SEQUENCING ON SINGLE MACHINE WITH GENERAL PRECEDENCE CONSTRAINTS — THE JOB MODULE ALGORITHM

By

GEORGE STEINER
Management Science and Information Systems Area

FACULTY OF BUSINESS
McMASTER UNIVERSITY
HAMILTON, ONTARIO

Research and Working Paper Series No. 209 1983
Abstract

We introduce a class of sequencing problems based on some common, easily verifiable properties. Some well-known members of this class are the Total weighted completion time problem and the Least cost fault detection problem, both with general precedence constraints. We present an efficient algorithm for solving these problems. It decomposes the original problem into smaller sub-problems, called job-modules and sequences these in an optimal way. The main result of this paper is a polynomial-time algorithm for finding job modules in a general precedence graph.
Introduction

In this paper we introduce a class of sequencing problems based on some common, easily verifiable properties. Some well-known members of this problem set are the Total weighted completion time problem and the Least cost fault detection problem. An efficient general algorithm is presented for solving these problems. The algorithm is based on finding smaller problems, called job-modules and sequencing these in optimal fashion. The main result of this paper is a polynomial-time algorithm for finding job-modules in a general precedence graph.

1. Preliminary definitions, notation and results

Consider the set \( J = \{1, 2, \ldots, n\} \) of \( n \) jobs to be sequenced for processing on a single machine. Each job is characterized by certain parameters. (E.g. in the total weighted completion time problem the job \( i \) is specified by its non-negative processing time \( p_i \) and a weighting factor \( w_i > 0 \).) A sequence \( s \) of \( k \) jobs is a function from \( \{1, 2, \ldots, k\} \) to \( J \) and will be represented by \((s(1), s(2), \ldots, s(k))\), where \( s(i) \) is the \( i \)-th job in the sequence \( s \). A job may appear several times or not at all in a sequence. A cost function assigns a real value to each sequence. (E.g. for the total weighted completion time problem \( f(s) = \sum_{i=1}^{n} w_s(i)c_i^s \), where \( c_i^s = \sum_{j=1}^{i} p_s(j) \) is the completion time of the \( i \)-th job in the sequence \( s \). For motivation we will use this problem throughout this paper.)

If we denote a permutation of the jobs in \( J \) by \( \pi \), then a sequencing problem, on a set of jobs \( J \) with cost function \( f \), is to find a permutation of \( J \) contained in a set of feasible permutations \( F \), which minimizes \( f \), i.e.

\[
\min_{\pi \in F} f(\pi)
\]

The problem is called unconstrained if all permutations are feasible. In many cases however, the set of feasible permutations is restricted by a set of
precedence relations $R$, such that if $(i,j) \in R$ (we will also use the notation $i\rightarrow j$) then job $i$ must precede job $j$ in any feasible sequence. A sequencing problem constrained by such precedence relations will be denoted by $A(J,R)$. The set of precedence relations can be represented by an acyclic digraph $G = (J,A)$, where each node $i \in J$ corresponds to one of the tasks in $J$ and the arc $(i,j) \in A$ iff $(i,j) \in R$. A subset $J' \subseteq J$ will be called a compound job if the jobs in $J'$ must be sequenced consecutively in every feasible sequence. If, in addition, a compound job must be processed in a fixed sequence of its jobs in every feasible permutation of $J$, then we refer to it as a string. The precedence constraints could be extended to strings of jobs, denoted by $\hat{s}$, $\hat{t}$, $\hat{u}$, etc.: $\hat{s} \rightarrow \hat{t}$ means, that the jobs in both $\hat{s}$ and $\hat{t}$ must be sequenced consecutively in the fixed order of each string and every job in $\hat{s}$ must precede every job in $\hat{t}$.

A common property of the class of sequencing problems discussed in this paper is the adjacent pairwise interchange (API) property: We say that a cost function $f$ satisfies the API property if there is a transitive and complete binary preference relation $\preceq$ defined on the jobs in $J$ s.t. for all $i,j \in J$, $i \preceq j$ implies that $f(u,i,j,v) \leq f(u,j,i,v)$ for all sequences $u$ and $v$.

We say that $i$ is strictly preferable to $j$ ($i \prec j$) if $i \preceq j$ but $j \preceq i$ does not hold.

Smith [17] presented a very simple and powerful solution to any unconstrained sequencing problem satisfying the API property, by the following theorem:

**Theorem 1.** (Smith [17]): Let $f$ satisfy the API property. Then any permutation satisfying the following property is optimal: $i \prec j$ implies that $i$ precedes $j$ in the permutation.

A simple proof of this theorem can be found e.g. in Monma and Sidney
[13].

The constrained problem had a long history of relatively slow progress from one special case to another. Conway, Maxwell and Miller [5] in 1964, gave a procedure for solving the total weighted completion time problem in case $G$ is in the form of parallel chains and all $w_j = 1$. In 1971, 1972 Baker [3] and Horn [8] proposed algorithms for the case in which $G$ is a rooted tree and in 1973 Adolphson and Hu [1] showed that such an algorithm can be implemented in $O(n \log n)$ running time. In 1975 Sidney [16] published a number of theorems which apply to the total weighted completion time problem with arbitrary precedence constraints and can serve as basis for decomposing the problem into smaller size similar problems. Lawler [10] in 1978 presented an algorithm for the special case of this problem when $G$ is a "series-parallel" graph and showed that this algorithm can be implemented in $O(n \log n)$ running time if the precedence graph $G$ is known to be "series-parallel" and is represented by special rooted binary trees called decomposition trees. This algorithm does not use information about the cost function other than the ordinal information defined by the API property. Monma [12] gave a set of transformations based on such information about the cost function and showed that any algorithm attempting to extend the "series-parallel" algorithm to a wider class of precedence graphs must use information about the cost function other than ordinal. Monma's transformations apply to the general case but do not guarantee the solution of the problem for "non-series-parallel" graphs. Lawler [10] also showed that the total weighted completion time problem under arbitrary precedence constraints is $NP$-complete, therefore it is doubtful that a really efficient (polynomial) algorithm ever will be found for the general case.

A subgraph $G' = (J',A')$ induced by $J' \subseteq J$ has $A' = \{(i,j) | i,j \in J' \text{ and } (i,j) \in A\}$. $J' \subseteq J$ is called a job-module of $G = (J,A)$ if for every job
\( k \in J \setminus J' \) either

a) \((k,i) \in A \) for all \( i \in J' \), or

b) \((i,k) \in A \) for all \( i \in J' \), or

c) \((k,i) \notin A \) and \((i,k) \notin A \) for all \( i \in J' \).

Monma and Sidney [13] generalized the API property from jobs to sequences of jobs (job-strings): A function \( f \) satisfies the adjacent sequence interchange (ASI) property if there exists a transitive and complete binary preference relation \( \preceq \) defined on sequences by the following:

For all sequences \( s \) and \( t \), \( s \preceq t \) implies that

\[ f(u,s,t,v) \leq f(u,t,s,v) \]

for all sequences \( u \) and \( v \).

The ASI property implies the API property. The main difficulty of sequencing with precedence constraints lies in the possible conflict between the ordering of jobs by the preference relation and the ordering by the precedence constraints. Monma and Sidney [13] presented the following generalization of Smith's result which resolves this conflict between the two orderings in certain cases:

**Theorem 2** (Sidney and Monma [13]): Let \( f \) satisfy the ASI property. Consider a job module \( \{s^+,t^+\} \) in a general precedence graph where \( s^+ \) and \( t^+ \) are strings of jobs from \( J \), and we have \( s^+ \rightarrow t^+ \). Further assume that for the corresponding sequences \( s \) and \( t \) we have the conflicting preference relation \( t \preceq s \). Then there is an optimal permutation with \( s^+ \) immediately preceding \( t^+ \).

Monma [7] defined the following additional properties for sequencing problems:

**Definition.** We say that a function \( f \) satisfies the **strong** ASI property when there exists a transitive and complete binary preference relation \( \preceq \) defined on sequences satisfying the following:
For all sequences \( s, t, u \) and \( v \)

\[ s \leq t \text{ if and only if } f(u, s, t, v) \leq f(u, t, s, v). \]

The preference relation \( \leq \) (defined by the ASI or strong ASI property) can be easily extended from sequences to subsets of jobs in \( J \). For \( U, T \subseteq J \) we say that \( U \leq T \) if and only if \( u \leq t \), where \( u \) and \( t \) are cost-minimal, feasible permutations of \( U \) and \( T \) respectively.

**Definition.** We say that a function \( f \) with preference relation \( \leq \) satisfies the **consistency property** when the following is true for all permutations \( s_1 \) and \( s_2 \) of the same set: If \( f(s_1) \leq f(s_2) \) then \( s_1 \leq s_2 \). That is any ordering of the permutations of a set by the cost function \( f \) is consistent with the ordering by the preference relation \( \leq \).

