JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 55, 215–235 (1998)
ARTICLE NO. PC981500

Complexity and Approximations for Multimessage Multicasting[1]

Teofilo F. Gonzalez

Department of Computer Science, University of California, Santa Barbara, California 93106
E-mail: teo@cs.ucsb.edu

Received June 13, 1997; revised August 31, 1998; accepted August 31, 1998

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 or 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 $k \geqslant 2$ our problem is NP-complete and remains NP-complete even when forwarding is allowed. 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 $e \leqslant nd$ (the input length), with an approximation bound of $qd+k^{1/q}(d-1)$, for any integer q such that $k > q \geqslant 2$.

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 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

[1] Preliminary versions 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), the Proceedings of the 30th Hawaii International Conference on System Science (HICSS-30), and in UCSB CS Technical Report TRCS96-15.

0743-7315/98 $25.00

from our results at the expense of doubling the number of communication phases.

Key Words: approximation algorithms; multimessage multicasting; dynamic networks; parallel iterative methods; communication schedules.

1. INTRODUCTION

1.1. The Problem

The multimessage multicasting problem over the n processor static network (or simply a 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, $P = \{P_1, P_2, ..., P_n\}$, 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. *Forwarding*, which means that messages may be sent through indirect paths even though a single link path exists, allows communication schedules with significantly smaller total communication time. This version of the multicasting problem is referred to as the MMF_C problem, and the 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. In most applications forwarding is allowed, but when security is an issue forwarding must not be permitted. Also, requiring that messages be forwarded may create additional traffic which under certain conditions may congest the communication network.

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 (or simply networks) with restricted structure such as rings, mesh, star, binary trees, hypercube, cube connected cycles and shuffle exchange, and dynamic networks (or multistage interconnection networks), such as Omega networks, Benes networks, and fat trees. 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. Dynamic networks that can realize all permutations (each in one communication phase) and replicate data (e.g., n by n Benes network based on 2 by 2 switches that can also act as data replicators) will be referred to as *pr-dynamic networks*. Multimessage multicasting for pr-dynamic and complete networks is not too different, in the sense that any communication schedule for a complete network can be translated automatically into an equivalent communication schedule for any pr-dynamic network. This is accomplished by translating each communication phase for the complete network into no more than two communication phases for the pr-dynamic networks. The first phase replicates data and transmits it to other processors, and the second phase distributes data to the appropriate processors [15, 16, 19]. 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 by about 50% [15, 16, 19]. In what follows we concentrate on the MM_C problem because it has a simple structure, and, as we mentioned before, results for the fully connected network can be easily translated to any pr-dynamic network.

Let us formally define our problem. Each processor P_i *holds* the set of messages h_i and *needs* to receive the set of messages n_i. We assume that $\bigcup h_i = \bigcup n_i$, and that each message is initially in exactly one set h_i. We define the *degree* of a problem instance as $d = \mathbf{max}\{|h_i|, |n_i|\}$, i.e., the maximum number of messages that any processor sends or receives. We define the *fan-out* of a problem instance as the maximum number of different processors that must receive any given message. Consider the following example.

EXAMPLE 1.1. There are nine processors ($n = 9$). Processors P_1, P_2, and P_3 send messages only, and the remaining six processors receive messages only. Note that in general processors may send and receive messages. The messages each processor holds and needs are given in Table 1. For this example the density d is 3 and the fan-out is 4.

One can visualize 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 multiset of directed edges or branches associated with each message are *bundled* together. The problem instance given in Example 1.1 is depicted in Fig. 1 as a directed multigraph with additional thin lines that identify all edges or branches in each bundle.

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

1. During each time unit each processor P_i may transmit one of the messages it holds (i.e., a message in its hold set h_i at the beginning of the time unit), but such a message can be multicasted to a set of processors. The message will not be deleted from the hold set h_i.

2. During each time unit each processor may receive at most one message. The message that processor P_i receives (if any) is added to its hold set h_i at the end of the time unit.

