ABSTRACT

Title of dissertation: MODELING AND OPTIMIZATION
OF TRANSMISSION NETWORKS

Ian Frommer, Doctor of Philosophy, 2005

Dissertation directed by: Professor Brian Hunt
Department of Mathematics

Professor James Yorke
Department of Mathematics

Professor Bruce Golden
Decision and Information Technologies Department

This dissertation focuses on transmission networks. These networks play an important role in communication (including data and voice), energy transmission (such as gas, electrical, and oil), and micro-electronics among other areas. In this dissertation, we consider two different problems associated with transmission networks: the dynamics on a data communication network and the optimal design of a general transmission network located on a non-uniform landscape. The first two parts of this dissertation develop and apply mathematical models of congested Internet connections and describe possible extensions to network traffic state estimation and network security. The third part looks at an optimal network design problem through a variation on the Euclidean Steiner tree problem, a well-known network optimization problem.

MODELING AND OPTIMIZATION OF TRANSMISSION NETWORKS

by

Ian Frommer

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

Advisory Committee:

Professor Brian Hunt, Chair/Advisor
Professor James Yorke, Co-Advisor
Professor Bruce Golden, Co-Advisor
Professor Edward Ott
Professor Kenneth Berg

To Laura and Arielle.

ACKNOWLEDGMENTS

I would like to thank Professors Brian Hunt, James Yorke, Bruce Golden, and Edward Ott for guiding me through the many challenges of my dissertation. Their knowledge, vision, patience, flexibility and good humor helped me at every step along the way. I would also like to thank Eric Harder for his support. I am grateful to Professor Kenneth Berg for serving as dean's representative on my committee. Finally, I would like to thank my family: Laura, Arielle, Myrna, Harvey, Jennifer and Freddy Frommer, Caroline Katz, Peter and Harriet Greenwald, and friends for their encouragement, advice, and support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

This dissertation focuses on transmission networks. These networks play an important role in communication (including data and voice), energy transmission (such as gas, electrical, and oil), and micro-electronics among other areas. In this dissertation, we consider two different problems associated with transmission networks: the dynamics on a data communication network and the optimal design of a general transmission network located on a non-uniform landscape. Chapters two and three develop and apply mathematical models of congested Internet connections and describe possible extensions to network traffic state estimation and network security. Chapter four looks at an optimal network design problem through a variation on the Euclidean Steiner tree problem, a well-known network optimization problem.

In chapter two, we introduce two deterministic models aimed at capturing the dynamics of congested Internet connections. The first model is a continuous-time model that combines a system of differential equations with an impulse, or sudden change in one of the state variables. The second model is a discrete-time model with a time step that arises naturally from the system. Results from these models show good agreement with the well-known *ns* network simulator, better than the results of a previous, similar model. This is due in large part to the use of the impulse to reflect the impact of lost data packets. We also discuss the potential use of these models in network traffic state estimation. This chapter is an extended version of a paper [1] that appeared in the Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN '04).

In chapter three we consider a network consisting of one TCP sender and one non-TCP sender that transmits uniformly sized bursts of packets at regular intervals. The non-TCP flow can represent probing activity of either a diagnostic measurement or attack. Using bifurcation diagrams we study how the network's behavior changes as a function of the probing frequency.

These diagrams reveal interesting, non-intuitive behavior. We first apply a stochastic version of our continuous-time model to this system. Results from the model show good qualitative agreement with those of the simulator. Next we show that a simpler deterministic version of the model is able to capture many of the main features of the behavior of this network as well. We then further simplify our representation of the system by introducing a piece-wise linear, one-dimensional map that is still able to reproduce much of the dynamics seen in the more complicated model. This map helps explain the structure of the bifurcation diagram, and allows us to directly determine Lyapunov exponents, which give a measure of the system's predictability. As a result, we are able to more precisely describe and categorize the dynamics, including chaos, exhibited by this network.

Chapter four discusses an algorithm we developed for the optimal design of transmission networks. These networks can be utilized for communication, power transmission, and other applications. We consider a variation of the Euclidean Steiner tree problem in which the space underlying the set of nodes has a specified non-uniform cost structure. This problem is significant in many practical situations, such as laying cable networks, where the cost for laying a cable can be highly dependent on the location and the nature of the area through which it is to be laid. We present a genetic-algorithm-based procedure and apply it to a variety of test cases of this problem. We show that our procedure is able to find optimal or near-optimal solutions in a small fraction of the time taken by an exact algorithm. In addition, we show that because of its novel features, our genetic algorithm outperforms a more traditional genetic algorithm. This chapter is based in part on an earlier paper [2] that appeared in the Proceedings of the INFORMS Computing Society Conference (ICS '05).

Chapter 2

Two Models for the Study of Congested Internet Connections

2.1 Introduction

An Internet connection consists of the exchange of data packets between a source and destination through intermediate computers, known as routers. The transmission of data in the majority of Internet connections is controlled by the Transport Control Protocol (TCP) [3, 4]. TCP is responsible for initializing and completing connections, controlling the rate of flow of data packets during a connection, ensuring that lost packets are retransmitted, etc.

Congestion can occur when the rate of flow of the connection is limited by some link on the path from source to destination. This link is known as the *bottleneck* link. Routers have buffers in which to store packets in case one of their outgoing links reaches full capacity. If new packets arrive at a router whose buffer is full, that router must drop packets. There exist various strategies, known as active queue management (AQM) policies, for determining how and when to drop packets prior to the buffer filling. One commonly used AQM is Random Early Detection (RED) [5].

In this chapter we model the interaction between TCP and RED for a simple network connection experiencing congestion. This scenario, or ones similar to it, have been modeled previously for the purposes of evaluating RED [6, 7, 8, 9], obtaining throughput expressions [10, 11], and obviating the need for time-consuming simulation [6, 12, 13, 14], among others. There are stochastic [11] and deterministic models [6, 7, 8], continuous-time [6] and discrete-time models [7, 8].

Our models are closest to that of Misra et al. [6]. That model successfully captures the *average* network state in a variety of situations. This allows the authors to analyze RED from a control theoretic basis. Our aim is to develop a model of greater accuracy that will be useful in estimation not only of the average network state, but of additional quantities. For example,

a model capable of capturing the mean queue length allows one to estimate the mean round-trip time. But a model that can capture the range, and better still, the variance of the queue length, will allow one to estimate the range and variance of the round-trip time. While this level of model accuracy can be useful, it is necessary in a model that is to be used for our ultimate goal of network traffic state estimation. Given some knowledge of the state of the network, an accurate model may be combined with a filter-based state-estimation scheme (which we discuss briefly in Sec. 2.4), in order to estimate the full network state at the current time. Network state estimation can be useful from the perspectives of both security and performance.

We make several assumptions about the Internet connection we are modeling. First, we assume it involves the transfer of a very large amount of data over an extended period of time. In this so-called *bulk transfer*, the senders always have data to transmit for the duration of the connection. Common TCP traffic such as FTP (File Transfer Protocol) file transfers, and non-streaming music and video downloads can all constitute bulk transfer. Second, we assume that the path from source to destination is fixed for the duration of the connection. Third, we assume the transfer is one-way, i.e., the data packets travel in only one direction. (Acknowledgments packets only travel in the other direction.) Fourth, any cross-traffic through the path is negligible. Fifth, the path contains one bottleneck link whose capacity is less than that of all other links in the path. Most of these assumptions reflect typical network scenarios with the possible exceptions of the cross-traffic and bottleneck assumptions.

Data packets traveling from the sender to the receiver are likely to pass through several intervening routers. However, based on our assumption that there is one bottleneck link, we need only consider the router whose outgoing link is this bottleneck link. All other routers simply forward the data along the path, and have unoccupied buffers. Combining all of the above assumptions, we can model the network we are attempting to represent with a network of one sender and one receiver separated by one router (see Fig. 2.1).

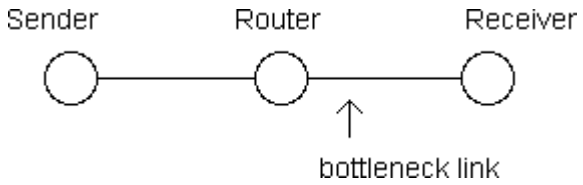

Figure 2.1: Network configuration

2.2 The Models

The primary mechanism used by TCP to regulate the flow of data is the *congestion window* on the sender's computer. A window size of W means that the number of unacknowledged packets from the sender that may be outstanding on the network at any one time is at most W. According to TCP, the sender's window size follows the additive-increase/multiplicative-decrease (AIMD) scheme. Typically, the window size W increases additively by $1/W$ with each successful packet transmission and decreases multiplicatively by a factor of 2 when packets are dropped.

Packets needing to be stored in a router's buffer due to congestion are placed in a queue. The RED module operating at this router keeps track of the queue length as well as of an exponentially-weighted average of the queue length, which we denote by x. When x exceeds a pre-defined minimum threshold, RED will cause the router to drop arriving packets with probability that increases with x. Note that this adds a stochastic element to the connection. We attempt to capture the system behavior with a deterministic model; our model will not reflect the random component of the dynamics. However, as we will show below, the random component is reasonably small, and so a purely deterministic model can be useful for state estimation.

Previous attempts to model this scenario deterministically have made use of either discrete-time maps [7, 8] or differential equations [6]. Continuous-time models employing delay differential equations may do well in capturing the aspects of network behavior that evolve on small-time scales, such as flow rate increases. However, they are likely to smooth out the effects of sudden large changes in state (such as flow rate reductions). Furthermore, systems of delay differential equations can be cumbersome to express and solve due to the delay.

We address the above issues through innovations in our models, namely, a discrete impulse

in the continuous-time model, the choice of time-step in the discrete-time model, and a packet-based frame of reference used by both models. The continuous-time model presented in this work combines a system of differential equations to capture the evolution of the small-time scale changes with a discrete impulse (sudden change in one of the state variables) to represent sudden large changes. The discrete-time model takes advantage of the fact that a congested bottleneck will impose uniform packet spacing. Using this spacing as the time step allows the discrete-time model to nicely capture the system behavior. In addition, as explained below, both models utilize a packet-based frame of reference for the state, which simplifies handling of the delay in the system, making the models easier to express and execute.

To describe the system, we consider the state variables, all of which have units of "packets" but are allowed to take non-integer values:

- W – the congestion window size of the sender;
- q – the length of the queue at the router;
- x – the exponentially-weighted average of the queue length at the router.

In our packet-based frame of reference, $W(t)$ represents the congestion window in effect for a packet *arriving at the queue* at time t. This was the sender's congestion window at an earlier time, when the packet left the sender.

In developing the models we will often refer to the round-trip time, R. The round-trip time is the time between the departure of a packet from the sender and the return to the sender of the acknowledgment for that packet. It consists of a fixed propagation delay, a, and a queuing delay. For a packet encountering a single congested router with queue length q, and outgoing link capacity c (measured in packets per unit time), the queuing delay is q/c. Thus,

$$R(q) = a + \frac{q}{c}. \tag{2.1}$$

2.2.1 The Continuous-Time Model

Our continuous-time model consists of two distinct parts. The first part is a system of differential equations similar to the one found in [6]. It is used to model the additive-increase of the window size, and both instantaneous and averaged queue lengths. The second part of the model is an impulse in which the window size state variables are instantaneously reset. This is used to model the multiplicative decrease in the window size caused by a dropped packet.

Modeling the Additive Increase

According to TCP, the window size, W, will normally increase by $1/W$ upon receipt of each acknowledgment. (We do not model the "slow start" phase in which the window size increases by one per received acknowledgment.) Since W specifies the number of unacknowledged packets that can be out in the network at any one time, it follows that about W packets are sent per round-trip time, R. Thus, in approximately one round-trip time, W acknowledgments return (assuming no lost packets), causing W to increase by 1. Although network data is transmitted in discrete packets, during the period of additive-increase and in the absence of dropped packets, quantities appear to vary smoothly when viewed over the time-scales we are interested in (on the order of 1 to 100 seconds) for our network settings. Hence we model the additive-increase of the window size continuously:

$$\frac{dW(t)}{dt} = \frac{1}{R(q(t))}. \tag{2.2}$$

The rate of change of the queue length at a given time is equal to the difference between the flows into and out of the router. By the explanation above, the flow into the router at time t is $W(t)/R(q(t))$. If the queue is occupied, the flow rate out will be equal to the capacity of the router's outgoing link, c. Otherwise the flow out is equal to the minimum of c and the incoming flow rate:

$$\frac{dq(t)}{dt} = \begin{cases} \frac{W(t)}{R(q(t))} - c & \text{when } q(t) > 0, \\ max\left(\frac{W(t)}{R(q(t))} - c, 0\right) & \text{when } q(t) = 0. \end{cases} \tag{2.3}$$

The exponentially-weighted average queue is determined by RED upon each packet arrival

as follows:

$$x_{n+1} = wq_{n+1} + (1 - w)x_n, \tag{2.4}$$

where w is the exponential-weighting parameter, $0 \leq w \leq 1$, and the subscripts denote successive measurements of q and x (which occur at packet arrivals) for a given router. This equation describes a low-pass filter, and, making use of the fact that w is small in practice, it can be approximated by the differential equation:

$$dx(t)/dt = w(q(t) - x(t))(W(t)/R(q(t))). \tag{2.5}$$

The term $W(t)/R(q(t))$ expresses the rate of packets arriving at the queue. The model is summarized by equations (2.2), (2.3), and (2.5).

Because of our bulk-transfer and bottleneck assumptions (see Sec. 2.1), the network is often congested to the point of saturation. This allows us to simplify the model by discarding the $q = 0$ case in equation (2.3), and replacing the packet rate term in equation (2.5) with the bottleneck link capacity, c, as follows:

$$\frac{dW(t)}{dt} = \frac{1}{R(q(t))} \tag{2.6}$$

$$\frac{dq(t)}{dt} = \frac{W(t)}{R(q(t))} - c \tag{2.7}$$

$$\frac{dx(t)}{dt} = wc(q(t) - x(t)). \tag{2.8}$$

Modeling the Multiplicative Decrease

In a real network situation, a router using RED will drop an arriving packet with probability p given by:

$$p(x) = \begin{cases} 0 & \text{when } x < q_{min}, \\ \frac{x - q_{min}}{q_{max} - q_{min}} p_{max} & \text{when } q_{min} \leq x \leq q_{max} \ , \\ 1 & \text{when } x > q_{max}. \end{cases} \tag{2.9}$$

In practice, this usually causes a packet to be dropped soon after x exceeds q_{min}. For the purposes of keeping our model simple and deterministic, we assume a drop occurs as soon as x exceeds q_{min}. We continue to evolve the continuous system for one round-trip time to reflect the delay in notification of the sender that a packet has been dropped. Then we cut the window state variable, W, in half. Next the continuous evolution resumes but we hold the window state variable constant for one round-trip time.[1] This is done in order to represent the Fast Recovery/Fast Retransmit(FRFR) algorithm in the NewReno[2] version of TCP. Once this round-trip time has elapsed, the model resumes continuous evolution as described above.

Denoting the continuous-time system of equations (2.6), (2.7), and (2.8) by **A**, we can summarize the continuous-time model in hybrid systems form as in Fig. 2.2, where T is a timer variable. The use of a hybrid system to express the model was suggested by [12]. Our model begins in the congestion avoidance phase, following the set of equations, **A**. Once x exceeds q_{min}, the model assumes a drop occurs, and it transitions to the delayed drop notification phase. In this phase, it continues following the original set of equations but now it also utilizes a count-down timer, T, which expires one round-trip time from the time x exceeds q_{min}. Once this timer expires, the model transitions into the recovery phase, first cutting W in half and initializing a new count-down timer at the value of the present round-trip time. In the recovery phase, the equations still hold, but with one exception: W is now held fixed. Finally, when this count-down timer expires, the model returns to the congestion avoidance phase.

[1] For simplicity, we are neglecting the period in which no data are transmitted which would necessarily occur after the window cut.

[2] Recent indications are that NewReno appears to be the most widely used variant of TCP [3].

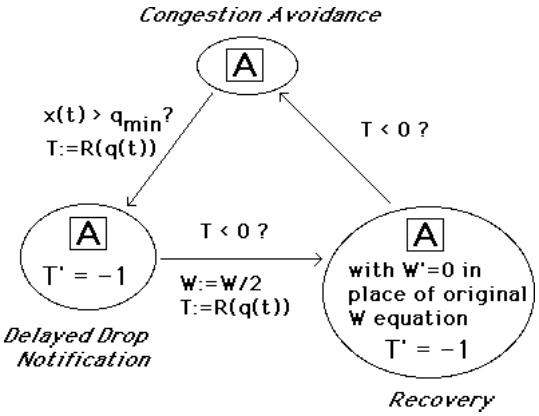

Figure 2.2: Hybrid systems view of the continuous-time model. **A** stands for equations (2.6), (2.7), and (2.8). $T' = dT/dt$ and $W' = dW/dt$.

Modeling Multiple Senders

The continuous-time model can be easily generalized to handle more than one sender. As in the case of one sender, we assume each additional sender is upstream from the bottleneck link and engaged in bulk transfer, sending data as fast as allowed by TCP. Following [6], denote the window size of the i^{th} sender by W_i. We keep track of each sender's window separately. The queue equation (2.3) is replaced by:

$$\frac{dq(t)}{dt} = \begin{cases} (\sum_{i=1}^{N} \frac{W_i(t)}{R_i(q(t))}) - c & \text{when } q(t) > 0, \\ max((\sum_{i=1}^{N} \frac{W_i(t)}{R_i(q(t))}) - c, 0) & \text{when } q(t) = 0. \end{cases} \tag{2.10}$$

The connections need not all share the same fixed propagation delay, as the R_i terms indicate. We use a separate copy of (2.2) for each window W_i, and a separate timer variable for each connection.

2.2.2 Towards a Discrete-Time Model

In practice, the differential equations in the continuous-time model can be solved by numerical integration using a simple scheme such as Euler's method. The main requirement is that the time step be small enough to reflect system behavior occurring on the smallest time-scale. In the given network scenario, the smallest time scale is the spacing between packets, which, as dictated by the bottleneck link, is equal to $1/c$. We will now show that the use of $1/c$ as a time step allows us to

express the model in a simplified way as a discrete-time system.

The Euler numerical integration step for the W equation (2.2), is

$$W_{k+1} = W_k + h\frac{1}{a + q_k/c}, \tag{2.11}$$

where h represents the time step. Setting $h = 1/c$, we have

$$W_{k+1} = W_k + \frac{1}{ac + q_k}. \tag{2.12}$$

For the q equation (2.3), neglecting the $q(t) = 0$ term for the moment, the Euler step is

$$q_{k+1} = q_k + h\left(\frac{W_k}{a + q_k/c} - c\right), \tag{2.13}$$

which simplifies to

$$q_{k+1} = q_k + \frac{W_k}{ac + q_k} - 1, \tag{2.14}$$

using the time step $1/c$. Note that in the above equation, the middle term represents the flow in per time step and the final term, the -1, represents the flow out per time step. In case the queue is empty, we omit the -1 term as no packets will leave within that time step:

$$q_{k+1} = \begin{cases} q_k + \frac{W_k}{ac+q_k} - 1 & \text{when } q_k > 0, \\ q_k + \frac{W_k}{ac+q_k} = \frac{W_k}{ac} & \text{when } q_k = 0. \end{cases} \tag{2.15}$$

Finally, it is easy to show that this choice of time step returns the equation for x back to its original form.

$$x_{k+1} = wq_{k+1} + (1 - w)x_k. \tag{2.16}$$

Note, we are still using the assumption that RED is updating the average queue size as if packets were arriving at the queue at a rate of c.

Summarizing these equations, we have a discrete-time model representation of the original

continuous-time system that gives an intuitive description of the TCP dynamics on the shortest time-scale. We can utilize the same overall procedure as the continuous-time model (see Fig. 2.2), substituting the maps (2.12), (2.15) and (2.16) above for the differential equations (2.6), (2.7) and (2.8).

2.2.3 The Discrete-Time Model

We can derive a similar but even more simplified discrete-time model by making the assumption that the system is saturated. As with the continuous-time model, the discrete-time model uses the three state variables W, q, and x and consists of two primary components. The first component is a discrete-time map used to model the change in each of the state variables during the additive increase phase. The second part is an impulse that works similar to the one in the continuous-time model – it accounts for the sudden adjustment in the window size due to a dropped packet.

Modeling the Additive Increase

In practice, after an initial transient, the bottleneck link in the scenario we model will reach saturation and a queue will form at the router. Packets leave the queue at evenly spaced intervals of length $\delta = 1/c$, where c is the capacity of the outgoing link. For instance, if the link has a capacity of 200 packets per second, the outgoing packets will be spaced (front-to-front) by $1/200s$ or $.005s$.[3] When these packets reach their destination, acknowledgment packets are sent out, with the same spacing. The arrival of the acknowledgment packets at the sender results in the sending of new data packets. Receipt of an acknowledgment frees one packet position in the TCP send window, W, and causes W to increase by $1/W$. Typically this means that one packet is sent for each arriving acknowledgment. Occasionally, when W reaches the next integer value, two packets are sent upon receipt of an acknowledgment. In other words, usually one packet arrives at the queue every δ seconds while one packet leaves the queue every δ seconds, resulting in no net change in queue length. But occasionally, 2 packets arrive at the queue in a δ-second interval while still only one leaves. This is the primary cause of queue length increase in this scenario.

[3]Throughout this dissertation we assume uniform packet size unless specified otherwise.

The description above refers to what is sometimes known as the *self-clocking* nature of TCP. That is, a TCP sender has the ability to determine the limiting capacity in its flow path and send its packets at that rate. We take advantage of this behavior in our model, by employing δ as the time step in a discrete-time model of this system.

The time step is given by:

$$t_k = t_0 + k\delta,\ k = 0, 1, 2, \ldots \tag{2.17}$$

where the time t_0 marks the start of saturation. Our equation for W follows directly from the rules of TCP:

$$W_{k+1} = W_k + \frac{1}{W_k}. \tag{2.18}$$

We express the change in the queue length as follows:

$$q_{k+1} = q_k + \frac{1}{W_k}. \tag{2.19}$$

While this yields fractional-valued queues, it does result in the queue length increasing by 1 each time W increases by 1, as described above, and is simpler to express than the more realistic alternative. The equation for x is the actual equation used by RED (the assumption of saturation means packets arrive each time step and so x is updated each time step):

$$x_{k+1} = wq_k + (1 - w)x_k. \tag{2.20}$$

The round-trip time, from (2.1), is:

$$R(q_k) = a + \frac{q_k}{c} = \left(\frac{a}{\delta} + q_k\right)\delta. \tag{2.21}$$

In multiples of the time step δ, the round-trip time is thus best approximated by:

$$m_k = nint(a/\delta + q_k)$$

where "nint" means the nearest integer.

We initialize the additive-increase phase of the model at the start of saturation, or, in other words, at the onset of queuing. Just prior to this time, there is an exact balance between the flow into and out of the router. The flow in is equal to $W(t_0)/R(q(t_0)) = W(t_0)/(a + q(t_0)/c)$. Since $q(t_0) = 0$ at this point, the flow in is $W(t_0)/a$ and the flow out is c. Thus the initial conditions for the discrete-time model are $q_0 = 0$ and $W_0 = ca$. (Note that W_0 corresponds to the so-called *bandwidth-delay product* which is often used to estimate buffering for network resources.)

