Deterministic Routing Table using Genetic Algorithm

EE 382C Final Project
Peggy Lin, Yi-Ting Wu, Andrew Romero, Erik Luna
Department of Electrical Engineering
Stanford University

1 Introduction

When selecting a routing algorithm for any network topology with path diversity, most options are either adaptive or deterministic. Adaptive algorithms are effective for traffic patterns that change over time, because the algorithm can change the way packets are sent through the network in real time, hopefully avoiding congestion. Adaptive methods also effectively handle faulty nodes without changing the routing algorithm. Deterministic routing algorithms designate one path for each source destination pair. Deterministic methods are easier to implement because there is less metadata flowing through the network, and are guaranteed to be deadlock free if certain rules are in place. The deterministic algorithms might include dimension-order-routing, which is a set of dimension orders in which data can travel, or a routing table can be used to manually guide the packets through the network. In this project, we attempt to find an optimal routing table for various traffic patterns and injection rates a network may encounter.

To find the optimal routing table, we use a Genetic Algorithm (GA) to iteratively explore possible routes and converge to an optimal solution, which we explain in section 2 [1]. We implement this algorithm using two and three dimensional mesh networks. These networks are common in many applications, as discussed later, and are feasible to implement in the project time-frame. Prior work shows preliminary results of the GA working on a 3D mesh with adversarial and random traffic. We implement a routing algorithm that achieves two goals: it avoids typical adaptive congestion information overhead, and it switches between genetic and adaptive routing in real time.

2 Implementation

2.1 Genetic algorithm

2.1.1 Initial table generation

The GA table initialization is a crucial step in this algorithm. The initial table stores all possible paths for a given source-destination (SD) pair. Once the initial table is determined, subsequent GA iterations will only focus on finding the best chromosome that represents the best combination of paths (genes) from the initial table.

Our proposed GA table initialization involves collecting all unique paths from simulations of multiple combinations of 11 traffic patterns, 11 routing algorithms provided in booksim [2], and 8 different injection rates. Populating the initial table using existing routing algorithms allows us to construct a more comprehensive routing search space compared to random neighboring path generation. This approach can also ensure that our GA algorithm operates within a well-informed and diverse routing solution space, enhancing its effectiveness in finding optimal routing paths.

But since the existing 11 traffic patterns don't cover all $N(N-1)$ possible SD pairs, we add a customized neighbor traffic pattern to generate packets sending from source node i to destination node $(i + \text{custom step } k)$, where we can iterate k from 1 to $(N-1)$ so that all SD pairs are guaranteed to be covered.

After collecting all potential paths, the GA table is constructed to have $N(N-1)$ columns, with each column representing one SD pair, and 2^b rows, with each row representing a potential packet path for the corresponding SD pair. To manage the size of the table, we use random selection to prune the number of path collections to 8 (where $b = 3$), and randomly duplicate the paths if there are less than 8 path collections originally to ensure consistency.

2.1.2 Training

The purpose of the training stage is to find the best path for each SD pair through iterations of the network simulation. Given the initial table we previously generated, we have eight candidates, or eight possible paths, for each SD pair to traverse. Through the iterative process, we test the fitness of different combinations of paths to minimize the overall network congestion.

To start with, we create a generation of n_chrom chromosomes. Each chromosome is a combination of $N(N-1)$ genes, or SD pair path, which is a b-bit index number used for indexing the initial table. The b-bit index encoded by the gene allows us to search the initial table and find the specific entry of the path corresponding to the SD pair.

The first generation of chromosomes is randomly generated, that is, the first set of path combinations is randomly chosen from all the feasible paths. Each subsequent generation of chromosomes is formed by the genetic process that includes score extraction, parent selection, crossover and mutation from the current generation.

