International Journal of Foundations of Computer Science Vol. 12 No. 6 (2001) 791–808

ON SOLVING MULTIMESSAGE MULTICASTING PROBLEMS

TEOFILO F. GONZALEZ
Department of Computer Science, University California
Santa Barbara, CA 93106, USA

Received 15 October 1999
Revised 10 July 2000
Communicated by Albert Zomaya

ABSTRACT

In this paper we survey algorithms and complexity results for the multimessage multicasting problem and its variations under a unified notation. The main results apply to the multimessage multicasting problem for complete networks and pr-networks (multistage interconnection networks that can realize all permutations in one communication phase and replicate data on each switch). We also discuss algorithms that allow message forwarding and distributed algorithms where each processor only knows local information. Different applications where this problem naturally arises are explored.

Keywords: Multimessage Multicasting, Forwarding, Randomized Algorithms, Fully Connected Networks, Approximation Algorithms.

1. Introduction

Suppose there are n processors, $P = \{P_1, P_2, \ldots, P_n\}$, interconnected via a fully connected network. Each processor is executing processes, and these processes are exchanging messages that must be routed through the network. Each message must be transmitted to a set of processors. These processors alternate between computation and communication in a synchronous way. The Multimessage Multicasting problem, MM_C, consists of finding specific times when messages are to be transmitted so that all the communications can be carried in the least total time. When *forwarding* is allowed messages may be sent through indirect paths even though a single link direct paths exist, and the problems is referred to as the MMF_C problem. Forwarding allows communication schedules with significantly smaller total communication time. In these two versions of the problem all the communication information is known ahead of time and the scheduling is off-line. Then the communication schedule is given to the processors to follow at each step. Further on we discuss applications where this type of situation arises. We also discuss the *distributed* version of the MMF_C, which we refer to as the $DMMF_C$ problem. In this problem each processor initially knows the value of n and d (the maximum number of messages that every processor may send or receive), plus the messages it will be

sending and their destinations. The non-distributed version is simpler because there is a preprocessing phase where all the information is available in one processor and it is used to construct communication schedules that are subsequently distributed to the individual processors. Hereafter we assume that each of the messages to be transmitted is at least n bits long. This assumption allow us to send messages with length at most n, other than the original ones, and just count and report the total number of messages.

Let us formally define our problem. Each processor P_i *holds* the set of messages h_i and for each of its messages $m_{i,j}$ it knows the set of processors $s_{i,j}$ that must receive the message. From this information one can compute for each processor P_i the set of messages it *needs* to receive, n_i. Some of our algorithms do not compute the n_is, but at the end each processor P_i will have all the messages it needs. We define the *degree* of an instance as $d = \mathbf{max}\{| h_i |, | n_i |\}$, i.e., the maximum number of messages that any processor sends or receives. Consider the following example.

bf Example 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* [a]. *The messages each processor holds and needs are given in Table 1. The density d is* 3 *and* $n = 9$.

Table 1. Hold and Need vectors for Example 1.

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

n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
$\emptyset$	$\emptyset$	$\emptyset$	$\{a,c,e\}$	$\{a,d,f\}$	$\{b,c,e\}$	$\{b,d,f\}$	$\{c,d,e\}$	$\{c,d,f\}$

Problem instances may be visualized by directed multigraphs as follows. 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 set of directed edges or branches associated with each message are *bundled* together. The problem instance given in Example 1 is depicted in Figure 1 as a directed multigraph with additional thick lines that identify all edges or branches in each bundle.

The communications allowed in our complete network must satisfy the restrictions given below. We should point out that the last part of rule 2 is only needed for the distributed algorithms, because the off-line algorithms generate conflict-free communications.

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 and for the MM_C it must be in its original hold set), but such message can be multicasted to a set of processors. The message will remain in the hold set h_i.

[a] Note that in general processors may send and receive messages.

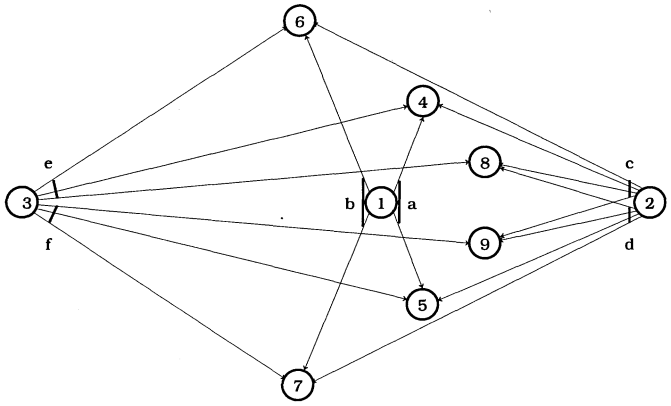

Fig. 1. Directed Multigraph Representation for Example 1.