Modeling the Multiplicative Decrease

We model the multiplicative decrease here in much the same way as in the continuous-time model but with one additional element. As in the continuous-time model, we assume a drop occurs as soon as x exceeds q_{min}: If $x_k > q_{min}$ then at time $k + m_k - 1$, replace (2.18) with

$$W_{k+m_k} = \frac{1}{2} W_{k+m_k-1}, \tag{2.22}$$

which reflects the delay of 1 round-trip time in drop notification. We take advantage of the packet-level timing of this model to incorporate an additional aspect of what occurs to a network sender following a window cut. In a network, after cutting its window from W to $W/2$, the sender will not send packets for $W/2$ time steps. This reflects the fact that acknowledgments for packets in the half of the window that is no longer accessible will not trigger the transmission of any new packets. As a result, the queue length will decrease. We model this by using the following equations for the next $W/2$ time-steps where W is the window size just prior to the window cut:

$$W_{k+1} = W_k, \tag{2.23}$$

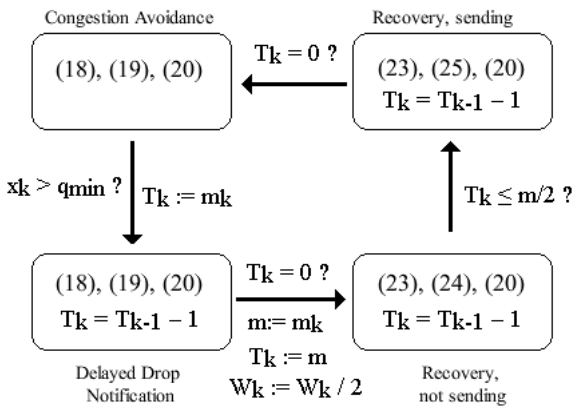

Figure 2.3: Hybrid systems view of the discrete-time model.

$$q_{k+1} = q_k - 1. \tag{2.24}$$

Note that W_k remains fixed following the rules of Fast Recovery/Fast Retransmit, as was done in the continuous-time model. Next we continue to hold W_k constant and now also hold q_k constant to reflect the rest of the recovery period. In other words, the sender is sending packets but not increasing its window, hence the queue length should remain fixed:

$$q_{k+1} = q_k. \tag{2.25}$$

As with the continuous-time model, we can present the discrete-time model in hybrid systems form as pictured in Fig. 2.3. Each rounded rectangle contains the numbers of the equations to be used (without chapter prefix), and a timer variable, T_k, is utilized. The model begins in the congestion avoidance phase, obeying equations (2.18) , (2.19), and (2.20). Once x exceeds q_{min}, the model assumes a drop occurs, and it transitions to the delayed drop notification phase. In this phase, it continues to obey the original set of equations, but now it also utilizes the count-down timer, T_k, which expires one round-trip time from the time x exceeds q_{min}. (The round-trip time is given by m_k, which was defined following equation (2.21).) Once this timer expires, the model cuts W_k in half, initializes a new count-down timer at the value of the present round-trip time (which we label m) and transitions into the recovery phase in which no packets are being sent. In this phase of the recovery, replace equations (2.18) and (2.19) used in the earlier phases, with (2.23)

and (2.24), respectively. As mentioned above, this phase should last $W/2$ time steps. It is not difficult to show that for a single sender, this $W/2$ is equal to $.5m$ though we leave this result out for the sake of brevity. When the timer has been reduced by $.5m$, the model moves into the final state – recovery with transmission. Here it uses equations (2.23), (2.25) and (2.20). Finally, when the count-down timer expires, the model returns to the congestion avoidance phase.

Modeling Multiple Senders

In order to account for multiple senders, we must modify the discrete-time model. Consider the case of two senders, with window sizes of $W^{(1)}$ and $W^{(2)}$. Typically the senders will alternate sending windows of packets. As a result, acknowledgments will arrive at a given sender in bunches. Thus, the $1/W_k$ increase per time step δ indicated in (2.18), will not actually occur each time step for each sender. However, each connection's window will still increase by 1 per round-trip time. One way to reflect this in the model is to choose a time step of 2δ, use the original W equation (2.18) for both senders, and change the queue equation to

$$q_{k+1} = q_k + \frac{1}{W_k^{(1)}} + \frac{1}{W_k^{(2)}}. \tag{2.26}$$

For n senders, use a time step of $n\delta$ and include the term $1/W_k^{(j)}$ in the queue equation for each sender $j = 1, 2, ..., n$.

2.2.4 Results

In this section we compare results obtained from applying our models to the network set-up mentioned above with those obtained using the *ns* simulator [15] on an equivalent network. We used the following settings (refer to Sec. 2.2 for variable definitions): $a = .01$ s, $c = 1.5$ Mbps, $q_{min} = 50$, $q_{max} = 100$, $p_{max} = .1$, $w = .003$, and packet size = 1000 bytes.

We implemented the models in MATLAB. Since we do not model the slow-start behavior of TCP, we begin the comparison after an initial transient has ended. In Fig. 2.4, the top plot shows a comparison of the queue length in the continuous-time model with the results from the

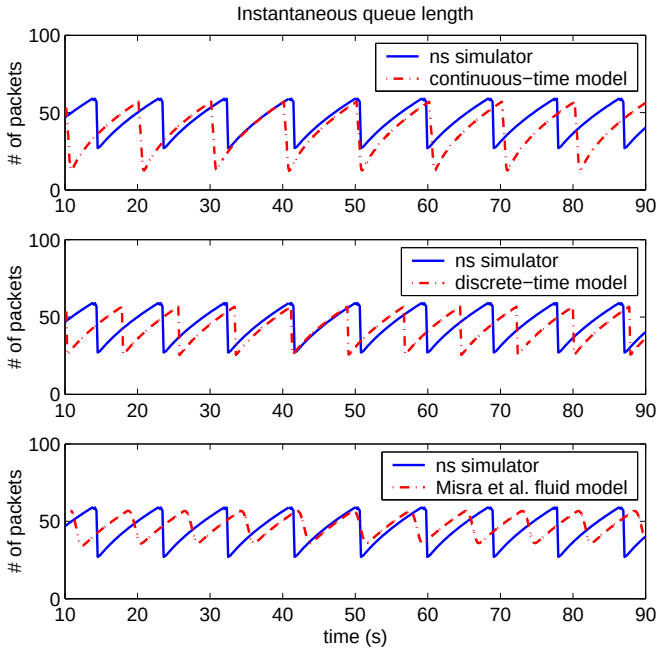

Figure 2.4: Comparison for one sender

ns network simulator. The middle plot shows the results of the discrete-time model compared to the simulator. The bottom plot compares results from the fluid model of Misra et al. [6] with the simulator.

Despite their lack of a stochastic component as well as several of the details of the TCP implementation, our models show good qualitative agreement with the simulator. We believe that the models have captured the essential behavior of this network under these flow conditions. The use of an impulse helps the models account for the sharp declines in the queue length caused by drop events. Lacking this feature, the fluid model of Misra et al. does not perform as well in this case of one sender.

One of the continuous-time model's inaccuracies is that the queue length drops down too far. Lacking the no-send phase of the recovery that was included in the discrete-time model, the continuous-time model's queue does not drop as much during the recovery period. This in turn causes x to stay artificially high, high enough so that after the recovery is complete, it is still above q_{min}. As a result, a subsequent drop occurs, followed by an additional window cut and recovery period, all resulting in a further drop in the queue length. The consecutive recoveries' negative

contribution to queue length outweigh the positive contribution on the queue length of the lack of a no-send phase. The discrete-time model does not have this problem due to the use of the no-send recovery phase.

Another discrepancy found between our models and the simulator is in the period length of the queue oscillations. The discrete-time model's period is too short because of the assumption of an instantaneous drop. At these settings, it takes, on average, around one second from the time RED turns on until a packet is dropped. In Sec. 2.3 we introduce a method to estimate this delay and incorporate it in the models. The continuous model's period is too long because of the dual recovery periods which outweigh the effect just mentioned.

Fig. 2.5 shows a similar comparison except now we consider two senders on the network with identical fixed delays. The network has become harder to model as evidenced by the increased randomness in queue variations. Around the region $60 \leq t \leq 80$, the simulator's behavior is analogous to its typical behavior in the one-sender case. Hence, the statements made in the previous paragraph apply there. What is occurring in that region is one drop for each sender for each drop event. For much of the rest of the time interval shown, the variation is due to different drop occurrences such as two drops for one sender, only one drop, three drops and others. This suggests a modeling approach more directed towards statistical agreement than short-term qualitative agreement for cases of anything but a small number of senders.

For larger numbers of senders the model of Misra et al. shows better agreement with the simulator than our models do. Thus our models are advantageous mainly in cases when the bottleneck link congestion is due to a small number of senders. We propose extensions to our models in the following section to improve their performance.

2.3 Extensions

Model simplicity is important if we are interested in being able to concisely describe a system, and for amenability to mathematical analysis. On the other hand, for the purpose of traffic state estimation, the key attributes we desire in a model are accuracy and computational efficiency.

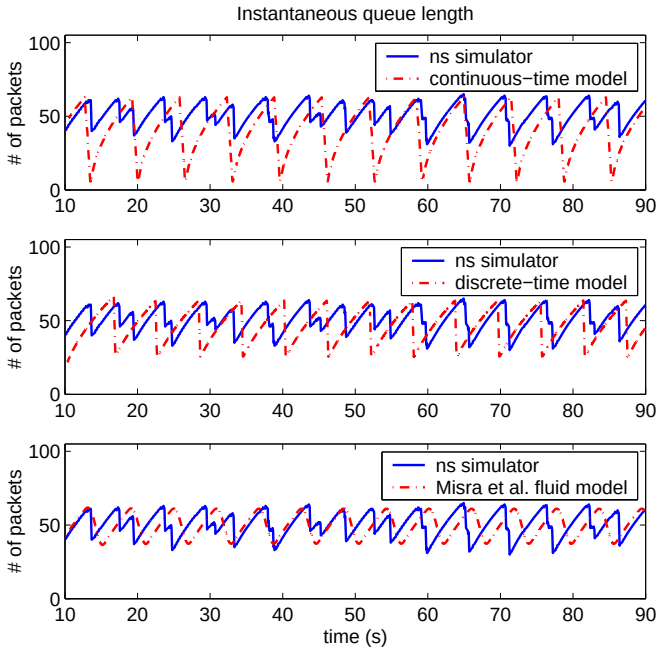

Figure 2.5: Comparison for two senders

With this in mind, we introduce extensions to the models to improve their accuracy while still keeping the models simpler than a full-fledged packet-level simulator. The primary modification we consider is to implement a more realistic packet dropping mechanism within the models, one that more closely resembles RED.

2.3.1 Estimating the Interdrop Time

As was mentioned in the previous section, one of the discrepancies between the models and simulator is that the models assume a drop occurs as soon as RED turns on. In reality, there tends to be a delay of about one second under the settings we are using. Here we describe a method to estimate this delay that we can incorporate into the models while still keeping them deterministic.

In practice, RED has an added layer of complexity. Namely, it has been designed so that effectively, the drop time is chosen from a uniform distribution on a time interval of length $1/p$, given a constant RED drop probability p. As implemented in the *ns* simulator [15], an additional *wait* parameter is used, which causes this time interval to lie between $1/p$ and $2/p$ from the time of the last drop. (This spacing also applies to the time between RED turning on and the first drop.)

As a result, the expected time between drops is $3/(2p)$.

More precisely, when x exceeds q_{min} and RED turns on, it begins counting packets in order to determine when a packet drop should occur. The actual drop probability used by RED, which we denote p_d, is given by:

$$p_d(p,k) = \begin{cases} 0 & \text{when } k < 1/p, \\ p/(2-kp) & \text{when } 1/p \leq k < 2/p, \\ 1 & \text{when } k \geq 2/p, \end{cases} \tag{2.27}$$

where k is the number of packets that have arrived since RED turned on, or since the previous packet drop. Let K be the random variable representing the number of packets that arrive between drops, or between RED turning on and the first drop. If x is constant, then by equation (2.9), p is constant, in which case it can be shown that K is uniformly distributed between $1/p$ and $2/p$. This means $E[K] = 3/(2p)$.

Of course x is generally not constant, and in our model we make the rough approximation that $E[K]$ is the solution to the equation $k_{drop} = 3/[2p(x_{k_{drop}})]$, where x_k is the value of x when the packet counter reaches k. We further approximate x as a linear function of k between packet drops. Rewriting (2.4) as

$$x_k = x_{k-1} + w(q_k - x_{k-1}), \tag{2.28}$$

we replace $q_k - x_{k-1}$ by $q_0 - x_0$:

$$x_k = x_0 + kw(q_0 - x_0) \tag{2.29}$$

Assuming that x_k remains between q_{min} and q_{max},

$$p(k) = a_1 k + a_2 \tag{2.30}$$

where a_1 and a_2 are constants determined by equations (2.9) and (2.29).

Since we want to find k_{drop} such that $k_{drop} = 3/[2p(x_{k_{drop}})]$, we substitute (2.30) into this

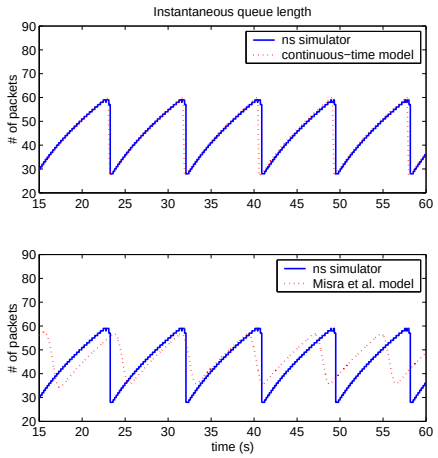

Figure 2.6: Comparison of models with *ns* simulator for the one-sender scenario described in Sec. 2.2.4.

equation. Solving the resulting quadratic produces the following solution:

$$k_{drop} = \frac{-a_2 + \sqrt{a_2^2 + 6a_1}}{2a_1} \tag{2.31}$$

If $a_2^2 + 6a_1 < 0$, then since $a_1 < 0$, $p(k)$ is a decreasing function of k, and we assume that it becomes zero before another drop occurs.

We change the model as follows. When RED turns on, we compute k, the expected time until the next packet drop. A timer variable counts down this amount and when it expires, we consider the drop to have occurred and follow the same procedure as in the previous models. This scheme can be implemented for both the continuous-time and discrete-time models. In Fig. 2.6, we present results of its application to the continuous-time model. The figure shows that using this extension, the model is able to very closely reproduce the behavior of the simulator. However, the accuracy of this approach diminishes as the number of senders increases. For even greater accuracy, we fully implement RED as it is executed in the *ns* simulator [15]. This is described in the following section.

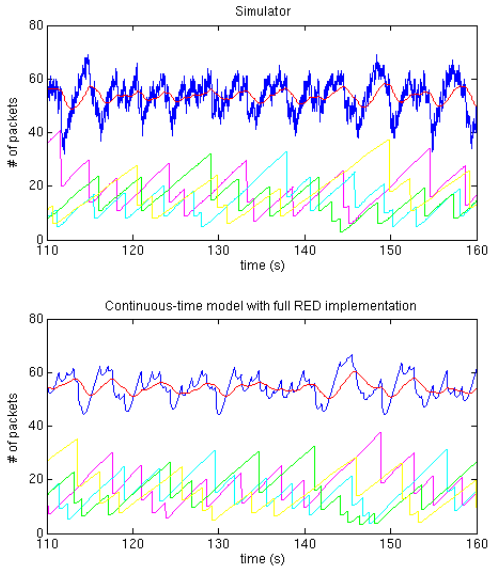

Figure 2.7: Comparison for four senders between the *ns* simulator and a realization of the continuous-time model using the full RED implementation. In both plots, the upper two curves represent the instantaneous and exponentially averaged queues. (The exponentially averaged queue is the curve showing less variation.) The lower four curves in both plots represent the window sizes of the four senders.

2.3.2 Full Red Implementation

In order to fully model RED, we begin by following the procedure laid out in the previous section up to equation (2.27). At that point, a uniform random number between 0 and 1 is generated and if this number is less than p_d, a drop occurs. In the case of multiple senders, the probability of a given sender experiencing the drop is proportional to that sender's share $(\frac{W_i}{R_i})/(\sum_{j=1}^{N} \frac{W_j}{R_j})$ of the overall flow. This introduces a greater degree of complexity and a stochastic component to the model. Since our ultimate goal is network traffic state estimation, our main priorities are model accuracy and computational efficiency.

We applied this model to the network described earlier, but now with four senders at varying fixed propagation delays from the receiver. Fig. 2.7 shows that there is good qualitative agreement between the model and the simulator. The additional jitter in the simulator's queue is due to packet level activity. The model shows good statistical agreement with the simulator as well. Fig. 2.8 shows a comparison of queue histograms. Note that the model correctly captures the range in queue values, which can be used to calculate the range of round-trip times.

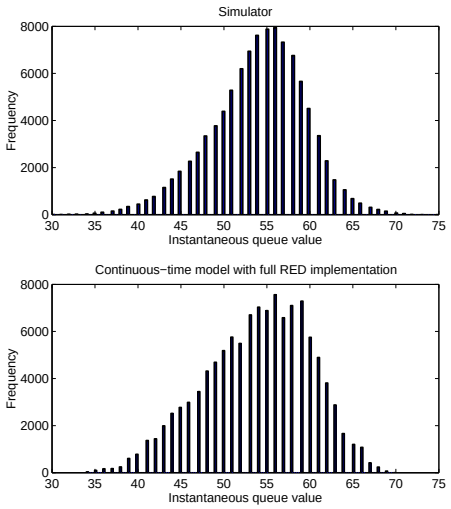

Figure 2.8: Comparison of instantaneous queue histograms for the four-sender network. The model is using the full RED implementation. Model queue values, which may be fractional, have been rounded for the purposes of comparison.

In order to evaluate the model's responsiveness to network changes, we tested it on a network with two classes of flows, one of which turns off for part of the run. The classes both consist of five bulk transfer senders, but the fixed propagation delay is different for each class: 20 ms for class 1 senders, 35 ms for class 2 senders. The class 2 senders turn off from 75 to 125 seconds after the start of the run. In practice, this effect could be caused by the sender-side application not having data to send for that stretch of time. Another possibility could be that the class 2 senders are temporarily rerouted.

The results, shown in Fig. 2.9, indicate that the model does a good job of capturing the changes in the network state caused by the senders turning off. This includes the sudden drop in the queue just after the senders turn off. The model also reproduces the overall decreased level of the queue and increased level of the class 1 senders' windows during the class 2 off period. Since both simulator and model have stochastic components, each plot represents a realization rather than the expected performance. Nevertheless, based on our observations of many realizations for both model and simulator, we expect the average performance to show good correspondence as well. We give statistics from sample realizations in Table 2.1, indicating the close statistical match between model and simulator. Note that the model is able to estimate the variance of the queue to within 10%. This implies that by using the model in conjunction with the round-trip time

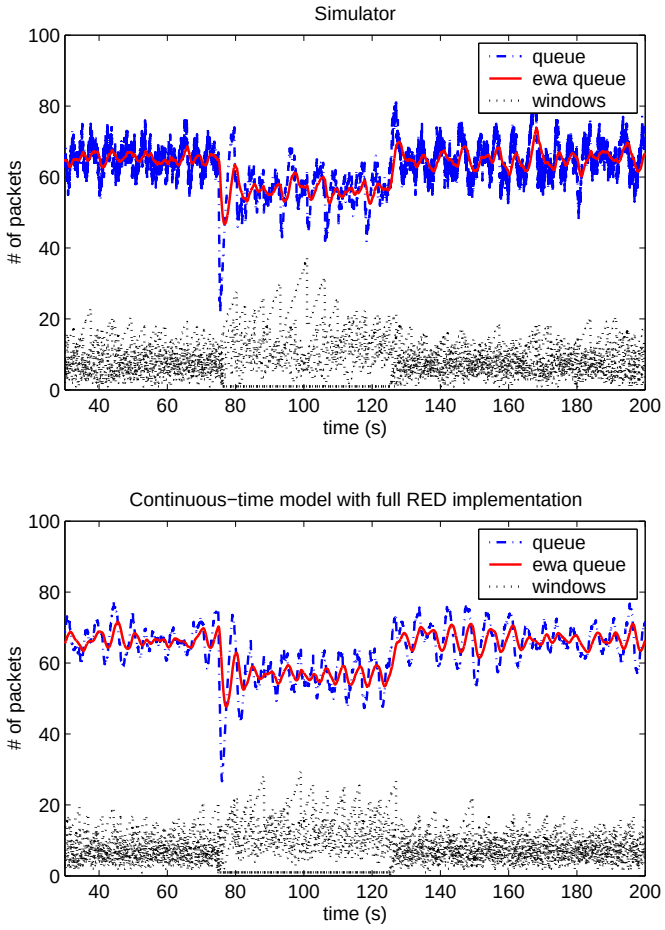

Figure 2.9: Comparison for two classes with five senders per class. Class 2 senders are off from 75 to 125 seconds.

equation (2.1), a reasonable estimate of the variance of the round-trip time can be obtained.

2.4 Towards Network State Estimation

Ultimately, we plan to use these models in a network traffic state estimation scheme. The problem can be posed as follows: Consider a network of N senders and one bottleneck router, whose state can be characterized by the sender window sizes and the router queue and exponentially averaged queue lengths. Given only a series of observations of some subset of the full network state, such as the window sizes and round-trip times of a small number of senders, what is our best estimate of the state of the system at the current time? That is, we must estimate all of the sender window sizes, using only the past history of some observed portion of the state of the network.

Table 2.1: Statistical comparison corresponding to Fig. 2.9.

Variable	**Quantity**	**Simulator**	**Model**
All Flows On			
q	mean	65.5	66.7
q	st.dev.	4.8	4.4
Half of the Flows Off			
q	mean	56.0	56.0
q	st.dev.	7.0	6.5

The estimation scheme we have in mind can be described as a particle filter or, more generally, a sequential Monte Carlo method [16, 17]. The basic idea is related to that of a Kalman filter. But unlike the Kalman filter, the particle filter makes no assumptions about the linearity of the model, or about probability distributions being Gaussian. Thus it is a better fit for our model and system.

The particle filter starts off with a random ensemble of states. It uses the observation to filter and re-weight the ensemble members based on how consistent they are with the observation. Then the ensemble members are advanced in time using the model, and the filter and re-weighting process repeats. We are currently testing this procedure in a variety of network scenarios.

2.5 Conclusion

We have developed two deterministic models with a discrete impulse that successfully capture the network behavior of TCP/RED in simple cases. Extending the models by adding a stochastic dropping mechanism consistent with RED, the models shows good correspondence in more complicated network situations, including those with larger numbers of senders and flows turning on and off.