TABLE 1

Hold and Need Vectors for Example 1.1

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
$\varnothing$	$\varnothing$	$\varnothing$	$\{a, c, e\}$	$\{a, d, f\}$	$\{b, c, e\}$	$\{b, d, f\}$	$\{c, d, e\}$	$\{c, d, f\}$

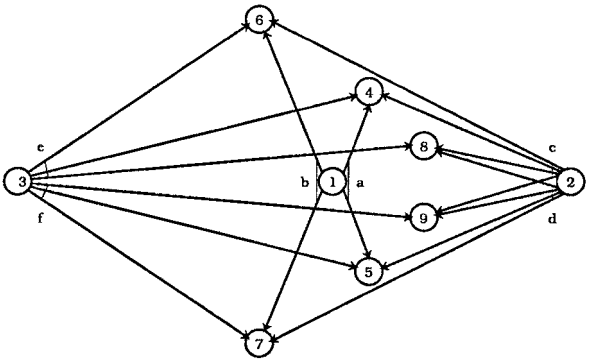

FIG. 1. Directed Multigraph Representation for Example 1.1. The thin line joins all the edges (branches) in the same bundle.

The communication process ends when each processor has $n_i \subseteq h_i$; i.e., every processor holds all the messages it needs. Our communication model allows us to transmit any of the messages in one or more stages. I.e., any message may be transmitted at different times. This added routing flexibility may reduce the total communication time considerably. We now show that it does reduce the total communication time. The problem instance given in Example 1.1 requires six communication steps if one restricts each message to be transmitted only at a single time unit. The reason for this is that no two of the six messages can be transmitted concurrently because every pair of messages either originates at the same processor, or has a common destination processor. However, by allowing messages to be transmitted at different times one can perform all communications in four steps. Let us now explain how this can be accomplished. In step $S1$ processor P_1 sends message a to processor P_5; processor P_2 sends message c to processors P_4, P_6, P_8, and P_9; and P_3 sends message f to processor P_7. In step $S2$ processor P_1 sends message a to processor P_4; processor P_2 sends message d to processors P_5, P_7, P_8 and P_9; and P_3 sends message e to processor P_6. In step $S3$ processor P_1 sends message b to processors P_6 and P_7, and processor P_3 sends message e to processor P_4 and P_8. In step $S4$ processor P_3 sends message f to processors P_5 and P_9. Table 2 shows the hold vectors at the end of each of these four steps.

To establish that forwarding reduces the total communication time Gonzalez [8] showed that when forwarding is not allowed all the communication schedules

TABLE 2

Hold vector after steps $S1$, $S2$, $S3$, and $S4$

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
$S1$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{c\}$	$\{a\}$	$\{c\}$	$\{f\}$	$\{c\}$	$\{c\}$
$S2$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, c\}$	$\{a, d\}$	$\{c, e\}$	$\{d, f\}$	$\{c, d\}$	$\{c, d\}$
$S3$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, c, e\}$	$\{a, d\}$	$\{b, c, e\}$	$\{b, d, f\}$	$\{c, d, e\}$	$\{c, d\}$
$S4$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, c, e\}$	$\{a, d, f\}$	$\{b, c, e\}$	$\{b, d, f\}$	$\{c, d, e\}$	$\{c, d, f\}$

for the problem instance given in Example 1.1 require at least four communication steps, but when forwarding is allowed, all the communications can be performed in three steps. For brevity we do not prove the first claim. But we show that when forwarding is allowed all the communications in the problem instance given in Example 1.1 can be performed in three steps. In step $T1$ processor P_1 sends message a to processors P_4 and P_5; processor P_2 sends message c to processors P_6, P_8, and P_9; and P_3 sends message f to processor P_7. In step $T2$ processor P_1 sends message b to processors P_6 and P_7; processor P_2 sends message d to processors P_5, P_8 and P_9; and P_3 sends message e to processor P_4. In step $T3$ processor P_2 sends message c to processor P_4; processor P_3 sends message f to processors P_5 and P_9; processor P_4 sends message e to processors P_6 and P_8; and processor P_5 sends message d to processor P_7. The last two messages were sent indirectly from their original location. Table 3 shows the hold vector at the end of each of these three steps.

A *communication mode* C is a set of tuples of the form (m, l, D), where l is a processor index ($1 \leqslant l \leqslant n$), and message $m \in h_l$ is to be multicasted from processor P_l to the set of processors with indices D. In addition the set of tuples in a communication mode C must obey the following communications rules imposed by our network:

1. All the indices l in C are distinct; i.e., each processor sends at most one message; and

2. Every pair of D sets in C is disjoint; i.e., every processor receives at most one message.

A *communication schedule* S for a problem instance I is a sequence of communication modes such that after performing all these communications $n_i \subseteq h_i$ for $1 \leqslant i \leqslant n$, i.e., every processor holds all the messages it needs. The *total communication time* is the number of communication modes in schedule S, which is identical to the latest time there is a communication. Our problem consists of constructing a communication schedule with least total communication time. From the communication rules we know that every 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. To simplify the analysis of our approximation algorithm we use this lower bound as the objective function value of an optimal solution. Another reason for using this lower bound is that load and communication balancing (placement) and multimessage multicasting

