Multimessage Multicasting: Complexity and Approximations*

Teofilo F. Gonzalez
Department of Computer Science
University of California, Santa Barbara, CA, 93106
E-mail: teo@cs.ucsb.edu
Revised: August 19, 1996

Abstract

We consider Multimessage Multicasting over the n processor complete (or fully connected) static network (MM_C). First we present a linear time algorithm that constructs for every degree d problem instance a communication schedule with total communication time at most d^2, where d is the maximum number of messages that each processor may send (receive). Then we present degree d problem instances such that all their communication schedules have total communication time at least d^2. We observe that our lower bound applies when the fan-out (maximum number of processors receiving any given message) is huge, and thus the number of processors is also huge. Since this environment is not likely to arise in the near future, we turn our attention to the study of important subproblems that are likely to arise in practice. We show that when each message has fan-out $k = 1$ the MM_C problem corresponds to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time, and show that for $k \geq 2$ our problem is NP-complete. We present an algorithm to generate a communication schedule with total communication time $2d - 1$ for any degree d problem instance with fan-out $k = 2$. Our main result is an $O(q \cdot d \cdot e)$ time algorithm, where $e \leq nd$ (the input length), with an approximation bound of $qd + k^{\frac{1}{q}}(d - 1)$, for any integer q such that $k > q >= 2$.

Our algorithms are centralized and require all the communication information ahead of time. Applications where all of this information is readily available include iterative algorithms for solving linear equations, and most dynamic programming procedures. The Meiko CS-2 machine and in general computer systems with processors communicating via dynamic permutation networks whose basic switches can act as data replicators (e.g., n by n Benes network with 2 by 2 switches that can also act as data replicators) will also benefit from our results at the expense of doubling the number of communication phases.

Keywords: Approximation Algorithms, Multimessage Multicasting, Dynamic Networks, Parallel Iterative Methods, Communication Schedules.

*A preliminary version of the results presented in this paper appear in the Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems (Irregular'96)

1 Introduction

1.1 The Problem

The Multimessage Multicasting problem over the n processor static network (MM_C) consists of constructing a communication schedule with least total communication time for multicasting (transmitting) any given set of messages. Specifically, there are n processors, $P = \{P_1, P_2, \ldots, P_n\}$, interconnected via a network N. Each processor is executing processes, and these processes are exchanging messages that must be routed through the links of N. Our objective is to determine when each of these messages is to be transmitted so that all the communications can be carried in the least total amount of time. Since the MM_C problem is not a well known problem, this section is lengthy.

Routing in the complete static network (there are bidirectional links between every pair of processors) is the simplest and most flexible when compared to other static networks with restricted structure like rings, mesh, star, binary trees, hypercube, cube connected cycles, shuffle exchange, etc., and dynamic networks, like Omega Networks, Benes Networks, Fat Trees, etc. The minimum total communication time for the MM_C problem is an obvious lower bound for the total communication time of the corresponding problem on any restricted communication network. But, most interesting, multimessage multicasting for dynamic networks that can realize all permutations and replicate data (e.g., n by n Benes network based on 2 by 2 switches that can also act as data replicators) is not too different, in the sense that the number of communication phases for these dynamic networks can be shown to be twice of that in the complete network. This is accomplished by translating each communication phase in the complete network into two communication phases for these dynamic networks. The first phase replicates data and transmits it to other processors, and the second phase distributes the data to the appropriate processors. This well-known approach ([14], [15], and [17]) will be discussed in more detail after defining some important terms. The IBM GF11 machine [1], and the Meiko CS-2 machine use Benes networks for processor interconnection. The two stage translation process can also be used in the Meiko CS-2 computer system, and any multimessage multicasting schedule can be realized by using basic synchronization primitives. This two step translation process can be reduced to one step by increasing the number of network switches about 50% ([14], [15], and [17]). In what follows we concentrate on the MM_C problem because it has a simple structure, and, as we mentioned before, results for this network can be easily translated to other dynamic networks.

Let us formally define our problem. Processor P_i needs to multicast s_i messages, each requiring one time unit to reach any of its destinations. The j^{th} message of processor P_i has to be sent to the set of processors $T_{i,j} \subseteq P - \{P_i\}$. Let r_i be the number of distinct messages that processor P_i should receive. We define the *degree* of a problem instance as $d = \mathbf{max}\{s_i, r_i\}$, i.e., the maximum number of messages that any processor sends or receives. We define the *fan-out* of a problem instance as $k = \mathbf{max}\{\mid T_{i,j} \mid\}$, i.e., the maximum number of different processors that must receive any given message. Consider the following example.

Example 1.1 *There are three processors ($n = 3$). Processors P_1, P_2, and P_3 must transmit 3, 4 and 2 messages, respectively (i.e., $s_1 = 3, s_2 = 4$, and $s_3 = 2$). The destinations of these messages in given in Table 1. For this example $r_1 = 4$, $r_2 = 4$, and $r_3 = 4$.*

Table 1: Destination of messages for Example 1.1

$T_{i,j}$	$j=1$	2	3	4
$i=1$	$\{2\}$	$\{3\}$	$\{2,3\}$	$\emptyset$
2	$\{1\}$	$\{1\}$	$\{3\}$	$\{1,3\}$
3	$\{1,2\}$	$\{2\}$	$\emptyset$	$\emptyset$

It is convenient to represent problem instances by directed multigraphs. Each processor P_i is represented by the vertex labeled i, and there is a directed edge (or branch) from vertex i to vertex j for each message that processor P_i needs to transmit to processor P_j. The $|T_{i,j}|$ directed edges or branches associated with each message are *bundled* together. The problem instance given in Example 1.1 is depicted in Figure 1 as a directed multigraph.