Aside from network traffic state estimation described in Sec. 2.4, other works in progress include estimation of some dynamical properties such as period length, maximum sender window size and drop rate. Additionally, we are applying our models to networks subject to cross-traffic, including one that is prone to irregular behavior due to the presence of a source that sends large bursts of packets at regular intervals. This is discussed in Chapter 3.

Chapter 3

Bifurcations and Chaos in a Periodically Probed Computer Network

3.1 Introduction

Because of the complex interaction of protocols, multiple flows, delays and feedback effects, networks can exhibit complicated behavior. This includes even fairly small, simple networks. We model the interaction between two network connections sharing a bottleneck link: a TCP bulk-transfer flow and a flow that transmits periodic bursts of packets. This network exhibits complicated non-intuitive behavior as system parameters, such as the period of the bursts, are varied (see Fig. 3.6). We first apply a stochastic continuous-time model to this system. Results from the model show good qualitative agreement with those of the *ns* simulator. Next we show that a simpler deterministic version of the model is able to capture many of the main features of the behavior of this network as well. The simplest model we present is a one-dimensional map. This map is helpful in explaining the changes we see as the burst period is varied. In addition, using this map we can easily compute Lyapunov exponents, which give an indication of the system's predictability. We find chaotic behavior (i.e., positive Lyapunov exponents) for some burst period value ranges.

Periodic packet bursts are commonly generated by network analyzer tools [18], multicast protocol bandwidth probes [19] and online games [20]. In addition, they can also signify an attack. For example, in [21] a low-rate TCP denial of service (DoS) attack is described consisting of periodic bursts of packets. The burst size is taken to be long enough to exceed the round-trip time of any of the normal TCP connections in the network. This combined with Drop-Tail queuing in the router leads to synchronous timeouts in each connection. The bursts are then repeated periodically at intervals coinciding with the TCP retransmission timers. The effect is that the queue is flooded by the attack each time a connection attempts to resume transmission. This in

turn leads to abnormally low throughputs for the TCP flows. This scenario differs from ours by considering Drop-Tail queuing and the Reno variant of TCP, while we consider RED queuing and NewReno.

Several studies have found chaotic behavior in networks. In [22] the authors observe chaotic behavior in a network of two or more TCP flows sharing a bottleneck link. They show phase space plots revealing strange attractors and compute positive Lyapunov exponents. These results are shown only for *ns*-simulated networks and without a model upon which to base the results. Our experiments are on similar networks, but with the bottleneck link's outgoing router using RED for queue management rather than Drop-Tail. And as mentioned above, we also develop a series of models that capture the chaotic and other behavior of the periodically forced network we are studying. These models can be analyzed to gain deeper insight into the system behavior.

Ranjan et al. [8] model the interaction of TCP flows with a RED bottleneck buffer. Bifurcation diagrams with respect to various RED parameters are presented showing period doubling, border collisions and other interesting behavior. In addition, the authors calculate Lyapunov exponents, finding some regions of positive values, indicating chaotic behavior. The accuracy of the model is not clearly demonstrated, however, as no comparisons between the model and simulated or real network runs are given.

These and other works [23, 24, 25] indicate that it is not surprising to find chaotic network behavior. Our contribution is to consider a particular type of network traffic, namely traffic containing a periodically bursting flow, to model this traffic accurately and to use our model to explain the observed behavior.

3.2 Models

We consider a simple network with one TCP connection and one non-TCP connection that injects periodic bursts of packets into the network (see Fig. 3.1). The traffic from both flows passes through a bottleneck link. In the absence of the periodic, non-TCP source, the network experiences periodic behavior (see Fig. 3.3 prior to $t = 200$ seconds). In general, a periodic system that is periodically

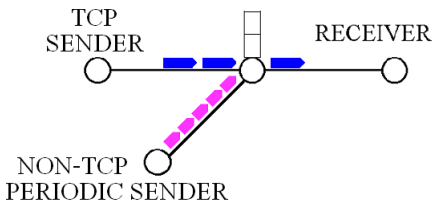

Figure 3.1: Network configuration

forced by an external source can have complicated behavior. For example, a forced undamped pendulum can exhibit periodic or chaotic motion.

We use a series of three models of decreasing complexity to represent and analyze this network. The first two models are described in greater detail in Chapter 2. The main difference between the models is that one handles drops stochastically while the other does so deterministically. We will refer to them as the *stochastic model* and the *deterministic model*. We briefly summarize both models here. The third model is a one-dimensional map that is empirically derived from runs of the deterministic model. It is described in the Results section.

3.2.1 Stochastic Model

The first part of the stochastic model handles the additive increase phase of TCP and can be summarized by the following equations:

$$\frac{dW(t)}{dt} = \frac{1}{R(q(t))} \tag{3.1}$$

$$\frac{dq(t)}{dt} = \frac{W(t)}{R(q(t))} - c \tag{3.2}$$

$$\frac{dx(t)}{dt} = wc(q(t) - x(t)). \tag{3.3}$$

where W is the TCP sender's congestion window, q is the instantaneous queue length at the router, x is the exponentially-weighted moving-average queue length, all in packets, R is the round-trip time in seconds, c is the bottleneck link capacity in packets per second, and w is the RED queue smoothing parameter.

The stochastic model replicates the procedure used by RED to determine when to drop packets. When x exceeds a threshold parameter q_{min}, RED turns on and begins counting packets in order to determine when a packet drop should occur. The actual drop probability used by RED, which we denote p_d, is given by:

$$p_d(p,k) = \begin{cases} 0 & \text{when } k < 1/p, \\ p/(2-kp) & \text{when } 1/p \leq k < 2/p, \\ 1 & \text{when } k \geq 2/p, \end{cases} \tag{3.4}$$

where k is the number of packets that have arrived since RED turned on, or since the previous packet drop, and p is given by:

$$p(x) = \begin{cases} 0 & \text{when } x < q_{min}, \\ \frac{x-q_{min}}{q_{max}-q_{min}} p_{max} & \text{when } q_{min} \leq x \leq q_{max}\ , \\ 1 & \text{when } x > q_{max}. \end{cases} \tag{3.5}$$

q_{max} and p_{max} are additional RED parameters.

Denoting the continuous-time system of equations (3.1), (3.2), and (3.3) by $\mathbf{A}$, we summarize the stochastic model in hybrid systems form as in Fig. 3.2, where T is a timer variable. Our model begins in the congestion avoidance phase, following the set of equations, $\mathbf{A}$. Once x exceeds q_{min}, the variable k begins increasing from 0 by 1 unit per time step to represent the number of packets that have arrived since RED turned on.[1] At each time step, a uniform random number between 0 and 1 is generated. If this random number is less than p_d as determined by equation (3.4) then a packet drop occurs. Next, the model goes into to the delayed drop notification phase. In this phase, it continues following the original set of equations but now it also utilizes a count-down timer, T, which expires one round-trip time from the time of the drop. Once this timer expires, the model goes into the recovery phase, first cutting W in half and initializing a new count-down timer at the value of the present round-trip time. In the recovery phase, equations (3.2) and (3.3)

[1] We are referring to the time step of the numerical ODE solver, which is set to $1/c$ and thus corresponds to the spacing in time between successive packets.

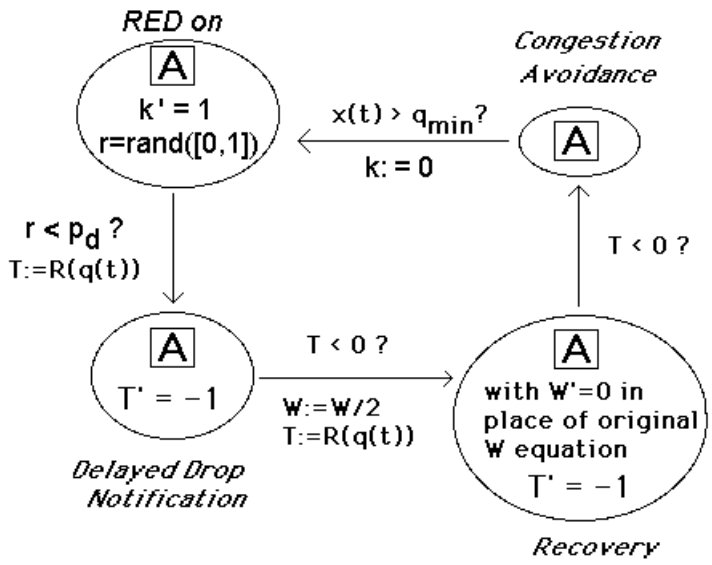

Figure 3.2: Hybrid systems view of the stochastic model.

still hold with W held fixed. Finally, when this count-down timer expires, the model returns to the congestion avoidance phase.

3.2.2 Deterministic Model

The deterministic model is similar to the stochastic model. Both models use equations (3.1), (3.2), and (3.3) and the procedure outlined in the last paragraph of Sec. 3.2.1 and Fig. 3.2. But the two models handle drops differently. In practice, RED is designed to have an expected drop-time spacing of $3/(2p)$ where p is defined in equation (3.5). In our deterministic model we use this $3/(2p)$ value to estimate the expected time until a drop occurs once x exceeds the threshold q_{min}. (See Chapter 2 for more details.) A timer variable counts down this amount, and when it expires, we consider the drop to have occurred and follow the same procedure as in the previous model.

In both models, we account for the periodic source by adding the burst amount to the instantaneous queue at the time the burst would reach the queue. In practice, the exponentially averaged queue would update as each packet in the burst arrived at the queue. In addition, the TCP sender would see a gap in its packet acknowledgments of duration equal to the size of the burst. To keep our models simple, we neglect both of these effects.

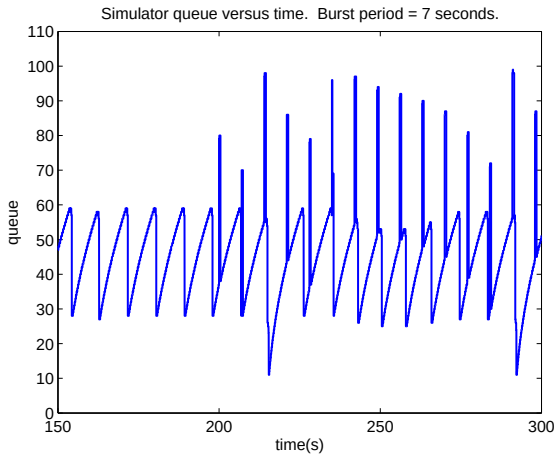

Figure 3.3: Instantaneous queue versus time for a simulator run. The vertical spikes are the bursts from the periodic source, which begins transmitting at $t = 200$ seconds.

3.3 Results

We begin by simulating the network (using the *ns* simulator) with the periodic source sending bursts of 40 packets every 7 seconds starting at $t = 200$ seconds. Fig. 3.3 shows the instantaneous queue as a function of time. The evenly spaced vertical spikes represent the periodic packet bursts. Note that prior to the bursts, the queue has nearly periodic behavior. (It is not strictly periodic because the randomness in the drop timings caused by RED leads to a slight variation in the cycle length over time.) In this figure the bursts are seen to land at various locations within the cycle. When they land late in the cycle, they cause packets to be dropped earlier than they otherwise would have. To see this, compare the peak values of the cycle at $t = 250$ seconds with the cycles before $t = 200$ seconds. The bursts may also cause a double-window cut, as they do at $t \approx 215$ and $t \approx 293$ seconds.[2]

Fig. 3.4 plots the queue versus time on a much longer time scale. Note that the queue appears to switch between different modes, which we name *plateau*, *downhill*, and *uphill*. Because the simulated network is stochastic, it is difficult to show close model-to-simulator correspondence between individual trajectories from each. Rather than make such local comparisons, we compare the global behavior of the network for the model and simulator as a function of the burst period. In order to accomplish this, we first define *burst-time queue values*.

[2] A double-window cut means two packets from separate windows were dropped, and so the sender treats this as if two separate drop events have occurred.

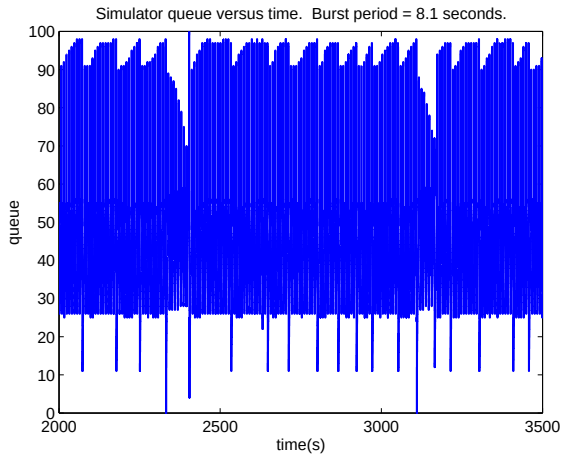

Figure 3.4: Instantaneous queue versus time for a simulator run, longer time scale.

The burst-time queue is the queue value just prior to each burst's packet arrivals at the router. An example of the burst-time queue value as a function of time is shown in the lower two panels of Fig. 3.5 for two different burst periods. The upper two panels show the burst-time queue values superimposed over the queue time series for a segment of the run. The lower plots show only the burst-time queue values over the entire run. Note that for a burst period of $T_b = 7.88$ the burst-time queue value is usually concentrated around 54 with occasional deviations. We think of this as a noisy period-1 orbit. For $T_b = 8.98$, the dynamics are usually less concentrated. To reflect the noisiness, we can compute a histogram of the burst-time queue values for this burst period value. These histograms taken over a series of burst period values can then be combined into a *bifurcation histogram* as in Fig. 3.6. Each vertical slice represents a histogram for a given burst period value. Darker colors indicate higher counts. Because the queue in the simulator is integer valued, the histogram bins are chosen to correspond to integers.

Note how the behavior changes as the burst period T_b increases: At low T_b, between 7 and 7.6, the histogram is broad with up to 6 peaks (the dark regions). Examining the time series in a region of T_b with six peaks, we find that the orbit tends to visit these dark areas in a well defined order (although noisily). Thus we refer to this behavior as *period-six-like*. As T_b increases past 7.6, the histogram concentrates strongly into a single dark band (period-one-like behavior). Further increase of T_b past about 8 results in a broadened histogram with period-two-like behavior, followed by period-three-like behavior for $T_b \gtrsim 8.5$.

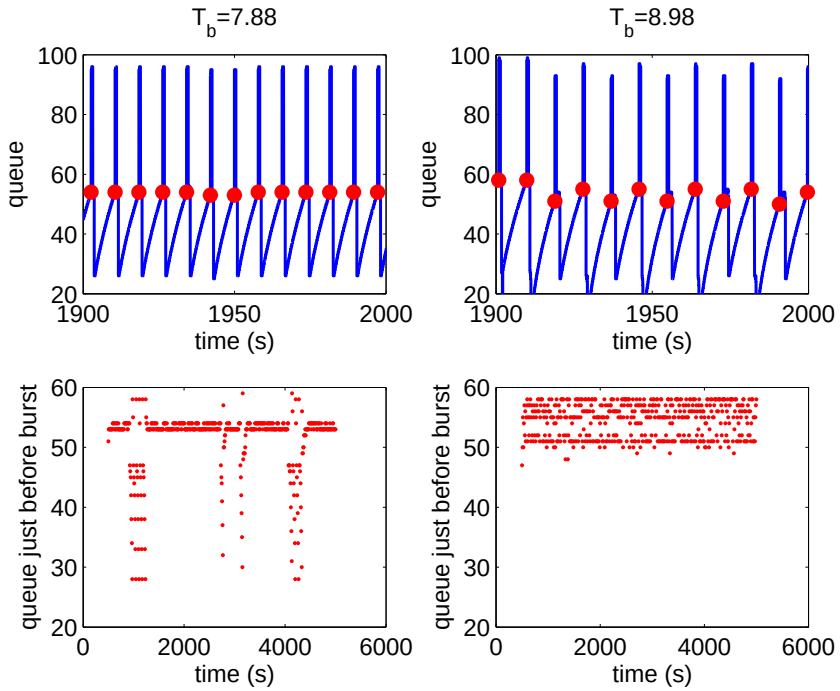

Figure 3.5: The burst-time queue is the value of the queue just prior to a burst. We use it to obtain a snap-shot of network behavior for a given burst period. The dots in the upper plots show the burst-time queue superimposed on queue time series. The lower plots show burst-time queue values only, and on longer time scales than the upper plots.

In Fig. 3.7 we show the bifurcation histogram generated by our stochastic model. (For the sake of comparison with the simulator, we use integer-centered bins for the model bifurcation histograms even though the model queues can take on fractional values.) Though the model does not completely capture the simulator's behavior, there are several qualitative similarities between the two. Both show the prominent upper dark band stretching across the range of burst periods. This band will be explained in the next section using a one-dimensional map.

Fig. 3.8 shows the bifurcation diagram generated by our deterministic model. Despite the many simplifications of this model, it is still capable of reproducing some of the key features from the simulator including the main dark band and the broadened histograms on the left side of the plot.

All of the bifurcation histograms we have shown so far are for a network in which there are 40 packets in each burst. Figures 3.9 and 3.10 show bifurcation histograms for burst sizes of 20 packets. Again, the bifurcation histogram generated by the deterministic model shows agreement

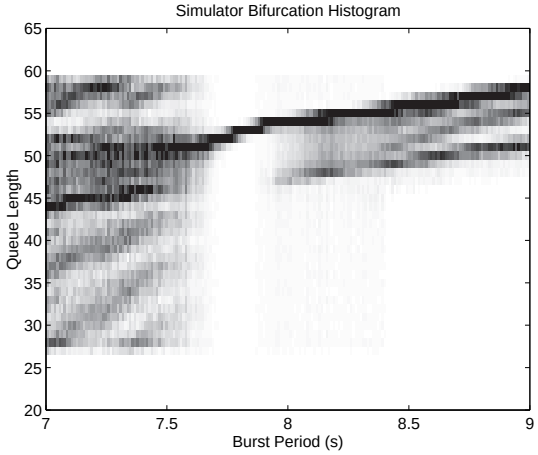

Figure 3.6: Simulator bifurcation histogram, burst size = 40 packets.

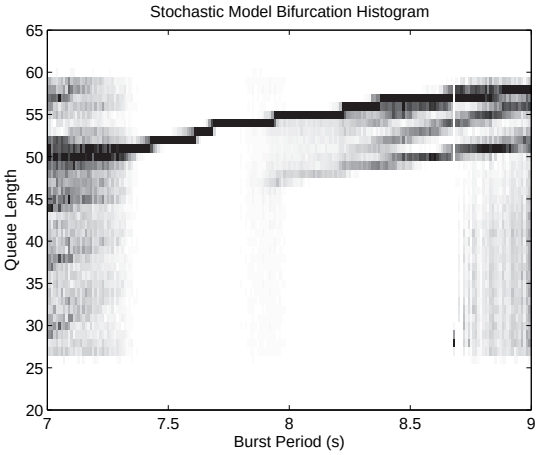

Figure 3.7: Stochastic model bifurcation histogram, burst size = 40 packets.

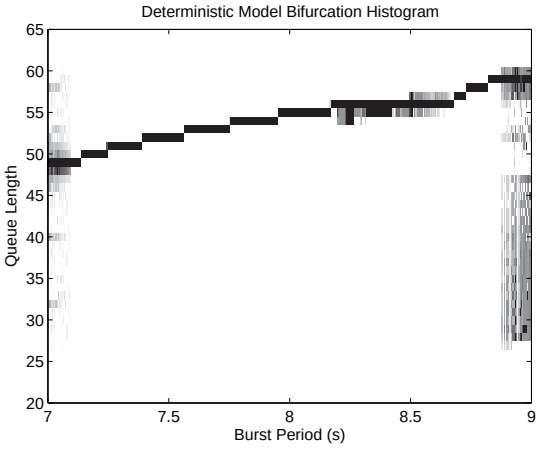

Figure 3.8: Deterministic model bifurcation histogram, burst size = 40 packets.

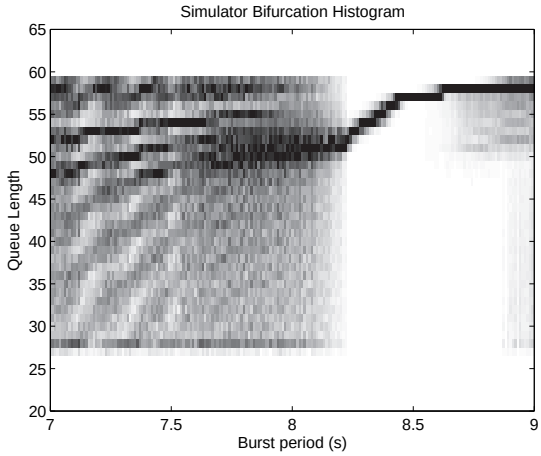

Figure 3.9: Simulator bifurcation histogram, burst size = 20 packets.

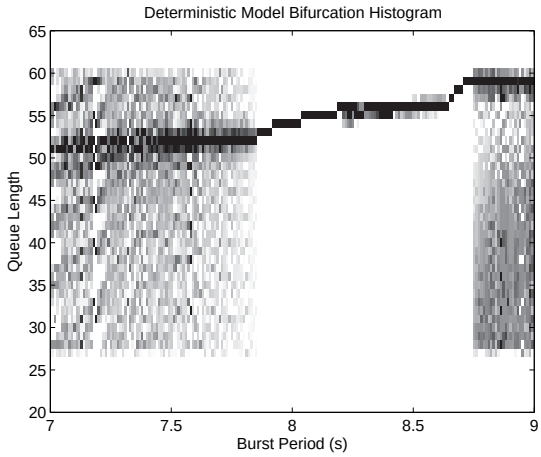

Figure 3.10: Deterministic model bifurcation histogram, burst size = 20 packets.

in key features with the one generated by the simulator. The broadened histograms extending from the left farther to the right as compared with the 40-packet burst case are evident in both simulator and model bifurcation histograms. This effect will be explained in the next section. The model has also captured the broadened histograms on the right hand side.

3.3.1 One-Dimensional Map

Aside from its utility in encapsulating network behavior at a particular parameter value, the burst-time queue can also be helpful in explaining the observed dynamics. If we know the queue length as a burst of probe packets is about to arrive at the router, we can define a burst-time map that tells

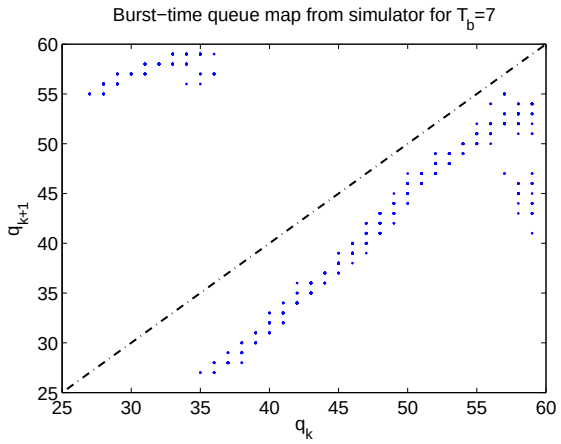

Figure 3.11: Simulator burst-time queue map, $T_b = 7$, burst size = 20. The diagonal is plotted for reference.

us what the queue length will be before the next burst arrives. In this way it gives us information about the impact of the burst on the dynamics. The bifurcation histograms for the simulator and models reveal only the long-term behavior of trajectories under this map. And, in the case of the simulator and stochastic model, the map will be noisy. Below we empirically derive a form of the map that it is well-defined (i.e. defined for all potential states) using the deterministic model. We show how a piece-wise linear approximation of the derived map can be used to reproduce and explain key features of the network dynamics.

