Journal of Universal Computer Science, vol. 10, no. 2 (2004), 106-119
submitted: 11/6/03, accepted: 29/12/03, appeared: 28/2/04 © J.UCS

A Message-optimal Distributed Graph Algorithm: Partial Precedence Constrained Scheduling[1]

Pranay Chaudhuri
(University of the West Indies, Cave Hill Campus, Barbados
pchaudhuri@uwichill.edu.bb)

Hussein Thompson
(University of the West Indies, Cave Hill Campus, Barbados
hthompson@uwichill.edu.bb)

Abstract: This paper presents a distributed algorithm for the partial precedence constrained scheduling problem. In the classical precedence constrained scheduling problem all the dependent tasks must be scheduled before the task itself can be scheduled. The partial precedence constrained scheduling problem is a generalized version of the original precedence constrained problem in the sense that the number of dependent tasks to be scheduled before the task itself can be scheduled is considered a variable. Using a directed graph to model the partial precedence constrained scheduling problem in which n nodes represent the tasks and e edges represent the precedence constraints, it is shown that the distributed algorithm requires O(e) messages and O(n) units of time and is optimal in communication complexity to within a constant factor.

Keywords: Distributed algorithm, directed graph, scheduling, task, precedence constraints
Categories: GT: Algorithms, SD:G.1.0, G.2.2, F.2.2

1 Introduction

Consider a directed graph, G = (V, E), with V a nonempty set of nodes and $E \subseteq V \times V$ the set of edges. Let $\|V\| = n$ and $\|E\| = e$. Without loss of generality, we assume V to be {1, 2, . . ., n}. The *precedence constrained scheduling* (PCS) problem is defined as follows: Given a set of n tasks denoted by {T(1), T(2), . . . , T(n)} and a set of precedence constraints, we are required to schedule each task to get the minimum total time for completion of all the tasks satisfying the given constraints, let us call this time the job completion time. This problem can be modelled by a directed graph G as follows. Each task T(i), $i \in V$, requires a duration d(i) time units for its completion. The set of precedence relations is given in the form of directed edges (i, j) $\in$ E and costs C(i, j) which indicate that the task T(i) can start only C(i, j) units of time after the completion of task T(j). T(l) is considered as the root task signifying that once task T(l) is completed the job is completed. Clearly G cannot have any cycle, since a cycle implies that a task can be started only after the completion of itself. This problem has widespread applications and can be solved in O(n + e) sequential time [Chakrabarti, 99; Coffman, 76]. Note that the well known Critical Path Scheduling (CPS) problem may be easily modelled by the PCS problem described above [Smith, 89].

---

[1] A preliminary version of this paper was presented in the 2003 Design, Analysis and Simulation of Distributed Systems Conference, Orlando, FL [Chaudhuri, 03]

The *partial precedence constrained scheduling* (PPCS) problem considers that in order to schedule a task it is not always necessary that all the dependent tasks have been scheduled before it can be scheduled. Instead, it is assumed that a task can be scheduled only after the completion of a given number, NUM(i), of its dependent tasks (c.f., figure 1). This is a more general problem, since when NUM(i) includes all the dependent tasks of T(i) the problem degenerates to that of the standard precedence constrained scheduling problem. This more general problem was first introduced in 1999 and a sequential O(e log n) time algorithm was also proposed [Chakrabarti, 99]. To the best of our knowledge no distributed algorithm has been reported in the literature for this problem.

