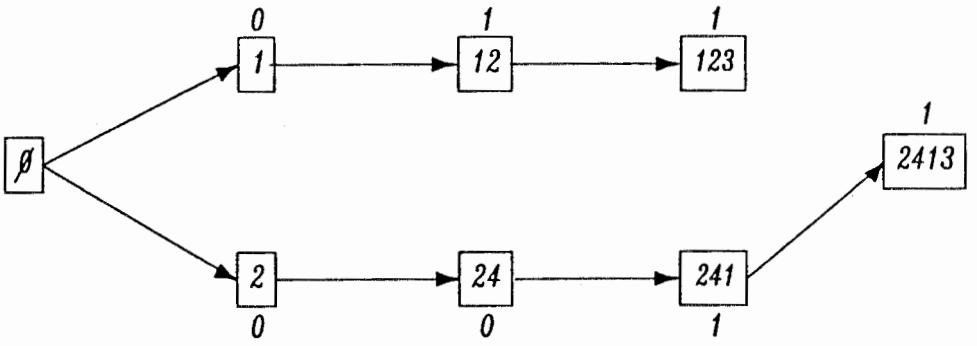


Figure 21: Tree of optimal partial solutions for the jumb number problem

It follows that the dynamic programming approach takes at most as many steps as there are nodes and edges in D. For m–machine schedules, and partial orders of width w there are at most $\binom{w}{m}$ edges entering a node. Each node corresponds to an ideal and so the bound (4.10) applies. Thus the dynamic programming algorithm takes $O(\binom{w}{m} \cdot n^w)$ time alltogether.

So on the class $\mathcal{P}_w$ of partial orders with width $\leq w$, $\binom{w}{m}$ is a constant and one obtains:

4.7 Theorem: *The m–machine problem, the 1–machine weigthed flowtime problem, and the jump number problem can be solved in $O(n^w)$ time on the class $\mathcal{P}_w$.*

More generally, this holds for every objective fulfilling the Recursion Property for which the edge costs can be computed efficiently, see [SiS] for more examples.

This approach of bounding the width can be combined with the substitution decomposition if the objective fulfills in addition to the Recursion Property a *Decomposition Property* stating that an optimal solution on an autonomous set can be "extentend" to an optimal solution of the whole partial order. This implies essentially that an optimal solution can be constructed "along" the canonical decomposition tree or along the inverse of a composition series.

The Decomposition Property holds for the weighted flowtime [Si], the jump number [Hab,FGS,St3], and several other objectives for 1–machine scheduling problems, cf. [MS2] for a systematic treatment. However, it does not hold for the makespan in the m–machine problem, since this problem is (for varying m) already NP–hard on series–parallel partial orders, cf. Section 2.4.

For 1–machine problems, the Decomposition Property yields that the autonomous sets associated with the sons of a prime–type node N can be replaced by optimal linear extensions, thus yielding an extension of $P \mid N$ with $w(P \mid N) \leq k$ if $\mathrm{decw}(P) \leq k$. This means that the width of the partial orders constructed along the tree remains bounded by k if $\mathrm{decw}(P) \leq k$.

So for any prime type node, an optimal sequence can be determined in $O(n^w)$ time by Theorem 4.7. For parallel nodes N with m sons $N_1, \ldots, N_m$ and associated optimal sequences $L_1, \ldots, L_m$, an optimal sequence of N can be obtained by considering successively N_1 and N_2, then $N_1 \cup N_2$ and N_3, etc. At each time the optimal sequence for $N_1 \cup \ldots \cup N_{i-1}$ is merged with L_i to give an optimal sequence of $N_1 \cup \ldots \cup N_i$. Because of Theorem 4.7, this takes $(m-1) \cdot O(n^2) \leq O(n^3)$ time alltogether. Finally, for series nodes N, the sequencing is trivial. This gives [SiS]:

4.8 Theorem: *Let, for a 1-machine problem, the objective have the Recursion Property and the Decomposition Property.*

Then for any partial order $P \in \mathcal{P}_{w \leq k}$ of size n, the problem can be solved in $O(n^{w+1})$ time.

This result extends the applicability of the substitution decomposition considerably in comparison with the class $\mathcal{P}_{d \leq k}$.

We demonstrate this for the weighted flowtime on the partial order P from Figure 16 whose decomposition tree is given in Figure 18. The data and the optimal sequences for the tree nodes are presented in Table 1.

Data

i	1	2	3	4	5	6	7	8	9	10	11	12
x_i	3	2	2	1	3	1	1	2	2	1	3	1
w_i	2	1	2	1	1	3	3	3	1	2	1	1

Sequences produced along the tree

tree node N	type	optimal sequence on N
$\{1,2\}$	P	1,2
$\{1,2,3\}$	S	1,2,3
$\{5,6,7\}$	P	6,7,5
$\{8,\ldots,12\}$	prime	8,10,9,12,11
$\{1,\ldots,12\}$	prime	4,8,10,1,2,3,6,7,9,12,5,11

Table 1: Optimal weighted flowtime schedule for the partial order from Figure 16

These applications from scheduling suggest that counting the number of ideals or linear extensions for the class $\mathcal{P}_{w \leq k}$ of partial orders with bounded decomposition width should also be possible. This is indeed the case and has been shown in [St7, St8]. For bounded width, see also [AC, FS2].

It is open whether the dimension problem and the isomorphism problem can also be solved efficiently on the class $\mathcal{P}_{w \leq k}$. Since both problems are compatible with the substitution decomposition, it would suffice to show their solution for partial orders with bounded width.

This seems to be possible for the isomorphism problem since each isomorphism must

respect the bounded levels. For the dimension problem, however, this seems to be unlikely. Although the dimension of a partial order may not exceed its width [Hi], its calculation is already NP–hard for dimension 3 [Ya]. So it may also be NP–hard for partial orders of bounded width. The reduction proof in [Ya] does not cover this case, since it uses partial orders whose width grows with the size of the problem instance.

5. 2–Dimensional Partial Orders

2–dimensional partial orders provide another generalization of series–parallel partial orders. They were introduced by Dushnik and Miller [DM] in connection with the dimension of a partial order. Different from the previously considered classes of partial orders, the main structural representation is by *lists* (i.e. the 2 linear extensions realizing P) rather than by *trees*, though the canonical decomposition tree from the substitution decomposition also proves to be usefull for recognition and isomorphism testing.

Another fundamental difference to the other classes in this paper is the fact that 2–dimensional partial orders are not characterized by finitely many forbidden substructures [BFR].

5.1 Deflnition And Structural Properties

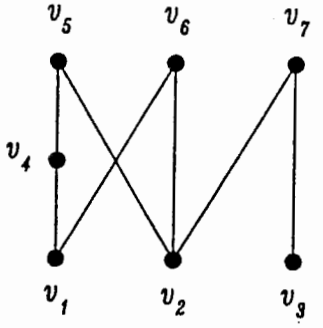
Recall from Section 1.3 that the *dimension* $dim(P)$ of a partial order $P = (V, <)$ is the smallest number of linear extensions $L_1, \ldots, L_k$ of P, $L_i = (V, <_i)$ whose intersection is P, i.e.

$$a < b \Leftrightarrow a <_i b \text{ for } i = 1, \ldots, k. \tag{5.1}$$

Any such system of linear extensions is called a *realizer* of P, and a *minimal realizer* if $k = dim(P)$. A partial order P is called *k–dimensional* ($k \in \mathbb{N}$), if $dim(P) \leq k$.

The notion of dimension has an immediate implication for representing partial orders. If a realizer $\{L_1, \ldots, L_k\}$ of $P = (V, <)$ with $|V| = n$ is known, it can be used for representing P by k lists of length n, where each list corresponds to a linear extension $L_i : v_{i1} < v_{i2} < \ldots v_{in}$ of the realizer. Then any $v \in V$ has a representation as a vector $(v^1, \ldots, v^k) \in \mathbb{R}^k$, where v^i gives its *position (coordinate)* in L_i $(i = 1, \ldots, k)$. If these vectors are known, then any test "$a < b$" can obviously be carried out in $O(k)$ time by componentwise comparisons. This interpretation of the elements of P as points of $\mathbb{R}^k$ with $<_P$ corresponding to the componentwise ordering of $\mathbb{R}^k$ is an equivalent notion of dimension; cf [Or]. That is, the least k such that such an embedding of P into $\mathbb{R}^k$ exists is equal to $dim(P)$.

So if k is a small fixed number, these ideas might be used to obtain a representation of any k–dimensional partial order in $O(k \cdot n) = O(n)$ space with $O(k) = O(1)$ time for answering queries "$a < b$". An example for a 2–dimensional partial order is given in Figure 22.



$L_1 : v_1\ v_4\ v_2\ v_5\ v_6\ v_3\ v_7$

$L_2 : v_3\ v_2\ v_7\ v_1\ v_6\ v_4\ v_5$

$v_1 = \binom{1}{4}, v_2 = \binom{3}{2}, v_3 = \binom{6}{1}, v_4 = \binom{2}{6}, v_5 = \binom{4}{7}, v_6 = \binom{5}{5}, v_7 = \binom{7}{3}$

Figure 22: A 2–dimensional partial order, a minimal realizer and a representation in $\mathbb{R}^2$.

Unfortunately, recognizing k–dimensionality is NP–complete for any fixed $k \geq 3$ [Ya], so that this approach is only promising when a realizer is known or $k \leq 2$. Since recognition is one of the major concerns of this proper, we will restrict to 2–dimensional partial orders in the following.

The main algorithmic characterizations of 2–dimensional partial orders appear already in [DM]. They use the following notions.

A *conjugate* of a partial order $P = (V, <_P)$ is a partial order $Q = (V, <_Q)$ such that

$$a \sim_P b \text{ iff } a \|_Q b, \tag{5.2}$$

i.e. elements comparable in P are incomparable in Q and vice–versa.

A linear extension $L = (V, <_L)$ of $P = (V, <_P)$ is called *non–separating* if

$$u <_L w <_L v, u <_P v \Rightarrow u <_P w \text{ or } w <_P v, \tag{5.3}$$

i.e. if no comparable pair $u <_P v$ of P is separated in L by some element w that is incomparable in P to both u and v.

5.1 Theorem: *The following statements are equivalent:*

(1) P is 2–dimensional.

(2) P has a conjugate Q.

(3) P has a non–separating linear extension L.

We give some hints on the proof. If a minimal realizer $\{L_1, L_2\}$ of P is known, then $\{L_1, L_2^{-1}\}$, where L_2^{-1} is the dual of L_2, is a realizer of a partial order Q which obviously fulfills (5.2), i.e. is a conjugate of P. This shows "(1)⇒ (2)". The conjugate of P from Figure 22 thus obtained is given in Figure 23.

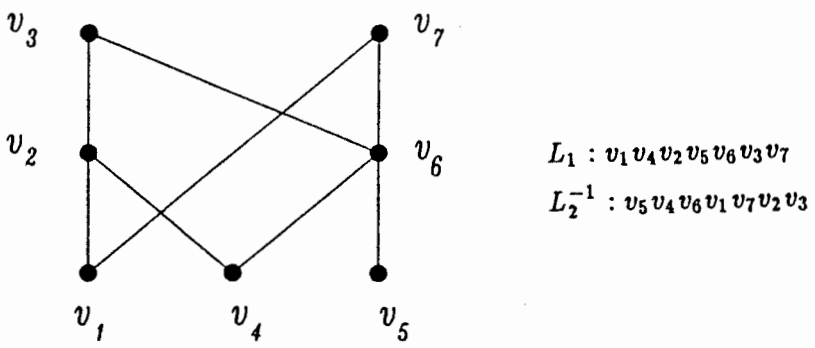


Figure 23: A conjugate of the partial order from Figure 22

If a conjugate Q of P is given, then a non–separating linear extension L of P can be produced by taking *any* linear extension L' of Q and by using L' as a priority list for producing a linear extension of P (i.e. ties are broken according to L'). For instance, the level extension $L' = v_1v_4v_5v_2v_6v_3v_7$ of Q in Figure 23 induces $L = v_1v_4v_2v_5v_6v_3v_7 = L_1$.

Similary, from a non–separating linear extension L of P, a minimal realizer can be obtained by taking $L_1 = L$ and by using L^{-1} as priority list to produce L_2. In the example, we have $L^{-1} = v_7v_3v_6v_5v_2v_4v_1$ and thus obtain $L_2 = v_3v_2v_7v_1v_6v_4v_5$.

The proof of Theorem 5.1 in [DM] is based on the following relationship between a conjugate Q and non–separating linear extensions of P.

5.2 Theorem: *Let $P = (V, <)$ be a partial order. Then $Q = (V, <_Q)$ is a conjugate of P iff $L = (V, <_P \cup <_Q)$ is a non–separating linear extension of P.*

Consider the comparability graph $G(P)$ of a partial order P. Obviously, a conjugate Q of P corresponds to a transitive orientation of the complementary graph $G(P)^c$ of $G(P)$, i.e. $G(Q) = G(P)^c$. This observation together with statement (2) gives the following graph–theoretic characterization of 2–dimensional partial orders [BFR].