Before proceeding, we exhibit a map empirically derived from the simulator in Fig. 3.11. To create this graph, we plotted the $k + 1^{st}$ value of the burst-time queue versus the k^{th} value over the course of a simulator run. Note that the values are all integers. Fig. 3.12 shows how a map generated by the deterministic model has a similar shape, and we will soon use it to explain the observed behavior. The procedure used to generate the deterministic model map is similar to the one that will be outlined in the next section. The fact that the deterministic model closely captures the shape of this map gives us confidence in using the model to derive the map we will ultimately use to analyze the network.

A limitation of working with q in the burst-time map is that a given q value can correspond to two different states: one in which the queue is increasing and the other in which the queue is decreasing. To circumvent this problem, we define a phase variable θ with respect to the original

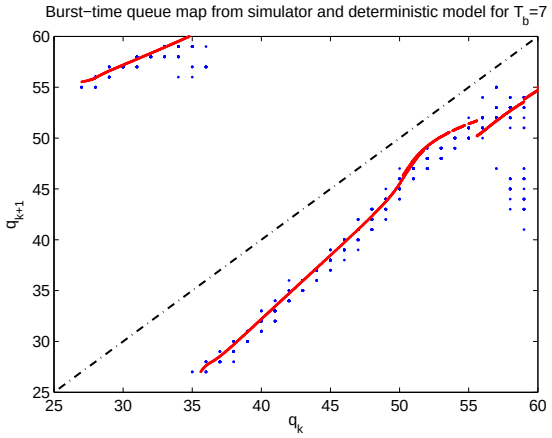

Figure 3.12: Simulator and deterministic model burst-time queue maps, $T_b = 7$, burst size = 20. The simulator values are the scattered points. The model values are the solid curve. The diagonal is plotted for reference.

unforced cycle. θ is defined to be in $[0, 1)$, where 0 corresponds to the start of the unforced cycle, and 1 corresponds to the end of the cycle period. Since the end of one cycle corresponds to the beginning of another, θ is an angle-like variable. That is, 0 and 1 can be identified.

Map Derivation

The map is derived as follows:

1. Define initial conditions based on the time-step used in the model.

2. For each possible initial condition run the deterministic model using the same burst start time for each run. This way we record the impact of the burst on each possible state in the original unforced cycle.

3. Record the burst-time map in the θ variable, that is, record the value of θ just prior to each burst.

4. Plot the second iterate of θ versus the first iterate (i.e. the initial value).

An example of this empirically derived map is shown in Fig. 3.13. We interpolate between undefined locations (recall, we used a finite set of initial conditions) so that the map is well-defined. Alternatively, we can fit a set of linear functions to this map. This will be described shortly. While

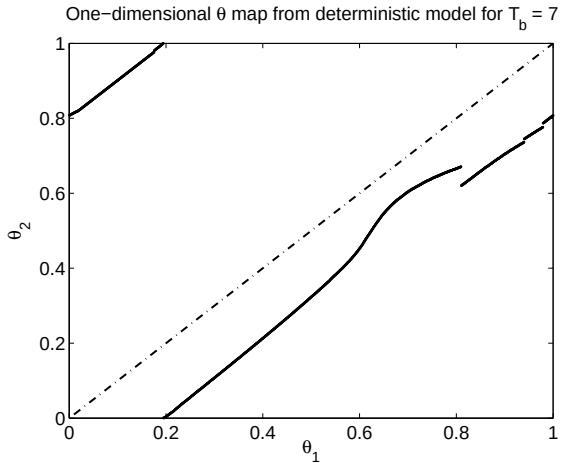

Figure 3.13: One-dimensional map derived from the deterministic model for a burst period of 7. The diagonal is plotted for reference.

the procedure to derive these maps can be repeated to determine maps for each burst period value, we only do it for two end-point values and interpolate to find the intervening maps. This is faster and proves to be a reasonable approximation. Before discussing the linear fit and map interpolation, we discuss some insights into the dynamics provided by the maps.

Insights

Here we will describe how the maps shed light on the bifurcation histograms and the network's behavior in general. Each map we consider in this section corresponds to a probe sending bursts of 20 packets. We start with the $T_b = 7$ map shown in Fig. 3.13. The piece in the upper left for which we see low values of θ mapping to high ones, corresponds to a burst occurring early in the cycle followed by another burst occurring later in that same cycle. This is possible here since the burst period of 7 is still less than the length of the cycle, which is roughly 8.6. We will see this piece shrink and then disappear as the burst period increases. The discontinuity at the right end of this piece reflects the boundary between points mapping to the end of the same cycle and those mapping to the beginning of the next one.

Most of the right-hand piece of the map ($\theta_1 \gtrsim 0.2$) is linear. The linear regions correspond to the burst not having a significant impact on the dropping of packets. Because the burst period is

shorter than the unforced cycle period, the bursts are effectively falling back within the cycle. This explains why the map lies below the diagonal. The vertical distance between the linear portions of the map and the diagonal represents the cycle differences.

The nonlinear part of the right-hand piece of the map, from about $\theta_1 = 0.6$ to $\theta_1 = 0.8$ corresponds to the burst landing at a point in the cycle at which it hastens the occurrence of the drop. The discontinuity at the right end of the nonlinear piece has to do with the way the deterministic model estimates the time until a drop occurs. The discontinuity in the queue caused by the arrival of the probe packets leads to a discontinuity in the drop time estimate and hence in the system response.

The same maps for burst periods of 8, 8.4, and 9 are displayed in Figures 3.14, 3.15 and 3.16 respectively. Note that the increase in T_b causes the map to shift upwards because the difference between the burst period and the unforced cycle period diminishes. At $T_b = 8$ the map crosses the diagonal at two points: $\theta \approx 0.62$ and $\theta \approx 0.78$. Dynamically, these correspond to fixed points (also called 'period one' in our previous discussion). Since the slope $d\theta_2/d\theta_1$ at $\theta \approx 0.62$ is larger than one, this fixed point is unstable. On the other hand, the fixed point at $\theta \approx 0.78$ is stable. The stable fixed point we see here explains the one seen in the bifurcation diagram in Fig. 3.10 in the range $7.8 \lesssim T_b \lesssim 8.2$. It does not emerge until the part of the map in the nonlinear region with slope less than one crosses the diagonal.

Later, at $T_b = 8.4$ (Fig. 3.15) the stable fixed point is gone, but the discontinuity in the map now straddles the diagonal. Because of the location of the map with respect to the diagonal, points to the right of the discontinuity map to the left and points to the left of the discontinuity map to the right. Furthermore, the nearby slopes on both sides of the discontinuity are less than one, meaning the map is not stretching in this region, hence not chaotic. We find that when this occurs, the orbit is a periodic orbit of period P greater than 1 but where all the P orbit points lie in a relatively small range of θ values. The period P depends on where the diagonal intersects the vertical discontinuity line joining the two branches of the map function curves at the discontinuity. This can also be seen in the bifurcation diagram. By $T_b = 9$, most of the map is above the diagonal

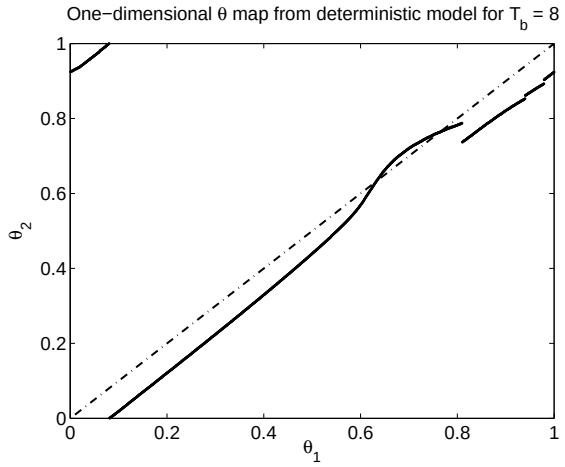

Figure 3.14: One-dimensional map derived from the deterministic map for a burst period of 8. The diagonal is plotted for reference.

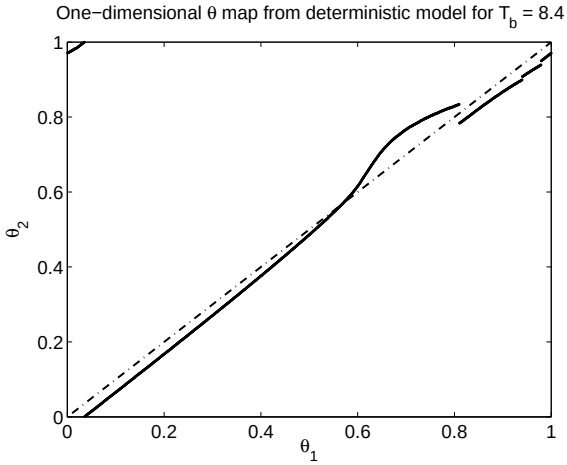

Figure 3.15: One-dimensional map derived from the deterministic map for a burst period of 8.4.

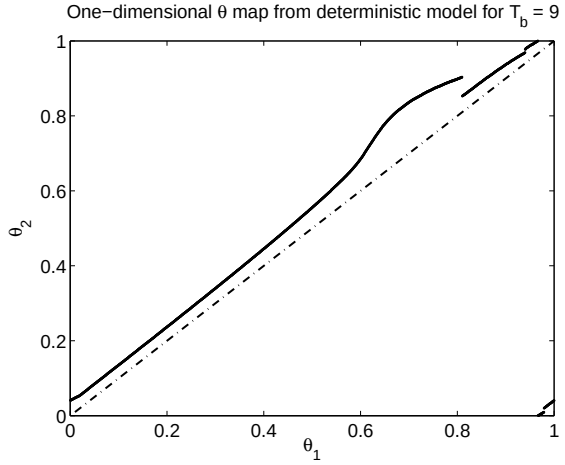

Figure 3.16: One-dimensional map derived from the deterministic map for a burst period of 9.

since the burst period exceeds the unforced period. For the same reason, the left-hand piece of the map present earlier is no longer present. A new discontinuous piece has been created in the lower right ($0.97 \lesssim \theta_1 \lesssim 1$). This corresponds to points that skip a cycle, that is, they map from the end of one cycle to the beginning of the cycle following the next one.

These maps can also help explain the differences between the bifurcation histograms for burst sizes of 20 and 40 packets. (See Figures 3.6 and 3.9.) Being smaller, the 20 packet bursts have a less significant effect on the network than the 40 packet bursts. For example, a 40 packet burst is more likely to lead to a drop than a 20 packet burst. The causing of a drop is reflected in the nonlinear portion of the one-dimensional map. This *bump*, evident in each of the 20 packet burst one-dimensional map plots, is present but even more pronounced in the 40 packet burst case. As the maps shift upward with increasing T_b, the bump nears, then crosses the diagonal, introducing the stable fixed point. The emergence of the stable fixed point signals the end of the broadened histogram region. Because the bump is larger in the 40 packet case, the bump will cross sooner than in the 20 packet case. Thus, the broadened histogram region starting on the left hand side ends sooner in the 40 packet case.

Piece-wise linear approximation

For an even simpler representation, we approximate the maps with three linear pieces as in Fig. 3.17. (Compare with Fig. 3.13.) This includes the left-most piece, which can be shifted down below the θ_1 axis for this purpose. This cleaner map avoids some artifacts introduced by the model, such as the small discontinuities near the right end (e.g., at $\theta_1 \approx 0.95$ and 0.97 in Fig. 3.13). It also allows for a more reliable calculation of Lyapunov exponents since we can take the slope of each piece rather than be concerned with possible jitter in the local slope calculations. In the neighborhood of the discontinuity at $\theta_1 \approx 0.8$, the map is in a form whose behavior has been looked at by [26]. The conclusions are similar to the ones made here. The motivating application for that work is a power electronic switching circuit.

We can use the 3-piece approximation to generate a bifurcation diagram. We do this by

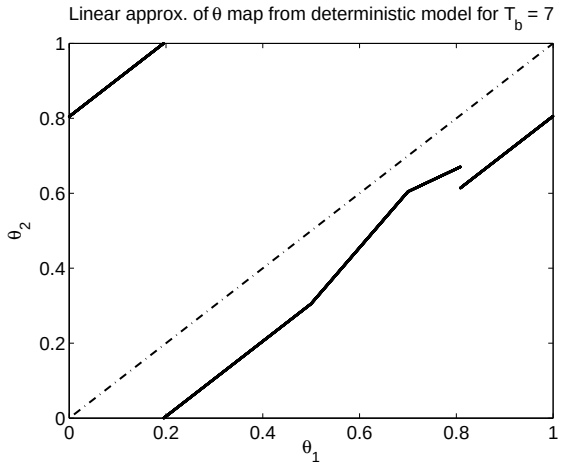

Figure 3.17: Three-piece linear approximation of the one-dimensional θ map derived from the deterministic model.

linearly fitting two empirically-derived maps, interpolating between them to find the intervening maps, and then iterating using each map. The results in Fig. 3.18 show a strong resemblance to the bifurcation diagram generated by the deterministic model in Fig. 3.19. (We use the bifurcation diagrams here rather than the bifurcation histograms since both are deterministic models.) The comparison shows that even a vastly simplified model is able to capture significant features of the dynamics.

We can also use the linear map to compute Lyapunov exponents as shown in Fig. 3.20. The positive ranges at either end indicate chaotic motion. The left-end region has two brief instances of negative Lyapunov exponents, corresponding to periodic windows within that region. At around $T_b = 7.8$ the values drop below 0 coinciding with the emergence of the stable fixed point discussed above.

3.4 Discussion

While the behavior observed above is interesting and may have implications for larger, more realistic networks, a natural question is what are the connections between this behavior and the performance of this network? In this section we consider how the behavior affects important performance measures including average round-trip time and packet loss rate. Because the network

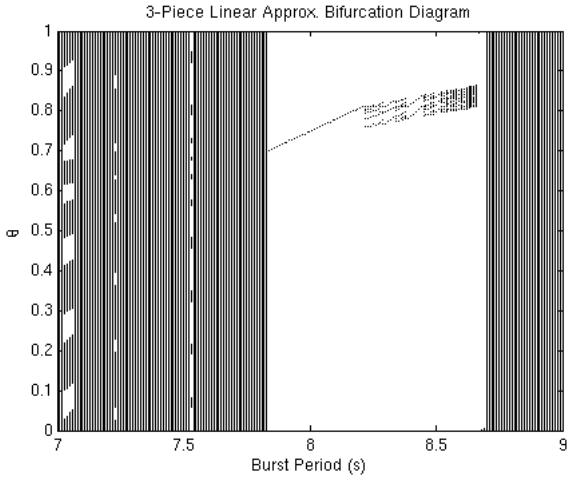

Figure 3.18: Bifurcation diagram generated using the 3-piece linear approximation to the one-dimensional θ map derived from the deterministic model

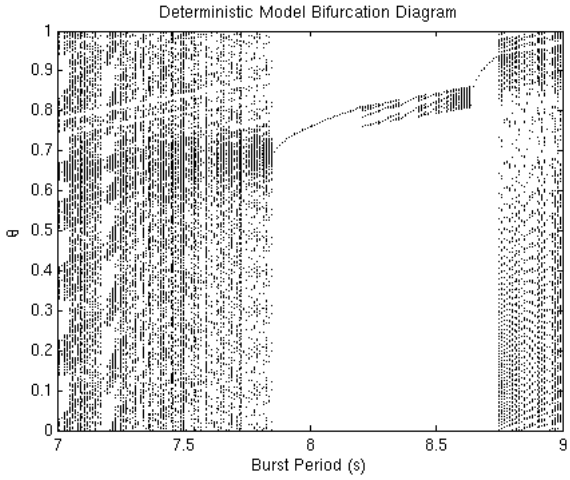

Figure 3.19: Bifurcation diagram generated using the deterministic model

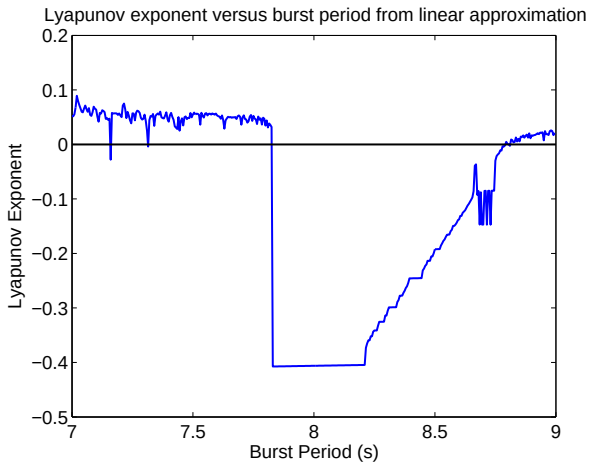

Figure 3.20: Lyapunov exponents computed using the 3-piece linear approximation.

is typically saturated in these experiments as evidenced by the positive queues, the throughput is usually equal to the bottleneck link capacity. So there is little variation in throughput as a function of T_b.

Fig. 3.21 shows the impact of the changing burst period on the average round-trip time and packet loss rate. To generate this plot, five simulations were run at each of 21 evenly spaced values of T_b. The average round-trip time and packet loss rate were computed for each run and averaged over all runs at a particular T_b value. The error bars indicate the standard deviation of the 5 runs at each setting. For the average round-trip time, the variation we see is very slight, less then 2% over the range of burst period values. Hence we conclude the average round-trip time is not very sensitive to the burst period. The drop rate shows more significant variation, as much as 16%. We explain some of this variation below.

As mentioned earlier, in the absence of the periodic bursts, the network is nearly periodic. The end of each cycle is marked by a dropped packet, which leads to a decline in the queue. The cycle lasts about 8.6 seconds at the network settings we use. At one drop per cycle, this corresponds to a drop rate of roughly 0.116. Under the periodic forcing, the situation closest to the unforced case is when the burst-time queue exhibits period-one-like behavior. That is, the bursts are coinciding with nearly the same queue value each time, and so the overall behavior is still nearly periodic. For the burst period values coinciding with this period-one-like behavior

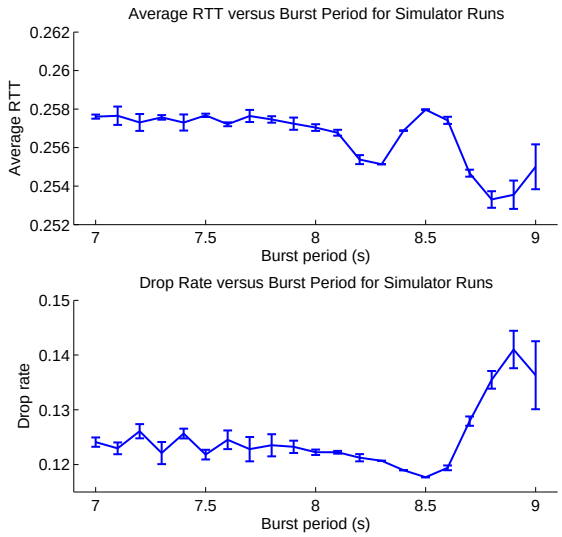

Figure 3.21: Average RTT and drop rate as a function of T_b. Values are averages over 5 simulator runs at each value of T_b. Error bars indicate the standard deviation of the 5 runs at each setting.

($8.2 \lesssim T_b \lesssim 8.6$), we see a drop rate closest to that of the unforced scenario.

At lower T_b values ($T_b \lesssim 8.2$), the burst-time queue behavior is less regular, but it does have some structure. As described earlier, the orbit of the burst-time queue tends to move through approximately six locations before cycling. One of these locations is near the peak queue value. Sometimes a burst landing near the peak queue can lead to a double-window cut, which means more than one packet is dropped in that cycle. This is why the drop rate is slightly higher at these T_b values, as compared with the period-one region.

For $T_b \gtrsim 8.6$, Fig. 3.21 shows that the drop rate increases. This is because at these settings, the orbit of the burst-time queue very frequently moves through a high queue location, resulting in a double-window cut, or sometimes even a timeout. This can be seen by noting the dark band near $q = 60$ and for $T_b \gtrsim 8.6$ in Fig. 3.9. Hence we see that the periodic source does impact the network in a way that depends on the particular value of T_b.

3.5 Conclusion

In conclusion, we have used a set of models to help analyze a network experiencing a periodically bursting transmitter. Even the simplest model we developed, a piecewise-linear one dimensional

map, is able to capture significant features of the network dynamics. We used bifurcation histograms to display the network's behavior as a function of burst period. Using the one-dimensional map, we were able to reproduce and explain many aspects of the bifurcation histogram. The piecewise-linear approximation allowed us to readily determine Lyapunov exponents, which indicated chaotic behavior along with periodic windows. Finally, we discussed the implications of varying the burst period on network quantities including average round-trip time and packet drop rate.

Chapter 4

Optimizing Transmission Network Layout

4.1 Introduction

In this chapter, we consider a variation of a well-known network optimization problem that has applications in the design of transmission networks. The Euclidean Non-Uniform Steiner tree problem (ENSTP) we consider is based on the minimal spanning tree problem (MSTP). The goal in the MSTP is to connect a set of nodes in a minimal cost tree.[1] In a Steiner tree problem, only a subset of the nodes require connecting. The other nodes may be used in the tree to help reduce the total cost, in which case they are known as Steiner nodes. In the ENSTP, all of the nodes are situated on a plane in which costs are associated with each location. The costs of the edges connecting the nodes depend on the costs of the locations through which the edges pass. This problem is relevant to situations in which a network must be constructed on a non-uniform cost landscape. For example, consider the problem of taking a set of locations in a city and connecting them in an underground cable-based communications network. If laying cable underground requires digging up the street, then it will usually be more costly to do so in a downtown location than in an industrial section on the edge of town. Other applications include transmission lines, printed circuits, heating and ventilation, water supply, fire sprinkler and drainage systems [27, 28, 29, 30, 31]. In this chapter, we present a genetic algorithm (GA) designed to solve this problem. We show that the GA finds optimal or near-optimal solutions to a variety of test problems and in a small fraction of the time required by an exact solver.

There are a number of variants of the Steiner tree problem. (Surveys of some of the standard versions may be found in [27, 32, 33, 34, 35].) In the Euclidean version, the nodes are points in the Euclidean plane and the edge costs are the Euclidean distances. In the Rectilinear Steiner

[1] The costs (often referred to as *weights*) of the edges connecting the nodes depend on the version of the problem being considered as we describe below.

tree problem, edges must be oriented horizontally or vertically. In this case, the edge costs are the so-called *taxicab* distances. A modified version of the Rectilinear problem considers nodes located on a hexagonal grid [28, 29]. One can also study the problem in an arbitrary network (also known as a *weighted graph*) in which case a graph with weighted edges is given but there is no underlying space. Later, we discuss how we convert our problem to one on a network in order to find optimal solutions for small to medium sized problem instances.

