The Fifth Conference on Operations Research and its Military Applications

On the M-Machine Scheduling for Dags with Bounded Number of Maximal Complete Bipartite Subgraphs

BY

S. M. Khamis, E. M. El-Sherif,
M. M. Kouta and B.I. Bayoumi.

ORGANIZED BY

MILITARY TECHNICAL COLLEGE
Cairo -Egypt
23-25 November 1993

1MILTAEY TECHNICAL COLLEGE
CAIRO-EGYPT

COR -6 | 314

PROCEEDJNGS Of TIJE FIFTH ORMA CONFE R E NCE
23 - 25 November 1993

On the M-Machine Scheduling for Dags with Bounded Number of Maxi mal Complete Bi partite Subgraphs

S.M .Khamis*, E.M El-Sherif*, M.M. Koutat and B.I. Bayoumi*.

ABSTRACT.

In this paper an upper bound on the number of solutions of m-machine scheduling problem is found, where the precedence relations between the tasks can be represented by a Complete Bipartite Composite, CBC, dag. ln fact, a CBC dag represents an N-free partially ordered set.

Also a polynomial time algorithm ($O(n^2.k!)$) to get an optimal solution is presented, where n is the number of tasks and k is the number of blocks (bounded) of the CBC dag.

Key wonts.

Scheduling problem, partially ordered sets (posets), Complete Bi partite Composite. (CBC) dags, N-free posets, totally-interacted graphs, m-machine, identical processors and N P-complete problem.

* Department of Mathematics, Faculty of Science.
Ain Shams University, Cairo, Egypt.

t Department of Computer Science and Operations Research
Military Technical College, Cairo, Egypt.

§1. INTRODUCTION.

Technological progress has made it possible to cons1ruct Computers with a large number of processors. The intention is to Make use of the apparent-mutual independence of many activities in (sequential or parallel) programs or task systems, thus achieving shorter over all execution times. Because of this hardware, time trade off is one of the main justifications to construct parallel computers, the scheduling problem, i.e. the problem to assign activities to processors such that to respect their inherent precedence constraints and simultaneously to minimize time, has attracted considerable practical and theoretical interest [9].

Scheduling of partially ordered unit time jobs on in machines, aiming at minimal schedule length, is known to be one of the notorious combinatorial optimization problems, for which the complexity status is still unresolved. The problem is known to be NP-hard when the number of processors is unbounded [l 3]. Available results give polynomial time algorithms for special classes of partial orders: e.g., when the precedence constraints form a tree [4]. In [11] a linear time algorithm (O($|V|+|E|$)) is given for scheduling a dag (V,E) whose incomparability graph is chordal.

For m = 2, the problem has been solved efficiently by several algorithms (see [4] and [5]). In [15], a polynomial time algorithm (O($n^{(m-1)h+1}$)) for scheduling a general dag of n tasks and of bounded height h on a bounded number of identical processors, m, is given. Also, a polynomial time algorithm (O(n^{m-1})) for scheduling level forests with n tasks and c components (trees) on a fixed number, m, of identical processors where C<m is presented in [7].

In section 2, some basic definitions and terminologies are demonstrated. Calculation of an upper bound on the number of solutions of the m-machine scheduling for N-free posets is given in section 3. Section 4 contains a polynomial time algorithm (O(n^2.k!)) to find an optimum schedule where n is the number or vertices and k is the bounded number of maximal complete bipartite subgraphs. Finally, the appendix contains an illustrative example.

§2. DEFINITIONS AND TERMINOLOGIES.

We recall the following definitions, terminologies, concepts and notations from [1], [2], [3], [7] and [10].

A *partially ordered set (poset)* P is an ordered pair P=(T,<), where T is the (finite) set of elements and < is the partial ordering relation on T. If u < v in P then v is called a *successor* of u or u is a *predecessor* of v, and in addition if there is no w in T such that u < w < v then v is an immediate successor (cover) of u (u <• v) or u is an *immediate predecessor* of v. *ImPred(u)* and *ImSuc(u)* denote the sets of all immediate predecessors and all immediate successors of u respectively. *Min(P)* and *Max(P)* denote the sets of all minimal and all maximal elements of P respectively.