5.3 Theorem: *A partial order P is 2–dimensional iff $G(P)^c$ is a comparability graph. In this case, any transitive orientation of $G(P)^c$ corresponds to a conjugate of P.*

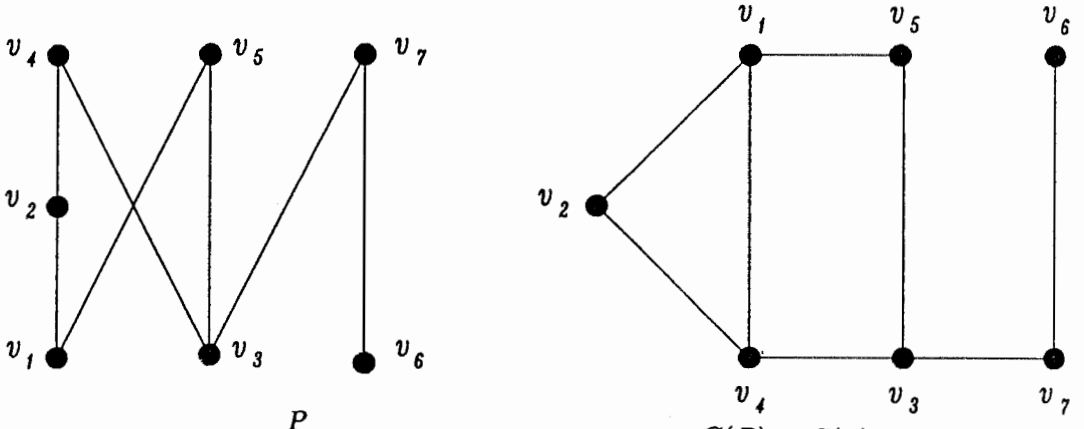
Theorem 5.3 may be rephrased by saying that a graph G is a comparability graph of a 2–dimensional partial order iff G and its complement G^c are comparability graphs. These graphs have been investigated independently under the name *permutation graphs* [PLE,GH], cf. also [Go1]. This name stems from the fact that a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ is a permutation graph iff there is a permutation π on V with

$$(v_i, v_j) \in E \text{ iff } i < j \text{ and } \pi^{-1}(v_i) < \pi^{-1}(v_j), \tag{5.4}$$

where $\pi^{-1}(v_i)$ denotes the position of v_i in π. This characterization is obtained by renumbering the elements of a 2–dimensional partial order in such a way that one of the two linear extensions in a minimal realizer is just $v_1v_2 \ldots v_n$. The other linear extension $v_{i_1}v_{i_2} \ldots v_{i_n}$

then defines the permutation π by letting $\pi(v_j) = v_{i_j}$. For the example from Figure 22, this renumbering, the permutation π and the induced permutation graph $G[\pi]$ are given in Figure 24.

This shows that permutation graphs and 2–dimensional partial orders can be *identified with a permutation* on the ground set.



$$\pi = (\pi(v_1), \ldots, \pi(v_7)) = (v_6, v_3, v_7, v_1, v_5, v_2, v_4)$$

Figure 24: The permutation graph of the 2–dimensional partial order from Figure 22.

There are still several other characterization of 2–dimensional partial orders. One given already in [DM] is based on *containment of intervals*. A partial order $P = (V, <)$ is 2–dimensional iff its elements $v \in V$ can be represented by intervals $(I_v)_{v \in V}$ on the real line such that $u < v$ iff $I_u \subseteq I_v$ for all $u, v \in V$.

A characterization using *planarity and lattice completions* is given in [BFR], cf. also the survey article [KT]. A partial order P is 2–dimensional iff it can be embedded into a *planar* lattice (i.e. a lattice with a planar Hasse diagram). In that case, a conjugate is obtained by taking the left–to–right ordering of the incomparable elements of P defined by the planar lattice diagram.

Another characterization based on *enumerating ideals* [St5] is presented in Section 5.3. An interval representation and an embedding into a planar lattice are given in Figure 25. We close this section with some comments on 2–dimensional partial orders and substitution decomposition. It follows from Hiragushi's formula (4.7) that a partial order is 2–dimensional iff all its factors are 2–dimensional. Interpreted in the canonical decomposition tree, this gives:

5.4 Theorem: *A partial order P is 2–dimensional iff all factors P_N associated with prime type nodes of the canonical decomposition tree of P are 2–dimensional.*

This means that testing for 2–dimensionality can be reduced to testing the prime factors associated with the prime type nodes. This is in fact the basis of the currently

fastest recognition algorithm [Sp4,SV], cf. also Section 5.2.

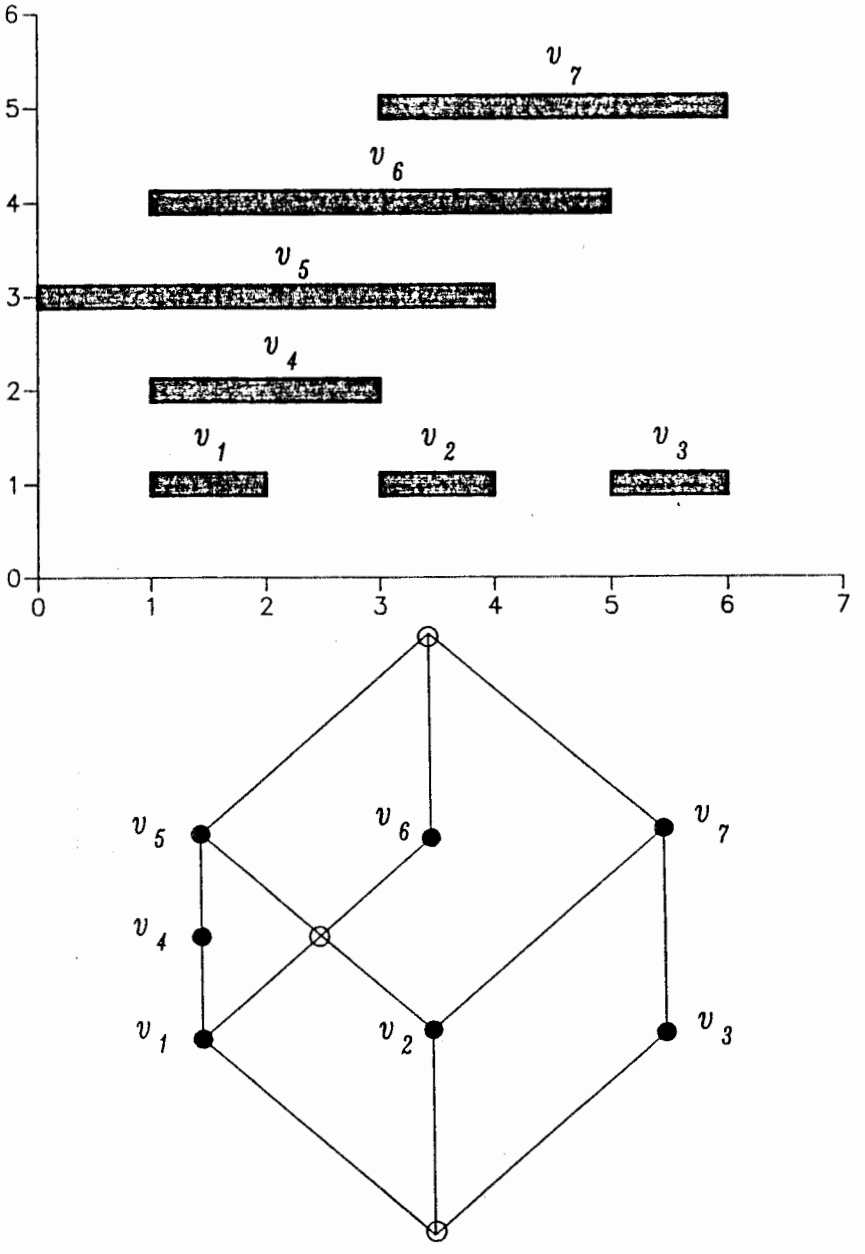


Figure 25: An interval–inclusion representation and an embedding into a planar lattice of the partial order from Figure 22

The prime factors of a 2–dimensional partial orders P have special properties that lead e.g. to polynomial algorithms for isomorphism testing and to methods for counting 2–dimensional partial orders. One of these properties follows immediately from Theorem 5.2 and the fact that the comparabilty graph of a prime partial order P has only two transitive orientations, viz P and P^{-1} (see e.g. [GH,TMS] and also [PLE]).

5.5 Theorem: *A prime 2–dimensional partial order P (with $n \geq 4$ elements) has exactly 2 non–separating linear extensions. Uniqueness is achieved by fixing the order of one incomparable pair $a \|_P b$.*

The structure of prime 2–dimensional partial orders is more closely investigated in [Ste1,Ste2] by exploiting properties of the permutation defining the permutation graph $G(P)$ in the sense of (5.4). Each prime factor is thus subdivided into paths with at most 3 vertices and their complements that are related to each other in a tree structure. This tree structure can be used to refine the decomposition tree to a "hybrid" tree by replacing the prime type nodes by their "refined" tree structure. For details, we refer to [Ste1,Ste2].

The structure of the permutations defining the prime permutation graphs is also used for counting 2–dimensional partial orders [Elz].

5.2 Recognition Algorithms

The early recognition algorithms [PLE,Go1] are based on Theorem 5.3. They use methods for comparability graph recognition and orientation to test wether the incomparability graph $G(P)^c$ of a partial order is a comparability graph (in which case P is 2–dimensional), and, if so, produce a transitive orientation of $G(P)^c$ (i.e. a conjugate of P). This is possible in $O(\delta \cdot e)$ time, where δ and e denote the maximum degree and the number of edges of $G(P)^c$, respectively. An overview of these methods is given in [Mö4]; for details, we refer to [Go1].

A different approach has been followed by Spinrad [Sp4,VS]. He uses the facts that (1) each linear extension of any partial order can be constructed along its canonical decomposition tree (cf.(4.9)), and that (2) 2–dimensionality is equivalent to the existence of a non–separating linear extension (Theorem 5.1).

The algorithm then constructs in the first phase the canonical decomposition tree, and, in the second phase, traverses it in a bottom–up fashion to produce a non–separating linear extension.

So for a tree node N with sons $N_1, \ldots, N_k$, the algorithm has already constructed non–separating linear extensions $L_1, \ldots, L_k$ for the suborders $P|N_1, \ldots, P|N_k$ of P induced by the set of elements in subtrees rooted in $N_1, \ldots, N_k$. It then must construct a non–separating linear extension L of $P|N$.

This is easy if N is a *series* node, since then, $L = L_1 * L_2 * \ldots * L_k$ is the only choice. If N is a *parallel* node, any order $L_{i_1} * L_{i_2} * \ldots * L_{i_k}$ of the L_i is obviously a non–separating linear extension. If N is a *prime–type* node, then a non–separating linear extension L of $P|N$ is obtained by constructing first a non–separating linear extension L_N of the prime factor P_N associated with the tree node N in the sense of Section 4.1, and by substituting the elements of L_N representing the autonomous sets $N_1, \ldots, N_k$ by the linear extensions

$L_1, \ldots, L_k$.

More formally, if $a_1, \ldots, a_k$ are representatives of the autonomous sets associated with the sons $N_1, \ldots, N_k$, and if $P_N = P|\{a_1, \ldots, a_k\}$ is the prime factor associated with N, then $P|N = (P_N)_{a_1,\ldots,a_k}^{P|N_1,\ldots,P|N_k}$, and $L = (L_N)_{a_1,\ldots,a_k}^{L_1,\ldots,L_k}$ is a non–separating linear extension of $P|N$ if L_N and the L_i are non–separating linear extensions of P_N and the $P|N_i$, respectively. This follows immediately from the properties of the substitution operation.

So the main problem the algorithm must solve repeatedly is the construction of a non–separating linear extension for a prime 2–dimensional partial order. This is done by a *refinement scheme* which first finds an "approximate" non–separating linear extension respecting only some order relations $a < b$ that is subsequently "refined" or "improved" to meet more such relations. For details of this rather involved procedure, we refer to [Sp4,SV].

The result of the refinement scheme is a linear extension L of the prime partial order Q considered that is non–separating if Q is 2–dimensional. In order to check this, a second linear extension L' of Q is produced from L by using the construction in the proof of "(3) $\Rightarrow$ (1)" in Theorem 5.1. Then Q is 2–dimensional iff $\{L, L'\}$ is a realizer of Q.

The total time required by the algorithm consists of the time for computing the canonical decomposition tree and for carrying out the steps described above. Here the refinement scheme dominates the running time. Spinrad shows that this can be done in $O(k^2)$ time for a prime partial order with k elements. Using induction on the depth of the decomposition tree, this gives $O(n^2)$ total time for a partial order with n elements. Since the decomposition tree can also be obtained in $O(n^2)$ time (Section 4.1), one obtains:

5.6 Theorem: *Recognizing a 2–dimensional partial order with n elements and producing a minimal realizer can be done in $O(n^2)$ time.*

5.3 Some Applications

Though there are many graph–theoretic applications of permutation graphs (see [Go1] for an overview), there seem to be only few applications exploiting the 2–dimensionality of partial orders, in particular in the area of scheduling and sequencing. The main order–theoretic applications I am aware of concern *counting ideals* with some consequences for scheduling and sorting, *testing isomorphism*, and *constructing edge diagrams* with a minimum number of dummies.

The ideals of a 2–dimensional partial order have first been considered in [St1] in connection with the dynamic programming approach of Section 4.2 for the $1\,|\text{prec}|\,\sum w_j C_j$ scheduling problem. There exists a surprisingly simple labeling scheme for generating and counting all ideals. However, the applications to scheduling are limited, since the number of ideals may be exponential in the size of the ground set. Instead, this scheme has some direct consequences for searching in 2–dimensional partial orders, as is demonstrated in [St5].

Let $P = (V, <)$ be a 2-dimensional partial order and let L be a non-separating linear extension of P. Assume w.l.o.g. that $L = v_1 v_2 v_3 \dots v_n$. Then the *labeling scheme* assigns labels $L(v_i)$ to each element v_i according to

$$\text{(5.5)} \qquad \begin{cases} L(v_1) = 1 \\ L(v_i) = 1 + \sum_{j<i, v_j \| _P v_i} L(v_j) \quad (i = 2, \dots, n). \end{cases}$$

For the partial order P from Figure 24, this gives $L(v_1) = L(v_2) = L(v_4) = 1, L(v_3) = L(v_5) = 3, L(v_6) = 10$, and $L(v_7) = 7$.

