March 1988

Synthesizing Minimum Total Expansion Topologies for Reconfigurable Interconnection Networks

David Smitley
Supercomputing Research Center

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-19.

Synthesizing Minimum Total Expansion Topologies for Reconfigurable Interconnection Networks

Abstract
The Performance of a parallel algorithm depends in part on how the interconnection topology of the target parallel system matches the communication patterns of the algorithm. We describe how to generate a topology for a network that can be configured into and *r*-regular topology. The topology generated has small *total expansion* with respect to a given task graph. The *expansion* of an edge in a task graph is the length of the shortest path that the edge maps to in the processor graph. The algorithm used to generate the topologies is analyzed and its average case behavior is determined. In addition, this synthesis method is compared to the conventional approach of mapping a task graph onto a fixed processor topology.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-19.

SYNTHESIZING MINIMUM TOTAL EXPANSION TOPOLOGIES FOR RECONFIGURABLE INTERCONNECTION NETWORK

David Smitley
Insup Lee

MS-CIS-88-19
GRASP LAB 137

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

March 1988

Acknowledgements: This work was supported in part by NSF IRI84-10413-AO2, MCS-8219196-CER, NSF/DCR grants 8501482, 8410771, U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027 and Airforce F-49620-85-K-0018.

Synthesizing Minimum Total Expansion Topologies for Reconfigurable Interconnection Networks*

David Smitley†

Insup Lee

General Robotics and Active Sensory Processing Group
Department of Computer and Information Science
Moore School of Electrical Engineering
University of Pennsylvania
Philadelphia, PA 19104-3897

*This work was supported in part by NSF DCR 8501482, NSF MCS-8219196-CER, ARO DAA6-29-84-k-0061, NSF/DCR 8410771 and Airforce F49620-85-K-0018.

†Current Address: Supercomputing Research Center, 4380 Forbes Blvd., Lanham, MD 20706

Running head:

Synthesis of MTE Topologies

Proofs to:

Insup Lee
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-3897

ABSTRACT

The performance of a parallel algorithm depends in part on how well the interconnection topology of the target parallel system matches the communication patterns of the algorithm. We describe how to generate a topology for a network that can be configured into any r-regular topology. The topology generated has small *total expansion* with respect to a given task graph. The *expansion* of an edge in a task graph is the length of the shortest path that the edge maps to in the processor graph. The algorithm used to generate the topologies is analyzed and its average case behavior is determined. In addition, this synthesis method is compared to the conventional approach of mapping a task graph onto a fixed processor topology.

Symbols

Σ

$\in$

TE_{lb}

TFD_{lb}

$\cup$

$\log_r$

$\notin$

$\overline{AFD}$

$\overline{TFD}$

E_m

d_G

d_P

l_P

1. Introduction

The performance of a parallel algorithm depends on how well its communication characteristics match the interconnection topology of the parallel system. Since different algorithms exhibit different communication patterns, it is important to build a parallel system with an interconnection network that is suitable for various communication requirements. One common approach is to connect processors using a reconfigurable network. An interconnection network is called *reconfigurable* if its topology can be altered between different algorithm executions or even between different phases of the same algorithm execution. Several architectures using reconfigurable networks have been proposed or built, which include the MPP [1], CHiP [2], and polymorphic [3] computers. These architectures directly support a variety of communication topologies and thus provide faster communication for a wider variety of parallel algorithms than fixed interconnected architectures such as a hypercube [4]. Each of these architectures, however, have limited reconfiguration ability that restricts the class of algorithms that they can effectively execute.

The class of effectively executed algorithms can be increased by connecting the processors with a r-reconfigurable network. A reconfigurable network is called *r-reconfigurable* if its topology can be configured into an *arbitrary* r-regular topology [5]. That is, each processor has r communication channels, and each channel can be connected to another processor's channel before the execution of the current parallel algorithm. We assume that the setup times of communication links are slow compare the execution speeds of processors. Thus, once the network has been configured, it remains in that configuration throughout the execution of the parallel algorithm. The network can, however, be reconfigured between different parallel algorithm executions or different phases of the same algorithm execution.

Given such a r-reconfigurable network, the problem is how to utilize it effectively; that is, how to find a network topology that *matches* the communication requirements of a parallel algorithm. We note that there is some debate as to what it means for a network topology to match the communication requirements of an algorithm [6]. In this paper, a parallel algorithm is represented by a task graph. A parallel system with its interconnection network configured to a particular topology is also represented by a processor graph. This paper investigates the synthesis problem of finding a processor graph for a give task graph so that the total expansion of the given task graph with respect to the processor graph is minimized. The expansion of an edge in a task graph is the length of the shortest path that the edge maps to in the processor graph. The expansion of an edge is what Rosenberg [7] terms dilation. The *total expansion* of a task graph with respect to a processor graph is the sum of expansions for each edge in the task graph. Minimizing the total expansion helps reduce the average delay a message encounters while traversing the network. To simplify the problem, we assume that the amount of communication between tasks is uniform and the

processors of the parallel system are homogeneous.

Another approach to solve the minimum total expansion problem is to fix the interconnection topology of the parallel system to have a short path between every two processors. Then, the corresponding mapping problem is to assign the nodes of a task graph to the nodes of a fixed processor graph so as to minimize total expansion [7]. The mapping problem to minimize total expansion is NP-complete [7]. Furthermore, even if an optimum mapping from task to processor graph is used, the total expansion might be large [8]. In this paper, we also compare the performance of the synthesis approach to that of the mapping approach.

The paper is organized into four major sections. The next section defines the minimum total expansion problems and gives complexity and lower bound results. Section 3 presents an algorithm that finds sub-optimum solutions to the synthesis problem. Section 3 also describes a set of simulation experiments and gives an analysis of the results. Section 4 then compares the synthesis approach to the mapping approach. Section 5 describes several different r-reconfigurable networks and Section 6 discusses other criteria that can be considered when synthesizing topologies.

2. Minimizing Total Expansion

Let $G=\{V(G),E(G)\}$ be a graph with node set $V(G)$ and edge set $E(G)$. The *degree* of a node v, denoted $d_G(v)$, is the number edges in $E(G)$ that are incident with v. A graph G is called *r-bounded* and *r-regular* if, for every node $v\in V(G)$, $d_G(v)\leq r$ and $d_G(v)=r$ respectively. For two nodes $u,v\in G$, the length of a shortest path between them is denoted $l_G(u,v)$. A graph is *connected* if every pair of nodes are joined by a path. For a connected graph G, the total forwarding distance of G, $TFD(G)$, is defined as

$$TFD(G)=\sum_{u\in V(G)}\sum_{v\in V(G)} l_G(u,v).$$

Note that each pair of nodes is counted twice to simplify the discussion in the rest of the paper. The average forwarding distance of a n-node graph G, $AFD(G)$, is $TFD(G)$ divided by $n(n-1)$.

Throughout the paper, G denotes a task graph that represents a parallel algorithm to be executed on the parallel system. The nodes in the task graph correspond to individual tasks and an edge between two nodes signifies that communication occurs between these two tasks. The parallel system is also represented as a graph with nodes corresponding to processors and edges to communication links. We say that the network of a parallel system with n processors is r-reconfigurable if the processors can be connected in an arbitrary r-regular graph definable on n nodes.