1) **Score Extraction**
   For each chromosome, we perform decoding to translate the SD paths from the initial table indices into the sequence of node numbers that the paths traverse. This decoded path is fed into `booksim` to construct the deterministic routing table for GA routing function. With the deterministic paths imported, we run booksim with the deterministic table routing and extract average packet latency as the fitness score for the particular chromosome. Latency is extracted through the log file generated by booksim, with infinity assigned to simulations that are saturated. The score extraction step is equivalent to testing the average latency of the network running on different path combinations specified by the chromosomes.
2) **Selection**
   After all chromosomes are assigned a score from the simulation packet latency, we perform roulette selection to choose a subset of chromosomes based on the fitness score as investigated in the literature [3]. This is a stochastic method in which the probability of selecting a chromosome is proportional to its fitness score. The `n_chrom` chromosomes selected from the current generation are called the "parent chromosomes" and are used to produce the next generation of "child chromosomes" in the next step. We added an extra step to exchange the selected chromosome with the lowest fitness score with a randomly chosen chromosome to mitigate worst-case bias of the roulette probability selection. An alternative approach to selecting the parents is the best-of-k tournament approach which is also random and achieves similar results [4].
3) **Crossover and Mutation**
   After obtaining the parent chromosomes from the above selection step, we perform crossover by combining two parent chromosomes to produce two children as our next generation chromosomes. Crossover is performed by exchanging parts of the parents' genes spliced from a random split index. The mutation step randomly flips a bit in the children chromosomes. Both the crossover and mutation are performed at a designated rate rather than on all chromosomes. We find that the mutation rate of 30%, i.e. flips 30% of the children's chromosomes, yields the optimal result.

In this work, we calculate the fitness score on uniform traffic with 5% packet injection rate, which is equivalent to 25% flit injection with our packet length fixed to 5. We finish training with 100 iterations and decode the chromosome with the best fitness score as our deterministic routing table.

2.1.3 Integration with `booksim`

Booksim assigns flit IDs according to the order of injection into the network. However, since the network is not always empty, this leads to unpredictable flit ID assignments, making it impossible to hardcode a watch list capable of covering packets from all N potential source nodes. To address this issue, we modified booksim to record 10 flit IDs sent from each source to the stats_out file. Subsequently, a watch list including those recorded flit IDs is generated. We also implement a `gWatchFlitPath` log file, similar to `gWatch`, to capture necessary information for extracting end-to-end flit paths from the simulation.

To simulate deterministic table routing, a deterministic routing table is read and constructed in booksim. We also create our own GA routing function. Leveraging the source and destination information carried by each flit, we can extract the routing path from the table. This allows us to locate the current node within the entire path and direct the flit to the next node accordingly.

In the next section, we will further expand the deterministic routing with adaptive switching mechanism.

2.2 VC hierarchical switching

2.2.1 Overview

During testing with plain GA routing, we frequently encounter network saturation even at low flit injection rates, which raises concerns about potential deadlocks. While we do include paths from known deadlock-free routing in our initial table, deadlock can still arise from the combination of different source-destination paths.

Employing a fitness score could penalize deadlock-prone chromosomes with lower scores, as deadlocks result in infinite latency. However, there could be cases where all chromosomes within the current generation encounter deadlock, leading to the failure of the GA to generate a satisfactory offspring generation. This highlights a significant challenge in effectively addressing deadlock issues within plain GA routing.

Thus, we propose a novel VC hierarchical switching mechanism designed to not only prevent deadlock but also adaptively switch between adaptive and deterministic routing strategies.

2.2.2 VC Priority

In our design, each router has 8 virtual channels (VCs). Drawing inspiration from resource classes deadlock avoidance techniques, we divide VCs into 3 distinct classes and impose a partial order on the resources to eliminate cycles in the resource dependence graph.

Out of the 8 virtual channels, VC 0 is dedicated to dimension-order routing (DOR), while in VCs 1-3 are allocated to minimal adaptive routing. The remaining VCs 4-7 are designated for GA routing. We prioritize GA VCs over minimal adaptive VCs and then DOR VC, as we hypothesize that the GA table offers a routing solution closer to optimality. However, recognizing that deterministic routing may lack the flexibility to live traffic flow, we include the second priority class, allowing it to switch from deterministic to adaptive routing. Finally, the last VC class is reserved for deadlock prevention, where DOR the deadlock free routing strategy is used.