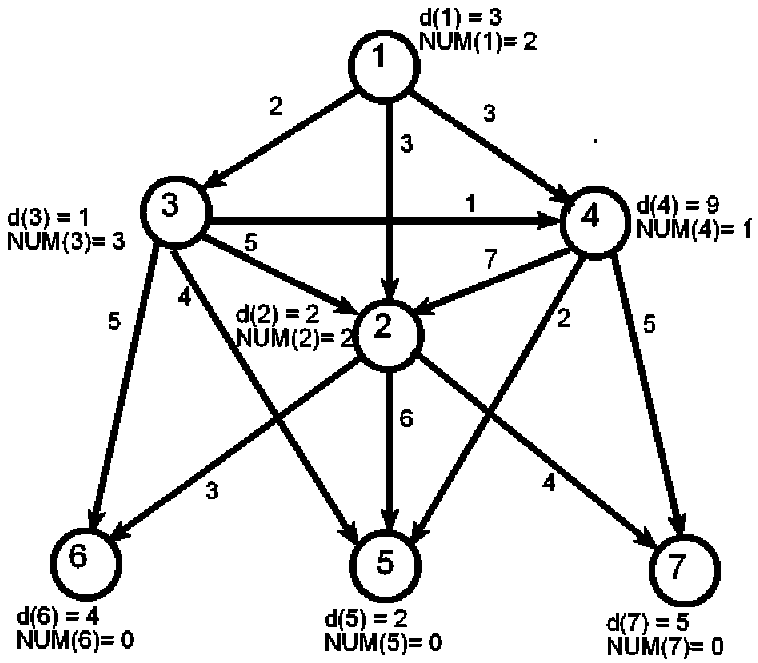

Figure 1 : An arbitrary precedence graph G_1. Each node represents a task with a certain duration and each directed edge together with the associated cost represents a precedence constraint. The minimum number of dependent tasks that must be completed before the task itself can be scheduled is provided with each task.

In this paper, a distributed algorithm for the more general PPCS problem is proposed that requires O(e) messages and O(n) time. The rest of the paper is organized as follows. Section 2 introduces the model of computation and the distributed algorithm for the PPCS problem is presented in Section 3. The proof of correctness and the complexity related issues are provided in Section 4 and Section 5 illustrates the algorithm with the help of an example. Section 6 examines some variations in the PPCS problem. Finally, Section 7 concludes the paper.

2 Model of Computation

The computational model used in this paper is an asynchronous network of processors with their own local memory, such that each processor of the network corresponds to a particular node of the digraph (that models the precedence constrained scheduling problem) under consideration. Moreover, corresponding to each edge of the digraph, one bidirectional, non-interfering communication link is assumed. A communication link between a pair of node processors i and j consists of two channels: one for transmitting messages from i to j and one for transmitting messages from j to i. No common or global memory is shared by the node processors for interprocessor communication. Instead, this is done by exchanging messages. It is assumed that the network is sufficiently reliable so that there is no node or channel failure during transmission. We consider a very simple protocol for message communication, similar to that used in [Dijkstra, 80; Misra, 82]. If node X sends a message to a neighbor node Y, then the message gets appended at the end of the input buffer of Y and this process takes a finite arbitrary time (due to the transmission delay). Two or more messages arriving simultaneously at an input buffer are ordered arbitrarily and appended to the buffer. A node receives a message by removing it from the corresponding buffer.

A simple message format is assumed for inter-node communication. A message is defined as a triple and is expressed as:

<type; sender id; parameter>.

The parameter field of a message depends on its type field. The exact content of the parameter field for different types of messages will be discussed in Section 3.

In such a computational model, two types of complexity measures are important. One is the time complexity and the other is the message or communication complexity. We assume that the insertion of an element in a given sorted list, so that the resulting list after the insertion also remains sorted is a primitive operation. Precisely, this insertion can be implemented by using a binary search like operation in O(log k) time if the given sorted list consists of k elements. However, later we shall see that the number of elements in the given sorted list prior to the insertion operation is always less than the maximum degree of any node in the digraph G. Although theoretically the maximum degree of a node in G can be n - 1, for all practical purposes it can be considered as no more than a small constant k and, as a consequence the assumption that is, the time required for the insertion operation can be considered as constant. However, the most important assumption in this model is that each processor processes messages from its neighbors, performs local computations and sends messages to its neighbors such that no time is required for all these actions. In order to obtain the time complexity in this model, we assume that the delay between the time any message is transmitted along any edge and the time it is processed at its destination is at most one time unit. It may be noted that this assumption is for obtaining complexity only and it has no effect on the correctness of the algorithm. That is, the algorithm will work correctly even without the assumptions regarding the transmission delay and local computations, etc. The communication or message complexity is the total number of messages (independent of types) sent during algorithm execution. This type of a computational model, or similar concepts

have been extensively used for solving various graph theory and related problems (see, e.g. [Awerbuch, 85; Chaudhuri, 94; Chaudhuri, 97; Chaudhuri, 98; Dijkstra, 80; Korach, 84; Misra, 82; Sharma, 89; Tel, 00]).

3 The Algorithm

The distributed algorithm following for the PPCS problem presented in this section uses the following terminology.