Two elements u, v ∈ T are *comparable* in P (denoted by u ~v) if u < v or v < u. Otherwise they are *incomparable* (denoted by u ‖v). A set of pairwise comparable (incomparable) elements is called a chain (an antichain). A Linear extension LE of a poset P 1s a total ordering of the elements of P such that u < v in P => u < v in LE.

A graph is denoted by G = (T,E), where T is vertex set and E is edge set. A *dag* is a directed *acyclic graph.* A dag is *transitively closed* if (u,v), (v,w) ∈ E => (u,w) ∈ E, and *transitively reduced* if (u,w) ∈ E => there is no v ∈ T with (u,v) , (v,w) ∈ E. Thus every poset P=(T,<) may be interpreted as a dag with vertex set T and edge set <.

Grillet, in [6], introduced the class of posets which satisfy the CAC property, (i.e. each maximal chain meets each maximal antichain). Grillet also showed that a poset has CAC property iff it does not contain a subposet on four elements a, b, c, d (say) with a < b, c <• b, c < d and a ‖c , a ‖d, b ‖d. In 1973, this characterization has been strengthened in [8] to be a <•b, c <•b, c <•d, and a ‖c , a ‖d, b ‖d (see Fig. 1). This poset looks like the letter 'N'. So, the term N-free was introduced in [12]. Thus a poset P has

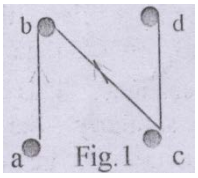

Fig.1

the CAC property iff it has no "N" in its diagram as an induced subdag. This forbidden structure is a necessary and sufficient condition for the sets of immediate predecessors(successors) to form a partition of T. More precisely, for any two points x,y ∈ T the sets of immediate predecessors (successors) of x , y are either disjoint or identical.

Another equivalent property of N-free posets has been given in [10] and [14]. The class *of Complete Bipartite Composite* (CBC) *dags* has been defined as the class of dags G=(T,E) for which there is a set {C1 , ... , C_k} of maximal complete bipartite subgraphs of G (called the *complete bipartite components of* G) such that

a) every edge of G belong to exactly one component,
b) for each non-sink vertex v, all edges leaving v belong to the same component,
c) for each non-source vertex v, all edges entering v belong to the same component.

It is clear that, *"A dag is CBC iff it is the transitive reduction of an N-free pose "*, [10]. A dag with this characterization is, also, introduced in [7], and is known as *totally-interacted* graph . Consequently any information or prope1ty for N-free poset is true For its transitive reduction (i.e. for CBC dag).

Let P=(T,<) be a finite N-free poset. A *block* of P means a maxi mal complete bipartite subgraph in the dag of P. A block has the form (A_i, B_i) where A_i , $B_i \subseteq$ T are such that A_i = lmSuc(u) for all u ∈ B_i and B_i = ImPred(v) for each v ∈ A_i. By convention, (Min(P),∅) and (∅, Max(P)) are also blocks. The block representation of *P (B(P))* is a 2×k matrix (k is the number of blocks), with the A_i 's in the first row and the B_i's in the second row in some order such that for all x ∈ T if x ∈ A_i and x ∈ B_j then i < j (see [2]). obviously every bipartite component C_i of a CBC dag is a block. Clearly apart from a possible permutation of the columns of B(P) every N-free poset P has a unique representation. Using the block representation one can define a k × k matrix M(P) called *the matrix representation*, M(P) = [m_{ij}], where m_{ij} = $|A_i \cap B_j|$. Note that m_{ij} = 0 whenever i≥j, that is, M(P) is a super diagonal matrix. Again M(P) is unique up to a possible permutation applied simultaneously

to the rows and the columns. Fig. 2 represents a poset P and its block and matrix representations.

It is clear that, (see [3]), if $m_{i,i+1}=0$, for any i < k, i.e. $A_i \cap B_{i+1}=\emptyset$, then the two blocks (A_i, B_i) and (A_i, B_{i+1}) can be interchanged. This is equivalent to interchange the rows i and i+1 and similarly the columns i and i+1 of M(P). Otherwise if $A_i \cap B_{i+1} \neq \emptyset$, i.e. $m_{i,i+1} \neq 0$, then the ith block must precede the (i+1)st block and the representation of P is unique and there are no possible permutations. The poset in this case is also an *interval order* (i.e. it does not contain two parallel edges).