TABLE 3

Hold vector after steps $T2$, $T2$, and $T3$

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
$T1$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a\}$	$\{a\}$	$\{c\}$	$\{f\}$	$\{c\}$	$\{c\}$
$T2$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, e\}$	$\{a, d\}$	$\{b, c\}$	$\{b, f\}$	$\{c, d\}$	$\{c, d\}$
$T3$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, c, e\}$	$\{a, d, f\}$	$\{b, c, e\}$	$\{b, d, f\}$	$\{c, d, e\}$	$\{c, d, f\}$

(routing) are normally separate procedures, and the load and communication balancing problem must have a simple objective function in terms of the problem instance it generates that somehow represents the total communication time for the placement and a reasonable routing procedure. In other words this allows us to define an optimal placement as one that generates a problem instance with minimum density, i.e., minimum value of d.

Under the 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. Note that for the MMF_C problem this correspondence is not adequate simply because an edge from a node i to a node j may be replaced by an edge from node i to a node l and an edge from node l to node j provided that the first communication takes place before the second one.

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 the 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. Also, the lower bound does not apply to the case when forwarding is allowed, because Gonzalez [8] has recently established that every instance of the MMF_C has a communication schedule with $2d$ total communication time. This procedure uses the results discussed in Section 3 and takes $O(r(\min\{r, n^2\} + n \log n))$ time, where $r \leqslant dn$. This time complexity bound grows faster than that of the approximation algorithm given in Section 5.

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 may send at most one message and receive at most one message. The communication schedule has only one communication mode. When the processors are connected via a prdynamic network a communication mode can be performed in two stages: the data replication step followed by the data distribution step [15, 16, 19]. Let us illustrate this two stage process for the example given in Fig. 2. A BM_C problem instance is given on the left hand side of Fig. 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

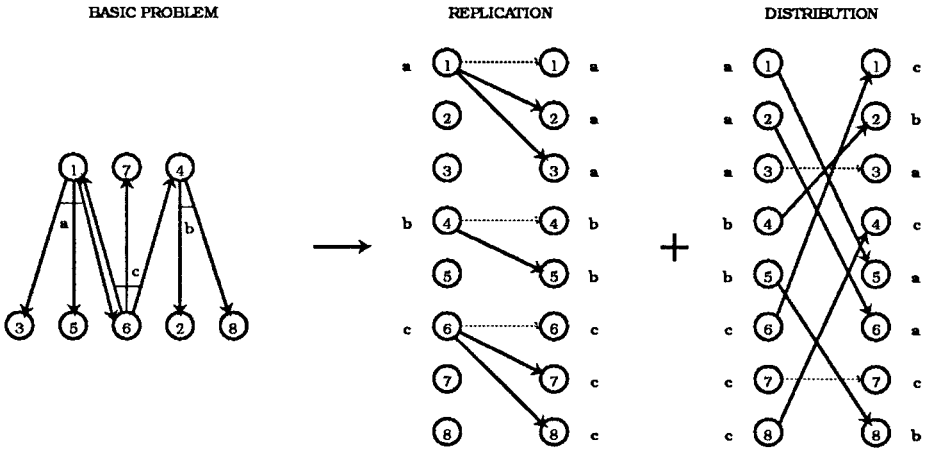

FIG. 2. Replication and Distribution.

message initially), and message c is sent to processors 7 and 8 (processor 6 has this message initially). Then in the distribution phase, message a in processor 1 is sent to processor 5, message a in processor 2 is sent to processor 6, message a is already in processor 3 so there is no need to send it there, and so on. As we said before, this two stage process can also be used in the MEIKO CS-2 machine.

In Section 3 we show that 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 [10]. Each degree d problem instance has an optimal coloring with d colors. 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 communication 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 *et al.* [2] studied a version of the multimessage unicasting problem when messages have different lengths, each processor has $\gamma(P_i)$ ports each of which can be used to send or receive messages, and messages are transmitted without interruption (nonpreemptive mode). Whitehead [21] 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 [3–5], Hajek and Sasaki [13], and Gopal *et al.* [11]. Some of these papers considered the case when the ports are not interchangeable, i.e., it is either an output port or an input port. Rivera-Vega, *et al.* [17] studied the file transferring problem, a version of the multimessage unicasting problem for the complete network when every vertex can send (receive) as many messages as the number of outgoing (incoming) links. All previous work has been limited to unicasting, and all known results about multicasting are limited to single messages,

except for the work by Shen [18], who studied multimessage multicasting for hypercube parallel computers. These algorithms are heuristic and try to minimize the maximum number of hops, amount of traffic, and degree of message multiplexing. Since hypercubes are static networks, there is no direct comparison to our work. The MM_C problem involves multicasting of any number of messages, and its communication model is similar in nature to the one in the Meiko CS-2 machine, after solving basic synchronization problems with barriers.