Given a task graph and a parallel system with r-reconfigurable network, we describe how to find a

topology of the network and a mapping from tasks to processors so that the sum of expansions of the edges in the task graph is minimized. We assume that $r \geq 2$ to ensure that a connected processor graph can be constructed. Since we assume that the parallel system consists of homogeneous processors, the mapping part of the synthesis problem is trivial and therefore an arbitrary mapping is used. We also assume that the number of nodes in a task graph equals the number of processors in a parallel system. This problem is called the minimum total expansion (MTE) synthesis problem which can be stated as follows:

Given

A connected graph $G=\{V(G),E(G)\}$ with n nodes;

A set of n nodes $V(P)$;

A bijective function $g: V(G) \rightarrow V(P)$; and

An integer $r \geq 2$

Find

A set of edges $E(P)$, where $P = \{V(P),E(P)\}$ is a r-regular connected graph, such that

$$TE(G,P) = 2 \sum_{(u,v)\in E(G)} l_P(g(u),g(v))$$

is minimized. $l_P(g(u),g(v))$ is the expansion of an edge (u,v) in G with g mapping G to P. Note that the expansion of an edge is counted twice to be consistent with the definition of TFD.

An edge $(u,v)\in E(G)$ is called *matched* if $(g(u),g(v))\in E(P)$. We note that if G is a complete graph, then $TE(G,P) = TFD(P)$ for any processor graph P and thus the MTE synthesis problem for complete graphs is that of finding an optimal mean distance graph defined by Cerf *et al* [9].

As an example of the MTE synthesis problem, consider three graphs G, P and P' in Figure 1. Graphs P and P' are 3-regular graphs. Graph P' has a total expansion of 32 with respect to G, whereas P has a total expansion of 30 which can be shown to be optimal. Therefore, an algorithm for solving the MTE synthesis problem should either construct P or find another 3-regular graph with total expansion 30 given graph G as input.

In contrast to the synthesis problem of finding edges for a processor graph, the mapping problem is to assign the nodes of a task graph to the nodes of a processor graph. Given a task graph and a processor graph, the problem of finding a mapping that minimizes total expansion is called the MTE mapping problem. Briefly stated, the MTE mapping problem is defined as follows: Given two connected graphs G and P, find a bijective function $g: V(G) \rightarrow V(P)$ such that $TE(G,P)$ is minimized.

To illustrate the difference between the synthesis and mapping approaches, consider two graphs in Figure 2. The left hand graph represents the communication structure of a parallel algorithm where each task exchanges information with its two nearest left and right neighbors. A parallel algorithm performing a one dimensional convolution would have such a communication structure. The right hand graph represents the communication topology of a processor system interconnected in a four nearest neighbor toroid. The nodes in this graph are labeled according to a mapping from the task to processor graph. For example, node 3 in the task graph is mapped to the processor in row 2, column 1. This particular mapping has a total expansion of 36. If the processors were connected with a 4-reconfigurable network, the network can be configured to exactly match the task graph since no node in the task graph has more than four incident edges. Here, the total expansion is 30 since each edge in the task graph has expansion one.

2.1. Complexity of the MTE Synthesis Problem

If r is restricted to 2, any solution to the MTE synthesis problem is a ring graph. Hence, an optimum solution to the MTE synthesis problem for a graph G with r restricted to 2 is equivalent to a bijective mapping f from G onto a ring with the property that

$$\sum_{(u,v)\in E(G)} \min(|f(u)-f(v)|, |V(G)|-|f(u)-f(v)|) \tag{1}$$

is minimized. Finding a function f that minimizes Eq. (1) is a variant of the NP-complete Optimal Linear Arrangement problem [10] and is also NP-complete [5]. Thus, the MTE synthesis problem is NP-complete for fixed r=2.

2.2. Lower Bound on Total Expansion

A lower bound on total expansion can be calculated by assuming that, for each node u, the nodes adjacent to u in G are at minimal distances from u in P. For instance, if a node in G has degree 11 and P is 3-regular, then the lower bound is realized if three of the edges have expansion 1, six have expansion 2, and the remaining two edges have expansion 3. If this situation holds for every node in the graph, then the total expansion is equal to the lower bound. Thus, for each $v\in V(G)$, if we let

$$k(v) \text{ be the least integer such that } r\sum_{i=0}^{k(v)}(r-1)^i \geq d_G(v)$$

and let

$$R(v) = d_G(v) - r\sum_{i=1}^{k(v)-1}(r-1)^{(i-1)}$$

then the total expansion of graph G with respect to P is greater than or equal to

$$TE_{lb}(G) = \sum_{v \in V(G)} expansion(v) \tag{2}$$

where

$$expansion(v) = r \sum_{i=1}^{k(v)-1} i\,(r-1)^{(i-1)} + k(v)\,R(v).$$

Similarly, a lower bound for $TFD(P)$ can be defined [9]. This is done by defining $k(v)$ with respect to the number of nodes in the graph instead of the degree of v. Thus, the lower bound on total forwarding distance from all the nodes to all other nodes in the graph, $TFD_{lb}(P)$, can be expressed as

$$TFD_{lb}(P) = n\left[r \sum_{i=1}^{k-1} i\,(r-1)^{(i-1)} + k\,R \right] \tag{3}$$

where

$$k \text{ is the least integer such that } r \sum_{i=0}^{k} (r-1)^i \geq n-1$$

and

$$R = (n-1) - r \sum_{i=1}^{k-1} (r-1)^{(i-1)}.$$

3. A Heuristic Algorithm for the MTE synthesis Problem

In a previous paper [6], we investigated how to generate a processor graph that maximizes the number of communicating tasks that fall on pairs of directly connected processors. Such a processor graph maximizes the number of edges of expansion one. This problem is called the *cardinality* problem. We showed that the cardinality problem is NP-complete. In addition, we presented a heuristic algorithm to solve the cardinality problem. The algorithm first finds a degree bounded subgraph of the task graph such that the subgraph has a maximum number of edges over all degree bounded subgraphs. The degree bounded subgraph is found using a graph factoring algorithm [11] with complexity $O(\sqrt{nr}\,|E(G)|)$. The subgraph, however, is not necessarily connected and therefore we use an $O(n\,|E(G)|)$ heuristic to connect the subgraph while insuring that no node has degree greater than r. We showed that the average case performance of the algorithm was close to optimum.

We now present a heuristic algorithm for the solution of the MTE synthesis problem. The algorithm is based on a hill climbing technique and is outlined in Figure 3. The algorithm starts on a connected graph generated by the cardinality synthesis algorithm mentioned above. The algorithm then searches for edges in G that have maximum expansion. It chooses randomly one of these edges, call it (x,y), and adds it to P. Since P is regular, an edge incident with x and another edge incident with y must be removed from $E(P)$. These edges are also randomly chosen. This process of adding and deleting edges is called a *reduction* step.

The starting topology is saved as the best topology synthesized so far. After each reduction step, if the resultant topology has a TE less than the one stored in **best_graph**, it becomes the best topology. Otherwise, a variable **poorer** is incremented. This hill climbing process is repeated on the topology stored in **best_graph** until **poorer** reaches a predetermined limit of **no_of_tries**. It may appear that after the first time that the algorithm fails to decrease the TE, subsequent reduction steps would be futile. This, however, is not the case since both the added and deleted edges are chosen randomly. Therefore, while one random reduction step may not decrease the TE any more, another might. Hence, further reduction steps are tried up to a limit defined by **no_of_tries**.