A related problem to the ENSTP is the Steiner tree problem with obstacles. (See [32] for a review and list of earlier references.) This can be viewed as a specific case of the ENSTP in which the cost structure is uniform except for the locations of obstacles, where the costs are infinite. As a practical example, consider configuring the layout of an electrical system on a building floor in the presence of walls and columns that may not be penetrated. (In the ENSTP formulation, one could assign a finite cost to penetrating a wall or column.) Zachariasen and Winter [36] present an exact algorithm for the Euclidean Steiner tree problem with polygonal obstacles.[2]

In VLSI micro-chip design, numerous components on a chip need to be connected efficiently in a tree with wires. This can be achieved by solving a Rectilinear Steiner tree problem. To anticipate requirements of later stages of the chip design process, it may be necessary for the Steiner tree to avoid certain locations. Thus, the problem can be viewed as a Rectilinear Steiner tree problem with obstacles. Alpert et al. [31] develops an algorithm to solve this problem, known as the buffered Steiner tree construction problem.

In some applications, Steiner nodes corresponding to bends or junctions may require the use of fixtures or other hardware that can increase the overall cost of the tree [30]. Charging additional penalties for Steiner nodes can reflect these issues. This *node-weighted* Steiner tree problem is described with references in [32]. In an earlier work [2], we considered this problem within the context of the ENSTP.

The Steiner tree problem can also be studied in three dimensions. With applications involving 3-D VLSI layout in mind, Kanemoto et al. [37] solves the rectilinear 3-D problem using

[2]One interesting case they solve is for a set of terminals located within a fractal (the Sierpinski triangle). As might be expected, the solution tree looks fractal as well.

a genetic algorithm. Stanton and Smith [38] use 3-D Steiner trees to model minimum energy configurations of large complex molecules, such as proteins and DNA.

In general, Steiner tree problems have been shown to be NP-hard [32, 33, 34, 39]. Therefore, algorithms designed to find optimal solutions have exponential computational complexity in the worst case. As we will soon explain, our formulation of the ENSTP can be converted to a Steiner tree problem in a network, and so it will also be NP-hard. This motivates our development of an approximation algorithm. Heuristic approaches to find near-optimal solutions for the NP-hard cases have been proposed for several other variants of the Steiner tree problem. Gröpl et al. [40] reviews several mostly greedy algorithms that have been applied to the Steiner tree problem in networks. Numerous randomized algorithms exist as well. Barreiros [41] uses a genetic algorithm to solve the Euclidean Steiner tree problem. Ribeiro and de Souza [42] use tabu-search-based techniques for solving Steiner tree problems in networks. For the same problem with directed edges, simulated-annealing-based procedures are proposed in [43]. Julstrom [44, 45] applies various GA's, including one using an edge-set encoding, to the Rectilinear Steiner tree problem. For the ENSTP, Coulston [29] applies a GA using a full-components-based encoding. Coulston's work is closest to the work being presented here, but he considers any path between two nodes to be an edge, while we restrict edges to be straight-line connections between nodes. This restriction is relevant to some applications, such as circuit design, in which components are connected by straight-line segments of rigid wire.

This chapter is organized as follows: in the next section, we formulate the problem, in Sect. 4.3, we describe our genetic-algorithm-based procedure used to find solutions, then, in Sect. 4.4, we present results, followed by a conclusion including potential directions for future work.

4.2 Problem Formulation

We consider the problem of finding near-optimal Steiner trees on a non-uniform surface. Each location on the surface has an associated cost. A given set of nodes, known as terminal nodes, has

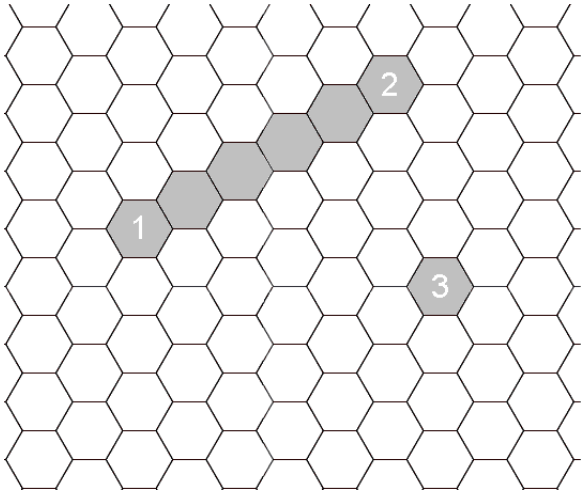

Figure 4.1: Hexagonal tiling of a 2-dimensional space. An edge is defined to be a straight line of cells.

to be connected in the form of a spanning tree. As in the usual Steiner tree problem, additional nodes not in the terminal-node set may be used to help connect the terminal nodes.

In order to represent the cost structure of the space underlying the nodes, it is necessary to use some type of grid. A hexagonal grid is a reasonable choice since hexagons are the highest degree regular polygon that can tile the plane, and such a tiling has the desirable property that distances between centers of adjacent cells are equal [29]. The size of the cells may affect the quality of the solution. However in this work, we focus on finding the lowest cost Steiner tree given a cost structure with a predetermined cell size.

We divide the two-dimensional Euclidean space into hexagonal cells as shown in Fig. 4.1. Each cell has a cost associated with it and may contain at most one node. The node is assumed to be located at the center of its cell. Two nodes can be connected directly only if a straight line of cells can be drawn between the cells containing the two nodes. For instance, in Fig. 4.1, cells 1 and 2 can be connected, but cell 3 can not be directly connected to either of the other two cells. The cost of an edge connecting two cells, such as 1 and 2, is equal to the sum of the costs associated with all the intermediate cells plus one half the costs of cells 1 and 2.

4.2.1 Network Steiner Tree Formulation

The use of the hexagonal grid and the restriction of one node per hexagonal cell simplify the search for optimal solutions because they reduce an otherwise uncountable search space to one of a finite size. In effect, this allows us to reduce the ENSTP to a Steiner tree problem in a network. This is illustrated in Fig. 4.2. The points in each cell represent potential terminal or Steiner nodes. Edges connect nodes to each of their neighbors forming a *triangle graph*. The cost of these nearest-neighbor edges is computed by adding together half of the cost of each cell the edge traverses. (Recall that we assume each node is located in the center of its cell, and so any edge incident upon it traverses half of the cell.) The problem is now represented in network form. This enables us to find optimal solutions using an exact algorithm for the network problem. In this chapter, we solve small and medium sized problems to optimality using the Dreyfus-Wagner algorithm [46], of which a lucid presentation may be found in [47]. The exponential computational complexity of this algorithm makes it impractical to solve larger problems. We will show that our GA compares favorably with the exact algorithm with respect to quality of solution.

Another benefit of representing the problem in a network is that it facilitates the determination of *shortest path minimal spanning trees* that we use to seed the initial population of the GA and to improve existing solutions. This is described in Sect. 4.3.

4.2.2 Problem Instances

The problem instances used in this chapter are listed in Table 4.1; solutions to a few are shown in Figures 4.7 through 4.12. (A full listing of the problem instance data and solutions can be found in Appendix A.) They consist of various random and ordered terminal node sets and grid cost structures. Some of the grid cost structures used were uniform, one hill, two hills, one pit, and two pits. The hill is essentially a discretized 2-dimensional Gaussian distribution. The cost, C, of a cell (i, j) is given by

$$C(i,j) = exp\left[-\left(\frac{i-i_o}{\sigma_i}\right)^2 - \left(\frac{j-j_o}{\sigma_j}\right)^2\right]$$

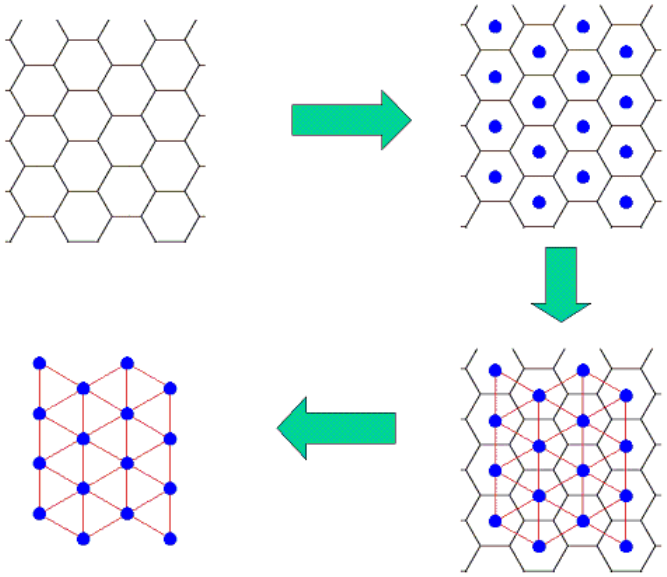

Figure 4.2: Transforming the ENSTP to a Steiner tree problem in a network. Starting with the hexagonal grid in the upper left, we designate potential node locations as in the upper right part of the figure. Next, each node location is connected to its nearest neighbors as the edges drawn in the lower right indicate. The cost of these edges is computed by adding together half of the cost of each cell the edge traverses. We now have a network and the grid can be discarded as in the lower left.

where i_o, j_o, σ_i, and σ_j are parameters. Letting (i_o, j_o) be the center of the grid causes the distribution to be centered on the grid. The standard deviations, σ_i and σ_j control the shape of the hill. Variations on this basic formula are used to create grid cost structures with various combinations of hills and pits. We also used random grids. For terminal node sets we used both randomly generated ones, and ones with specific structures, such as a ring.

4.3 Genetic Algorithm: Queen Bee Selection with Spatial-Horizontal Crossover

4.3.1 The Algorithm

The optimal solution to a Steiner tree problem is known as the Steiner minimal tree (SMT). We implement a genetic algorithm to find a Steiner tree (ST) whose cost is as close as possible to that of the SMT. This GA was the top performer out of several variants we tested, some of which

Table 4.1: Problem Instances

Problem #	Grid		Terminal Node Set	
	Size	Type	# of Nodes	Type
1	21 x 17	Hill	10	Random
2	21 x 17	Hill	7	Ring
3	21 x 17	2-Hill	10	Random
4	21 x 17	2-Hill	7	Ring
5	24 x 15	Random	7	Random
6	35 x 35	4-Hill	15	Random
7	35 x 35	Hill	15	Random
8	35 x 35	4-Hill	15	Ring
9	35 x 35	Hill	15	Ring
10	35 x 35	Random	15	Random
11	50 x 50	Hill	20	Random
12	50 x 50	2-Pit	20	Ring
13	80 x 80	Hill	32	Random

were similar to ones used to solve the Euclidean Steiner tree problem. Others used common GA or heuristic search strategies such as population diversity, and divide and conquer.

Before detailing the algorithm, we describe the population and solution encodings, which were arrived at through empirical testing. We use a population size of 40. Each individual is represented by a fixed-length chromosome that can hold up to $4N$ Steiner node locations where N is the number of terminal nodes. The method for determining the locations of the initial set of individuals is described in the next section. An example of a chromosome for $N = 4$ and a grid size of 20 by 20 is:

$$individual_k = \{(18,6),(11,17),(2,4),(13,3),(4,10),(10,20),$$
$$(2,14),(7,9),(3,9),(16,16),(5,5),(9,17),(2,8),(1,9),(20,14),(4,4)\},$$

where each of the 16 (=$4N$) pairs represents a Steiner node location.[3] Checks are performed to ensure that the Steiner node locations do not coincide with the terminal node locations.

The algorithm is as follows:

1. Input: terminal node set, grid cost structure

2. Generate **Initial Population**

Steps 3–7 repeated for TMAX iterations

[3]Strictly speaking, since some of the nodes in an individual will not be used in the Steiner tree, then by definition, they are not Steiner nodes. Nevertheless, we will use the term *Steiner node* loosely to refer to the nodes in an individual.

3. Find **Fitness** (= ST cost) of each individual

4. Queen Bee Selection to select parents

5. Spatial-Horizontal Crossover on parents to produce offspring

6. Mutation 1 – add Steiner nodes at edge crossings

7. Mutation 2 – randomly move Steiner nodes

8. Output: final Steiner tree, time series of best Steiner tree costs, MST on the terminal node set (for rough comparison), and total run time.

4.3.2 Explanation

Initial Population

In order to speed up the overall algorithm, we use two methods to seed the initial population with individuals that are better than purely random solutions. The first method is to generate what we call *shortest path minimal spanning trees*. Recall from Sect. 4.1 that we restrict edges to be straight line segments (see Fig. 4.3). To generate the seed solutions, we begin by temporarily redefining an edge to be the shortest path between its endpoint nodes. Such edges can have multiple turns in them. These edges can be found for all of the terminal nodes at once using an all-pairs shortest paths algorithm such as Floyd's algorithm [48]. The terminal nodes and this set of edges form a complete graph, upon which it is simple to find a minimal spanning tree using Prim's algorithm [49]. The resulting tree is what we refer to as the *shortest path minimal spanning tree* (see Fig. 4.4).

Because we are restricting the edges to be straight-line segments, we can convert the shortest path minimal spanning tree into a viable (by our edge definition) Steiner tree by introducing Steiner nodes at all turn and junction locations within the edges. This is demonstrated in Fig. 4.5. This procedure will yield one individual in the population by encoding the Steiner node locations that are introduced. We generate 10 such solutions by repeatedly perturbing the underlying grid cost structure by a small random amount. (We add a value taken from a uniform distribution ranging between 0 and 2.5 to the cost of each cell location. The unperturbed cell costs are between 0 and

1. The value 2.5 was chosen because it was found to be large enough to produce diverse solutions.)

Another 20 members of the initial population contain Steiner node locations that are randomly generated but with a bias towards low-cost regions of the grid. The remaining 10 members of the initial population have purely random Steiner node locations.

Fitness

For each individual in the population, the algorithm determines the Steiner tree as follows. Given the complete graph over the terminal nodes and the Steiner nodes encoded in a particular individual, a MST is found. Degree-1 Steiner nodes and their incident edges are removed as described in Sect. 4.3.3, to yield the individual's Steiner tree. The fitness of the individual is the cost of its Steiner tree. The tree cost is the sum of the tree's edge costs, which were defined in Sect. 4.2. This method for finding a Steiner tree is quick, though not necessarily optimal for the individual's set of Steiner nodes.

Queen Bee Parent Selection

The fittest individual (*the Queen Bee*) mates with the remainder of the population [50]. Out of the resulting set of parents and offspring, the 40 fittest individuals are chosen to replace the current population. The mating procedure used is the spatial-horizontal crossover described below. Queen Bee selection appears to be successful for two reasons. First, it allows for the incremental improvement of an already good solution. It ensures that many of the best solution elements are preserved in each offspring. Second, unlike a more globally elitist scheme such as tournament or roulette wheel selection, it allows even the worst individuals to pass parts of their solutions to the next generation. This is advantageous since, as we learned when studying individual solutions, very poor ones often have high quality subtrees within them.

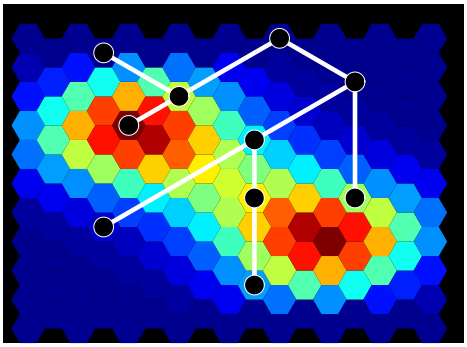

Figure 4.3: A minimal spanning tree (MST) using our straight-line edge definition. Black circles indicate terminal nodes.

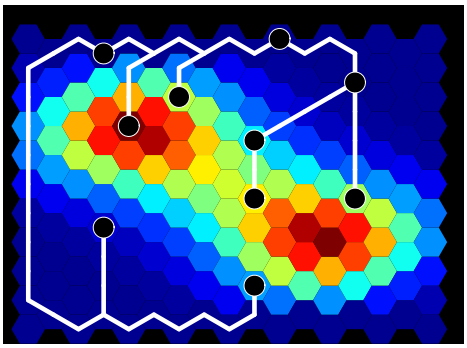

Figure 4.4: A *shortest path* MST for which an edge is taken to be the shortest path between the nodes it connects.

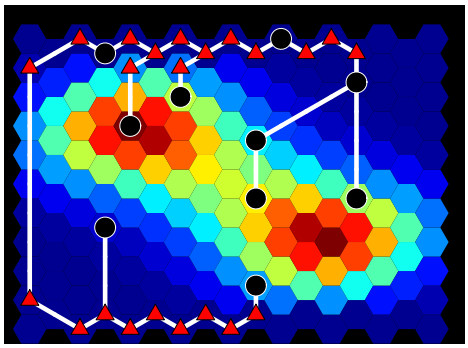

Figure 4.5: The MST in the previous figure can be viewed as a Steiner tree consistent with our edge definition if Steiner nodes (indicated by red triangles) are designated at all bends and junctions. The initial population of our GA is seeded with individuals whose Steiner nodes are generated this way.

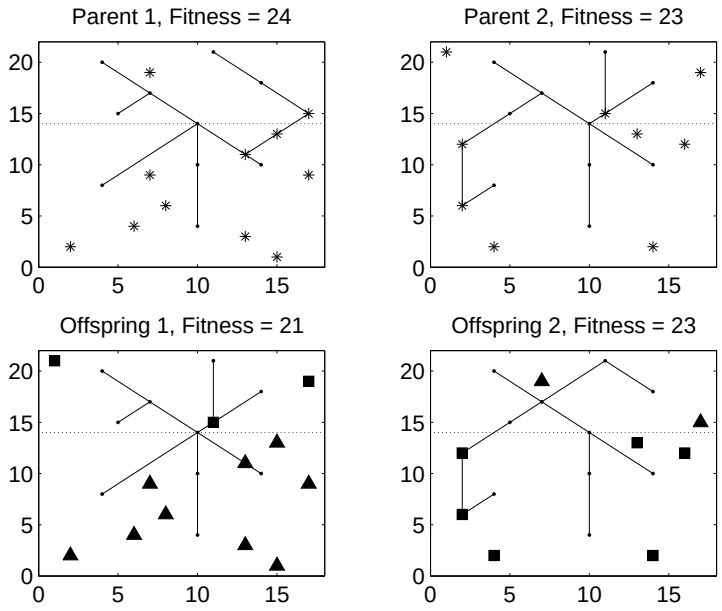

Figure 4.6: Horizontal crossover. The small dots in each plot represent the terminal nodes. In the top two plots, stars represent the Steiner nodes for the two parents. In the bottom two plots, triangles and squares represent the Steiner nodes in each offspring that came from Parents 1 and 2, respectively. The horizontal crossover location is indicated by the dotted line. The solid line segments indicate the Steiner tree edges.

Spatial-Horizontal Crossover

This operator produces two offspring from two parents by first splitting the grid into a top and bottom (see Fig. 4.6). The exact location of the horizontal line is the vertical midpoint of the grid plus a normal random variable (of mean 0 and positive standard deviation). The Steiner nodes from Parent 1 that are below the horizontal line are combined with the Steiner nodes from Parent 2 that are above the horizontal line to form Offspring 1. Likewise, Offspring 2 is formed from the Steiner nodes of Parent 1 that are above the horizontal line combined with the Steiner nodes from Parent 2 that are below the horizontal line. The Queen Bee is always one of the two parents. The idea behind using crossover with a spatial component is that even though the problem can be viewed as a network problem, the nodes actually do have physical locations. This operator makes use of this spatial information by preserving some physically-near components of the tree while breaking up others. How the grid is split (horizontally, vertically, diagonally, etc.) is arbitrary.

Mutation 1

It is possible that some of the edges in the Steiner tree found for a given individual, may cross each other. It is advantageous to add Steiner nodes at these crossing locations. Within the algorithm, these new Steiner nodes replace Steiner nodes in the individual's chromosome that are unused in the current tree. In practice, the percentage of Steiner nodes in an individual that are used in a tree tends to be small. Thus, there are typically many free locations into which a new Steiner node may be added.

Mutation 2

This operator randomly moves Steiner nodes belonging to a given individual. The two fittest individuals in the population are not subject to this mutation. A randomly chosen Steiner node is moved to a random location with probability 0.20. This helps prevent solutions from stagnating at local minima.

Output

Steps 4 through 8 are repeated TMAX times. Testing indicated a value of 70 for TMAX was reasonable. At the conclusion of a run, the following are reported:

a) the coordinates of the Steiner nodes in the best final Steiner tree,

b) a record of the Steiner tree cost of the fittest individual in the population after each iteration, and

c) the time it took for the algorithm to run.

In addition, the best final Steiner tree is displayed along with the terminal nodes and Steiner nodes over a color or gray-scale image of the grid cost structure (e.g., see Figures 4.7 through 4.12).

4.3.3 Improvement Procedures

In order to improve each Steiner tree, the GA utilizes additional procedures that remove degree-1 Steiner nodes and suggest potential Steiner nodes from edge intersections (see Sect. 4.3.2).

The degree of a node is defined as the number of edges incident to that node. Any degree-1 Steiner node in a Steiner tree can be removed since it is not needed to connect the terminal nodes. Removing these nodes will lead to an improvement in the solution value, and may also cause the degree of other Steiner nodes to drop to one. Hence, we apply an iterative procedure to remove degree-1 Steiner nodes until all Steiner nodes have degree greater than one.

As was mentioned previously, if there are any intersecting edges in the Steiner tree, the solution value might be improved by introducing a new Steiner node at the intersection location. The solution improvement is possible because one of the four edges originating at that intersection can be deleted from the tree if a Steiner node is created there. This is implemented in the GA as a mutation as described in Sect. 4.3.2.

4.4 Results

We ran the exact algorithm, our GA and a standard GA on the first 10 problems in Table 4.1. We also ran our GA and the standard GA on problems 11, 12, and 13 in Table 4.1. We did not solve these three problems to optimality due to the prohibitively long running time required by the exact algorithm. The idea behind comparing to a standard GA is to show that there is an advantage gained by the specific components we used (Queen Bee selection, spatial crossover, shortest path minimal spanning tree seeds, etc.). The standard GA uses tournament parent selection rather than Queen Bee selection, and one-point crossover instead of spatial-horizontal crossover. (In tournament selection, we draw subsets of size 5 from the population and choose the fittest 2 individuals in each subset to be parents. This process is repeated until 20 pairs of parents have been chosen. In one-point crossover, the parents are viewed as lists of Steiner node locations, and a position in the list is chosen at random. The entries before this position in parent 1 are combined with the entries after this position from parent 2 to create offspring 1, and vice versa to create

Table 4.2: Final Steiner Tree Costs for Each Algorithm. The values for each GA are calculated using 10 runs per problem. The numbers in parenthesis are the percent above the optimal value. The exact algorithm was not run on the last three problems because of their large size.

