Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

1-1-2011

An Exploration Of Heterogeneous Networks On Chip

Allen Gary Grimm
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Let us know how access to this document benefits you.

Recommended Citation
Grimm, Allen Gary, "An Exploration Of Heterogeneous Networks On Chip" (2011). *Dissertations and Theses.* Paper 185.
https://doi.org/10.15760/etd.185

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

An Exploration Of Heterogeneous Networks On Chip

by

Allen Gary Grimm

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Christof Teuscher, Chair
Dan Hammerstrom
Xiaoyu Song

Portland State University
©2011

Abstract

As the the number of cores on a single chip continue to grow, communication increasingly becomes the bottleneck to performance. Networks on Chips (NoC) is an interconnection paradigm showing promise to allow system size to increase while maintaining acceptable performance. One of the challenges of this paradigm is in constructing the network of inter-core connections. Using the traditional wire interconnect as long range links is proving insufficient due to the increase in relative delay as miniaturization progresses. Novel link types are capable of delivering single-hop long-range communication. We investigate the potential benefits of constructing networks with many link types applied to heterogeneous NoCs and hypothesize that a network with many link types available can achieve a higher performance at a given cost than its homogeneous network counterpart. To investigate NoCs with heterogeneous links, a multiobjective evolutionary algorithm is given a heterogeneous set of links and optimizes the number and placement of those links in an NoC using objectives of cost, throughput, and energy as a representative set of a NoC's quality. The types of links used and the topology of those links is explored as a consequence of the properties of available links and preference set on the objectives.

As the platform of experimentation, the Complex Network Evolutionary Algorithm (CNEA) and the associated Complex Network Framework (CNF) are developed. CNEA is a multiobjective evolutionary algorithm built from the ParadisEO framework to facilitate the construction of optimized networks. CNF is designed and used to model and evaluate networks according to: cost of a given topology; performance in terms of a network's throughput and energy consumption; and graph-theory based metrics including average distance, degree-, length-, and

link-distributions.

It is shown that optimizing complex networks to cost as a function of total link length and average distance creates a power-law link-length distribution. This offers a way to decrease the average distance of a network for a given cost when compared to random networks or the standard mesh network. We then explore the use of several types of constrained-length links in the same optimization problem and find that, when given access to all link types, we obtain networks that have the same or smaller average distance for a given cost than any network that is produced when given access to only one link type. We then introduce traffic on the networks with an interconnect-based packet-level shortest-path-routed traffic model. We find heterogeneous networks can achieve a throughput as good or better than the homogeneous network counterpart using the same amount of link. Finally, these results are confirmed by augmenting a wire-based mesh network with non-traditional link types and finding significant increases the overall performance of that network.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Moore's law states that processor performance doubles every 18 months [48]. Traditionally, these performance increases came from miniaturization allowing an increasing number of transistors per chip. The relative delay of wires scales quite poorly compared to the relative delay of transistors [7, 29, 60] resulting in an increasing relative delay between distant components. However, new links types including Carbon Nanotubes (CNTs) [16], millimeter wave wireless (mmWave) antennae [31], and photonic links [4] are capable of directly connecting distant components. For example, as seen in Figure 1.1, communication between distant

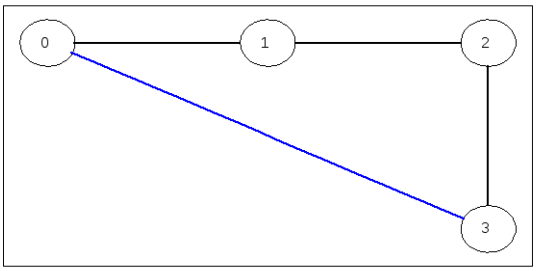


Figure 1.1: Example network with short- and long-range links.

components 0 and 3 would have to travel across wired links through components 1 and 2, resulting in additional communication overhead, but using one of the novel link types will allow a direct connection between node 0 and 3 (as shown by the blue link) making inter-component communication more effective. For this reason, we adopt the hypothesis that building networks with a heterogeneous set of links

will give better performance for a given cost due to enabling long-range single-hop communication.

1.2 Challenges

The integration of all the components of an electronic system into a single integrated circuit is known as *System on Chip* (SoC) [9]. Through enhanced parallelism, improving system performance is possible without the need for better individual processors. SoC communication structures are given in Figure 1.2. The

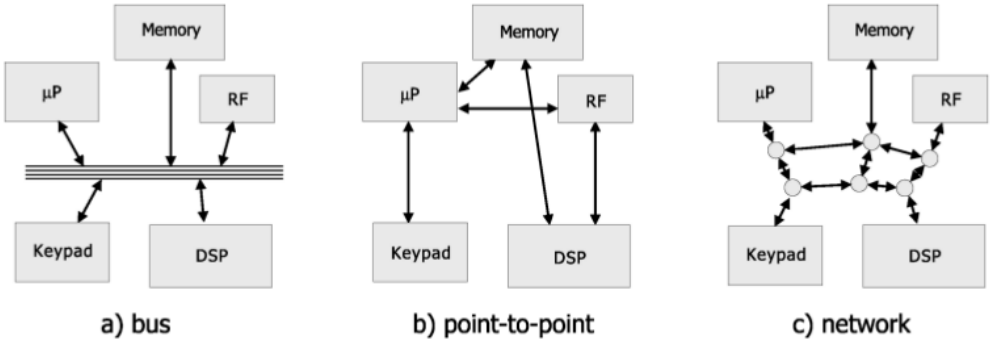
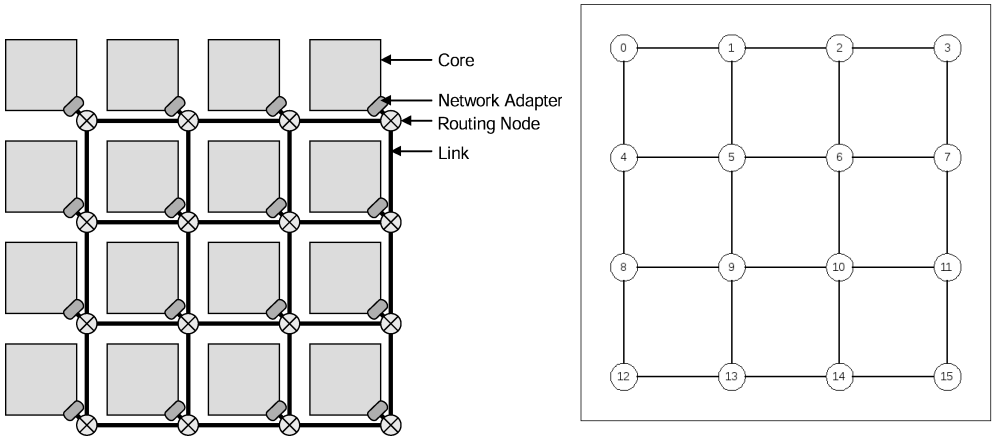


Figure 1.2: SoC communication structures. Source: [9].

bus connects all cores to a single communication link which quickly becomes a bottleneck to communication as the system scales in size. Point-to-point is often used with buses ad hoc to form customized SoCs. Arranging the components an inter-component communication as a network offers a scalable paradigm that reduces the time-to-market associated with fully ad hoc designs [8]. The current bottleneck to performance in such a network is in the communication between cores [29].

Networks on Chips (NOC) is an emerging paradigm of SoC where each component known as a core is connected through a network adapter to a router. Information travels through the network originating at a core, passing through the

adapter into a router, through a sequence of links and routers until a given piece of information reaches is destination where it travels from the final router through the adapter and into the destination core. An example 4x4 "mesh" NoC is shown in Figure 1.3a. We group each core with its associated adapter and router into a



(a) Mesh NoC. Source: [9]. (b) Our mesh representation.

Figure 1.3: Network representations.

node, leaving our representation of a *network* to be the set of nodes and links. The representation of the 4x4 mesh network is shown in Figure 1.3b. Decoupling the simulation of the processing nodes with the simulation of the inter-node communication saves significant computation time. Supporting this method, [49] employs the transaction level approach to improve simulation performance while maintaining simulation accuracy. [43] also operates at the transaction level through the decoupling of simulation of processing nodes and the interconnection fabric and shows this method to maintain a very high (near 100%) simulation accuracy.

In this thesis, we investigate the value of integrating multiple type of links into a single network – what we call a heterogeneous network – by defining a sets of links with user-chosen attributes, using an Evolutionary Algorithm (EA) to optimize

the number of each link type and the placement of those links, and observing the relation between what link types were available, what fitness function was used in the EA, and the resultant networks. The first challenge of this thesis is to assess the value of heterogeneous links in abstract networks. Then, through a series of experiments, we establish an experimentation methodology and build a case supporting the use of many link types in NoCs.

1.3 Methodology

The methodology used in this thesis is designed to investigate properties of optimized networks without imposing any preconceptions of other construction techniques. The network design problem, even when each link is equal and only spanning trees are considered, is proven to be NP-Hard [33] making the systematic enumeration and evaluation of every possible network computationally infeasible. Rather, we select the stochastic search method of optimization by utilizing an *Evolutionary Algorithm* (EA) to construct the networks [24].

Our experimentation process is broken into three stages. First, all initial parameters of the evolution are selected. This includes all characteristics of the networks to be evolved (number of nodes, node placement, the number of link types, and the characteristics of each link type), as well as all settings on the EA itself (the metrics to use in the fitness function, how those metrics are to be aggregated, the population size, the number of generations, and the mutation and crossover rates). Second, the evolutionary algorithm is ran until completion. Finally, the resulting optimized networks are analyzed and connections are drawn between the metrics chosen, link types available, and properties of the resulting networks.

1.4 My Contributions

All experiments included in this thesis were performed using the *Complex Network Evolutionary Algorithm* (CNEA) - a combination of three software components. First, the *Complex Network Framework* (CNF) is a network simulation and evaluation utility I wrote in C++ that is designed to remain scalable (both in terms of network size and simulation complexity), computationally efficient, and usable both as an API and a stand-alone program. The second is the EA itself constructed as an application/extension of ParadisEO - a template-based C++ evolutionary computation library [14]. The third and final piece is the set of supporting scripts written in bash, gnuplot and octave that made the experimentation possible both in the sense of data analysis (network visualization, data analysis, and general graphing) and data collecting (initialization of experimentation file structure, systematic instantiation of CNEA, data compilation, and the systematic executing of data analysis scripts).

I have contributed the following in this thesis:

- Complex Network Framework (CNF). (See Section 3.1)
  - CNF is a network simulator. Given a network through one of its various special-purpose constructors (i.e., mesh, fully connected, random, or ring network) or through a file, it is capable of evaluating a network's objectives (including cost, average distance, throughput at a given injection rate, and critical traffic level) and evaluating various distributions (including link-length, node-throughput, link-throughput, and shortest-path-length).
- Complex Network Evolutionary Algorithm (CNEA). (See Section 3.2)

- CNEA is a stochastic optimization tool used to construct networks (both complex networks and networks on chips are built in this thesis). It is designed to remain as expandable as possible through application-independent interfaces and representation. The representation independent components of the EA are used from the ParadisEO API's library. I wrote the representation and all corresponding functions (copy constructor, destructor, print, mutation, crossover and fitness functions). The fitness function interfaces with CNF to model a network and evaluate it to return the desired objectives.

- Supporting scripts. (See Section 3.4.6)
  - The file structure is standardized for all experiments. CNEA is set to save its best networks and fitness data into standardized files. These standards are used as the basis for creating universal "parsing scripts" that scan through an experiment's files for all evolutions ran and compiles that evolution's data to return network plots, convergence curves, files for objective data, and network file saves. This data is compiled into a single folder and copied off server onto my work computer for further specialized analysis.

Experiments

- Optimization of abstract networks.
  - In Experiment 1, networks are evolved to average distance and cost. Short- and long-range links are used given any evaluated preference to cost and average distance while more links are used when preference is placed on performance. In Experiment 2, given sets of links with

mutually exclusive lengths, it is found that the more costly links are used only when preference is placed in the performance of a network rather than cost. This methodology was also used to aid in [69]. In Experiment 3, the relationship of using the more costly links when preference is placed in performance is again seen when performance is the result of a traffic situation.

- Optimization of heterogeneous networks on chip
  - The effects of optimization using different traffic patterns is first explored in Experiment 4. It is found that specialization to a traffic pattern does occur but the effect can be somewhat alleviated if all traffic patterns of interest are included in the evolution process. Finally, in Expeirment 5, the application of NoCs with heterogeneous links is explored. A correlation is again found between the properties of an optimized network and the properties of the links used in that network.

Chapter 2

Background

2.1 Foundation in Graph Theory

The origins of graph theory reach back to the Konigsberg bridges problem done by Leonhard Euler in 1736 [23]. Given the two islands and seven bridges as shown in Figure 2.1, the citizens of Konigsberg asked Euler if it is possible to cross every

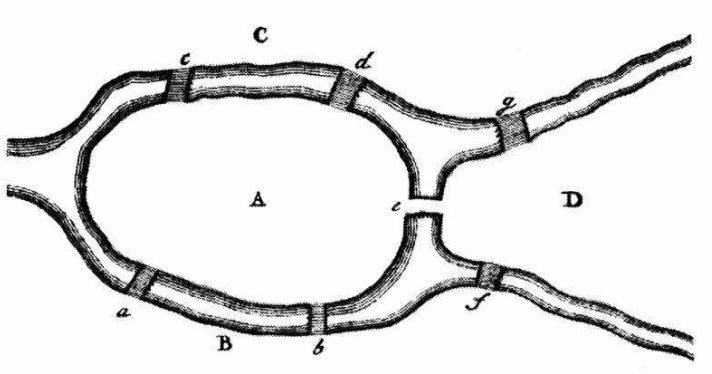


Figure 2.1: The Konigsberg bridges problem. Source: [23].

bridge exactly once during a walk. Euler modeled the scenario as a graph to prove that such a path is indeed not possible. A graph G is an ordered pair consisting of a set of vertices V and a set of edges E where each edge is a two element subset of V [73]. These definitions are shown in Equations 2.1, 2.2, and 2.3 respectively.

$$G = \{V, E\} \tag{2.1}$$

$$V = \{v_1, v_2, \ldots, v_N\} \tag{2.2}$$

$$E = \{e_1, e_2, \ldots, e_M\} \tag{2.3}$$

Within this notation, the Konigsberg bridge problem, as seen in Figure 2.2 is modeled as graph $G = \{V, E\}$, where V is the set of land masses $\{0, 1, 2, 3\}$ and E is the set of bridges $\{\{0, 2\}, \{0, 2\}, \{0, 3\}, \{0, 3\}, \{2, 1\}, \{0, 1\}, \{3, 2\}\}$.

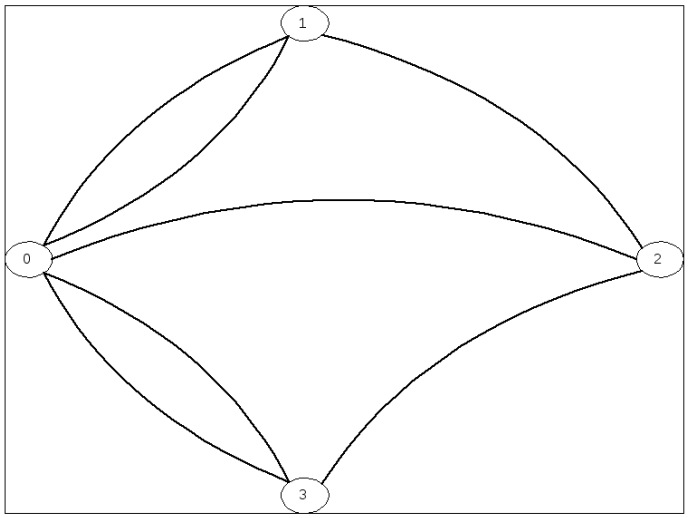


Figure 2.2: The Konigsberg bridges problem: graphical representation.

One of the many significant developments in the field was the conception of random graphs through the work done by Erdős and Rényi [20–22]. With the models of random graphs came a new way to model and study real-world networks. As the field progressed, these real-world networks became known as complex networks.

2.2 Complex Networks

A complex network is a system of entities with interconnections where the interconnections form a non-trivial topology [66]. Examples of complex networks are integrated circuits [59] and the internet [76]. We model a network as a graph where the entities, called *nodes* are modeled as vertices and the interconnections, called *links*, are modeled as edges. We will investigate abstract networks and networks

applied to integrated circuits, but first we need to discuss the objectives we use to determine the cost and quality of a network and survey some related work.

2.2.1 Cost of a Graph

The first form of analysis of a graph we look at is cost. We define the cost of a graph cG as the sum of the cost of the edges in that graph

$$cG = \sum_{i=1:nL} cost(e_i) \tag{2.4}$$

and the cost of a link as a function of the length of that link. For this discussion, the cost of a link will equal the euclidean length of that link.

$$cE = length(e_i) \tag{2.5}$$

Given the 4x4 mesh network in Figure 1.3b, the cost is 24.00.

2.2.2 Average Distance

The next form of analysis of a network is the average distance. The distance between two nodes is the length (as the number of links traversed) of the shortest path between those two nodes. The average distance AD is the sum of the paths from every node i to every other node j divided by the total number of i, j pairs.

$$AD = \sum_{i,j \in E(G) | i \neq j} d_{i,j}/(N * (N-1)) \tag{2.6}$$

Given the 4x4 mesh network in Figure 1.3b, the average distance is 2.67.

2.2.3 Small World Networks

The *small world* phenomena describes the surprisingly common occurrence of encountering friends of friends in unexpected situations. It was experimentally investigated by [28]. People were asked to keep a list of every person they encountered for 100 days resulting in lists of 500 people on average. Given the United State's population of 200 million, Gurevitch claimed that any two randomly selected people have a 1-in-200,000 chance of begin directly associated directly while, then two random people are allowed to have two intermediate contacts, the chances of the two selected people being associated drops to 1-in-2.

Another approach used to "flesh out" the small world phenomena was done by Stanley Milgram in 1967 [47]. Randomly chosen individuals from Wichita, Kansas and from Omaha, Nebraska were asked to send a package to a specific "target" individual in Cambridge and Sharon, Massachusetts respectively. If they didn't know the target on a first-name basis, they were asked to send the package to someone they were on a first-name basis with who would be most likely to know the target individual. Of the 160 packages sent to volunteers in Nebraska, 44 packages arrived at their target with an average of 5 intermediate contacts. This procedure was repeated [70] and, through a pool of 296 starting volunteers, 64 packages arrived at the target with an average number of intermediate contacts numbering 5.2. These studies are the foundation of the idea called *six degrees of separation* [71] - the powerful idea that any person is separated from any other person on the planet by an average of only six social connection. In network term, this is the claim the the social network has an average distance of around 6 which turn out to be a remarkably low number compared to any totally random or totally ordered networks.