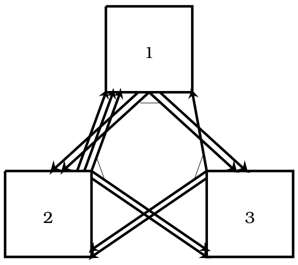

Figure 1: Directed Multigraph Representation for Example 1.1. The thin line joins all the edges (branches) in the same bundle

The communications allowed in our complete network satisfy the following two restrictions.

1.- During each time unit each processor may transmit one message, but such message can be multicasted to a set of processors; and

2.- During each time unit each processor may receive at most one message.

Our communication model allows us to transmit any of the messages in one or more stages. I.e., each set $T_{i,j}$ can be partitioned into subsets, and each of these subsets is transmitted at a different time. Of course, messages can also be sent to all its recipients at the same time. It is intersting to note that if one requires every message to be transmitted to all its recipients at the same time, then there are degree d problem instances such that all their feasible communication schedules have a total communication time that cannot be bounded (above) by any function $f(d)$. This situation arises when every pair of messages need to be transmitted to a common processor, and thus cannot be sent at the same time. In order to generate communication schedules with small total communication time, one needs to allow the partition of the set of destinations of at least some messages and transmit to each subset of processors at a different time.

A *communication mode* C is a collection of subsets of branches from a subset of the bundles that obey the following communications rules imposed by our network:

1.- Branches may emanate from at most one of the bundles in each processor; and

2.- All of the branches end at different processors.

A *communication schedule* S for a problem instance I is a sequence of communication modes such that each branch in each message is in exactly one of the communication modes. The *total communication time* is the latest time at which there is a communication which is equal to the number of communication modes in schedule S, and our problem consists of constructing a communication schedule with least total communication time. From the communication rules we know that a degree d problem instance has at least one processor that requires d time units to send, and/or receive all its messages. Therefore, d is a trivial lower bound for the total communication time. A communication schedule with total communication time equal to four for the problem instance given in Example 1.1 is given in Table 2

Table 2: Communication Schedule for Example 1.1

Time 1	$T_{1,1} : P_1 \rightarrow P_2$	$T_{2,4} : P_2 \rightarrow (P_1, P_3)$	-
Time 2	$T_{1,2} : P_1 \rightarrow P_3$	$T_{2,1} : P_2 \rightarrow P_1$	$T_{3,2} : P_3 \rightarrow P_2$
Time 3	$T_{1,3} : P_1 \rightarrow P_3$	$T_{2,2} : P_2 \rightarrow P_1$	$T_{3,1} : P_3 \rightarrow P_2$
Time 4	$T_{1,3} : P_1 \rightarrow P_2$	$T_{2,3} : P_2 \rightarrow P_3$	$T_{3,1} : P_3 \rightarrow P_1$

To simplify the analysis of our approximation bound we use this simple measure. Another reason is that load balancing (placement) and multimessage multicasting (routing) are normally separate procedures, and load balancing must use a simple objective function in terms of the problem instance it generates that somehow represents the final communication time for the particular placement and some reasonable routing procedure.

Using our multigraph representation one can visualize the MM_C problem as a generalized edge coloring directed multigraph (GECG) problem. This problem consists of coloring the edges with the least number of colors (positive integers) so that the communication rules (now restated in the appropriate format) imposed by our network are satisfied: (1) every pair of edges from different bundles emanating from the same vertex must be colored differently; and (2) all incoming edges to each vertex must be colored differently. The colors correspond to different time periods. In what follows we corrupt our notation by using interchangeably colors and time periods; vertices and processors; and bundles, branches or edges, and messages.

1.2 Previous Work and New Results

In Section 2 we present a linear time algorithm to construct for any degree d problem instance a communication schedule with total communication time at most d^2, and present problem instances for which this upper bound on the communication time is best possible, i.e. the upper bound is also a lower bound. Our lower bound applies when the fan-out is huge, and thus the number of processors is also huge. Since this environment is not likely to arise in the near future, we turn our attention in subsequent sections to important subproblems likely to arise in practice.

The *basic multicasting problem* (BM_C) consists of all the degree $d = 1$ MM_C problem instances, and can be trivially solved by sending all the messages at time zero. There will be no conflicts because $d = 1$, i.e., each processor must send at most one message and receive at most one message. When the processors are connected via a dynamic network whose basic switches allow data replication, the basic multicasting problem can be solved in two stages: the data replication step followed by the data distribution step ([14], [17], [15]). Let us illustrate this two stage process for the example given in Figure 2. A BM_C problem instance is given on the left hand side of Figure 2. We transmit the messages in two stages. In the first stage (data replication) we send message a to processors 2 and 3 (processor 1 has this message initially); message b is sent to processor 5 (processor 4 has this message initially); and message c is sent to processors 7 and 8 (processor 7 has this message initially). Then in the distribution phase, message a in processor 1 is sent to processor 5, and message a in processor 2 is sent to processor 6, message a is already in processor 3, and so on. As we said before, this two stage process can also be used in the MEIKO CS-2 machine.