**Definition.** We say that a function \( f \) satisfies the **series network decomposition (SND)** property when the following is true for all permutations \( s_1 \) and \( s_2 \) of the same subset: If \( f(s_1) \leq f(s_2) \) then \( f(u, s_1, v) \leq f(u, s_2, v) \) for all sequences \( u \) and \( v \).

We say that a function \( f \) satisfies the **strong SND** property if the following holds for all permutations \( s_1 \) and \( s_2 \) of the same set and sequences \( u \) and \( v \):

\[ f(s_1) \leq f(s_2) \text{ if and only if } f(u, s_1, v) \leq f(u, s_2, v) \]

**Definition.** A set \( U \subseteq J \) is called **initial** in \( G = (J, A) \) if there are no jobs \( i \in J \setminus U \) and \( j \in U \) s.t. \( i \rightarrow j \).

A set \( U \subseteq J \) is called **p-minimal** ("preference minimal") in \( G = (J, A) \) if 1) \( U \) is a nonempty initial set in \( G \) and 2) \( U \leq T \) for all nonempty, initial sets \( T \) in \( G \).

A set \( U \subseteq J \) is called **p*-minimal** in \( G = (J, A) \) if 1) \( U \) is p-minimal in \( G \), and 2) no proper subset of \( U \) is p-minimal in \( G \).

Sidney and Monma [13] presented a polynomial time algorithm for a sequencing problem if its cost function \( f \) satisfies the ASI and SND properties.
and the precedence graph G is a series-parallel digraph.

For the total weighted completion time problem with general precedence relations Sidney [16] presented two decomposition algorithms:

1) The Decomposition Algorithm, decomposing the job set into p*-minimal (or p-minimal) subsets

2) The Job-Module Algorithm, decomposing the job set into job-modules.

Monma [12] introduced the **consecutive set assumption**: If U is a p*-minimal set in G = (J,A), then the jobs in U occur consecutively in every optimal permutation. He also showed that both decomposition algorithms produce optimal solutions for any sequencing problem satisfying the strong ASI, strong SND and consistency properties and the consecutive set assumption. Sidney [16] showed that these are true for the total weighted completion time problem, while Monma [12] showed that the least cost fault detection problem satisfies the first three of the above properties. Kadane and Simon [9] proved that even a generalized version of the least cost fault detection problem satisfies the consecutive set assumption. Therefore the Job-Module Algorithm is applicable to this problem too. The applicability of the Job-Module Algorithm to these problems does not necessarily mean that we have an efficient solution technique for these problems. The main difficulties are:

a) Finding the job-modules

b) Sequencing the job-modules.

In this paper we present a polynomial-time algorithm for finding job-modules in a general precedence graph. This algorithm enables us to determine in polynomial time for any precedence graph whether it contains a "non-trivial" job-module, and if the answer is yes, such a job-module is identified. We also characterize the types of graphs which can constitute a job-module in a general precedence graph.
The presentation of the algorithm requires the brief discussion of the theory of $\Gamma$-chains and a decomposition theory for precedence graphs based on the earlier work of Golumbic [6,7], Pnueli, Lempel, Even [14] and others.

2. $\Gamma$-chains and implication classes

Let $\bar{G} = (V,E)$ be an undirected graph with vertex set $V$ and edge set $E$. Any edge $(a,b) \in E$ can be oriented two ways: from $a$ to $b$ $(a,b)$ or from $b$ to $a$ $(b,a)$. Let $\hat{E}$ be the set of all edges in $E$ with both possible orientations i.e. $(a,b) \in \hat{E}$ and $(b,a) \in \hat{E}$ iff $(a,b) \in E$.

Golumbic [7] defines the binary relation $\Gamma$ on the oriented edges of an undirected graph $\bar{G} = (V,E)$ as follows: Let $(a,b) \in \hat{E}$ and $(c,d) \in \hat{E}$ be oriented edges of $\bar{G}$. We say that

\[
(a,b) \Gamma (c,d) \text{ iff } \begin{cases} 
\text{either } a = c \text{ and } (b,d) \notin \hat{E} \\
\text{or } b = d \text{ and } (a,c) \notin \hat{E}.
\end{cases}
\]

In other words, $(a,b) \Gamma (a,d)$ iff $(b,d) \notin \hat{E}$ or $(a,b) \Gamma (c,b)$ iff $(a,c) \notin \hat{E}$. We say that $(a,b)$ "forces" $(c,d)$ whenever $(a,b) \Gamma (c,d)$, meaning that if two arcs are $\Gamma$-related, then the orientation of either one of them forces the orientation of the other one if we want to maintain the transitivity of the orientation. (See Figure 1.)

The binary relation $\Gamma$ is not necessarily transitive, but if $\Gamma^*$ is the reflexive and transitive closure of $\Gamma$ (Figure 2.), it can be easily shown to be an equivalence relation on $\hat{E}$ and so it partitions $\hat{E}$ into disjoint classes, called implication classes of $\bar{G}$. Thus the arcs $(a,b)$ and $(c,d)$ are in the same implication class i.e. $(a,b) \Gamma^*(c,d)$ iff there exists a sequence of oriented arcs such that

\[
(a,b) = (a_0,b_0)\Gamma(a_1,b_1)\Gamma\ldots\Gamma(a_k,b_k) = (c,d) \quad (k \geq 0)
\]

Such a sequence is called a $\Gamma$-chain from $(a,b)$ to $(c,d)$, and it can be seen that the practical meaning of the existence of such a $\Gamma$-chain between $(a,b)$
and (c,d) is that if we direct the edge (a,b) ∈ E as (a,b), that forces us to
direct the edge (c,d) ∈ E as (c,d) if we want to make the orientation
transitive (Figure 3.).

The graph $\bar{G}$ in Figure 4.a has eight implication classes:
$B_1 = \{(a,b)\}, B_2 = \{(c,d)\}, B_3 = \{(a,c),(b,c),(e,c)\}, B_4 = \{(a,d),(b,d),(e,d)\}$
$B_1^{-1} = \{(b,a)\}, B_2^{-1} = \{(d,c)\}, B_3^{-1} = \{(c,a),(c,d),(c,e)\}, B_4^{-1} = \{(d,a),(d,b),(d,e)\}$.

If $B$ is an implication class of $\bar{G}$ then let
$B^{-1} = \{(b,a) | (a,b) \in B\}$ and $\bar{B} = \{(a,b) | (a,b) \in E$ and $(a,b) \in B$ or $(a,b) \in B^{-1}\}$. $\bar{B}$ is called a color class of $\bar{G}$.

Columbic [7 pp. 107-112] proved the following theorems:
Theorem 3: Let $B$ be an implication class of $\bar{G} = (V,E)$. If $\bar{G}$ has a transitive
orientation $F$ (considered as a set of directed edges), then either $B \subseteq F$ and
$F \cap B^{-1} = \emptyset$ or $B \cap F = \emptyset$ and $B^{-1} \subseteq F$.

Theorem 4: $\bar{G} = (V,E)$ is a comparability graph (i.e. transitive orientable)
iff $B \cap B^{-1} = \emptyset$ for every implication class $B$ of $\bar{G}$.

Theorem 5: Let $B$ be an implication class of $\bar{G} = (V,E)$. Exactly one of the
following alternatives holds:

i) $B = B^{-1}$ and $\bar{G}$ is not transitively orientable

ii) $B \cap B^{-1} = \emptyset$, $B$ and $B^{-1}$ are transitive and they are the only
transitive orientations of $\bar{B}$.

Theorem 5 clearly implies that each color class of an undirected graph $G$
has exactly two transitive orientations (one being the reversal of the other)
or has no transitive orientation at all. E.g., the graph in Figure 4b has
only one implication class
$B = \{(a,b),(b,c),(d,c),(c,e),(b,e),(e,f),(f,e),(e,b),(e,c),(c,b),(b,a),(c,d)\}$
and clearly $B = B^{-1}$, meaning that the graph is not transitively orientable.
Definition: Let \( H_0 \) be a graph on \( n \) vertices \( v_1,v_2,\ldots,v_n \) and let \( H_1,H_2,\ldots,H_n \) be \( n \) disjoint graphs.* The composition graph \( H = H_0[H_1,H_2,\ldots,H_n] \) is formed as follows: For all \( 1 \leq i,j \leq n \) replace vertex \( v_i \) in \( H_0 \) with the graph \( H_i \) and make each vertex of \( H_i \) adjacent to each vertex of \( H_j \) whenever \( v_i \) is adjacent to \( v_j \) in \( H_0 \). Formally if \( H_i \) has vertex set \( V_i \) and edge set \( E_i (H_i = (V_i,E_i)) \) we define \( H = (V,E) \) as follows:

\[
V = \bigcup_{i=1}^{n} V_i \quad \text{and} \quad E = \bigcup_{i=1}^{n} E_i \cup \{(x,y) \mid x \in V_i, y \in V_j \text{ and } (v_i,v_j) \in E_0\}
\]

We call \( H_0 \) the outer factor and \( H_1,H_2,\ldots,H_n \) the inner factors, similarly the edges in \( E_i \) are called internal while all other edges of \( E \) are called external. (See Figure 5.)