		Our GA			*Standard GA*		
#	*Optimal*	*best*	*mean*	*stdev*	*best*	*mean*	*stdev*
1	11.138	11.138(.00)	11.155(.15)	.01	11.215(.69)	11.360(1.99)	.10
2	10.001	10.001(.00)	10.012(.12)	.03	10.277(2.76)	10.416(4.15)	.08
3	4.806	4.806(.00)	4.806(.00)	.00	4.907(2.09)	4.942(2.84)	.04
4	4.158	4.158(.00)	4.158(.00)	.00	4.367(5.04)	4.562(9.73)	.14
5	5.605	5.605(.00)	5.605(.00)	.00	5.605(.00)	5.605(.00)	.00
6	26.606	26.653(.18)	27.104(1.87)	.21	30.015(12.8)	30.930(16.3)	.48
7	31.310	31.405(.30)	31.513(.65)	.07	34.016(8.64)	34.892(11.4)	.53
8	31.965	32.324(1.12)	32.574(1.91)	.19	38.450(20.3)	39.322(23.0)	.50
9	32.744	32.744(.00)	32.752(.02)	.02	33.381(1.95)	34.455(5.22)	.66
10	19.435	19.494(.31)	19.730(1.52)	.26	23.030(18.5)	23.760(22.3)	.46
11	—	32.227	32.479	.24	33.776	34.579	.44
12	—	38.234	38.240	.01	38.407	38.541	.08
13	—	72.325	72.471	.22	77.546	79.224	1.34

offspring 2. The 40 offspring that are generated become the new population.) The standard GA uses a purely random initial population.

For each problem, both GAs were run 10 times at 70 iterations per run. The cheapest tree cost found by each run was returned and then the mean, minimum, and standard deviation of these values over the 10 runs were recorded. The results are listed in Table 4.2. (We provide a more detailed listing of the solution data along with solution plots in Appendix A.) The computations were carried out in MATLAB on a 3 GHz computer with 3 GB of RAM.

On the small problems (1 through 5), our GA finds the optimal solution on nearly every run. The computation time was approximately 1 to 2 minutes per run. The exact algorithm required roughly 1.5 minutes for the small cases with 7 terminal nodes (problems 2, 4, and 5) and 20 minutes for those with 10 terminal nodes (problems 1 and 3). This is consistent with the Dreyfus-Wagner algorithm's complexity of about 3^N, where N is the number of terminal nodes.[4] Our GA outperforms the standard GA, which does not regularly find the optimal solution, by an average of about 4%. Run times for the two GAs were comparable.

For the medium-sized problems (6 through 10), our GA finds near-optimal solutions that are

[4]Through numerical investigation, we estimated that our GA scales roughly like N^2.

within around 1% of optimal on average. We present several examples in Figures 4.7 through 4.12. These plots show that even when the GA solution is not optimal, it reproduces the most important structural features of the optimal solution in the way that it avoids high cost regions. They also illustrate how our GA works well on problems with either structured or random input data (grids and nodes sets). Our GA was significantly faster than the exact algorithm in these medium-sized problems: 15 minutes versus 12 days. And it outperformed the standard GA by 14% on average, with similar run times. For problems 1 to 10, our GA (best run) was, on average, 0.19% above optimality, whereas the standard GA (best run) was, on average, 7.25% above optimality.

As mentioned above, problems 11, 12, and 13 were too large to be solved with the exact algorithm in a reasonable amount of time. But we do compare the two GAs on these problems and find that our GA remains superior to the standard GA by an average of around 6%. Here, as throughout the results, our GA exhibits a lower standard deviation over the 10 runs per problem than the standard GA, indicating it is also the more stable algorithm. One issue that becomes a more serious concern for our GA with increasing problem size is run time. The determination of the shortest path minimal spanning trees begins to slow the algorithm down compared to the standard GA. This effect becomes apparent in problem 13, which is the largest problem we solve. At present, we are studying ways to circumvent this problem, such as a divide and conquer approach.

4.5 Conclusion

In conclusion, we have considered a variation of the Euclidean Steiner tree problem with non-uniform underlying cost structure. We developed a genetic algorithm that finds optimal or near-optimal solutions over a variety of problem instances. As in the optimal solutions, the GA solution trees are able to correctly avoid high cost areas while finding low cost regions. We point out, however, that the GA scales much better with increasing problem size than the exact algorithm. In addition, our GA outperforms a different GA that uses more standard components.

In future work, we plan to improve our current algorithm by employing smarter, simpler operators. One possibility is to find the exact solution for some individuals, such as the Queen Bee,

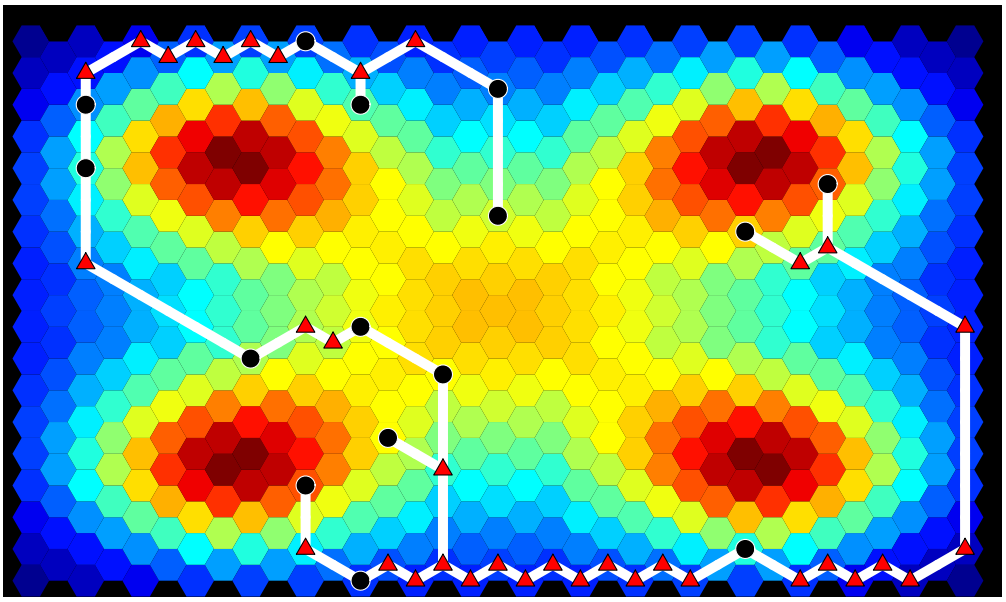

Figure 4.7: Optimal solution for Problem 6, Cost = 26.606. Red triangles are Steiner nodes, black circles are terminal nodes. Red cells are the most costly, blue cells are the least costly.

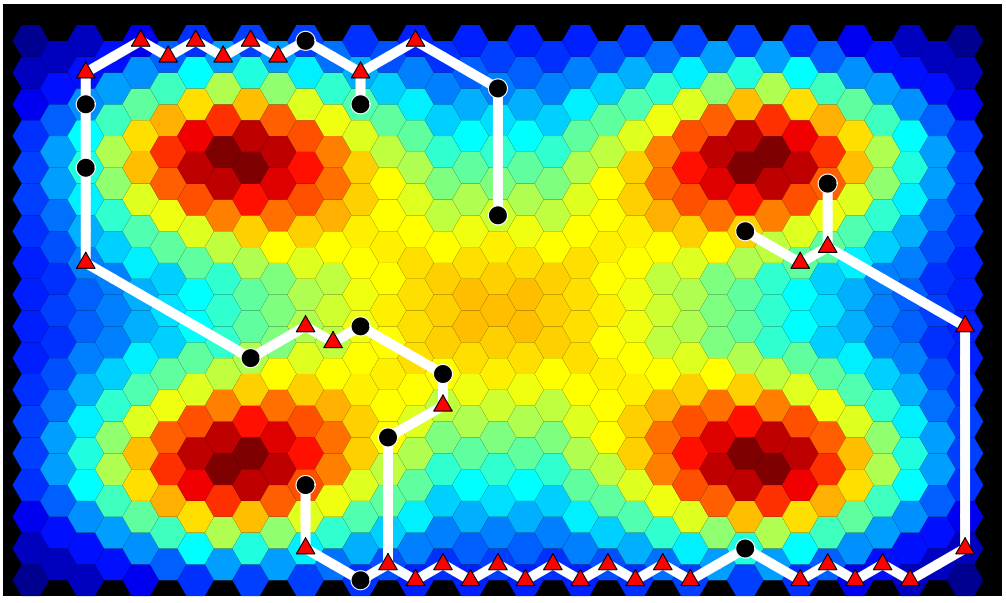

Figure 4.8: Sample solution found by our GA for Problem 6, Cost = 26.653. Red triangles are Steiner nodes, black circles are terminal nodes. Red cells are the most costly, blue cells are the least costly.

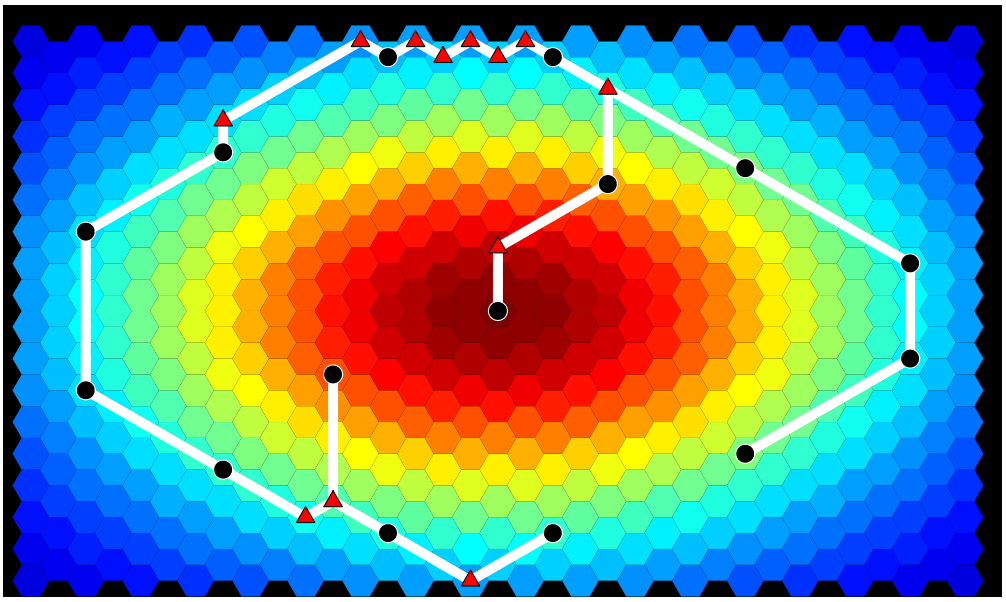

Figure 4.9: Optimal solution for Problem 9, Cost = 32.744. Red triangles are Steiner nodes, black circles are terminal nodes. Red cells are the most costly, blue cells are the least costly.

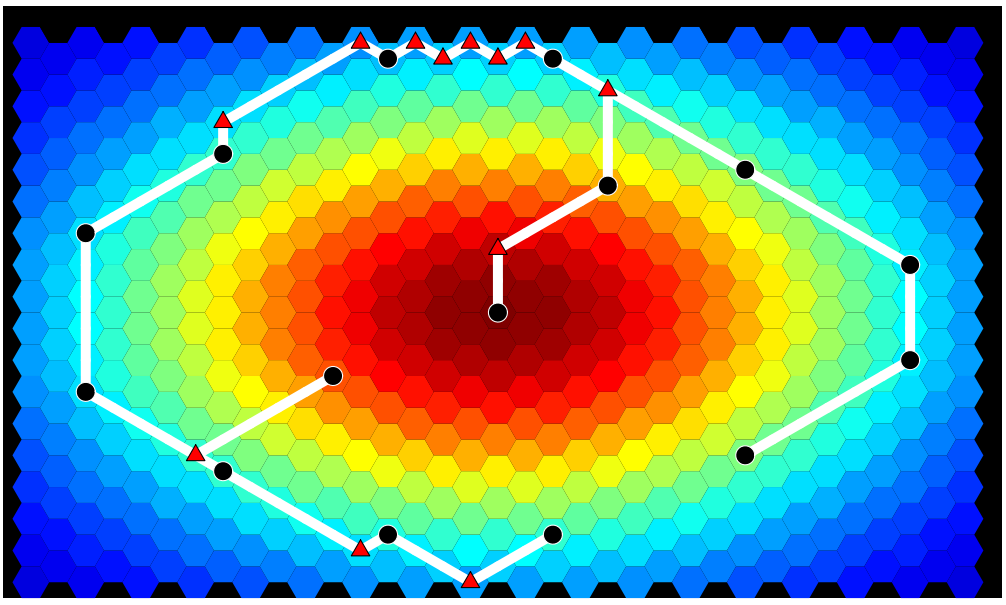

Figure 4.10: Sample solution found by our GA for Problem 9, Cost = 33.028. Red triangles are Steiner nodes, black circles are terminal nodes. Red cells are the most costly, blue cells are the least costly. Our GA was usually able to find the optimal solution for this problem.

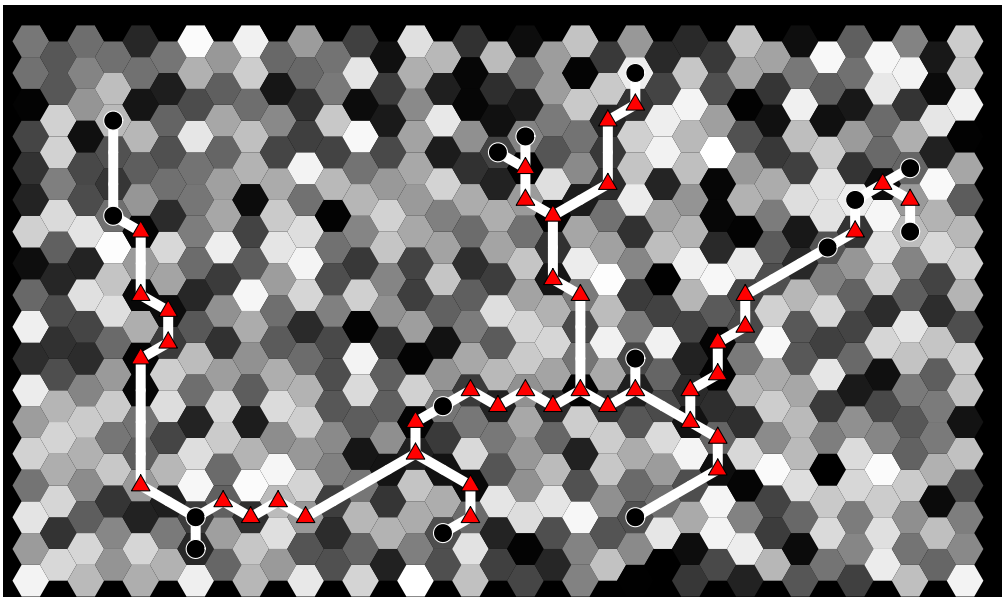

Figure 4.11: Optimal solution for Problem 10, Cost = 19.435. Red triangles are Steiner nodes, black circles are terminal nodes. The lighter the shading, the more costly the cell. Both the grid and terminal node set were randomly generated.

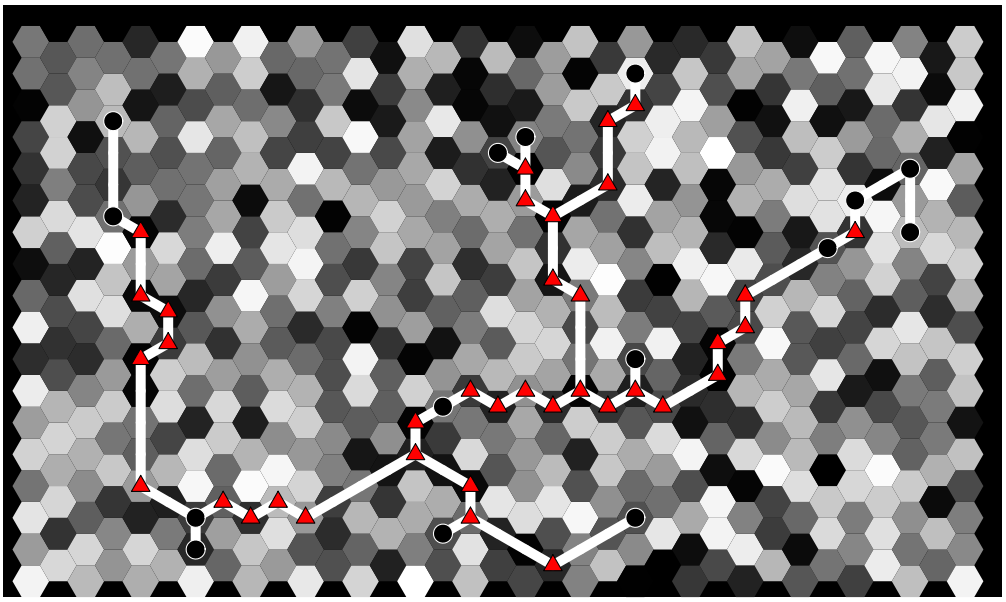

Figure 4.12: Sample solution found by our GA for Problem 10, Cost = 19.668. Red triangles are Steiner nodes, black circles are terminal nodes. The lighter the shading, the more costly the cell. Both the grid and terminal node set were randomly generated.

rather than approximating them by the MST. The algorithm could also be applied to problems utilizing geographic information systems (GIS) data to determine the grid cost structure and node locations. For example, the GIS data might factor steepness and vegetation into the grid cost determination [51, pages 142–147]. Or, as in the case of a recent study [52] on routing trucks bearing hazardous materials, the factors could include population counts, accident probability, risk of hijack, traffic conditions, emergency response, etc. We also plan to solve larger problems and remove the hexagonal grid restriction. Another potential research direction is the rectilinear version of the non-uniform problem, which easily lends itself to discretization using a square grid.

Appendix A

Data Sets for Optimizing Transmission Network Layout

In this section, we include the data sets from the problem instances used in Chapter 4. In the foreseeable future, these data sets may be obtained electronically by contacting the author via email at `ifrommer@post.harvard.edu`. They are provided here in the event that a future researcher is unable to locate electronic versions. We provide:

1. Grid cost values[1]
2. Terminal node locations
3. Problem solutions (optimal and best found by our GA):
   (a) Steiner node locations
   (b) Solution tree plots

A.1 Grid Data

The grid cost values are stored in arrays. The correspondence between the arrays and the hexagonal grid is illustrated in Fig. A.1. The way the values are stored allows for their use in rectilinear problems as well. For the hexagonal case, only data located at same-parity coordinates in the array are used.

Small hill grid, 21 by 17, used in problems 1 and 2

row 1 = {0.0550 0.0743 0.0963 0.1200 0.1437 0.1653 0.1827 0.1940 0.1979 0.1940 0.1827 0.1653 0.1437 0.1200 0.0963 0.0743 0.0550}

row 2 = {0.0773 0.1044 0.1353 0.1686 0.2019 0.2322 0.2567 0.2725 0.2780 0.2725 0.2567 0.2322 0.2019 0.1686 0.1353 0.1044 0.0773}

row 3 = {0.1044 0.1409 0.1827 0.2276 0.2725 0.3135 0.3465 0.3679 0.3753 0.3679 0.3465 0.3135 0.2725 0.2276 0.1827 0.1409 0.1044}

row 4 = {0.1353 0.1827 0.2369 0.2952 0.3535 0.4066 0.4493 0.4771 0.4868 0.4771 0.4493 0.4066 0.3535 0.2952 0.2369 0.1827 0.1353}

row 5 = {0.1686 0.2276 0.2952 0.3679 0.4404 0.5066 0.5599 0.5945 0.6065 0.5945 0.5599 0.5066 0.4404 0.3679 0.2952 0.2276 0.1686}

row 6 = {0.2019 0.2725 0.3535 0.4404 0.5273 0.6065 0.6703 0.7118 0.7261 0.7118 0.6703 0.6065 0.5273 0.4404 0.3535 0.2725 0.2019}

row 7 = {0.2322 0.3135 0.4066 0.5066 0.6065 0.6977 0.7711 0.8187 0.8353 0.8187 0.7711 0.6977 0.6065 0.5066 0.4066 0.3135 0.2322}

[1]For space considerations, the grid cost values have been rounded to four decimal places. This might lead to small discrepancies between solution costs constructed using these printed values compared with ones from the actual data sets.