PRED(i): The set of predecessors of node i, i.e., PRED(i) = $\{j \mid j \in V \wedge (j, i) \in E\}$.

SUCC(i): The set of successors of node i, i.e., SUCC(i) = $\{j \mid j \in V \wedge (i, j) \in E\}$.

NUM(i): The minimum number of dependent tasks, represented by the successor nodes, which must be completed before task represented by node i can be initiated. Note that if $|SUCC(i)| \geq 1$, then $NUM(i) \geq 1$.

C(i, j): The time i must wait after j's completion before i itself can start.

LIST(i): The list of the sum of the completion time of task j and cost C(i, j), $\forall j \in$ SUCC(i), sorted in increasing order.

ST(i): Earliest possible starting time of task represented by node i.

CT(i): Earliest possible completion time of task i.

d(i): The duration of task i.

JCT: The minimum total time required for completion of the job satisfying all the constraints.

In addition, we use the following functions:

insert(x, y): Inserts element x in a sorted list y such that the resulting list also remains sorted.

view(L, k): Returns the kth largest element of list L. Duplicate entries are taken into account.

The algorithm uses the following types of messages:

SCHEDULE: A SCHEDULE message from node j to node i, where $i \in$ PRED(j), informs node i about the earliest possible completion time of the task represented by node j.

TERMINATE: A TERMINATE message from node j to node i, where $i \in$ SUCC(j), provides with node i the minimum total time required for completion of the job satisfying all the constraints and also informs node i that node j has terminated its algorithm.

The parameter field of a SCHEDULE message from node j to node i carries the earliest possible completion time of task j, i.e., CT(j). On the other hand, the parameter field of any TERMINATE message carries the earliest possible completion time of the root and hence of the job. A detailed description of the algorithm is given below. Although, we have written the algorithm for the root node (task) $r \in V$ separately, the nodes which have no successors (i.e. the terminal nodes) act as the initiator nodes. However, all the nodes of G act as the terminator nodes. The algorithm is terminated when all the nodes of G have terminated.

Algorithm D_SCHEDULE:

Input: The set of predecessors, PRED(i), the set of successors, SUCC(i), and the minimum number of dependent tasks which must be completed before task represented by node i can be initiated, i.e. NUM(i), are available at each $i \in V$.

Output: At the termination of the algorithm, each node $i \in V$ knows the earliest starting and completion times of task T(i) and also the minimum total time required for completion of the job satisfying all the constraints.

Algorithm for root: r

```
Initialization
do
   count(r):= |SUCC(r)|;
   ST(r):= CT(r):= 0;
od


Upon receiving <SCHEDULE, i, CT(i)> message
do
  count(r) := count(r) - 1;
  insert(CT(i) + C(r, i), LIST(r));
        if count(r) = 0 then
           ST(r):= view( LIST(r), (NUM(r));
           CT(r):= ST(r) + d(r);
           JCT:= CT(r);
                for each j ∈ SUCC(i) do
                    send <TERMINATE, r, CT(r)> to j;
                od
             terminate
        fi
od
```

Algorithm for node i ($i \neq r$):

```
Initialization
do
 count(i) := |SUCC(i)|;
```

```
 termc(i) := |PRED(i)|;

ST(i):= CT(i) := 0;
       if count(i) = 0 then
        ST(i) := 0;
        CT(i):= d(i);
           for each j ∈ PRED(i) do
              send <SCHEDULE, i, CT(i)> to j;
           od
        fi
od


Upon receiving <SCHEDULE, j, CT(j)> message
do
 count(i) := count(i)-1;
 insert(CT(j) + C(i, j), LIST(i));
        if count(i) = 0 then
          ST(i) := view( LIST(i), (NUM(i));
          CT(i) := ST(i) + d(i);
             for each j ∈ PRED(i) do
                send <SCHEDULE, i, CT(i)> to j;
             od
        fi

od


Upon receiving <TERMINATE, j, CT(r)> message

do

         if termc(i) = |PRED(i)| then
           JCT:= CT(r);
         fi

  termc(i) := termc(i) - 1 ;

        if termc(i) = 0 then
            for each j ∈ SUCC(i) do
                  send <TERMINATE, i,CT(r)> to j;
             od
           terminate
         fi
od
```

Note that in PPCS the final optimal solution may not require to schedule all the tasks. However, in the classical precedence constrained scheduling, every task must be scheduled before the root task can be scheduled.