Every graph \( G \) can be written as the trivial composition \( G = K_1[G] \) or \( G = G[K_1,K_1,\ldots,K_1] \), where \( K_1 \) denotes a digraph on a single node with no arcs. A graph is called decomposable if it can be expressed as a nontrivial composition of some of its induced subgraphs; otherwise, it is called indecomposable.

Theorem 6: Let \( G = G_0[G_1,\ldots,G_n] \) be the composition of disjoint acyclic transitively oriented graphs \( G_i = (V_i,A_i) (i = 0,1,\ldots,n) \). Then \( G \) is an acyclic transitively oriented graph and if \( B \) is an implication class of its undirected version \( G \), then one of the following alternatives holds:

a) \( B \subseteq A_i \) for exactly one index \( i \geq 1 \), or
b) \( B \cap A_i = \emptyset \) for all indices \( i \geq 1 \).

Definition: Let \( \bar{G} = (V,E) \) be an undirected graph, A subset \( Y \subseteq V \) is called partitive if for each \( x \in V \setminus Y \) either there is no \( y \in Y \) s.t. \( (x,y) \in E \) or for every \( y \in Y \) \( (x,y) \in E \). It is clear that \( Y = \{x\} \) for any \( x \in V \) or \( Y = V \) are partitive sets. A partitive set \( Y \) is nontrivial if \( 1 < |Y| < |V| \) (Figure 6).

---

*The graphs may be all directed or all undirected. The definition is the same for both cases, except in the directed case all edges should be considered with their direction.
An internal factor of a decomposition of $\tilde{G}$ is partitive. On the other hand, a partitioning of the vertices $V = \{v_1\} \cup \cdots \cup \{v_k\} \cup Y$, where $Y$ is a nontrivial partitive set induces a nontrivial decomposition of $\tilde{G}$. This proves the following proposition:

**Proposition 7:** $\tilde{G}$ has a nontrivial partitive set if and only if $\tilde{G}$ is decomposable.

**Theorem 8:** If $Y$ is the set of vertices spanned by an implication class $B$ of $\tilde{G} = (V,E)$ then $Y$ is partitive.

**Theorem 9:** An undirected graph $\tilde{G} = (V,E)$ may have at most one color class which spans all of $V$.

We call a comparability graph **uniquely partially orderable** (UPO) if it has exactly two transitive orientations, one being the reversal of the other. It is clear that a comparability graph is UPO iff it has exactly one color class (Figure 7a).

**Definition:** If $\tilde{G} = (V,E)$ is an undirected graph, a subset $U \subseteq V$ is called a **stable set** if for all $x, y \in U$, $(x, y) \not\in E$.

Shevrin and Filippov [15] and independently Trotter, Moore and Summer [18] proved the following theorem:

**Theorem 10:** Let $\tilde{G}$ be a connected comparability graph. The following conditions are equivalent:

i) $\tilde{G}$ is UPO.

ii) Every nontrivial partitive set of $\tilde{G}$ is a stable set.

iii) For every nontrivial decomposition of $\tilde{G}$, each internal factor is a stable set. (i.e., all edges are external).

**Corollary 11:** Let $\tilde{G}$ be a comparability graph. If $\tilde{G}$ is indecomposable, then $\tilde{G}$ is UPO.

Corollary 11 means that if in the decomposition of a graph $\tilde{G}$ none of the inner factors can be nontrivially decomposed, then each of these is a UPO graph.
The following characterizations are known for UPO graphs:

Theorem 12 (Aigner and Prins [2]): Let \( \overline{G} \) be a UPO graph and assume that the complementary graph \( \overline{G}^c \) is not connected. Then \( \overline{G} \) is isomorphic to the complete bipartite graph \( K_{m,n} \) for suitable \( m, n \).

For the case when \( G \) is UPO and \( G^c \) is connected Aigner and Prins [2] found sufficient but no necessary Kuratowski type conditions:

Theorem 13. Let \( G \) be a connected comparability graph, whose complement \( G^c \) is connected and assume that \( G^c \) does not contain a subgraph \( \overline{K}_{1,3} \), then \( G \) is UPO.

These conditions however are not necessary, as the graph \( G \) on Figure 7b shows. The undirected version of \( G \), \( \overline{G} \) is UPO, \( G \) has no nontrivial job-module, while \( G^c \) is connected and contains a \( \overline{K}_{1,3} \) (e.g. the subgraph induced by \( \{2,4,5,6\} \)).

3. The Job-Module Algorithm

In section 1. we have defined the job-modules in a general precedence graph \( G = (J,A) \) (Figure 8).

It is clear that any \( i \in J \) is a job-module and \( J \) itself is a job-module too. We call a job-module \( S \subseteq J \) nontrivial, if \( 1 < |S| < |J| \).

Sidney [16] devised the following decomposition algorithm for the total weighted completion time problem, and Monma [13] proved its applicability to the class of problems defined in Section 1.

Job-Module Algorithm: Let \( A(J,R) \) be a sequencing problem, satisfying the strong ASI, strong SND and consistency properties and the consecutive set assumption, then the following algorithm will find an optimal solution for \( A(J,R) \):

1. Find an \( S \subseteq J \) s.t. \( S \) is a job-module and \( |S| > 1 \).
2. Find an optimum permutation \( \pi \) for \( A(S,R) \), where \( A(S,R) \) is the subproblem of \( A(J,R) \) on \( S \), the precedence relations being those
induced by S.

3. Set $R^0 = R \cup \{(i,k) \mid i,k \in S \text{ and } i \text{ precedes } k \text{ in } \pi\}$

4. Find any optimal sequence for $A(J,R^0)$.

Monma [12] proved that a sequence is optimal for $A(J,R)$ iff it can be generated by the above algorithm. The difficulty of course lies in trying to identify nontrivial job-modules and then optimally sequence them. The following development will answer the first question and characterize the sequencing problems that can be encountered in step 2 of the Job-Module Algorithm.

4. Finding job-modules

In this section we will present an algorithm for finding job-modules. To be able to do this however, we need to present a few more results to link the previous sections together. In the following development if $G = (J,A)$ is a precedence graph we always assume that $A$ contains all the transitive arcs, and denote by $\bar{G} = (J,\bar{A})$ the undirected version of $G$.

Lemma 14. Let $G = (J,A)$ be a precedence graph and let $Y$ be a partitive set of $\bar{G} = (J,\bar{A})$. If $i$ and $j$ are unrelated elements of $Y$ then

i) if $k \in J \setminus Y$ s.t. $(i,k) \in A$ then $(j,k) \in A$ too

ii) if $k \in J \setminus Y$ s.t. $(k,i) \in A$ then $(k,j) \in A$ too.

Proof: i) If $(i,k) \in A$, since $Y$ is a partitive set, we must have $(j,k) \in A$ i.e. $(j,k) \in A$ or $(k,j) \in A$. But the latter case is impossible since $(i,k) \in A$, $(k,j) \in A$ would violate the transitivity of the orientation. ($i$ and $j$ being unrelated (see Figure 9)).

ii) can be proved similarly. $\square$

Lemma 15: If $S \subset J$ is a nontrivial job-module of $A(J,R)$ then $S$ is a nontrivial partitive set of the undirected graph $\bar{G} = (J,\bar{A})$.

Proof: By the definition of $S$ for every $i \in J \setminus S$ we have
a) for every $j \in S$ $(i,j) \in A \Rightarrow (i,j) \in \overline{A}$ or
b) for every $j \in S$ $(j,i) \in A \Rightarrow (i,j) \in \overline{A}$ or
c) for every $j \in S$ $i$ is unrelated to $j \Rightarrow (i,j) \notin \overline{A}$. □

By lemma 15 and Proposition 7, every nontrivial job-module in $G$ defines a nontrivial decomposition of $\overline{G}$ (and $G$), also if $\overline{G}$ is indecomposable $G$ cannot have a non-trivial job-module. Job-modules play the same role in the decomposition of digraphs as partitive sets in the decomposition of undirected graphs. In the following development we will identify those partitive sets which are job-modules, and also characterize the job-modules, which are minimal in the sense, that no proper subsets of them are non-trivial job-modules.