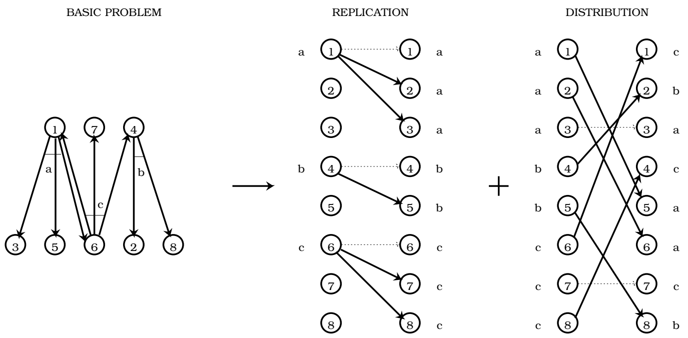

Figure 2: Replication and Distribution.

Let us now consider the case when each message has fixed fan-out k. When $k = 1$ (multimessage unicasting problem MU_C), our problem corresponds to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time (Section 3). In this case, each degree d problem instance has a d color optimal coloration. The interesting point is that each communication mode translates into a single communication step for processors interconnected via permutation networks (e.g., Benes Network, Meiko CS-2, etc.), because in these networks all possible one-to-one communications can be performed in one step.

It is not surprising that several authors have studied the MU_C problem as well as several interesting variations for which NP-completeness has been established, subproblems have been shown to be polynomially solvable, and approximation algorithms and heuristics have been developed. Coffman, Garey, Johnson and LaPaugh [2] studied a version the multimessage unicasting problem when messages have different lengths, each processor can send (receive) $\alpha(P_i) \geq 1$ ($\beta(P_i) \geq 1$) messages simultaneously, and messages are transmitted without interruption (non-preemptive mode). Whitehead [19] considered the case when messages can be sent indirectly. The preemptive version of these problems as well as other generalizations were studied by Choi and Hakimi ([4], [5], [3]), Hajek and Sasaki [12], Gopal, Bongiovanni, Bonuccelli, Tang, and Wong [10]. Some of these papers considered the case when the input and output units are interchangeable, i.e., each processor can be involved in at most $\gamma(P_i)$ message transmissions (sending and/or receiving). Rivera-Vega, Varadarajan and Navathe [16] studied, the file transferring problem, a version the multimessage unicasting problem for the complete network when every vertex can send (receive) as many messages as the number of outgoing (incoming) links. Our MM_C problem is closest to the communication model in the Meiko CS2 machine and it involves multicasting rather than just unicasting.

The MM_C problem is significantly harder than the MU_C. We show that even when $k = 2$ the decision version of the MM_C problem is NP-complete (Section 4). We also present an algorithm to construct a communication schedule with total communication time $2d-1$ for the case when the fan-out is two, i.e., $k = 2$. Our main result is an efficient algorithm to construct for problem instances of degree d a communication schedule with total communication time $qd + k^{\frac{1}{q}}(d-1)$, where q is the maximum number of colors one can use on each bundle and $k > q >= 2$.

1.3 Applications

Multimessage multicasting arises in many applications. Suppose that we have a sparse system of linear equations to be solved via an iterative method (e.g., a Jacobi-like procedure). We are given the vector $X(0)$ and we need to evaluate $X(t)$ for $t = 1, 2, \ldots$, using the iteration $x_i(t+1) = f_i(X(t))$. But since the system is sparse every f_i depends on very few terms. A placement procedure assigns the x_is and $f_i()$s to the processors. Good placement procedures assign a large number of $f_i()$s to the processor where the vector components it requires are being computed, and therefore can be computed locally. However, the remaining $f_i()$s need vector components computed by other processors. So at each iteration these components have to be multicasted (transmitted) to the set of processors that need them. The strategy is to compute $X(1)$, then perform the multimessage multicasting, then compute $X(2)$, and so on. The same communication schedule is used at each iteration, and it can also be used to solve other systems with the same structure, but different coefficients. Our approximation bounds are in terms of the lower bound d. This facilitates the placement procedure since it seeks a placement that induces a multimessage multicasting problem with minimum d. Speedups of n for n processor systems may be achieved when the processing and communication load is balanced, by overlapping the computation and computation time. Another class of applications include most dynamic programming procedures. When all the multicasting information is not known in advance, the schedule that one can generate may be arbitrarily bad. However, even in this environment one can use the work in this paper a posteriori to evaluate the performance of the on-line scheduling heuristics employed relative to the one when all the multicasting information is known ahead of time.

2 Approximation Bounds for the MM_C Problem

We show that for every degree d instance of the MM_C problem one can construct in linear time, with respect to input length, a schedule with total communication time d^2. We then present degree d problem instances such that all their communication schedules have total communication time at least d^2.

Let P be any n processor instance of the MM_C problem of degree d. The set of d^2 colors is defined as $\{(i,j) | 1 \leq i \leq d \text{ and } 1 \leq j \leq d\}$. Now assign an order the incoming edges to each vertex, and assign an order all the bundles emanating from each vertex. Assign color (i,j) to edge $e = \{p, q\}$ if e belongs to the i^{th} bundle emanating form vertex p, and e is the j^{th} incoming edge to vertex q.