4 Correctness and complexity

In this section, we first establish the correctness of the algorithm and then provide an analysis of the complexity of the algorithm. In this section it is also shown that the algorithm is optimal in communication complexity to within a constant factor.

Lemma 4.1: Every node $i \in V - \{k \mid SUCC(k) = \emptyset\}$ receives a SCHEDULE message from a node $j \in SUCC(i)$ within a finite time.

Proof: According to algorithm D_SCHEDULE, a node $x \in \{k \mid k \in V \wedge SUCC(k) = \emptyset\}$ in G initiates a SCHEDULE message after initializing some local variables. Node x sends this SCHEDULE message with parameter field CT(x) to each of its predecessor nodes $y \in PRED(x)$. Each internal node $i \in V - \{r\} \cup \{k \mid SUCC(k) = \emptyset\}$ upon receiving a SCHEDULE message from each of its successor nodes finds the earliest time when task T(i) can be started. This is basically the NUM(i)-th smallest earliest start time obtained from the sequence of the earliest starting times of all the successor tasks of task T(i). This information is then transmitted in terms of a SCHEDULE message to all the predecessor nodes of node i. Since every node which has no successors initiates the algorithm by sending a SCHEDULE message to each of its predecessor nodes and the network is assumed to be sufficiently reliable with finite transmission delay, each predecessor node receives SCHEDULE messages from all of its successor nodes within a finite time. With this as the basis, we now complete the proof by induction.

Assume that an internal node $i \in V - \{r\} \cup \{k \mid SUCC(k) = \emptyset\}$ has received SCHEDULE messages from all of its successor nodes SUCC(i). Then according to algorithm D_SCHEDULE, node i sends a SCHEDULE message to each of its predecessor nodes PRED(i) after some local computations. All these local computations take no time according to the assumption made in the computational model. Therefore, each node $j \in PRED(i)$ must receive a SCHEDULE message from each node $i \in SUCC(j)$ within a finite time.

We have seen that the lemma holds for every node $x \in \{k \mid k \in V \wedge SUCC(k) = \emptyset\}$ (since each of these nodes has no successors, there is no question of any SCHEDULE message arriving at these nodes) and every node which is a predecessor of node x. Also we have shown that if the lemma holds for any internal node $i \in V - \{r\} \cup \{k \mid SUCC(k) = \emptyset\}$ then it also holds for all of its predecessors $j \in PRED(i)$. Therefore, the lemma follows. □

Lemma 4.2: Every node $i \in V - \{r\}$ receives a TERMINATE message from a node $j \in PRED(i)$ within a finite time.

Proof: From lemma 4.1 it follows that within a finite amount of time the root node $r \in V$ receives SCHEDULE messages from all of its successors SUCC(r). Once the root node r receives SCHEDULE messages from all of its successors it performs some local computations and then terminates its algorithm after sending a TERMINATE message to all of its successors with the earliest completion time of the root task (which is also the same as that of the job). With this as the induction basis, we can prove the lemma by using a same line of reasoning as in Lemma 4.1, the only

difference being that the propagation of TERMINATE messages would be in the opposite direction to that of SCHEDULE messages. □

Lemma 4.3: At the termination of algorithm D_SCHEDULE, every node $i \in V$ knows ST(i), CT(i), and JCT.

Proof: It can be verified from algorithm D_SCHEDULE that during the propagation of SCHEDULE messages, starting from those nodes which have no successors until the root node is reached, each node $i \in V$ computes ST(i) and CT(i). While computing the earliest starting time ST(i) of task i, represented by node $i \in V$, NUM(i) is used. Recall that NUM(i) indicates the minimum number of dependent tasks (i.e., tasks dependent on task i) which must be completed before task i can be scheduled. In order to find ST(i), a list LIST(i) containing CT(j) + C(i, j), $\forall j \in$ SUCC(i), is created. The NUM(i)-th smallest entry in LIST(i) corresponds to the earliest starting time of task i, i.e. ST(i). Note that the task represented by node i cannot be scheduled until at least NUM(i) of its dependent tasks represented by the successors of node i have been completed. Once ST(i) is available CT(i) is simply obtained by adding with ST(i) the duration d(i) of task i. In the final phase of the computation when TERMINATE messages are received by a node i from all of its predecessors PRED(i), it simply copies the CT(r) from the parameter field of the TERMINATE message which gives the job completion time denoted by JCT. Thus, when algorithm D_SCHEDULE terminates each node $i \in V$ knows ST(i), CT(i) and JCT(i.e. CT(r)). Hence the lemma follows. □

Lemma 4.4: Algorithm D_SCHEDULE eventually terminates.