The time complexity of the algorithm is bounded by the calculation of $l_P(g(u),g(v))$, that is, the shortest path between nodes $g(u)$ and $g(v)$ in P for every edge (u,v) in G. Using a breadth first search [12] rooted at each node in P, this can be done in $O(n|E(P)|)$ time. The calculation must be performed after every reduction step.

3.1. Simulation

To determine the effectiveness of the algorithm, simulation studies were conducted. The algorithm was given as input connected random graphs. A random graph is a graph in which the probability of an edge between two nodes is fixed. A connected random graph is obtained by generating random graphs until a connected one is found. The average number of edges per node in a n node random graph with edge probability p is np. Since the algorithm is heuristic, we evaluate its performance with respect to three parameters: the number of nodes in a task graph (n), the edge density of a task graph (p) and the degree bound of a processor graph (r).

For the number of nodes, we arbitrarily choose graphs with 24, 48, 64 and 100 nodes. The $O(\sqrt{nr}\,|E(G)|)$ complexity of factoring combined with the $O(n|E(P)|)$ complexity of the hill climbing part of the algorithm limited the maximum size to approximately 100 nodes (approximately 45 minutes of CPU time for a 100 node graph on a HP9000-300 workstation). For the degree bound, we used r=3, 6 and

8.

To determine performance with respect to the *density* or number of edges in the task graph, a wide range of edge probabilities was used starting at $r/(n-1)$. This starting probability insured that the average degree of a node in the graph was greater than or equal to r. Tasks graphs in which all nodes have degree less than r have trivial optimal solutions. For dense task graphs, *i.e.*, $p >> r/(n-1)$, the performance of the algorithm increases monotonically as p approaches 1 as explained in Section 3.2. However, when the average degree of a node in the task graph was close to the degree of a node in the processor graph, the performance is difficult to predict since it first decreases and then increases as p increases. Therefore, we used small increments in probability when the average degree of the task graph was close to r, the degree of a node in the processor graph. The actual edge probabilities were chosen to be .15, .2, .25, .35, ... ,.95 for 24 nodes, .065, .09, .115, .140, .165, .190, .2, .3, ... ,.9 for 48 nodes, .05, .075, .1, .125, .15, .175, .2, .3, ... ,.9 for 64 nodes and .04, .05, .06, .1, .2, ... ,.9 for 100 nodes.

The algorithm was run on each of the graphs with the constant **no_of_tries** set to 3. For each combination of n and p, 10 different graphs were used as input and two statistics were collected and averaged. The statistics collected were as follows:

(1) The lower bound on TE given in Eq. (2).

(2) The TE of the graph generated by the MTE synthesis algorithm with respect to an input graph.

The ratio of these two values, (1)/(2), is plotted in Figure 4 for all combinations of n and r. The ratio for r=3 is plotted as a solid line, r=6 as a dotted line and r=8 as a dashed line.

3.2. Expected Performance

This section explains the performance of the MTE synthesis algorithm by assuming that all unmatched edges in the task graph have expansion equal to the AFD of the synthesized graph. Thus, if the AFD of the synthesized graph and the number of matched edges can be predicted, then the performance of the algorithm can also be predicted.

Given G and r, let $TE_{opt}(G)$ be the TE of the graph generated by an optimal MTE synthesis algorithm with respect to G. To evaluate the goodness of our algorithm, we wish to determine the value of the ratio $TE_{opt}(G)/TE(G,P)$ for the graph P synthesized by the MTE synthesis algorithm. The value of $TE_{opt}(G)$ is bounded below by $TE_{lb}(G)$ given in Eq. (2). The expected value for the denominator can be estimated as follows:

Proposition

The expected value of the denominator is

$$E[TE(G,P)] = 2(E[|E_m(G,P)|] + \overline{AFD}(P)(|E(G)|-E[|E_m(G,P)|])).$$

where

$$\overline{TFD}(P)= \quad 2 \sum_{(u,v)\notin E(P)} l_P(u,v)$$

is the total forwarding distance attributable to those edges that are not in P,

$$\overline{AFD}(P)=\frac{\overline{TFD}(P)}{n(n-1)-nr}, \tag{4}$$

and $E[|E_m(G,P)|]$ is the expected number of *matched* edges.

Justification

Each matched edge in $E_m(G,P)$ increases the TE by one. Each edge $(u,v)\in E(G)$ such that $(g(u),g(v))\notin E(P)$ defines a random variable $L_{(u,v)}=l_P(u,v)$. Since G is random, the random variables $L_{(u,v)}$ are independent. Using the central limit theorem, it can be shown that the mean of the $L_{(u,v)}$ random variables is $\overline{AFD}(P)$. Since expected values sum linearly, the expected value of the sum of these expansions is

$$\overline{AFD}(P)(|E(G)|-E[|E_m(G,P)|]).$$

□

Hence, the ratio of the lower bound on TE to that actually achieved can be expressed as

$$\frac{TE_{opt}(G)}{TE(G,P)} \geq \frac{TE_{lb}(G)}{2(E[|E_m(G,P)|] + \overline{AFD}(P)(|E(G)|-E[|E_m(G,P)|]))}. \tag{5}$$

3.2.1. AFD of Synthesized Topologies

The value of Eq. (5) depends in part on the AFD of the synthesized graph. To determine an expected value for AFD, we compared the AFD of the graphs synthesized by our algorithm to the AFD of random regular graphs. The average AFD of a random regular graph was determined by generating 100 random regular graphs [13] and averaging their AFD. Results for the r=8 case were not obtained since the random

graph generating algorithm was slow due to its time complexity which is exponential in r. Using a HP9000-300 workstation, 12 hours of CPU time was required to generate one 8-regular random graph on 100 nodes.

The results of the study showed that the ratio of the AFD of the graphs synthesized by the MTE synthesis algorithm to the AFD of random regular graphs is not higher than 1.007 for low edge probability task graphs and not lower than .93 for high edge probability graphs. Therefore, we assume that the AFD of a graph synthesized by the MTE synthesis algorithm can be approximated by the AFD of a random regular graph.

3.2.2. Random Regular Graphs

Since the AFD of the synthesized topologies is close to that of random regular graphs, we studied the AFD of random regular graphs. In this study, random regular graphs with a relatively large number of nodes were generated and the AFD of each graph was calculated. More specifically, 3,4 and 6-regular graphs with n ranging from 24 to 1000 were generated using the algorithm described by Wormald [13] and their AFD calculated. Table 1 shows the average and variance of the AFD. The 24, 48, 64 and 100 node graphs were generated to predict the expected performance of the synthesis algorithm; the graphs with greater than 100 nodes were generated to determine how the AFD of random regular graphs increased with graph size.

In addition to calculating the AFD, a lower bound on AFD was calculated. The lower bound on AFD, $AFD_{lb}(P)$, is equal to $TFD_{lb}(P)$ divided by $n(n-1)$, the number of edges in a complete graph. Table 1 gives this bound for each n and r that a random regular graph was generated for. The fourth row of the table is the ratio of the AFD for the generated random regular graphs to the lower bound on AFD.

Table 1 shows that the ratio of the AFD of a random regular graph to the lower bound on AFD is less than 1.135. The variance of order 10^{-3} implies that the ratio is typical of random regular graphs. This result shows that random regular graphs have a AFD that is within 14% of a lower bound for AFD. Thus, if one wants a r-regular topology with small AFD, simply generate a random regular graph. Much work in the field of interconnection network design has centered on the problem of designing such graphs [14]; this result shows that random regular graphs are very good interconnection networks with respect to AFD.