Let $B$ be an implication class of $G = (J,A)$. By theorem 8, if $Y$ is the set of vertices spanned by $B$, then $Y$ is a partitive set of $\overline{G}$. This means that $Y$ partitions $J \setminus Y$ in the following way:

If $C = \{i | i \in J \setminus Y, (i,k) \in \overline{A}$ for every $k \in Y\}$ and $D = \{i | i \in J \setminus Y, (i,k) \notin \overline{A}$ for every $k \in Y\}$ then $C \cap D = \emptyset$ and $J \setminus Y = C \cup D$.

C can be further partitioned:

$C_1 = \{i | i \in C$ and $(i,k) \in A$ for every $k \in Y\}$
$C_3 = \{i | i \in C$ and $(k,i) \in A$ for every $k \in Y\}$
$C_2 = C \setminus (C_1 \cup C_3)$ (Figure 10a).

Lemma 16. Let $Y$ be the set of vertices spanned by an implication class of $G = (J,A)$. Let $J \setminus Y = C_1 \cup C_2 \cup C_3 \cup D$ be the partition defined above. Then for any $j \in C_2$ and $k \in D$ $j$ and $k$ are unrelated in $G$.

Proof: For any $j \in C_2$ we must have $i_1, i_2 \in Y$ such that $(i_1,j) \in A$ and $(j,i_2) \in A$. For any $k \in D$ if we had $(j,k) \in A$ by transitivity we should have $(i_1,k)$ too, contradicting $(i_1,k) \notin \overline{A}$. Similarly if we had $(k,j) \in A$ by transitivity we would have $(k,i_2) \in A$, contradicting $(k,i_2) \notin \overline{A}$. □
Lemma 17. Let \( Y \) be the set of vertices spanned by an implication class \( B \) of \( G = (J, A) \). Let \( J \setminus Y = C_1 \cup C_2 \cup C_3 \cup D \) be the partition defined above. If \( C_2 \neq \emptyset \), then every element of \( C_2 \) has the same relationship with any element of \( Y \). (i.e. if \( j \in C_2 \) and \( i \in Y \) are s.t. \((i,j) \in A\) then this is true for every \( j \in C_2 \) or if \( j \in C_2 \) and \( i \in Y \) are s.t. \((j,i) \in A\) then this is true for every \( j \in C_2 \)).

Proof: We define "cuts" in the subgraph \( G(Y, A) \):

Let \( Y^0_{\text{min}} = \{i | i \in Y \text{ and } i \text{ has no predecessor in } G(Y, A)\} \)

Let \( Y^1 = Y \setminus Y^0_{\text{min}} \) and

\[ Y^1_{\text{min}} = \{i | i \in Y^1 \text{ and } i \text{ has no predecessor in } G(Y^1, A)\} \]

Similarly if \( Y^r \) and \( Y^r_{\text{min}} \) have been defined for some positive integer \( r \) and \( Y^0_{\text{min}} \neq Y \) then let \( Y^{r+1} = Y^r \setminus Y^r_{\text{min}} \) and

\[ Y^{r+1}_{\text{min}} = \{i | i \in Y^{r+1} \text{ and } i \text{ has no predecessor in } G(Y^{r+1}, A)\} \]

It is clear that after a finite number of steps this process ends and we will have an index \( t \) s.t. \( \bigcup_{s=0}^{t} Y^s_{\text{min}} = Y \).

For any \( r \) (\( 0 \leq r \leq t \)) if there is \( j \in C_2 \) and \( i \in Y^r_{\text{min}} \) s.t. \((j,i) \in A\) then by lemma 14. and the definition of \( C_2 \) we must have \((j,i) \in A\) for every \( i \in Y^r_{\text{min}} \) and by transitivity we also have \((j,i) \in A\) for every \( i \in Y^r \).

Let us assume that \( u \) (\( 0 \leq u \leq t \)) is the smallest such index for which there is \( j_0 \in C_2 \) and \( i \in Y^u_{\text{min}} \) s.t. \((j_0,i) \in A\). Also assume that contrary to the statement of this lemma this precedence relation is not true for every element of \( C_2 \), i.e. there is \( k \in C_2 \) s.t. \((i,k) \in A\) for every \( i \in Y^u_{\text{min}} \). By the definition of \( C_2 \) for every such \( k \) there is a smallest \( r \) (\( u < r \leq t \)) s.t. \((k,i) \in A\) for every \( i \in Y^r_{\text{min}} \). Let \( v \) (\( u < v \leq t \)) be the smallest such index over all the possible \( k \)-s of \( C_2 \setminus \{j_0\} \) and \( k_0 \in C_2 \setminus \{j_0\} \) the corresponding element.

Let \( Y_1 = \bigcup_{s=0}^{u-1} Y^s_{\text{min}}, Y_2 = \bigcup_{s=u}^{v-1} Y^s_{\text{min}} \) and \( Y_3 = \bigcup_{s=v}^{t} Y^s_{\text{min}} \) (Figure 10b.).
Since $B$ is an implication class $G_Y = G(Y, B)$ is a connected subgraph of $G$ s.t. for every $i \in Y$ there is an arc $(i, j) \in B$ or $(j, i) \in B$.

Let us consider any arc $(i_1, j_1) \in B$ s.t. $i \in Y_1$ and $j_1 \in Y_3$. This arc cannot be in $\Gamma$ relationship with any $(i_1, j) \in A$ where $j \in Y_1$, since for every $j \in Y_1$ we have $(j, j_1) \in A$ too. Similarly for any $(i, j_1) \in A$ s.t. $i \in Y_3$ we cannot have $(i_1, j_1) \in \Gamma$ $(i, j)$ since $(i_1, i) \in A$. In summary any arc with start point in $Y_1$ and end point in $Y_3$ can be in $\Gamma$ relationship only with arcs having their start in $Y_1$ and endpoint in $Y_3$.

Similar reasoning proves that for any arc with start in $Y_l$ (for any $l$ s.t. $1 \leq l \leq 2$) and endpoint in $Y_m$ (for any $m$ s.t. $2 \leq m \leq 3$) the only arcs it can be in $\Gamma$ relationship with are those with start in $Y_l$ and end in $Y_m$. This means that if $(i, j) \in B$ is s.t. $i$ and $j$ are in different components $Y_l$ (we must have at least one such arc because $G_Y$ is connected) then the only arcs we can have in $B$ must be connecting the same components $Y_l$ as $(i, j)$. From this it follows that either $Y_2 = \emptyset$ or equivalently $Y_3 = \emptyset$, contradicting the existence of $k_0$.

**Theorem 18.** Let $A(J, R)$ be a sequencing problem and $G = (J, A)$ its precedence graph. If $Y$ is the set of points spanned by an implication class of $G$ and $J \setminus Y = C_1 \cup C_2 \cup C_3 \cup D$ is the partition of $J \setminus Y$ defined above then

a) If $C_2 = \emptyset$ then $Y$ is a job-module

b) If $C_2 \neq \emptyset$ then $C_2$ partitions $Y$ into $Y_1$ and $Y_2$ so that every point of $Y_1$ precedes every point of $C_2$ and every point of $Y_2$ succeeds every point of $C_2$, and $Y_1$, $Y_2$ and $C_2$ are job-modules. (Figure 10c).

**Proof:** Case a) is clear by the definition of $C_2$.

If $C_2 \neq \emptyset$ then by lemma 17. $C_2$ partitions $Y$:

$Y_1 = \{i | i \in Y \text{ and } (i, j) \in A \text{ for every } j \in C_2\}$

$Y_2 = \{i | i \in Y \text{ and } (j, i) \in A \text{ for every } j \in C_2\}$

From this by the transitivity of $G$ we have $(i, k) \in A$ for any $i \in Y_1$ and
k ∈ Y₂, and it is clear that Y₁ and Y₂ are job-modules.

For every i ∈ C₁ and k₁ ∈ Y₁ we have (i,k₁) ∈ A and for every i ∈ C₃ and k₂ ∈ Y₂ we have (k₂,i) ∈ A, by transitivity for any j ∈ C₂ we must have (i,j) ∈ A and (j,i) ∈ A. By lemma 16, the elements of C₂ and D are unrelated.

These observations combined prove that C₂ is a job-module too. □

Remark 1: If |Y| = 2 and |C₂| = 1 then each of the job-modules identified in the theorem (Y₁,Y₂,C₂) consists of a single point so these are trivial job-modules. However in this case Y₁, C₂ and Y₂ are single points on a chain in G, therefore it is clear that Y₁∪C₂, C₂∪Y₂ and Y₁∪C₂∪Y₂ are all non-trivial job-modules.