5.7 Theorem: *The labels $L(v_i)$ produced by the labeling scheme for a non-separating linear extension have the following properties:*

(1) $L(v_i)$ is the number of order ideals containing v_i and otherwise only elements v_j with $j < i$.

(2) $\sum_{i=1}^{n} L(v_i)$ is the number N of all non-empty order ideals of P.

The proof makes essential use of the fact that $L = v_1 \dots v_n$ is non-separating. In fact, it is shown in [St1] that Theorem 5.7 does not remain valid if P is not 2-dimensional or a separating linear extension is chosen for the scheme.

5.8 Corollary: *Counting the number of order ideals can be done in polynomial time for 2-dimensional partial orders.*

It is still an open problem wether this is also true for the number of linear extensions or other coefficients of the order polynomial.

Using Corollary 5.8 and the fact that the number $N(v_i)$ of all order ideals containing v_i is related to the labels $L(v_j)$ by

$$\text{(5.6)} \qquad N(v_i) \leq 1 + N - L(v_i),$$

the following result is derived in [St5]:

5.9 Theorem: *Every 2-dimensional partial order $P = (V, <)$ contains a δ- central element with $\delta = 1/4$, i.e. there is some $v \in V$ with $1/4 \leq N(v)/N \leq 3/4$. Such an element v can be found in $O(|V|^3)$ time.*

This generalizes the result of [FLRT] for series-parallel partial orders. In fact, the labeling scheme used in [FLRT] and also in [St2] can be interpreted as a special instance of the scheme (5.5) by using a particular non-separating linear extension.

The *isomorphism problem* was first solved in [Co] for permutation graphs instead of 2-dimensional partial orders. The algorithm first performs successive identification of related or unrelated twins as for cograph isomorphism testing. In a second phase it exploits the structure of transitive orientations on the resulting "irreducible" graph without twins. So it essentially uses some "weak" aspects of the substitution decomposition together with a

"stronger" version of Theorem 5.5. Its running time is $O(n^3)$ for a permutation graph with n vertices.

More efficient algorithms for testing isomorphism of both permutation graphs and 2–dimensional partial orders were developed by Spinrad [Sp1,Sp4,SV]. It uses the canonical decomposition tree and Theorem 5.5. Their combination gives that two 2–dimensional partial orders P_1, P_2 are isomorphic iff their canonical decomposition trees T_1, T_2 are isomorphic as labeled trees, and all prime factors of corresponding prime type nodes N_1, N_2 of T_1, T_2 have the same non–separating linear extension. This latter condition can be tested by the same refinement scheme as in the recognition algorithm that constructs such a linear extension. This gives:

5.10 Theorem: *Isomorphism testing for 2–dimensional orders with n elements can be done in $O(n^2)$ time.*

The *dummy minimization problem* for 2–dimensional partial orders can be solved in polynomial time [Sp3] for the case that no extra nodes in the edge diagram (other than those adjacent with real activities) are allowed. This problem is NP–hard for arbitrary partial orders [KD].

A recent application concerning the construction of a *minimum chain decomposition* when the partial order is represented by points in the plane with the natural order in $\mathbb{R}^2$ has been considered in [Sup]. He presents an algorithm that traverses the points in increasing x–coordinates and puts the currently visited point $v_i = (x_i, y_i)$ into an already existing chain C if $y_i \geq y$ for all $v = (x, y) \in C$, and if C contains a point with higher y–coordinate than all other already existing chains in which v_i could be put. Otherwise, a new chain $C = \{v_i\}$ is created. This algorithm works in $O(nlogn)$ time for n points, while the fastest known previous algorithm work in $O(n^2)$ time. Instead of working with an embedding into $\mathbb{R}^2$, they construct a minimum antichain partition on a conjugate, which is possible in $O(n^2)$ time, see e.g. [Mö4]. An example of the algorithm is presented in Figure 26.

Since 2–dimensional partial orders can be identified with permutations on the ground set, it is possible to express problems dealing with permutations as problems on 2–dimensional partial orders and vice–versa. In some cases, results obtained for one viewpoint may be useful also for the other viewpoint.

For instance, counting permutations leads to results for counting 2–dimensional partial orders [Elz]. In the other direction, Dilworth's theorem restricted to 2–dimensional partial orders states (in the permutation viewpoint) in view of (5.4) that the minimum number of increasing subsequences that partition π is equal to the maximum length of a decreasing subsequences of π. This has been shown directly in [Kal]. Other applications to permutations are contained in [BK2].

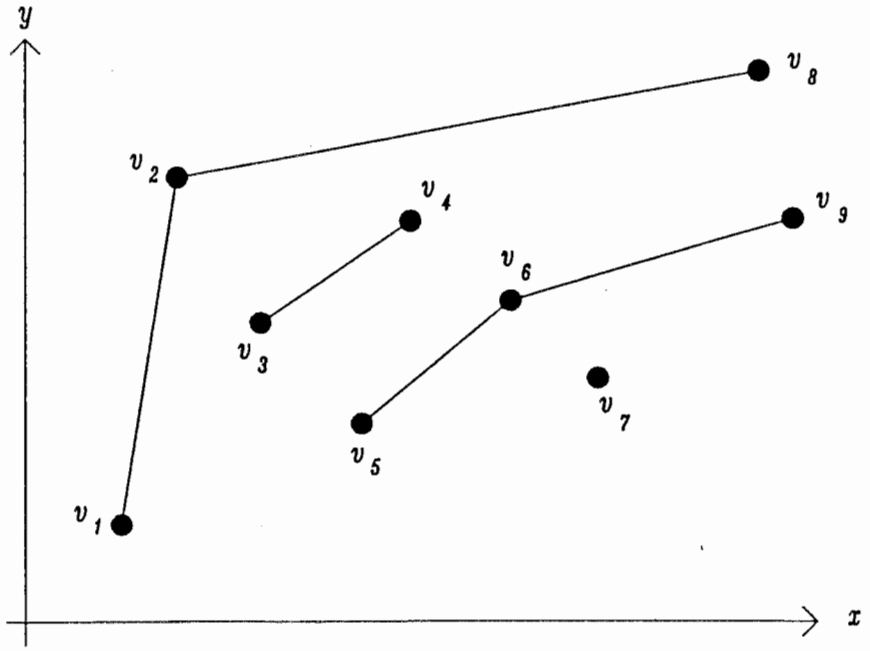


Figure 26: A set of points and a minimum chain partition

5.4 Bipartite 2–Dimensional Partial Orders

Bipartite 2–dimensional partial orders are 2–dimensional partial orders of height 1, i.e. with only 2 levels. These partial orders and their associated comparability graphs have recently been investigated with respect to (mostly graphtheoretic) problems that are NP–hard on arbitrary bipartite graphs [BK1,Ste1,Sp1]. They have a very simple characterization that has been obtained independly in [Sp1,BK1].

5.11 Theorem: *Let $P = (V, <)$ be a bipartite partial order with lower level $V_1 = Min(P)$ and upper level $V_2 = V - V_1$.*

P is 2–dimensional iff there exist an ordering $a_1, a_2, \ldots, a_{n_1}$ of V_1 and $b_1, b_2, \ldots, b_{n_2}$ of V_2 such that

$$\begin{cases} a_i < b_k,\ a_j < b_l & \text{with } i < j \text{ and } k > l \\ \text{implies } a_i < b_l & \text{and } a_j < b_k. \end{cases} \tag{5.7}$$

Such an ordering is called a *strong ordering* for a bipartite partial order. It may be equivalently expressed by certain "convexity" and "monotonicity" conditions of the *leftmost* predecessor l_i and the *rightmost* predecessor r_i of $b_i \in V_2$ with respect to the ordering on V_1.

5.12 Corollary: *A numbering $a_1, \ldots, a_{n_1}, b_1, \ldots, b_{n_2}$ of the levels of a bipartite partial order is a strong ordering iff*

(1) For each b_i, $Pred(b_i) = [l_i, r_i]$ is an interval with respect to $a_1, \ldots, a_{n_1}$. (convexity condition.)

(2) The left and right endpoints of these intervals form nondecreasing sequences, i.e.

$$l_1 \leq l_2 \leq \ldots \leq l_{n_2} \quad \text{and} \quad r_1 \leq r_2 \leq \ldots \leq r_{n_2}$$

(monotonicity condition)

Two examples are given in Figure 27.

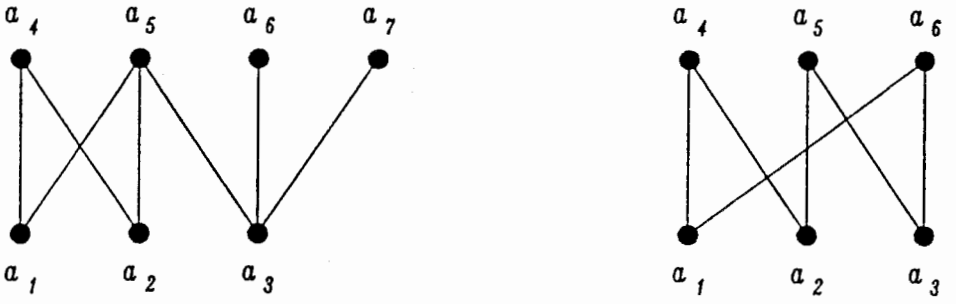


Figure 27: A 2–dimensional and a non 2–dimensional bipartite partial order.

Based on these conditions, a linear time recognition algorithm for bipartite 2–dimensional partial orders has been developed in [Sp2].

The convexity condition implies the existence of so–called perfect elimination schemes that can be used to solve matching problems on bipartite graphs with the convexity condition in a simple and efficient way [DGS].

The convexity and the monotonicity condition together imply that if there is an M–alternating cycle with respect to a matching M, then there is also an M–alternating cycle of length 4 [StS]. This result, together with a relationship between matchings and jumps in linear extensions [CCMP] is used in [StS] to derive a polynomial algorithm for the jump number problem on bipartite 2–dimensional partial orders.

This problem is NP–hard for arbitrary bipartite partial orders [Pu], but still open for 2–dimensional partial orders. For more details about the jump number problem on these classes, we refer to [BH] in this volume.

6. Interval Orders

Interval orders and interval graphs model the sequential and overlapping structure of a set of intervals of the real line. This is why they have many applications dealing with overlapping and consecutiveness such as the gene structure in molecular biology, seriation

in archeology, preference and indifference relations in measurement theory, and consecutive retrieval and VLSI channel routing in computer science.

Due to this wide range of applications, there are several good monographs and survey articles available on interval orders and interval graphs, the most recent being [Go1,Mö4,Go2,Fi2]. The presentation here will therfore be largely "supplementary", and deal mostly with algorithmic aspects and applications not covered in the mentioned monographs and surveys. In particular, it will put the main emphasis on interval orders and give only occasionally hints on interval graphs.

Compared with the other classes of partial orders covered in this article, interval orders do not generalize any of these classes. However, they still have some common features with them such as the existence of linear representations (linear in the number of elements), a characterization by forbidden substructures, and a polynomial isomorphism testing based on a tree representation that can be derived from the substitution decomposition.

6.1 Definition And Structural Properties

Let V be a finite set and $(I_v)_{v\in V}$ be a collection of (not necessarily distinct) intervals I_v of a linearly ordered universe (such as the real line). Such a collection of intervals $(I_v)_{v\in V}$ defines a partial order $P = (V, <)$ on V by putting

(6.1) $$u < v \Leftrightarrow I_u \text{ is entirely to the left of } I_v.$$

It also defines an undirected graph $G = (V, E)$ on V by putting

(6.2) $$(u, v) \in E \Leftrightarrow I_u \text{ and } I_v \text{ overlap (i.e. } I_u \cap I_v \neq \emptyset).$$

A partial order P and a graph G obtained in that way are called an *interval order* and an *interval graph*, respectively, and an associated collection of intervals $(I_v)_{v\in V}$ is called an *interval representation* of P or G. Examples are given in Figure 28.

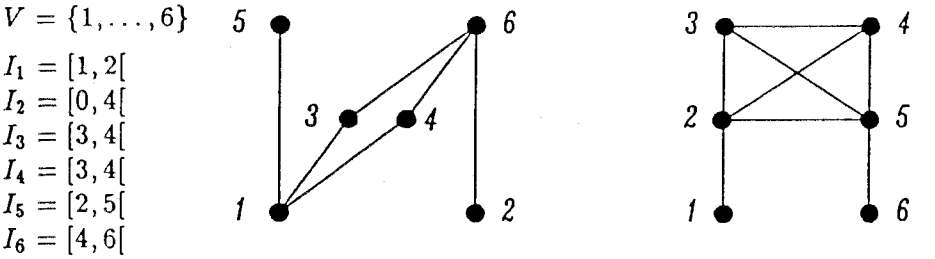


Figure 28: A collection of intervals and associated interval graph G and interval order P

From these definitions, one obtains immediately:

6.1 Proposition: *G is an interval graph iff it is the incomparability graph of an interval order, i.e. iff $G = G(P)^c$ for some interval order P.*

An interval order P with $G = G(P)^c$ is called *compatible* with the interval graph G or an *associated* interval order of G. Many applications deal with the question to construct for a given interval graph G an interval order P that is compatible with G and possibly fulfills some additional constraints, cf. the *seriation problem with side constraints* in Section 6.3.

Because of this close connection, interval graphs and interval orders can be characterized by similar properties. We will start with interval orders.

6.2 Theorem: *Let $P = (V, <)$ be a partial order. The following statements are equivalent:*

(1) P is an interval order.

(2) P does not contain a suborder isomorphic to P_1 from Figure 8.