Theorem 2.1 *The informal algorithm described above generates a communication schedule* *with total communication time at most d^2 for every degree d instance of the MM_C problem. Furthermore, the algorithm takes linear time with respect to the number of nodes and edges in the multigraph.*

Proof: The proof follows from the observation that edges emanating from the same processor belonging to different bundles are colored with different colors, and all the incoming edges to a node are colored with different colors. Clearly, the algorithm uses at most d^2 colors. The time complexity bound for the algorithm is linear with respect to the input size, because the ordering of the branches and bundles can be arbitrary, and once an ordering is selected, the color assignment is straight forward.

□

For $d = 1$ the total communication time for the communication schedule constructed is one, which is the best possible, but as d increases the total communication time gets farther away from the trivial lower bound d. In the next subsection we show that there are problem instances such that all their communication schedules have total communication time at least d^2.

We now show that there are problem instances such that all their communication schedules have total communication time at least d^2. For all $d \geq 1$ the problem instances I_d defined below have the property that all their communication schedules have total communication time at least d^2. The problem instance I_2 is depicted in Figure 3. For $d \geq 1$ the problem instance, I_d, contains two type of processors: s-processors (sending), and r-processors (receiving). The s-processors (r-processors) send (receive) only messages. The problem instance has n_s s-processors each with d bundles, where

$$n_s = \sum_{i=1}^{d-1} i \cdot \left(\begin{array}{c} d^2 - 1 \\ i \end{array} \right) + 1.$$

For $d = 2$, n_s is 4; for $d = 3$, n_s is 65; and so on.

Between each subset of d bundles from d different s-processors, there is a different r-processor that receives a message from each of these s-processors. Therefore, the total number of r-processors, n_r, is $d^d \left(\begin{array}{c} n_s \\ d \end{array} \right)$. For $d = 2$, n_r is 24; for $d = 3$, n_r is 1179360; and so on. Let us now establish that every communication schedule for every problem instance, I_d, has total communication time at least d^2.

Theorem 2.2 *Every communication schedule for every problem instance, I_d, has total communication time at least d^2.*

Proof: The proof is by contradiction. Suppose not. Suppose that there is a communication schedule S_d for problem instance I_d with total communication time less than d^2. For the communication schedule S_d, let $R_{i,j}$ be the set of time periods where the communications of bundle $T_{i,j}$ take place. We claim that for $1 \leq l \leq d-1$ there are at most l identical sets of time periods $R_{i,j}$ with cardinality l. The proof of this claim is by contradiction. Suppose that there are $l+1$ of such sets. Then either at least two of the corresponding bundles belong to the same s-processor and hence cannot be assigned to the same time periods. Or there are $l+1$ bundles belonging to different s-processors and by the definition of I_d all transmit a message to a common r-processor, but then this r-processor cannot receive $l+1$ different

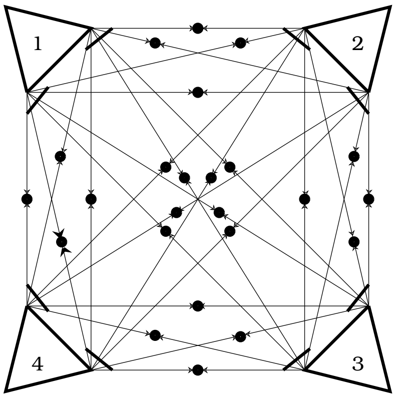

Figure 3: Problem instance I_2. The triangles represent s-processors, and the solid circles represent r-processors

messages from these $l+1$ bundles since all these bundles transmit only during the same l time periods. Therefore, there can be at most

$$\sum_{i=1}^{d-1} i \cdot \left(\begin{array}{c} d^2 - 1 \\ i \end{array} \right)$$

bundles having their $R_{i,j}$ with cardinality at most $d-1$. But since n_s is greater than this number, it then follows that there is at least one s-processor all of whose bundles have $|R_{i,j}| \geq d$. Since all the bundles emanating from a node must have disjoint $R_{i,j}$ sets, it then follows that such s-processor requires d^2 time periods to communicate, which contradicts the assumption that S has total communication time less than d^2. A contradiction. So all the communication schedules for problem instance I_d have total communication time at least d^2.

□

To achieve the bound of d^2 the problem instance I_d has huge fan-out and as a result of this a huge number of processors. Since this environment is not likely to arise in the near future, we turn our attention in subsequent sections to important subproblems likely to arise in practice.

3 Algorithm for the MU_C Problem

Let us now consider the multimessage unicasting, MU_C, problem, i.e., we restrict to the case when the fan-out is equal to 1 (i.e., $k=1$). Remember that for this type of problem instances each message is to be delivered to exactly one processor, but the degree d of a problem can be arbitrary large. The problem of constructing a communication schedule with optimal total communication time can be reduced to the Makespan Openshop Preemptive Scheduling problem. This problem can be solved by the polynomial time algorithm given in [9].

An openshop consists of $m \geq 1$ machines, and $n \geq 1$ jobs. Each job consists of m tasks. The j^{th} task of job i ($T_{i,j}$) must be executed by the j^{th} machine for $t_{i,j} \geq 0$ time units. A schedule is an assignment of each task to its corresponding machine for a total of $t_{i,j}$ time units in such a way that at each time instance one task from each job may be assigned to a machine, and each machine may be assigned at most one task at a time. Note that the task processing need not be continuous, that is why this type of schedules are called *preemptive*. The finish time for schedule S ($f(S)$) is the latest time a task being processed by a machine. The makespan openshop scheduling problem consists of constructing a minimum finish time schedule.