Remark 2: If Y is the set of points spanned by an implication class B of G and in the partition of J \ Y defined above C₂ ≠ ∅, then if we consider the partition of \{Y₁,Y₂\} of Y defined in theorem 18, any minimal job-module within Y must be either in Y₁ or Y₂ and must be a pair of independent elements in G_Y = (Y,B) by theorem 10. If these two elements are also independent in G, then they will be spanned by the implication class of G which contains the arcs between the two elements and the points of C₂. If these two elements are not independent in G, then the arc between them must be in an implication class of G which can span vertices only either in Y₁ or in Y₂.

By using these arguments in an inductive fashion it is clear that any minimal job module within Y₁ or Y₂ will be spanned by an implication class, where the arc(s) spanning the two elements will connect immediate predecessors (or successors) to each of the two elements.

Theorem 19: Let Y be the set of vertices spanned by an implication class B of G = (J,A). Then Y contains a smaller nontrivial job-module iff it contains a job-module of exactly two elements.

Proof: The subgraph G_Y = (Y,B) has clearly exactly one color class B,
therefore it must be a UPO graph. Then by Lemma 15 and theorem 10, any job-module found within $Y$ must be either $Y$ itself or a stable set of $G_y$. Such a stable set is a job-module only if any of its subsets is a job-module, so that any of its subsets having two elements is a job-module. \(\square\)

Corollary 20: If $Y$ is the set of vertices spanned by an implication class $B$ then to determine whether $Y$ properly contains a nontrivial job-module or not, it is sufficient to check all the stable pairs of points within $G_y = (Y, B)$ to see whether they form a job-module or not. It is also clear that any (if there is) one of these job-modules will be minimal.

Theorem 21: If $A(J, R)$ is a sequencing problem whose precedence constraints are represented by the connected, acyclic digraph $G = (J, A)$, and if $\bar{G}$ (the undirected version of $G$) is not UPO then the problem has a nontrivial job-module.

Proof: If $\bar{G}$ is not UPO it must have more than one color class. By theorem 9 only one of these can span the whole $J$, therefore any other color class spans only a proper subset of $J$. Let one of these subsets be $Y$. This subset clearly has to have at least two elements.

If $Y$ has exactly two elements say $i_1$ and $i_2$, these must be connected i.e. $(i_1, i_2) \in A$. If $i_1$ is an immediate predecessor of $i_2$ in $G$ (in the earlier defined partition of $J \setminus Y$, $C_2 = \emptyset$) then $Y$ is a nontrivial job-module. If $C_2 \neq \emptyset$ and $|C_2| > 1$ then by theorem 18. $C_2$ is a nontrivial job-module. If $|C_2| = 1$ then by the first remark after theorem 18. $\{i_1\} \cup C_2$, $C_2 \cup \{i_2\}$ and $\{i_1, i_2\} \cup C_2$ are all nontrivial job-modules.

If $Y$ has more than two elements then by theorem 18. either $Y$ is a nontrivial job-module itself or it partitions into two job-modules $Y_1$ and $Y_2$ and at least one of these is a nontrivial job-module. \(\square\)
Remark: Theorem 21 could be considered a straight corollary of corollary 11 and lemma 15. We decided to detail the above proof because of its constructive nature.

Now we are ready to present the algorithm which will find all the minimal job-modules in polynomial time and space and also define a decomposition of any precedence graph:

Assume that \( G = (J, A) \) represents the precedence relations for the sequencing problem \( A(J, R) \). In the algorithm we will use the notation \( B_k \) for the \( k \)-th implication class of \( G \) and \( Y_k \) for the set of vertices spanned by it and the function

\[
\text{CLASS}(i, j) = \begin{cases} 
0 & \text{if } (i, j) \not\in A \\
k & \text{if } (i, j) \text{ has been assigned to } B_k \\
\text{Undefined} & \text{if } (i, j) \text{ has not yet been assigned}
\end{cases}
\]

Algorithm \textsc{MODSEARCH}:

0. Let \( k = 1 \) and \( B_k = \emptyset \)
1. Select an arbitrary \((i, j) \in A\) s.t. \( i \) is an immediate predecessor of \( j \) in \( G \) and \((i, j) \) has not yet been assigned. If no such \((i, j) \) exists stop.
2. Let \( \text{CLASS}(i, j) = k \) and enumerate the implication class \( B_k \) of \( G \).
3. If \( |Y_k| = 2 \) go to 4. Otherwise, enumerate all the pairs of independent elements in \( G_{Y_k} = (Y_k, B_k) \) and check whether they form a job-module or not. Output those pairs for which the answer is yes. If no such pair was found \( Y_k \) is a job-module, and the only one spanned by \( B_k \). In this case output \( Y_k \) and go to 5.
4. \( Y_k \) is a minimal job-module. Output \( Y_k \) and go to 5.
5. Let \( k = k+1 \), \( B_k = \emptyset \) and go to 1.

Theorem 22. The algorithm \textsc{MODSEARCH} will identify all the minimal job-modules of \( G \) in at most \( O(\delta m + n^2 r) \) running time and \( O(n + m) \) space, where \( n \) is the
number of points, \( m \) is the number of arcs, \( \delta \) is the maximum degree of any point in \( G \) and \( r \) is the number of implication classes of \( G \).

Proof: Any minimal job-module \( Y \) defines a decomposition of \( \tilde{G} \) s.t. the inner factor containing the job-module does not have a nontrivial decomposition. This means by corollary 11. that this inner factor is UPO, meaning that the induced subgraph of \( \tilde{G} \), \( \tilde{G}_Y = (Y, \tilde{A}) \) has one color class. This color class must be a subset of a color class of \( \tilde{G} \). From this it follows that it suffices to search the sets of vertices spanned by the implication classes of \( G \) for potential minimal job-modules. The fact that the algorithm will identify every one of these job-modules is a straight-forward consequence of corollary 20 and the following observation: The requirement that allows in step 1 only the selection of such an arc \((i, j)\) where \( i \) is an immediate predecessor of \( j \) is used to eliminate from consideration such implication classes of \( G \), where the set of vertices \( Y \) spanned by the implication class would be partitioned into \( Y_1 \cup Y_2 \) by a set of vertices \( C_2 \) as in theorem 18. Based on the remarks after theorem 18. every minimal job-module in \( Y_1 \) or \( Y_2 \) will be spanned by other implication classes enumerated by the algorithm.

The only part still to be proved is the bounds for the complexity of the algorithm. This we will do by using some special data structures. We store the adjacency sets of \( \tilde{G} \) as linked lists sorted into increasing order. The element of the list \( \text{Adj}(i) \) which represents the edge \((i, j) \in \tilde{G} \) will have five fields containing \( j \), +1 or -1 depending whether \((i, j) \in G \) or \((j, i) \in G \) resp., \( \text{CLASS}(i, j) \), a pointer to \( \text{CLASS}(j, i) \) and a pointer to the next element of \( \text{Adj}(i) \) (Figure 11). The storage requirement for this data structure is \( O(n + m) \) and the preparation and sorting of the lists can be done in \( O(n + m) \) time. (See e.g. Golumbic [7, pp. 33-36]. For step 1 of the algorithm we prepare a separate data structure also organized as linked lists of adjacency sets sorted into increasing order, but in this list only the immediate
successors will be stored for each point \( i \). We will denote the list for \( i \) by \( \text{Adj}_1(i) \). The preparation of this list can be done in \( O(\delta m) \) time using the previous adjacency lists: For any \( j \in \text{Adj}(i) \) s.t. \((i,j) \in G \) i won't be an immediate predecessor of \( j \) iff there is \( \ell \) s.t. \((i,\ell) \in G \) and \((\ell,j) \in G \). This is so iff there is \( \ell \) s.t. \( \ell \in \text{Adj}(i) \) with the direction indicator being 1 and \( \ell \in \text{Adj}(j) \) with the direction indicator being -1. This means that for each \((i,j) \in G \) to determine whether \( i \) is an immediate predecessor of \( j \) requires \( O(d_i + d_j) \) time where \( d_i \) and \( d_j \) are the degrees of the points \( i \) and \( j \) resp. in \( G \). Since \( O(d_i + d_j) \leq O(2\delta) = O(\delta) \) the preparation of the second adjacency list structure requires at most \( O(\delta m) \) time indeed. As soon as an edge is assigned to an implication class of \( G \) it will be erased from the second Adjacency list structure which again requires at most \( O(\delta) \) time (See e.g. Golumbic [7, p. 35]), so the overall creation and maintenance of the second adjacency list will require at most \( O(\delta m) \) time and clearly \( O(n + m) \) space again. Because of this second list Step 1 of the algorithm can be performed in \( O(1) \) time.