The reasons why random regular graphs have an AFD close to the lower bound on AFD can be explained as follows. We note that if a graph has an AFD equal to the lower bound, then each node in the graph looks like the root of a r-tree [15]. In this way a maximum number of nodes can be packed at a minimum distance from a given node. Edges must be added to the leaf nodes of the tree to make it into a

r-regular graph. The added edges must ensure that every node in the graph have the same tree like structure. This tree like structure implies that the smallest cycle in a minimum AFD graph is size $O(log_r n)$. If the graph has a cycle smaller than this, there exist nodes in the graph that do not have the tree structure. At these nodes, fewer nodes are packed at a minimal distance away and hence the graph has a larger AFD. Therefore, for a fixed n and r, the AFD of a regular graph increases with the number of small cycles.

To determine the expected AFD of a random regular graph, the expected number of small cycles must be known. Bollobas [16] and Wormald [17] have independently studied the expected number of small cycles in random regular graphs. They showed that, for any fixed i, the expected number of i-cycles in a random r-regular graph is Poisson distributed with mean $(r-1)^i/2i$ as the number of nodes in the graph approaches infinity. This number is independent of n, assuming that n is large. Therefore, the number of i cycles remains constant as n increases. This implies that the ratio of the expected AFD to the lower bound should approach 1 as n increases. However, the simulation results given in Table 1 show that the ratio is neither strictly increasing nor decreasing as n is increased. This result can be partially explained by the asymptotic nature of the theorem of Bollobas and Wormald giving the expected number of small cycles. The theorem holds only as n approaches infinity. For instance, a 24 node graph may have zero 6-cycles whereas a 1000 node graph may have the predicted number. Another factor that affects the AFD is the manner in which the cycles overlap [5]. Since we have not been able to predict the effects of cycle overlap, a closed form equation for expected ratio of generated AFD to the lower bound on AFD has not been developed.

3.2.3. Comparison of Predicted and Simulation Results

To compute $E[TE(G,P)]$ and thus Eq. (5), we need to estimate $E[|E_m(G,P)|]$ and $\overline{AFD}(P)$. Let c be defined as

$$c = \sum_{u \in V(G)} min(d_G(u), r).$$

We note that $E[|E_m(G,P)|] \leq c$ since c is an upper bound on cardinality. As was shown using the cardinality synthesis algorithm [6], c is almost always (probability approachs 1 as the number of nodes in the graph approaches infinity) [18] equal to $E|[E_m(G,P)|]$. The simulation results (those comparing the AFD of the synthesized topology to that of random regular graphs and Table 1) show that $AFD(P)$ can be approximated by $z^* AFD_{lb}$ for a small constant z (where z is a function of n and r). Since

$$\overline{AFD}(P) = \frac{n(n-1)AFD(P) - nr}{n(n-1) - nr},$$

$\overline{AFD}(P)$ can be estimated by

$$\overline{AFD}(P) = \frac{zn(n-1)AFD_{lb}(P)-nr}{n(n-1)-nr}.$$

Figure 5 is a plot of Eq. (5) for the graphs used in the 24 and 100 node simulation. For the r=3 plots z was set to 1.13. This was the ratio of the TFD of a random regular graph to the lower bound on TFD for both the 24 and 100 node cases. As was mentioned, generating random 8-regular graphs took 12 hours of CPU time on a HP9000-300 workstation due to the exponential complexity of the generation algorithm. Thus, for the r=8 plots we used an estimate for z of 1.05 for n=24 and z=1.035 for n=100. These values were obtained by extrapolating from the r=3, 4 and 6 values.

The predicted and generated plots in Figure 5 have the same shape. That is, the ratio decreases to a minimum as the edge probability increases, and then the ratio increases to the value used for z as the input graph approaches a complete graph. The value of r affects how far the ratio decreases. The differences between the predicted and achieved values in the plots are caused by the estimate for $\overline{AFD}(P)$.

4. Mapping to Reduce TE

The problem of finding mappings that minimize communication delay is an area of active research in the field of parallel processing. Rosenberg and Synder [7, 19] defined the mapping problem in terms of total expansion and presented lower bound results together with specific mappings for a variety of task and processor graph topologies. Bokhari [20] presented a hill climbing algorithm for solving the cardinality mapping problem. Fukunaga, Yamada and Kasai [21] modeled the task and processor graphs as hypergraphs and then used a relaxation technique to find maximum cardinality mappings. Napolitano [22] used minimum bandwidth mappings to line graphs to find both minimum total and minimum maximum expansion mappings. In these last two papers the algorithms were compared to hill climbing as implemented by Bokhari for the respective optimization criteria. In both cases, the quality of the mappings found were approximately equal. The advantage of the algorithms over hill climbing were primarily speed. Therefore, to compare the TE of synthesis approach to the TE achievable through the mapping approach, we implemented a modified version of the algorithm developed by Bokhari [20]. It is similar to the MTE synthesis algorithm since both are based on a hill climbing heuristic.

The algorithm starts with a randomly chosen mapping and then examines all pairwise exchanges of two nodes to see if any exchange decreases the TE. The exchange, if it exists, that reduces the TE the most is then selected. Exchanges occur until no further reduction in TE is possible. At this point the mapping is randomly perturbed and the hill climbing procedure repeats again. After the first perturbation, if the hill

climbing procedure gets stuck in a local optimum with TE greater than the best mapping found so far, the algorithm terminates.

4.1. Simulation

The algorithm was used to map the 24, 64 and 100 node random task graphs described in Section 3.1 onto fixed topology processor graphs. The processor graphs for r=3 experiments had small TFD. For the n=24, r=3 case, the graph given by Cerf *et. al.* [15] was used. This graph has a TFD equal to the lower bound for TFD. For the 64 and 100 node, r=3 cases, the graphs were generated using simulated annealing according to a procedure given in the first author's dissertation [5]. The TFD of the 64 node graph was 1.4% higher than the lower bound on TFD, and the TFD of the 100 node graph was 2.8% higher. The r=8 experiments used an 8-nearest neighbor torus as a processor graph. These graphs had TFDs that were 10.6%, 45.7% and 48.4% higher than the lower bound for the n=24, 64 and 100 cases respectively. The algorithm was also used to map the 64 node task graphs onto a 64 node hypercube. The hypercube has a TFD that is 25.5% higher than the lower bound.

The results of the simulation are plotted in Figure 6. The solid line is the ratio of the lower bound on TE to synthesized TE. The dotted line is the same ratio computed for the mapping algorithm.

4.2. Comparison to Synthesis

For the r=3 set of experiments, the mapping algorithm produces results that are approximately 5% to 10% better in terms of TE for all but the lowest probability task graphs (p=$r/(n-1)$). At the low probabilities, more edges are matched by the synthesis algorithm and hence it performs better than the mapping method. As the task graphs became denser, however, the mapping algorithm outperforms the synthesis algorithm. This occurred since the processor graphs used for mapping have TFD that is only 0% to 2.8% higher than the lower bound on TFD, whereas the graphs synthesised by the synthesis algorithm have TFDs that were approximately 10% higher than the lower bound.