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). The problem remains NP-complete even when forwarding is allowed (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 $qd+k^{1/q}(d-1)$, where q is the maximum number of colors one can use on each bundle and $k>q\geqslant 2$.

1.3. Applications

Multimessage multicasting problems arise when solving sparse systems of linear equations via iterative methods (e.g., a Jacobi-like procedure), most dynamic programming procedures, etc. Let us now discuss the application involving linear equations. We are given the vector $X(0)$ and we need to evaluate $X(t)$ for $t=1, 2, ...,$ 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 each x_i to a processor where it will be computed at each iteration by evaluating $f_i(\ )$. 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 $X(1)$ and perform the required multimessage multicasting, then compute $X(2)$ and perform the multicasting, and so on. The same communication schedule is used at each iteration and can be computed off-line once the placement of the x_is has been decided. The same communication schedule can also be used to solve other systems with the same structure, but different coefficients. Speedups of n for n processor systems may be achieved when the processing and communication load is balanced, by overlapping the computation and communication time. This may be achieved by executing two concurrent tasks in each processor. One computes the x_is, beginning with the ones that need to be multicasted, and the other deals with the multicasting of the x_i values. If all the transmissions can be carried out by the time the computation of all the x_is is finished then we have achieved maximum performance. But if the communication takes too long compared to the computation, then one must try another placement or try alternate methods.

When all the multicasting information is not known in advance, the message routing must be performed online and the total communication time may be very

large. However, even in this environment one can use the results reported in this paper *a posteriori* to evaluate the performance of the online scheduling heuristics employed relative to the case when all the multicasting information is known ahead of time.

2. UPPER AND LOWER 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 $\{(i, j) \mid 1 \leqslant i \leqslant d \text{ and } 1 \leqslant j \leqslant d\}$. Now assign an order to the incoming edges to each vertex, and assign an order to all the bundles emanating from each vertex. Assign color (i, j) to edge $e = \{p, q\}$ if e belongs to the ith bundle emanating form vertex p and e is the jth 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 straightforward. ▮

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. Before we establish that there are problem instances such that all their communication schedules have total communication time at least d^2, we outline their basic properties. These problem instances have for each subset of d bundles emanating for different processors a distinct processor that must receive the message associated with all of these d bundles. We claim that there cannot be $l+1$ bundles (for any $1 \leqslant l \leqslant d-1$) all of which send their messages during the same l time periods because either two of these bundles originate at the same processor and therefore cannot transmit at the same times, or all the bundles emanate out of different processors and it would be impossible to transmit to the receiving processor(s) they have in common $l+1$ different messages in only l time units. Now by introducing an appropriate number of processors we can guarantee that here is at least one processor with d bundles such that all its bundles must transmit their messages at d different time units. Therefore every schedule for these problem instances must have total communication time at least d^2. A problem instance for $d = 2$ that does not have a schedule with total

communication time less than 4 is given in Fig. 3. Any schedule with total communication time at most three for this problem instance cannot have for each processor a bundle that transmits all its messages in exactly one time period, because there would be two bundles emanating out of different processors transmitting at the same time. But then the receiving processor these two bundles have in common cannot receive two different messages in just one time unit. Therefore, for at least one processor its two bundles must transmit at two different times which establishes that one needs at least four time units to transmit all messages.

We now formally establish that for all $d \geqslant 1$ the problem instance I_d defined below has the property that all its communication schedules have total communication time at least d^2. The problem instance I_2 is depicted in Fig. 3. For $d \geqslant 1$ 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

$$n_s = \sum_{i=1}^{d-1} i \cdot \binom{d^2-1}{i} + 1.$$

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

For each subset of d bundles from d different s-processors there is a unique r-processor that receives a message from each of these s-processors. Therefore, the total number of r-processors, n_r, is $d^{d\binom{n_s}{d}}$. 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 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