In step 2 of the algorithm we will use the first adjacency list. Each arc \((i,j) \in G \) will be "explored" exactly once (in a breadth first fashion), where by exploration we mean finding all the arcs of \( G \) in \( \Gamma \) relationship with \((i,j) \). For an arc \((i,\ell) \in G \) we have \((i,\ell) \Gamma (i,j) \) iff \( \ell \in \text{Adj}(j) \) and for an arc \((\ell,j) \in G \) we have \((i,j) \Gamma (\ell,j) \) iff \( \ell \not\in \text{Adj}(i) \). If we use two temporary pointers which simultaneously scan \( \text{Adj}(i) \) and \( \text{Adj}(j) \) for values of \( \ell \) satisfying one of the above conditions, then since the lists are sorted and we have pointers to the next element of each list, the exploration of the arc \((i,j) \) can be done in \( O(d_i + d_j) \) time. As soon as a value \( \ell \) is found the corresponding arc \((i,\ell) \) or \((\ell,j) \) gets the CLASS value \( k \) and by using the pointer to its "second copy" that gets the value \( k \) too. Since every arc of \( B_k \) gets explored exactly once and this requires \( O(d_i + d_j) \leq O(\delta) \) time and every
arc of G belongs to exactly one implication class $B_k$ the overall time requirement of finding all the implication classes by iterations through step 2 is $O(\Delta m)$. It is clear that $G_{Y_k} = (Y_k, B_k)$ can be identified at the same time the arcs get assigned to $B_k$, without increasing the time complexity of this step.

To facilitate the fast enumeration of the job-modules in step 3 we use a third special data structure: After the preparation of the second adjacency list structure which shows only the immediate successors for each point of G we prepare two characteristic vectors for each point of G: Vector1(i) will contain the value 1 in position j if i $\in$ Adj1(j) and zero otherwise, while vector2(i) will contain the value 1 in position j if j $\in$ Adj1(i). Since each of these vectors requires for its storage only n binary bits, following the quite common assumption of having bitstrings of length n available, we can store the 2n vectors in $O(n)$ space. We can also assume that the initial setting triggered by an element of the second set of adjacency lists can be performed in $O(1)$ time, so that the initial preparation of these characteristic vectors can be done in $O(m)$ time too. Having these 2 characteristic vectors available for each point, to decide whether a pair of points \{i, j\} forms a job-module can be done in $O(1)$ time: \{i, j\} is a job-module iff vector1(i) is identical to vector1(j) bit by bit with the possible exception of the i-th and j-th bits and vector2(i) is identical to vector2(j) bit by bit with the possible exception of the i-th and j-th bits. The comparison needed to determine whether these conditions are satisfied for a pair \{i, j\} can be done in $O(1)$ using the vector strings as numbers. Since the number of point pairs to be checked is at most $O(n^2)$, an iteration through step 3 of the algorithm requires at most $O(n^2)$ time. (This shows that the complexity for this step will not increase if we check every pair \{i, j\} s.t. i, j $\in$ $Y_k$, instead of checking only the independent pairs in $G_{Y_k}$.)
of iterations $k$ through the algorithm will be at most the number of implication classes for $G$, which is an invariant of the graph $G$ no matter in what sequence the implication classes get enumerated. If we denote this constant by $r$ than the overall complexity of the algorithm will have the following components:

Initial data preparation: $O(n + m + \delta m)$

Iterations through Step 1: $O(1) \ r$

  "   Step 2: $O(m)$

  "   Step 3: $O(n^2) \ r$

  "   Step 4: $O(1) \ r$

  "   Step 5: $O(1) \ r$

This proves the theorem. $\Box$

Remark 1: If we use only one major iteration of $MODSEARCH$ (for a fixed value of $k$) then this requires no more than one iteration through each step of $MODSEARCH$ resulting in a reduced time complexity of $O(\delta m + n^2)$. One such iteration will identify all the minimal job-modules within the set of vertices spanned by the implication class identified in this iteration. If we need only one such job-module $MODSEARCH$ may be stopped after the first one has been identified resulting in further potential savings.

Remark 2: One major iteration of $MODSEARCH$ suffices to answer the question whether $G$ is UPO since this is true iff $|B_1| = m$. It also suffices to determine whether $G$ is indecomposable since this is true iff $G$ is UPO and no nontrivial job-module was identified in the first major iteration of $MODSEARCH$. This means that both problems can be answered by $MODSEARCH$ in $O(\delta m + n^2)$ time.

Remark 3: The Algorithm as defined above will identify all the minimal job-modules in $G$. Some of these can "overlap", i.e. some points of $G$ can be in several of these minimal job-modules. If we wanted to use the job-modules for
a decomposition of $G$, then we need only such a set of job-modules, which cover every point of $G$ at most once. To get such a set of job-modules some additional implication classes of $G$ can be left out of the enumeration in the algorithm: If a set of points $Y$ is spanned by an implication class already enumerated and in the earlier defined decomposition of $J \setminus Y (= C_1 \cup C_2 \cup C_3 \cup D)$ $C_2 = \emptyset$ then $Y$ defines a decomposition of $G$, where all the arcs between $Y$ and $J \setminus Y$ will be external to this decomposition. Therefore all these arcs can be deleted from consideration for additional implication classes. This way every pair of points can be considered at most once for minimal job-modules, so that the iterations through step 3 of the algorithm will require at most $O(n^2)$ time. Since in every major iteration of the algorithm at least one nontrivial job-module is identified the algorithm will find a decomposition (where none of the inner factors is decomposable further) after at most $O(n/2)$ major iterations. In summary this means that this revised version of the algorithm will result in the above mentioned decomposition of the graph $G$ in at most $O(\delta m + n^2)$ time and $O(n + m)$ space.

5. Applications of MODSEARCH in the Job-Module Algorithm

The Job-Module Algorithm requires the identification of only one job-module in each iteration through its step 1, and although we could use the full MODSEARCH algorithm which identifies every minimal job-module, in many cases of the Job-Module algorithm the precedence graph will become completely sequenced without the direct sequencing of every minimal job-module. Therefore in the following development we will assume that in every application of MODSEARCH we identify only the job-modules within one implication class and substitute these into step 1 of the Job-Module algorithm one by one. We continue with MODSEARCH for another implication class only if this is necessary. We may identify the following types of job-modules:

a) Two unrelated jobs $i$ and $j$ ($(i, j) \not\in A$)
b) Two related jobs $i$ and $j$ ($(i, j) \in A$)

bl. $(i, j) \in A$ and $j \preceq i$

b2. $(i, j) \in A$ and $i \preceq j$

c) A job-module having more than two jobs with the corresponding points representing the set of vertices spanned by an implication class of the precedence graph. (If this is the only job-module then the precedence graph is indecomposable and UPO.)

In case a, we can simply arrange the two jobs into their optimal sequence by ordering them according to the preference relation by $\pi$ and add the resulting additional precedence relations to $R$ as described in step 3 of the Job-Module Algorithm. (Adding additional arcs to $G$ will cancel certain $\Gamma$ relationships, and so will result in the partitioning of some implication classes of the old precedence graph.)

If we have the case bl. we can apply Theorem 2, from which we know that there is an optimal permutation of the jobs in $J$ in which $i$ is an immediate predecessor of $j$. Therefore the jobs $i$ and $j$ can be combined into a new compound job $ij$, contracting the arc $(i, j)$ into a node $ij$, deleting any arc incident with $i$ or $j$ and adding the new arc $(k, ij)$ for every $k$ s.t. $(k, i) \in A$ and $(ij, k)$ if $(j, k) \in A$.

If we have a b2. type job-module $Y = \{i, j\}$, then the preference relation agrees with the precedence relation between $i$ and $j$. In this case let us consider the partition of $J \setminus Y$ created by the job-module $Y$:

$$C = \{k | (k, i) \in A \text{ or } (k, j) \in \bar{A}\}$$

$$D = \{k | (k, i) \notin A \text{ and } (k, j) \notin \bar{A}\}$$

$$J \setminus Y = C \cup D \text{ and } C \cap D = \emptyset.$$ 

$C$ can be partitioned further:

$$C_1 = \{k | (k, i) \in A\}$$

$$C_3 = \{k | (j, k) \in A\}$$
\[ C_2 = C \setminus (C_1 \cup C_3) \]

Since \( Y = \{i, j\} \) is a job-module we must have \( C_2 = \emptyset \) in this partitioning. If we also have \( D = \emptyset \), then it is clear that in every feasible permutation of \( J \) \( i \) must be an immediate predecessor of \( j \), and so as before the jobs \( i \) and \( j \) can again be combined into a new compound job \( ij \), deleting the nodes \( i \) and \( j \) and all arcs incident with them, and adding the arc \((k, ij)\) for every \( k \in C_1 \) and adding the arc \((ij, k)\) for every \( k \in C_3 \).

If \( D \neq \emptyset \) then some of the jobs in \( D \) may be between \( i \) and \( j \) in an optimal permutation of \( J \), therefore in this case we cannot contract the jobs \( i \) and \( j \) into a compound job.