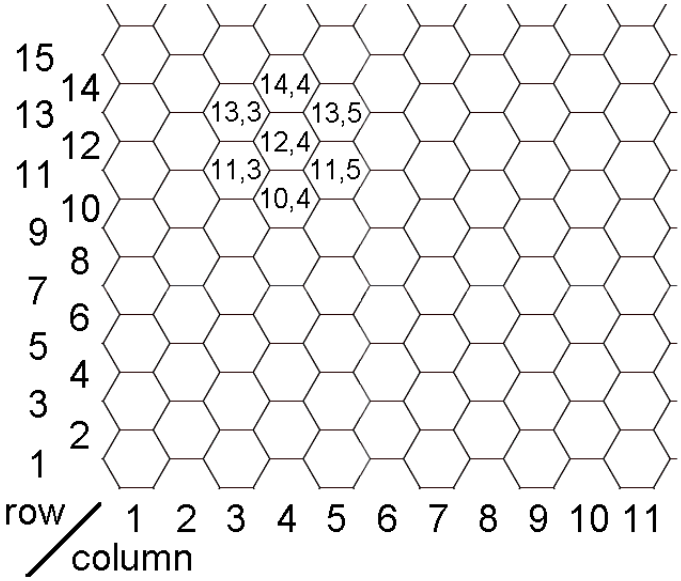

Figure A.1: Grid data coordinate system. Some sample cell coordinates are indicated.

row 8 = {0.2567 0.3465 0.4493 0.5599 0.6703 0.7711 0.8521 0.9048 0.9231 0.9048 0.8521 0.7711 0.6703 0.5599 0.4493 0.3465 0.2567}

row 9 = {0.2725 0.3679 0.4771 0.5945 0.7118 0.8187 0.9048 0.9608 0.9802 0.9608 0.9048 0.8187 0.7118 0.5945 0.4771 0.3679 0.2725}

row 10 = {0.2780 0.3753 0.4868 0.6065 0.7261 0.8353 0.9231 0.9802 1.0000 0.9802 0.9231 0.8353 0.7261 0.6065 0.4868 0.3753 0.2780}

row 11 = {0.2725 0.3679 0.4771 0.5945 0.7118 0.8187 0.9048 0.9608 0.9802 0.9608 0.9048 0.8187 0.7118 0.5945 0.4771 0.3679 0.2725}

row 12 = {0.2567 0.3465 0.4493 0.5599 0.6703 0.7711 0.8521 0.9048 0.9231 0.9048 0.8521 0.7711 0.6703 0.5599 0.4493 0.3465 0.2567}

row 13 = {0.2322 0.3135 0.4066 0.5066 0.6065 0.6977 0.7711 0.8187 0.8353 0.8187 0.7711 0.6977 0.6065 0.5066 0.4066 0.3135 0.2322}

row 14 = {0.2019 0.2725 0.3535 0.4404 0.5273 0.6065 0.6703 0.7118 0.7261 0.7118 0.6703 0.6065 0.5273 0.4404 0.3535 0.2725 0.2019}

row 15 = {0.1686 0.2276 0.2952 0.3679 0.4404 0.5066 0.5599 0.5945 0.6065 0.5945 0.5599 0.5066 0.4404 0.3679 0.2952 0.2276 0.1686}

row 16 = {0.1353 0.1827 0.2369 0.2952 0.3535 0.4066 0.4493 0.4771 0.4868 0.4771 0.4493 0.4066 0.3535 0.2952 0.2369 0.1827 0.1353}

row 17 = {0.1044 0.1409 0.1827 0.2276 0.2725 0.3135 0.3465 0.3679 0.3753 0.3679 0.3465 0.3135 0.2725 0.2276 0.1827 0.1409 0.1044}

row 18 = {0.0773 0.1044 0.1353 0.1686 0.2019 0.2322 0.2567 0.2725 0.2780 0.2725 0.2567 0.2322 0.2019 0.1686 0.1353 0.1044 0.0773}

row 19 = {0.0550 0.0743 0.0963 0.1200 0.1437 0.1653 0.1827 0.1940 0.1979 0.1940 0.1827 0.1653 0.1437 0.1200 0.0963 0.0743 0.0550}

row 20 = {0.0376 0.0508 0.0659 0.0821 0.0983 0.1130 0.1249 0.1327 0.1353 0.1327 0.1249 0.1130 0.0983 0.0821 0.0659 0.0508 0.0376}

row 21 = {0.0247 0.0334 0.0433 0.0539 0.0646 0.0743 0.0821 0.0872 0.0889 0.0872 0.0821 0.0743 0.0646 0.0539 0.0433 0.0334 0.0247}

Small 2-hill grid, 21 by 17, used in problems 3 and 4

row 1 = {0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000}

row 2 = {0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000}

row 3 = {0.0000 0.0001 0.0004 0.0013 0.0038 0.0100 0.0237 0.0503 0.0944 0.1539 0.2149 0.2530 0.2472 0.1973 0.1268 0.0645 0.0256}

row 4 = {0.0001 0.0004 0.0011 0.0031 0.0080 0.0195 0.0437 0.0894 0.1638 0.2651 0.3730 0.4495 0.4565 0.3848 0.2651 0.1469 0.0645}

row 5 = {0.0004 0.0011 0.0028 0.0067 0.0158 0.0352 0.0736 0.1424 0.2511 0.3969 0.5543 0.6730 0.6997 0.6132 0.4460 0.2651 0.1268}

row 6 = {0.0013 0.0031 0.0067 0.0143 0.0296 0.0597 0.1146 0.2068 0.3452 0.5250 0.7162 0.8629 0.9040 0.8109 0.6132 0.3848 0.1973}

row 7 = {0.0038 0.0080 0.0158 0.0296 0.0539 0.0966 0.1676 0.2777 0.4323 0.6228 0.8172 0.9619 1.0000 0.9040 0.6997 0.4565 0.2472}

row 8 = {0.0100 0.0195 0.0352 0.0597 0.0966 0.1515 0.2341 0.3506 0.5011 0.6730 0.8365 0.9471 0.9619 0.8629 0.6730 0.4495 0.2530}

row 9 = {0.0237 0.0437 0.0736 0.1146 0.1676 0.2341 0.3169 0.4224 0.5457 0.6730 0.7800 0.8365 0.8172 0.7162 0.5543 0.3730 0.2149}

row 10 = {0.0503 0.0894 0.1424 0.2068 0.2777 0.3506 0.4224 0.4933 0.5673 0.6325 0.6730 0.6730 0.6228 0.5250 0.3969 0.2651 0.1539}

row 11 = {0.0944 0.1638 0.2511 0.3452 0.4323 0.5011 0.5457 0.5673 0.5718 0.5673 0.5457 0.5011 0.4323 0.3452 0.2511 0.1638 0.0944}

row 12 = {0.1539 0.2651 0.3969 0.5250 0.6228 0.6730 0.6730 0.6325 0.5673 0.4933 0.4224 0.3506 0.2777 0.2068 0.1424 0.0894 0.0503}

row 13 = {0.2149 0.3730 0.5543 0.7162 0.8172 0.8365 0.7800 0.6730 0.5457 0.4224 0.3169 0.2341 0.1676 0.1146 0.0736 0.0437 0.0237}

row 14 = {0.2530 0.4495 0.6730 0.8629 0.9619 0.9471 0.8365 0.6730 0.5011 0.3506 0.2341 0.1515 0.0966 0.0597 0.0352 0.0195 0.0100}

row 15 = {0.2472 0.4565 0.6997 0.9040 1.0000 0.9619 0.8172 0.6228 0.4323 0.2777 0.1676 0.0966 0.0539 0.0296 0.0158 0.0080 0.0038}

row 16 = {0.1973 0.3848 0.6132 0.8109 0.9040 0.8629 0.7162 0.5250 0.3452 0.2068 0.1146 0.0597 0.0296 0.0143 0.0067 0.0031 0.0013}

row 17 = {0.1268 0.2651 0.4460 0.6132 0.6997 0.6730 0.5543 0.3969 0.2511 0.1424 0.0736 0.0352 0.0158 0.0067 0.0028 0.0011 0.0004}

row 18 = {0.0645 0.1469 0.2651 0.3848 0.4565 0.4495 0.3730 0.2651 0.1638 0.0894 0.0437 0.0195 0.0080 0.0031 0.0011 0.0004 0.0001}

row 19 = {0.0256 0.0645 0.1268 0.1973 0.2472 0.2530 0.2149 0.1539 0.0944 0.0503 0.0237 0.0100 0.0038 0.0013 0.0004 0.0001 0.0000}

row 20 = {0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000}

row 21 = {0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000}

Small random grid, 24 by 15, used in problem 5

row 1 = {0.9501 0.2311 0.6068 0.4860 0.8913 0.7621 0.4565 0.0185 0.8214 0.4447 0.6154 0.7919 0.9218 0.7382 0.1763}

row 2 = {0.4057 0.9355 0.9169 0.4103 0.8936 0.0579 0.3529 0.8132 0.0099 0.1389 0.2028 0.1987 0.6038 0.2722 0.1988}

row 3 = {0.0153 0.7468 0.4451 0.9318 0.4660 0.4186 0.8462 0.5252 0.2026 0.6721 0.8381 0.0196 0.6813 0.3795 0.8318}

row 4 = {0.5028 0.7095 0.4289 0.3046 0.1897 0.1934 0.6822 0.3028 0.5417 0.1509 0.6979 0.3784 0.8600 0.8537 0.5936}

row 5 = {0.4966 0.8998 0.8216 0.6449 0.8180 0.6602 0.3420 0.2897 0.3412 0.5341 0.7271 0.3093 0.8385 0.5681 0.3704}

row 6 = {0.7027 0.5466 0.4449 0.6946 0.6213 0.7948 0.9568 0.5226 0.8801 0.1730 0.9797 0.2714 0.2523 0.8757 0.7373}

row 7 = {0.1365 0.0118 0.8939 0.1991 0.2987 0.6614 0.2844 0.4692 0.0648 0.9883 0.5828 0.4235 0.5155 0.3340 0.4329}

row 8 = {0.2259 0.5798 0.7604 0.5298 0.6405 0.2091 0.3798 0.7833 0.6808 0.4611 0.5678 0.7942 0.0592 0.6029 0.0503}

row 9 = {0.4154 0.3050 0.8744 0.0150 0.7680 0.9708 0.9901 0.7889 0.4387 0.4983 0.2140 0.6435 0.3200 0.9601 0.7266}

row 10 = {0.4120 0.7446 0.2679 0.4399 0.9334 0.6833 0.2126 0.8392 0.6288 0.1338 0.2071 0.6072 0.6299 0.3705 0.5751}

row 11 = {0.4514 0.0439 0.0272 0.3127 0.0129 0.3840 0.6831 0.0928 0.0353 0.6124 0.6085 0.0158 0.0164 0.1901 0.5869}

row 12 = {0.0576 0.3676 0.6315 0.7176 0.6927 0.0841 0.4544 0.4418 0.3533 0.1536 0.6756 0.6992 0.7275 0.4784 0.5548}

row 13 = {0.1210 0.4508 0.7159 0.8928 0.2731 0.2548 0.8656 0.2324 0.8049 0.9084 0.2319 0.2393 0.0498 0.0784 0.6408}

row 14 = {0.1909 0.8439 0.1739 0.1708 0.9943 0.4398 0.3400 0.3142 0.3651 0.3932 0.5915 0.1197 0.0381 0.4586 0.8699}

row 15 = {0.9342 0.2644 0.1603 0.8729 0.2379 0.6458 0.9669 0.6649 0.8704 0.0099 0.1370 0.8188 0.4302 0.8903 0.7349}

row 16 = {0.6873 0.3461 0.1660 0.1556 0.1911 0.4225 0.8560 0.4902 0.8159 0.4608 0.4574 0.4507 0.4122 0.9016 0.0056}

row 17 = {0.2974 0.0492 0.6932 0.6501 0.9830 0.5527 0.4001 0.1988 0.6252 0.7334 0.3759 0.0099 0.4199 0.7537 0.7939}

row 18 = {0.9200 0.8447 0.3678 0.6208 0.7313 0.1939 0.9048 0.5692 0.6318 0.2344 0.5488 0.9316 0.3352 0.6555 0.3919}

row 19 = {0.6273 0.6991 0.3972 0.4136 0.6552 0.8376 0.3716 0.4253 0.5947 0.5657 0.7165 0.5113 0.7764 0.4893 0.1859}

row 20 = {0.7006 0.9827 0.8066 0.7036 0.4850 0.1146 0.6649 0.3654 0.1400 0.5668 0.8230 0.6739 0.9994 0.9616 0.0589}

row 21 = {0.3603 0.5485 0.2618 0.5973 0.0493 0.5711 0.7009 0.9623 0.7505 0.7400 0.4319 0.6343 0.8030 0.0839 0.9455}

row 22 = {0.9159 0.6020 0.2536 0.8735 0.5134 0.7327 0.4222 0.9614 0.0721 0.5534 0.2920 0.8580 0.3358 0.6802 0.0534}

row 23 = {0.3567 0.4983 0.4344 0.5625 0.6166 0.1133 0.8983 0.7546 0.7911 0.8150 0.6700 0.2009 0.2731 0.6262 0.5369}

row 24 = {0.0595 0.0890 0.2713 0.4091 0.4740 0.9090 0.5962 0.3290 0.4782 0.5972 0.1614 0.8295 0.9561 0.5955 0.0287}

Medium 4-hill grid, 35 by 35, used in problems 6 and 8

row 1 = {0.0000 0.0195 0.0435 0.0713 0.1008 0.1290 0.1526 0.1688 0.1765 0.1764 0.1708 0.1626 0.1545 0.1482 0.1441 0.1335 0.1363 0.1372 0.1363 0.1335 0.1441 0.1482 0.1545 0.1626 0.1708 0.1764 0.1765 0.1688 0.1526 0.1290 0.1008 0.0713 0.0435 0.0195 0.0000}

row 2 = {0.0195 0.0475 0.0827 0.1237 0.1674 0.2088 0.2422 0.2632 0.2699 0.2637 0.2485 0.2295 0.2111 0.1964 0.1863 0.1662 0.1694 0.1705 0.1694 0.1662 0.1863 0.1964 0.2111 0.2295 0.2485 0.2637 0.2699 0.2632 0.2422 0.2088 0.1674 0.1237 0.0827 0.0475 0.0195}

row 3 = {0.0435 0.0827 0.1326 0.1913 0.2539 0.3127 0.3590 0.3859 0.3908 0.3757 0.3470 0.3127 0.2800 0.2536 0.2349 0.2016 0.2053 0.2065 0.2053 0.2016 0.2349 0.2536 0.2800 0.3127 0.3470 0.3757 0.3908 0.3859 0.3590 0.3127 0.2539 0.1913 0.1326 0.0827 0.0435}

row 4 = {0.0713 0.1237 0.1913 0.2714 0.3567 0.4363 0.4979 0.5319 0.5341 0.5080 0.4625 0.4094 0.3591 0.3182 0.2890 0.2394 0.2436 0.2450 0.2436 0.2394 0.2890 0.3182 0.3591 0.4094 0.4625 0.5080 0.5341 0.5319 0.4979 0.4363 0.3567 0.2714 0.1913 0.1237 0.0713}

row 5 = {0.1008 0.1674 0.2539 0.3567 0.4663 0.5681 0.6462 0.6875 0.6870 0.6490 0.5856 0.5123 0.4431 0.3867 0.3463 0.2793 0.2840 0.2856 0.2840 0.2793 0.3463 0.3867 0.4431 0.5123 0.5856 0.6490 0.6870 0.6875 0.6462 0.5681 0.4663 0.3567 0.2539 0.1674 0.1008}

row 6 = {0.1290 0.2088 0.3127 0.4363 0.5681 0.6905 0.7837 0.8320 0.8293 0.7808 0.7015 0.6102 0.5240 0.4538 0.4033 0.3207 0.3260 0.3278 0.3260 0.3207 0.4033 0.4538 0.5240 0.6102 0.7015 0.7808 0.8293 0.8320 0.7837 0.6905 0.5681 0.4363 0.3127 0.2088 0.1290}

row 7 = {0.1526 0.2422 0.3590 0.4979 0.6462 0.7837 0.8883 0.9424 0.9388 0.8837 0.7939 0.6906 0.5931 0.5136 0.4565 0.3631 0.3690 0.3709 0.3690 0.3631 0.4565 0.5136 0.5931 0.6906 0.7939 0.8837 0.9388 0.9424 0.8883 0.7837 0.6462 0.4979 0.3590 0.2422 0.1526}

row 8 = {0.1688 0.2632 0.3859 0.5319 0.6875 0.8320 0.9424 1.0000 0.9978 0.9418 0.8496 0.7434 0.6429 0.5611 0.5024 0.4058 0.4122 0.4144 0.4122 0.4058 0.5024 0.5611 0.6429 0.7434 0.8496 0.9418 0.9978 1.0000 0.9424 0.8320 0.6875 0.5319 0.3859 0.2632 0.1688}

row 9 = {0.1765 0.2699 0.3908 0.5341 0.6870 0.8293 0.9388 0.9978 0.9991 0.9488 0.8634 0.7643 0.6704 0.5941 0.5395 0.4480 0.4550 0.4574 0.4550 0.4480 0.5395 0.5941 0.6704 0.7643 0.8634 0.9488 0.9991 0.9978 0.9388 0.8293 0.6870 0.5341 0.3908 0.2699 0.1765}

row 10 = {0.1764 0.2637 0.3757 0.5080 0.6490 0.7808 0.8837 0.9418 0.9488 0.9097 0.8392 0.7562 0.6773 0.6133 0.5679 0.4889 0.4965 0.4990 0.4965 0.4889 0.5679 0.6133 0.6773 0.7562 0.8392 0.9097 0.9488 0.9418 0.8837 0.7808 0.6490 0.5080 0.3757 0.2637 0.1764}

row 11 = {0.1708 0.2485 0.3470 0.4625 0.5856 0.7015 0.7939 0.8496 0.8634 0.8392 0.7890 0.7278 0.6694 0.6222 0.5892 0.5277 0.5358 0.5385 0.5358 0.5277 0.5892 0.6222 0.6694 0.7278 0.7890 0.8392 0.8634 0.8496 0.7939 0.7015 0.5856 0.4625 0.3470 0.2485 0.1708}

row 12 = {0.1626 0.2295 0.3127 0.4094 0.5123 0.6102 0.6906 0.7434 0.7643 0.7562 0.7278 0.6905 0.6542 0.6254 0.6059 0.5634 0.5720 0.5749 0.5720 0.5634 0.6059 0.6254 0.6542 0.6905 0.7278 0.7562 0.7643 0.7434 0.6906 0.6102 0.5123 0.4094 0.3127 0.2295 0.1626}

row 13 = {0.1545 0.2111 0.2800 0.3591 0.4431 0.5240 0.5931 0.6429 0.6704 0.6773 0.6694 0.6542 0.6387 0.6269 0.6201 0.5953 0.6043 0.6073 0.6043 0.5953 0.6201 0.6269 0.6387 0.6542 0.6694 0.6773 0.6704 0.6429 0.5931 0.5240 0.4431 0.3591 0.2800 0.2111 0.1545}

row 14 = {0.1482 0.1964 0.2536 0.3182 0.3867 0.4538 0.5136 0.5611 0.5941 0.6133 0.6222 0.6254 0.6269 0.6292 0.6327 0.6226 0.6320 0.6351 0.6320 0.6226 0.6327 0.6292 0.6269 0.6254 0.6222 0.6133 0.5941 0.5611 0.5136 0.4538 0.3867 0.3182 0.2536 0.1964 0.1482}

row 15 = {0.1441 0.1863 0.2349 0.2890 0.3463 0.4033 0.4565 0.5024 0.5395 0.5679 0.5892 0.6059 0.6201 0.6327 0.6439 0.6446 0.6543 0.6575 0.6543 0.6446 0.6439 0.6327 0.6201 0.6059 0.5892 0.5679 0.5395 0.5024 0.4565 0.4033 0.3463 0.2890 0.2349 0.1863 0.1441}

row 16 = {0.1335 0.1662 0.2016 0.2394 0.2793 0.3207 0.3631 0.4058 0.4480 0.4889 0.5277 0.5634 0.5953 0.6226 0.6446 0.6608 0.6706 0.6739 0.6706 0.6608 0.6446 0.6226 0.5953 0.5634 0.5277 0.4889 0.4480 0.4058 0.3631 0.3207 0.2793 0.2394 0.2016 0.1662 0.1335}

row 17 = {0.1363 0.1694 0.2053 0.2436 0.2840 0.3260 0.3690 0.4122 0.4550 0.4965 0.5358 0.5720 0.6043 0.6320 0.6543 0.6706 0.6806 0.6840 0.6806 0.6706 0.6543 0.6320 0.6043 0.5720 0.5358 0.4965 0.4550 0.4122 0.3690 0.3260 0.2840 0.2436 0.2053 0.1694 0.1363}

row 18 = {0.1372 0.1705 0.2065 0.2450 0.2856 0.3278 0.3709 0.4144 0.4574 0.4990 0.5385 0.5749 0.6073 0.6351 0.6575 0.6739 0.6840 0.6873 0.6840 0.6739 0.6575 0.6351 0.6073 0.5749 0.5385 0.4990 0.4574 0.4144 0.3709 0.3278 0.2856 0.2450 0.2065 0.1705 0.1372}

row 19 = {0.1363 0.1694 0.2053 0.2436 0.2840 0.3260 0.3690 0.4122 0.4550 0.4965 0.5358 0.5720 0.6043 0.6320 0.6543 0.6706 0.6806 0.6840 0.6806 0.6706 0.6543 0.6320 0.6043 0.5720 0.5358 0.4965 0.4550 0.4122 0.3690 0.3260 0.2840 0.2436 0.2053 0.1694 0.1363}

row 20 = {0.1335 0.1662 0.2016 0.2394 0.2793 0.3207 0.3631 0.4058 0.4480 0.4889 0.5277 0.5634 0.5953 0.6226 0.6446 0.6608 0.6706 0.6739 0.6706 0.6608 0.6446 0.6226 0.5953 0.5634 0.5277 0.4889 0.4480 0.4058 0.3631 0.3207 0.2793 0.2394 0.2016 0.1662 0.1335}

row 21 = {0.1441 0.1863 0.2349 0.2890 0.3463 0.4033 0.4565 0.5024 0.5395 0.5679 0.5892 0.6059 0.6201 0.6327 0.6439 0.6446 0.6543 0.6575 0.6543 0.6446 0.6439 0.6327 0.6201 0.6059 0.5892 0.5679 0.5395 0.5024 0.4565 0.4033 0.3463 0.2890 0.2349 0.1863 0.1441}

row 22 = {0.1482 0.1964 0.2536 0.3182 0.3867 0.4538 0.5136 0.5611 0.5941 0.6133 0.6222 0.6254 0.6269 0.6292 0.6327 0.6226 0.6320 0.6351 0.6320 0.6226 0.6327 0.6292 0.6269 0.6254 0.6222 0.6133 0.5941 0.5611 0.5136 0.4538 0.3867 0.3182 0.2536 0.1964 0.1482}

row 23 = {0.1545 0.2111 0.2800 0.3591 0.4431 0.5240 0.5931 0.6429 0.6704 0.6773 0.6694 0.6542 0.6387 0.6269 0.6201 0.5953 0.6043 0.6073 0.6043 0.5953 0.6201 0.6269 0.6387 0.6542 0.6694 0.6773 0.6704 0.6429 0.5931 0.5240 0.4431 0.3591 0.2800 0.2111 0.1545}

row 24 = {0.1626 0.2295 0.3127 0.4094 0.5123 0.6102 0.6906 0.7434 0.7643 0.7562 0.7278 0.6905 0.6542 0.6254 0.6059 0.5634 0.5720 0.5749 0.5720 0.5634 0.6059 0.6254 0.6542 0.6905 0.7278 0.7562 0.7643 0.7434 0.6906 0.6102 0.5123 0.4094 0.3127 0.2295 0.1626}

row 25 = {0.1708 0.2485 0.3470 0.4625 0.5856 0.7015 0.7939 0.8496 0.8634 0.8392 0.7890 0.7278 0.6694 0.6222 0.5892 0.5277 0.5358 0.5385 0.5358 0.5277 0.5892 0.6222 0.6694 0.7278 0.7890 0.8392 0.8634 0.8496 0.7939 0.7015 0.5856 0.4625 0.3470 0.2485 0.1708}

row 26 = {0.1764 0.2637 0.3757 0.5080 0.6490 0.7808 0.8837 0.9418 0.9488 0.9097 0.8392 0.7562 0.6773 0.6133 0.5679 0.4889 0.4965 0.4990 0.4965 0.4889 0.5679 0.6133 0.6773 0.7562 0.8392 0.9097 0.9488 0.9418 0.8837 0.7808 0.6490 0.5080 0.3757 0.2637 0.1764}

row 27 = {0.1765 0.2699 0.3908 0.5341 0.6870 0.8293 0.9388 0.9978 0.9991 0.9488 0.8634 0.7643 0.6704 0.5941 0.5395 0.4480 0.4550 0.4574 0.4550 0.4480 0.5395 0.5941 0.6704 0.7643 0.8634 0.9488 0.9991 0.9978 0.9388 0.8293 0.6870 0.5341 0.3908 0.2699 0.1765}

row 28 = {0.1688 0.2632 0.3859 0.5319 0.6875 0.8320 0.9424 1.0000 0.9978 0.9418 0.8496 0.7434 0.6429 0.5611 0.5024 0.4058 0.4122 0.4144 0.4122 0.4058 0.5024 0.5611 0.6429 0.7434 0.8496 0.9418 0.9978 1.0000 0.9424 0.8320 0.6875 0.5319 0.3859 0.2632 0.1688}

row 29 = {0.1526 0.2422 0.3590 0.4979 0.6462 0.7837 0.8883 0.9424 0.9388 0.8837 0.7939 0.6906 0.5931 0.5136 0.4565 0.3631 0.3690 0.3709 0.3690 0.3631 0.4565 0.5136 0.5931 0.6906 0.7939 0.8837 0.9388 0.9424 0.8883 0.7837 0.6462 0.4979 0.3590 0.2422 0.1526}

row 30 = {0.1290 0.2088 0.3127 0.4363 0.5681 0.6905 0.7837 0.8320 0.8293 0.7808 0.7015 0.6102 0.5240 0.4538 0.4033 0.3207 0.3260 0.3278 0.3260 0.3207 0.4033 0.4538 0.5240 0.6102 0.7015 0.7808 0.8293 0.8320 0.7837 0.6905 0.5681 0.4363 0.3127 0.2088 0.1290}

row 31 = {0.1008 0.1674 0.2539 0.3567 0.4663 0.5681 0.6462 0.6875 0.6870 0.6490 0.5856 0.5123 0.4431 0.3867 0.3463 0.2793 0.2840 0.2856 0.2840 0.2793 0.3463 0.3867 0.4431 0.5123 0.5856 0.6490 0.6870 0.6875 0.6462 0.5681 0.4663 0.3567 0.2539 0.1674 0.1008}

row 32 = {0.0713 0.1237 0.1913 0.2714 0.3567 0.4363 0.4979 0.5319 0.5341 0.5080 0.4625 0.4094 0.3591 0.3182 0.2890 0.2394 0.2436 0.2450 0.2436 0.2394 0.2890 0.3182 0.3591 0.4094 0.4625 0.5080 0.5341 0.5319 0.4979 0.4363 0.3567 0.2714 0.1913 0.1237 0.0713}

row 33 = {0.0435 0.0827 0.1326 0.1913 0.2539 0.3127 0.3590 0.3859 0.3908 0.3757 0.3470 0.3127 0.2800 0.2536 0.2349 0.2016 0.2053 0.2065 0.2053 0.2016 0.2349 0.2536 0.2800 0.3127 0.3470 0.3757 0.3908 0.3859 0.3590 0.3127 0.2539 0.1913 0.1326 0.0827 0.0435}

row 34 = {0.0195 0.0475 0.0827 0.1237 0.1674 0.2088 0.2422 0.2632 0.2699 0.2637 0.2485 0.2295 0.2111 0.1964 0.1863 0.1662 0.1694 0.1705 0.1694 0.1662 0.1863 0.1964 0.2111 0.2295 0.2485 0.2637 0.2699 0.2632 0.2422 0.2088 0.1674 0.1237 0.0827 0.0475 0.0195}

row 35 = {0.0000 0.0195 0.0435 0.0713 0.1008 0.1290 0.1526 0.1688 0.1765 0.1764 0.1708 0.1626 0.1545 0.1482 0.1441 0.1335 0.1363 0.1372 0.1363 0.1335 0.1441 0.1482 0.1545 0.1626 0.1708 0.1764 0.1765 0.1688 0.1526 0.1290 0.1008 0.0713 0.0435 0.0195 0.0000}