The Watts-Strogatz Model

The Watts-Strogatz model [72] builds networks by interpolating between a totally ordered and a totally random topology. It is the first construction technique to build graphs that show the small world properties of a low average distance and a high clustering coefficient. It interpolates between an ordered graph and a random graph by first placing the vertices in a D-dimensional lattice and connecting each node to its E nearest neighbors and then randomly rewiring the current edges with a chosen probability p. In such a way, the parameter p explicitly sets the amount of disorder to inject into a network. The full construction procedure is shown in Algorithm 1.

Algorithm 1 The Watts-Strogatz model. Source: [72].

```
1: procedure CONSTRUCT WATTS-STROGATZ RANDOM GRAPH(N, K, p)
2:     for all v ∈ V(G) do
3:         connect v to K nearest neighbors
4:     end for
5:     for all e ∈ E(G) do
6:         r ← rand(0, 1)
7:         if p < r then
8:             randomly choose u, v ∈ V(G) | u, v ∋ E(G) and u ≠ v
9:             rewire e to connect u, v
10:        end if
11:    end for
12: end procedure
```

These Watts-Strogatz networks are innovative in several ways. This model can construct networks that exhibit the behavior observed in real-world networks including the social network discussed in Section 2.2.3. Secondly, it enables the systematic exploration of networks with user-defined amounts of disorder. Through the latter, it is found that a small amount of disorder introduced into a graph that was otherwise regular transforms the locally connected architecture into a small

world architecture by introducing "short cuts" that connect distant vertices that would have otherwise been far apart (relative to path length).

2.2.4 Physically Realizable Networks

One discrepancy between the Watts-Strogatz model and real-world networks like integrated circuits [59] and the internet [76] is that the link-length distribution is uniform rather than exponential respectively. The exponential link-length distribution shows that there are fewer medium- and long-range links leaving most connections to locally connect nodes. Petermann and de Los Rios claim that the exponential distribution emerges as a way to save cost in constructing a network while maintaining a low average distance [57]. Lending more significance to this line of work, the total wire length has also been viewed as a measure of network complexity [39, 65].

Peterman and de Los Rios [57] give a method to construct graphs with an link-length distribution following the exponential $q(l) \sim l^{-\alpha}$. Given graph G consisting of a set of vertices V. Each vertex in V is placed one unit apart from each other in D dimensions of length L with every vertex connected to its immediate neighbors. Let nL be the number of links to be randomly added of a length distribution $q(l) \sim l^{-/alpha}$. As shown in Algorithm 2, randomly place all links.

Petermann and de Los Rios showed that a greater α caused the construction of a network with a better average distance per unit cost [57]. This translates to having a network with a better global connectivity with less net wire length used.

Algorithm 2 The Petermann and De Los Rios Model. Source: [57].

procedure CONSTRUCT A "PHYSICALLY REALIZABLE" NETWORK(D, L, nL, α)
    place nodes in D dimensional lattice with lengh L
    add link connecting every node to it's immediate neighbors.
    $q(l) \sim l^{-/alpha}$
    for $l = 1 : nL$ **do**
        pick random node $u \in V$
        pick random node $v \in V \mid d(u, v) \in q(l)$
    end for
end procedure

2.2.5 Multiple Scales in Graphs

Another case is made relating the link-length distribution to it's classification of SW. Kasturirangan [37] claims a graph is SW if and only if it possesses many "scales" and that the Watts-Strogatz model is simply a way to achieve multiple scales. That is, a graph G' is SW relative to a regular graph G with the same number of vertices and edges iff there are many different links lengths present in G' compared to G because the "multiple scales" are alone enough to cause the characteristic high clustering and the low average distance.

In some cases, links within a network have implicit limitations on length (i.e. roads in a transportation network cannot reach across oceans while aircraft cannot be used to fly to a neighbor's house.). In such cases, networks would need a heterogeneous set of links to achieve a greater number of scales causing the network to achieve lower average distance for a given amount of link length. This is addressed in Section 4.2

2.3 Optimization of Graphs

Mathias and Gopal [44] find the power-law length distribution resulting in optimized networks. In this experiment, simulated annealing is used to optimize the placement of links in a network relative to a fitness function that is a linear combination of the network's average distance and cost. We will repeat this type of experiment in the beginning of Section 4.1 using an evolutionary algorithm as the starting point for our experimentation. But we first need to introduce several more concepts pertaining to network optimization as well as the background of what will be our real-world application of network optimization.

2.3.1 The Evolutionary Algorithm

The *Evolutionary Algorithm* (EA) is a nature-inspired stochastic search tool that is used for optimization, constraint satisfaction, and data mining [18]. The EA is also used by [6] pertaining to the construction of NoCs to evolve solutions to the topological mapping problem. The generic evolutionary process calls for randomly building a set of *individuals* (where each individual is the representation of a possible solution) and gradually "evolving" the individuals to reach solutions of better fitness. This process is seen in Algorithm 3 [34], where a *population* is a set of $nPop$ individuals and $rand(0, 1)$ is a random number between 0 and 1. We will cover the particulars of our EA implementation in Chapter 3.2. But first, we need to cover a few more concepts to show how we implement an EA to construct networks.

Algorithm 3 The abstract evolutionary algorithm. Source: [34].

```
1: procedure THE GENERIC EA(popSize)
2:     randomly initialize a population
3:     loop
4:         parent ← randomly pick an individual.
5:         child ← parent
6:         if pMut > rand(0, 1) then mutate(child)
7:         end if
8:         evaluate fitness of child
9:         add child to population
10:        randomly remove an individual from population biased towards individuals with low fitness
11:    end loop
12: end procedure
```

2.3.2 Example Graph Two

For comparison we construct a second reference graph by adding a link between v_0 and v_1. The vertex set remains $V = \{0, 1, 2, 3\}$ and edge set becomes $E = \{\{0, 1\}, \{1, 2\}, \{2, 3\}, \{0, 3\}\}$ as seen in Figure 2.3. The additional two links serving as shortcuts added between nodes 1 and 14 and nodes 7 and 8 increases the cost of the network to 30.32 and decreases the average distance to 2.22.

2.3.3 Aggregate Fitness Function

We now have two example networks - one with a lower cost and the other with a lower average distance. Both can be considered optimal in their own way. To quantitatively compare graphs of the sort, we employ the aggregate fitness function as seen in Equation 2.7,

$$fitness = w * AD + (1 - w) * cG, \tag{2.7}$$

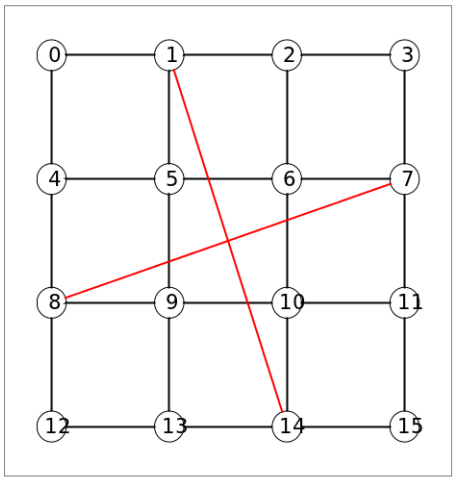


Figure 2.3: Example mesh with shortcuts.

	Cost	Average Distance	Fitness ($w = 0.0$)	Fitness ($w = 0.5$)	Fitness ($w = 1.0$)
Mesh network	2.67	24.00	**24.00**	**13.34**	2.67
Example mesh with shortcuts	2.22	30.32	30.32	16.27	**2.22**

Table 2.1: Fitness comparison of example networks.

where a lower fitness is considered "better" and $w = [0, 1]$ denotes the preference granted to average distance or to cost. Applying this formula to our two example networks where $w = 0.0$ (totally favoring cost), $w = 0.5$ (equally favoring cost and performance), and $w = 1.0$ (totally favoring average distance), as seen in Table 2.1, shows the effect of our choice of preference.

As you see, the network with a lower cost has a better fitness when the weightage favors cost, the network with a smaller average distance shows a better fitness when the weightage is chosen to favor average distance, and in this case it worked out that the mesh network had a better fitness when both cost and performance were considered equal.

2.4 Optimization of Networks on Chip

Networks on Chip have been built through optimization methods before [50, 54]. Ogras et al. uses a local search method to look for "good" placement of wire links according to a deterministic approximation for the critical traffic level. Pande et al. uses simulated annealing to augment a standard wire-based NoC topology with wireless links. We are the first to construct networks using an EA joined with a traffic simulation. Details of this implementation are covered in Chapter 3.

Chapter 3

Framework

The *Complex Network Evolutionary Algorithm (CNEA)* is a stochastic search engine designed work in conjunction with the *Complex Network Framework (CNF)* to construct networks by placing links in a way that optimizes a set of objectives defined by the user. Discussion of the particulars of CNEA's and CNF's construction along with supporting scripts and experimentation procedure are included in this chapter. This chapter may be skipped if the reader already has an understanding of network evaluation and evolutionary algorithms or used as a reference.

3.1 Complex Network Framework

The *Complex Network Framework* (CNF) is a network simulator written in C++ with interfaces built to work as a command-line tool and an API capable of evaluating a network's objective values of cost, average distance, throughput, and critical traffic level and return distributions including including link-length, node-throughput, link-throughput, and shortest-path-length and is documented in Section 3.1. At the time of writing this document, CNF consists of 5 classes, 172 functions, and over $1,000$ lines of code.

3.1.1 Classes of CNF

Recall that a network is defined as a set of *nodes* a set of the node's interconnections known as *links* as shown in Equation 2.1. To model networks with more than one type of link, we need an entity to store characteristics that define the behavior of

a given type. This class that acts as this entity and provides ways to manipulate the included data is called *linkType*. Given the goal of this thesis is to experiment with NoCs with heterogeneous links, we also need to be able to add packets to and manipulate packets with the network. The simulator's class structure, as seen in Figure 3.1, is made to model our definition of a network with heterogeneous links with the possibility of running traffic simulations where each block represents a class and the arrow indicated a dependency.

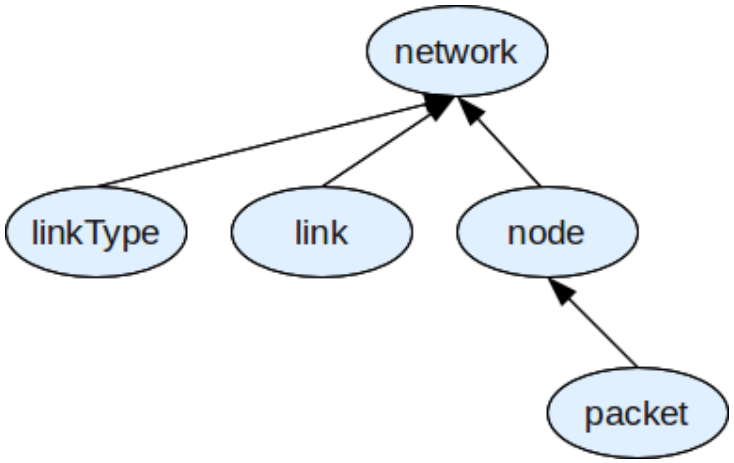


Figure 3.1: Class structure of CNF.

Class: packet

The packet is the basic unit of our traffic simulation. Each packet stores the node index of its current location and its destination node. Counters maintaining the number of hops traveled and the number of clock cycles present on the network are also included. More explanation on the traffic implementation is given in Section 3.1.2.

Class: node

The *node* class represents individual nodes. Each node needs to have variables to store the node's physical location represented with x, y coordinates. To enable traffic simulation, a routing table built as an array of length N and a packet buffer with FIFO protocol are needed. More on traffic implementation in Section 3.1.2.

Class: linkType

The *linkType* class is used as a header to store all relevant information for each type of link used. The relevant information includes the possible range of the length of a link, the attributes of cost, delay, and energy (to be described in the following section), a flag denoting if that linkType is a standard two-endpoint link or a bus (more on this in Section 3.1.1), and a flag that denotes if links are allowed to physically cross over each other.

The method of modeling a linkType's attribute is designed to be the simplest model that is capable of delivering an acceptably accurate representation of that link's behavior. Say, in the most abstract case, the cost of a link could be a constant 1 making the cost of a network equal the number of links. Also abstractly, the desired time it takes for a packet to travel across a link during a traffic simulation could be a constant. This constant has to be modifiable by the user because, say, an experiment may call for using two linkTypes where one is twice as costly as the other. In the most realistic case that we will investigate in this thesis, an NoC's wire link's delay and energy use both follow an exponential as a function of the link's length [60]. This results in the generic attribute function given in Equation 3.1 where a, b, and c are user-defined parameters and A is the attribute value for a given link as a function of that link's length.

$$A(length) = a * length^b + c \tag{3.1}$$

Another possible constraint placed on links is a feasible range. This range is represented by a minimum and maximum distance that is stored in the *linkType* object. The last two pieces needed to sufficiently customize the properties of a link are two flags. The *bus* flag denotes whether or not a given linkType is to act as a set of endpoints in a mutual bus or if that linkType is to act as a set of standard, point-to-point links. More on this is Section 3.1.1. The *overlap* flag denotes if the links within the given type are permitted to overlap one another. In the case of carbon nanotubes [16], links are permitted to cross while photonic links [3] are not.

Class: link

To enable modeling of the standard bidirectional link and of a bus, the definition of "link" needs some abstraction. Each link is defined as an array of endpoints of length nP as seen in Equation 3.2. By setting $nP = 2$, the link is given two endpoints and function as the link originally defined in Section 2.1. If $nP > 2$, then the link functions as a bus where all endpoints are connected directly through the same interconnection fabric. Assignment of a particular type is left to the network class as described in the next section.

$$link_{t,l} = \{endpoint_{t,l,1}, endpoint_{t,l,2}, \ldots, endpoint_{t.l,nP}\} \tag{3.2}$$

Class: network

To store and manipulate a network with heterogeneous links, we need to extend the definition of "network" to allow for more than one type of link. As seen in Equation

3.3, a *network* object is made of a set of *node*s called *nodeVec* (see Equation 3.4), a set of *linkType*s called *linkTypeVec* (see Equation 3.5), and the set of *link*s is defined as a series of subsets of links (see Equation 3.6) where each subset is the list of all links of one particular type (see Equation 3.7).

$$network = \{NodeVec, linkTypeVec, linkVec\} \tag{3.3}$$

$$nodeVec = \{node_1, node_2, \ldots, node_N\} \tag{3.4}$$

$$linkTypeVec = \{linkType_1, linkType_2, \ldots, linkType_{nT}\} \tag{3.5}$$

$$linkVec = \{Type_1, Type_2, \ldots, Type_{nT}\} \tag{3.6}$$

$$Type_t = \{link_{t,1}, link_{t,2}, \ldots, link_{t,nL[t]}\} \tag{3.7}$$

3.1.2 Objective Evaluations

All network evaluations are built into the *network* class. The process of calling these function is to create a network object. While creating the network object, a network is either loaded from file, built from CNEA using a special constructor, or initialized as a preset topology. Then, for every objective value desired, the corresponding objective function must be called.

Cost Algorithm

As defined in Equation 2.4, the cost of a network is the sum of the cost of that network's links. This procedure is given in Algorithm 4.

Algorithm 4 Network cost algorithm.

```
1: procedure RETURN NETWORK COST
2:     cost = 0
3:     for i=1:nT do
4:         for j=1:nL_i do
5:             cost = cost + calculateCost(link_{i,j})
6:         end for
7:     end for
8:     return cost
9: end procedure
```

Average Distance Algorithm

As defined in Equation 2.6, the average distance of a network is the average distance between all i, j node pairs where $i \neq j$. First Dijkstra's algorithm [19] is used to calculate the distance from one node to all other nodes. Repeating this procedure for each node lets us build a matrix holding the distance from each i, j pair as stored in $distMat$. Taking the average value of this matrix gives us the average shortest path [15]. This procedure is given in Algorithm 5, where $dijkstraAlg(i)$ evaluates the distance from node i to all other nodes and returns it as an array of length N.

Traffic Model

Two design constraints governed most of the decisions made in the construction of this model. The first comes from the traffic simulation being used within an optimization loop. With, for example, a single evolution of a 64 node network as done in 4.2 requires processing around 2.4 million networks, the compute time must remain as small as possible. Second, the traffic needs to be simulated on a network that has no constraint on topology. Many routing and flow control methods rely on a prescribed topology - a simplification we are not making. Discussion of our

Algorithm 5 Average distance algorithm. Source: [15].

```
1: procedure POPULATE distMat
2:     distMat = zeros(N, N)
3:     for i=1:N do distMat(i, :) = dijkstraAlg(i)
4:     end for
5: end procedure
6: procedure CALCULATE AVERAGE DISTANCE
7:     AD = 0
8:     for i=1:N do
9:         for j=1:N do
10:            if i! = j then
11:                AD = AD + distMat_{i,j}
12:            end if
13:        end for
14:    end for
15:    AD = AD/(N * (N − 1))
16:    return AD
17: end procedure
```

traffic implementation begins with the unit of information to be simulated.

Information exchange in an NoC can be modeled at one of several different levels [46]. Information is contained in messages. Messages are split up into packets that can be sent independently across the network. A packet can be split into flow control units (called *flits*) which are made of a set of bits. Flits are sent one-after-the-other through the network increasing routing and flow control complexity, but also increasing the amount of traffic a network can handle. In the interest of model simplicity and, thus, the minimization of simulation compute time, we adopt the packet-level of information exchange. We also adopt the synchronous movement of packets through the network as governed by the *clock*.

Routing

Regarding the method of moving packets through the network, we adopt shortest-path routing. The routing table $rTable$, an array of length N where $rTable_i$ denotes a neighbor of current node that is next in the shortest path to destination node i, is populated before a traffic simulation commences. This way, all packets sent from a node refer to that node's $rTable$ for their next step and always take the most direct route to their destination. When the amount of traffic in a network reaches over a certain threshold, there will be scenarios when more than one packet will be directed to use a resource (a node or link) at the same time. To help negate this effect, adaptive routing methods do exist, but to optimize compute time and to aid in the comprehensibility of results, we adopt the oblivious mode of routing where a routing decision is done without consideration to a network's congestion. FIFO buffers are added to the nodes to store any packets that arrived and could not yet be sent out. However, a flow control is needed to regulate traffic the eventuality that buffers start filling under too much traffic.