If MODSEARCH does not find job-modules of the first two types (a or b) then it clearly means that the current problem \( A(J, R) \) has no such job-modules and the corresponding graph \( G \) is either a UPO graph with no nontrivial partitive sets or \( G \) is the union of such UPO graphs. If the Job-Module Algorithm encounters such precedence graphs, some of the earlier known efficient solution methods may be applicable (e.g. Monma's transformations [12] or suboptimization over \( p \)-minimal subsets [10,16] as the next iterative step, possibly resulting in a new precedence graph which has type a, or b, job-modules. If all such attempts fail, it is always possible to use some well-known dynamic programming or branch and bound technique to do the suboptimization of the current job-module. Having found an optimal permutation \( \pi_1 \) of the jobs in a type c job-module \( Y \), we can add all the precedence relations defined by \( \pi_1 \) on \( Y \) to the current precedence relation set \( R : R = R \cup \{(i, j)|i,j \in Y \text{ and } i \text{ precedes } j \text{ in } \pi_1\} \).

Assuming that we are able to optimally sequence the type c job-modules by one of the methods mentioned, we can categorize the situations encountered in the Job-Module Algorithm with MODSEARCH in the following way:
Case 1. A job-module is identified which can be optimally sequenced and as a result of this, new precedence relations are added to the current precedence relation set. (In some of the cases e.g. bl, some nodes and arcs can get deleted.)

Case 2. A b2 type job-module is identified which cannot be sequenced optimally. (D ≠ ∅)

If a b2 type job-module Y = \{i_1, i_2\} is identified with D ≠ ∅, then the arc (i_1, i_2) cannot be in relationship with any other arc of the current graph G and so (i_1, i_2) is an implication class by itself. (Figure 12). In this case we can perform the next major iteration of MODSEARCH which will identify another implication class and job-module(s) within the set of vertices spanned by it. We can keep doing this until we can identify a job-module which belongs to case 1., i.e. it can be optimally sequenced. It is impossible for the graph to have only job-modules belonging to case 2, since this would mean that the graph decomposes into a union of implication classes, all of these consisting of a single arc, which is possible only if the graph was completely sequenced by its arcs contradicting the fact that D ≠ ∅ for all the b2 type job-modules.

In summary a graph cannot have only job-modules belonging to case 2, so that applying MODSEARCH to different implication classes of the graph we must find a job-module belonging to case 1. In this case we always either add at least one new arc to the original precedence graph (type a and type c job-modules) or in the case of a bl type job-module reduce the number of points in the graph by 1. Therefore it is clear that case 1 can happen at most 

$$|J|^2 - |A|^2 + |J|$$

times, where G = (J,A) denotes the original precedence graph. This means that after a finite number of operations through cases 1. and 2. the Job-Module Algorithm will end by completely sequencing the original precedence graph. We can summarize the observations in the previous
development in the following form:

Theorem 23: Let $\mathcal{F}$ be the family of sequencing problems solvable by the Job-Module Algorithm and let $\mathcal{F}_1 \subseteq \mathcal{F}$ be the sub-family in which the precedence graph of the problems is indecomposable (and UPO). If we have an algorithm $\alpha$ capable to sequence any member of $\mathcal{F}_1$ in a polynomial amount of time and space (as a function of the size of the sequencing problem) then the Job-Module Algorithm with MODSEARCH in combination with $\alpha$ solves any problem in $\mathcal{F}$ in polynomial amount of time and space.

Corollary: The total weighted completion time problem is NP-complete if the set of precedence relations is represented by an arbitrary graph, which is indecomposable (and UPO).

Proof: Lawler [10] proved that the total weighted completion time problem with general precedence constraints is NP-complete. In light of the previous theorem the complexity of the total weighted completion time problem with indecomposable (and UPO) precedence graphs is the same as with general precedence relations. $\blacksquare$

In the following we show that the Job-Module Algorithm using MODSEARCH for job-module identification gives a polynomial time algorithm for the class of "series-parallel" graphs and we will also show some "non-series parallel" graphs for which the algorithm works, proving its applicability beyond the class of "series-parallel" graphs.

Lawler, Tarjan and Valdes [11] used the following definitions to develop the theory of "series-parallel" graphs:

i) The directed acyclic graph having a single vertex and no edges is MSP.

ii) If $G_1 = (V_1, A_1)$ and $G_2 = (V_2, A_2)$ are two MSP digraphs, $V_1 \cap V_2 = \emptyset$, so is either one of the graphs constructed by the following operations:
a) Parallel composition: \( G_P = (V_1 \cup V_2, A_1 \cup A_2) \)

b) Series composition: \( G_S = (V_1 \cup V_2, A_1 \cup A_2 \cup (M_1 \times M_2)) \), where \( M_1 \) is the set of maximal elements of \( G_1 \) and \( M_2 \) is the set of minimal elements of \( G_2 \).

Definition: The arc \((u,v)\) of an acyclic directed graph is redundant if there is a directed path from \( u \) to \( v \) in the graph that does not include the arc \((u,v)\). A digraph not containing any redundant arcs is called minimal. The transitive reduction of a digraph \( G \) is the unique minimal digraph having the same transitive closure as \( G \).

Definition (General Series Parallel) (GSP) digraphs): A directed acyclic graph is GSP iff its transitive reduction is MSP.

Definition: A digraph \( G = (V,A) \) is complete bipartite if \( V \) can be partitioned into \( H \) and \( T \) so that \( A = H \times T \). The set \( H \) is called the head and \( T \) is called the tail of \( G \).

Definition: (Complete Bipartite Composite (CBC) digraphs). A directed acyclic graph \( G \) is CBC if there exists a set of complete bipartite subgraphs of \( G \): \( B_1, B_2, \ldots, B_k \) called the bipartite components of \( G \), such that

a) each arc of \( G \) belongs to exactly one bipartite component;

b) every vertex \( v \) of \( G \), except the maximal elements, belongs to the head of exactly one bipartite component denoted \( h(v) \);

c) every vertex \( v \) of \( G \), except the minimal elements, belongs to the tail of exactly one bipartite component denoted \( t(v) \).

Lemma 24: Let us consider a sequencing problem \( A(J,R) \) and its precedence graph \( G = (J,A) \). Assume that \( G \) is GSP and let \( S \subseteq J \) be a job-module of \( G \). If \( \pi \) is an optimal sequence for \( A(S,R) \) and \( A_1 = A \cup \{(i,k) | i,k \in S \text{ and } i \text{ precedes } k \text{ in } \pi \} \) then \( G = (J,A_1) \) is a GSP.
Proof: Let $G_0 = (J,A_0)$ be the transitive reduction of $G$. Similarly, let $G_2 = (J,A_2)$ be the transitive reduction of $G_1$.

It is clear that $G_0$ is MSP by definition and Lawler et.al. [11] proved that every MSP digraph is CBC too. So $G_0$ is CBC. Let $i_0$ and $j_0$ be the first and last vertex resp. in the sequence $\pi$. By the minimal property of $G_2$ clearly no vertex other than $i_0$ or $j_0$ from $S$ can be connected to any vertex from $J \setminus S$. So the induced subgraph of $G_2$, $G_3 = (J \setminus S,A)$ is the same as the subgraph of $G_0$ induced by the same set $J \setminus S$. Therefore $G_3$ is CBC. On the other hand the subgraph of $G_2$ induced by $S$ is a simple chain, and for every vertex in $J \setminus S$ it is true that it either precedes $i_0$ or succeeds $j_0$ by $S$ being a job-module. Combining these observations it is clear that $G_2$ is CBC.

We will call the graph shown on Figure 13 N, for obvious reasons. Lawler et.al. [11] proved that an MSP graph cannot contain N as an induced subgraph, so $G_0$ cannot contain N as an induced subgraph. So the only way $G_2$ could contain N as an induced subgraph if this N would contain at least one new arc added. However $S$ induces a chain as a subgraph of $G_2$ and since every vertex from $J \setminus S$ again either precedes or succeeds the points of $S$ it is clearly impossible for $G_2$ to contain N as an induced subgraph. Again Lawler et. al. [11] proved in the same paper that a CBC digraph is MSP iff it does not contain N as an induced subgraph, so $G_2$ must be MSP, which completes the proof of the Lemma. \square

Theorem 25. The Job-Module Algorithm with MODSEARCH will always solve a sequencing problem $A(J,R)$ in polynomial time if its precedence graph $G = (J,A)$ is GSP.

Proof: It is clear from the definition of GSP graphs that they always must contain a nontrivial job-module. Since every induced subgraph of a GSP graph
is again GSP, it can be seen through a simple induction, that MODSEARCH always will produce a type a) or b) job-module for a GSP graph, but never can produce a type c) job-module for it. This fact in combination with Lemma 24 and Theorem 23 proves the theorem. ⊓⊔

Remark: The theorem does not suggest solving the sequencing problem with GSP precedence graphs by the Job-Module Algorithm, since more efficient methods [13] are known for this case.