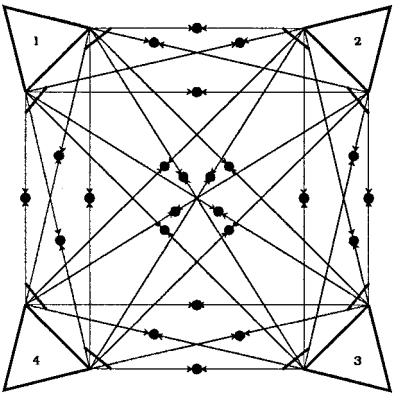

FIG. 3. Problem instance I_2. The triangles represent s-processors, and the solid circles represent r-processors.

communications of bundle $T_{i,j}$ take place. We claim that for $1 \leqslant l \leqslant d-1$ 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 messages from these $l+1$ bundles since all these bundles transmit only during the same l time periods. Therefore, there can be at most

$$\sum_{i=1}^{d-1} i \cdot \binom{d^2-1}{i}$$

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 $|R_{i,j}| \geqslant d$. 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. Coffman, *et al.* [2, p. 746] showed that the restricted MU_C problem in which each processor can send or receive (but not at the same time) a message at a time is an NP-complete problem. However, the MU_C problem can be reduced to the makespan openshop preemptive scheduling problem, which can be solved in polynomial time [10].

An openshop consists of $m \geqslant 1$ machines and $n \geqslant 1$ jobs. Each job consists of m tasks. The jth task of job i ($T_{i,j}$) must be executed by the jth machine for $t_{i,j} \geqslant 0$ 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 is called *preemptive*. The finish time for schedule S ($f(S)$) is the latest time a task is 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 [10] showed 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 [10] 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 $\{0, 1, ..., d\}$. Each of the n vertices in the communication graph represents 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 jth 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 [10] to construct one such communication schedule. The time complexity is $O(r(\min\{r, n^2\} + n \log n))$ time, where $r \leqslant dn$. 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* $O(r(\min\{r, n^2\} + n \log n))$ *time, where r is the total number of messages with distinct origin and destination* ($r \leqslant dn$).

Proof. For brevity the proof is omitted. ▮

The schedule can also be generated by Choi and Hakimi's algorithm [3, p. 230] in $O(r^2 n)$ time. Since $r \geqslant n$, Choi and Hakimi's algorithm [3] is not as time efficient as the one in [10]. However, Choi and Hakimi's algorithm [3] also solves generalizations of the preemptive open shop problem.

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 $k = 2$ and show that the problem remains NP-complete even when forwarding is allowed. 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 coloring (EC) problem to it. The edge coloring problem was shown to be NP-complete in [14].

Edge Coloring 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 graph edge coloring problem to the MM_C problem with $k=2$. Given any instance I_{EC} of the graph edge coloring 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) v_i. For each edge j in E there is a send processor (s-processor) e_j and an r-processor f_j. The s-processor e_j, that represents edge j in G incident to vertices p and q in G, has d bundles. The first bundle has two directed edges emanating from it and ending at r-processors v_p and v_q. This means that an identical message has to be sent to processor v_p and v_q. The remaining $d-1$ bundles each represent one distinct message to be transmitted to r-processor f_j. In Fig. 4 we give an instance I_{EC} of the graph edge coloring problem 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 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

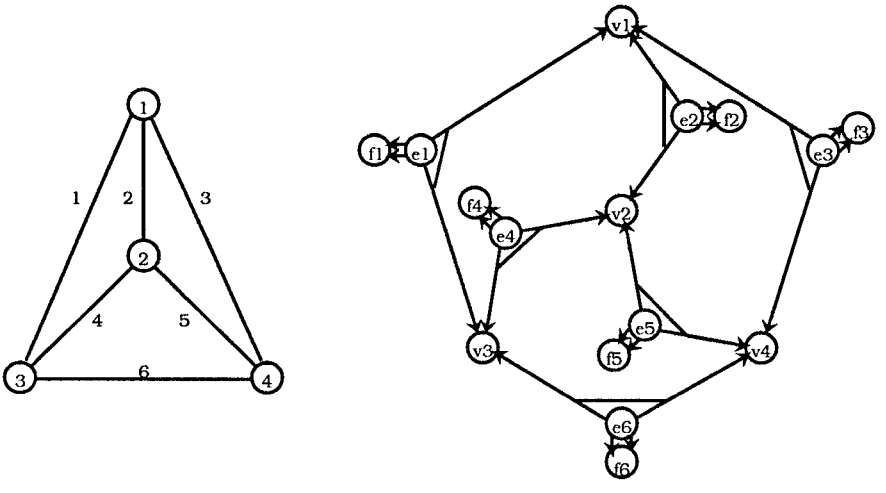

FIG. 4. Graph edge coloring instance and corresponding MM_C instance.