Flow control regulates the assignment of resources in a network. This can take the form of allocation of resources or contention resolution [17]. Again in the interest of optimizing compute time and comprehensibility, we adopt the method of dropping packets when a node's buffer that is already full receives another packet. The problem with this method is that in the congestion regime (that is, when a network has more traffic than is capable of handling), performance severely degrades [68]. Although this method has been panned as an ineffective method of flow control, recent research finds some value in dropping packets, due to the resulting simplicity of implementation as in [27] and [51], is a feasible and potentially more efficient method.

Traffic

Packets are randomly added to the network according to an injection rate iR. A network's injection rate is the average number of packets added to the network per node per cycle. So, $iR = 1$ would denote an average of one packet added to each node in the network every cycle. Say $N = 100$, a simulation where $iR = 0.01$ means that an average of one packet is injected to the *network* per cycle. A network's throughput TP equals the number of packets reaching their destination per node per cycle. This is seen in Equation 3.8, where *duration* is the number of cycles a simulation is ran and $nPArrived$ is the total number of packets that arrived at their destination during a traffic simulation.

$$TP = nPArrived/(N * duration) \tag{3.8}$$

A traffic simulation is fundamentally a stochastic evaluation due to the complexities associated with selecting sources and destination for each packet as well as the order of placing those packets. This creates slight variations in the returned throughput for each traffic evaluation that can be modeled as noise [32]. Given an 8x8 mesh network, the standard deviation of 30 throughput evaluations is given in Figure 3.2 as a function of the duration of a simulation. We choose durations of $8,000$ clock cycles as a good balance between noise and compute time.

To demonstrate the correlation between injection rate and throughput, we run a traffic simulation on a 6x6 mesh network ($N = 36$) at a series of injection rates and superimpose the bandwidth (number of packets leaving the network per cycle) % of nodes with full buffers. In this network, an injection rate under 0.075 results in all packets being delivered. As the injection rate is increased above 0.075, we

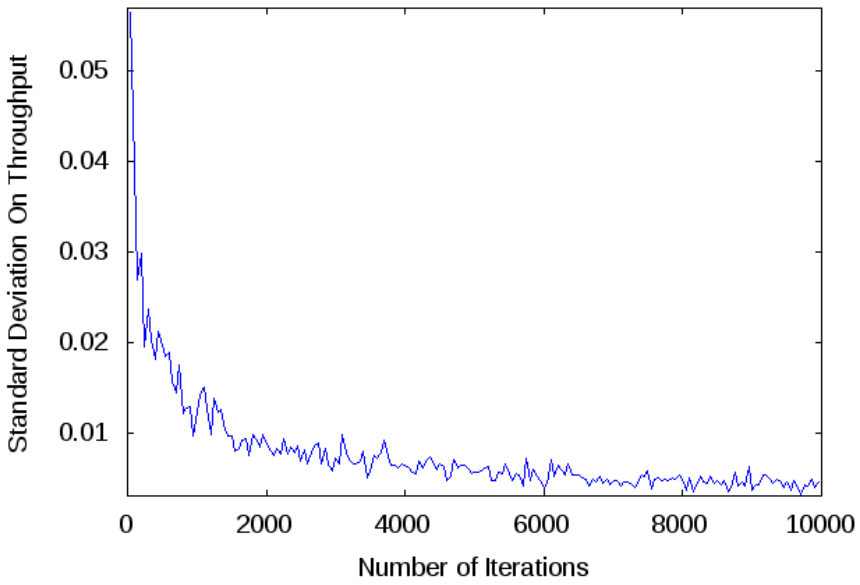


Figure 3.2: Standard deviation of 30 throughput evaluations.

start seeing nodes with full buffers. Due to the flow control of dropping packets when a buffer overfills, additional packets added to a full buffer results in the loss a packet. The higher the injection rate, the higher rate packets are dropped. All "travel" done by a dropped packet results in wasted resources which explains the degradation of bandwidth as the injection rate continues to increase. We refer to this state of having buffers overfill as the *congestion regime* with the threshold marking its start as the *critical traffic level*. We will discuss our method of finding the critical traffic level in Section 3.1.3. But first, we need to discuss other methods of choosing the source and destination of packets.

A major point of NoC traffic simulation accuracy is in the traffic pattern applied in the evaluation. The default traffic pattern in *uniform random*. When a packet is being added to the network, the source node is chosen from the set of all nodes in the network with uniform probability and the destination node is chosen from all of the remaining nodes with a uniform probability. A more realistic traffic

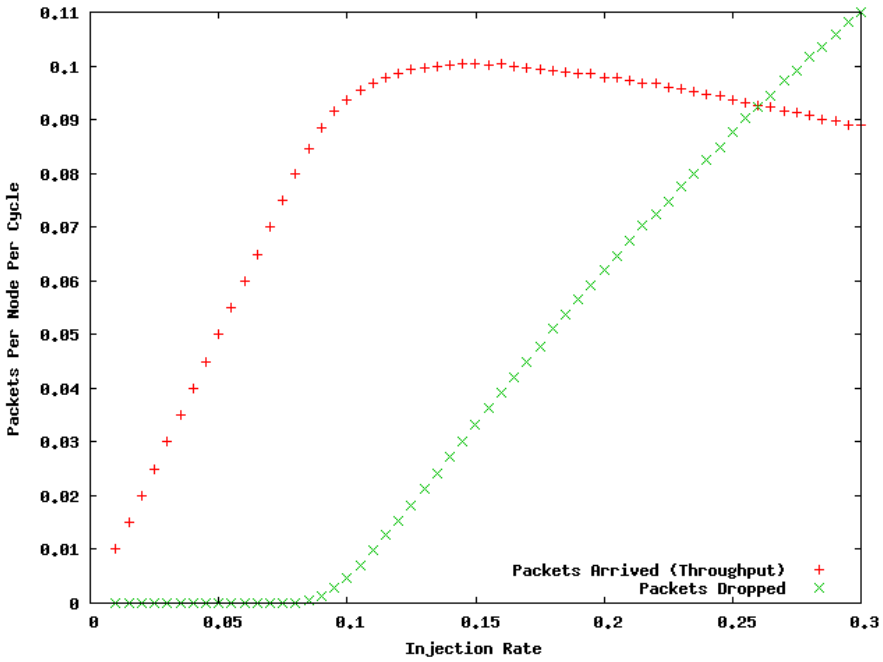


Figure 3.3: Throughput as a function of injection rate.

pattern is known as *hotspot* [58]. The hotspot routing algorithm uniform-randomly selects a source node. With a probability p, one of the hotspot nodes is chosen as the sink. With the probability $1 - p$, one of the non-hotspot nodes are selected. This results in the ability to model networks where specific nodes attract more traffic than other nodes [64]. A generalized way to store traffic patterns is by assigning distinct probabilities for selecting each source/destination pair. This is implemented through the *Traffic Generation Matrix* (TGM) and is stored in the network class. It is a two dimensional array where the first dimension represents the source node and the second dimension represents the sink node. Before running a traffic simulation, CNF converts the TGM into a roulette wheel. During a traffic simulation, the roulette wheel is used to used to pick a source/sink pair according to the prescribed bias.

3.1.3 Critical Traffic Level

This function utilizes the traffic simulation function to derive the critical traffic level (CTL): the injection rate on the threshold marking the start of the congestion regime. An approximation of the critical traffic level as metric of performance for networks in [50]. We, too, use it as a measure of a network's performance and find it through simulation. As already stated, the flow control method of dropping packets makes the throughput of a network degrade as injection rate is increased far into the congestion regime. However, at the CTL, the method of dropping packets does not result in lost performance.

We define the injection rate that results in an average of a half of a percent of packets dropped to be the CTL. The CTL of an $N = 64$ mesh network is 0.100. This is seen in Figure 3.4 by scanning across the injection rates evaluated for the rate closest to the 5% packet-drop-rate. While this algorithm of scanning

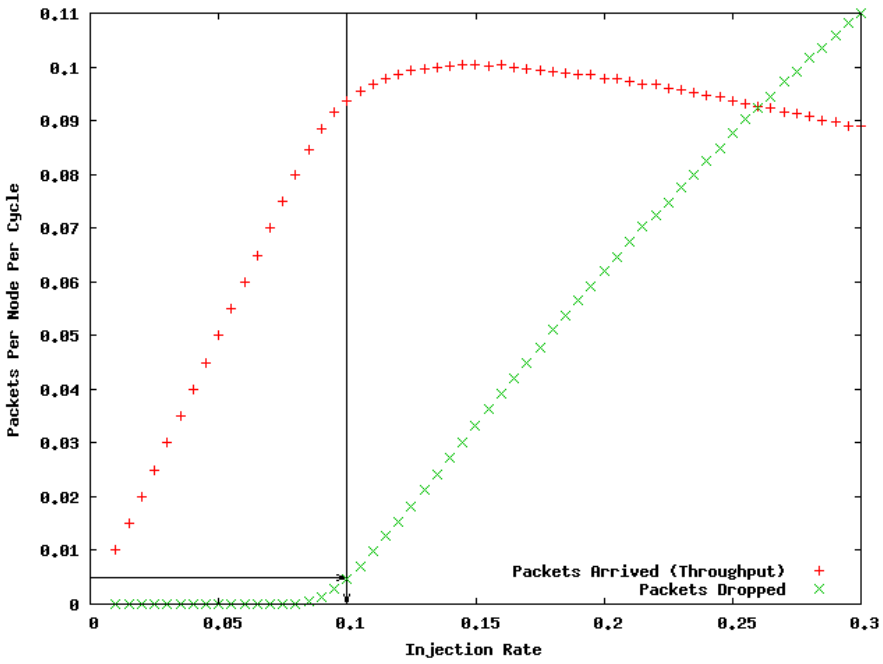


Figure 3.4: Throughput as a function of injection rate with CTL marked.

up through the injection rate in increments of 0.01 works for a visual aid, it is not sufficient for efficient use in a fitness function. The worst-case run time at an increment size (resolution) of r is $1/r$. So, with a resolution of 0.01 as in Figure 3.4, the worst-case runtime requires $1/0.01 = 100$ traffic simulations. Rather, we choose a method of more intelligently searching for the injection rates for the CTL.

This procedure, given in Algorithm 6, can achieve a precision under 0.01 with

Algorithm 6 Critical traffic level algorithm.

```
1: procedure FIND CTL(depth)
2:     low = 0.0
3:     high = 1.0
4:     for d=1:depth do
5:         guess = (high + low)/2
6:         simulate traffic
7:         nD =packet drop rate
8:         error = (high-low)*0.05
9:         if nD > 0.005 then high = guess + error
10:        elselow = guess - error
11:        end if
12:    end for
13:    guess = (high + low)/2
14: end procedure
```

only 7 traffic simulations. The CTL must be between an injection rate of 0 and 1 so, to begin, the first guess is half-way between the known upper and lower limits; at 0.5. Depending on if the number of packets dropped is above or below the 0.5%, the next guess is made within the $[0, 0.5]$ or $[0.5, 1]$ range respectively. Repeat this procedure of guessing and evaluating the number of packets dropped until an acceptable precision is achieved. To account for the inevitability of a *guess* being right next to the CTL and receiving in improper classification, we make each new guess include the injection rates within 5% of the last *guess* as seen implemented with the *error* variable.

3.2 Complex Network Evolutionary Algorithm

The *Complex Network Evolutionary Algorithm (CNEA)* is an evolutionary algorithm constructed from the *ParadisEO* [42] white-box framework. The *ParadisEO* framework handles the evolution process while leaving the encoding of solutions and interfacing with those solutions to the user. Particularly, the representation (definition of, initialization and deletion) and the representation-specific evolutionary functions (mutation, crossover, and fitness evaluation) are left to the user. It has two interfaces. First, a command line interface that supplies the second and primary interface is the "evolutionSeedN" file supplying the majority of the options and constraints for consistent and intuitive control over experimentation. At the time of writing this document, additions of the ParadisEO API to construct CNEA consists of 17 functions, and also over $1,000$ lines of code.

The top-level implementation of CNEA is shown in in Algorithm 7. *Topology* represents the placement of the nodes and any constraints on link placement required for a particular topology. The parameter w denotes the weightage assigned to the objectives in the fitness function. The number of individuals in a population and the number of generations an evolution is ran for are denoted by *nPop* and *nGens*. The *crossover*, *mutate*, and *selection* function are discussed in Section 3.2.2. *Evaluate* calls the fitness function as discussed in Section 3.2.3 on each individual in the given population.

3.2.1 Representation

Similar to the representation of a network in CNF as given in Section 3.1.1, a network is built from nodes, links, and the linkType objects. In this case, representation of the nodes is done using a N-by-2 array holding the x, y coordinates

Algorithm 7 The Complex Network Evolutionary Algorithm.

```
1: procedure POPULATE distMat(topology, w, nGens, nPop)
2:     Randomly initialize population P
3:     for t=1:nGens do
4:         Evaluate P(t)
5:         P' = crossover(P(t))
6:         P' = mutate(P')
7:         Evaluate P'
8:         P(t+1) = selection(P(t),P')
9:     end for
10:    return best individual from P(t+1)
11: end procedure
```

for each node. The links in CNF were stored in an array. However, with the expectation to frequently change the number of links, it is implemented as a vector in CNEA. An addition to the representation is a constraint on the number of links the EA is allowed to place for each linkType. These are also stored as vectors.

3.2.2 Evolutionary Functions

Crossover

Crossover points are chosen and standard 2-point crossover is used on the set of each type of link.

Mutation

Two types of mutations are used. The first randomly selects a linkType and changes the number of links of that type. This is done by uniform randomly choosing between adding and removing a link. If adding, a linkType is randomly chosen and a link of that linkType is randomly initialized and appended to the end of the link vector. If removing a link, a link is uniform randomly chosen and removed from the vector of links. The other type of mutation moves a current link.

Selection

After a "child" population is generated from the "parents" and evaluated, the next generation of individuals is chosen using the deterministic tournament.

3.2.3 Fitness Function

Fitness is the quantification of the quality of an individual. Of the various types of multiobjective evolutionary algorithm [67], the aggregate function [40] is used as a simple method to quantitatively set the preference of objectives used in evolution. We use the special case of having two objectives as shown in Section 2.3.3.

In addition to the objectives and the weightage, we use fitness penalties to impose two constraints onto all evolutions. The first constraint is to make the networks connected. The second is to prevent overlaps in the linkTypes that are not physically able to overlap (in this thesis, the only link to follow that constraint is the photonic link used in Experiment 5). If any given linkType prohibits overlap, then the number of overlaps of that linkType is scaled to be one order of magnitude larger than weighted objectives and used as a fitness penalty. If the network is not connected, then the percentage of nodes that are not connected is scaled to a second order of magnitude larger than the weighted objectives.

3.2.4 Normalization of Objectives

One more issue needs to be addressed regarding our application of the aggregate function. The objectives of a network evaluation tend to frequently span different orders of magnitude. To help alleviate this effect, we normalize each objective using that objective's worst-case value before plugging them into the aggregate fitness function. For this thesis' purposes, the worst-case scenario for cost is the

N	nL	cost	L
16	120	257.02	1.00
25	300	796.11	1.00
36	630	1996.04	1.00
49	1176	4333.16	1.00
64	2016	8471.48	1.00
81	3240	15294.04	1.00
100	4950	25934.12	1.00
121	7260	41806.69	1.00
144	10296	64639.44	1.00
169	14196	96506.32	1.00
196	19110	139853.32	1.00
225	25200	197532.42	1.00

Table 3.1: Statistics of a complete graph: worst-case in cost.

complete graph. CNF is used to construct and evaluate 4x4, 5x5, ..., and 15x15 networks with unit-spaces between each node as shown in Table 3.1 where N is the number of nodes in the network, nL is the number of links, *cost* is the network's cost, and L is the average distance.

Worst-case scenario for a network's average distance is a little more complicated. The graph needs to be connected or else the average distance isn't valid. The graph needs to be a tree (as few links as possible to make it connected) or else it won't be a worst-case scenario. Then, the worst-case topology for a tree's average distance is to have all links arranged in a single path. The cheapest way to do so is to "draw" the path in a way that each link is only one unit long. Again using CNF, these graphs are constructed and the characteristics saved as seen in Table 3.2 where N is the number of nodes in the network, nL is the number of links, *cost* is the network's cost, and L is the average distance. Throughput and the critical traffic level are by definition within the unit interval so they needs no normalization.

N	nL	cost	L
16	15	15.00	5.66
25	24	24.00	8.66
36	35	35.00	12.33
49	48	48.00	16.66
64	63	63.00	21.66
81	80	80.00	27.33
100	99	99.00	33.66
121	120	120.00	40.66
144	143	143.00	48.33
169	168	168.00	56.66
196	195	195.00	65.66
225	224	224.00	75.33

Table 3.2: Statistics of the single-path graph: worst-case average distance.

3.2.5 Convergence

EAs are notorious for being good at exploration but being bad at exploitation. In other words, the EA can find solutions that are better than random quite quickly compared to other optimization methods, but aren't that efficient at evolving those "good" solutions into "great" solutions. Deciding how much computation time to invest into returning solutions is entirely based off the goal of a given evolution. Where a low fitness is considered better, a convergence curve denoting the fitness of the best individual for each generation is given in Figure 3.5. We see that quick improvements to fitness happen for the first few hundred generations. After the next $1,000$ generations or so, improvement slows down. Finally, around generation $1,500$, improvement appears to stop. For our purposes, we want to find solutions that are at least close to optimal. While running many evolutions for only a few hundred generations each would still reveal trends, the curves showing those trends would likely be irregular making them good for a first impression of what a particular evolution will return but would prove inadequate for this thesis' results.

To say an evolution is "converged," we mean that an evolution convergence curve shows no sign of further improvement.