A_i	1, ..., 6	7	8	9,10	11	12	13	14	∅
B_i	∅	2,3	4	5	1,7,8	6,9,10	12	11,13	14

B(P)

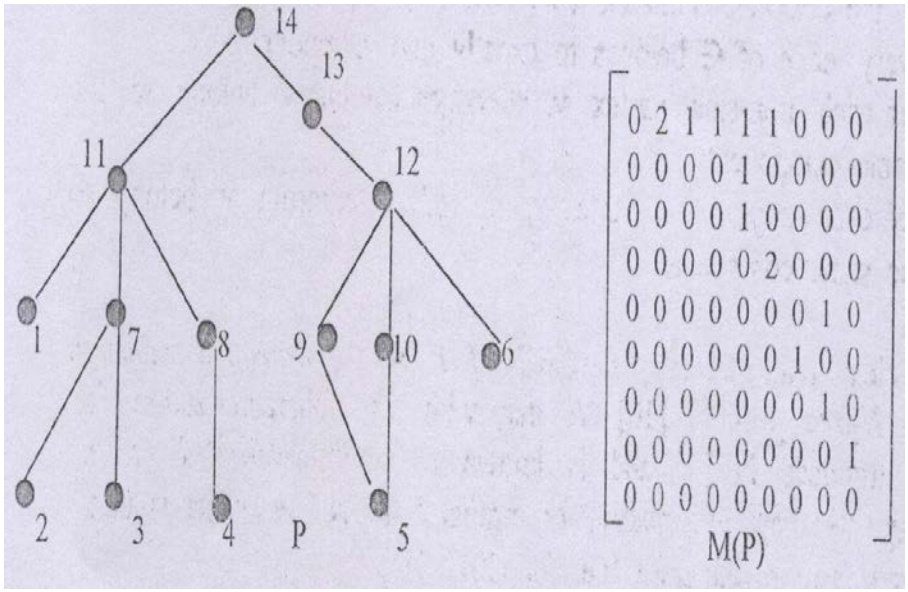

Fig 2: A poset P and its block and matrix representations.

Now, let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of tasks, where each task t_i requires one unit of execution time and let the partial ordering relation, $<\bullet$ on T be the precedence constraints on these tasks. The relation $t_i <\bullet t_j$ means that the execution of t_j cannot be started until t_i is finished. The poset P = (T, <) ls usually represented by a dag D = (T,E) where $E = \{(t_i, t_j): t_i t_j \in T \wedge t_1 <\bullet t_j\}$, i.e. D is transitively reduced. informally, a schedule or an assignment for a given computation is a description of the work done by some processors. Of course the schedule must not violate the precedence relations given by the partial order and it is not permitted to assign more than one processor to a task or more than one task to a processor at any time. formally, a schedule for P on m processors (m∈N) is a surjective mapping $s : T \rightarrow \{1, 2, ..., L\}$,

defined by

(i) $\forall t_i, t_j \in T, t_i < t_j$ in P => $s(t_i) \leq s(t_j)$;

(ii) $1 < |s^{-1}(r)| \leq m \forall r \in \{1, 2, ..., L\}$.

L is called the *length of the schedule* and the objective is to Ind s with minimum L. A schedule of minimum length is called an *Optimum* schedule, r ∈ {l, 2, ... , L}, is called a *time slot,* $s(t_i) = r$ Means that t_i will be executed during the slot r and $s^{-1}(r)$ is the set of tasks to be executed by the schedule s during the slot r Any slot that contains m tasks is called *complete or samrated.* Otherwise, it contains $m - |s^{-1}(r)|$ gaps. A task t_i is said to be *available* at time u if all the predecessors of t_i have been executed before time u.