Monma [12] recently presented a set of transformations based on ordinal information, which can be useful to reduce the size of a general problem or simplify the precedence structure. However he also showed that no combination of these transformations (or any other algorithm based only on ordinal information) will guarantee the solution of a sequencing problem with "non-series-parallel" precedence graphs, while for GSP graphs he gives a combination of his transformations always guaranteeing the solution. The following example is a demonstration for the first part of this statement.

Example 1. Consider a sequencing problem on four jobs with a precedence graph N as shown on Figure 13. If the processing times \( p_i \) and their weights \( w_i \) are such that

\[
p_3/w_3 < p_4/w_4 < p_2/w_2 < p_1/w_1
\]

then it cannot be decided using only ordinal information whether job 1 should be followed by job 2 or the other way around in the optimal sequence. The actual numbers shown in Table I. show two problems satisfying the above ordinal sequence and the optimum sequence is \((2,1,3,4)\) for the first one and \((1,4,2,3)\) for the second one. ⊓⊔

Monma [12] also shows that if the data is such that the ordinal sequence is different from the above, a combination of his transformations will always solve the problem. Naturally if we use in addition other than ordinal
information then the above problem could always be optimally sequenced by Monma's transformations and this additional information:
The only case when the use of ordinal information is not sufficient, is when \( p_3w_3 < p_4/w_4 < p_2/w_2 < p_1/w_1 \). The question to be answered in this case is whether (2,1,3,4) or (1,4,2,3) is the optimal sequence. This can be determined by the following test we call the N-test:

\[(1,4,2,3) \text{ is at least as good as (2,1,3,4) iff } w_2(p_1+p_4) + p_4w_3 \leq p_2(w_1+w_4) + p_3w_4.\]

The validity of the test is clear if we substitute the full cost function in detail.

Now consider a graph which is not GSP but the non-GSP components are simple Wheatstone bridges, one such graph is shown as an example on Figure 14. These graphs contain induced N-s, so based on ordinal transformation they cannot always be sequenced optimally. On the other hand, applying the Job-Module Algorithm and MODSEARCH in it, plus using the N-test and Monma's transformations in the GSP algorithm for N-type job-modules, all these graphs can always be sequenced optimally.

We introduce more formally two new families of precedence graphs which properly contain the classes of MSP and GSP graphs resp.:

i) The directed acyclic graph having a single vertex and no arc is WMSP.

ii) The directed acyclic graph shown in Figure 13 is WMSP.

iii) If \( G_1 = (V_1,A_1) \) and \( G_2 = (V_2,A_2) \) are WMSP digraphs and \( V_1 \cap V_2 = \emptyset \), then either one of the following directed acyclic graphs is WMSP too:

a) Parallel composition: \( G_p = (V_1 \cup V_2, A_1 \cup A_2) \)

b) Series: \( G_s = (V_1 \cup V_2, A_1 \cup A_2 \cup (O_1 \times I_2)) \), where \( O_1 \) is the set of exit nodes in \( G_1 \) and \( I_2 \) is the set of entry nodes in \( G_2 \).
Definition (WGSP digraphs):

A directed acyclic digraph is WGSP iff its transitive reduction is WMSP.

Theorem 26. Let A(J,R) be a total weighted completion time problem with a WGSP precedence graph. Then the Job-Module Algorithm with MODSEARCH in combination with the N-test solves the problem in polynomial amount of time and space.

Proof: The theorem is a straightforward corollary of theorems 23 and 25, if we substitute the N-test into the algorithm α of theorem 23. □

Remark: It is clear that the Job-Module Algorithm can be replaced by the appropriate set of Monma's transformations (those solving the sequencing problem with GSP precedence graphs), which combined with the N-test will solve any problem with WGSP precedence graph in polynomial amount of time and space.

Finally we present an interesting example which is not solvable by

1. An algorithm based strictly on ordinal information [12],
2. The Decomposition Algorithm into p-minimal initial sets [16,10], or
3. The Job-Module Algorithm with MODSEARCH.

On the other hand the combination of 1. and 3. or the algorithm defined for WGSP will find the optimal solution:

Example 2. Let us consider the precedence graph on Figure 15a with the following data:

\[
\begin{array}{cccccccc}
J_1 & w_1 & J_2 & w_2 & J_3 & w_3 & J_4 & w_4 & J_5 & w_5 \\
2 & 4 & 3 & 10 & 1 & 4 & 1 & 39/10 & 1 & 5
\end{array}
\]

Clearly \( J_3/w_3 < J_4/w_4 < J_2/w_2 < J_1/w_1 \) and \( J_5/w_5 < J_4/w_4 \).

Applying Monma's transformation T2 [12] for the "locally minimal" node 5, we get the modified precedence graph shown on Figure 15b. For this problem the data is
<table>
<thead>
<tr>
<th>P(1,5)</th>
<th>W(1,5)</th>
<th>P2</th>
<th>W2</th>
<th>P3</th>
<th>W3</th>
<th>P4</th>
<th>W4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>39/10</td>
</tr>
</tbody>
</table>

For this we have $p_3/w_3 < p_4/w_4 < p_2/w_2 < p(1,5)/w(1,5)$, which belongs to the case shown to be unsolvable based only on ordinal information. It also does not have a p*-minimal set other than the full set of nodes, so Lawler's [10] algorithm is not applicable either. On the other hand, applying MODSEARCH to the original graph of Figure 15a, it will identify \{4,5\} as a nontrivial job-module. The optimal sequence for this is clearly 5 preceding 4, and the resulting new problem is shown on Figure 15c. Here we have $p_5/w_5 < p_3/w_3 < p_2/w_2 < p_1/w_1$, so we have a solvable case of the N-type precedence graph. (By Monma's transformations e.g.).
<table>
<thead>
<tr>
<th></th>
<th>( P_1 )</th>
<th>( w_1 )</th>
<th>( P_2 )</th>
<th>( w_2 )</th>
<th>( P_3 )</th>
<th>( w_3 )</th>
<th>( P_4 )</th>
<th>( w_4 )</th>
<th>Optimal sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>( (2,1,3,4) )</td>
</tr>
<tr>
<td>Problem 2</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>( (1,4,2,3) )</td>
</tr>
</tbody>
</table>

\[(a, b) \Gamma (a, d)\]

\[(a, b) \text{ forces } (a, d)\]

\[(b, a) \Gamma (c, a)\]

\[(b, a) \text{ forces } (c, a)\]

Figure 1
(b, a) \Gamma (b, c) \quad \text{while} \quad (b, a) \not\in (d, c) \quad \text{but} \\
(b, c) \Gamma (d, c) \quad \text{while} \quad (b, a) \not\in (d, c) \quad \text{but} \\
(b, a) \Gamma^* (d, c), \quad \text{since } \Gamma^* \text{ must be transitve.}

\textbf{Figure 2}

Directing \((a_0, b_0)\) as \((a_0, b_0)\) forces to direct \((a_5, b_5)\) as \((a_5, b_5)\) in order to maintain transitivity.

\textbf{Figure 3}
The internal arcs are shown by solid lines while the external ones by dashed lines.

**Figure 5**

\[ Y = \{a_2, a_3, a_4\} \text{ is a partitive set, but } Y = \{a_2, a_3\} \text{ is not.} \]
A UPO graph.

G is not UPO since (a, b) can be directed either way, it forms a color class by itself.

Figure 7a

G is UPO

G is connected and contains \( K_{1,3} \)

Figure 7b
$J = \{a_1, a_2, a_3, a_4\}$ is a job-module

$\{a_1, a_2, a_3\}$ is not a job-module

Figure 8

Figure 9
The partition of $J$ by a nontrivial partitive set $Y$.

$C = C_1 \cup C_2 \cup C_3$

Figure 10a.

Figure 10b.
For clarity only non-transitive arcs are shown.

Figure 10c.
Example for the data structures in MODSEARCH:

Figure 11
A b2. type job-module, where \{(i, j)\} is an implication class by itself.

Figure 12.

Figure 13.
Figure 15b.

Figure 15c.
References


Continued on Page 2...


Continued on Page 3...
154. Szendrovits, A.Z. and Drezner, Zvi, "Optimizing N-Stage Production/Inventory Systems by Transporting Different Numbers of Equal-Sized Batches at Various Stages", April, 1979. Continued on Page 4...


Continued on Page 5...


187. Love Robert F., Dowling, Paul D., "Optimal Weighted \( \ell_\infty \) Norm Parameters For Facilities Layout Distance Characterizations", April, 1982.