For the r=6 and 8 mapping experiments, the graphs produced by the synthesis algorithm had smaller TFD than those used for mapping for all probabilities. The TFD of the processor graphs used for the mapping experiments ranged from 10.6% to 48.4% higher than the lower bound. Since the TFD of the graphs synthesized by the MTE algorithm were always less than 10% higher than the lower bound, the TE of the synthesized topologies was less than that found by the mapping algorithm.

In summary, if a network topology with a TFD within approximately 0% to 10% of the lower bound is known, the mapping algorithm will perform better than our MTE algorithm for all but sparse graphs. For

sparse graphs, the synthesis algorithm performs better. However, graphs with small TFD are known for only a small subset of n and r [14, 15]. Furthermore, the technique used for generating the processor graphs for the r=3, n=64, 100 cases requires an inordinate amount of computation time; for example, 36 hours of CPU time on a HP9000-300 workstation were needed to generate a 100 node, r=3 graph. Thus, the technique is limited to relatively small graphs.

5. Implementation of r-reconfigurable Networks

We have shown that through the use of a r-reconfigurable network, a better match between a parallel task and processor system can be achieved. Constructing a r-reconfigurable network for n processors requires r copies of a n line permutation network. If electronic components are used to implement the permutation networks, either n^2 components (crossbar network) or complex wiring patterns (baseline network) are needed to implement the network. In contrast, we have investigated a r-reconfigurable network based on optical components [23]. The network has the potential for connecting up to 1,000 processors using only $O(nr)$ components. A n line permutation network consists of n optical sources, n electronically controllable optical deflectors (such as acousto-optic deflectors or mirrors mounted on servo motors) and n photosensitive detectors. Each processor sends data to its optical source for transmission and the address of the target processor to its deflector. The output of the optical source is directed to its corresponding deflector. The deflector steers the beam to the detector of the target processor. The data received by the detector is demodulated and sent to the target processor. Since laser beams do not interfere, any permutation of inputs to outputs can be formed. Due to the switching time of the deflectors reconfiguration time is limited to approximately 10 milliseconds for mirrors and 2 microseconds for acousto-optic deflectors.

There are also other networks that could be used as a r-reconfigurable network. Sawchuk, Jenkins and Raghavendra [24] describe an optical crossbar based on a spatial light modulator. Like our network, this network has a reconfiguration time in the millisecond range and therefore could be used as a r-reconfigurable network. In addition, Meiko Incorporated [25] is packaging the Inmos Transputer [26] in a computer system that allows manual reconfiguration of the network topology. Since each transputer has four i/o links, the processors in the system can be configured into any 4-regular graph.

6. Discussion and Summary

6.1. Dynamic Behavior

Up to this point the synthesis algorithm has been defined with respect to the static graph property of total expansion. The next question is what effect total expansion has on average message delay at run time. To answer this question we developed a queueing model which was used to calculate the average message delay through the network given a particular routing procedure, a traffic load between processors, and the task and processor topologies [5]. Using shortest path routing, the model showed that average message delay is a linear (unit slope) function of total expansion as long as the network was lightly loaded.

As the traffic load to the network increases, one or more processors in the network are forced to forward many messages. Hence, messages passing through these heavily loaded processors incur large delays, even if the number of processors through which a message is forwarded is small. Therefore, under heavy loading conditions the average message delay through the network is no longer a linear function of total expansion but instead is dominated by the average delay through the most heavily loaded queue in the network.

To minimize the delay caused by congestion, a more sophisticated routing scheme needs to be used. The problem of routing messages through a fixed network to minimize congestion is well studied [27,28,29]. Most of the algorithms to solve the routing problem start with a shortest path routing scheme. Therefore, it seems likely that a processor graph with small TE will provide smaller average message delays than processor graphs with larger TE.

6.2. Other Optimization Techniques and Criteria

In this paper we have taken the approach of generating a processor graph with small total expansion for a given task graph. The dynamic models show that routing is also an important factor in determining average message delay. Thus, an alternate means of synthesizing topologies would entail generating processor graphs for which messages could be routed evenly among the processors along with minimizing the total expansion. To be effective, such an approach would require that the amount of traffic between processors be known. This could be accomplished if the edges in the task graph were weighted according to the amount of traffic that passed between the tasks. Given the amount of traffic flowing between processors, the problem of choosing the right combination of routing function and processor topology seems to be a difficult problem due to the complex interplay between topology and routing function. This problem might be formulated in terms of finding a minimum *forwarding index* routing [30]. For a given set of source-destination pairs, a minimum forwarding index routing is a routing that minimizes the maximum number of paths that pass through any given vertex. Further research is needed to determine whether such

an approach can provide better results than the approach of first synthesizing a small total expansion processor graph and then determining an optimum route through that graph.

We have attempted in this paper to reduce average message delay. An alternative would be to reduce the maximum delay that any one message might encounter. Here, an optimization criteria that minimizes the maximum length that any one edge gets expanded could be used. This problem is different than the total expansion problem since an optimum solution for one criteria is not necessarily an optimum solution for the other. Nevertheless, a hill climbing heuristic similar to the one given in this paper could be used to find a processor graph with small maximum expansion. We have implemented such a heuristic [5] and its simulation results are summarized as follows: For dense random task graphs, the synthesis approach produces smaller maximum expansion than the similar mapping approach unless a static regular processor graph with a diameter within 35% of the Moore bound [14] is used. For sparse random task graphs, the synthesis algorithm generates smaller maximum expansion than mapping, even if the the processor graph used for the mapping approach has diameter close to the Moore bound.

This paper used a hill climbing for finding small total expansion processor graphs. As an alternative, we could have used a probabilistic hill climbing technique known as simulated annealing [31]. Using simulated annealing, perturbations that do not improve the criteria function are made according to a time dependent parameter. In this manner, jumps out of local optima can be made. Simulated annealing has been shown be be more effective than hill climbing for the cardinality mapping problem [5]. Similar results might hold for the total expansion mapping problem but we have not conducted any such experiments since the cost of evaluating the effect of a perturbation precludes the use of simulated annealing, which requires the large number of trial perturbations. Therefore, in this paper we have chosen to compare the synthesis and mapping techniques through the use of a simple hill climbing algorithm.

6.3. Summary

We presented an algorithm to synthesize r-regular processor topologies that have small TE with respect to a given task graph. In addition, we compared the performance to a similar mapping algorithm. The performance of the MTE synthesis algorithm presented can be explained by assuming that all of the unmatched edges in the task graph have an expansion equal to the average forwarding distance of the synthesized topology. It appears that the synthesized topologies have TFD that are approximately equal to the TFD of random regular graphs. Simulation studies show that the TFD of random regular graphs is close to the lower bound on TFD, which explains why the MTE synthesis algorithm generates topologies that have TE close to the lower bound on TE. In comparison to mapping, the synthesis results are better unless a

topology with a TFD within 10% of the lower bound is known. In this case, mapping is better except for sparse random graphs. We note that if an optimum synthesis algorithm for the MTE synthesis problem is known, it should always produce results at least as good as any mapping algorithm used with any fixed processor topology.

Acknowledgements

We would like to thank the referees, whose comments and suggestions helped to improve the presentation of this paper.

References

[1] Batcher, K.E., "Design of a Massively Parallel Processor," *IEEE Transactions on Computers*, pp. 836-840 (Sept. 1980).

[2] Snyder, L., "Introduction to the Configurable, Highly Parallel Computer," *Computer* **15**(1), pp. 47-56 (Jan. 1982).