(3) The maximal antichains of P can be linearly ordered such that, for each vertex v, the maximal antichains containing v occur consecutively.

(4) The sets of predecessors $Pred(v) := \{u \in V | u < v\}, v \in V$, are linearly ordered with regard to inclusion.

(4)' The sets of successors $Suc(v) := \{u \in V | v < u\}, v \in V$, are linearly ordered with regard to inclusion.

The equivalence (1) $\Leftrightarrow$ (2) is due to [Fi1]. For an elegant proof of (2) $\Leftrightarrow$ (4) cf. [Bo]. In both cases, the implications (1) $\Rightarrow$ (2) and (2) $\Rightarrow$ (4) are obvious, as is the equivalence (4) $\Leftrightarrow$ (4)'. Concerning statement (3), again (1) $\Rightarrow$ (3) is obvious by considering an interval representation of P. In the converse direction let $A_1, \ldots, A_m$ be a numbering of the antichains in the sense of (3), and let f_v and l_v denote the first and last index i with $v \in A_i$, respectively. Then $([f_v, l_v[)_{v \in V}$ defines an interval representation of P.

Combining Proposition 6.1 and Theorem 6.2 yields similar characterizations of interval graphs, which were already obtained in [GH] without considering interval orders.

6.3 Theorem: *Let G be an undirected graph. The following statements are equivalent:*

(1) G is an interval graph.

(2) G^c is a comparability graph and G contains no induced subgraph isomorphic to C_4 (the cycle with 4 vertices).

(3) The maximal cliques of G can be linearly ordered such that, for each vertex v, the maximal cliques containing v occur consecutively.

Statement (3) of Theorems 6.2 and 6.3 can be formulated as a property of 0,1–matrices. A matrix with 0,1–entries is said to have the *consecutive 1's property for columns* if its rows can be permuted in such a way that the 1's in each column occur consecutively. Theorem 6.2 and 6.3 then state that the *antichain matrix* (maximal antichains–versus–vertices incidence

matrix) of a partial order P [similarly, the *clique matrix* of a graph G] has the consecutive 1's property for columns iff P is an interval order [iff G is an interval graph]. This result is due to [FuG] for interval graphs.

An ordering of the maximal cliques in the sense of Theorem 6.3 (3) is called a consecutive arrangement. They are directly related to the interval orders compatible with G.

6.4 Theorem: *There is a 1–1 correspondence between consecutive arrangements of the maximal cliques of an interval graph G and the interval orders P with $G = G(P)^c$. In particular, the number of different consecutive arrangements is equal to the number of transitive orientations of G^c.*

This follows easily by observing that any consecutive arrangement $C_1, \ldots, C_m$ of the maximal cliques of G induces an interval representation $(I_v)_{v \in V}$ of G by putting $I_v = [f_v, l_v[$, where f_v and l_v denote the *first* and *last* index i with $v \in C_i$. This in turn induces an interval order compatible with G. Conversely, for any interval order P compatible with G, there is only one consecutive arrangement of the maximal cliques of G yielding an interval representation of P in the above sense.

As a consequence of Theorem 6.4, the seriation problem with side constraints can be solved by constructing a particular consecutive arrangement of the maximal cliques of the given interval graph G.

All these consecutive arrangements can be efficiently represented in a tree structure in $O(n)$ space for a graph with n vertices. This structure, the so–called *MPQ–tree* is obtained in [KM1,KM2] by a modification of the PQ–trees developed in [BL] as a data structure for handling permutations with constraints of consecutiveness.

An MPQ–tree is a rooted tree T with the two types of inner nodes: *P–nodes* and *Q–nodes*, which will be represented by circles and rectangles, respectively. A Q–node is subdivided into as many *sections* as it has sons. The leaves, the P–nodes and the sections of the Q–nodes are assigned sets of vertices (possibly empty) of the interval graph $G = (V, E)$ to be represented by the MPQ–tree T.

The idea behind this assignement of vertex sets to tree nodes and sections is to obtain a representation of the maximal cliques $C_1, \ldots, C_m$ of G with the following properties:

(6.3) Every maximal clique corresponds to a path from a leaf to the root (i.e. is the union of the sets along the path), where each vertex v is as close as possible to the root.

(6.4) The left–to–right ordering of the leaves defines a consecutive arrangement of the maximal cliques

(6.5) Any permutation of the sons of a P–node preserves (6.4).

(6.6) { The reversal of the order of the sons of a Q–node preserves (6.4)

(6.7) { Operations (6.5) and (6.6) give all consecutive arrangements of the maximal cliques of G

It follows that the PQ–tree representation of interval graphs introduced in [BL], where the maximal cliques are the leaves of the tree, fulfills (6.5)–(6.7), but not (6.3).

So the MPQ–tree has the same structure as the PQ–tree, but with sets assigned to all nodes and sections of Q–nodes.

For a P–node N or a leaf, this sets consists of those vertices of G that are contained in *all* maximal cliques of the induced subgraph of G represented by the subtree with root N, but in *no* other clique.

For a Q–node Q, the definition is more involved. Let $Q_1, \ldots, Q_m$ be the sons (in consecutive order) of Q, and let T_i be the subtree of T with root Q_i. We identify T_i with the subgraph of G represented by T_i. We then assign a set S_i (a so–called *section*) to Q for each Q_i. Section S_i consists of all vertices that are contained in all maximal cliques of T_i *and* some other T_j, but *not* in any clique belonging to some other subtree of T that is not below Q. Since the maximal cliques containing a fixed vertex occur consecutively in T, it suffices to represent a vertex v belonging to several sections only in the leftmost and rightmost section in which it appears. This convention assures that T represents G in $O(|V|)$ space, whereas PQ–trees require $O(|V| + |E|)$ space since each maximal clique is stored separately. An example MPQ–tree is given in Figure 29.

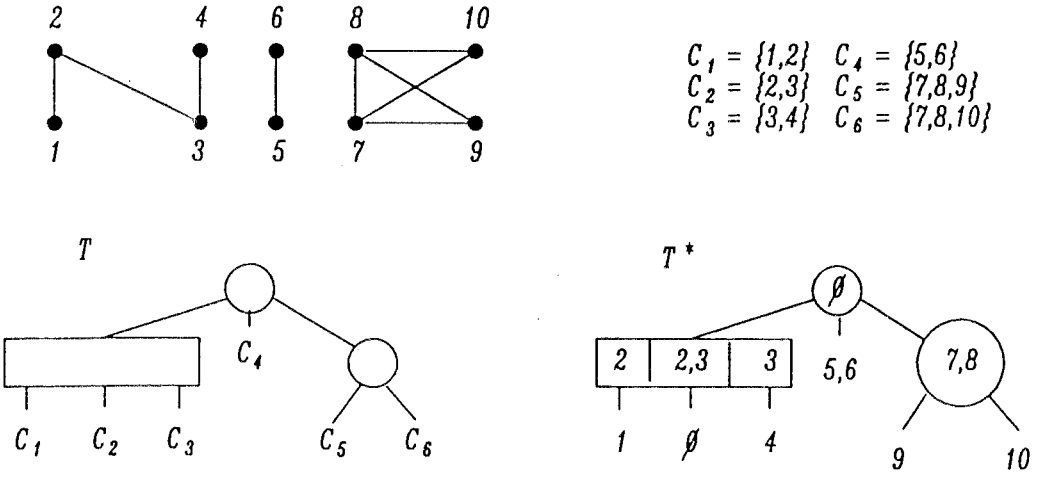


Figure 29: An interval graph G, its PQ–tree T, and its modified PQ–tree T^*

6.5 Theorem: *The MPQ–tree represents an interval graph $G = (V, E)$ and all associated interval orders in $O(|V|)$ space.*

The MPQ-tree has a close relationship with the substitution decomposition [KM1,Ko]. It can be defined by a recursive decomposition scheme similar to that leading to the canonical decomposition tree in Section 4.1. The main point is that one case occuring in the Decomposition Theorem 4.2 is rather uninteresting since an interval order cannot contain the forbidden suborder in Theorem 6.2 (2). This implies immediately:

6.6 Proposition: *If P is interval order then in its canonical decomposition tree, every parallel node has at most one son that is not a leaf.*

So essentially, only series nodes and prime type nodes are important. They correspond exactly to the P-nodes and Q-nodes of the MPQ tree. This also explains the operations (6.5) and (6.6) in P- and Q-nodes. They correspond essentially to permutations of the series components of a series node and to taking the dual of a prime type partial order. (See also the discussion of transitive orientations and decomposition in [Mö4,p.56ff]).

The precise relationship between MPQ-trees and canonical decomposition trees is here expressed by specifying transformation rules that, applied top-down on the MPQ-tree, produce the canonical decomposition tree. These rules are presented in Figure 30. They work as follows. For a P-node N with assigned set V_N and with subtrees $T_1, \ldots, T_k$ (identified with the set of elements of the represented partial order) there are two cases. If $V_N = \emptyset$, N is transformed into a series node $T_1 \cup \ldots \cup T_k$ with subtrees $T_1, \ldots, T_k$. Otherwise, a parallel node is created that has as one (big) son the series node $T_1 \cup \ldots \cup T_k$ with subtrees $T_1, \ldots T_k$, and as additional sons the leaves $\{v\}, v \in V_N$.

For a Q-node N with sections $S_1, \ldots, S_k$ and non-empty subtrees $T_1, \ldots, T_k$, there is a similar subdivision into two cases depending on wether the set A_N of elements contained in the first and the last section is empty or not. In the latter case, the elements $v \in A_N$ and $V_N - A_N$ are made the sons of a parallel node. In both cases, $V_N - A_N$ is transformed into a prime-type node that has as subtrees the original non-empty subtrees $T_1, \ldots, T_k$ and possibly some other subtrees $A_1, \ldots, A_s$. Here $A_1, \ldots, A_s$ are classes of unrelated twins of $V_N - A_N$ that are combined into a parallel node A_i with sons $\{v\}, v \in A_i$ if $|A_i| > 1$. Two vertices u, v are unrelated twins iff they are contained in the same sections, but not in all sections (which can be tested by looking at the leftmost and the rightmost section containing u, v). Since the "twin" relation is an equivalence relation, the A_i are just the equivalence classes of this relation. These classes of unrelated twins are autonomous antichains of $V_N - A_N$ that have no influence on the orientation of $V_N - A_N$. The effect of the transformation rules on the MPQ-tree from Figure 29, and the resulting canonical decomposition tree are presented in Figure 31.

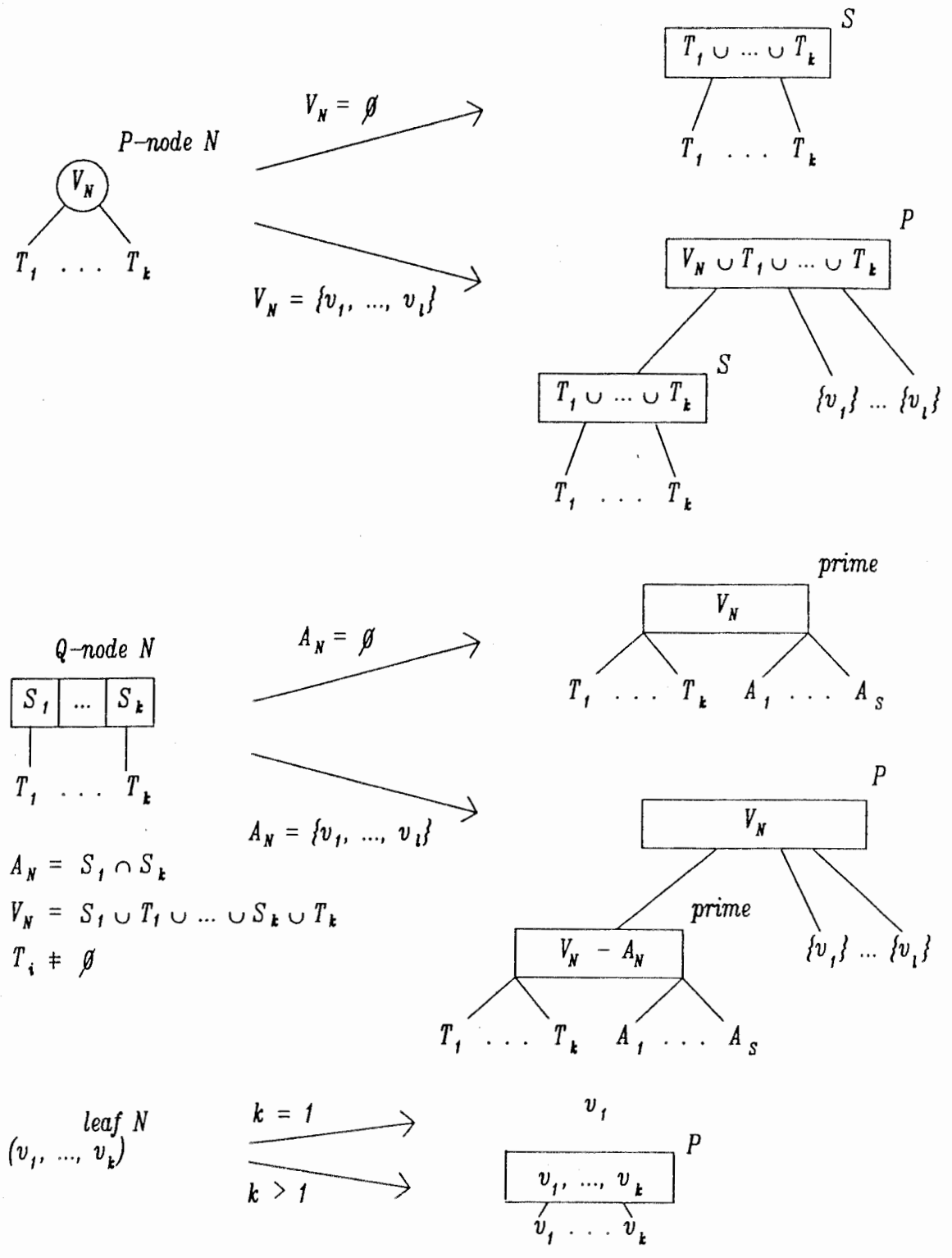


Figure 30: Tree modification rules for transforming the MPQ-tree into the canonical decomposition tree