In the proposed VC hierarchical switching scheme, we leverage the embedded booksim VC priority flag to implement our switching mechanism. When routing, each flit initially attempts to route through the GA VCs. However, if all GA VCs are congested, the flit will then opt for adaptive routing VCs. In the event of both failures, the flit will resort

to using the DOR VC. This hierarchical approach ensures efficient routing by prioritizing GA routing while providing fallback options to adapt to varying network conditions.

Following the resource classes, a flit can only route to classes in ascending order. If the flit fails to obtain allocation in the GA VCs (class 0) and instead is allocated to the adaptive VCs (class 1), it is constrained to subsequent routing paths using only adaptive VCs (class 1) and DOR VC (class 2). The flit cannot revert to utilizing GA VCs (class 0) again once it has been assigned to a higher class. This restriction ensures a strict adherence to the hierarchical switching mechanism and prevents backtracking to higher-priority VC classes after allocation to lower-priority ones.

Furthermore, the switching mechanism operates at the flit level, rather than globally switching the routing function altogether. This means that the VC class utilized by one flit does not impact the VC class used by another flit. Each flit independently determines its routing path based on the availability and congestion of the designated VCs. This approach also offers an easy recovery mechanism. As each flit determines its routing path independently, there's no need to define a specific threshold or decide when to switch back and forth between deterministic and adaptive routing strategies.

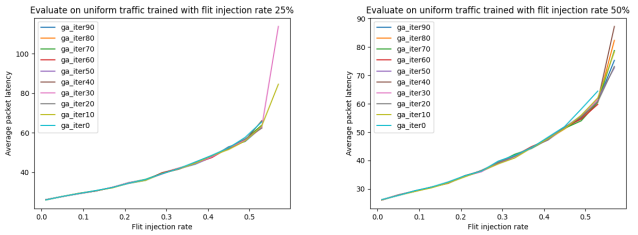
3 Experimental Results

We conduct experiments using a fixed n=3, k=4 mesh network to explore path diversity. We choose k=4 as it is suitable for running bitcomp traffic and is within our computational capacity. The GA training consists of 100 iterations, except for the subsection on exploring different iteration counts. We use a 3-bit representation for the initial table rows, with a crossover rate of 0.2 and a mutation rate inversely proportional to the network size. We will evaluate various aspects of our GA algorithm.

3.1 Comparison of GA iterations

Our hypothesis is that the GA algorithm can find close-to-optimal routing solutions over iterations and randomization. We compare results using the best-fit path from each generation as the booksim deterministic routing path. The GA table is pre-trained using uniform traffic with 25% and 50% flit injection rates, and transpose traffic with 5% and 15% flit rates. We then validate the GA iteration outcomes on uniform and transpose traffic, respectively.

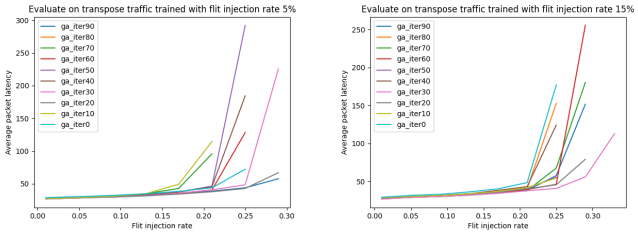
In the case of uniform traffic, little difference in performance is observed over iterations, as seen in Figure 1. For balanced loaded traffic like uniform, our randomized first generation chromosome could have indeed provided a decent routing, and subsequent randomization and selection of chromosomes have a limited improvement space to explore.



(a) 25% training flit rate (b) 50% training flit rate

Fig. 1: Comparison of GA iterations for uniform traffic

Conversely, in the case of transpose traffic, performance variation between iterations is evident, as shown in Figure 2. Since transpose traffic patterns create more congestion, a randomized or simple one-pass heuristic may lead to congested paths, which means that the search space for an optimal path combination enlarges. With more iterations, GA explores varying path assortments, increasing the probability of finding an optimal path combination. While later iterations do not guarantee better performance, they enhance the chance of locating an improved routing solution.