[3] Li, H., Wang, C., and Lavin, M., "The V Language for Polymorphic Architectures and Algorithms," in *Intermediate-Level Image Processing*, ed. M.B. Duff, Academic Press (1986).

[4] Heath, M.T., "The Hypercube: A Tutorial Overview," pp. 7-11 in *Hypercube Multiprocessors 1986*, ed. M.T. Heath, Society for Industrial and Applied Mathematics, Philadelphia, PA.

[5] Smitley, D.L., *The Utilization of Processors Interconnected with a Reconfigurable Network, Ph.D. Dissertation*, Dept. of Computer and Information Science, University of Pennsylvania (April 1987).

[6] Lee, I. and Smitley, D.L., "A Synthesis Algorithm for Reconfigurable Interconnection Networks," *To Appear: IEEE Trans. on Computers.*

[7] Rosenberg, A.L., "Data Encodings and Their Costs," *Acta Informatica* **9**, pp. 273-292 (1978).

[8] Agrawal, D.P., Janakiram, V.K., and Pathak, G.C., "Evaluating the Performance of Multicomputer Configurations," *Computer* **19**(5), pp. 23-37 (May 1986).

[9] Cerf, V., Cowan, D.D., Mullin, R.C., and Stanton, R.G., "A Lower Bound on the Average Shortest Path Length in Regular Graphs," *Networks* **4**, pp. 335-342 (1974).

[10] Garey, M.R., Johnson, D.S., and Stockmeyer, L., "Some Simplified NP-complete Graph Problems," *Theoretical Computer Science* **1**, pp. 237-267 (1976).

[11] Gabow, H.N., "An Efficient Reduction Technique for Degree-Constrained Subgraph and Bidirected

Network Flow Problems," *Proc. 1983 ACM Symposium on Theory of Computing*, pp. 448-456 (1983).

[12] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, Mass. (1974).

[13] Wormald, N.C., "Generating Random Regular Graphs," *Journal of Algorithms* **5**, pp. 247-280 (1984).

[14] Bermond, J.C., Bond, J., Paoli, M., and Peyrat, C., "Graphs and Interconnection Networks: Diameter and Vulnerability," in *Surveys in Combinatorics, London Mathematical Society Lecture Note Series #82*, ed. E.K. LLoyd, Cambridge University Press (1983).

[15] Cerf, V.G., Cowan, D.D., Mullin, R.C., and Stanton, R.G., "A Partial Census of Trivalent Generalized Moore Networks," *Lecture Notes in Mathematics*(452), pp. 1-27 (1975).

[16] Bollobas, B., "A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs," *European Journal of Combinatorics* 1, pp. 311-316 (1980).

[17] Wormald, N.C., "The Asymptotic Distribution of Short Cycles in Random Regular Graphs," *Journal of Combinatroical Theory* **Series B 31**, pp. 168-182 (1981).

[18] Palmer, E.M., *Graphical Evolution, An Introduction to the Theory of Random Graphs*, John Wiley, New York (1985).

[19] Rosenberg, A.L. and Snyder, L., "Bounds on the Costs of Data Encodings," *Mathematical Systems Theory* **12**, pp. 9-39 (1978).

[20] Bokhari, S., "On the Mapping Problem," *IEEE Transactions on Computers* **C-30**(3), pp. 207-214 (Mar. 1981).

[21] Fukunaga, K., Yamada, S., and Kasai, T., "Assignment of Job Modules onto Array Processors," *IEEE Transactions on Computers* **C-36**(7), pp. 888-891 (July 1987).

[22] Napolitano, L.M. Jr., "Revisting the Mapping Problem," Technical Report, Sandia National Lab, Livermore, CA (1986).

[23] Smitley, D.L., Lee, I., and Goldwasser, S., "The Design of an Optical Network and Its Utilization," *Proc. 1986 Optoelectronics and Laser Applications in Science and Engineering Conference*, Los Angeles, CA (Jan. 1986).

[24] Sawchuk, A.A., Jenkins, B.K., Raghavendra, C.S., and Varma, A., "Optical Matrix-Vector Implementation of Crossbar Interconnection Networks," *Proc. 1986 Int. Conf. on Parallel Processing*, pp. 401-404 (Aug. 1986).

[25] Meiko, Incorporated, *The Computing Surface*, 6201 Ascot Drive, Oakland CA 94611.

[26] Inmos, Limited, *IMS T424 Transputer Reference Manual*, 1984.

[27] Kleinrock, L., *Queueing Systems, Volume 2: Computer Applications*, John Wiley and Sons, Inc. (1976).

[28] Bianchini, R.P. Jr. and Shen, J.P., "Interprocessor Traffic Scheduling Algorithm for Multiple-Processor Networks," *IEEE Trans. on Computers* **C-36**(4), pp. 396-409 (April 1987).

[29] Fratta, L., Gerla, M., and Kleinrock, L., "The Flow Deviation Method - An Approach to Store-and-Forward Communication Network Design," *Networks* **3**, pp. 97-133 (1973).

[30] Chung, F.R.K., Coffman, E.G. Jr., Reiman, M.I., and Simon, B., "The Forwarding Index of Communication Networks," *IEEE Transactions on Information Theory* **IT-33**(2), pp. 224-232 (March 1987).

[31] Kirkpatrick, S., Gelatt, C.D. Jr., and Vecchi, M.P., "Optimization by Simulated Annealing," *Science* **220**(4598), pp. 671-680 (May 1983).