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. These facts will 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 coloring, we color the edges in the instance I_{MM} as follows. If edge j in G joining vertices p and q is colored with color c, then the two edges in I_{MM} from s-processor e_j to r-processor v_p and from e_j to v_q are colored with color c, and the remaining $d-1$ edges emanating from e_j and ending in f_j are colored with the remaining $d-1$ colors. It is simple to see that this coloring 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 e_j and ending at r-processors v_p and v_q must be sent at the same time and that all the messages received by each r-processor v_i must arrive at distinct times. These facts together with the property that each message emanating at each s-processor e_j and ending at r-processors v_p and v_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. ▮

The above reduction cannot be used to show that the MMF_C problem is NP-complete. The reason for this is that the processors f_j and v_i may be used for forwarding in schedules with total communication time equal to d. To show that the MMF_C problem is NP-complete even when $k=2$ we modify the previous reduction by introducing additional vertices and edges in such a way that none of the vertices may be used for forwarding in a communication schedule with total communication time d (see Fig. 5). Let us now discuss the modifications. For each vertex i in V, add the r-processor w_i, and for each edge j in E, add the r-processor g_j. For each edge j in E, add d edges from processor f_j to processor g_j, and for each vertex i in V, add d edges from processor v_i to processor w_i.

A processor is said to be *input saturated* if it has d edges incoming to it, and a processor is said to be *output saturated* if it has d edges emanating from it. We say that a processor is used for *forwarding messages* if at some time it sends a message it did not have at time zero. An input saturated processor that receives d messages that are not needed by other processors cannot be used for forwarding messages in a communication schedule with total communication time d. In our reduction the input saturated processors g_j and w_i cannot be used for forwarding in a communication schedule with total communication time d. An output saturated processor that sends d different messages cannot be used for forwarding messages in a communication schedule with total communication time d. In our reduction output saturated processors e_j, f_j, and v_i cannot be used for forwarding in a communication schedule with total communication time d.

THEOREM 4.2. *The decision version of the MMF_C problem is NP-complete even when $k=2$.*

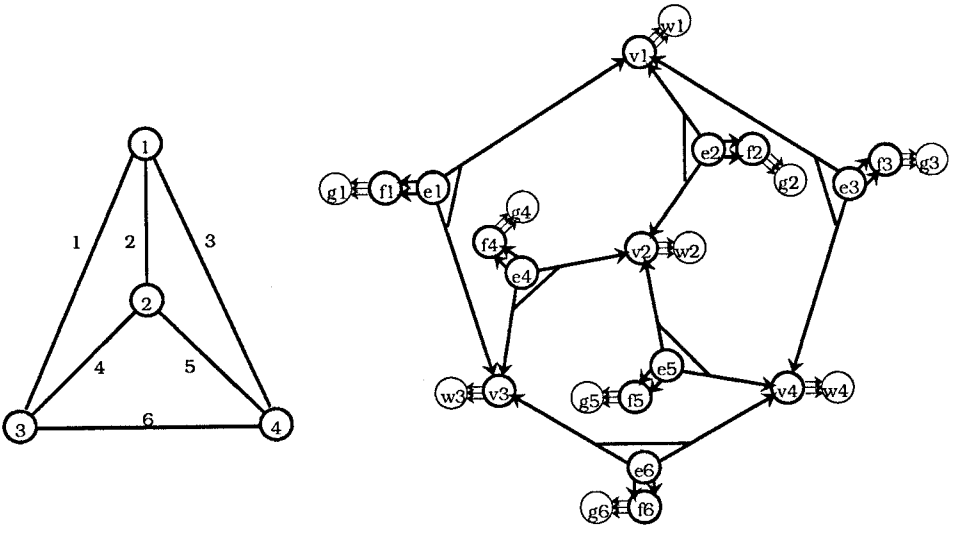

FIG. 5. Graph edge coloring instance and corresponding MMF_C instance.

Proof. The proof is similar to the one for the previous theorem, but uses the arguments given just before this theorem. ▮

5. APPROXIMATING THE MM_C WITH FAN-OUT $k = 2$

Let us now discuss a simple approximation algorithm for the MM_C problem with $k = 2$ but arbitrary degree d. Given any instance P of this problem we break each message with two destinations into two different messages with one destination each. Since $k = 2$ the resulting problem instance is a multimessage unicasting problem of degree $2d$. From the results in Section 3 we know a communication schedule with total communication time equal to $2d$ can be constructed for this problem in $O(r(\min\{r, n^2\} + n \log n))$ time, where $r \leqslant dn$. This communication schedule is also a communication schedule for the instance P of the MM_C problem.