The procedure followed when choosing the size of the evolutions for a given experiment (the population size and the number of generations) starts with a rough guess. Using a population size and a generation count based of either arbitrary choice or what seemed to work on a past experiment, several test evolutions are put on the servers. Once the few sample evolutions complete, their convergence curves are graphed as in Figures 3.5, 3.6, and 3.7 and their successful convergence can be determined. Figure 3.5 shows the first sample convergence curve. In this

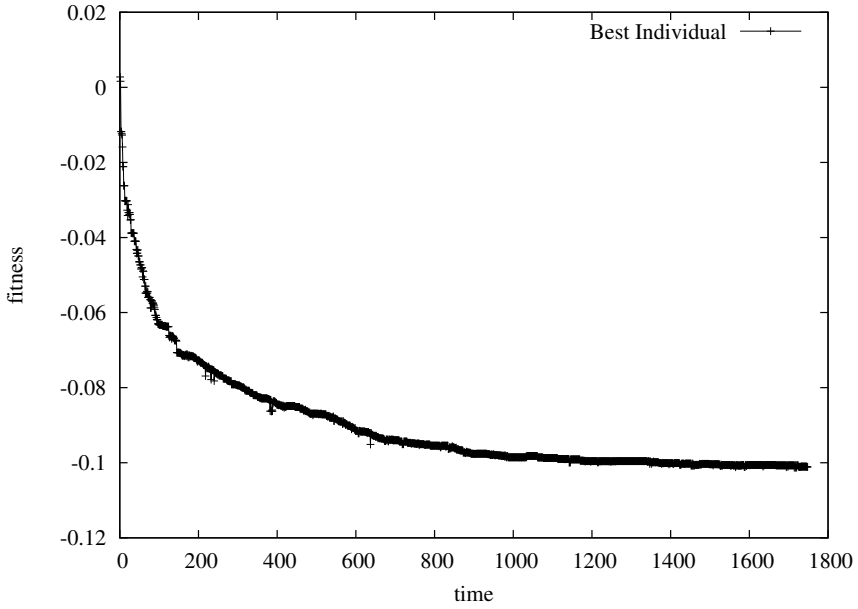


Figure 3.5: Sample convergence curve 1.

evolution, it is seen that the last few hundred generations resulted in no significant improvement to fitness. It is then a somewhat safe assumption that the population has converged to a solution that is as good as this evolution will ever get. We see the second convergence curve example in Figure 3.6 where the fitness still appears to still be improving slightly with each generation. As a more extreme case, the

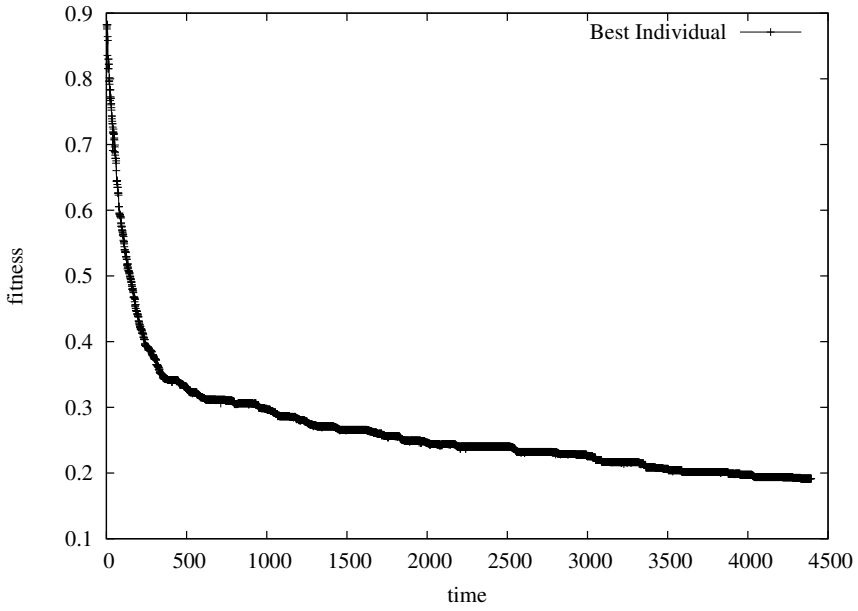


Figure 3.6: Sample convergence curve 2.

third convergence example given in Figure 3.7 is quite clearly still improving each generation and would benefit from being ran for more generations. Again, choosing if these sorts of convergence curves are acceptable is totally up to the experimenter and their goals. For the purposes of this thesis, given that theses three curves were pulled from the same experiment and were a representative set of what the other evolutions looked like, we would rerun the evolution with a larger population and/or more generations.

One other concern of ensuring consistent convergence comes from the uncertainty associated with a stochastic search. In Figure 3.5, it looks like the evolution would find no better solution. Yet, it is possible that within a few more generations, some greatly superior individual would be found and the population would "spike" downwards. Without some meta-level knowledge of what the fitness of the best individual would be, this must always be deemed as possible. Any sort of "jumps" in fitness after convergence appears to be almost complete is seen as a sign that the

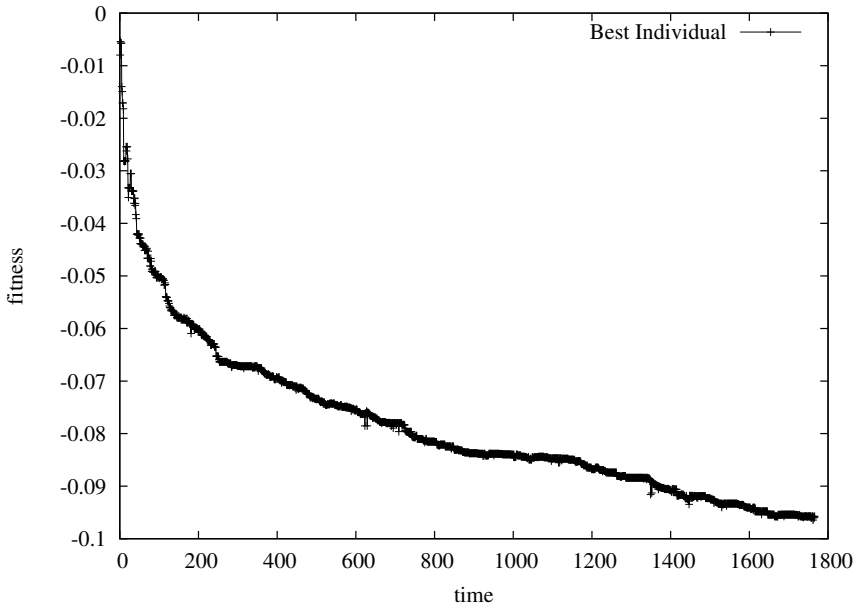


Figure 3.7: Sample convergence curve 3.

search breadth of the evolution is not adequate for the search space that exists. As a response, increasing the population size "smoothes out" the convergence curves by more adequately exploring the available space. Thus, if a significant number of evolutions within an experiment show irregularities in the convergence curve, then the experiment is re-ran using a larger population size.

3.3 On Performance

Regarding the $linkVec$ representation, given a network with the normal point-to-point links, the memory required to store a vector of links directly would scale with the number of links following the function $2 * nLinks$. Given the adjacency matrix representation (where an $N \times N$ matrix stores where position i, j in the matrix denotes if there is a connection from node i to node j), the memory required to store the links of a network becomes N^2. At the time of initially building this

framework, it was believed that memory would be the CNEA's bottleneck, so the representation chosen is the link vector. This representation also had the advantage of naturally extending to allow redundant links.

It worked out that the evolutions utilize about 0.1% of the available memory and 100% of processor availability. As such, another potential optimization of this framework would be to change this fundamental representation to the adjacency matrix. Extending the adjacency matrix to, rather than being a bool holding a 0 or a 1 representing a link or no-link, being an int that denotes the *number* of links connecting the two nodes would allow for multiple connection. Then, to model multiple linkTypes, you could simply have multiple adjacency matrices.

3.4 How To Run an Experiment

We now walk through the steps needed to run an evolution using CNEA. We start with a minimal example to make certain everything is properly installed and running. We then proceed to a more complex example evolving networks with two linkTypes using traffic to illustrate the more complex features of CNEA's interface.

3.4.1 A Minimal Example

For instructions on the acquisition and installation of CNEA, see tech report "HowToBuildCNEA." Once CNEA is installed, several steps are needed to set up an evolution. In a terminal, "cd" into CNEA's "src" directory and run "make" to build the code. If you want to use a different directory to store the results of the evolution, then copy the CNEA executable into that directory.

From here, the minimum needed to run an evolution is to create the "results"

folder in the same directory as CNEA, create a folder named "0" within the "results" folder, generate the "evolutionSeed0" file as given in Figure 3.8, and then run CNEA within a terminal using no arguments. The default values of CNEA will run a small evolution on a network size set by the seed file and optimize to an equal preference of cost and average distance.

The final network saved at the end of the evolution within the "results/0" folder as a .net file and a .obj holding the network data and objective data respectively. CNEA will generate statistics of the evolution including the number of individuals processed, and the best, average, and standard deviation of the population each generation. This output is directed to the terminal that it was ran on. To redirect this output, using the command

```
$ ./CNEA > results/0.ea $
```

will direct the statistics output to the file "0.ea" within the "results" folder and have the instance of CNEA run in the background.

To add the normalization of objectives, change the objectives' weights, modify the evolution parameters, or change other settings of the evolution, see Section 3.4.2 for an explanation of the command line options. To modify the number of nodes or their topology, add other linkTypes, change the parameters of links, or change the number of links available to the evolution, see Section 3.4.3 for an explanation of the evolution seed file. For a walk-through of conducting a more complex evolution, see Section 3.4.4.

3.4.2 CNEA Command Line Options

The command line options are implemented through *ParadisEO*'s parser class. Run the command

Command Name	Default Value	Description
cC	1.0	cost coefficient
tC	1.0	throughput coefficient
eC	1.0	energy coefficient
aC	1.0	average distance coefficient
rC	1.0	CTL coefficient

Table 3.3: CNEA command-line options for aggregate function coefficients.

Command Name	Default Value	Description
w1	0.5	cost weight
w2	0.0	throughput weight
w3	0.0	energy weight
w4	0.5	average distance weight
w5	0.0	CTL weight

Table 3.4: CNEA command-line options for aggregate function weights.

```
$ ./CNEA --help
```

for the full list of command line options, The options used for the experiments within this thesis are discussed here. The format of

```
$ ./CNEA [options]
```

is used to set the options, where each option follows

```
--name=[value]
```

as the format.

The parameters of the aggregate function consist of coefficients and weights. The coefficients are used to normalize the objectives to be within the unit interval and are given in Table 3.3. The weights are used to set priority to their respective objective and are given in Table 3.4. The evolution parameters are given in Table 3.5. The evolution runs for a fixed number of generations as set with *maxGen*.

Command Name	Default Value	Description
maxGen	100	number of generations in evolution
popSize	20	number of individuals in the population
pCross	0.6	probability of crossover
pMut	0.1	probability of mutation

Table 3.5: CNEA command-line options for the evolution parameters.

Command Name	Default Value	Description
seedIndex	0	selects which evolutionSeed file to use
printK	100	print best network every K generations
evolutionIndex	0	unique index assigned to a given instance of CNEA
iR	0	toggles activation of traffic simulation and sets injection rate

Table 3.6: Other CNEA command-line options.

The number of individuals within a population is set with *popSize*. The crossover and mutation rates are set with *pCross* and *pMut* respectively.

The other settings used in running the experiments of this thesis are given in Table 3.6. The name of the evolution seed file is "evolutionSeed$N$," where N is an integer set by the *seedIndex* option. If no *seedIndex* option is given, the default seed file is "evolutionSeed0." Although most experiments only require changes in the command line options, a systematic way of selecting different evolution seed file is still needed for some experiments. For example, in Experiment 1, the Pareto front shown in Figure 4.3 was found by running many evolutions for a range of cost and average distance weights. Running such an experiment only requires changing the evolution index and the weights on cost and average distance which are settable on the command line requiring only one evolution seed file. However, again in Experiment 1, evaluating the scalability of the average distance of evolved

networks as seen in Figure 4.5 requires running evolutions with varying network sizes. The network size (number of nodes and their topology) is only settable in the evolution seed file and, thus, requires one seed file for each network size.

CNEA saves the best individual and the objective values every K generations. The *printK* option is used to set K. Saving the best individual every generation ($K = 1$) is useful for how networks "grow" as the evolution progresses. Independent of K, the best individual is saved at the last generation of an evolution. So, setting K to be greater than the number of generations of an evolution will result in saving the best individual from only the final generation. This conserves hard disk space and is sufficient if you are only concerned with looking at the final individual. For most of the experiments within this thesis, K was left at its default of 100 as a good balance of conserving hard disk space but still being able to analyze the population early in the evolution's progress.

As discussed in Section 3.2.5, EAs are good at exploration but not exploitation, meaning that you will find a "good" solution quickly but take a long time to find a slightly better solution. For our purposes, this translates to seeing trends (e.g., the application of the costly links only when preference is set to performance in Figure 5.2 in Experiment 3) within the first few hundred generations, but needing thousands of generations for the noise in the trends to become negligible. There were several cases where anomalies were observed in the trends within the first few hundred generations, which resulted in finding an error in the setup of an experiment.

As described back in Section 3.4.5, the evolution index is the unique integer label assigned to an evolution to differentiate between it and other evolutions within a version of an experiment. All individuals saved during an evolution are

saved within the "results" folder (as discussed in Section 3.4.5) within a folder with the index value as its name. CNEA itself does not automatically generate the folder within "results," so you need to either manually create the folders to match the evolution indices you plan on running or use a script to run CNEA that generates the necessary folder before an evolution is ran.

The injection rate of 0 deactivates traffic simulation. An injection rate within the range of $(0, 1]$ runs the throughput evaluation at the given injection rate. An injection rate of -1 runs the CTL evaluation. In both the throughput and CTL evaluations, the traffic patterns used are set in the seed file. If multiple patterns are given, then one traffic simulation for each pattern is ran and the average of all simulations is calculated and returned.

3.4.3 The Evolution Seed File

The evolution seed file is designed to store the information about a particular network's topology that is likely not to change between instances in a given experiment. This information includes the number of nodes, the node locations, the linkType parameters, and the constraints on the links themselves. The seed file follows the generic format of first giving node information, then linkType information, then the link information. The pound (#) sign acts as a spacer within the file structure to allow for some redundancy and, thus, allow error checking within the evolution seed file parser. An example of a very simple seed file follows.

A Simple Example

The default evolution seed index in CNEA is 0, making the default evolution seed filename "evolutionSeed0" as used in this example. Within the file, the first word

evolutionSeed0
mesh 5 4 # 1 # # limit 0 50 # #

Figure 3.8: A simple evolution seed example.

"mesh" tells CNEA to arrange nodes in an $n \times n$ lattice. It expects "mesh" to be immediately followed by an int and then a double. The int sets the value of n. The double sets the euclidean width of the whole network with evenly spaced nodes. For example, "mesh 4 1" creates a 4×4 network mapped to fit evenly within a 1 unit $\times$ 1 unit space and "mesh 10 9" creates a 10×10 network mapped to a 9 units $\times$ 9 units resulting in a unit of space between each node. So, this example seed file creates a 5×5 network with one unit between each node. The node-based information is done and a "#" is used as a divider within the file.

The linkType information comes next, starting with the number of linkTypes. For this simple example, we are using one type of link so the first entry in the linkType section is a 1. The next line holds another "#" as a divider. Now the link characteristics can be manually set. Any characteristic not set in the seed file are left at their default values. The default linkType's characteristics are designed to make cost equal the length of a link, have unit delay and unit energy consumption, and have no constraint on length. Since the default values are acceptable for this example and there is only one linkType, we use another "#" and proceed to the link data. A more thorough explanation of setting the link characteristics is given in the next example.

CNEA needs to be told how to handle initialization and evolution of the links in each linkType. Any linkType can be used to construct a preset topology and left unmodified during an evolution or randomly initialized and marked for evolution. At the time of writing this thesis, the only preset topology is the mesh network. As we will see in the next example, this option can be selected by using the word "mesh" at this point in the seed file and CNEA will know to construct a mesh network using the given linkType. Alternatively, CNEA can be used to select the number of links and the links' placement. To do so, use the word "limit" followed by two integers – the lower and upper limit of links – is used. For example, "limit 25 25" will tell CNEA to place exactly 25 of the given linkType. As we see in "evolutionSeed0," we are telling CNEA to place between 0 and 50 links of the given type. Once this section is concluded, another "#" is used as a divider.

The last part of the seed file is used to set which traffic pattern to use in an evolution. In this example, we choose to not use a traffic simulation to determine a network's fitness (as is done in Chapter 4's experiments).

Another Example

We now discuss a more complex example that involves two linkTypes in a traffic-based optimization. As in Experiment 5, we will be adding links to a mesh network that is optimized using hotspot traffic.

Nodes are again arranged in a 5×5 lattice mapped onto a 4×4 unit square. This time, there are two type of links. For the first type, the default characteristics are ok except for the name and cost. To change the value of any characteristic, you give the name of the characteristic followed by the value(s) expected. To change the name of the first linkType, "name" is first given followed by a string that will

evolutionSeed1
mesh 5 4 # 2 # name type1 cost 0.00 0.00 0.00 # name type2 range 0.00 0.00 cost 0.00 0.00 1.00 delay 0.00 0.00 1.00 energy 0.00 0.00 1.00 bus 0 allowOverlap 1 # mesh limit 0 15 # hotspot 0.10 2 6 18 #

Figure 3.9: A more complex evolution seed example.

be saved as the linkType's name. On the next line, the cost of this linkType is set to always be 0. In this example, we are using the first linkType to construct a mesh and don't want to consider the constant cost of constructing that mesh in the network's fitness. A "#" is then given to mark the end of that linkType's characteristic settings.

In the next linkType, for discussion's sake, we manually set all possible characteristics including those that are set to match the default value. In the second linkType, the line "name type2," as in the first linkType, sets the name. The next line, "range 0.00 0.00" sets the lower and upper limit on the constraint of the link's length. Setting and upper and lower limit of 0 is a special case that tells CNEA to not constrain the link's length. For this example seed file, we are leaving the length of the links unconstrained. For another example, "range 0.00 5.00" would only allow links of that type to be from 0.0 to 5.0 units long.

Using one of the keywords "cost," "delay," or "energy" sets the values of a, b, and c of the respective function as described in Equation 3.1. In this example, we change the cost from being equal to the length of a link (so the cost of a network is the sum of the link lengths) to being equal to 1 (so the cost of a network equals the number of links). The following two options are flags set or cleared by using the keyword followed by a 1 or a 0 respectively. The keyword "bus" says whether or not a given linkType should be treated like a bus where, instead of placing links between nodes, the EA places termination points of a single bus (as is done with the mmWave link in Experiment 5). If "bus" is false, then the links are treated as the standard, two-endpoint, bidirectional link that is used in all other parts of the thesis. The keyword "allowOverlap," when set, tells the EA that a given linkType's links are allowed to overlap one another. If this is false – if links of a

given type are not allowed to overlap – as happens with the photonic linkType in Experiment 5, then there is a fitness penalty applied for each faulty overlap of the given linkType applied during an evolution. This section of the seed file is again terminated with "#."