Let m_i be the total time that machine i must be busy, and t_j be the total time that job j needs to be executed. Let $t = \mathbf{max}\{m_i, t_j\}$. Gonzalez and Sahni [9] have shown that there is always a preemptive schedule with finish time t, which is the best possible, and that one such schedule can be constructed in $O(r(min\{r, m^2\} + m\ log\ n))$ time, where r is the number of nonzero tasks. Furthermore, when all the $t_{i,j}$s are integers, there is a schedule where preemptions occur only at integer points, and one such schedule is generated by Gonzalez and Sahni's [9] algorithm.

The MU_C problem of degree one is a special case of the preemptive openshop problem with all the $t_{i,j}$s in $\{0, 1, \ldots, d\}$. Each of the n vertices in the communication graph represent a job, and a machine. The multiset of edges T indicating that processor i must send $|T|$ messages to processor j is now translated to the statement that the j^{th} task of job i must be executed by machine j for $t_{i,j} = |T|$ time units. Translating the results from the openshop problem back to the communication problem, it means that every problem of degree d, has a communication schedule with total communication time equal to d time units. Furthermore, one can easily adapt the algorithm for the minimum finish time openshop problem given in [9] to construct one such communication schedule. The time complexity is $O(r(min\{r, m^2\} + m\ log\ n))$ time, where $r \leq dn$, and $m = n$. For brevity we omit the proof of the following theorem.

Theorem 3.1 *The above informal procedure constructs a communication schedule with total communication time equal to d for any multimessage unicasting problem of degree d with n processors. The procedure takes $O(r(min\{r, m^2\} + m\ log\ n))$ time, where $r \leq dn$, and $m = n$.*

Proof: For brevity the proof is omitted.

□

4 The MM_C Problem with Fan-Out $k = 2$

First we establish that the decision version of the MM_C problem is NP-complete even when $k = 2$. Then we show that there is a communication schedule with total communication time equal to $2d - 1$ for every problem instance of degree d and fan-out 2, and that one such schedule can be generated in $O(nd^{2.5})$ time.

4.1 NP-completeness

In this subsection we show that the decision version of the MM_C problem is NP-complete even when $k = 2$, by reducing the edge coloration (EC) problem to it. The edge coloration problem was shown to be NP-complete in [13].

Edge Coloration Problem:
INPUT: Undirected graph $G = (V, E)$ of degree d, i.e., each vertex has at most d edges incident to it.

QUESTION: Is there an assignment of one of d colors to each edge in G so that no two edges incident to the same vertex are colored identically?

Theorem 4.1 *The decision version of the MM_C problem is NP-complete even when $k = 2$.*

Proof: It is simple to show that the decision version of the MM_C problem is in NP. We now present a polynomial time reduction from the edge graph coloration problem to the MM_C problem with $k = 2$. Given any instance I_{EC} of the edge graph coloration problem, i.e., an undirected graph $G = (V, E)$ of degree d, we construct an instance of the MM_C as follows. For each vertex i in V we create the receive processor (r-processor) i. For each edge j in E there is a send processor (s-processor) j', and an r-processor j''. The s-processor j', that represents edge j in G incident to vertices p and q, has d bundles. The first bundle has two directed edges emanating from it and ending at r-processors p and q. This means that an identical message has to be sent to processor p and q. The remaining $d-1$ bundles each represent one distinct message to be transmitted to r-processor j''. In Figure 4 we give an instance I_{EC} of the edge graph coloration, and the instance I_{MM} of the MM_C problem generated from it by our reduction.

Clearly the reduction takes polynomial time with respect to the number of vertices and edges in the graph G. We now show that the instance I_{MM} of the MM_C problem has a communication schedule with total communication time at most d iff in the the I_{EC} problem instance the edges in G can be colored with d colors. The most important property of the reduction is that if I_{MM} has a communication schedule with total communication time d then the two branches in each bundle emanating from an s-processor must be colored with the same color, and all the branches incoming to each processor must be colored with a different color, as we shall see. These facts ensure that I_{EC} can be colored with d colors.

First we prove that if G can be colored with d colors, then I_{MM} has a communication schedule with total communication time equal to d. Given any d coloration, we color the edges in the instance I_{MM} as follows. If edge j joining vertices p and q is colored with color c, then the two edges in I_{MM} from s-processor j' to r-processor p, and from j' to q are colored with color c, and the remaining $d-1$ edges emanating from j' and ending in j'' are colored with the remaining $d-1$ colors. It is simple to see that this coloration gives rise to a communication schedule with total communication time equal to d for I_{MM}.

We now prove that if I_{MM} has a communication schedule with total communication time equal to d then G can be colored with d colors. It is easy to establish that in any schedule with total communication time equal to d for I_{MM} the message emanating at each s-processor j' and ending at r-processors p and q must be sent at the same time and that all the messages received by each r-processor i must arrive at distinct times. These facts together with the property that each message emanating at each s-processor j' and ending at r-processors p and q represents an edge between vertices p and q in G can be easily combined to establish that G can be colored with d colors. This completes the proof of the theorem.