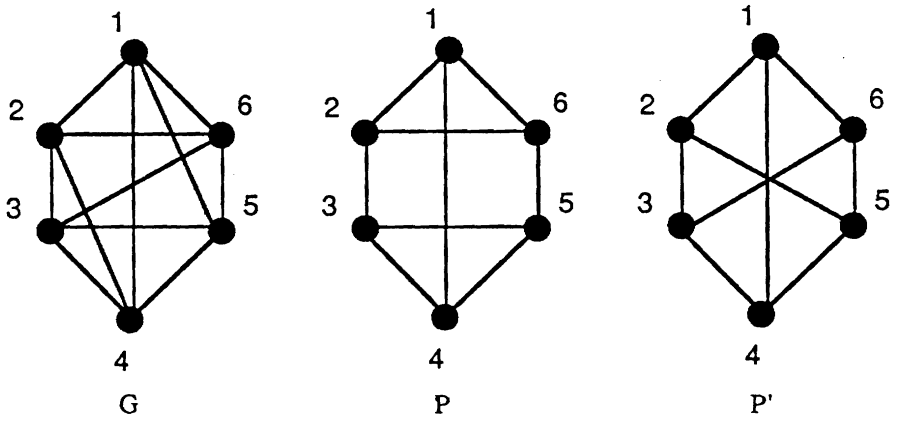


Figure 1. A Sythesis Example

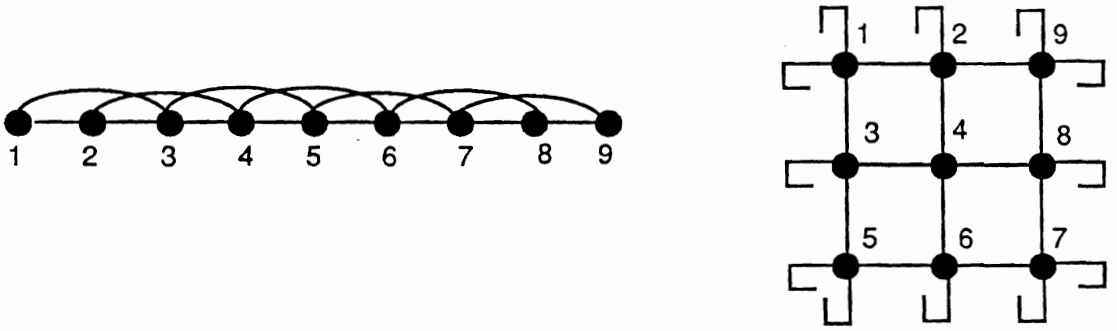
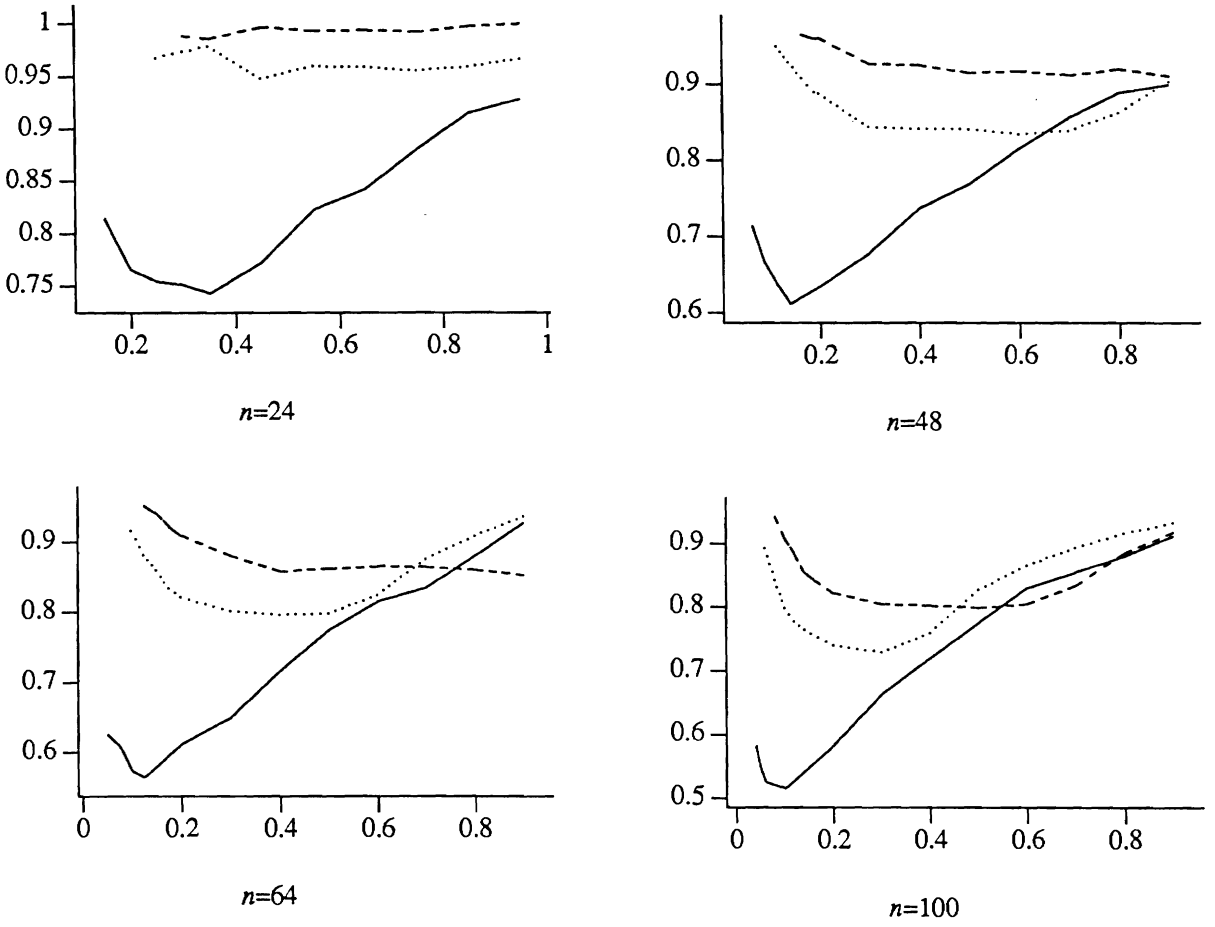


Figure 2. Synthesis vs. Mapping

```
input = a connected graph G, and an integer r≥2
output = a r-regular connected graph best_graph

best_graph := An r-regular connected graph generated by the cardinality algorithm
poorer := 0
while (poorer < no_of_tries) do
      P := best_graph
      Calculate l_P(g(u),g(v)) for all (u,v)∈E(G)
      max_expansion := maximum l_P(g(u),g(v)) for all (u,v)∈E(G)
      max_edges := {(u,v):(u,v)∈E(G) and l_P(g(u),g(v))= max_expansion}
      (x,y) := an edge chosen with equal probability from max_edges
      x_edges := {(x,w):(x,w)∈E(P)}
      y_edges := {(y,w):(y,w)∈E(P)}
      (x,x_delete) := an edge chosen with equal probability from x_edges
      (y,y_delete) := an edge chosen with equal probability from y_edges
      P = {V(P),E(P)∪(x,y)∪(x_delete,y_delete)}
      P = {V(P),E(P)−(x,x_delete)−(y,y_delete)}
      if total_expansion(G,P) ≤ total_expansion(G, best_graph) then
            best_graph := P
            poorer := 0
      else
            poorer := poorer + 1
      end if
end while
```

Figure 3. A Hill Climbing Heuristic to Reduce TE



Solid - Lower Bound TE / TE algorithm TE, r=3
Dotted - Lower Bound TE / TE algorithm TE, r=6
Dashed - Lower Bound TE / TE algorithm TE, r=8

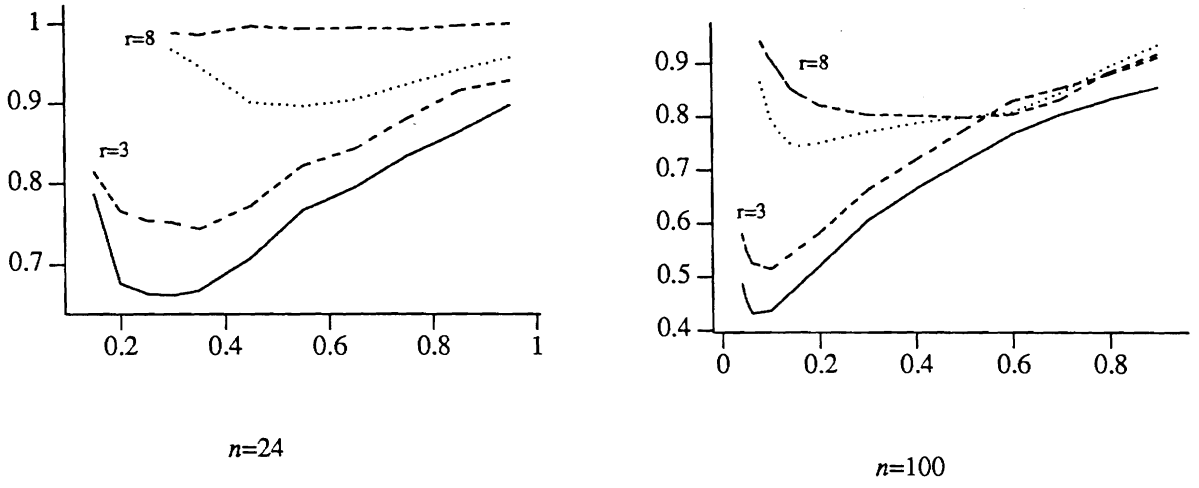
Figure 4. Comparison of TE Ratios

Table 1. Average Forwarding Distances of Random Regular Graphs

	r = 3						
	24	48	64	100	200	500	1000
Average	2.884	3.791	4.183	4.827	5.753	7.071	8.061
Variance	5.1e-3	8.5e-3	6.8e-3	5.7e-3	1.6e-3	1.2e-3	4.1e-4
Low Bound	2.565	3.340	3.762	4.273	5.191	6.515	7.492
Avg./Low Bound	1.125	1.135	1.112	1.130	1.107	1.086	1.076