Proof: Follows directly from Lemma 4.2. □

Theorem 4.1: Algorithm D_SCHEDULE is correct.

Proof: Follows from Lemmas 4.3 and 4.4. □

The message and time complexities of algorithm D_SCHEDULE are considered in the following theorem.

Theorem 4.2: Algorithm D_SCHEDULE requires O(e) messages and O(n) time.

Proof: It can be verified from algorithm D_SCHEDULE that SCHEDULE messages are initiated by the nodes of G which have no successors and propagated up to the root node. Once the root node receives SCHEDULE messages from all of its successors it terminates its algorithm and prior to that it initiates TERMINATE messages which are propagated up to the terminal nodes (i.e., the nodes with no successors). So every edge of G carries exactly one SCHEDULE message and exactly one TERMINATE message during the execution of the algorithm. Therefore, the message complexity of algorithm D_SCHEDULE is $2e = O(e)$.

Assuming that a message takes at most one time unit to travel from a node to its neighbor, the total time required by algorithm D_SCHEDULE is equal to the

maximum distance between the root r and any node of G. Since this distance cannot exceed n-1, the time complexity if algorithm D_SCHEDULE is O(n). □

Theorem 4.3: Algorithm D_SCHEDULE is optimal in communication complexity.

Proof: Any algorithm that solves the PPCS problem must examine each precedence constraint at least once. In a distributed or network environment, where the nodes represent tasks and directed edges represent the precedence constraints at least one message is required to examine each edge (i.e., precedence constraint) of G. Clearly, there cannot exist a distributed algorithm to solve the PPCS problem that requires less than e messages. Therefore, $\Omega(e)$ is the lower bound to the communication complexity for this problem. The communication complexity of algorithm D_SCHEDULE is shown to be O(e) in Theorem 4.2. Hence, algorithm D_SCHEDULE is optimal in communication complexity to within a constant factor. □

5 Illustrative example

The algorithm is illustrated with the help of an example. Initially, the duration of the task d(i), the minimum number NUM(i) of dependent tasks to be completed before the task i can be scheduled and the cost C(i, j) for each edge (i, j) $\in$ V (representing a precedence constraint) are available as the input (c.f., figure 1). Also each node knows its neighbor information, i.e., its predecessor and successor sets. During the first phase of the algorithm, SCHEDULE messages are propagating from the terminal nodes (i.e., nodes having no successors) towards the root. The starting time and the completion time for each of the tasks are computed during this phase. Once this phase is complete, the final phase starts by the root by sending TERMINATE messages to each of its successors informing about the job completion time and also that it has terminated its algorithm. When TERMINATE messages are received by all the terminal nodes from their predecessors the entire algorithm terminates. In Figure 2, for each task i, its starting time ST(i) and completion time CT(i) are shown.

6 Some Variations

We now discuss two variations to the standard PPCS problem. The first is simple and we provide a sample implementation. The second variation seems just as easy, but upon further inspection (see section 6.2.1) we show that this is not the case.

6.1 Classical PCS

It may be noted that algorithm D_SCHEDULE can also be adapted to work for finding a solution to the classical PCS problem with the same time and message complexity. In this case, the only modification required to algorithm D_SCHEDULE will be to replace NUM(i) by |SUCC(i)| for every node i $\in$ V. However, to solve the classical PCS problem, the algorithm can be further simplified, since it is not necessary to maintain the list LIST(i) by each node i $\in$ V. Instead, each time node i receives a SCHEDULE message from node j, j $\in$ SUCC(i), node i requires to update