Let us now discuss our algorithm *GM* (general matching) to color any instance of the MM_C problem with no more than $2d - 1$ colors in $O(nd^{2.5})$ time. Algorithm *GM* colors the edges 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 coloring.

Algorithm GM colors the bundles emanating from each processor at a time. When considering a processor it colors a maximal set of bundles with one color per bundle. The remaining bundles are colored with two colors. This is accomplished by constructing a bipartite graph in which the lefthand side vertices represent the uncolored branches and the righthand 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 lefthand 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 [12], and an edge coloring 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 per bundle;
  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);
    Add edge $\{x \in X, y \in Y\}$ to 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 5.1. *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 per bundle at the beginning of the iteration. Let us now establish that $\alpha \geqslant 1$. Let $B_{i,j}$, $1 \leqslant j \leqslant 2$, be the set of colors that the jth branch of the ith bundle can be colored without violating Rule 2. Clearly $|B_{i,j}| \geqslant d$ at the beginning of the P_j loop because every branch is incident to a processor with in-degree d, there are $2d-1$ different colors, and at most $d-1$ of the other branches incident to it have been colored. Since $|B_{i,j}| \geqslant d$ 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 $\alpha \geqslant 1$. 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, $|B_{i,j}| \geqslant d-\alpha$, 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, $B_{i,1} \cap B_{i,2} = \varnothing$. 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 jth uncolored branch of the ith 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 coloring because it identifies for each uncolored branch a color which when assigned to it does not create conflicts. 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 that every subset of uncolored branches Q has the property that $|Q| \leqslant |\bigcup_{\{i,j\} \in Q} B_{i,j}|$. The reason for this is simple. If the set Q contains the two branches from a bundle, then

$|\bigcup_{\{i,j\}\in Q} B_{i,j}| \geqslant 2(d-\alpha)$ because for each uncolored branch, $|B_{i,j}| \geqslant d-\alpha$, and for each uncolored bundle, $B_{i,1} \cap B_{i,2} = \varnothing$. Since $|Q| \leqslant 2(d-\alpha)$, Hall's property follows. On the other hand if the set Q contains at most one branch for each uncolored bundle, then $|Q| \leqslant d-\alpha$. Since for each uncolored branch, $|B_{i,j}| \geqslant d-\alpha$, then $|\bigcup_{\{i,j\}\in Q} B_{i,j}| \geqslant d-\alpha$. 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. Adding to this bound the previous α colors used completes the correctness proof.

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 [12]. Therefore the overall time complexity for procedure *GM* is $O(nd^{2.5})$. ▮

We should point out that when $d \leqslant n$ procedure GM has time complexity bound $O(nd^{2.5})$ that grows slower than that of the previous algorithm $O(d^2n^2)$. When d is very small compared to n there is significant difference in the time complexity bounds for these two procedures and the number of different colors procedure GM introduces is normally less than $2d-1$. The main reason is that one may color a large number of bundles with a single color per bundle. For brevity we cannot elaborate of this further.

6. APPROXIMATING THE MM_C WITH FAN-OUT $k \geqslant 3$

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, ..., P_{j-1} and then colors the bundles emanating out of P_j one bundle at a time. It colors all the edges emanating out of a bundle with $q > 2$ different colors, where q is an input. The coloring of the bundle is a greedy one, it first colors the largest number of edges with one color, then the largest number of uncolored edges with another color, and so on. By setting the total number of colors to an appropriate value, we can show that our procedure always generates a valid solution. Let us now define some terms and formally define our algorithm.

The algorithm colors all edges emanating from P_1, P_2, ..., P_{j-1}. With respect to this partial recoloring we define the following terms: Each branch emanating from P_j leads to a processor with at most $d-1$ other (incoming) 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, i.e., a color is t_{j-1}-forbidden (target forbidden) if it has been used in a branch that ends at the same processor as the branch in question. 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, i.e., a color is s_j-free (source free) if it has not been used in a branch emanating from processor j.