For both types of links, the seed file declares what to do with the respective linkType. In this example, type1 is used to construct a mesh. CNEA will not add, move, or remove any of the links of this type because it is set to a fixed topology. Regarding the second linkType, type2, CNEA is instructed to optimize the number of links withing the range of $[0, 15]$ and optimize the placement of those links.

The final section selects which traffic patterns to run. The available traffic patterns are uniform random, hotspot, and a special case. To run uniform random, no additional parameters are needed, so the keyword "uniform" is enough. To run hotspot, use the keyword "hotspot" followed by the bias to the hotspots (a double in the range [0,1], the number of hotspots, and the node indices that are to be hotspots. The special case (used for the transpose traffic pattern in Experiment 5) reads an $N \times N$ matrix from file that is the probability of choosing a given source / destination pair. To run the special case, use the keyword "file" followed by a space and the relative location and filename of the special case traffic pattern.

Every traffic pattern that is entered in the seed file before the next "#" separated by newlines are ran separately and averaged together. For example, if both uniform and hotspot traffic patterns were supplied in the seed file, then, every fitness evaluation, CNEA would run a traffic simulation using a uniform pattern and run another simulation using the hotspot pattern and return an average of the two for use in the fitness function. In our example, we use only the hotspot traffic pattern with a 10% bias using two hotspots: one at node 6 and the other at node

18.

3.4.4 Example CNEA Implementation

For this example evolution, we need to setup a folder with the appropriate files. Within the same directory, add a copy of the CNEA executable, create a results folder, and add a copy of "evolutionSeed1" from Figure 3.9. In a terminal, "cd" into the project's directory. We now need to construct the command's options. First, we are using seed index 1 rather than the default 0, so we need to use the option "–seedIndex=1." For this experiment, we will be evolving to cost and CTL with equal preference. The default weight on cost is 0.5, so we can leave that alone. However, we need to set the weight on average distance to 0.0 and set the weight on CTL to 0.5. To do so, we use the options "–w4=0 –w5=0.5." Regarding the normalization of the objectives, CTL is, by definition, within $[0, 1]$ and needs no scaling. Cost in this example is a function of the number of links added. Since the upper limit on the number of links CNEA is allowed to place is 15, the upper limit on cost is 15. The scaling coefficient on cost is set to $1/15 = 0.0667$ using the option "–cC=0.0667." We also need to tell CNEA to run the CTL evaluation by setting the injection rate to -1 using the option "–iR=-1." Finally, we want to let this run in the background while saving the statistics to a file in the results folder by ending the command with "> results/0.ea &."

To make a place to save the resulting networks, create a folder titled "0" within the results folder to match the default evolution index. We can now run CNEA using the following command.

```
$ ./CNEA --seedIndex=1 --w4=0 --w5=0.5
    --cC=0.0667 --iR=-1  > results/0.ea &
```

To run a second evolution with the same settings, you need to to create another folder for results to fit in. Call it "1" to match the next available evolution index. Additionally, set the evolution index to 1 when calling CNEA the second time using the option "–evolutionIndex=1

```
$ ./CNEA --seedIndex=1 --w4=0 --w5=0.5
    --cC=0.0667 --iR=-1 --evolutionIndex=1 > results/1.ea
```

Repeat this process for as many iterations as you want. For the experiments contained in this thesis, I wrote a script called "runAlg.sh" that automatically detected the next available evolution index, created the necessary folder within "results," set the evolution index automatically, and managed the file where CNEA's output is directed.

3.4.5 File Structure

To organize the iterative running of experiments as well as the analysis of those experiments, I standardized the file structure. The storage space is split into separate experiment folders Ex1, Ex2, ..., where the evolutions within a given experiment folder are themed towards investigating one particular relationship. Each experiment folder is split into version folders v1, v2, ..., where each version represents one particular attempt at exploring the given relationship. For example, the experiment could be the investigation of the link length distribution as a result of optimization to cost and average distance as done in Experiment 1. Iterating to another version could be due to anything from changing the size of the network to changing how cost is defined. However, changing the evolution to include traffic simulation could warrant iterating to a new experiment. The file structure is shown in Figure 3.10, where each block represents a folder. The development of

this process came from a high demand for a quick experiment turnaround rate while having essentially no limit on hard disk space.

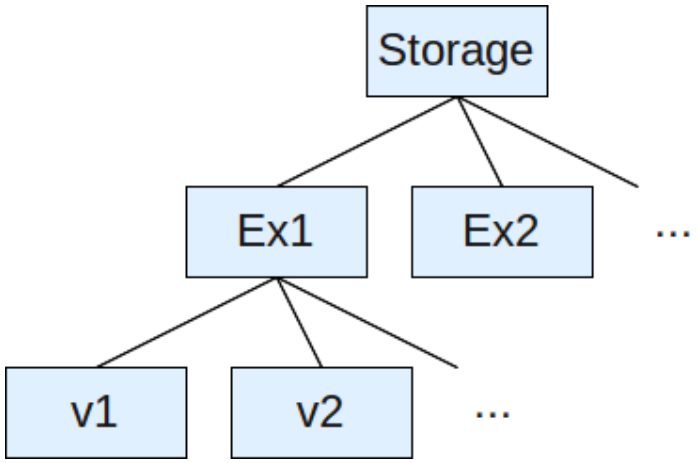


Figure 3.10: File structure of an experiment.

The file organization of a particular experiment is given in Figure 3.11. Each version folder holds all files needed to run the evolution and a results folder. The contents of this folder matches what was called for in Section 3.4.4.

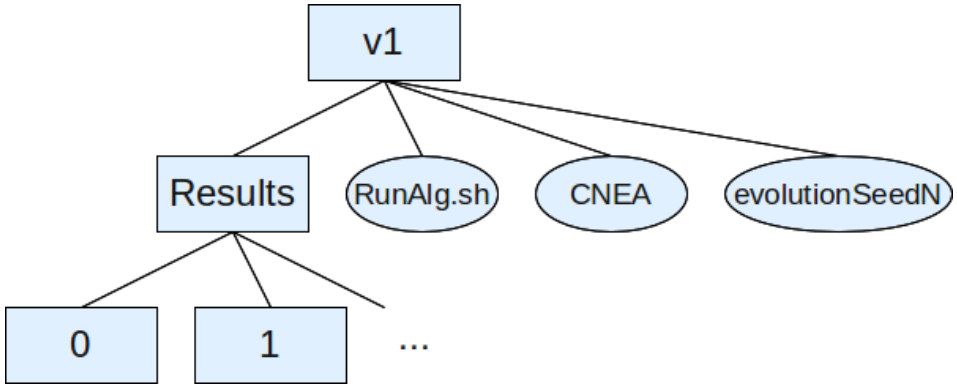


Figure 3.11: File structure of a version.

3.4.6 Supporting Scripts

A dedicated "runAlg.sh" script is written for each experiment. All options passed to CNEA that are constant across an experiment are hardcoded. All options that

may change between evolutions are passed through the script's command line. The result is being able to run additional iterations of a particular experiment passing only the relevant values and all else takes care of itself. Additional specialized scripts were written to match the number of processes to the available resources.

To aid in the data compilation process, a folder titled "tools" is added to each experiment directory. Scripts written in octave are constructed to perform the function of reading the standardized network files, generating network plots as seen throughout this thesis, compiling distributions, and compiling the objective data. Another script is written to scan through all versions within an experiment folder for all evolutions with results and run the data collection/analysis scripts.

Chapter 4

Optimization of Complex Networks

4.1 Experiment 1 – Abstact Network Optimization

We start our investigation with evolving complex networks using abstract links to establish our methodology by recreating the experiment done in [44]. Cost and average distance are chosen as simple and complimentary objectives. We adopt the sum of interconnect lengths as the cost of a network [25]. The total interconnect length also serves as a metric of complexity [39, 65]. The average distance is adopted as the measure of a network's performance [36] and also serves a "best-case" approximation for traffic simulation [13]. Through this experiment, we will draw a relation between evolving a network according to cost and average distance and the resultant network's link length distribution.

Nodes are arranged in an n-by-n grid where each node is a distance of 1 unit away from its neighboring node as seen in Figure 4.1. The parameter of the link characteristics to be used in evolution are shown in Table 4.1. The evolutions are set to run for $6,000$ generations with a population size of 150. This procedure is repeated for $w = 0.1$ to $w = 0.9$ with a resolution of 0.1 at repeated for 20 iterations at each w at network sizes of 4x4, 5x5, ..., and 15x15. For each w evaluated, the average of each of the objectives (cost and average distance) are calculated.

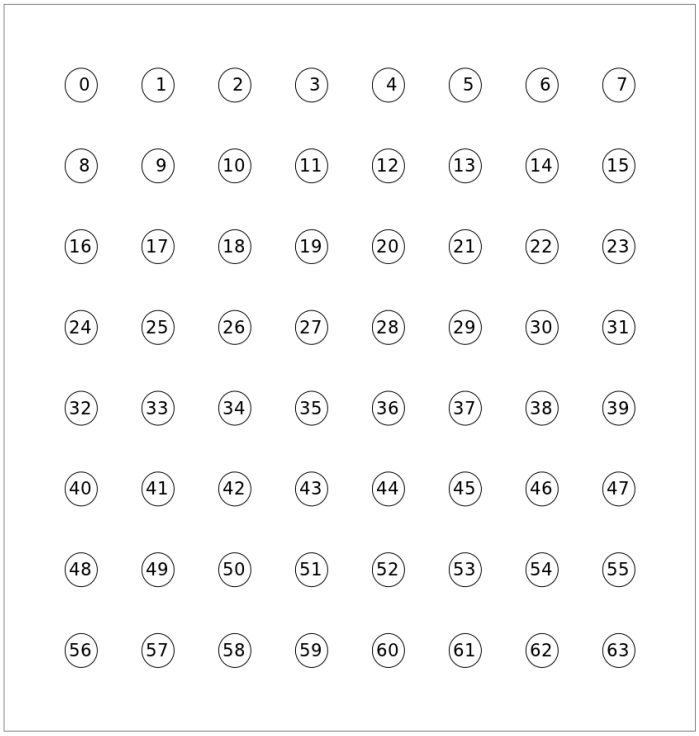


Figure 4.1: The 8x8 lattice.

4.1.1 Results

Starting with the 8x8 case, two of the evolved networks can be seen in Figure 4.2. The average objective values for the three of the w values compared to the mesh network is shown in Table 4.2. With the weightage set to 0.2 (which favors cost), CNEA returned a network that had a low cost but a high average distance (compared to the other two networks). Conversely, the result of the evolution with a high weightage (favoring performance) returned a network with a low average distance and a high cost. Setting w to 0.5 (asking the EA to balance cost and performance) returned a network in the middle relative to both objectives.

Link Type	Cost
Type 1	length(link)

Table 4.1: Parameters of abstract link.

network	Avg.Distance	Cost
mesh	5.33	112.00
$w = 0.20$	3.7725	68.4927
$w = 0.50$	2.8315	103.2886
$w = 0.80$	2.1236	207.7332

Table 4.2: Objective values for mesh and $w = 0.2, 0.5, 0.8$. $N = 64$.

Another way to represent this data is to plot the networks within objective space – that is, the space where each objective represents one dimension and each network is mapped to a location in the objective space representative of its objective values. Within this space is a set of points S that can actually be achieved by a network. For example, the goal is to find a network with minimal cost and minimal average distance, but a network with a cost of 2 and an average distance of 1 is not possible because, with only two links with a length of 1 or one link with a length of 2, there's no way to connect the network let alone create direct paths from ever node to every other node. Similarly, a 8x8 network with an average distance of 1.0 is possible if the cost is 8471.48 or greater (as shown in Table 3.1, that would be the complete network).

The average objective values of the evolutions at the various w for the 8x8 network are shown in Figure 4.3. It can be seen that, by accepting a higher cost, a lower average distance is possible. Improving a poor performing network (say, the networks resulting from evolutions where $w = 0.2$) would cost relatively little while the expense associated with "upgrading" from a high-performing network (say, $w = 0.8$) to an even higher performance network would cost significantly

more. At the cost of an 8x8 mesh network (with an average distance of about 5.3) the EA was able to construct a network with an average distance of about 2.5. The $w = 0.3$ evolutions with an average distance of 3.3429 is near to the performance of the random networks showing an average distance of 3.516. However, the evolved networks at $w = 0.3$ with a cost of 77.22 only used 16.43% of the link length needed to construct the random network with a cost of 469.87.

In an effort to understand how the networks were able to achieve such a more efficient use of links, we look at the length distribution of three of the resulting networks and the mesh network as seen in Figure 4.4. Each optimized network returned a link length distribution that appears exponential showing a high amount of short-ranged links with a small amount of long ranged links. As a greater preference is set to performance, more links are used total while the relationship betweens short and long ranged links remains consistent. Meanwhile, the 8x8 mesh network is constructed of 112 unit-length links as shown in Figure 4.4 by all of the mesh's links falling into the first bin.

Other construction methods, like [44] discussed in Section 2.3, observed length distributions that fit a power law was correlated to networks with a better-than-average average distance for a lower cost. Our findings agree with this.

Figure 4.5 shows these networks to follow the small world property of having the average distance scale as a function of N. Random networks are represented by the diamond, mesh networks are represented by the square, and the evolved networks with $w = 0.2, 0.5, 0.8$ are represented by the +, x, and * respectively.

Type	Range	Cost
Link 1	[1 3)	1
Link 2	[3 5)	3
Link 3	[5 7)	5

Table 4.3: Parameters of constrained abstract links.

4.1.2 Conclusion

As seen earlier in Table 4.2 and Figure 4.3, the mesh network has a worse average distance compared to any of the evolved networks while only costing more than the networks that were evolved with preference placed on performance. We ascribe this performance increase for a given cost to the evolved network's topology that included a mix of short and long range links while the mesh network is, by definition, limited to strictly short-ranged connections.

4.2 Experiment 2 – Heterogeneous Complex Networks

In many networks, one type of link is not sufficient to connect all components. On a large scale, it is infeasible to drive a car around the world and equally infeasible to fly a jet to the local grocery store. On a much smaller scale, the wires in a nanoscale network must be grown in batches. In this growing process, all wires in a batch are constructed to be the same length [55]. Pertaining to the original challenge presented in this thesis, long range links are becoming infeasible in NoCs while other link type are proving capable of working at long ranges. The link types chosen for this experiment are designed to represent sets of links with constrained feasible lengths as seen in Table 4.3. We adopt three link types – a short, medium, and long range link. The minimum length of any possible link is 1 unit, so that will be our lower limit. The long link seen in Experiment 1 was seven units long,

network	Avg. Dist.	Cost
mesh	5.33	112.00
$w = 0.4$	3.23	63.00
$w = 0.92$	2.75	89.00

Table 4.4: Objective values for mesh and $w = 0.4, 0.92$. $N = 64$. Constrained link lengths.

so that will be our upper limit. The cost of each link is chosen to be constant for each type such that the longer links cost more than the shorter links.

We run this experiment on an 8x8 network to strike a good balance between the computation time needed to run an evolution while avoid the discretization effects of constructing networks that are too small. The evolution is ran for $8,000$ generations with a population size of 400. CNEA is given access to all three link types for all evolutions while evolution are ran across the range of w.

4.2.1 Results

Example networks are shown in figure 4.6. The most apparent pattern visible in the network plots is the existence of hubs – a small number of highly connected nodes and a large number of nodes with a small number of connections. Supporting our current expeirment, the formation of hubs has been observed by [44] and said to be a result of optimizing a network to cost and average distance.

The corresponding objective values to the networks shown in Table 4.6 with a comparison to the mesh network are shown in Table 4.4. As in Experiment 1, the EA built networks that were all better than the mesh in terms of average distance. This time, however, even the evolution that favored performance ($w = 0.92$) resulted in a lower cost than the mesh.

Figure 4.7 maps the number of links used with the distribution of link types

as a function of the weightage chosen in a stacked bar chart. From a weightage of 0.0 to 0.6, only 63 of the short links are used, which is exactly the amount needed to create a tree – a graph that is connected, but no more. At a weight of 0.8, a few more of the short links are used. From a weight of 0.9 and up, an increasing number of medium- and long-range links are used as well as more short-ranged links.

4.2.2 Conclusion

Through the formation of hubs, the EA was able to generate networks that were better than the mesh network in terms of cost and average distance. Additionally, as preference is placed on performance, the EA began using the mid- and long-range link types to further increase performance. These additional link types were used to both connect hubs that would have otherwise been unable to be connected with short-range links and connect individual nodes to a distant hub. The longer (and thus, more costly) links were used if and only if preference was set favoring performance.

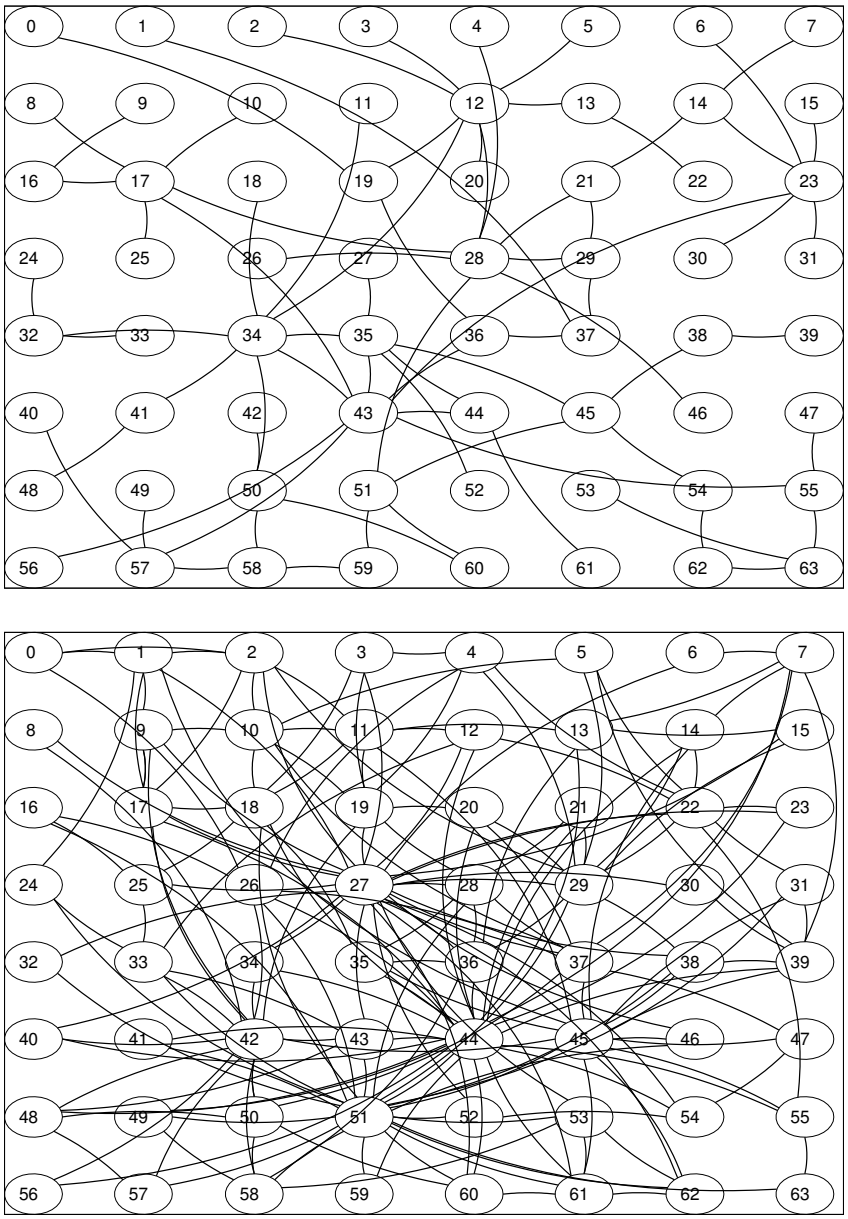


Figure 4.2: Evolved Networks. $N = 64$. $w = 0.2(\text{top}), 0.8(\text{bottom})$.