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. If two or more messages are sent to a processor at a time period, then the messages are garbled and the processor does not receive any of the messages. The sending processor will know at the end of the time period whether or not the message it sent reached all its destinations. When a message does not reach all its destinations the sending processor will not know which processors received the message.

The communication process ends when $n_i \subseteq h_i$ for all i, i.e., each processor holds all the messages it needs. The *total communication time* is the total number of communication steps.

The MM_C problem can also be viewed as a edge coloring problem for the directed multigraph representation, where the edges are colored with the least number of colors in such a way that no two edges incident to the same vertex are assigned the same color and no two edges from different bundles emanating out of the same processor are assigned the same color. The colors correspond to the time periods and the two restrictions on the colorings guarantee that no two messages are allowed to reach the same processor at the same time, and no two (different) messages are transmitted from the same processor at the same time. In what follows we corrupt our notation by using interchangeably colors and time periods; vertices and processors; bundle, branches or edges, and messages. The above correspondence is not appropriate for the case when forwarding is allowed. For these situations multicolorings with additional restrictions can be used to describe some forwarding algorithms [15], but this correspondence becomes more obscure for the general multimessage multicasting problem with forwarding.

Our communication model allows us to transmit any of the messages in one or more stages. I.e., any given message may be transmitted at several different times. This added routing flexibility reduces the total communication time, and in many cases it is a considerable reduction. The problem instance given in Example 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 originate at the same processor, or have a common destination processor. However, by allowing messages to be transmitted at different times there are communication schedules for Example 1 with total communication time four. Table 2 shows the message transmissions for one such schedule which we call S.

Table 2. Message transmissions for schedule S.

Step	Concurrent Communications		
1	a: $P_1 \rightarrow \{P_5\}$	c: $P_2 \rightarrow \{P_4, P_6, P_8, P_9\}$	f: $P_3 \rightarrow \{P_7\}$
2	a: $P_1 \rightarrow \{P_4\}$	d: $P_2 \rightarrow \{P_5, P_7, P_8, P_9\}$	e: $P_3 \rightarrow \{P_6\}$
3	b: $P_1 \rightarrow \{P_6, P_7\}$	e: $P_3 \rightarrow \{P_4, P_8\}$	-
4	f: $P_3 \rightarrow \{P_5, P_9\}$	-	-

Gonzalez [15] showed that when forwarding is not allowed all the communication schedules for the problem instance given in Example 1 require at least four communication steps, but when forwarding is allowed all the communications can be performed in three steps. Table 3 shows the message transmissions for one such schedule which we call T.

Table 3. Message transmissions (with forwarding) for schedule T.

Step	Concurrent Communications		
1	a: $P_1 \rightarrow \{P_4, P_5\}$	c: $P_2 \rightarrow \{P_6, P_8, P_9\}$	f: $P_3 \rightarrow \{P_7\}$
2	b: $P_1 \rightarrow \{P_5, P_7\}$	d: $P_2 \rightarrow \{P_5, P_8, P_9\}$	e: $P_3 \rightarrow \{P_4\}$
3	c: $P_2 \rightarrow \{P_4\}$	f: $P_3 \rightarrow \{P_5, P_9\}$	e: $P_4 \rightarrow \{P_6, P_8\}$
	d: $P_5 \rightarrow \{P_7\}$		

In this paper we survey algorithms and complexity issues for the multimessage multicasting problem and its variations using a unified notation. We discuss results for multimessage multicasting on complete networks and on pr-networks. We explore algorithms that allow message forwarding and distributed algorithms where each processor only knows local information. Different applications where this problem arises naturally are examined.

1.1. Applications

The multimessage multicasting problem arises naturally when solving large sparse systems of linear equations via iterative procedures, and when executing most dynamic programming procedures in a parallel and/or distributed computing environment. Multimessage multicasting also arises when multicasting information over a k channel wireless communication network.

Let us now discuss in more detail how to solve sparse systems of linear equations via iterative methods in a parallel computing environment. Initially 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 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 multicast the elements in $X(0)$, then 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 such schedule can be computed off-line once the placement of the x_is has been decided. 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 finishes, 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.