A coloring 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 $qd+k^{1/q}(d-1)$ colors. We also show that the total time complexity for our procedure is $O(q \cdot d \cdot e)$, where $e \leqslant nd$ 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 $r \leqslant q$ **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 $l_1, l_2, ..., l_{r-1}$;
      let n_r be the number of branches of b where color l_r is t_{j-1}-forbidden together with $l_1, l_2, ..., l_{r-1}$;
      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 coloring.//
    endfor;
  endfor;
end of Procedure *q*-Coloring

LEMMA 6.1. *Just before the condition of the while statement is evaluated for the rth time for bundle b, $n_r < k^{(q-r)/q}$ for $1 \leqslant r \leqslant q$.*

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

THEOREM 6.1. *For every instance of the MM_C problem with fan-out $k \geqslant 3$, 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 $qd+k^{1/q}(d-1)$.*

Proof. The previous lemma implies that $n_q < 1$; therefore $l_1, l_2, ..., l_q$ are not t_{j-1}-forbidden in the same branch of bundle b. Hence, at most q colors are needed to color bundle b. It is simple to establish the time complexity bound. The proof 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. ∎

7. DISCUSSION

All of our approximation algorithms for the MM_C problem generate a coloring that use 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 4. 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 recoloring [20]. For brevity we do not discuss the other methods in this paper. We should point out

TABLE 4

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

that the method in this paper is among the fastest, and asymptotically it provides solutions equivalent to the ones of other methods. For the MMF_C problem, Gonzalez [8] developed an algorithm that generates communication schedules with total communication time at most $2d$. The algorithm invokes the procedure given in Section 3 for the multimessage unicasting problem. The approximation algorithm given in Section 5 for the MM_C problem is in general faster than the one given in [8].

REFERENCES

1. G. S. Almasi and A. Gottlieb, "Highly Parallel Computing," The Benjamin–Cummings, 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**, No. 3 (1985), 744–780.
3. H.-A. Choi and S. L. Hakimi, Data transfers in networks, *Algorithmica* **3** (1988), 223–245.
4. H.-A. Choi and S. L. Hakimi, Scheduling file transfers for trees and odd cycles, *SIAM J. Comput.* **16**, No. 1 (1987), 162–168.
5. H.-A. Choi and S. L. Hakimi, Data transfers in networks with transceivers, *Networks* **17** (1987), 393–421.
6. T. F. Gonzalez, Multi-message multicasting, *in* "Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems (Irregular'96)," Lecture Notes in Computer Science, pp. 217–228, Springer-Verlag, Berlin/New York, July 1996.
7. T. F. Gonzalez, Improved approximation algorithms for multimessage multicasting, *in* "Proceedings of the Ninth International Conference on Parallel and Distributed Computing Systems PDCS'96," July 1996, pp. 456–461. [Full paper in UCSB Department of Computer Science, Technical Report TRCS-96-16, July 1996.]
8. T. F. Gonzalez, "Multimessage Multicasting with Forwarding," UCSB Department of Computer Science, Technical Report TRCS-96-24, September 1996.
9. T. F. Gonzalez, Unit execution time shop problems, *Mathematics of Operations Research* **7**, No. 1 (1982), 57–66.
10. T. F. Gonzalez and S. Sahni, Open shop scheduling to minimize finish time, *J. Assoc. Comput. Mach.* **23**, No. 4 (1976), pp. 665–679.
11. 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 Trans. Commun.* **30**, No. 11 (1982), 2475–2481.
12. J. Hopcroft and R. M. Karp, An $n^{2.5}$ algorithm for maximum matchings in bipartite graphs, *SIAM J. Comput.* (1973), 225–231.
13. B. Hajek and G. Sasaki, Link scheduling in polynomial time, *IEEE Trans. Inform. Theory* **34**, No. 5 (1988), 910–917.
14. I. Holyer, The NP-completeness of edge-coloring, *SIAM J. Comput.* **11** (1982), 117–129.
15. T. T. Lee, Non-blocking copy networks for multicast packet switching, *IEEE J. Selected Areas Commun.* **6**, No. 9 (1988), 1455–1467.
16. S. C. Liew, A general packet replication scheme for multicasting in interconnection networks, *Proc. IEEE INFOCOM '95* **1** (1995), 394–401.
17. P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe, Scheduling file transfers in fully connected networks, *Networks* **22** (1992), 563–588.
18. H. Shen, Efficient multiple multicasting in hypercubes, *J. Systems Architecture* **43**, No. 9 (1997).

19. J. S. Turner, A practical version of Lee's multicast switch architecture, *IEEE Trans. Commun.* **41**, No. 8 (1993), 1166–1169.

20. V. G. Vizing, On an estimate of the chromatic class of a *p*-graph, *Diskret. Analiz.* **3** (1964), 25–30 [In Russian]

21. J. Whitehead, The complexity of file transfer scheduling with forwarding, *SIAM J. Comput.* **19**, No. 2 (1990), 222–245.