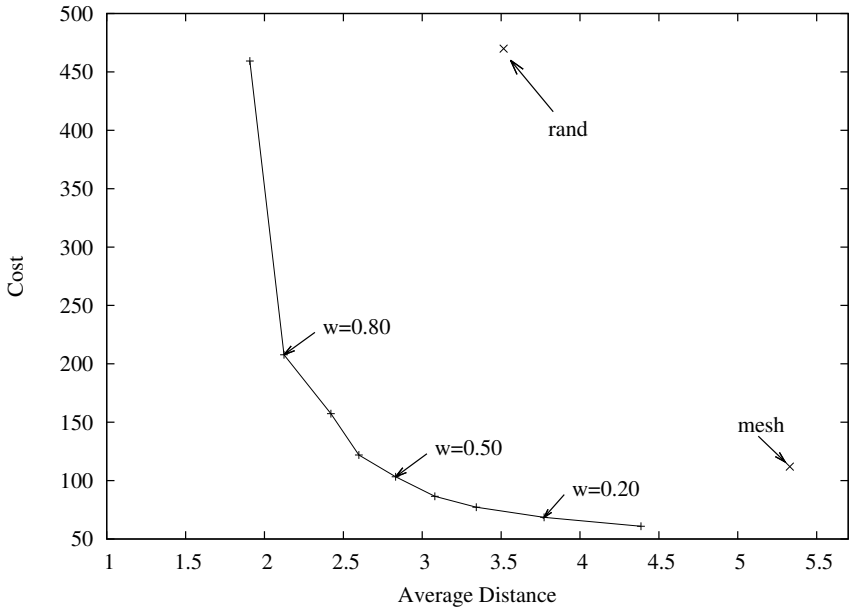


Figure 4.3: Pareto front. N=64, $w = 0.1, 0.2, \ldots, 0.9$. All evolved networks achieved a lower cost than random networks with most evolved networks also achieving a lower average distance. Evolved networks achieved a lower average distance than the mesh with many evolved networks also costing less.

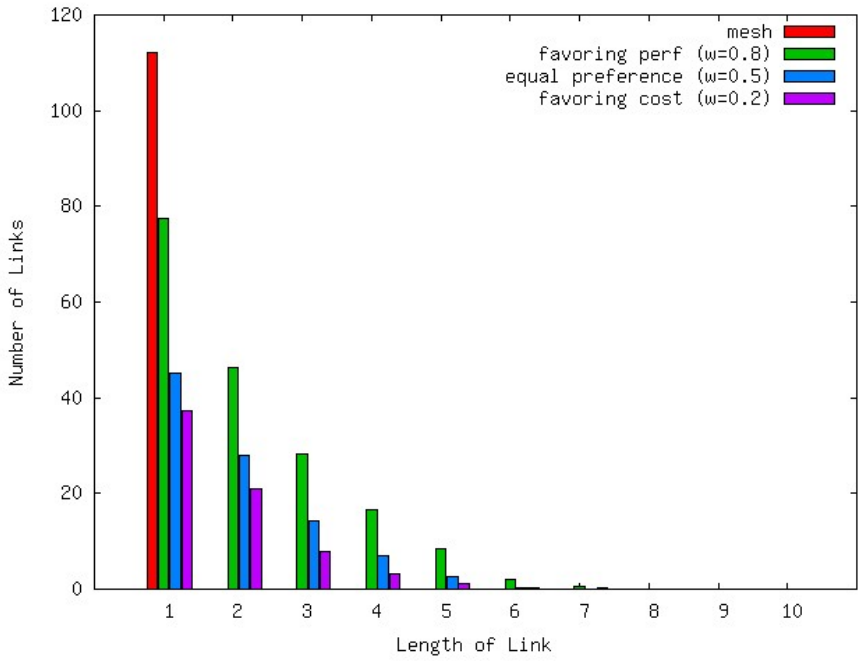


Figure 4.4: Link length distribution. N=64, $w = 0.2, 0.5, 0.8$. All evolved networks show a mix of short- medium- and long-range links. Mesh network only has short-ranged links.

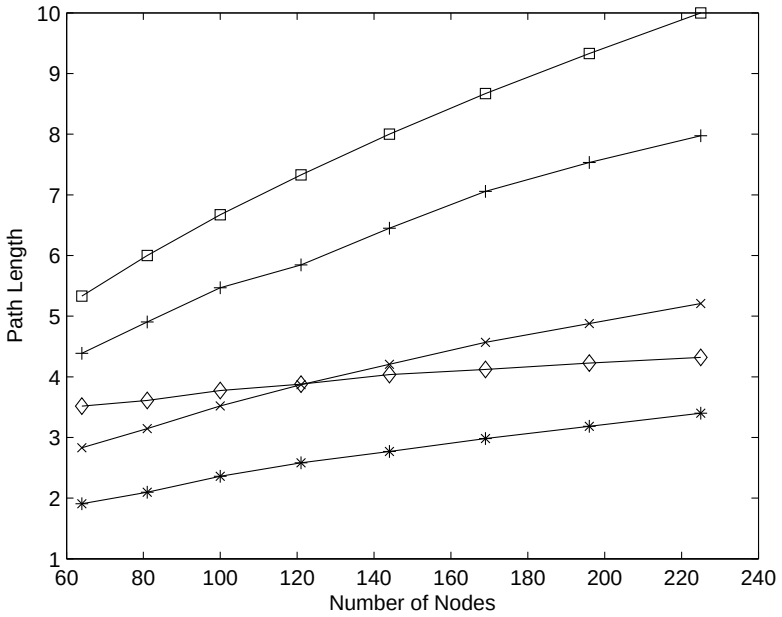


Figure 4.5: Average distance as function of N shows scalability of network performance comparing the evolved networks with mesh and random networks. Evolved networks with $w = 0.2, 0.5, 0.8$ are represented by the +, x, and * respectively. Random and mesh networks are denoted by the diamond $\diamond$ and the $\square$ respectively. The mesh and random networks show the worst and best scalability respectively. The evolved network's scalability does better or worse depending on if preference is placed towards performance or cost respectively.

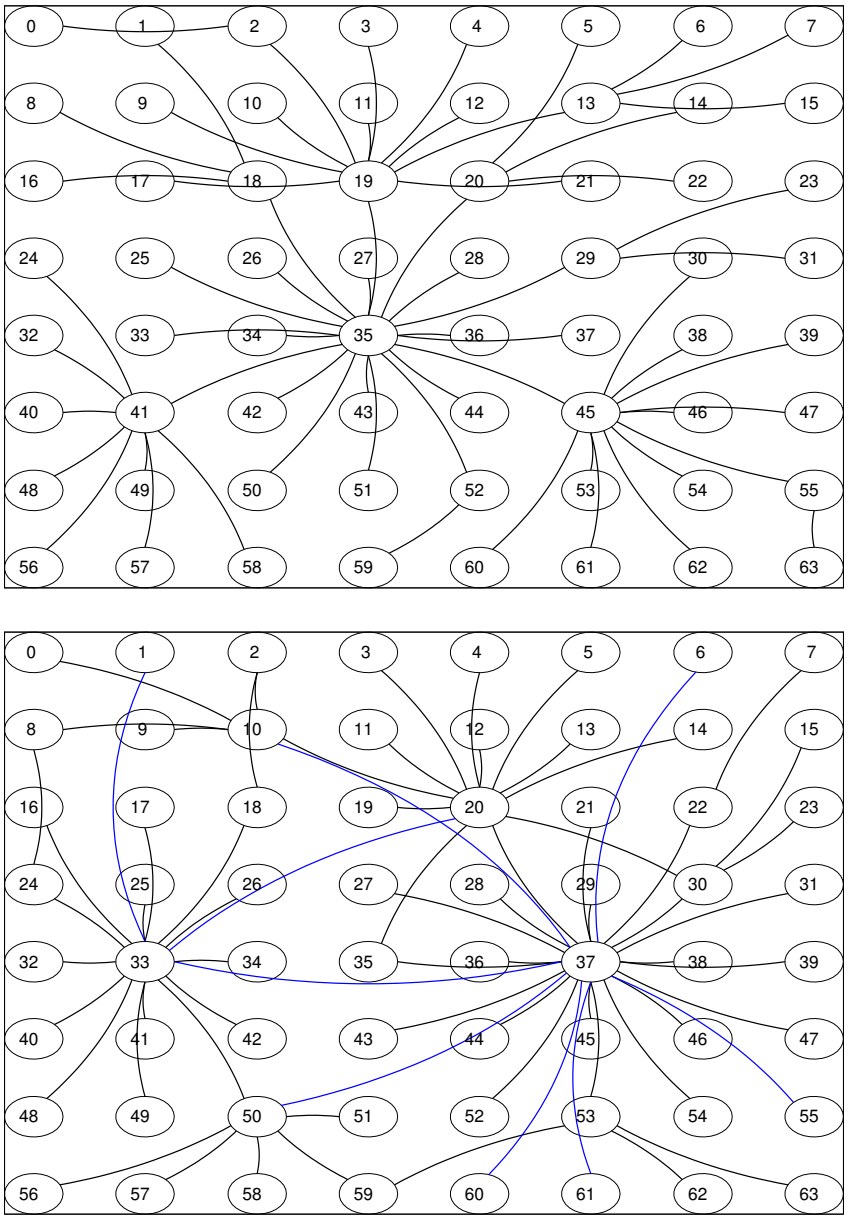


Figure 4.6: Evolved networks. N=64. $w = 0.4(\text{top}), 0.92(\text{bottom})$. Constrained link lengths.

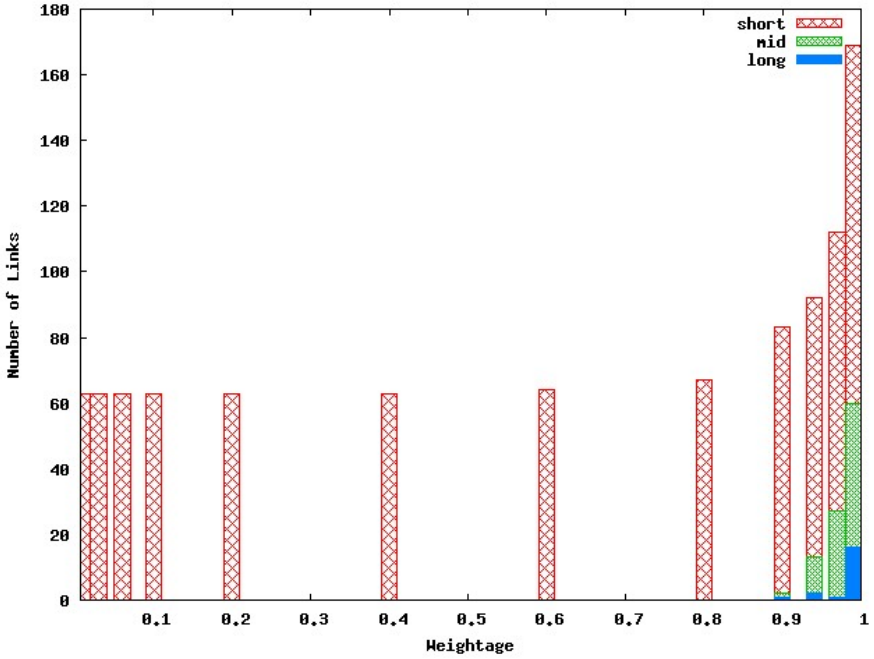


Figure 4.7: Number of links and type distribution for $w = [0, 1]$. Abstract networks. Mid- and long- range links are only used when preference is set to greatly favor performance (i.e., mid- and long-range links start being used at $w = 0.9$).

Chapter 5

Optimization With Traffic Simulation

5.1 Experiment 3 – Network Optimization With Traffic

In the last chapter, we observed that a network evolved to cost and average distance resulted in the formation of hubs. This is not practical in many real-world networks because hubs tend to get too congested and cripple system performance. This experiment will be using cost and the *Critical Traffic Level (CTL)* to continue investigation into heterogeneous networks using a more realistic measure of performance.

When constructing real-world networks, we often have access to many types of links – each with their own unique set of characteristics. For example, the Photonic link applied to NoCs has a very high throughput at long ranges compared to a wire link but costs more to implement [4]. To abstractly investigate this inevitability of having different link types with various qualities, we adopt two link types with parameters given in Table 5.1. The first type labeled as "cheap" has a low cost and a low bandwidth while the other link type labeled as "costly" has a high bandwidth and a high cost.

These evolutions are ran for $10,000$ generations with a population size of 400 to help ensure unbiased convergence. Evolutions, as in past experiments, are repeated across the range of w. Uniform random traffic is used for all evolutions in this experiment.

Link Type	Cost	Bandwidth
cheap	1	1
costly	2	2

Table 5.1: Parameters of abstract links with traffic.

network	CTL	Cost
cheap mesh	0.070	112.00
costly mesh	0.100	224.00
$w = 0.0$	0.019	63.00
$w = 0.1$	0.180	364.06

Table 5.2: Objective values for cheap and costly mesh networks and $w = 0.0, 0.1$. $N = 64$. Networks with traffic.

5.1.1 Results

Example networks are shown in Figure 5.1. Comparison of these networks' objective values with the mesh network built with the cheap link type and the costly link type are given in Table 5.2. It is seen that totally favoring cost, by setting $w = 0.0$, produces a tree with unit-length links: the most sparse but still connected network possible. This network costs less than either mesh or the other evolved network, but also has the worst performance. With $w = 0.1$, a network with about 50% more cost and 80% more performance than the costly mesh is constructed.

The stacked bar chart showing the number of links and link type distribution (similar to Figure 4.7 from Experiment 2) is shown in Figure 5.2. When $w = 0.0$, the minimum number of links is used. As the weight increases, more and more of the cheap links are used to achieve a higher CTL. Once preference is placed around 0.8, then the links with a higher bandwidth are used. As weightage increases up to 1.0, the high-bandwidth links are increasingly used.

5.1.2 Conclusion

We conducted a traffic-based optimization where all links' range are unconstrained but the more costly links have a higher bandwidth. The link with a low cost is used as preference is placed on optimizing cost. More of the cheap links are used and the costly links become worth using as preference is placed on performance. This sort of relationship was first observed in Experiment 1, where less of all ranges of links (from short- to long-range) were used when preference favored cost and more of all ranges of links were used when preference favored performance. It was verified in Experiment 2, where the more costly links had a longer feasible range and were only used when preference was placed towards performance.

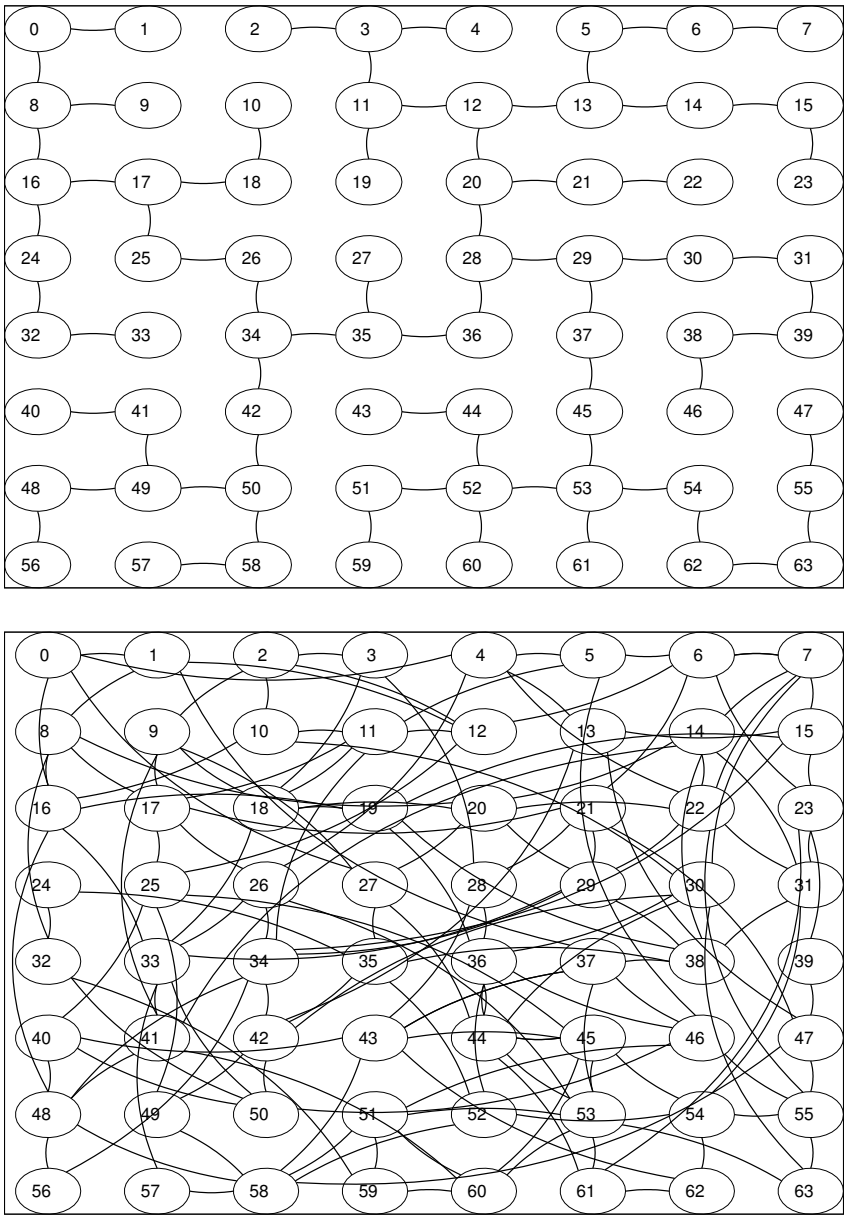


Figure 5.1: Example networks. N=64. $w = 0.0(\text{top}), 0.1(\text{bottom})$. Networks with traffic.