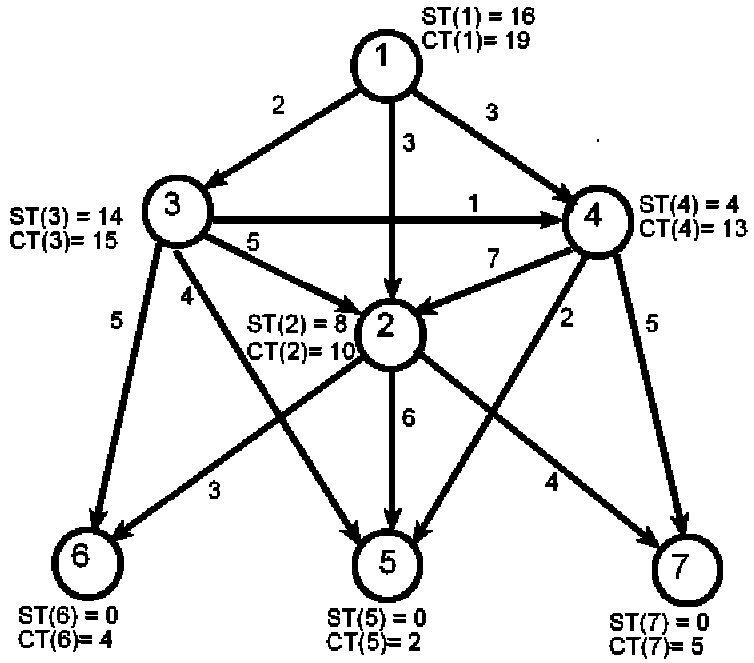

Figure 2: The output of the algorithm on G_1 is shown alongside each node (task). Although not shown above, the job completion time JCT = 19 is available in a distributed manner to each node at the termination of the algorithm.

ST(i) as Max{ST(i), CT(j) + C(i, j)}. There is no need of function insert(x, y) and whenever node i finds count(i) = 0 the current value of ST(i) gives the earliest start time of task i.

6.2 Identifying *critical paths* in PPCS

In the traditional CPS problem each task has an associated earliest start (finish) time EST (EFT) and latest start (finish) time LST (LFT). Here, the critical path is defined to be the ordered set of tasks for which EST = LST, i.e. these are the tasks which cannot be delayed without delaying the overall job completion time. With classical PCS, the LST of a task i is found by taking the minimum possible value of LST(j) – (C(j, i) + d(i)), (j, i) ∈ E. For PPCS however, identifying the LST's and hence the critical tasks is not as straightforward since for each task, not all of its dependant tasks are scheduled. This gives rise to the cases described below.

We denote LST(i) as the actual LST of node i and $LST_j(i)$ as the value of LST(j) – (C(j, i) + d(i)) for some predecessor j, of i (i.e. where (j, i) ∈ E). Also, for each identified case, an example edge (i, j) ∈ E is given relative to Figure 3.