The above applications justify the off-line Multimessage Multicasting problem. Another interesting variation that arises naturally is the "nearly on-line" or "distributed" Multimessage Multicasting problem. In this case the multicasting destinations are not know until we have executed part of the task and such information is only known locally, i.e., each processor only knows about the information it will be multicasting and has no knowledge about the multicasting operations the other processors are planning. However, all processors know that at a given synchronization point, all processors will be ready to start performing their multicasting operations. For this case we need to compute the communication schedule on-line, and this time must also be taken into account when evaluating performance. In what follows when we refer to the multimessage multicasting problem we mean the the off-line version of the problem, unless we mention on-line or distributed explicitly.

1.2. Multistage Interconnection Networks

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 like rings, mesh, star, binary trees, hypercube, cube connected cycles, shuffle exchange, etc., and dynamic networks (or multistage interconnection 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. 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 ([21], [22], [25]). Figure 2 gives an example of this process.

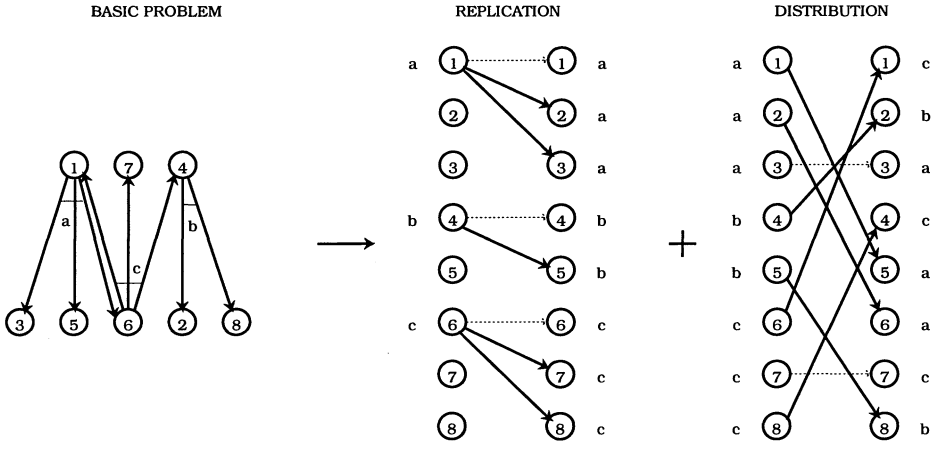

Fig. 2. Replication and Distribution.

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% ([21], [22], [25]). 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. The translation process may double the total communication time, but we can show that for some of our algorithms the translation process does not introduce additional communication steps. This latter situation arises when every multicasting operation has as destinations adjacent processors, or when the set of destinations of any two messages that will be transmitted at the same time are not interleaving, i.e., if one of the messages is to be transmitted to processors P_i and P_j for $i < j$ then the other message cannot be transmitted to any processor P_k such that $i \leq k \leq j$. One can easily see that the latter situation can be easily transformed to the former one. The schedules generated by the forwarding algorithms in Section 5 satisfy the above property. The multimessage multicasting in completely connected networks is also worth while studying because it models a fixed number of processors connected by an optical communication ring.

1.3. Related Work

The case when each message has fixed *fan-out* k (maximum number of processors that may receive any given message) has been studied [11]. For $k = 1$ (multimessage unicasting problem MU_C), Gonzalez showed that the problem corresponds to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time, and each degree d problem instance has a communication schedule with total communication time equal to d [17]. The makespan openshop preemptive scheduling problem is a generalization of the edge coloring of bipartite multigraphs for which the algorithm in [2] generates an optimal coloration. However, Gonzalez and Sahni's [17] algorithm is faster and solves a more general problem.

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 [4] 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 [27] 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 ([6, 5]), Hajek and Sasaki [19], Gopal, Bongiovanni, Bonuccelli, Tang, and Wong [18]. Rivera-Vega, Varadarajan and Navathe [23] 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.

The distributed version of the multimessage unicasting problem with forwarding, $DMUF_C$, has been studied in the context of optical-communication parallel computers [2, 7, 8, 26]. Valiant [26] presented a distributed algorithm with $O(d+\log n)$ total expected communication time. The algorithm is based in part on the algorithm by Anderson and Miller [2]. The communication time is optimal, within a constant factor, when $d = \Omega(\log n)$, and Gereb-Graus and Tsantilas [7] raised the question as to whether a faster algorithm for $d = o(\log n)$ exits. This question was answered in part by Goldberg, Jerrum, Leighton and Rao [8] who show all communication can take place in $O(d + \log\log n)$ communication steps with high probability, i.e., if $d < \log n$ then the failure probability can be made as small as n^{α} for any constant α. Gereb-Graus and Tsantilas [7] presented distributed algorithms without forwarding that guarantees solutions with $\Theta(d + \log n \log\log n)$ expected total communication time.