Let D be a given CBC dag. The tasks in D are partitioned into *Layers* $\gamma_0, \gamma_1, \ldots, \gamma_h$, where h is the height of D. The layer γ_0 is the set of minimal tasks in D. γ_i is the set of available tasks in D after removing all the tasks in layers $\gamma_0, \gamma_1, \ldots, \gamma_{i-1}$. The set of tasks in γ_i can be partitioned into two disjoint subsets W_i and R_i where $W \in W_i$ iff w has at least one immediate successor in γ_{i+1}. R_i is the complement of W_i with respect to y. The set of tasks in W_i can be decomposed into working subsets $\zeta_i^1, \zeta_i^2, \ldots, \zeta_i^q$ for some q, $1 \leq q \leq |W_i|$ A working subset in γ_i is the set of tasks in W_i which are the all (common) immediate predecessors of one (or more) task in γ_{i+1}. Removing the tasks of a ζ_i^j makes at least one task from $\gamma_{i+1}, 1 \leq j \leq q, 0 \leq i \leq h-1$ available. A task $v \in R_i$ is said to be a reserve task for ζ_i^j if there exists an immediate successor of v which is a successor of some task $w \in \zeta_i^j$, and p_i^j is the set or the reserve tasks of ζ_i^j Observe that removing all the tasks in R_i. does not make any task from γ_{i+1} available. For example the layer structure of the dag in Fig. 2 is

i	γ_i	W_i	R_i
0	1, ..., 6	2,3,4,5	1,6
1	7,8,9,10	7,8,9,10	∅
2	11,12	12	11
3	13	13	∅
4	14	14	∅

	ζ_i^k			p_i^k		
i \ k	1	2	3	1	2	3
0	2,3	4	5	1	1	6
1	7,8	9,10		∅	∅	
2	12			11		
3	13			∅		
4	14			∅		

Clearly, from Fig.2, each B_i, $2 \leq i \leq k$ contains exactly one working

Subset. A reserve task appears in exactly one B_i, $2 \leq i \leq k$ when A_i is the set of its immediate successors. Thus, it is easy to show that:

Lemma 1.

There exists 1-1 correspondence between the working subsets of a CBC dag and its blocks. □

This relation help to fi11d an upper bound for the number of solutions, N(k) , of m-machine scheduling problem for a CBC dag with k blocks. If k is bounded, the correspondence leads, also, to construct a polynomial ·time algorithm for finding an optimum schedule (see §4).

§3 An Upper Bound for N(k).

In this section, the authors found an upper bound for the number of solutions of the m-machine scheduling problem of a CBC dag with k blocks. First, recall the following facts that proved in [7].

Let ζ_i^j (t) be the jth working subset at γ_i which is available to be Executed at slot t. Let S be an optimum schedule. If S executes all the tasks in ζ_i^j(t) during the slots t, t+1, ... , $t+\lceil|\zeta_i^j|/m\rceil - 1$, then we say that S *discloses* ζ_i^j(t) starting from slot t. If at the beginning of every slot, S starts by disclosing one working subset then we say that S obeys the disclosure property. A dag D is said to possess the disclosure property if there always exists an optimum schedule of D that obeys the disclosure property.

In [7], Kouta proves that "*totally - interacted graph, CBC dag, possesses the disclosure property* " and that " *there ah1Y.1ys exists an optimum schedule that starts by executing working subsets rather than reserve subsets* ".

Theorem 2.

The number of solutions, N(k), for scheduling a CBC dag with n unit time tasks and k blocks on m identical processors does not exceed k^{3k}

Proof

The proof depends on the above facts. Assume that the first i slots are saturated. Let E be the number of available blocks

(containing the available tasks). According to the second fact, above, the set of available blocks can be partitioned into two subsets: the set of W blocks that contain the available working subsets and the set of R blocks that contain unavailable working subsets but the reserve tasks are available, i.e. E = W + R. Note that some of the E blocks might be used at any previous slots but not totally executed. So, for simplicity the unexecuted parts of these blocks will be considered as new available blocks. To fill the next slots starting with slot i+1, there are W ways to choose a first working subset (block). Let ζ be the chosen working subset.