□

4.2 Approximation Algorithm

Let us now discuss our algorithm to color any instance of the MM_C problem with no more than $2d - 1$ colors. We present algorithm, GM (General Matching), that colors the edges

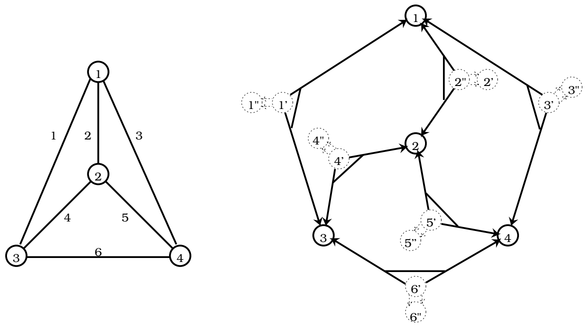

Figure 4: Edge graph coloration instance and corresponding MM_C instance

emanating from each processor at a time using no more than $2d-1$ colors. First we present our algorithm and then we show that it always constructs a valid coloration.

This algorithm colors the bundles emanating from each processor at a time. When considering a processor it colors a maximal set of bundles with one color. The remaining bundles are colored with two colors. This is accomplished by constructing a bipartite graph in which the left-hand side vertices represent the uncolored branches, and the right-hand side vertices represent "available" colors. An edge from vertex x to vertex y indicates that the branch represented by vertex x can be colored y. Then a matching that includes all the left-hand side vertices is constructed. The existence of the matching is established by proving that Hall's conditions hold for the graph. The matching is constructed by Hopcroft and Karp's algorithm [11], and an edge coloration can be easily obtained from the matching.

GM Procedure
 for each processor P_j
  Color all the branches from a maximal set of bundles emanating from P_j with one color;
  Construct the bipartite graph $G = (X + Y, E)$ as follows:
   Each vertex in X represents an uncolored branch, and
     each vertex in Y represents a color (one of the $2d-1$ colors);
   $\{x \in X,\ y \in Y\} \in E$ if "branch" x can be "colored" y;
   Find a matching in G that covers all the vertices in X;
   Construct a schedule with total communication time $2d-1$
     for P_j from the maximal set and the complete matching;
 endfor;
end of GM Procedure

Theorem 4.2 *Given a degree d problem instance of the MM_C with fan-out $k=2$ and n processors (or vertices), Procedure GM constructs a communication schedule with total communication time at most $2d-1$. The time complexity of the procedure is $O(nd^{2.5})$.*

Proof: Let α be the maximal number of bundles emanating from processor P_j colored with one color at the beginning of the iteration. Let us now establish that $\alpha \geq 1$. Let $B_{i,j}$, $1 \leq j \leq 2$, be the set of colors that the j^{th} branch of the i^{th} bundle can be colored without violating Rule 2. Clearly $|B_{i,j}| \geq d$ at the beginning of the P_j loop because the branch is incident to a processor with in-degree d, at most $d-1$ of the other branches incident to it have been colored, and there are $2d-1$ different colors. Since $|B_{i,j}| \geq d$ at the beginning of the P_j loop, at least one bundle can be colored with exactly one color that both of its branches have available, so $\alpha \geq 1$. Since each time a bundle is colored with one color, each set $B_{i,j}$ corresponding to an uncolored branch decreases by at most one. It then follows that just after coloring a maximal number of bundles (α) with one color, for each uncolored branch, $|B_{i,j}| \geq d-\alpha$, and the total number of uncolored bundles is $d-\alpha$. Since no more bundles can be colored with exactly one color, it then follows that for each uncolored bundle, $B_{i,1} \cap B_{i,2} = \emptyset$. Consider the bipartite graph in which each node in the left hand side represents an uncolored branch, and each node in the right hand side is one of the $2d-1$ colors. There is an edge from the node representing the j^{th} uncolored branch of the i^{th} bundle to node q, iff $q \in B_{i,j}$. A matching that includes all the vertices in the left hand side provides us with a coloration. Let us now show that one such matching always exists.

We now claim that Hall's theorem holds for the bipartite graph just constructed and therefore the above matching exists. Hall's condition for this graph is: every subset of uncolored branches Q has the property that $|Q| \leq |\bigcup_{\{i,j\} \in Q} B_{i,j}|$. The reason for this is simple. If the set Q contains the two branches from a bundle, then $|\bigcup_{\{i,j\} \in Q} B_{i,j}| \geq 2(d-\alpha)$ because for each uncolored branch, $|B_{i,j}| \geq d-\alpha$, and for each uncolored bundle, $B_{i,1} \cap B_{i,2} = \emptyset$. Since $|Q| \leq 2(d-\alpha)$, Hall's property follows. On the other hand if the set Q contains at most one branch for each uncolored bundle, then $|Q| \leq d-\alpha$. Since for each uncolored branch, $|B_{i,j}| \geq d-\alpha$, then $|\bigcup_{\{i,j\} \in Q} B_{i,j}| \geq d-\alpha$. Therefore, Hall's conditions follows.

By Hall's theorem, there is an assignment of colors so that all branches can be colored by using at most $2(d-\alpha)$ colors. Add to the previous bound the previous α colors, completes the correctness proof of the theorem.

The for-loop is repeated n times, once for each processor. A maximal set of bundles that can be colored completely with one color can be found in $O(d^2)$ time. The construction of the bipartite graph takes $O(d^2)$ time, and a complete matching in it can be constructed in $O(d^{2.5})$ time [11]. Therefore the overall time complexity for procedure GM is $O(nd^{2.5})$.