With the exception of the work reported in [9, 10, 11, 12, 13, 15, 24], research has been limited to unicasting and all known results about multicasting are limited to single messages. Shen [24] has studied multimessage multicasting for hypercube connected processors. His procedures are heuristic and try to minimize the maximum number of hops, amount of traffic, and degree of message multiplexing. The MM_C problem involves multicasting of any number of messages, and its communication model allows the concurrent transmission of a large set of messages.

In this paper we survey recent results for the MM_C, MMF_C and the $DMMF_C$ problems using a unified notation. We discuss results for multimessage multicasting on complete networks and on pr-networks.

2. NP-completeness of the MM_C and MMF_C problems

The decision version of the MM_C problem is NP-complete even when $k = 2$. This can be established by reducing the edge coloring (EC) problem to it. The edge coloring problem was shown to be NP-complete in [20]. The input to the Edge Coloring problem is an undirected graph $G = (V, E)$ of degree d, i.e., each vertex has at most d edges incident to it. The problem is to determine if there is 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.

Rather than giving a formal proof that the decision version of the MM_C problem is NP-complete, we just outline a polynomial time reduction from the 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 (in the graph theory sense), 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 Figure 3 we give an instance I_{EC} of the graph edge coloring, 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. For brevity we do not include the correctness proof, but the main idea is to establish that in any d coloring of the instance MM_C corresponds to a d coloring of the I_{EC} problem, and vice-versa. A formal proof of the following theorem appears in [9].

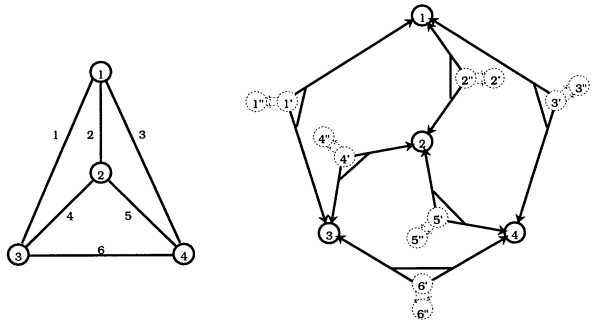

Fig. 3. Graph edge coloring instance and corresponding MM_C instance.

Theorem 1 *The decision version of the MM_C problem is NP-complete even when $k = 2$.*

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. The specific details of the reduction are given in [11].

3. Upper and Lower Bounds for the MM_C

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. There exist 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. Our algorithm colors each edge with a color defined by a first and a last name. We claim that all edges can be colored with d first names and d last names. Therefore, one needs no more than d^2 colors ($\{(i,j)|1 \leq i \leq d \text{ and } 1 \leq j \leq d\}$) to color every degree d problem instance. Our algorithm (arbitrarily) orders all the incoming edges to each vertex, and (arbitrarily) orders all the bundles emanating from each vertex. Edge $e = \{p, q\}$ is assigned the first name (color) i if e belongs to the i^{th} bundle emanating form vertex p, and it is assigned the last name (color) j if e is the j^{th} incoming edge to vertex q. Clearly, no two incoming edges to a node have the same last name, and no two edges emanating from the same processor belonging to a different bundle have the same first name. Therefore our coloring is a valid one and can be generated in linear time with respect to the input length. A formal proof of the following theorem appears in [11].

Theorem 2 *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.*

There are several ways to speed-up communication. One such techniques [3] consists of adding l buffers at the receiving end of each processor and developing controlling hardware so the buffering behaves as follows: (1) if at the beginning of a phase at least one buffer has a message, then one such message (perhaps in a FIFO fashion) is passed to the processor and at the end of the step the buffer will be labeled empty; and (2) if at the end of a communication step there are j empty buffers, then the processor may receive j messages which are stored in the free buffers. For the MM_C problem when each processor has l buffers available for incoming messages we have developed an efficient approximation algorithm that generates communication schedules with total communication time at most $d^2/l - d/l + d$. For brevity we do not elaborate on this results.

The above algorithms are for the completely connected network, but as mentioned in Section 1, the algorithm can also generate schedules for pr-networks. In this case the total communication time will at most double.

Figure 4 gives a problem instance with $d = 2$ that does not have a schedule with total communication time less than 4. Any schedule with total communication time at most three for this problem instance must have for each processor a bundle that transmits all its messages in exactly one time period. But then there are two bundles emanating out of different processors transmitting at the same time to a common processor. A contradiction to our scheduling rules. 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.