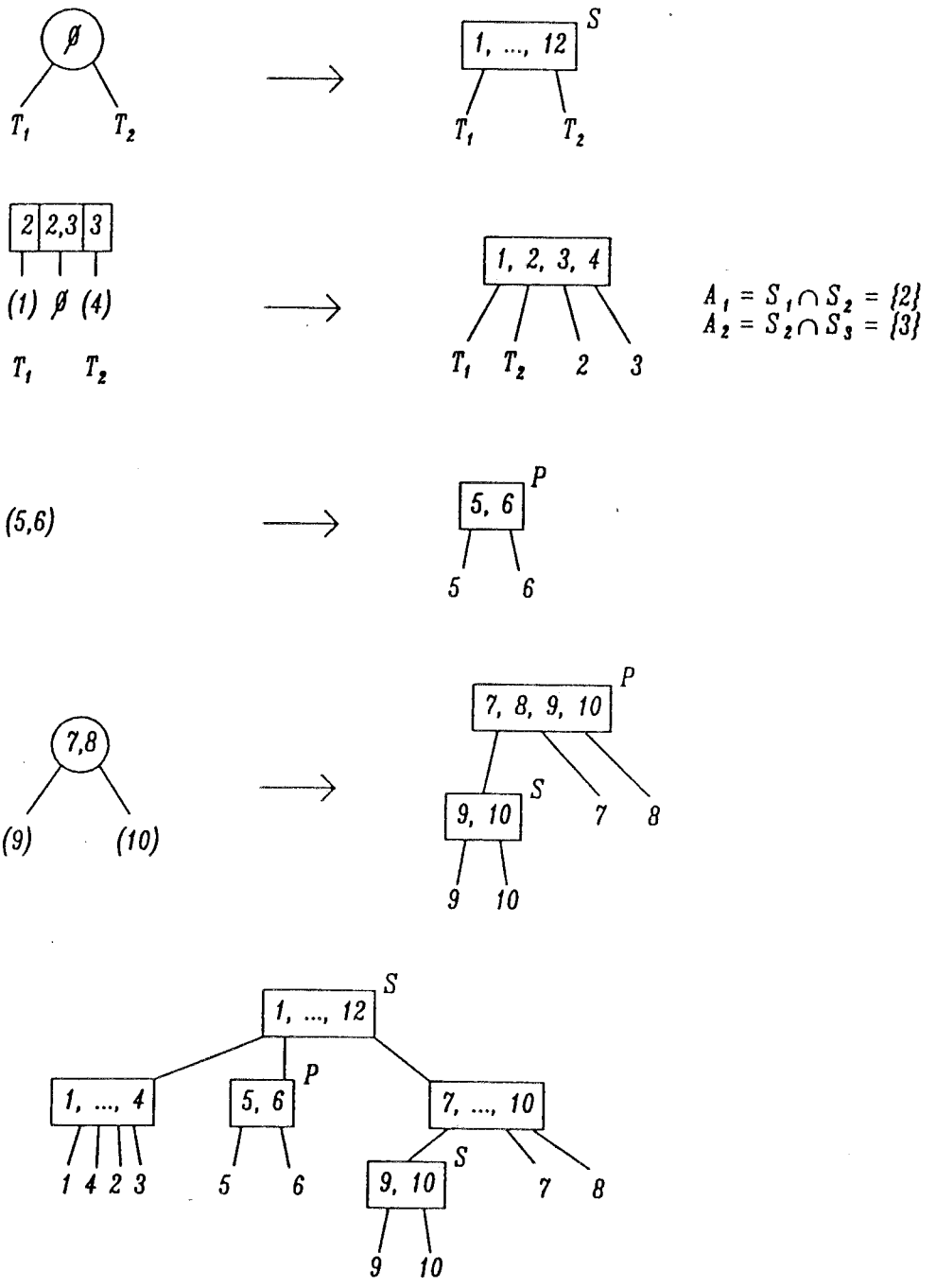


Figure 31: Transforming the MPQ–tree from Figure 29 into the canonical decomposition tree

The MPQ–tree proves to be a powerful tool for algorithmic problems dealing with the construction of particular interval orders compatible with a given interval graph, cf. Section

6.3. It is also the basis for fast incremental recognition algorithms for interval graphs, cf. Section 6.2. It may also prove to be the right tool for various questions related to interval representations of interval graphs such as the minimum number of different interval lengths required and others (see [Fi3] for a treatment of these questions). Two important theoretical consequences of the MPQ–tree follow directly from the relationship to the decomposition tree on the one hand, and the relationship between the substitution decomposition and the structure of comparability graphs (cf. [Mö4] for an overview) on the other.

6.7 Theorem: *Let G be an interval graph with MPQ–tree T. Then the number $t(G)$ of interval orders compatible with G is given by*

$$(6.8) \qquad \begin{cases} t(G) = \Pi_{N \in T}\ t(N), \text{ where} \\ t(N) = \begin{cases} 2 & \text{if } N \text{ is a } Q\text{–node} \\ (\#ofsons)! & \text{if } N \text{ is a } P\text{–node.} \\ 1 & \text{if } N \text{ is a leaf.} \end{cases} \end{cases}$$

In particular, $t(G) = 2$ iff T has height 1 and the root is a Q–node, or a P–node with two sons.

This covers the characterization of interval graphs with $t(G) = 2$ via (maximal) "burried" subgraphs by [Han]. Maximal burried subgraphs just correspond to subtrees of a Q–node with depth ≥ 2.

6.2 Recognition Algorithms

The fastest algorithm for recognizing interval orders has been developped in [PY]. It is based on property (4) or (4)' of Theorem 6.2. Basically, it uses bucket sort to sort the successor sets $Suc(v), v \in V$, with respect to decreasing size and then verifies the inclusion property (4)'. We present here a version that combines this verification with the construction of an interval representation. In order to have fast access to successors, the algorithm uses adjacency arrays instead of adjacency lists, thereby achieving a linear running time at the cost of quadratic space requirement.

6.8 Algorithm: (*Recognition of interval orders*):

Input: A partial order $P = (V, <)$ by a list of adjacency arrays $A(v_i)$ containing the name of the element v_i, its number of successors $\#suc(v_i)$, and the adjacency vector $a_i = (a_{i1}, \ldots, a_{in})$ with $a_{ij} = 1$ if $v_j \in Suc(v_i)$ and 0 otherwise.

Output: An interval representation if P is an interval order, NO otherwise.

Step 1: Sort the list of adjacency arrays according to decreasing numbers $\#suc(v_i)$ using bucket sort. Number the non-empty buckets in the sorted list consecutively as $B_1, B_2, B_3, \ldots$ etc.

Step 2: Define right interval endpoints r_i of v_i as $r_i := k$ if v_i is in bucket B_k.

Step 3: For every v_i scan the sorted list of adjacency arrays until the first $v_j \neq v_i$ with $v_i \notin Suc(v_j)$ is found. For all $v_l \neq v_i$ preceding v_j in the list decrease $\#suc(v_l)$ by 1. Define the left interval endpoint l_i of v_i as $l_i := r_j$.

Step 4: P is an interval order iff $\#suc(v_i) = 0$ for all v_i. Then $(I_{v_i} = [l_i, r_i[)_{i=1,\ldots,n}$ defines an interval representation of P.

Decreasing $\#suc(v_l)$ in Step 3 is equivalent to eliminating v_i from $Suc(v_l)$. Thus the decrease operation verifies condition (4)' in Theorem 6.2. The construction of the interval representation is obvious. All this gives:

6.9 Theorem: *Algorithm 6.8 recognizes wether a partial order P with n elements and m ordered pairs is an interval order and produces an interval representation in $O(n+m)$ time and $O(n^2)$ space.*

An illustration of the algorithm is given in Figure 32 for the partial order P from Figure 28.

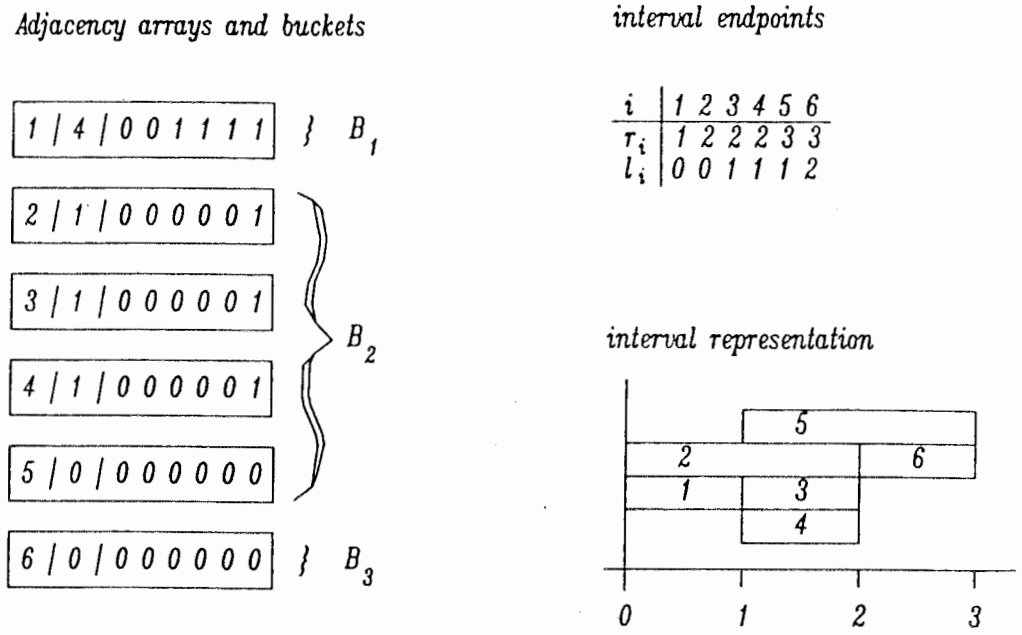


Figure 32: Algorithm 6.8 applied to the partial order from Figure 28

For interval graphs, linear–time recognition algorithms are much more complicated. The first linear–time recognition algorithm was developed by Booth and Lueker [BL] as a particular instance of an algorithm for testing the consecutive ones property of $0-1$ matrices. It uses PQ–trees whose leaves correspond to the maximal cliques of the given graph and modifies the tree successively (starting with the tree of depth 1 with a P–node as root and all the maximal cliques as leaves) by making for each $v \in V$ the maximal cliques containing v consecutive. The maximal cliques are determined beforehand by recognition methods for the larger class of triangulated graphs. For details of this rather involved algorithm, we refer to [BL]. Overviews on the algorithm are presented in [Go1, Mö4].

Using MPQ–trees, a much simpler linear–time recognition algorithm has been developed in [KM2]. It is *incremental* (as the cograph recognition algorithm discussed in Section 2.3), i.e. processes the vertices one at a time and grows the MPQ–tree as the vertices are added. Contrary to the cograph algorithm, however, the order in which the vertices are processed is not arbitrary. They are processed according to *lexicographic breadth–first–search* (LEXBFS), a refinement of the usual breadth–first–search that was introduced in [RTL] for the purpose of recognizing triangulated graphs. The key property of LEXBFS for the recognition of interval graphs is [KM2]:

6.10 Theorem: *If G is an interval graph, then the last vertex in a LEXBFS of G belongs to a maximal clique that, as a path of the MPQ–tree of G, does not go through an inner section of a Q–node.*

Consider now the incremental algorithm. Assume that the vertices from V' have already been considered and that the subgraph $G' := G \mid V'$ of G is represented by its (already constructed) MPQ–tree T'. Let $u \in V - V'$ be the next vertex to be visited in the LEXBFS. Then Theorem 6.10 implies that $G' + u$ $(= G \mid V' \cup \{u\})$ is an interval graph iff u is only adjacent to vertices in a path of T' that does not go through an inner section of a Q–node.

As a consequence, only this path of T' has to be modified to obtain the MPQ–tree T" of $G' + u$. Moreover, no interior sons of Q–nodes need to be considered. This results in fewer and simpler modification rules compared with the PQ–tree algorithm in [BL]. In particular, one of the major headaches for any PQ–tree implementation, viz. the handling of interior sons of Q–nodes, is avoided.

Using the technique of amortized complexity, the linear time bound is achieved.

6.11 Theorem: *There exist incremental algorithms that test wether a graph $G = (V, E)$ is an interval graph and construct its MPQ–tree in $O(|V| + |E|)$ time and $O(|V| + |E|)$ space.*

These principles can be extended (through at the cost of a higher worst case complexity of $O(|V|^2)$ and a more complicated tree updating scheme that also has to consider interior sons of Q–nodes) to general *on–line recognition algorithms*, in which the graph is not known in advance, and vertices are added one at a time [Ko].

6.3 Some Applications

We will restrict ourselves to applications in scheduling, sorting and seriation with side constraints. For other applications, in particular of interval graphs, the reader is referred to [BFM, Bo,Fi2,Go1,Go2,Ro].

There are at least two applications of interval orders in scheduling and sequencing. One concerns the case when the precedence constraints of the scheduling problem form an interval order, while the other considers interval orders as solutions to scheduling problems.