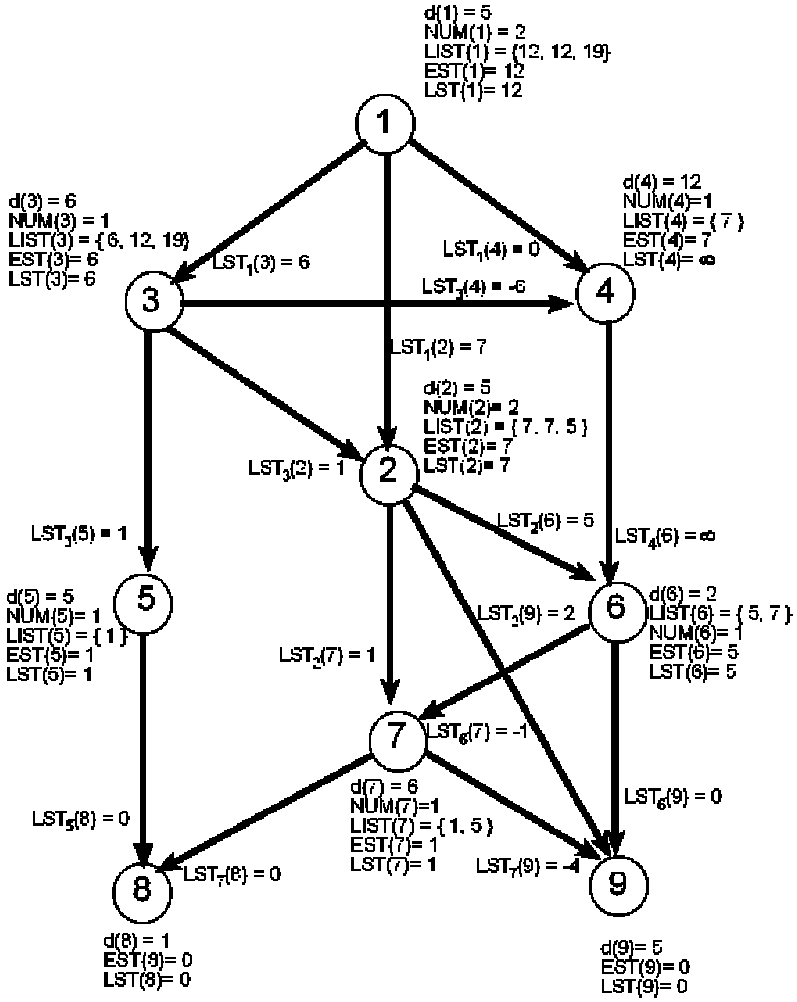

Figure 3: A sample precedence graph G_3 which illustrates the various cases described in section 6.2. For all (i, j) ∈ E, C(i, j) = 0. Note that Task 2 has dependent tasks 6 and 7 which seem to be critical but if either one delays indefinitely, task 2 is not delayed. See section 6.2.1 for more details on this special case.

1. Non-negative LSTj(i) value

(a)LSTj(i) > EST(i): This indicates that task i can delay by LSTj(i) - EST(i) time units without delaying task j beyond LST(j); see edge (2, 9).

(b)LSTj(i) = EST(i): Task i cannot delay without delaying task j beyond LST(j) and is therefore considered as a critical task; see edges (1,3) and (2, 6).

(c)LSTj(i) < EST(i): Task i did not play any role in the scheduling of task j and may delay indefinitely without delaying task j beyond LST(j); see (1, 4) and (3, 2).

(d) LSTj(i) = ∞ : Task i may delay indefinitely without delaying task j since LST(j) = ∞; see (4, 6).

2. Negative LSTj(i) value
This implies that LSTj(i) < EST(i) and therefore degenerates to case 1(c) above; see edges (6, 7) and (7, 9).

The LST(i) for each i ∈ V may be calculated as follows:

Let $\lambda(i) = \{ \; LST_j(i) \mid LST_j(i) \geq EST(i) \; \}$

$$LST(i) = \begin{cases} \min(\ell \in \lambda(i)), & for\ \lambda(i) \neq \{\} \\ \infty, & for\ \lambda(i) = \{\} \end{cases}$$

6.2.1 Special Cases 1(a) and 1(b)

Let $r_k(i)$ denote the k distinct values in LIST(i) for any node i and $m(r_k(i))$ denote the corresponding multiplicity. For instance, in figure 4 let $r_1(i) = 2$, $r_2(i) = 1$ and $r_3(i) = 5$ with multiplicities 2, 2 and 4 respectively. If $m(r_k(i)) > 1$ for some $r_k(i)$, it may be possible for that given node to tolerate a certain number of delays in its successive nodes. In the case of critical nodes (special case 1(b)) this may mean that that node is really "partially-critical" rather than critical. In other words, that node may delay provided a predetermined number of its siblings do not. As an example, let LST(jx) = EST(jx) and NUM(i) = 5 in Figure 4. Then by case 1(*), all the successors of i are definitely-critical (cannot be delayed). However, if say nodes j1 – j4 and j6 all start on time, then j5, j7, j8 may be delayed by some amount of time. Therefore we say that j5 – j8 are partially-critical. Additionally we define nodes with infinite delay as non-critical and all others as semi-critical (this includes special case 1(a)).

We have therefore shown that in PPCS, identification of a definite-critical path is straight forward, but other critical paths may exist which appear to be dynamic in nature as defined by the delay tolerance D_T. This situation does not arise in classical PCS and demonstrates a significant contrast in the two problems.