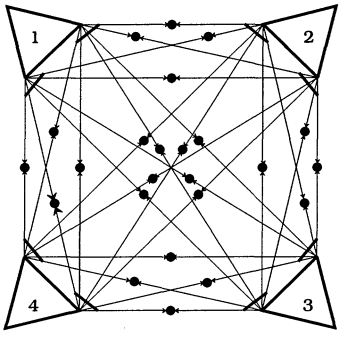

Fig. 4. Problem instance of degree 2 that requires 4 colors. The triangles and and solid circles represent processors.

A generalization of the above problem instance for all $d > 2$ has the property that all its communications schedules have communication time of at least d^2 appear in [11] together with a formal proof of the following theorem.

Theorem 3 *The are problem instances of the MM_C of degree d such that all their communication schedules have total communication time at least d^2.*

For $d = 2$ our problem instances require more than 24 processors; and for $d = 3$, there are more than 1,179,360 processors; and so on. Therefore to achieve the bound of d^2 the problem instances have huge fan-out and as a result of this a huge number of processors. Since this environment is not likely to arise in commercial systems in the near future, we turn our attention in the next section to important subproblems likely to arise in practice.

4. Approximation Algorithms for the MM_C Problem

In this section we begin by describing an approximation algorithm for the case when each message has exactly two destination. Then we present algorithms for the more general case, i.e., messages may be sent to at most k destinations. In the third subsection we discuss improved approximations algorithms for the case when each bundle has to be colored with at most two colors. All these algorithms are for the completely connected network, and as mentioned in Section 1, the algorithms can also generate schedules for pr-networks. In this case the total communication time will at most double.

4.1. Messages with at most Two Destinations

Let us now discuss a simple approximation algorithm for the MM_C problem for $k = 2$ 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 $2d$. Since the multimessage unicasting problem can be solved optimally in polynomial time [11], 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 $r \leq dn$. This communication schedule is also a communication schedule for P.

A faster algorithm that generates slightly better coloring using only $2d-1$ colors is given in [11]. The 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 per bundle. The remaining bundles are colored with two colors. This is accomplished by constructing a bipartite graph and then finding a complete matching in it. The existence of the matching is established by proving that Hall's conditions hold for the graph. The matching can be constructed by a well-known algorithm, and an edge coloring can be easily obtained from the matching. A formal proof of the following theorem appears in [11].

Theorem 4 *Given a degree d problem instance of the MM_C with fan-out $k = 2$ and n processors (or vertices), procedure GM in [11] constructs a communication schedule with total communication time at most $2d - 1$. The time complexity of the procedure is $O(nd^{2.5})$.*

4.2. Multiple Destinations per Message

The algorithms in this subsection color every bundle with at most q colors, where q is an input parameter. In the next subsection we consider improved algorithms for the case when $q = 2$.

The algorithm colors all edges emanating from P_1, $P_2, \ldots, P_{j-1}$. 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.

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. We have also derived asymptotic bounds for this case ($q=2$), but for brevity we do not include this result.

Our simple and very fast approximation algorithm for the MM_C problem colors all edges emanating from P_1, $P_2,\ldots,P_{j-1}$ 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 value. The coloring of bundles is greedy, i.e., 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 feasible solution.

Gonzalez [11] has shown that the algorithm always colors each of the bundles with at most q colors using a total of $qd+k^{\frac{1}{q}}(d-1)$ colors. The time complexity for the procedure is $O(q\cdot d\cdot e)$, where $e\leq nd$ is the number of edges in the multigraph. A formal proof of the following theorem appears in [11].

Theorem 5 *For every instance of the MM_C problem with fan-out $k\geq 3$, the above informal algorithm generates in $O(q\cdot d\cdot e)$ time, where e is the number of edges in the multigraph and $q>2$, a schedule with total communication time $qd+k^{\frac{1}{q}}(d-1)$.*

4.3. Improved Approximation Algorithms

Let us now discuss fast approximation algorithms with an improved approximation bound for problems instances with any arbitrary degree d, but small fan-out. All of our approximation algorithms generate a coloring that uses at most $a_1\cdot d+a_2$ colors. The value of a_1 for the different methods we have developed and for different values for k is given in Table 4. The number inside the parenthesis that follows the method's names indicates the maximum number of different colors one may use to color the branches in each bundle. The lines labeled "Simple" and "Asymptotic" are for the methods described in the previous subsection. The method labeled "Involved (2)" is the fastest and it is discussed in [12]. The method labeled "With Matching" is similar to the one [12], but uses matching [13], and the one labeled "Best Bound" is the best one so far for small values of k [12]. The proofs for these bounds are tedious because the equations and algorithms are more complex, but the proof technique is in general similar to the one for the method given in [12]. By selecting three colors per bundle instead of two colors, one can also improve the asymptotic bound reported in [13] (see Table 4 the line labeled "Improved (3)").