When the precedence constraints of a scheduling problem form an interval order, then the nesting of the successor sets in the sense of Theorem 6.2 (4)' yields a natural priority ordering

$$v_1, v_2, \ldots, v_n \text{ with } Suc(v_1) \supseteq Suc(v_2) \supseteq \ldots \supseteq Suc(v_n). \tag{6.9}$$

Scheduling the elements according to this priorty list then means that jobs that set free more successor jobs are preferred. This idea leads to optimal schedules in the following problems [PY].

6.12 Theorem: *Priority list scheduling according to the list (6.9) yields optimal schedules for minimizing the makespan C_{max} and the flowtime $\sum C_j$ on any number of parallel machines and with arbitrary release dates if the precedence constraints form an interval order and all processing times are equal.*

This can be shown by a simple exchange argument that, starting from an optimal schedule, produces a schedule according to the list (6.9) by a finite number interchanges of adjacent jobs. These interchanges are possible because of the nesting property of the successor sets.

The priority list (6.9) is in fact on interval orders identical with the priority list used in [CG] for the optimal solution of the 2–machine scheduling problem.

An example of the list–scheduling algorithm for the C_{max} criterion on 3 machines is given in Figure 33, where the numbering of the jobs gives the priority ordering (6.9). Equal processing times are essential for Theorem 6.12. For arbitrary processing times, already the 2 machine problem $2\,|\text{prec}|\,C_{max}$ is NP–hard in the strong sense on interval orders [PY]. This result can be extended to the $2\,|\text{prec}|\,\sum C_j$ problem by modifying the reduction from 3–partition given in [PY]. For the *weighted flowtime* $\sum w_j C_j$, the complexity status is still unresolved, even in the 1–machine case.

The priority list (6.9) can sometimes also be used for obtaining approximate solutions to certain problems. For instance, for the *jump number problem*, a relative performance bound of 2 has been obtained in [FS1]. It is still open whether there are better bounds or wether the jump number problem is even polynomial on interval orders.

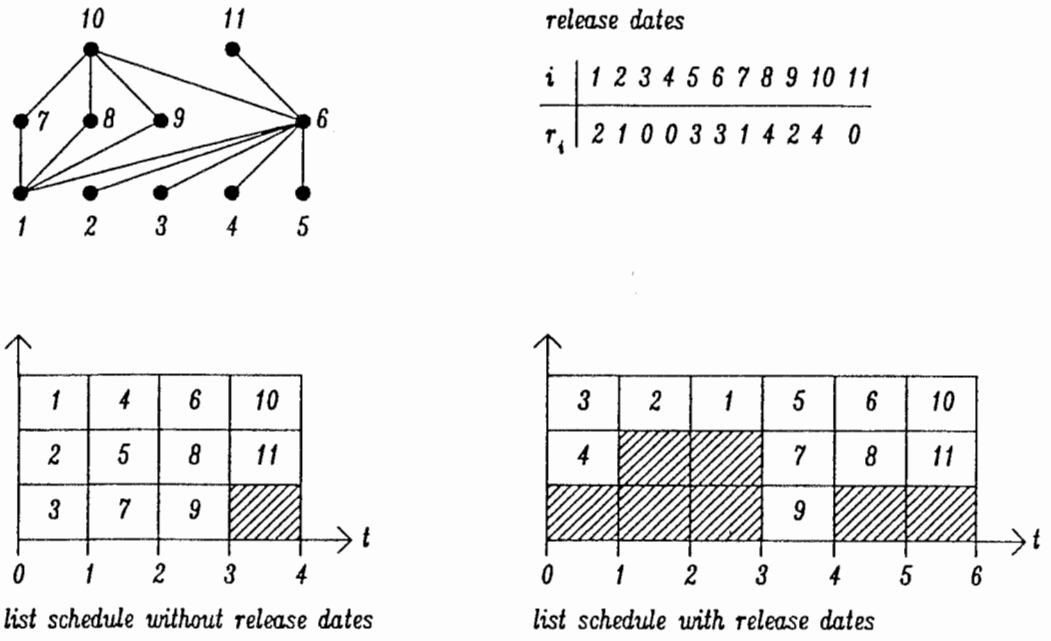


Figure 33: List schedules for interval–ordered jobs

The other main application of interval orders to scheduling is more fundamental and concerns the use of interval orders as a basic instrument to investigate the structure of scheduling problems.

In this so–called *order–theoretic approach* to scheduling, feasible solutions of a scheduling problem are viewed as *interval order extensions* of the partial order P_o of precedence constraints. This interpretation is possible since every (non–preemptive) feasible schedule may be viewed as an interval representation of an *interval order* P that *extends* P_o (i.e. $u <_{P_o} v \Rightarrow u <_P v$). The *earliest start schedules* of these interval orders then constitute a finite set of schedules containing an optimal schedule.

This means that optimal schedules can be obtained by constructing an appropriate (interval) extension of the given partial order P_o of precedence constraints. This is a algorithmic task in the set $\mathcal{E}(P_o)$ of all extensions of P_o. $\mathcal{E}(P_o)$ is a partial order under the *extension* or *inclusion* relation " $\subseteq$ " defined by (2.21) and even a lattice (the *extension lattice*) if a greatest element is adjoined.

The usual scheduling objectives fulfill a *monotonicity property* on $\mathcal{E}(P_o)$, i.e.

$$P_1 \subseteq P_2 \Rightarrow \tau(P_1, x) \leq \tau(P_2, x) \tag{6.10}$$

where $\tau(P, x)$ denotes the cost for the earliest start schedule associated with P and the processing times x, cf. (1.5). This monotonicity property also holds for other objectives such as the bump number or the negative of the jump number. It is the basis for a number

of branch-and-bound algorithms for constructing an optimal extension in $\mathcal{E}(P_o)$.

The notion of feasiblity for schedules can be extended to extensions P of P_o by requiring that any antichain of P can be scheduled simultaneously in a feasible schedule. That this extends the feasibility notions for schedules follows from the fact that the simultaneously scheduled sets of jobs in a schedule correspond to the antichains of the associated interval order.

For the usual (i.e. time–independent) resource constraints, this implies that partial orders $P \in \mathcal{E}(P_o)$ are feasible iff they have "small" antichains of resource–consuming jobs. As a consequence, the set of $\mathcal{F}(P_o)$ of *feasible* extensions of P_o is a *filter* of $\mathcal{E}(P_o)$. For instance, for m–machine problems, $\mathcal{F}(P_o)$ consists of all extensions P of P_o with width $w(P) \leq m$.

A more detailed overview on the order–theoretic approach to scheduling problems is given in [Mö4,p.85ff]. For recent publications, see [Ra2,BMR2]. This approach can be extended to temporal constraints involving arbitrary minimal and maximal time lags between jobs (instead of just precedence constraints), cf. [BMR2].

This approach has also led to investigations on how the set of interval orders is embedded into the *lattice* $\mathcal{E}(V)$ *of all partial orders* on V (ordered under inclusion as $\mathcal{E}(P_o)$.) An easy, but interesting result is

6.13 Theorem: *For any interval order P on V, there exists a maximal chain $P_o \subseteq P_1 \subseteq \ldots \subseteq P_k = P$ in $\mathcal{E}(V)$ that consists entirely of interval orders.*

In other words, each interval order P can be constructed from the totally unordered set P_o by successively adding a single precedence constraint "$a < b$" such that all intermediate partial orders $P_1, P_2, \ldots$ are also interval orders. This is easily obtained by taking an interval representation $(I_v)_{v \in V}$ of P and by adding precedence constraints in the order of increasing left endpoints of intervals. An example is given in Figure 34.

The order–theoretic approach to scheduling shows that, for any vector $x = (x_1, \ldots, x_n)$ of processing times x_i for $v_i \in V$ and any partial order P on V, there exists an interval order P' such that the associated earliest start schedules $ES_P[x]$ and $ES_{P'}[x]$ coincide.

So in a certain sense, interval orders are *dense* in $\mathcal{E}(V)$ and *approximate* every other partial order. In fact, the set of interval orders can be characterized as the *least* set of partial orders with this approximation property. This follows from two recent characterization results for interval orders with respect to their embedding into $\mathcal{E}(V)$ in [Ra3]. They are stated in Theorem 6.14 below. We need the following notation.

A partial order P induces two filters in $\mathcal{E}(V)$. First, the filter $\mathcal{F}_1 = \mathcal{E}(P)$ of all its extensions, and second, the filter $\mathcal{F}_2$ of all partial orders not contained in P, i.e. $\mathcal{F}_2 = \mathcal{E}(V) - \{P' \in \mathcal{E}(V) \mid P' \subseteq P\}$. For any vector x of processing times and any scheduling objective τ that is non–decreasing in the completion times of a schedule (so–called *regular* cost function), one may define the *cost* $\tau(P, x)$ of P as the cost of the associated earliest

start schedule $ES_P[x]$, i.e. $\tau(P,x) = \tau(ES_P[x] + x)$; cf. Section 1.3 for details. The characterization results in [Ra3] then deal with the minimization of $\tau(P',x)$ on the ideals $\mathcal{F}_1$ and $\mathcal{F}_2$ generated by a partial order P.

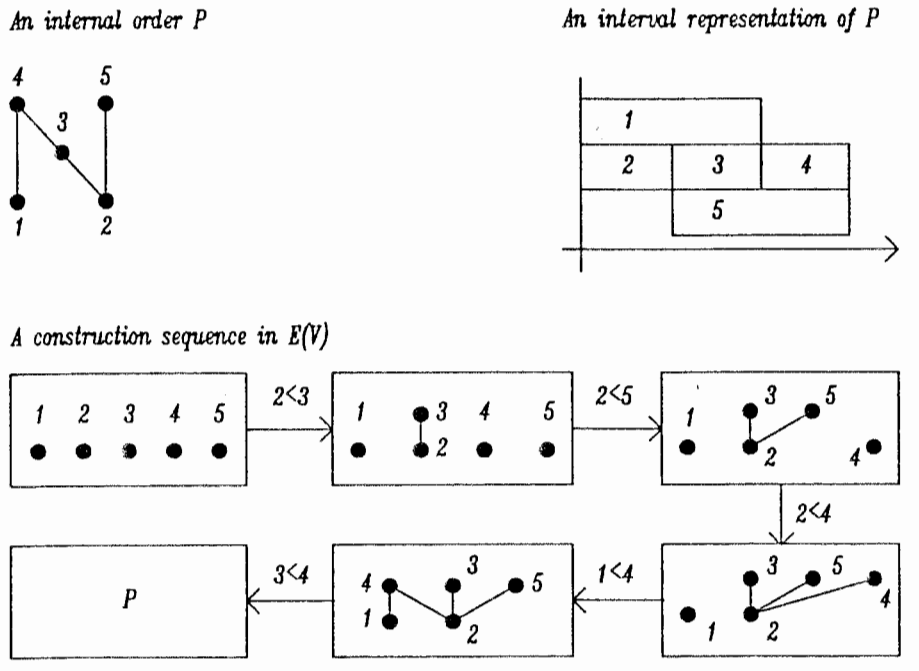


Figure 34: An illustration of Theorem 6.13

6.14 Theorem: *The following statements are equivalent:*

(1) P is an interval order.

(2) There is a vector x of processing times such that $C_{max}(P,x) < C_{max}(P',x)$ for all $P' \neq P$ with $P \subseteq P'$.

(3) There are a vector x of processing times and a regular cost function τ such that $\tau(P,x) < \tau(P',x)$ for all P' with $P' \not\subseteq P$.

Applications concerning *searching and sorting* are presented in [FLRT]. They show that the number $N(u)$ of ideals containing a given element u and the number of all order ideals can be determined in polynomial time by exploiting the nesting of the successor sets in the sense of Theorem 6.2(4)'.

In addition, the existence of central elements u with bounds $1/4 \leq N(u)/N \leq 3/4$ is derived and the bounds are shown to be the best possible.

From these results, searching algorithms are obtained that solve the search problem with at most $2.41 \cdot log_2 N$ evaluations of the function f (cf. Section 1.3).

The *isomorphism problem* can also be solved in polynomial time already for interval graphs [LB] by using the PQ-tree structure and the fact that Q-nodes have essentially a unique consecutive arrangement of the maximal cliques (Theorem 6.7). For details, we refer to [LB].

Finally, the *dummy minimization problem* can be solved for interval orders in polynomial time [Sp3], again for the case that no extra nodes are allowed in the edge diagram. The solution is based on the nesting property of the predecessor sets. It implies that any two sets of immediate predecessors are either disjoint, or one is contained in the other. This weakening of property (4) of N-free partial orders in Theorem 3.2 gives the information of how to introduce the dummy-edges. If the interior nodes of the edge diagram are (as in the proof of Theorem 3.2) identified with the non-empty sets $ImPred(v), v \in V$, then a dummy is introduced between any two nodes $ImPred(u), ImPred(v)$ such that $ImPred(u) \subset ImPred(v)$ and there is no $ImPred(w)$ with $ImPred(u) \subset ImPred(w) \subset ImPred(v)$.

An application involving MPQ-trees is the *seriation problem with side constraints.* An instance of this problem consists of an interval graph $G = (V, E)$ and a binary relation R on V. The task then is to test wether or not there is a partial order $P = (V, <)$ that is *compatible* with G (i.e. $G(P)^c = G$) and *respects* R (i.e. $u < v$ if $(u, v) \in R$), and to construct such a partial order if it exists.

Typical applications are *seriation in archeology* (where the edges of G express simultaneous occurences of items and R expresses known chronological relations among the items), and consecutive retrieval in database management (where G models the containment of records in queries, and R models access priorities).
This problem can be solved in $O(|V|^2)$ time by the use of MPQ-trees, see [KoM1,Mö4].