In the case, when $|\zeta|/m \notin [$N, according to disclosure property, the slots i+1, ... , i + $\lceil|\zeta|/m\rceil$-1 will be saturated, but the slot number i + $\lceil|\zeta|/m\rceil$= r (say) is not, i.e. the number of ways to fill the slots i+1, ... , r-1 is W. Now, the gaps in the slot r can be filled using the remaining (W-1) blocks. This can be done by (W-1) ways. If again, the slot r is not saturated, there will be (W-2) ways to fill the gaps, and so on. If the W working subsets are executed and the slot r is still unsaturated, then the gaps will be filled from the reserve tasks. So, the total number U(r-i) of ways lo choose the first working subset and fill the (r-i) slots; from i+1 to r does not exceed

$$W \cdot (W-1) \cdot (W-2) \cdot \ldots \cdot R \cdot (R-1) \cdot (R-2) \cdot \ldots ,$$

where the number of product terms e (r-i) (say) is equal to the exact number of blocks that used to saturate the group of (r-i) slots; from i+1 to r. Since each of W or R does not exceed k, then

$$U(r\text{-}i) < k^{\theta(r-i)}$$

To find a schedule, we repeat this procedure until all tasks are scheduled. Thus the maximum number of ways to fill all slots in a schedule does not exceed

$$\Pi \cup(r\text{-}i) < \Pi\, k^{\theta(r-i)} = k^{\Sigma\theta(r-i)}$$

where the product and summation are taken over all possible groups of slots, (r-i). It is clear that the more complicated case occurs when the last block used to saturate the last slot of any group is not totally executed. Therefore

Σ e(r-i) = the total number of blocks having working subsets,
+ the total number of blocks having reserve tasks,
+ the total number of blocks where some of their tasks

(not all) has been used to saturate some slots.

From lemma 1 the first term is exactly equal to k. The second term does not exceed k because reserve tasks (if they exist) arc distributed in some A_j's, $1 \leq j \leq k-1$. Also, the last term depends on the total number of totally executed blocks and so it is also bounded by k. Hence

$$\Sigma e\,(r-i) < 3\,k \qquad \square$$

§4 An Algorithmic Solution of the Problem.

In this section, an algorithm to find an optimal solution of The m-machine scheduling problem is given, where the precedence constraints can be represented by a CBC dag with bounded number of maximal complete bipartite subgraphs (blocks).

The following identifiers will be used:

D : the given CBC dag with tasks $u_1, u_2, \ldots, u_n$.
h(D) : the height of D.
k : the number of blocks in the block representation of D.
m : the number of machines (identical processors) in each slot.
CL : the length of current schedule (number of slots).
OL : the length of an optimum schedule (the minimum number of slots).
LB : the theoretical lower bound of OL.
CS : the current solution of the problem.
OS : an optimal solution of the problem.

The Algorithm.

Input : The transitive reduction of the dag **D** (a list of immediate predecessors of each element in D), and the number or identical processors in each slot.

Output : The first optimal solution of the m-machine scheduling problem, i. e. an optimum schedule and its length.

Method : Constructing some linear extensions via creating possible permutation to be applied on M(D). One of these linear extensions must be an optimal solution of the problem.

(Note : The details of an efficient algorithm that constructs all needed permutations to be applied similarly on rows and columns or the matrix representation of CBC dag can be found in [2]).

Step 1 : Construct the blocks of the input dag D by grouping all elements with the same set of immediate predecessors in one block (A_i, B_i), (see [3]).

Step 2 : Construct the block representation of D which is the 2×k matrix B(D) = $[b_{ij}]$, where $b_{1j} \leftarrow A_j$ and $b_{2j} \leftarrow B_j, 1 \le j \le k$.

Step 3 : Construct the matrix representation M(D) from B(D) : M(D) = $[m_{ij}]$, where $m_{ij} \leftarrow |b_{1j} \cap b_{2j}|, 1 \le i \le j \le k$.

Step 4 : Initially, let OL ← n, and LB ←max $\{\lceil n/m \rceil$, h(D)+ l$\}$.

Step 5 : Get the first permutation Π (identity mapping) at which Π(i) ← i , 1≤i≤k.

Step 6 : Set the linear extension L corresponding to Π by arranging the tasks as follows :

$$L \leftarrow \{u_1^2, u_2^2, \dots, u_{l_2}^2, u_1^3, \dots, u_{l_3}^3, u_1^k, \dots, u_{l_k}^k\},$$

where $l_i = |B_{\pi(i)}|$, 2≤i≤ k and $u_j \in B_{\Pi(z)}, 1 \le j \le l_z$.