The algorithm is slower, its correctness proof is quite involved, but the proof for the approximation bound is similar in nature to the one in [13]. The benefit when k is 15 is about 10% improvement and about 40% when k is 100. In what follows we discuss the algorithms whose performance is given by the entry labeled "Involved (2)".

Table 4: Value of a_1 For Different Methods.

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

The algorithm behind entry "Involved(2)" is similar in nature to the one in the previous subsection except that when coloring the bundles emanating out of a processor one selects a color with a special property from each bundle and then such color is used to color as many of the branches of the bundle. The remaining uncolored branches are colored with a second color whose existence is guaranteed by setting the total number of colors available to an appropriate value. The results in [12] establish though very elaborate proofs the entry labeled "Involved (2)". The proof involves the manipulation of very long expressions, however several portions of the proofs can be established using symbolic manipulation programs such as Mathematica [14].

5. Message Forwarding Algorithms

Before we discuss our algorithm we define additional terms. One of the disadvantages of the notation given in Section 1 for the MMF_C problem is that after the transmission of the messages in a communication mode several processors may be holding the same message and it becomes difficult to decide which of these processors should be the one to transmit the message at subsequent steps. To minimize the time required to make this decision the algorithm will at all times keep a list of the messages that each processor will be responsible to transmit to other processors at a subsequent time. This information is represented by a directed multigraph (see Figure 1) G that changes after each communication mode.

The idea behind the algorithm is to forward all the messages in d communication steps and transforms the problem to a multimessage unicasting problem of degree d, which can be solved in polynomial time [11]. The forwarding is such that at each time unit each processor sends at most one message and receives at most one message. Our algorithm generates the specific operations to accomplish this based on two labelings and two functions (one indicates the time when the message will

be forwarded, and the other the processor where it will be forwarded). Rather than giving the specific details of the algorithm, we explain how it works by applying it to the problem instance given in Figure 5. The problem instance consists of 12 processors, 11 messages, and has degree $d = 2$.

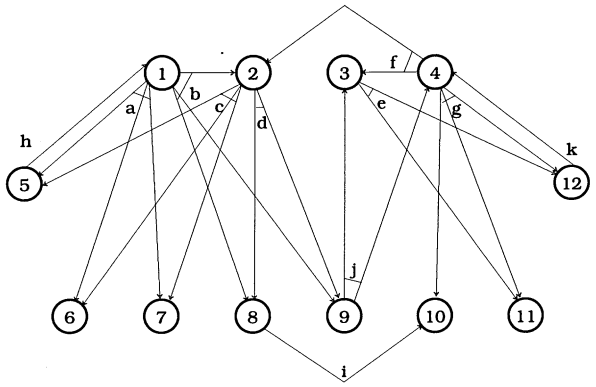

Fig. 5. MM_C problem instance (I, G).

In Figure 6 we show all the processors with a list of labels assigned to the bundles and edges that are defined as follows. The top set of numbers is the bundle number which is defined by labeling the bundles emanating out of processor P_1, then the one emanating out of P_2, and so forth. The next label is the message for the bundle and the third one is the bundle number modulo (d) plus 1. This third number is the time at which the message associated with the bundle will be forwarded. From the way these labels are generated, we know that no two bundles emanating out of a processor will forward a message at the same time. The edges are labeled beginning with the ones emanating out of the first bundle, then the second one, and so forth. These labels are shown in the fourth line. The last set of numbers is the ceil of the edge number divided by d. This last row indicates the processor index where the message will be forwarded. It is simple to see that each processor will receive at most d messages and all these messages will be received at different times.

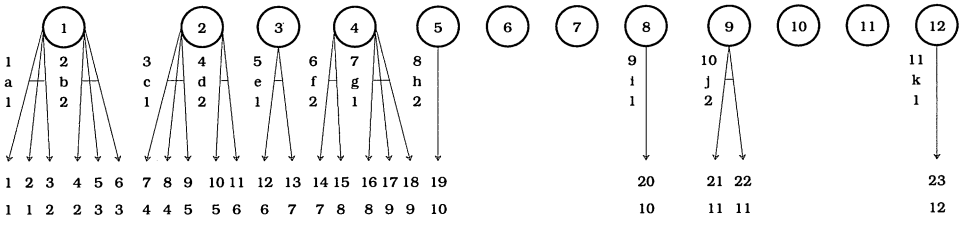

Fig. 6. Labeling performed by procedure FORWARD.