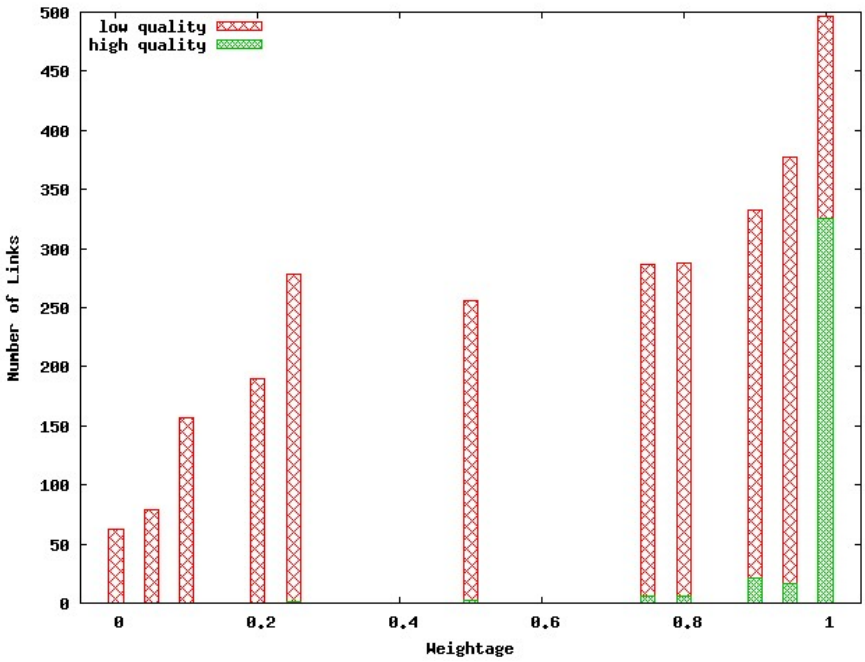


Figure 5.2: Number of links and type distribution for $w = [0, 1]$. Networks with traffic. High bandwidth links are only used when preference is set to greatly favor performance.

Chapter 6

Optimization of Heterogeneous Network on Chip

6.1 Experiment 4 – Optimization to Different Traffic Patterns

In the last chapter, we began exploring evolutions using the results of a traffic simulation as the "performance" part of the fitness function. In real implementations of NoCs, traffic patterns nearly never follow a uniform distribution and even frequently change from one pattern to another as a particular application changes. One concern of optimizing networks to one traffic pattern is that it may result in impairing the performance other traffic patterns. This experiment is a case study designed to evaluate if optimizing to one traffic pattern causes a loss of performance on other traffic patterns.

We select the traffic patterns of uniform, hotspot, and transpose for the evolutions. Four sets of evolutions are ran. The critical traffic level (CTL) is for all three traffic patterns and the average is taken for the first set of evolutions. The other three sets use only one of the traffic patterns when finding the critical traffic level. Once evolutions take place, we test the networks on all three traffic patterns individually and all three averaged to evaluate the bias formed to a traffic pattern during an evolution.

We select an evolution with a population size of 400 that runs for $10,000$ generations to again oversize the algorithm to help ensure unbiased convergence. We also select a weight of 0.5 to balance our preference to cost and performance.

6.1.1 Results

The results are given in Figure 6.1. Each major cluster denotes which traffic pattern is used during a network's construction as denoted in the x-axis (i.e., the first cluster shows the networks that used the *uniform* traffic pattern when evaluating a network's fitness within the EA). Within each cluster, the results of testing a network on each of the traffic patterns is shown, as denoted by the plot's key.

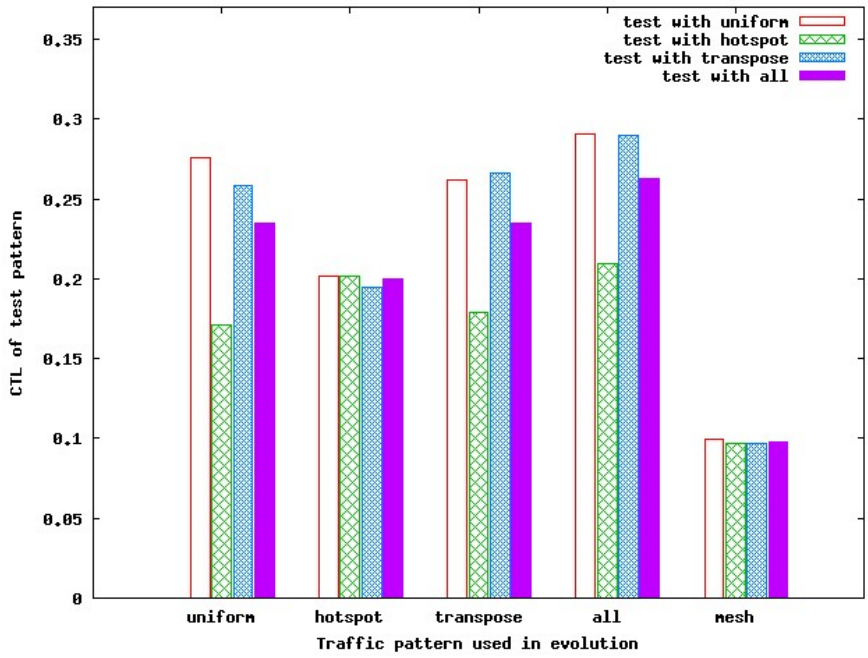


Figure 6.1: Comparative traffic optimization analysis. $w = 0.5$.

Of the evolution that used uniform traffic, it tested higher on uniform traffic than any other. Of the hotspot-based evolutions, both hotspot and uniform traffic reached the same CTL while any traffic pattern fared almost the same. The transpose-based evolution did do the best on the transpose-based test. The evolutions that ran all given traffic patterns ended up doing the best on all available traffic patterns. The mesh network shows insignificant bias towards any one traffic

pattern.

6.1.2 Conclusion

This experiment shows that evolving a network to a particular traffic pattern does create specialization to that traffic pattern as expected. This both reiterates the importance of building accurate traffic patterns relative to the application of a specific network while lending credibility to this methodology's ability to build networks that are specialized once an accurate traffic approximation is built for use in the fitness function.

6.2 Experiment 5 – Optimization of Heterogeneous NoCs

We take the mesh network constructed with the traditional wire interconnects [60] on an 8x8 lattice as the base network. The goal of this experiment is to investigate the potential benefits of augmenting wire links with several novel interconnect fabrics to act as long range links were wires would prove insufficient. The other interconnect fabrics to be used are carbon nanotube (CNT) [16, 35, 52, 53, 75], wireless [2, 31, 41, 61, 62], and photonic [3, 4].

We will address the wireless links as *mmWave*. The mmWave link acts as a set of antennas where each antenna is placed on a node joining all connected nodes in a single bus. Although various methods of bus optimization exist by linking several bus units together [30], we model each mmWave interface (endpoint of the mmWave bus) as operating on the same frequency and thus joined in the same bus. Researchers have successfully applied 24 links to an NoC, so we will limit the number of available CNTs to 24. Photonic links, due to their physical implementation, are not realizable if two or more links cross each other. Thus, we

Link	Bandwidth (Gbits/s)	Energy (pj/bit)	Reference
Wire	$0.04 * L^2 + 0.8$	$0.04 * L^2$	[60]
mmWave	16	2.3	[31]
CNT	10	0.33	[16]
Photonic	240	2	[4]

Table 6.1: Characteristic of heterogeneous NoC links.

apply the constraint forbidding link overlap on the photonic link type.

The network is physically modeled as a 20mm die where each node is evenly spaced apart. The clock frequency is chosen to be 3GHz. Each packet is made of 5 flits while each flit is made of 16 bytes making each packet 640 bits. The wires are modeled as 32 bit links. After appropriate conversions, the delay and energy use for each link is shown in table 6.1.

6.2.1 Results

The traditional mesh network is used as the foundation for the following evolutions, and so we start off with an analysis of the normal mesh network. All networks discussed here are 8x8 (64 node) and have hotspots on node 9 and node 54. In the network diagrams, nodes are shown as black circles. The nodes serving as hotspots have a dark red X drawn behind them. Traffic is modeled at the packet level using shortest path routing. The throughput at a network-level is recorded and given at the end of this document. Throughput at a node-level is presented in the network plot in Figure 6.2. The Red "X" behind nodes 9 and 54 denote the location of hotspots. The boldness of each node within the plot for all networks shown in this experiment is a function of the average throughput on that node as found through the traffic simulation. You can see how the nodes within the two hotspots experience significantly more traffic than the nodes outsize of that region.

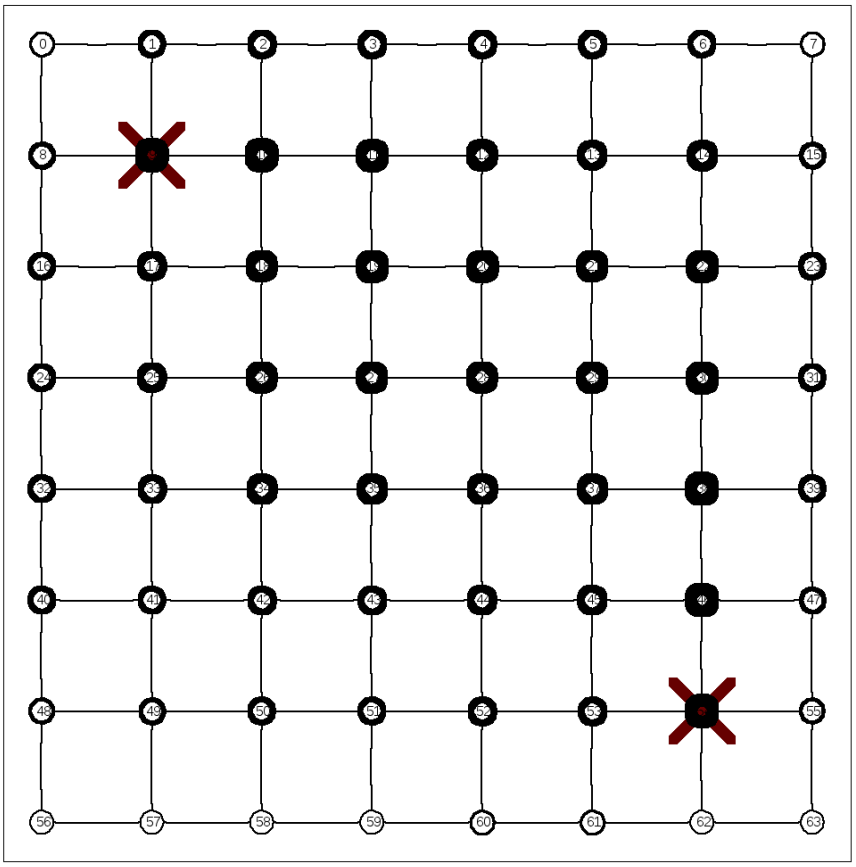


Figure 6.2: Example mesh NoC.

Another way of representing that data is given on the right.

In Figure 6.3, you see the results of an evolution that was designed to favor throughput. Given the choice to use mmWave, CNT, and photonic links, the EA used 7 photonic links to reach its goal. The black links are wire interconnects and the red links are the photonic interconnects. The node-level throughput is again shown through the boldness of each node. The EA, through this placement of photonic links, greatly decreased the number of nodes operating over a relatively high traffic load.

The link length distribution of this network is shown in Figure 6.4. Since

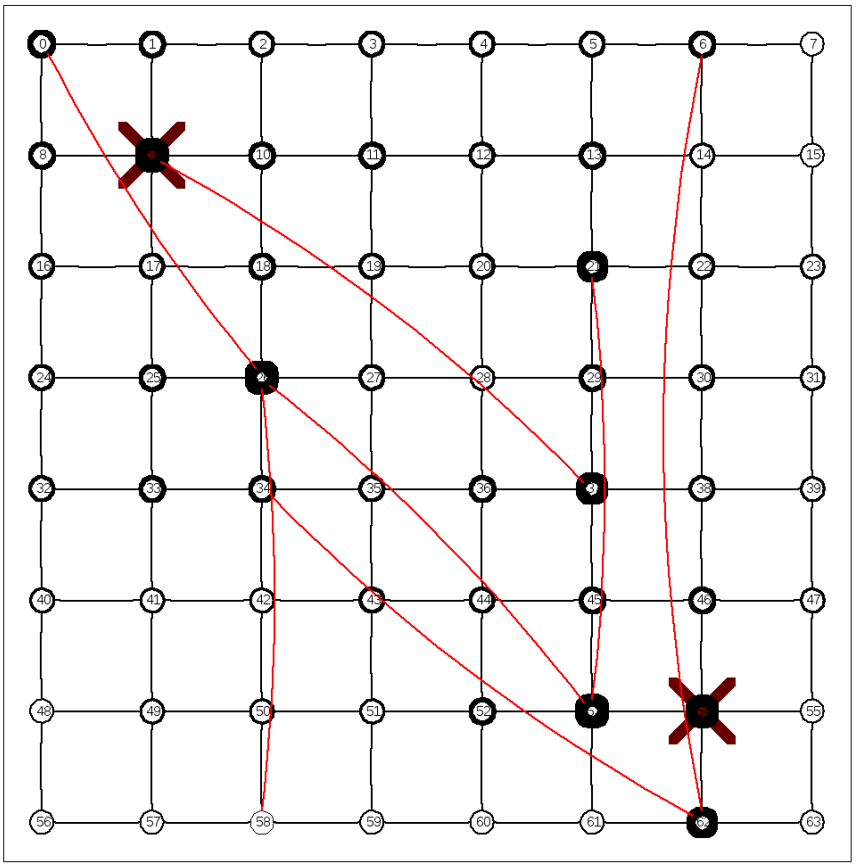


Figure 6.3: Evolved NoC 1.

these networks are mapped onto a 20mm die, each node is 2.86 units away from each other. This is seen by the 112 wire links in the first bin of the link length distribution. The 7 additional photonic links were used as mid-range interconnects averaging a length of about 12 units.

Figure 6.5 shows a heterogeneous network that was evolved to favor energy. Traditional wires are again drawn as black links. CNTs are drawn as blue links. Wireless interfaces are drawn as a dark purple + behind the node with that given interface. Photonic links are again drawn as red links. We observe the predominant use of the CNTs – the most energy efficient link available – with a total of 21 links

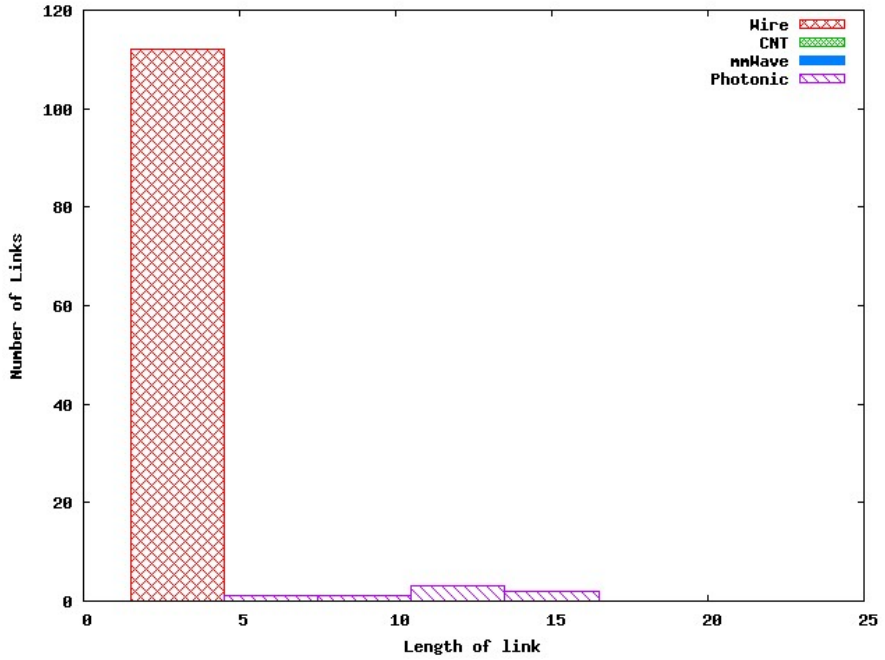


Figure 6.4: Link length distribution with link type distribution for evolved NoC 1. Photonic links are added as mid-range interconnects to aid the wire mesh.

Network	Throughput (packets/cycle/N)	Energy (pj/packet)	Extra Links
Mesh	0.020	35.31	0
Evolved NoC 1	0.034	38.54	7
Evolved NoC 2	0.023	21.48	32

Table 6.2: Objective values of heterogeneous NoCs.

of this type added. A total of 9 of the also-very-energy-efficient photonic links are also used. Of the mmWave link type, 2 interfaces are added near opposing ends of the network.

The link length distribution of this network is shown in Figure 6.6. The 112 wire links are again seen making up the base mesh network. All three additional link types are used as mid- and long-range links averaging a length of about 17 units.