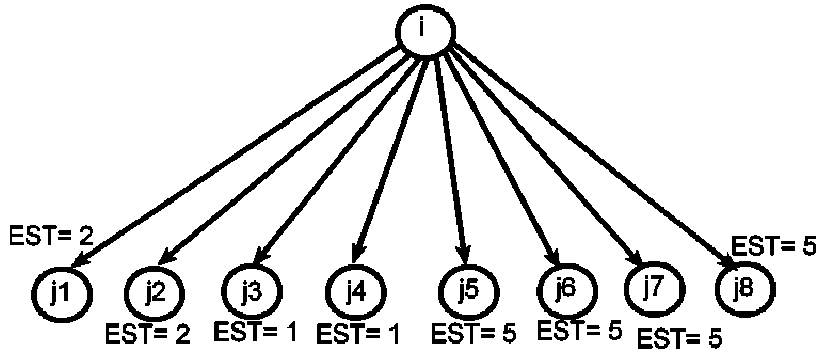

Figure 4: An illustration of the special cases where some tasks may delay further than calculated. For each task jx, x ∈ {1, 2, 3, …,8}, the EST is shown and it is assumed that C(i, jx) = 0. Therefore LIST(i) = { 2, 2, 1, 1, 5, 5, 5, 5}.

NUM(i)	1	2	3	4	5	6	7	8
$D_T(i)$	1	0	1	0	3	2	1	0

Table 1: The Delay tolerance $D_T(i)$ for node i (in Figure 4) is shown for each possible value of NUM(i). Note that $D_T(i) = \Sigma\, m(r_k(i)) - NUM(i)$, where $r_k \leq$ view(LIST(i),NUM(i)).

7 Conclusions

In this paper, we have presented a distributed algorithm for solving the PPCS problem. The algorithm requires O(e) messages and O(n) time. It has been proved that the algorithm is optimal in communication complexity to within a constant factor. Some variations to algorithm D_SCHEDULE which deal with the classical PCS problem and identifying critical paths in PPCS were discussed.

Acknowledgements

The authors wish to thank the anonymous referees for their suggestions and constructive comments on an earlier version of the paper. Their suggestions have greatly improved the readability of the paper. In particular, one of the referees' suggestions have led us to include section 6 in the revised manuscript.

References

[Awerbuch, 85] Awerbuch, B.: "A New Distributed Depth-First Search Algorithm"; Information Processing Letters 20, 3 (1985), 147 - 150.

[Chakrabarti, 99] Chakrabarti, P. P.: "Partial Precedence Constrained Scheduling"; IEEE Transaction on Computers 48, 10 (1999), 1127 - 1130.

[Chaudhuri, 94] Chaudhuri, P.: "An Efficient Distributed Bridge Finding Algorithm"; Information Sciences 81, 1&2 (1994) 73 - 85.

[Chaudhuri, 97] Chaudhuri, P.: "An Optimal Distributed Algorithm for Computing Bridge-Connected Components"; Computer Journal 40, 4 (1997) 200 - 207.

[Chaudhuri, 98] Chaudhuri, P.: "An Optimal Distributed Algorithm for Finding Articulation Points in a Network"; Computer Communications 21, 18 (1998) 1707-1715.

[Chaudhuri, 03] Chaudhuri, P., Thompson, H.: "An Optimal Distributed Algorithm for Partial Precedence Constrained Scheduling"; Proc. DASD'03, SCS, USA (2003), 76 – 82.

[Coffman, 76] Coffman Jr., E. G.: "Computer and Job Shop Scheduling Theory"; John Wiley & Sons, New York, New York (1976).

[Dijkstra, 80] Dijkstra, E. W., Scholten, C. S.: "Termination Detection for Diffusing Computation"; Information Processing Letters II, II (1980) 1 - 4.

[Korach, 84] Korach, E., Rotem, D., Sontoro, N.: "Distributed Algorithms for Finding Centers and Medians in Networks"; ACM Transactions on Programming Languages and Systems 6, 3 (1984) 380 - 401.

[Misra, 82] Misra, J., Chandy, K. M.: "A Distributed Graph Algorithm: Knot Detection"; ACM Transactions on Programming Languages and Systems 4, 4 (1982) 678 - 686.

[Sharma, 89] Sharma, M. B., Iyenger, S. S., Mandyam, N. K.: "An Efficient Distributed Depth-First Search Algorithm"; Information Processing Letters 32, 4 (1989) 183 - 186.

[Smith, 89] Smith, J. D.: "Design and Analysis of Algorithms"; PWS-Kent Publishing Co., Boston (1989).

[Tel, 00] Tel, G.: "Introduction to Distributed Algorithms"; Cambridge University Press 2nd Ed., United Kingdom (2000)