(a) 5% training flit rate (b) 15% training flit rate

Fig. 2: Comparison of GA iterations for transpose traffic

3.2 Comparison between GA and other routing algorithms

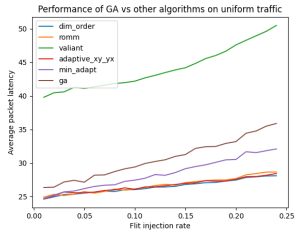
In this section, we compare the performance of GA with other routing algorithms. We evaluate the GA paths using the same traffic patterns as those used for training and evaluation. Specifically, GA models trained on uniform, diagonal, bitcomp, and transpose traffic patterns are evaluated against their respective training patterns, as depicted in Figure 3.

In benign traffic patterns like diagonal, GA performs comparably to other adaptive routing algorithms. However, in uniform traffic scenarios, GA exhibits higher packet latency compared to other methods. The well-balanced nature of uniform traffic limits GA's potential for improvement through iterations and may lead to over-optimization and reduced real-time flexibility.

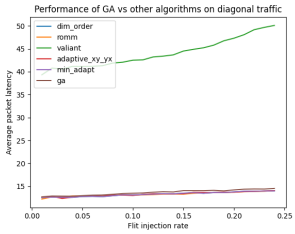
In bitcomp traffic, GA shows similar performance to other routing algorithms, but experiences a sharp increase in packet latency at higher injection rates. Notably, at high injection rates, GA may exhibit higher latency than valiant routing due to its switching mechanism to minimum adaptive routing under excessive congestion. This switching behavior can cause packets to take additional hops, resulting in increased latency.

In transpose traffic scenarios, GA performs moderately well and even outperforms dimension order and min adaptive routing at high injection rates. The high volume of traffic flowing through the same nodes in the same direction makes the dimension order heuristic less effective. In contrast, GA can learn from past experiences and maintain a better understanding of the global network, leading to improved performance.

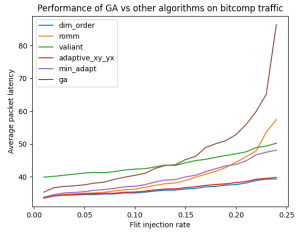
Overall, GA has advantages in certain challenging traffic patterns but not universally, possibly due to suboptimal hyperparameters or specific overhead issues, which we aim to address in future work.



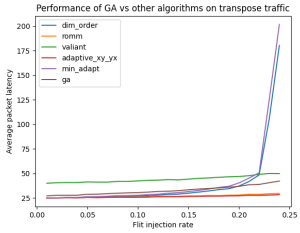
(a) Uniform training traffic



(b) Diagonal training traffic



(c) Bitcomp training traffic

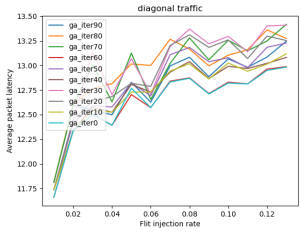


(d) Transpose training traffic

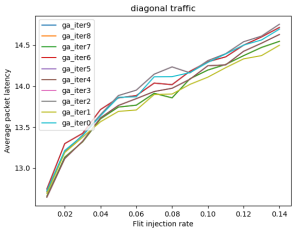
Fig. 3: Evaluation of GA versus other routing algorithms

3.3 Comparison between network sizes

Larger networks have more path diversity, which is where we expect GA to perform well. Using a diagonal traffic pattern, we compare a small and large three-dimensional networks, as shown in Figure 4. For a small network, the first generation of chromosomes are the most fit. In the larger network, we see slight improvement in model performance for later generations.