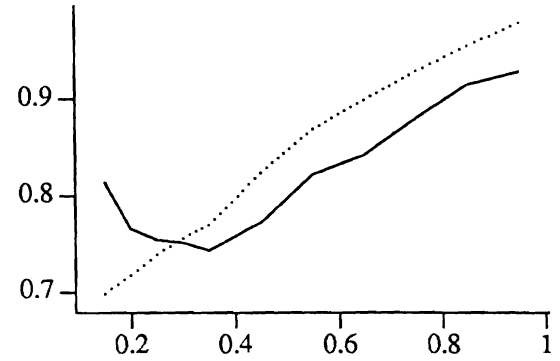
	r = 4						
	24	48	64	100	200	500	1000
Average	2.283	2.901	3.163	3.558	4.177	5.004	5.635
Variance	1.4e-3	1.0e-3	1.25e-3	1.1e-3	5.7e-4	4.4e-5	8.3e-5
Low Bound	2.130	2.574	2.857	3.273	3.834	4.565	5.283
Avg./Low Bound	1.072	1.015	1.107	1.087	1.090	1.096	1.067

	r = 6						
	24	48	64	100	200	500	1000
Average	1.854	2.291	2.475	2.745	3.179	3.748	4.183
Variance	2.9e-4	1.5e-4	2.0e-4	1.7e-4	6.9e-5	1.3e-5	9.5e-6
Low Bound	1.739	2.234	2.429	2.636	2.945	3.579	3.871
Avg./Low Bound	1.067	1.026	1.019	1.041	1.080	1.047	1.081

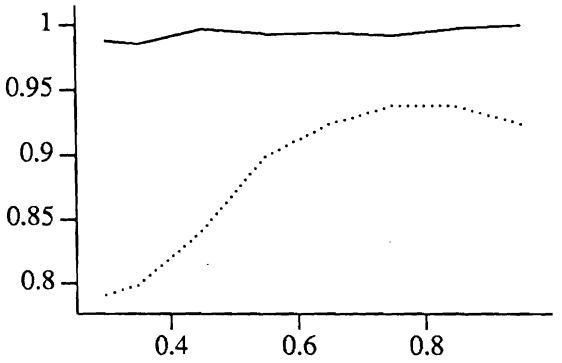


Solid - r=3, Predicted
Dotted - r=8, Predicted
Dashed - TE from Simulation

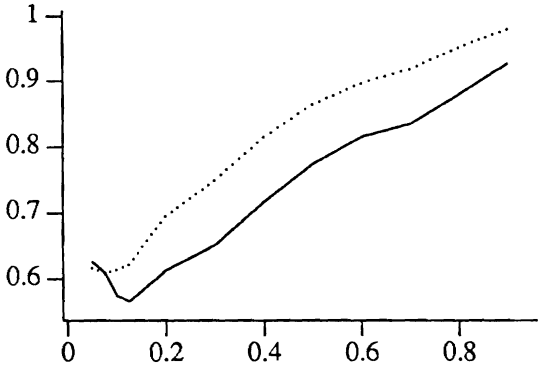
Figure 5. Predicted vs. Simulated Ratios



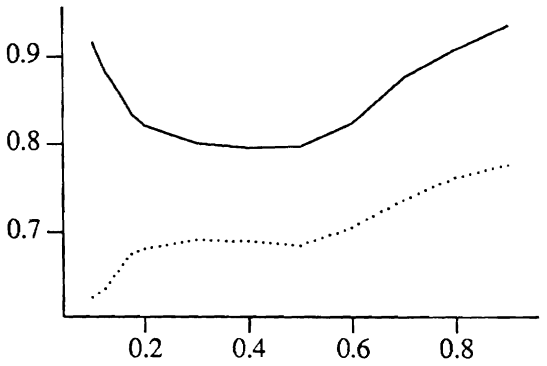
n=24, r=3



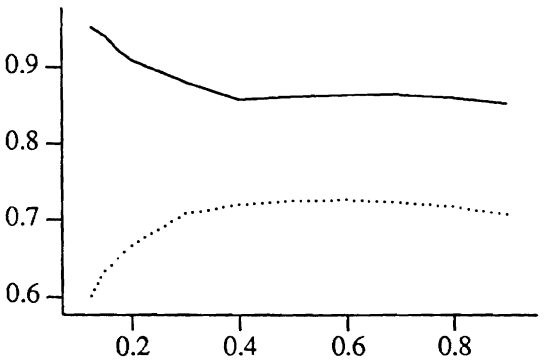
n=24, r=8



n=64, r=3



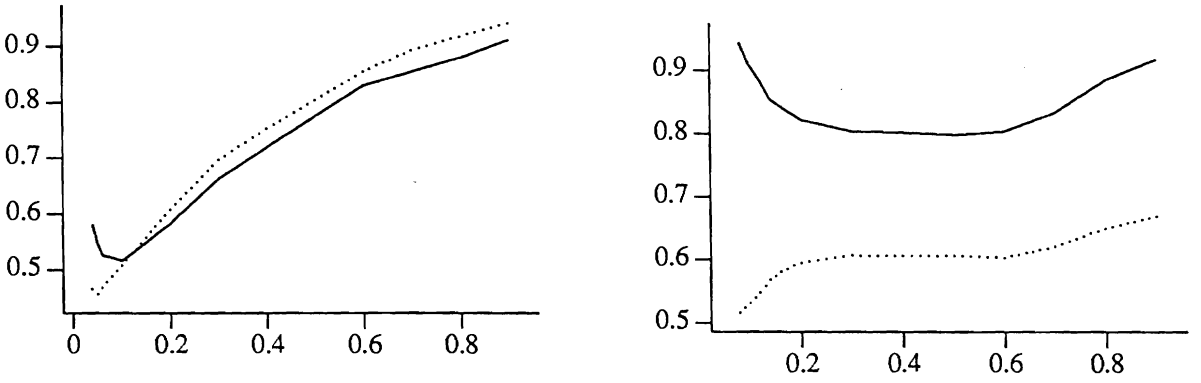
n=64, r=6



n=64, r=8

X axis - Probability
Y axis - $TE_{lb}(G) / TE(G,P)$
Solid - Synthesis Ratios
Dotted - Mapping Ratios

Figure 6. Synthesis vs. Mapping



n=100, r=3

n=100, r=8

X axis - Probability
Y axis - $TE_{lb}(G) / TE(G,P)$
Solid - Synthesis Ratios
Dotted - Mapping Ratios

Figure 6. (Continued)