□

5 Approximating the MM_C with Fan-Out $k \geq 3$

Let us now consider our simple and very fast approximation algorithm for the MM_C problem. The algorithm colors all edges emanating from $P_1, P_2, \ldots, P_{j-1}$. With respect to this partial recoloration we define the following terms: Each branch emanating from P_j leads to a processor with at most $d-1$ other edges incident to it, some of which have already been colored. These colors are called t_{j-1}-*forbidden* with respect to a given branch emanating from P_j. Just after coloring a subset of branches emanating from processor P_j, we say that a color is s_j-*free* if such color has not yet been used to color any of the branches emanating from processor P_j.

A coloration in which every message is colored with exactly one color may require as many as $d+k(d-1)$ colors. The reason is that each branch has $d-1$ t_{j-1}-forbidden colors, and none of the t_{j-1}-forbidden colors in a branch can be used to color the corresponding bundle. Therefore, there can be $k(d-1)$ t_{j-1}-forbidden colors, none of which can be used to color the bundle. Since there are at most d bundles emanating from a processor P_j, and every bundle is assigned one color, then $d+k(d-1)$ colors are sufficient to color all the bundles emanating from processor P_j, and hence the multigraph.

The above upper bound can be decreased substantially by assigning up to two colors per message (bundle). Again, each branch has $d-1$ t_{j-1}-forbidden colors. But, two colors that are not t_{j-1}-forbidden in the same branch of a bundle can be used to color that bundle. So the question is: What is the largest number of t_{j-1}-forbidden colors in a bundle such that no two of them can be used to color the bundle? For $k=3$ and $d=7$ it is nine. The t_{j-1}-forbidden colors in the three branches are: $\{1,2,4,5,7,8\}$, $\{1,3,4,6,7,9\}$, and $\{2,3,5,6,8,9\}$. Note that no two of the nine colors can color competely the bundle. We have established that the largest number of t_{j-1}-forbidden colors in a bundle such that no two of them can color completely the bundle is $d-1$ for $k=2$, about $1.5(d-1)$ for $k=3$, etc. For brevity we do not include these results.

In what follows we show that it is always possible to color each of the bundles with at most q colors using a total of $qd + k^{\frac{1}{q}}(d-1)$ colors. We also show that the total time complexity for our procedure is $O(q \cdot d \cdot e)$, where $e \leq nd$ is the number of edges in the multigraph. The procedure is given below.

Procedure q**-Coloring** (G, q, k, d)
  for each processor P_j **do**
    for each bundle b emanating from processor P_j **do**
      $l_1 \leftarrow s$-free color that is t_{j-1}-forbidden in the least number of branches of bundle b;
      let n_1 be the number of branches of b where color l_1 is t_{j-1}-forbidden;
      use l_1 to color as many branches of b as possible;
      $r \leftarrow 1$;
      while $r \leq q$ **do** // one can also add the condition $n_r > 0$ //
        $r \leftarrow r+1$
        $l_r \leftarrow s$-free color that is t_{j-1}-forbidden in the least number of branches of
          bundle b together with $l_1, l_2, \ldots, l_{r-1}$;
        let n_r be the number of branches of b where color l_r is t_{j-1}-forbidden
          together with $l_1, l_2, \ldots, l_{r-1}$;
        use l_r to color as many of the uncolored branches of b as possible;
      endwhile
    // As we prove later on, bundle b has been colored at this point//
    // Exiting the loop when $n_r = 0$ will also generate a valid coloration//
    endfor;
  endfor;
end of Procedure q**-Coloring**

Lemma 5.1 *Just before the condition of the while statement is evaluated for the* r^{th} *time for bundle* b*,* $n_r < k^{\frac{q-r}{q}}$ *for* $1 \leq r \leq q$*.*

Proof: The proof is by contradiction. Let b be the first bundle for which the condition of the case does not hold, and let r be the smallest positive integer such that $n_r \geq k^{\frac{q-r}{q}}$ for b. The number of colors used so far to color the bundles emanating from P_j when n_r is calculated by the algorithm for b is at most $q(d-1)+r-1$. Therefore there are at least $q-r+1+k^{\frac{1}{q}}(d-1)$ s-free colors, since the total number of colors is $qd + k^{\frac{1}{q}}(d-1)$. By definition of n_r each of these s-free colors is t_{j-1}-forbidden with $l_1, l_2, \ldots l_{r-1}$ in at least $k^{\frac{q-r}{q}}$ branches emanating out of bundle b. Therefore, the total number of occurrences of t_{j-1}-forbidden colors with colors $l_1, l_2, \ldots l_{r-1}$ is at least $(q-r+1)k^{\frac{q-r}{q}} + k^{\frac{q-r+1}{q}}(d-1)$. Since each branch of bundle b with t_{j-1}-forbidden colors $l_1, l_2, \ldots l_{r-1}$ can have at most $(d-r)$ other t_j-forbidden colors, it then follows that $n_{r-1} \geq k^{\frac{q-r+1}{q}}$, or that there are more than k branches in the bundle. In either case, there is a contradiction. So, $n_r < k^{\frac{q-r}{q}}$ for $1 \leq r \leq q$.

□

Theorem 5.1 *For every instance of the MM_C problem with fan-out $k \geq 3$, the informal algorithm generates in $O(q \cdot d \cdot e)$ time, where e is the number of edges in the multigraph, a schedule with total communication time $qd + k^{\frac{1}{q}}(d-1)$.*

Proof: The previous lemma implies that $n_q < 1$, therefore $l_1, l_2, \ldots l_q$ are not t_{j-1}-forbidden in the same branch of bundle b. Hence, at most q colors are needed to color bundle b. The time complexity bound is straight forward to establish, and it is based on the observations that each branch has at most $d-1$ t_{j-1}-forbidden colors and that the edges emanating out of each bundle have to be considered at most q times because of the for-loop for r.