Step 7 : Construct the current solution CS from L by taking sequentially its elements to fill the current slot S_i (1≤i≤n) by m tasks if it is possible. Notice that an element u in L cannot be entered to S_i if one (or more) of its predecessors is not scheduled yet or exists in this slot. In this case, search for a new task in L which can be assigned to S_i . If thereis no such task and remain unassigned elements, put i ← i+ 1 and begin to fill a new slot until all tasks are scheduled. Then let CL ← i and CS ← $\{S_1 , S_2 , \dots , S_i \}$.

Step 8 : If CL = LB then go to *Step 13.*

Step 9 : If CL < OL then OL←CL, and OS ←CS.

Step 10 : Get a next possible permutation Π, (see [2]), which can Not produce non-zero element in the lower triangle of M(D). If there is no such Π go to *Step 13.*

Step 11 : Apply Π similarly to rows and columns of M(D) to obtain $\bar{M}$(D); [$\bar{m}_{i,j}$] $[m_{\Pi(i),\Pi(j)}]$.

Step 12 : If $\bar{M}$(D) = M(D), (i.e. the permutation Π does not produce a schedule with length smaller than OL), then go to *Step 10.* Otherwise go to *Step 6.*

Step 13 : Output and stop.

Obviously, the complexity status of the algorithm depends on *Step 7* and the Number of Accepted Permutations, NAP. It is clear that *Step 7* modifies a constructed linear extension L (if it is

possible) to obtain the current schedule CS_2 with minimum idle processors. Therefore, there exist at most n^2 comparisons of the n tasks. It is, also, clear that NAP depends on the number of independent blocks, i.e. blocks that can be interchanged together. So it does not exceed k!. So, we have the following result.

Theorem 3.

The above algorithm finds an optimal solution of m-machine scheduling problem for a CBC dag with n tasks and k (bounded) blocks in a time $O(n^2 .k!)$. □

Conclusion.

1) It is well-known, [14], that a poset P is series-parallel iff every induced subposet of P is N-free. In fact, the class of series-parallel posets is the smallest class of partial orders that contains one-element and is closed under parallel and series compositions. In [7], Kouta proved that an optimum schedule for series-parallel posets of height h and width w ≤ f(m) on m identical processors can be found by an algorithm of complexity

$(n \cdot (F(m,h)^m \cdot (h+2)^{F(m,h)}))$, where F(m,h)≤ h•(f(m)-1)+1. For bounded h and fixed m, series-paallel poset can be scheduled in a polynomial time and when m is unbounded, the problem is NP-complete (see [9]). Since the class of series-parallel posets is a subclass of the N-free posets class then one can use the given algorithm to schedule series parallel posets with k (bounded) blocks on m (fixed, bounded or unbounded) identical processors in a polynomial time.

2) Every poset P can be embedded into a larger N-free poset by subdividing each edge in the transitive reduction D of P by a new task. This process can be easily done by a linear time algorithm O(A), where A is the number of edges in D (see [lO]). So, the algorithm can (by making a little change in *Step 7* to deal with new tasks which are dummy ones) schedule any general poset that can be embedded into an N-free one with bounded number of blocks.

3) The above algorithm also solves the unit execution time scheduling problem for N-free interval orders in polynomial time. This problem was solved in [11] by an algorithm with complexity $O(n+|E|)$, where |E| is the number of edges which is bounded by n(n-1) / 2 (i.e. $O(n^2)$ in general).

Appendix.

Finally, we have the following illustrative example. Applying the algorithm to the dag in Fig. 2, and taking m=3, it would run as follows:

Step 1 to *Step 3* : Construct block and matrix representations of the given dag (see fig. 2).

Step 4 and *Step 5* : Initiate, OL = 14, LB = 5 and Π : 123456789, (the identity permutation).

Step 6 : L is the set of elements in B_1 , B_2 , ... , B_9

L = { 2, 3, 4, 5, 1, 7, 8, 6, 9, 10, 12, 11, 13, 14}.

Step 7 : CS = { (2,3,4) , (5,1,7) , (8,6,9) , (10,11) , (12) , (13) , (14) } and CL = 7.

Step 8 : CL ≠5, continue.