(a) Small Network: n=3, k=2



(b) Large Network: n=3, k=6

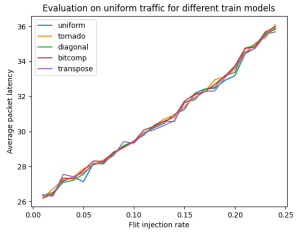
Fig. 4: Comparison of GA iterations for network sizes

3.4 Comparison of different training traffic pattern

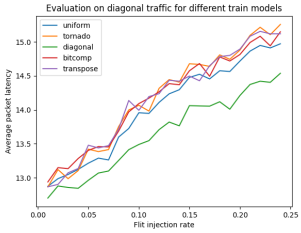
During GA training, fitness scores are extracted from chromosome performance in selected traffic patterns. We hypothesize that models trained on worst-case traffic scenarios will generalize well, effectively handling both adversarial and benign traffic. We investigate how GA paths trained on different traffic patterns generalize to others by training four models on uniform, diagonal, bitcomp, and transpose traffic patterns and evaluating their performance on alternative patterns (Figure 5).

For benign traffic like uniform and diagonal, differences across training models are minimal. However, in high-congestion scenarios like bitcomp, models trained on uniform and diagonal traffic exhibit significantly higher packet latency at higher injection rates. Conversely, models trained on bitcomp and transpose traffic effectively handle both benign and congested conditions. We conclude that training on adversarial traffic enhances GA model generalizability, as hypothesized.

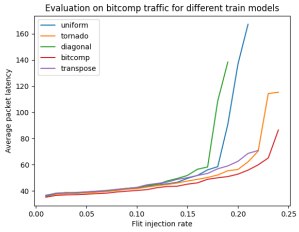
Additionally, the best-performing model for each evaluation traffic pattern is the one trained on the same pattern, aligning with expectations. When evaluation traffic differs from training traffic, model performance varies based on traffic characteristics. In real-world applications with known traffic characteristics, training the GA algorithm on the closest pattern can yield optimal routing paths.



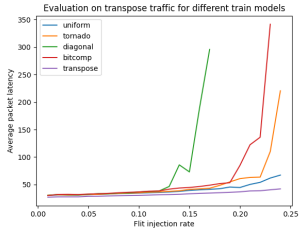
(a) Uniform traffic evaluation



(b) Diagonal traffic evaluation



(c) Bitcomp traffic evaluation



(d) Transpose traffic evaluation

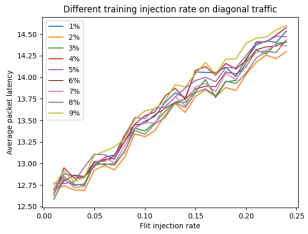
Fig. 5: Evaluation on different train traffic patterns

3.5 Comparison on different training injection rates

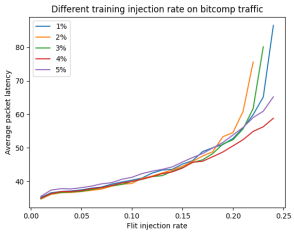
One GA hyperparameter under examination is the packet injection rate used during training, as depicted in Figure 6. Initially, we hypothesized that models trained on lower injection rates would perform better on evaluation sets with lower injection rates, and vice versa. Our experiments reveal that in bitcomp traffic, the model trained on a 5% packet injection rate initially exhibits lower performance at low evaluation injection rates but improves as the injection rate increases, consistent with our hypothesis. However, we observe little difference in uniform traffic and a relatively random correlation in diagonal and transpose traffic. This discrepancy may be attributed to suboptimal solutions found in GA iterations, which may not perform well in low-congestion networks if actual source-destination pairs

differ from those in the training traffic. Consequently, we conclude that the training injection rate has minimal impact on performance.

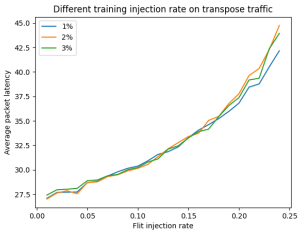
(a) Uniform traffic evaluation



(b) Diagonal traffic evaluation



(c) Bitcomp traffic evaluation



(d) Transpose traffic evaluation

Fig. 6: Evaluation on different train packet injection rate

4 Real-world application

2D and 3D Mesh topologies are ubiquitous in modern silicon and server applications. For example, 3D memory technologies are required to keep lots of data close to compute units in hardware accelerators, like GPUs. Some servers, like Google's TPUv4 services, use a 3D mesh to group TPUs together, allowing for a modular and scalable configuration [5].