A slightly more general problem is obtained by specifying relations between the endpoints l_v, r_v of the intervals $I_v, v \in V$, of an interval representation of P. The previously by considered relation R on V is then contained as a special case by putting $r_u < l_v$ if $(u, v) \in R$.

For this problem, an $\Omega(|V|^3)$ algorithm was developed in [Sk2]. Using MPQ-trees, an $O(|V|^2)$ algorithm has been obtained in [Ko].

Still another variant of the seriation problem arises if $G = (V, E)$ is an arbitrary graph modeling *known,* but not necessarily *all* simultaneous occurences. Then the partial order $P = (V, <)$ we look for may have *more* incomparable pairs than G has edges. In this case, the existence of such a partial order can be tested in polynomial time, but the construction of a partial order with the fewest possible additional incomparable pairs is NP-hard since it contains (if $R = \emptyset$) an NP-complete edge deletion problem [GJ] as a special case (Given G and k, can the deletion of at most k edges turn G into an interval graph?).

6.4 Some Subclasses

There are several classes of interval orders that are obtained by specifying, in addition to **2+2** from Figure 35, more forbidden suborders.

For instance, if the partial order N is forbidden, one obtains the class of all *series-parallel interval orders* because of Corollary 2.7. This class contains the threshold orders and weak orders discussed in Section 2.5.

Using the methods leading to Theorem 2.16 and the transformation rules in Figure 31, one obtains a characterization of series–parallel interval orders in terms of their canonical decomposition tree and their MPQ–tree. The characterization via the associated interval representations is shown in [Sk1].

6.15 Theorem: *The following statements are equivalent:*

(1) *P is a series–parallel interval order.*

(2) *The canonical decomposition tree of P contains no prime–type node, and each parallel node has at most one son that is not a leaf.*

(3) *The MPQ–tree of P contains only P–nodes.*

(4) *In any interval representation of P, any two intervals are either disjoint, or one is contained in the other.*

Figure 35: The partial orders **2+2** and **1+3**

Another subclass of interval orders is obtained by forbidding the partial order **1+3** from Figure 35. The resulting partial orders are called *semiorders*. They arise in measurement theory as those partial orders that model preference defined by numerical values with a *constant* tolerance limit for indifference (see e.g. the survey articles [BFM,Bo]. Expressed in terms of an interval representation, this means that all intervals can be chosen to have the same length. This has a direct consequence for the nesting of successor and predecessor sets that is expressed in property (4) of Theorem 6.16.

6.16 Theorem: *The following statements are equivalent for $P = (V, <)$:*

(1) *P is a semiorder (i.e. does not contain **2+2** and **1+3**).*

(2) *P has an interval representation with unit length intervals.*

(3) *There are a function $f : V \to \mathbb{R}^1$ and a constant $c > o$ such that*

$$u < v \Leftrightarrow f(u) + c \leq f(v)$$

for all $u, v \in V$.

(4) The elements can be numbered as $v_1, v_2, \ldots, v_n$ *such that*

$$Suc(v_1) \subseteq Suc(v_2) \subseteq \ldots \subseteq Suc(v_n).$$

and

$$Pred(v_1) \supseteq Pred(v_2) \supseteq \ldots \supseteq Pred(v_n).$$

Note that for arbitrary interval orders, both nesting properties in (4) are valid (Theorem 6.2), but with the *same* numbering only for semiorders.

The class of semiorders contains the weak orders, but not the threshold orders.

The special properties of semiorders are sometimes useful in algorithmic respect. An example is the determination of the dimension of a semiorder. It is shown in [Rab1] that $dim(P) \leq 3$ if P is a semiorder. Together with the polynomial recognition algorithms for 2–dimensional partial orders, this result leads to a polynomial method for determining $dim(P)$ for semiorders P. Either P is 2–dimensional or $dim(P) = 3$. For arbitrary interval orders (which can have arbitrarily high dimension [Rab2]), the complexity status of the dimension problem is still open.

Still other classes of interval orders and interval graphs such as 2–dimensional interval orders whose conjugates are again interval orders, or interval graphs having for every vertex v an interval representation in which the interval representing v is the left–most or rightmost interval are investigated in [BHW,Sk1, SG].

Acknowledgement:

This work was done at the Institut für Operations Research of the University of Bonn, and partially supported by the Sonderforschungsbereich 303. I would like to thank U. Faigle, M. Habib, J. Spinrad, R. Schrader, G. Steiner and L. Stewart for various discussions and many helpfull remarks, and N. Korte for his carefull proofreading of the manuscript.

REFERENCES:

[AGU] A.V. Aho, M.R. Garey and J.D. Ullman (1972) 'The transitive reduction of a directed graph', *SIAM J. Comput.* **1**, 131–137

[AHU] A.V. Aho, J.E. Hopcroft and J.D. Ullman (1975) *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, Ma.

[At] M.D. Atkinson (1987) 'The complexity of orders', *this volume*

[AC] M.D. Atkinson and H.W. Chang (1986) 'Extensions of partial orders of bounded width', *Congressus Num.* **52**, 21–35

[BFR] K.A. Baker, P.C. Fishburn and F.S. Roberts (1971) 'Partial orders of dimension 2', *Networks* **2**, 11–28

[BFM] J.P. Barthèlemy, C. Flambert and B. Monjardet (1982) 'Ordered sets and social sciences', in [Ri2], 721–758

[BMR1] M. Bartusch, R.H. Möhring and F.J. Radermacher (1987) m–machine unit time scheduling: a report on ongoing research, *preprint*

[BMR2] M. Bartusch, R.H. Möhring and F.J. Radermacher (1987) 'Scheduling project networks with resource constraints and time windows', to appear in *Annals of Oper. Res.*

[BHW] C. Benzaken, P.L. Hammer and D. de Werra (1985) 'Split graphs of Dilworth number 2', *Discrete Math.* **55**, 123–128

[Bo] K.P. Bogart (1982) 'Some social science applications of ordered sets', in [Ri2], 759–787

[BHe] B. Bollobás and P. Hell (1985) 'Sorting and graphs', in [Ri3], 169–184

[BL] S. Booth and S. Lueker (1976) 'Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms', *J. Comput. Syst. Sci.* **13**, 335–379

[BH] V. Bouchittè and M. Habib (1987) 'The calculation of invariants', *this volume*

[BC] G. Boudol and I. Castellani (1986) 'On the semantics of concurrency: partial orders and transition systems', *preprint*, INRIA, 06560–Valbonne, France

[BK1] A. Brandständt and D. Kratsch (1984) 'On the restriction of some NP–complete graph problems to permutation graphs', *Report No. N/84/80*, Friedrich–Schiller Universität Jena, German Democratic Republic

[BK2] A. Brandständt and D. Kratsch (1986) 'On partitions of permutations into increasing and decreasing subsequences', *J. Inform. Processing and Cybernetics (EIK)* **22**, 263–273

[BM] H. Buer and R.H. Möhring (1983) 'A fast algorithm for the decomposition of graphs and posets', *Math. Oper. Res.* **8**, 170–184

[CCMP] G. Chaty, M. Chein, P. Martin, G. Potella (1976) 'Some results about the jump number of jumps in aircuit digraphs', *Proc. 5^{t}h Conf. Combinatorics, Graph Theory and Computing, Utilitas Math.* 267–279

[CHab] M. Chein and M. Habib (1980) 'The jump number of dags and posets: an introduction', *Ann. of Discrete Math.* **9**, 189–194

[CHam] V. Chvatal and P.L. Hammer (1977) 'Aggregation of inequalities in integer programming', *Annals Discrete Math.* **1**, 145–162

[CG] E.G. Coffman, Jr. and R.L. Graham (1972): 'Optimal scheduling for two–processor systems'. *Acta Informatica* **1**, 200–213

[CLS] D.G. Corneil, H. Lerchs and L. Stewart Burlingham (1981) 'Complement reducible graphs', *Discr. Appl. Math.* **3**, 163–174

[CPS] D.G. Corneil, Y. Perl and L. Stewart (1985) 'A linear recognition algorithm for cographs', *SIAM J. Comput.* **14**, 926–934

[Co] C.J. Colbourn (1981) 'On testing isomorphism of permutation graphs', *Networks* **11**, 13–21

[CP] C.J. Colbourn and W.R. Pulleyblank (1985) 'Minimizing setups in ordered sets of fixed width', *Order* **1**, 225–228

[DGS] U. Derigs, O. Goecke, and R. Schrader (1984) 'Bisimplicial edges, Gaussian elimination and matchings in bipartite graphs', *Proc. Int. Workshop on Graphtheoretic Concepts in Computer Science*, Trauner Verlag, Berlin, W.Germany, 79–87

[Du] R.J. Duffin (1965) 'Topology of series–parallel networks', *J. Math. Analysis Appl.* **10**, 303–318

[DM] B. Dushnik and E. Miller (1941) 'Partially ordered sets', *Amer. J. Math.* **63**, 600–610

[El] S.E. Elmaghraby (1977) *Activity Networks*, Wiley, New York

[Elz] M. El–Zahar (1987) *this volume*

[FGS] U. Faigle, G. Gierz and R. Schrader (1985) 'Algorithmic approaches to setup minimization', *SIAM J. Comput.* **14**, 954–965

[FLRT] U. Faigle, L. Lovász, R. Schrader and Gy. Turan (1986) 'Searching in trees, series–parallel and interval orders', *SIAM J. Comput.* **15**, 1075–1084

[FS1] U. Faigle and R. Schrader (1985) 'A setup heuristic for interval orders', *Oper. Res. Letters* **4**, 185–188

[FS2] U. Faigle and R. Schrader (1986) 'On the computational complexity of the order polynomial', *Discrete Appl. Math.* **15**, 261–269

[FS3] U. Faigle and R. Schrader (1987) private communication

[Fi1] P.C. Fishburn (1970) 'Intransitive indifference in preference theory: A survey', *Oper. Res.* **18**, 207–228

[Fi2] P.C. Fishburn (1985) *Interval orders and Interval Graphs*, Wiley, New York

[Fi3] P.C. Fishburn (1985) 'Interval graphs and interval orders', *Discrete Math.* **55**, 135–150

[FiG] P.C. Fishburn, W.V. Gehrlein (1986) 'Minimizing bumps in linear extensions of ordered sets', *Order* **3**, 3–14

[FF] L.R. Ford and D.R. Fulkerson (1962) *Flows in Networks*, Princeton Univ. Press, Princeton, New Jersey

[Fo] D.J. Foulis (1970) *Empirical logic*, Course Notes, Univ. of Massechusetts

[FuG] D.R. Fulkerson and O.A. Gross (1965) 'Incidence matrices and interval graphs', *Pacific J. Math.* **15**, 835–855

[Gab] H.N. Gabov (1982) 'An almost–linear algorithm for two–processor scheduling', *J.ACM* **29**, 766–780

[Ga] T.Gallai (1967) 'Transitiv orientierbare Graphen', *Acta Math. Acad. Scient. Hung. Tom.* **18**, 25–66

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

[GJTY] M.R. Garey, D.S. Johnson, R.E. Tarjan and M. Yannakakis (1983) 'Scheduling opposing forests', *SIAM J. Alg. Disc. Meth.* **4**, 72–93

[GH] P.C. Gilmore and A.J. Hoffman (1964) 'A characterization of comparability graphs and of interval graphs', *Canad. J. Math.* **16**, 539–548

[Gi] J.L. Gischer (1984) *Partial orders and the axiomatic theory of shuffle*, Thesis, Stanford University

[Go1] M.C. Golumbic (1980) *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York

[Go2] M.C. Golumbic (1985) 'Interval graphs and related topics', *Discrete Math.* **55**, 113–121

[GM] S. Goto and T. Matsuda (1986) 'Partitioning, assignment and placement', in T. Ohtsuki (ed.) *Layout Design and Verification*, North Holland, Amsterdam, 55–98

[Gr] P. Grillet (1969) 'Maximal chains and antichains', *Fund. Math.* **65**, 157–167

[Hab] M. Habib (1984) 'Comparability invariants', *Annals Discrete Math.*, **23** 331–386

[HJ] M. Habib and R. Jegou (1985) 'N–free posets as generalizations of series–parallel posets', *Discrete Appl. Math.* **12** (3), 279–291

[HMa] M. Habib and M.C. Maurer (1979) 'On the X–join decomposition for undirected

graphs', *Appl. Discr. Math.* **3**, 198–207

[HMö] M. Habib and R.H. Möhring (1987) 'On some complexity properties of N-free posets and posets with bounded decomposition diameter', *Discrete Math.* **63**, 157–182

[HMS] M. Habib, R.H. Möhring and G. Steiner (1987) 'Computing the bump number of ordered sets is easy', *Report No. 87475*, Institut für Operations Research, University of Bonn, West Germany

[Han] P. Hanlon (1982) 'Counting interval graphs', *Trans. Amer. Math. Soc.* **272**, 383–426

[HN] F. Harary and Z.R. Norman (1960) 'Some properties of line digraphs', *Rend. Circ. Math.* Palermo **9**, 161–168

[HT] D. Harel and R.E. Tarjan (1984) 'Fast algorithms for finding nearest common ancestors', *SIAM J. Comput.* **13**, 338–355

[HB] R.L. Hemminger and L.W. Beineke (1978) 'Line graphs and line digraphs', in: L.W. Beineke, R.J. Wilson, eds., *Selected Topics in Graph Theory* (Academic Press, London, 1978) 271–305

[He] C. Heuchenne (1964) 'Sur une certaine correspondance entre graphes', *Bull. Soc. Roy. Sci.*, Liège **33**, 743–753

[Hi] T. Hiraguchi (1951) 'On the dimension of partially ordered sets', *Sci. Rep. Kanazawa Univ.* **1**, 77–94

[Jo] D.S. Johnson (1982–87) 'The NP-completeness column: an ongoing guide', *J. Algorithms* **3** ff

[Ju] H.A. Jung (1978) 'On a class of posets and the corresponding comparability graphs', *J. Comb. Th. (B)* **24**, 125–133

[Ka1] R. Kaerkes (1970) 'Zum Begriff des orientierten Graphen', *Methods of Oper. Res.* **7**, 122–125

[Ka2] R. Kaerkes (1977) 'Netzplan Theory' *Methods of Oper. Res.*, **27**, 1–65

[KM] R. Kaerkes and R.H. Möhring (1978) *Vorlesungen über Ordnungen und Netzplantheorie*, Schriften zur Informatik und Angewandten Mathematik 45, RWTH Aachen

[Kal] A. Kaldewaij (1985) 'On the decomposition of sequences', *Inform. Process. Letters* **21**, 69

[Ke] D. Kelly (1985) 'Comparability graphs', in [Ri3], 3–40

[KT] D. Kelly and W.T. Trotter, Jr. (1982) 'Dimension theory for ordered sets', in

[Ri2], 171–211

[Ko] N. Korte (1987) *Intervallgraphen und MPQ-Bäume: Algorithmen und Datenstrukturen zur Manipulation von Intervallgraphen und der Lösung von Serationsproblemen*, 'Report No. 87461-Or Thesis, Institut für Operations Research, University of Bonn, West Germany

[KoM1] N. Korte und R.H. Möhring (1985) 'Transitive orientation of graphs with side constraints', in H. Noltemeier (ed.) *Procedings of WG'85*, Trauner Verlag, 143–160

[KoM2] N. Korte und R.H. Möhring (1986) 'A simple linear–time algorithm to recognize interval graphs', *Report No. 86421-OR*, Institut für Operations Research, Universität Bonn, to appear in SIAM J. Comput.