Step 9 : CL < 14 then OL = 7; and
OS = { (2,3,4) , (5,1,7) , (8,6,9) , (10,11) , (12) , (13) , (14) }

Step 10 : the next permutation interchanges B_5 and B_6 ; Π : 123465789.

Step 11 and *Step 12* : the condition $\bar{M}$(D) = M(D) is not satisfied and the algorithm goes back to *Step 6*.

Step 6 : the algorithm gets the next linear extension L that is the set of elements in $B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9$, respectively.

L = { 2, 3, 4, 5, 6, 9, 10, 1, 7, 8, 12, 11, 13, 14}.

Step 7 : CS = { (2,3,4), (5,6,1), (9,10,7), (8,12), (11,12), (14) } and CL = 6.

Step 8 : CL is still greater than LB

Step 9 : the current solution is better than the previous one and
OS = { (2,3,4) , (5,6,1) , (9,10,7) , { (8,12) , (11,13) , (14) };
OL = 6.

After ten permutations, the algorithm gets Π = 142356789 and then produces the following results :

L = { 5, 2, 3, 4, 1, 7, 8, 6, 9, 10, 12, 11, 13, 14 },
CS = { (5,2,3), (4,1,7), (8,6,9), (10,11), (12), (13), (14}), and
CL = 7.

The algorithm continues and the value of CL varies between 6 and 7. It does not reach the LB because the given dag has a unique maximum task, so the algorithm creates 40 (all possible) permutations and then it halts. The result of the algorithm is
OS : S_1 = 2,3,4 S_2 = 1,5,6 S_3 = 7,9,10 S_4 = 8,12 S_5 =11,13 S_6 = 14.

Note that : The number of created permutations 40 is less than 9!, (9! = 362880). In general the permutations are made among blocks, and the total number of created permutations is much less than k!.

REFERENCES

[1] B. I. Bayoumi, (1993), "An Algorithm To Comparability Graphs of N-free Posets", Congressus Nurnerantuim, to appear.

[2] 13. I. Bayoumi, M. H. El-Zahar and S. M. Khamis, (1989), "An Algorithm To Count Prime N-free Posets", The 24 th Annual Conference on Statistics, Computer Science and Operation Research, Egypt, p. 27-49.

[3] B. I. Bayoumi and S. M. Khamis, (1992), "Testing Isomorphism and Recognition of N-free Posets[11] . The 27th Annual Conference on Statics, Computer Science and Operations Research, p. 106 -117.

[4] E. G. Coffman and J. R. Graham, (1973), "Optimal scheduling for Two Processor Systems", Acta Information,1,3, p. 200-213.

[5] H. Gabow, July (1982), "An Almost Li11ear Algorithm for Two Processors Scheduling", JACM vol. 29, no. 3, p. 766-780.

[6] P. A. Grillet, (1969), "Maximal chains and antichains", Fund. Math. 65, p. 157-167.

[7] M. M. Kouta, (1985), "Scheduling Unit Execution Time Tasks Subject to Precedence Constraints", Ph. D. Thesis, Clarkson University.

[8] B. Leclerc and B. Monjardet (1973), "Orders CAC", Fund. Math. 11-22.

[9] E. Mayer, Sep. (1981), "Well Structured Parallel Programs Are Not Easier to Schedule", Department of Computer Science, Stanford University.

[10] R. H. Mohring, (1989), "Computationally tractable classes of ordered sets", (I. Rival ed.), Algorithms and Order, Kluwer Academic Publishers, p. 105-193.

[11] C. H. Papadinitriou and M. Yannakakis, (1979), "Scheduling Interval-Ordered Tasks", SIAM J. Comp., p. 405-2109.

[12] I. Rival, (1982), "Optimal linear extensions by interchanging chains", Proc. Amer. Math. Soc. 89, p. 387-394.

[13] J. D. Ulman, (1975), "NP-Complete Scheduling Problems", J. Comp. and System Science 10, p. 384-J95.

[14] J. Valdes, R. E. Tarjan, and E. L. Lawler, (1982), "The Recognition of Series-Parallel Digraphs", SJAM J. Comput. 11 p. 298-313.

[15] M. K. Warmuth, Aug. (1981), "Scheduling on Profile of Constant Breadth", PHD. Thesis, Department of Computer Science, University of Colrado , Boulder.