The specific communication operations for time 1 and 2 are given in the forests labeled $T1$ and $T2$ in Figure 7. The resulting unicasting problem $(\hat{I}, \hat{G})$ of degree d is given in Figure 7 (all objects in (c)).

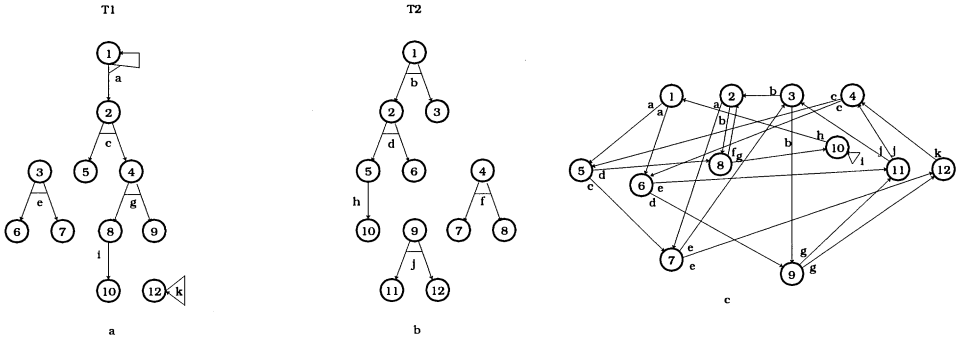

Fig. 7. Communications at time one (a) and time two (b). In (c) the resulting MU_C problem instance.

In [15] we prove the following theorem which is based in some of the above observations.

Theorem 6 *The procedure in [15] constructs for any instance of the MMF_C problem a schedule with total communication time $2d$. The process takes polynomial time and the same schedule can be used by any pr-dynamic network without increasing the total communication time.*

The procedure [10] accomplishes the same result in exactly the same time. The main advantage of the procedure in [10] is that it performs in general fewer communication, but it is a more complex procedure. We should point out that the approximation algorithms in the previous section are faster than the one discussed in this section. However, this algorithm generates communication schedules with significantly smaller total communication time.

For $2 \leq l \leq d$, we define the $l - MMF_C$ as the MMF_C in which each processor has at most ld edges emanating from it. Gonzalez [15] also presents an algorithm to generate a communication schedule with total communication time at most $\lfloor (2 - \frac{1}{l})d \rfloor + 1$ for the $l - MMF_C$ problem. The procedure is similar to the one in this paper. The Multisource MMF_C problem, which is exactly like the MMF_C except that the multicasting data may appear in several processors has also been studied [15]. Gonzalez [15] reduces the Multisource MMF_C problem to the MMF_C problem by solving a set of matching problems in bipartite graphs.

6. Distributed Approximation Algorithm

In this section we discuss the distributed version of the MMF_C problem which we call the $DMMF_C$ problem. In this version of the problem each processor initially knows the value of n and d, plus the messages it will be sending and their destinations.

The strategy to solve this problem is to use the classic parallel prefix algorithm to compute and exchange information, and then use this information to run a distributed version of the forwarding algorithm in the previous section. By forwarding all the messages, the resulting problem becomes a multimessage unicasting problem.

All of the resulting communications can be performed by Valiant's [26] distributed algorithm. Let us discuss this process in more detail and state the main result in [16].

1. **Compute and Broadcast Basic Information.** Each processor P_i needs to know the total number of messages that processors $P_1, P_2, \ldots, P_{i-1}$ need to send as well as the total number of destinations for all of these messages. I.e., the total number of bundles and the total number of edges emanating out of all of these processors. This information is needed to label the bundles and edges emanating out of P_i, and it can be easily computed via the classic parallel prefix in $O(\log n)$ communication steps.

2. **Transform to the Multimessage Unicasting Problem.**

   This operation is performed by making the algorithm given in the previous section distributed. Each processor will run its own version of the procedure.

3. **Solving the resulting Multimessage Unicasting Problem Instance.**

   At this point each processor just runs Valiant's algorithm [26] and all the messages are delivered to their destinations in $O(d + \log n)$ expected communication steps.

Theorem 7 *[16] Our algorithm performs all the multicasting for every instance of the $DMMF_C$ problem with $O(d + \log n)$ expected communication steps.*

7. Discussion

It is simple to see that the $DMMF_C$ problem is more general than the MMF_C and the MM_C problems, but the best communication schedule for the $DMMF_C$ problem has total communication time $\Omega(d+\log n)$ where as for the MMF_C problem is just $2d$, and the MM_C problem is d^2. Therefore, knowning all the communication information ahead of time allows one to construct significantly better communication schedules. Also, forwarding plays a very important role in reducing the total communication time for our scheduling problems.