[KD] M.S. Krishnamoorthy and N. Deo (1979) 'Complexity of the minimum–dummy–activities problem in a PERT network', *Networks* **9**, 189–194

[La1] E.L. Lawler (1976) *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York

[La2] E.L. Lawler (1976) 'Graphical algorithms and their complexity', *Math. Centre Tracts* **81**, 3–32

[La3] E.L. Lawler (1978) 'Sequencing jobs to minimize total weighted completion time subject to precedence constraints', *Ann. Discrete Math.* **2**, 75–90

[LLR] E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan (1982) 'Recent developments in deterministic sequencing and scheduling: A survey', in M.A.H. Dempster et al. (eds.) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 35–73

[LM] B. Leclerc and B. Monjardet (1973) 'Orders "C.A.C."', *Fund. Math.* **79**, 11–22

[LMü] T. Lengauer and R. Müller (1986) 'Linear algorithms for optimizing the layout of dynamic *CMOS* cells', *Report No. 32*, Fachbereich Mathematik–Informatik, Universität Paderborn, West–Germany

[Li] N. Linial (1986) 'Hard enumeration problems in geometry and combinatorics', *SIAM J. Alg. Disc. Meth.* **7**, 331–335

[LS] N. Linial and M.E. Saks (1985) 'Searching ordered structures', *J. Algorithms* **6**, 86–103

[LB] G.S. Lueker and K.S. Booth (1979) 'A linear time algorithm for deciding interval graph isomorphism', *J. ACM* **26**, 183–195

[Ma] E. Mayr (1981) 'Well structered programs are not easier to schedule', *Report Stan-CS-81-880*, Department of Computer Science, Stanford University

[MP] J.J. Moder and C.R. Phillips (1964) *Project management with CPM and PERT*, Reinhold, New York

[Mö1] R.H. Möhring (1982) 'Scheduling problems with a singular solution', *Ann. Discrete Math.* **16**, 225–239

[Mö2] R.H. Möhring (1983) 'On the characterization of series–parallel networks, complement reducible graphs, and threshold graphs', *Methods of Oper. Res.* **45**, 287–291

[Mö3] R.H. Möhring (1984) 'Almost all comparability graphs are UPO', *Discrete Math.* **50**, 63–70

[Mö4] R.H. Möhring (1985) 'Algorithmic aspects of comparability graphs and interval graphs', in [Ri3], 41–101

[MR1] R.H. Möhring und F.J. Radermacher (1984) 'Substitution decomposition of discrete structures and connections with combinatorial optimization', *Annals Discrete Math.* **19**, 257–356

[MR2] R.H. Möhring und F.J. Radermacher (1985) 'Generalized results on the polynomiality of certain weighted sum scheduling problems', *Methods of Oper. Res.* **49**, 405–417

[MS1] C.L. Monma and J.B. Sidney (1979) 'Sequencing with series–parallel precedence constraints', *Math. of Oper. Res.* **4**, 215–224

[MS2] C.L. Monma and J.B. Sidney (1987) 'Optimal sequencing via modular decomposition: Characterization of sequencing functions', *Math. Oper. Res.* **12**, 22–31

[MSp] J.H. Muller and J. Spinrad (1984) 'On–line modular decomposition', *Technical report GIT–ICS–84/11*, Georgia Institute of Technology, to appear in J. ACM

[Or] O. Ore (1962) *Theory of Graphs*, Coll. Publ. 38, Amer. Math. Soc., Providence, R.I.

[PY] C.H. Papadimitriou and M. Yannakakis (1979) 'Scheduling interval–ordered tasks', *SIAM J. Comp.* **8**, 405–409

[PLE] A. Pnueli, A. Lempel and W. Even (1971) 'Transitive orientation of graphs and identification of permutation graphs' *Canad. J. Math.* **23**, 160–175

[PB] J.S. Provan and M.O. Ball (1983) 'The complexity of counting cuts and of the probability that a graph is connected', *SIAM J. Comput.* **12**, 777–788

[Pu] W.R. Pulleyblank (1981) 'On minimizing setups in precedence constrained scheduling problems', *Report 81185–OR*, Institut für Operations Research, University of Bonn, West Germany

[Rab1] I. Rabinovitch (1978) 'The dimension of semiorders', *J. Comb. Th.(A)* **25**, 50–61

[Rab2] I. Rabinovitch (1978) 'An upper bound on the dimension of interval orders', *J. Comb. Th.(A)* **25**, 68–71

[Ra1] F.J. Radermacher (1977) 'Floats in project networks', *Methods of Oper. Res.* **27**, 163–224

[Ra2] F.J. Radermacher (1986) 'Scheduling of project networks', *Annals of Oper. Res.* **4**, 227–252

[Ra3] F.J. Radermacher (1986) 'Schedule–induced posets', *Discrete Appl. Math.* **14**, 67–91

[Ra4] F.J. Radermacher (1987) Characterization of the critical posets in two–machine unit time scheduling, *preprint*, University of Passau, West Germany

[Re] F. Rendl (1986) 'Quadratic assignment problems on series–parallel digraphs', *Zeitschrift Oper. Res. 30(A)* 161–173

[Ri1] I. Rival (1982) 'Optimal linear extensions by interchanging chains', *Proc. Amer. Math. Soc.* **89**, 387–394

[Ri2] I. Rival (1982) (ed.) *Ordered Sets*, Reidel, Dordrecht

[Ri3] I. Rival (1985) (ed.) *Graphs and Order*, Reidel, Dordrecht

[RZ] I. Rival and N. Zaguia (1986) 'Constructing N–free, jump–critical ordered sets', *Congressus Num.* **55**, 199–204

[Ro] F.S. Roberts (1976) *Discrete Mathematical Models, with Applications to Social, Biological and Environmental Problems*, Prentice Hall, Englewood Cliffs, NJ

[RTL] D.J. Rose, R.E. Tarjan, and G.S. Luecker (1976) 'Algorithmic aspects of vertex elimination on graphs', *SIAM J. Comput.* **5**, 266–283

[Sa] M.E. Saks (1985) 'The information theoretic bound for problems on ordered sets and graphs', in [Ri3], 137–168

[San] B. Sands (1981) 'Counting antichains in finite partially ordered sets', *Discrete Math.* **35**, 213–228

[ScS] A.A. Schäfer and B.B. Simons (1987) 'Computing the bump number of a partial order with techniques from two–processor scheduling', *preprint*

[Se] D. Seinsche (1974) 'On a property of the class of n–colorable graphs', *J. Comb. Th.(B)* **16**, 191–193

[SKOM] S. Shinoda, Y. Kajitani, K. Onaga, W. Mayeda (1979) 'Various characterizations of series–parallel graphs', *Proc. 1979 ISCHS*, 100-103

[Si] J.B. Sidney (1975) 'Decomposition algorithms for single machine sequencing with precedence relations and deferral costs', *Oper. Res.* **23**, 283–298

[SiS] J.B. Sidney and G. Steiner (1986) 'Optimal sequencing by modular decomposition: polynomial algorithms', *Oper. Res.* **34**, 606–612

[Sk1] D.J. Skrien (1982) 'A relationship between triangulated graphs, comparability graphs, proper interval graphs, proper circular–arc graphs, and nested interval graphs', *J. Graph Theory* **6**, 309–316

[Sk2] D.J. Skrien (1984) 'Chronological orderings of interval graphs', *Discrete Appl. Math.* **8**, 69–83

[SG] D.J. Skrien and J. Gimble (1985) 'Homogeneousely representable interval graphs', *Discrete Math.* **55**, 213–216

[Sp1] J. Spinrad (1982) *Two dimensional partial orders*, Ph.D. Thesis, Princeton University

[Sp2] J. Spinrad (1984) 'Bipartite permutation graphs', *preprint*, School of ICS, Georgia Inst. of Technology, Atlanta

[Sp3] J. Spinrad (1984) 'The minimum dummy task problem', *preprint* School of ICS, Georgia Inst. of Technology, Atlanta

[Sp4] J. Spinrad (1985) 'On comparability and permutation graphs', *SIAM J. Comput.* **14**, 658–670

[Sp5] J. Spinrad (1987) 'P_4–trees and substitution decomposition', *preprint*, Dept. of Computer Science, Vanderbilt Univ., Nashville, Tennesse

[SV] J. Spinrad and J. Valdes (1983) 'Recognition and isomorphism of two dimensional partial orders', 10^th Coll. on Automata, Language and Programming. *Lecture Notes in Computer Science* **154**, Springer, Berlin, 676–686

[St1] G. Steiner (1984) 'Single machine scheduling with precedence constraints of dimension 2', *Math. Oper. Res.* **9**, 248–259

[St2] G. Steiner (1985) 'A compact labeling scheme for series–parallel graphs', *Discrete Appl. Math.* **11**, 281–297

[St3] G. Steiner (1985) 'On finding the jump number of a partial order by substitution decomposition', *Order* **2**, 9–23

[St4] G. Steiner (1986) 'An algorithm to generate the ideals of a partial order', *Oper. Res. Letters* **5**, 317–320

[St5] G. Steiner (1985) 'Searching in 2–dimensional partial orders', *Report No 240*, Faculty of Business, McMaster University, Hamilton, Ontario

[St6] G. Steiner (1987) 'Minimizing bumps in ordered sets by substitution decomposition', *preprint*, Faculty of Business, McMaster University, Hamilton, Ontario

[St7] G. Steiner (1987) 'On computing the information theoretic bound for sorting: counting the linear extensions of posets', *preprint*, Faculty of Business, McMaster University, Hamilton, Ontario

[St8] G. Steiner (1987) 'On the information theoretic bound for sorting: balancing the linear extensions of posets', *preprint*, Faculty of Business, McMaster University, Hamilton, Ontario

[StS] G. Steiner and L.K. Stewart (1987) 'A linear time algorithm to find the jump number of 2–dimensional bipartite partial orders', *Order* **3**, 359–367

[Ste1] L.K. Stewart (1985) *Permutation graph structure and algorithms*, Thesis, Department of Computer Science, University of Toronto

[Ste2] L.K. Stewart (1986) 'Permutation graph structure', *Congressus Num.* 54, 87–100

[Su1] D.P. Sumner (1973) 'Graphs undecomposable with respect to the X–join', *Discrete Math.* **6**, 281–298

[Su2] D.P. Sumner (1974) 'Dacey graphs', *J. Australian Math. Soc.* **18**, 492–502

[Sup] K.J. Supowit (1985) 'Decomposing a set of points into chains, with applications to permutation and circle graphs', *Inform. Processing Letters* **21**, 249–252

[Sy1] M.M. Syslo (1982) 'A labeling algorithm to recognize a line digraph and output its root graph', *Inform. Processing Letters* **15**, 241–260

[Sy2] M.M. Syslo (1984) 'On the computational complexity of the minimum–dummy–activities problem in a PERT network', *Networks* **14**, 37–45

[Sy3] M.M. Syslo (1984) 'Series–parallel graphs and depth–first search trees', *IEEE Trans. on Circuits and Systems* CAS–31, 1029–1033

[Sy4] M.M. Syslo (1985) 'A graph theoretic approach to the jump number problem', in [Ri3], 185–215

[TNS] K. Takamizawa, T. Nishizeki, N. Saito (1982) 'Linear–time computability of combinatorial problems on series–parallel graphs', *J. ACM* **29**, 623–641

[TMS] W.T. Trotter,Jr., J.I. Moore,Jr. and P. Sumner (1976) 'The dimension of a comparability graph', *Proc. Amer. Math. Soc.* **60**, 35–38

[Va] J. Valdes (1978) 'Parsing flowcharts and series–parallel graphs', *Technical report STAN–CS–78–682*, Computer Science Department, Stanford University, Stanford, Ca.

[VTL] J. Valdes, R.E. Tarjan and E.L. Lawler (1982) 'The recognition of series–parallel digraphs', *SIAM J. Comput.* **11**, 298–314

[Ya] M. Yannakakis (1982) 'The complexity of the partial order dimension problem', *SIAM J. Discr. Meth.* **3**, 351–358