□

6 Discussion

All of our approximation algorithms generate a coloration that used at most $a_1 \cdot d + a_2$ colors. The value of constant a_1 for the different methods we have developed and for different values for k is given in Table 3. The methods labeled "simple" are for the method in the previous section. The other methods appear in [6] and [7], and allow for a limited form of recoloration [18]. For brevity we do not discuss the other methods in this paper. We should point out that the method in this paper is among the fastest, and asymptotically it provides solutions equivalent to the ones of other methods.

Table 3: Constant a_1 For Different Methods.

Method \ k	3	4	5	7	10	15	20	50	100
Simple (2 colors)	3.73	4.00	4.23	4.65	5.16	5.87	6.47	9.07	12.00
Involved (2c)	3.33	3.50	3.60	4.43	4.60	5.53	6.00	8.56	11.54
With Matching	2.67	3.00	3.50	4.29	4.50	5.47	6.00	8.54	11.53
Better Bound	2.50	3.00	3.50	4.14	4.40	5.40	5.75	8.52	11.52
Simple (3 colors)	–	–	4.00	4.55	4.81	5.27	5.60	6.67	7.62
Involved (3c)	–	3.56	4.00	4.26	4.67	5.00	5.20	6.23	7.24
Simple (4 colors)	–	–	5.50	5.63	5.78	5.97	6.11	6.66	7.16
Simple (5 colors)	–	–	–	6.48	6.58	6.72	6.82	7.19	7.51

References

[1] G. S. Almasi, and A. Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Co., Inc., New York, 1994.

[2] E. G. Coffman, Jr, M. R. Garey, D. S. Johnson, and A. S. LaPaugh, Scheduling File Transfers in Distributed Networks, *SIAM J. on Computing*, 14(3) (1985), pp. 744 – 780.

[3] H.-A. Choi, and S. L. Hakimi, Data Transfers in Networks, *Algorithmica*, Vol. 3, (1988), pp. 223 – 245.

[4] H.-A. Choi, and S. L. Hakimi, Scheduling File Transfers for Trees and Odd Cycles, *SIAM Journal on Computing*, Vol. 16, No. 1, February 1987, pp. 162 – 168.

[5] H.-A. Choi, and S. L. Hakimi, "Data Transfers in Networks with Transceivers," *Networks*, Vol. 17, (1987), pp. 393 – 421.

[6] Gonzalez, T. F., "Multi-Message Multicasting," Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems (Irregular'96), to appear.

[7] Gonzalez, T. F., "Improved Approximation Algorithms for Multimessage Multicasting," UCSB Department of Computer Science, Technical Report TRCS-96-16, July 1996.

[8] Gonzalez, T. F., "Unit Execution Time Shop Problems, " *Mathematics of Operations Research,"* Vol. 7, No. 1, February 1982, pp. 57 - 66.

[9] T. F. Gonzalez, and S. Sahni, Open Shop Scheduling to Minimize Finish Time, *JACM*, Vol. 23, No. 4, October 1976, pp. 665 – 679.

[10] I. S. Gopal, G. Bongiovanni, M. A. Bonuccelli, D. T. Tang, and C. K. Wong, An Optimal Switching Algorithm for Multibean Satellite Systems with Variable Bandwidth Beams, *IEEE Transactions on Communications*, COM-30, 11 (1982) pp. 2475 – 2481.

[11] A J. Hopcroft, and R. M. Karp, An $n^{2.5}$ Algorithm for Maximum Matchings in Bipartite Graphs, *SIAM J. Computing,* (1973), pp. 225 – 231.

[12] B. Hajek, and G. Sasaki, Link Scheduling in Polynomial Time, *IEEE Transactions on Information Theory*, Vol. 34, No. 5, Sept. 1988, pp. 910 – 917.

[13] I. Holyer, The NP-completeness of Edge-Coloring, *SIAM J. Comp.,* 11 (1982), 117 – 129.

[14] T. T. Lee, Non-blocking Copy Networks for Multicast Packet Switching, *IEEE J. Selected Areas of Communication*, Vol. 6, No 9, Dec. 1988, pp. 1455 – 1467.

[15] S. C. Liew, A General Packet Replication Scheme for Multicasting in Interconnection Networks, *Proceedings IEEE INFOCOM '95*, Vol.1 (1995), pp. 394 – 401.

[16] P. I. Rivera-Vega, R, Varadarajan, and S. B. Navathe, "Scheduling File Transfers in Fully Connected Networks," *Networks*, Vol. 22, (1992), pp. 563 – 588.

[17] J. S. Turner, A Practical Version of Lee's Multicast Switch Architecture, *IEEE Transactions on Communications*, Vol. 41, No 8, Aug. 1993, pp. 1166 – 1169.

[18] V. G. Vizing, On an Estimate of the Chromatic Class of a p-graph, *Diskret. Analiz.*, 3 (1964), pp. 25 – 30 (In Russian).

[19] J. Whitehead, The Complexity of File Transfer Scheduling with Forwarding, *SIAM Journal on Computing* Vol. 19, No 2, April 1990, pp. 222 – 245.