We also note the value of pre-computing network parameters (and topologies) for specific applications. State-of-the-art network topologies for LLM inference use fiber optic switches that are configured before the LLM is made accessible, as attempted successfully by companies like qroq.ai [6]. This allows the topology to adapt to its application and available nodes using an algorithm like ours to maximize throughput of token generation.

5 Future Work

As shown in the results above, we notice that GA does not always improve the average packet latency in our tested networks. More hyper-parameter tuning may cause an overall increase in performance. We'd also like to explore situations where GA works well so that we can better switch between algorithms or tables to achieve the lowest packet latency at all times. Another consideration is training on higher injection rates to improve performance for higher injection rate incoming test traffic. Since this algorithm was trained at 5% to 10%, it learned on less congested paths that may not represent higher injection rates.

It would be ideal to test the GA in networks where the traffic pattern is changing, especially in situations that are encountered frequently in networking applications used today. We'd expect an adaptive routing algorithm to perform well for changing traffic patterns, but using a mix of adaptive and GA where GA is trained on the most frequently seen traffic pattern should perform best. This opens the possibility to a predictive routing algorithm that uses time of day, week, and year to select unique tables based on previously seen traffic.

Another idea to explore is if GA can train as the network is running, generating tables adaptively. This would be difficult because the network needs to keep track of the traffic it sees over time and if the traffic is always changing, or changing right before the adaptive GA can find a reasonable table, this method will not work.

6 Work division

The contributors of this paper collaborated together on various fronts to achieve the results outlined above. Our group of four, Peggy Lin, Andrew Romero, Yi-Ting Wu, and Erik Luna, delineated our work as shown below to deliver our tiered switching routing algorithm.

Andrew researched the genetic algorithm and proposed this as a viable method for routing 3D meshes. He engineered the initialization of a routing table based on pre-existing booksim routes, followed by work with Peggy to solve filling all SD pair issue. This was crucial work for initializing the paths that many of our chromosomes read from. Andrew designed the structure of the initial GA paths table and the chromosomes.

Peggy and Yi-Ting delved into the booksim routing code. Yi-Ting construct the GA table class in C++ for booksim to interpret deterministic paths for our GA routing algorithm. Peggy engineered the end-to-end path extraction from booksim simulations and resolve deadlock conflicts as they appeared in our simulations. Peggy and Yi-Ting integrated code into the `ga_mesh` routing and solved our network problem of saturation by proposing the VC hierarchy to avoid deadlock in our network, which includes GA, adaptive, and dimension-order-routing VCs, in order of decreasing priority.

Erik investigated the theory of the genetic algorithm and its use in traffic networks, and constructed the GA iteration template in Python. He also implemented the optimizer code using Optuna to find optimal hyperparameters for the genetic algorithm to improve its convergence to find the lowest latency paths.

References

[1] M. Bougherara and A. Rafik, "Routing using genetic algorithm in network on chips with 3d mesh topology," *2023 International Conference on Computer and Applications (ICCA)*, pp. 1–5, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:267203440

[2] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, J. Kim, and W. J. Dally., "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software*, 2013.

[3] M. A. Mohammed, M. S. Ahmad, and S. A. Mostafa, "Using genetic algorithm in implementing capacitated vehicle routing problem," in *2012 International Conference on Computer & Information Science (ICCIS)*. IEEE, 2012, pp. 257–262.

[4] R. Amara, M. B. Aissa, R. Kemcha, M. Bougherara, and N. Louam, "Geographical information system for air traffic optimization using genetic algorithm," *GeoInformatica*, pp. 1–25, 2022.

[5] [Online]. Available: https://cloud.google.com/tpu/docs/system-architecture-tpu-vm

[6] [Online]. Available: https://wow.groq.com/why-groq/