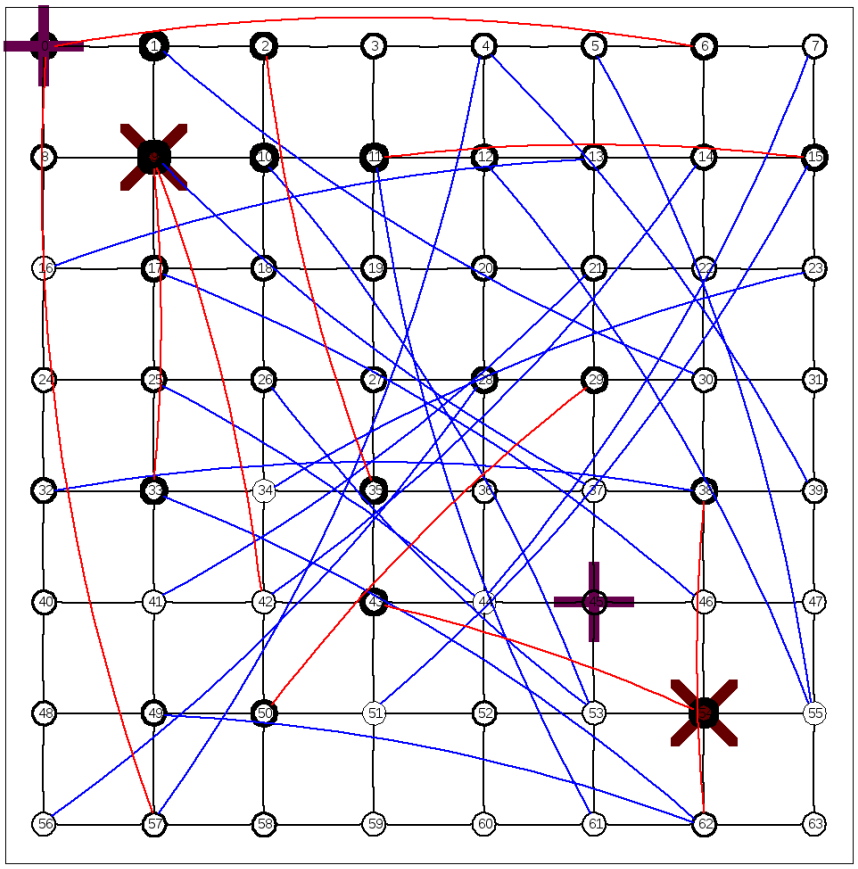


Figure 6.5: Evolved NoC 2.

6.2.2 Conclusion

The network with the best (lowest) energy usage came from the network that used the link with the best energy use of the links available. To match what was observed in Experiment 3 where preference to CTL resulted in using more of the low-delay links, the network with the best throughput came from the network using only the links with the highest bandwidth of the set of available links. Both evolved NoCs achieved a higher performance than the mesh by adding mid- and long-range matching what we first observed in Experiment 1. In both evolved NoCs given, as was also observed in Experiment 2, placing preference on performance resulted in

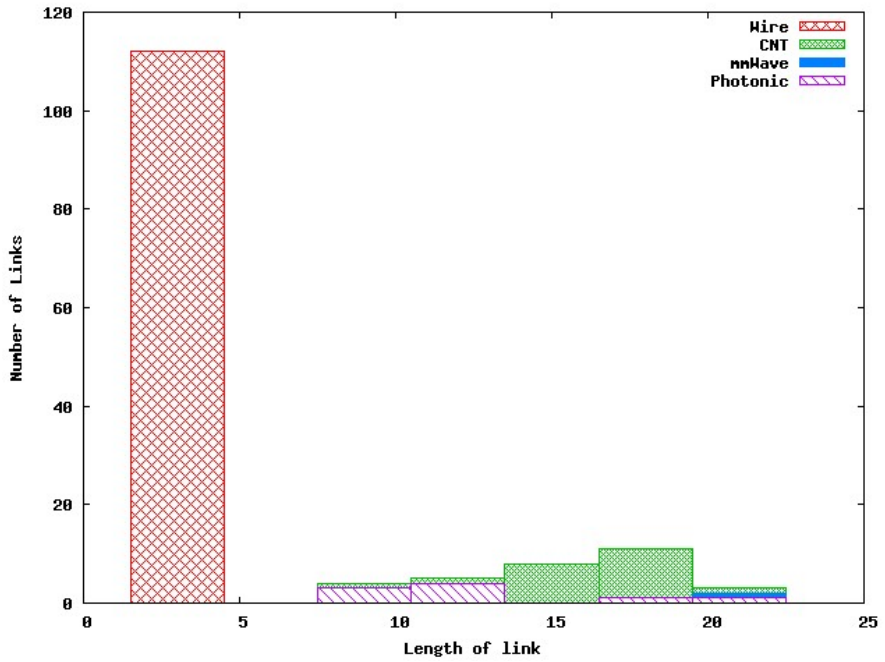


Figure 6.6: Link length distribution with link type distribution for evolved NoC 2. The wire mesh network is augmented with 21 CNTs, 9 photonoic links, and 2 mmWave interfaces used as mid- and long-range interconnects to optimize energy.

more links used.

Even though both evolved NoCs achieved a higher throughput than the mesh, only the network constructed with an evolution favoring energy used less energy than the mesh. However, the mesh network requires no new technologies and no new links. When cost is preferred and the performance of the mesh is acceptable, then the cost associated with additional links is not worth it. But, there are cases when the cost of a network is not as significant. When optimizing the energy is preferred, the links with the most efficient energy usage are implemented by the EA. When throughput is desired, the links with the highest bandwidth are implemented by the EA.

Chapter 7

Conclusion and Next Steps

7.1 Summary

In Chapter 2, we first introduced graph theory, random graphs, complex networks as the abstract view of networks. We then discussed the application of network theory to integrated circuit design in the field of *Networks on Chips (NoCs)*. Due to poor scaling of the delay across global interconnects as miniaturization progresses causing multi-hop long-range communication, we discuss augmenting the traditional wire-based NoC with novel link types. We hypothesize that mixing different link types capable of long-range single-hop communication with short-range wire links in what we call a heterogeneous network will result in networks with a better performance for a given cost compared to networks constructed only with wires. As the platform of experimentation, the *Complex Network Evolutionary Algorithm (CNEA)* with the associated *Complex Network Framework (CNF)* and supporting scripts are constructed. The specifics of that implementation is covered in Chapter 3.

Starting with Experiment 1 (in Chapter 4), networks are evolved to average distance and cost and we observe that both short- and long-range links are used by networks optimized to average distance and cost. A higher preference to average distance results in seeing more short- and long-range links. Experiment 2 shows that, given a set of length-constrained links where the shorter links cost less than the longer links, preference to a network's cost results in using only the short, less costly, links. Changing the preference to increasingly favor average distance results

in an increasing number of mid- and long-range links to be used.

Optimization to a traffic simulation is introduced in Chapter 5 by evolving networks to cost and the critical traffic level in Experiment 3. Two link types are used: one with a low cost and high delay and another with a high cost and low delay. As preference is placed to cost, only the low-cost high-delay link is used. As preference increasingly favors performance, an increasing number of high-cost low-delay links are used.

After establishing our methodology and exploring abstract networks in Experiment 1-3, we explore the optimization of NoCs in Chapter 6. Beginning with a case-study of the bias towards a particular traffic pattern resulting from optimization using that traffic pattern in Experiment 4, we find that the traffic pattern used in optimization does determine how well the resultant network performs on that traffic pattern. Finally, in Experiment 5, we do two case studies of augmenting the standard mesh NoCs with three novel link types. In one case, we give preference to efficient energy usage of the network in the traffic simulation and in the other case, we give preference to network throughput. We find that the EA primarily uses the link with the least energy consumption when preference is set to favor energy usage. Also, the EA uses only the link with the highest bandwidth when preference is set to optimize network throughput.

7.2 Conclusion

When we optimized to average distance and cost with preference placed on average distance, the more costly links better suited to optimize average distance were used. Whenever optimization favored cost, only the less costly links were used. When optimization favored throughput, higher bandwidth links were used. Implicit in all

of the previous experiments involving heterogeneous networks was the fact that no one link type dominated another in performance (if one type had a longer range, then it cost more. If one type had a lower delay, then it spent more energy). This was deliberately built into the sets of links that were fabricated for experimentation and happened to be true in the realistic link exploration. Indeed, a link type with a higher cost and worse performance than a link already available to the network would not be useful unless it had some other desirable property. So, in conclusion, the addition of a link type to a network is only beneficial if the properties of that link can be exploited to improve the desirable properties of the network. In the case of NoCs, the addition of novel links types as long range links will be beneficial if resultant performance gains are worth the cost of implementing the new technology.

References

[1] A. Ganguly, K. Chang, S. Deb, P. P. Pande, B. Belzer, and C. Teuscher. Scalable hybrid wireless network-on-chip architectures for multi-core systems. *Computers, IEEE Transactions on*, 2010.

[2] A. Ganguly, K. Chang, P. P. Pande, B. Belzer, and A. Nojeh. Performance evaluation of wireless networks on chip architectures. pages 350–355, march 2009.

[3] A. Joshi, C. Batten, Y. Kwon, S. Beamerand, I. Shamim, K. Asanovic, and V. Stojanovic. Silicon-photonic clos networks for global on-chip communication. In *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 124–133, May 2009.

[4] A. Shacham, K. Bergman, and L. P. Carloni. Photonic networks-on-chip for future generations of chip multiprocessors. *Computers, IEEE Transactions on*, 57(9):1246–1260, 2008.

[5] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74(1):47–97, Jan 2002.

[6] G. Ascia, V. Catania, and M. Palesi. Multi-objective mapping for mesh-based NoC architectures. In *International Conference on Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004*, pages 182– 187. IEEE, September 2004.

[7] Semiconductor Industry Association. International technology roadmap for semiconductors (ITRS), 2003.

[8] L. Benini and G. De Micheli. Networks on chips: A new soc paradigm. *Computer*, 35(1):70–78, January 2002.

[9] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38, June 2006.

[10] B. Bollobas. *Modern Graph Theory*. Springer, corrected edition, July 1998.

[11] M. Buchanan. *Nexus: Small Worlds and the Groundbreaking Theory of Networks*. W. W. Norton & Company, June 2003.

[12] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, and F. Hessel. Exploring noc mapping strategies: an energy and timing aware technique. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 502–507 Vol. 1, 2005.

[13] C. Teuscher, N. Parashar, M. Mote, N. Hergert, and J. Aherne. Wire cost and communication analysis of self-assembled interconnect models for networks-on-chip. In *Proceedings of the 2nd International Workshop on Network on Chip Architectures*, NoCArc '09, pages 83–88, New York, NY, USA, 2009. ACM.

[14] S. Cahon, N. Melab, and E. G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, May 2004.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2nd edition, September 2001.

[16] H. Dai. Carbon nanotubes: Synthesis, integration, and properties. *Accounts of Chemical Research*, 35(12):1035–1044, 2002.

[17] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[18] W. Darrell. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, 2001.

[19] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[20] P. Erdös and A. Rényi. On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[21] P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.

[22] P. Erdös and A. Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12:261–267, 1961.

[23] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Graph Theory 1736-1936*, February 1736.

[24] X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'Kindt. *Metaheuristics for Multiobjective Optimisation*. Springer, first edition, March 2004.

[25] M. T. Gastner and M. E. J. Newman. The spatial structure of networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 49:247–252, 2006.

[26] P. Gehlot and S. S. Chouhan. Performance evaluation of network on chip architectures. In *Emerging Trends in Electronic and Photonic Devices Systems, 2009. ELECTRO '09. International Conference on*, pages 124–127, 2009.

[27] M. Gómez, P. López, C. Gómez, and J. Duato. Reducing packet dropping in a bufferless NoC. In *Proceedings of the 14th International Euro-Par conference on Parallel Processing*, Euro-Par '08, pages 899–909, Berlin, Heidelberg, 2008. Springer-Verlag.

[28] M. Gurevitch. The social structure of acquaintanceship networks.

[29] M. Horowitz and W. Dally. How scaling will change processor architecture. In *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pages 132–133 Vol. 1, 2004.

[30] C.-T. Hsieh and M. Pedram. Architectural energy optimization by bus splitting. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(4):408–414, April 2002.

[31] J.-J. Lin, H.-T. Wu, Y. Su, L. Gao, A. Sugavanam, J. E. Brewer, and O K. K. Communication using antennas fabricated in silicon integrated circuits. *Solid-State Circuits, IEEE Journal of*, 42(8):1678–1687, Aug. 2007.

[32] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005.

[33] D. S. Johnson, J. K. Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978.

[34] K. A. De Jong. *Evolutionary Computation*. The MIT Press, 1st edition, March 2002.

[35] K. Kempa, J. Rybczynski, Z. Huang, K. Gregorczyk, A. Vidan, B. Kimball, J. Carlson, G. Benham, Y. Wang, A. Herczynski, and Z. F. Ren. Carbon nanotubes as optical antennae. *Advanced Materials*, 19(3):421–426, 2007.

[36] J. Karbowski. Optimal wiring principle and plateaus in the degree of separation for cortical neurons. *Phys. Rev. Lett.*, 86(16):3674–3677, Apr 2001.

[37] R. Kasturirangan. Multiple scales in Small-World graphs. *cond-mat/9904055*, April 1999.

[38] J. Kleinberg. The Small-World phenomenon: An algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.

[39] R. A. Legenstein and W. Maass. Foundations for a circuit complexity theory of sensory processing. In *Advances in Neural Information Processing Systems 13*, pages 259–265. MIT press, 2001.

[40] M. O. Lorenz. Methods of measuring the concentration of wealth. *Publications of the American Statistical Association*, 9(70):pp. 209–219, 1905.

[41] M.-C. F. Chang, E. Socher, S.-W. Tam, J. Cong, and G. Reinman. RF interconnects for communications on-chip. In *ISPD'08*, pages 78–83, 2008.

[42] M. Keijzer, J. Guervós, G. Romero, and M. Schoenauer. Evolving objects: A general purpose evolutionary computation library. In *Selected Papers from the 5th European Conference on Artificial Evolution*, pages 231–244, London, UK, 2002. Springer-Verlag.

[43] S. Mahadevan, F. Angiolini, M. Storgaard, R. Olsen, J. Sparso, and J. Madsen. A network traffic generator model for fast network-on-chip simulation. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 2*, DATE '05, pages 780–785, Washington, DC, USA, 2005. IEEE Computer Society.

[44] N. Mathias and V. Gopal. Small worlds: How and why. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 63(2 Pt 1):021117, February 2001.

[45] A. Meyerson and B. Tagiku. Minimizing average shortest path distances via shortcut edge addition. In *Proceedings of the 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX '09 / RANDOM '09, pages 272–285, Berlin, Heidelberg, 2009. Springer-Verlag.

[46] G. De Micheli and L. Benini. *Networks on Chips: Technology and Tools*. Morgan Kaufmann, first edition, July 2006.

[47] S. Milgram. The small-world problem. *Psychology Today*, 2:60–67, 1967.

[48] G. E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, January 1998.

[49] O. Ogawa, S. Bayon de Noyer, P. Chauvet, K. Shinohara, Y. Watanabe, H. Niizuma, T. Sasaki, and Y. Takai. A practical approach for bus architecture optimization at transaction level. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 176–181 suppl., 2003.

[50] U. Y. Ogras and R. Marculescu. "It's a small world after all": NoC performance optimization via long-range link insertion. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(7):693–706, 2006.

[51] N. Oshida and S. Ihara. Packet traffic analysis of scale-free networks for large-scale network-on-chip design. *Phys. Rev. E*, 74(2):026115, Aug 2006.

[52] P. J. Burke, S. Li, and Z. Yu. Quantitative theory of nanowire and nanotube antenna performance. *Nanotechnology, IEEE Transactions on*, 5(4):314–334, 2006.

[53] P. P. Pande, A. Ganguly, K. Chang, and C. Teuscher. Hybrid wireless network on chip: A new paradigm in multi-core design. In *Network on Chip Architectures, 2009. NoCArc 2009. 2nd International Workshop on*, pages 71–76, 2009.

[54] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *Computers, IEEE Transactions on*, 54(8):1025–1040, 2005.

[55] P. Xu, S.-H. Jeon, H. T. Chen, H. Luo, G. Zou, Q. Jia, M. Anghel, C. Teuscher, D. Williams, B. Zhang, X. Han, and H.-L. Wang. Facile synthesis and electrical properties of silver wires through chemical reduction by polyaniline. *The Journal of Physical Chemistry C*, 114(50):22147–22154, 2010.

[56] L.-S. Peh and N. E. Jerger. *On-Chip Networks*. Morgan and Claypool Publishers, first edition, July 2009.

[57] T. Petermann and P. De Los Rios. Physical realizability of small-world networks. *Phys. Rev. E*, 73(2):026114, Feb 2006.

[58] G. F. Pfister and V. A. Norton. Interconnection networks for high-performance parallel computers. chapter Ḧot spotc̈ontention and combining in multistage interconnection networks, pages 276–281. 1994.

[59] Chen Q., J. A. Davis, Payman Z.-H., and J. D. Meindl. A novel via blockage model and its implications. In *Interconnect Technology Conference, 2000. Proceedings of the IEEE 2000 International*, pages 15 –17, 2000.

[60] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[61] S.-B. Lee, S.-W. Tam, I. Pefkianakis, S. Lu, M. F. Chang, C. Guo, G. Reinman, C. Peng, M. Naik, L. Zhang, and J. Cong. A scalable micro wireless interconnect structure for CMPs. In *MOBICOM'09*, pages 217–228, 2009.

[62] S. Deb, A. Ganguly, K. Chang, P. P. Pande, B. Beizer, and D. Heo. Enhancing performance of network-on-chip architectures with millimeter-wave wireless interconnects. In *Application-specific Systems Architectures and Processors (ASAP), 21st IEEE International Conference on*, pages 73–80, 2010.

[63] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Extending the transaction level modeling approach for fast communication architecture exploration. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 113–118, 2004.

[64] I. D. Scherson. *Interconnection Networks for High-Performance Parallel Computers*. Institute of Electrical & Electronics Enginee, 1st edition, January 1994.

[65] J. Sima and P. Orponen. General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation*, 15(12):2727–2778, 2003.

[66] S. H. Strogatz. Exploring complex networks. *Nature*, 410:267–276, Mar 2001.

[67] E. Talbi. *Metaheuristics: From Design to Implementation*. Wiley, June 2009.

[68] A. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, fourth edition, 2002.

[69] C. Teuscher, H. Chung, A. Grimm, A. Amarnath, and N. Parashar. The power of power-law's: Or how to save power in SoC. In *Proceedings of the Second International Green Computing Conference (IGCC'11)*, 2011. In press.

[70] J. Travers and S. Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):pp. 425–443, 1969.

[71] D. J. Watts. *Six degrees: the science of a connected age*. W. W. Norton & Company, January 2004.

[72] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[73] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.

[74] J. L. Wong, A. Davoodi, V. Khandelwal, A. Srivastava, and M. Potkonjak. A statistical methodology for Wire-Length prediction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(7):1327–1336, 2006.

[75] Y. Huang, W. Yin, and Q. Liu. Performance prediction of carbon nanotube bundle dipole antennas. *Nanotechnology, IEEE Transactions on*, 7(3):331–337, May 2008.

[76] S.-H. Yook, H. Jeong, and A.-L. Barabsi. Modeling the internet's large-scale topology. *Proceedings of the National Academy of Sciences of the United States of America*, 99(21):pp. 13382–13386, 2002.