The most important open problem is to develop distributed algorithms with similar performance guarantees for processors connected via pr-dynamic networks. Algorithms exist for the non-distributed version of this problem [15]. The main difficulty in extending that work to the distributed case is the construction of the routing tables with only local information.

Another very challenging open problem is to develop efficient approximation algorithms for the MMF_C problem that generate schedules with communication time significantly smaller than $2d$. There are several variations of the MM_C problem that are worth studying. For example the case when there are precedence constraints between the messages seems to be one that arises in several applications.

References

1. G. S. Almasi, and A. Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Co., Inc., New York, 1994.
2. R. J. Anderson, and G. L. Miller, Optical Communications for Pointer Based Algorithms, TRCS CRI 88 – 14, USC, Los Angeles, 1988.
3. J. Bruno, Personal Communication, October 1995.
4. E. G. Coffman, Jr, M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling File Transfers in Distributed Networks," *SIAM J. on Comp.*, 14(3) (1985), pp. 744 – 780.
5. H.-A. Choi, and S. L. Hakimi, "Data Transfers in Networks," *Algorithmica*, Vol. 3, (1988), pp. 223 – 245.
6. H.-A. Choi, and S. L. Hakimi, "Data Transfers in Networks with Transceivers," *Networks*, Vol. 17, (1987), pp. 393 – 421.
7. M. Gereb-Graus and T. Tsantilas, "Efficient Optical Communication in Parallel Computers," Proceedings of 4th SPAA, 1992, pp. 41 – 48.
8. L. A. Goldberg, M. Jerrum, T. Leighton, and S. Rao., "Doubly Logarithmic Communication Algorithms for Optical-Communication Parallel Computers," *SIAM J. Comp.*, 26, No. 4, (1997), pp. 1100 – 1119.
9. T. F. Gonzalez, "Multi-Message Multicasting," Proceedings of Irregular'96, Lecture Notes in Computer Science (1117), Springer, (1996), pp. 217-228.
10. T. F. Gonzalez, "Algorithms for Multimessage Multicasting With Forwarding," Proceedings of the 10^{th} PDCS, 372 – 377, 1997.
11. T. F. Gonzalez, "Complexity and Approximations for Multimessage Multicasting," *Journal of Parallel and Distributed Computing,* 55, 215 – 235, 1998.
12. T. F. Gonzalez, "Improved Approximation Algorithms for Multimessage Multicasting," *Nordic Journal on Computing,* Vol. 5, 1998, 196 – 213.
13. T. F. Gonzalez, "Improved Approximation Algorithms for MultiMessage Multicasting," UCSB Department of Computer Science, TRCS-96-16, July 1996.
14. T. F. Gonzalez, "Proofs for Improved Approximation Algorithms for MultiMessage Multicasting," UCSB Department of Computer Science, TRCS-96-17, July 1996.
15. T. F. Gonzalez, "Simple Multimessage Multicasting Approximation Algorithms With Forwarding," *Algorithmica* Vol. 29, 2001, pp. 511 – 533.
16. T. F. Gonzalez "Distributed Multimessage Multicasting," *Journal of Interconnection Networks*, Vol. 1, No. 4, Dec 2000, pp 303 – 315.
17. T. F. Gonzalez, and S. Sahni, "Open Shop Scheduling to Minimize Finish Time," *JACM*, Vol. 23, No. 4, (1976), pp. 665 – 679.
18. 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.
19. B. Hajek, and G. Sasaki, "Link Scheduling in Polynomial Time," *IEEE Transactions on Information Theory*, Vol. 34, No. 5, (1988), pp. 910 – 917.
20. I. Holyer, "The NP-completeness of Edge-Coloring," *SIAM J. Comp.*, 11 (1982), 117 – 129.
21. T. T. Lee, "Non-blocking Copy Networks for Multicast Packet Switching," *IEEE J. Selected Areas of Communication*, Vol. 6, No 9, (1988), pp. 1455 – 1467.

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

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

24. H. Shen, "Efficient Multiple Multicasting in Hypercubes," *Journal of Systems Architecture,* Vol. 43, No. 9, Aug. 1997.

25. J. S. Turner, "A Practical Version of Łee's Multicast Switch Architecture," *IEEE Transactions on Communications*, Vol. 41, No 8, (1993), pp. 1166 – 1169.

26. L. G Valiant, "General Purpose Parallel Architectures," Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, New York, 1990, Chapter 18 (see p 967).

27. J. Whitehead, "The Complexity of File Transfer Scheduling with Forwarding," *SIAM Journal on Computing* Vol. 19, No 2, (1990), pp. 222 – 245.