Medium hill grid, 35 by 35, used in problems 7 and 9

row 1 = {0.0766 0.0887 0.1018 0.1158 0.1306 0.1460 0.1617 0.1775 0.1931 0.2083 0.2226 0.2359 0.2477 0.2578 0.2660 0.2719 0.2756 0.2768 0.2756 0.2719 0.2660 0.2578 0.2477 0.2359 0.2226 0.2083 0.1931 0.1775 0.1617 0.1460 0.1306 0.1158 0.1018 0.0887 0.0766}

row 2 = {0.0887 0.1027 0.1179 0.1341 0.1512 0.1690 0.1872 0.2055 0.2236 0.2412 0.2578 0.2731 0.2868 0.2985 0.3080 0.3149 0.3191 0.3205 0.3191 0.3149 0.3080 0.2985 0.2868 0.2731 0.2578 0.2412 0.2236 0.2055 0.1872 0.1690 0.1512 0.1341 0.1179 0.1027 0.0887}

row 3 = {0.1018 0.1179 0.1353 0.1540 0.1736 0.1940 0.2149 0.2359 0.2567 0.2768 0.2959 0.3135 0.3292 0.3426 0.3535 0.3614 0.3662 0.3679 0.3662 0.3614 0.3535 0.3426 0.3292 0.3135 0.2959 0.2768 0.2567 0.2359 0.2149 0.1940 0.1736 0.1540 0.1353 0.1179 0.1018}

row 4 = {0.1158 0.1341 0.1540 0.1751 0.1975 0.2207 0.2444 0.2683 0.2920 0.3149 0.3366 0.3566 0.3745 0.3898 0.4021 0.4111 0.4166 0.4185 0.4166 0.4111 0.4021 0.3898 0.3745 0.3566 0.3366 0.3149 0.2920 0.2683 0.2444 0.2207 0.1975 0.1751 0.1540 0.1341 0.1158}

row 5 = {0.1306 0.1512 0.1736 0.1975 0.2226 0.2488 0.2756 0.3025 0.3292 0.3550 0.3795 0.4021 0.4222 0.4395 0.4533 0.4635 0.4697 0.4718 0.4697 0.4635 0.4533 0.4395 0.4222 0.4021 0.3795 0.3550 0.3292 0.3025 0.2756 0.2488 0.2226 0.1975 0.1736 0.1512 0.1306}

row 6 = {0.1460 0.1690 0.1940 0.2207 0.2488 0.2780 0.3080 0.3381 0.3679 0.3968 0.4241 0.4493 0.4718 0.4911 0.5066 0.5180 0.5250 0.5273 0.5250 0.5180 0.5066 0.4911 0.4718 0.4493 0.4241 0.3968 0.3679 0.3381 0.3080 0.2780 0.2488 0.2207 0.1940 0.1690 0.1460}

row 7 = {0.1617 0.1872 0.2149 0.2444 0.2756 0.3080 0.3411 0.3745 0.4075 0.4395 0.4697 0.4977 0.5226 0.5440 0.5611 0.5738 0.5815 0.5840 0.5815 0.5738 0.5611 0.5440 0.5226 0.4977 0.4697 0.4395 0.4075 0.3745 0.3411 0.3080 0.2756 0.2444 0.2149 0.1872 0.1617}

row 8 = {0.1775 0.2055 0.2359 0.2683 0.3025 0.3381 0.3745 0.4111 0.4473 0.4824 0.5157 0.5464 0.5738 0.5972 0.6160 0.6299 0.6383 0.6412 0.6383 0.6299 0.6160 0.5972 0.5738 0.5464 0.5157 0.4824 0.4473 0.4111 0.3745 0.3381 0.3025 0.2683 0.2359 0.2055 0.1775}

row 9 = {0.1931 0.2236 0.2567 0.2920 0.3292 0.3679 0.4075 0.4473 0.4868 0.5250 0.5611 0.5945 0.6243 0.6498 0.6703 0.6854 0.6946 0.6977 0.6946 0.6854 0.6703 0.6498 0.6243 0.5945 0.5611 0.5250 0.4868 0.4473 0.4075 0.3679 0.3292 0.2920 0.2567 0.2236 0.1931}

row 10 = {0.2083 0.2412 0.2768 0.3149 0.3550 0.3968 0.4395 0.4824 0.5250 0.5662 0.6052 0.6412 0.6733 0.7008 0.7229 0.7392 0.7491 0.7524 0.7491 0.7392 0.7229 0.7008 0.6733 0.6412 0.6052 0.5662 0.5250 0.4824 0.4395 0.3968 0.3550 0.3149 0.2768 0.2412 0.2083}

row 11 = {0.2226 0.2578 0.2959 0.3366 0.3795 0.4241 0.4697 0.5157 0.5611 0.6052 0.6469 0.6854 0.7197 0.7491 0.7728 0.7901 0.8007 0.8043 0.8007 0.7901 0.7728 0.7491 0.7197 0.6854 0.6469 0.6052 0.5611 0.5157 0.4697 0.4241 0.3795 0.3366 0.2959 0.2578 0.2226}

row 12 = {0.2359 0.2731 0.3135 0.3566 0.4021 0.4493 0.4977 0.5464 0.5945 0.6412 0.6854 0.7261 0.7625 0.7937 0.8187 0.8371 0.8484 0.8521 0.8484 0.8371 0.8187 0.7937 0.7625 0.7261 0.6854 0.6412 0.5945 0.5464 0.4977 0.4493 0.4021 0.3566 0.3135 0.2731 0.2359}

row 13 = {0.2477 0.2868 0.3292 0.3745 0.4222 0.4718 0.5226 0.5738 0.6243 0.6733 0.7197 0.7625 0.8007 0.8334 0.8598 0.8791 0.8909 0.8948 0.8909 0.8791 0.8598 0.8334 0.8007 0.7625 0.7197 0.6733 0.6243 0.5738 0.5226 0.4718 0.4222 0.3745 0.3292 0.2868 0.2477}

row 14 = {0.2578 0.2985 0.3426 0.3898 0.4395 0.4911 0.5440 0.5972 0.6498 0.7008 0.7491 0.7937 0.8334 0.8674 0.8948 0.9149 0.9272 0.9314 0.9272 0.9149 0.8948 0.8674 0.8334 0.7937 0.7491 0.7008 0.6498 0.5972 0.5440 0.4911 0.4395 0.3898 0.3426 0.2985 0.2578}

row 15 = {0.2660 0.3080 0.3535 0.4021 0.4533 0.5066 0.5611 0.6160 0.6703 0.7229 0.7728 0.8187 0.8598 0.8948 0.9231 0.9439 0.9565 0.9608 0.9565 0.9439 0.9231 0.8948 0.8598 0.8187 0.7728 0.7229 0.6703 0.6160 0.5611 0.5066 0.4533 0.4021 0.3535 0.3080 0.2660}

row 16 = {0.2719 0.3149 0.3614 0.4111 0.4635 0.5180 0.5738 0.6299 0.6854 0.7392 0.7901 0.8371 0.8791 0.9149 0.9439 0.9651 0.9780 0.9824 0.9780 0.9651 0.9439 0.9149 0.8791 0.8371 0.7901 0.7392 0.6854 0.6299 0.5738 0.5180 0.4635 0.4111 0.3614 0.3149 0.2719}

row 17 = {0.2756 0.3191 0.3662 0.4166 0.4697 0.5250 0.5815 0.6383 0.6946 0.7491 0.8007 0.8484 0.8909 0.9272 0.9565 0.9780 0.9912 0.9956 0.9912 0.9780 0.9565 0.9272 0.8909 0.8484 0.8007 0.7491 0.6946 0.6383 0.5815 0.5250 0.4697 0.4166 0.3662 0.3191 0.2756}

row 18 = {0.2768 0.3205 0.3679 0.4185 0.4718 0.5273 0.5840 0.6412 0.6977 0.7524 0.8043 0.8521 0.8948 0.9314 0.9608 0.9824 0.9956 1.0000 0.9956 0.9824 0.9608 0.9314 0.8948 0.8521 0.8043 0.7524 0.6977 0.6412 0.5840 0.5273 0.4718 0.4185 0.3679 0.3205 0.2768}

row 19 = {0.2756 0.3191 0.3662 0.4166 0.4697 0.5250 0.5815 0.6383 0.6946 0.7491 0.8007 0.8484 0.8909 0.9272 0.9565 0.9780 0.9912 0.9956 0.9912 0.9780 0.9565 0.9272 0.8909 0.8484 0.8007 0.7491 0.6946 0.6383 0.5815 0.5250 0.4697 0.4166 0.3662 0.3191 0.2756}

row 20 = {0.2719 0.3149 0.3614 0.4111 0.4635 0.5180 0.5738 0.6299 0.6854 0.7392 0.7901 0.8371 0.8791 0.9149 0.9439 0.9651 0.9780 0.9824 0.9780 0.9651 0.9439 0.9149 0.8791 0.8371 0.7901 0.7392 0.6854 0.6299 0.5738 0.5180 0.4635 0.4111 0.3614 0.3149 0.2719}

row 21 = {0.2660 0.3080 0.3535 0.4021 0.4533 0.5066 0.5611 0.6160 0.6703 0.7229 0.7728 0.8187 0.8598 0.8948 0.9231 0.9439 0.9565 0.9608 0.9565 0.9439 0.9231 0.8948 0.8598 0.8187 0.7728 0.7229 0.6703 0.6160 0.5611 0.5066 0.4533 0.4021 0.3535 0.3080 0.2660}

row 22 = {0.2578 0.2985 0.3426 0.3898 0.4395 0.4911 0.5440 0.5972 0.6498 0.7008 0.7491 0.7937 0.8334 0.8674 0.8948 0.9149 0.9272 0.9314 0.9272 0.9149 0.8948 0.8674 0.8334 0.7937 0.7491 0.7008 0.6498 0.5972 0.5440 0.4911 0.4395 0.3898 0.3426 0.2985 0.2578}

row 23 = {0.2477 0.2868 0.3292 0.3745 0.4222 0.4718 0.5226 0.5738 0.6243 0.6733 0.7197 0.7625 0.8007 0.8334 0.8598 0.8791 0.8909 0.8948 0.8909 0.8791 0.8598 0.8334 0.8007 0.7625 0.7197 0.6733 0.6243 0.5738 0.5226 0.4718 0.4222 0.3745 0.3292 0.2868 0.2477}

row 24 = {0.2359 0.2731 0.3135 0.3566 0.4021 0.4493 0.4977 0.5464 0.5945 0.6412 0.6854 0.7261 0.7625 0.7937 0.8187 0.8371 0.8484 0.8521 0.8484 0.8371 0.8187 0.7937 0.7625 0.7261 0.6854 0.6412 0.5945 0.5464 0.4977 0.4493 0.4021 0.3566 0.3135 0.2731 0.2359}

row 25 = {0.2226 0.2578 0.2959 0.3366 0.3795 0.4241 0.4697 0.5157 0.5611 0.6052 0.6469 0.6854 0.7197 0.7491 0.7728 0.7901 0.8007 0.8043 0.8007 0.7901 0.7728 0.7491 0.7197 0.6854 0.6469 0.6052 0.5611 0.5157 0.4697 0.4241 0.3795 0.3366 0.2959 0.2578 0.2226}

row 26 = {0.2083 0.2412 0.2768 0.3149 0.3550 0.3968 0.4395 0.4824 0.5250 0.5662 0.6052 0.6412 0.6733 0.7008 0.7229 0.7392 0.7491 0.7524 0.7491 0.7392 0.7229 0.7008 0.6733 0.6412 0.6052 0.5662 0.5250 0.4824 0.4395 0.3968 0.3550 0.3149 0.2768 0.2412 0.2083}

row 27 = {0.1931 0.2236 0.2567 0.2920 0.3292 0.3679 0.4075 0.4473 0.4868 0.5250 0.5611 0.5945 0.6243 0.6498 0.6703 0.6854 0.6946 0.6977 0.6946 0.6854 0.6703 0.6498 0.6243 0.5945 0.5611 0.5250 0.4868 0.4473 0.4075 0.3679 0.3292 0.2920 0.2567 0.2236 0.1931}

row 28 = {0.1775 0.2055 0.2359 0.2683 0.3025 0.3381 0.3745 0.4111 0.4473 0.4824 0.5157 0.5464 0.5738 0.5972 0.6160 0.6299 0.6383 0.6412 0.6383 0.6299 0.6160 0.5972 0.5738 0.5464 0.5157 0.4824 0.4473 0.4111 0.3745 0.3381 0.3025 0.2683 0.2359 0.2055 0.1775}

row 29 = {0.1617 0.1872 0.2149 0.2444 0.2756 0.3080 0.3411 0.3745 0.4075 0.4395 0.4697 0.4977 0.5226 0.5440 0.5611 0.5738 0.5815 0.5840 0.5815 0.5738 0.5611 0.5440 0.5226 0.4977 0.4697 0.4395 0.4075 0.3745 0.3411 0.3080 0.2756 0.2444 0.2149 0.1872 0.1617}

row 30 = {0.1460 0.1690 0.1940 0.2207 0.2488 0.2780 0.3080 0.3381 0.3679 0.3968 0.4241 0.4493 0.4718 0.4911 0.5066 0.5180 0.5250 0.5273 0.5250 0.5180 0.5066 0.4911 0.4718 0.4493 0.4241 0.3968 0.3679 0.3381 0.3080 0.2780 0.2488 0.2207 0.1940 0.1690 0.1460}

row 31 = {0.1306 0.1512 0.1736 0.1975 0.2226 0.2488 0.2756 0.3025 0.3292 0.3550 0.3795 0.4021 0.4222 0.4395 0.4533 0.4635 0.4697 0.4718 0.4697 0.4635 0.4533 0.4395 0.4222 0.4021 0.3795 0.3550 0.3292 0.3025 0.2756 0.2488 0.2226 0.1975 0.1736 0.1512 0.1306}

row 32 = {0.1158 0.1341 0.1540 0.1751 0.1975 0.2207 0.2444 0.2683 0.2920 0.3149 0.3366 0.3566 0.3745 0.3898 0.4021 0.4111 0.4166 0.4185 0.4166 0.4111 0.4021 0.3898 0.3745 0.3566 0.3366 0.3149 0.2920 0.2683 0.2444 0.2207 0.1975 0.1751 0.1540 0.1341 0.1158}

row 33 = {0.1018 0.1179 0.1353 0.1540 0.1736 0.1940 0.2149 0.2359 0.2567 0.2768 0.2959 0.3135 0.3292 0.3426 0.3535 0.3614 0.3662 0.3679 0.3662 0.3614 0.3535 0.3426 0.3292 0.3135 0.2959 0.2768 0.2567 0.2359 0.2149 0.1940 0.1736 0.1540 0.1353 0.1179 0.1018}

row 34 = {0.0887 0.1027 0.1179 0.1341 0.1512 0.1690 0.1872 0.2055 0.2236 0.2412 0.2578 0.2731 0.2868 0.2985 0.3080 0.3149 0.3191 0.3205 0.3191 0.3149 0.3080 0.2985 0.2868 0.2731 0.2578 0.2412 0.2236 0.2055 0.1872 0.1690 0.1512 0.1341 0.1179 0.1027 0.0887}

row 35 = {0.0766 0.0887 0.1018 0.1158 0.1306 0.1460 0.1617 0.1775 0.1931 0.2083 0.2226 0.2359 0.2477 0.2578 0.2660 0.2719 0.2756 0.2768 0.2756 0.2719 0.2660 0.2578 0.2477 0.2359 0.2226 0.2083 0.1931 0.1775 0.1617 0.1460 0.1306 0.1158 0.1018 0.0887 0.0766}

Medium random grid, 35 by 35, used in problem 10

row 1 = {0.9501 0.4186 0.7271 0.2259 0.6833 0.6756 0.9342 0.5527 0.7165 0.9159 0.9090 0.0712 0.8289 0.9327 0.9995 0.0129 0.7540 0.7157 0.2760 0.9418 0.5102 0.9070 0.0073 0.6155 0.2168 0.0311 0.1351 0.9697 0.3356 0.5422 0.2527 0.2491 0.7486 0.6363 0.9523}

row 2 = {0.2311 0.8462 0.3093 0.5798 0.2126 0.6992 0.2644 0.4001 0.5113 0.6020 0.5962 0.3143 0.1663 0.1310 0.2120 0.3104 0.6632 0.2507 0.3685 0.1500 0.7140 0.7586 0.5887 0.0034 0.6518 0.2886 0.2411 0.7148 0.2751 0.4557 0.5849 0.9249 0.3741 0.4010 0.4577}

row 3 = {0.6068 0.5252 0.8385 0.7604 0.8392 0.7275 0.1603 0.1988 0.7764 0.2536 0.3290 0.6084 0.3939 0.9408 0.4984 0.7791 0.8835 0.9339 0.0129 0.3844 0.5152 0.3807 0.5421 0.9820 0.0528 0.9711 0.9275 0.7820 0.0445 0.8631 0.5237 0.6295 0.4542 0.4866 0.5369}

row 4 = {0.4860 0.2026 0.5681 0.5298 0.6288 0.4784 0.8729 0.6252 0.4893 0.8735 0.4782 0.1750 0.5208 0.7019 0.2905 0.3073 0.2722 0.1372 0.8892 0.3111 0.6059 0.3311 0.6535 0.8995 0.2293 0.9505 0.3911 0.2376 0.0939 0.8552 0.1634 0.8783 0.0386 0.7505 0.0665}

row 5 = {0.8913 0.6721 0.3704 0.6405 0.1338 0.5548 0.2379 0.7334 0.1859 0.5134 0.5972 0.6210 0.7181 0.8477 0.6728 0.9267 0.4194 0.5216 0.8660 0.1685 0.9667 0.5041 0.3134 0.6928 0.6674 0.2280 0.5113 0.1957 0.4100 0.4723 0.4864 0.6417 0.5624 0.1262 0.4939}

row 6 = {0.7621 0.8381 0.7027 0.2091 0.2071 0.1210 0.6458 0.3759 0.7006 0.7327 0.1614 0.2460 0.5692 0.2093 0.9580 0.6787 0.2130 0.8952 0.2542 0.8966 0.8221 0.5646 0.2312 0.4397 0.3109 0.9585 0.0929 0.2632 0.8169 0.7869 0.4961 0.7984 0.3723 0.0431 0.4175}

row 7 = {0.4565 0.0196 0.5466 0.3798 0.6072 0.4508 0.9669 0.0099 0.9827 0.4222 0.8295 0.5874 0.4608 0.4551 0.7666 0.0743 0.0356 0.9424 0.5695 0.3227 0.3178 0.7672 0.4161 0.7010 0.3066 0.6799 0.0217 0.7138 0.8705 0.6560 0.8432 0.4350 0.7928 0.3709 0.2923}

row 8 = {0.0185 0.6813 0.4449 0.7833 0.6299 0.7159 0.6649 0.4199 0.8066 0.9614 0.9561 0.5061 0.4453 0.0811 0.6661 0.0707 0.0812 0.3351 0.1593 0.7340 0.5877 0.7799 0.2988 0.6097 0.7207 0.0550 0.1595 0.9776 0.0226 0.0000 0.8062 0.9811 0.7952 0.6933 0.2897}

row 9 = {0.8214 0.3795 0.6946 0.6808 0.3705 0.8928 0.8704 0.7537 0.7036 0.0721 0.5955 0.4648 0.0877 0.8511 0.1309 0.0119 0.8506 0.4374 0.5944 0.4109 0.1302 0.4841 0.6724 0.2999 0.9544 0.5998 0.8445 0.6371 0.7272 0.1312 0.8578 0.0960 0.3829 0.9358 0.7538}

row 10 = {0.4447 0.8318 0.6213 0.4611 0.5751 0.2731 0.0099 0.7939 0.4850 0.5534 0.0287 0.5414 0.4435 0.5620 0.0954 0.2272 0.3402 0.4712 0.3311 0.3998 0.2544 0.8022 0.9383 0.8560 0.1311 0.3931 0.8792 0.5459 0.8480 0.4949 0.6098 0.5275 0.2528 0.4776 0.0968}

row 11 = {0.6154 0.5028 0.7948 0.5678 0.4514 0.2548 0.1370 0.9200 0.1146 0.2920 0.8121 0.9423 0.3663 0.3193 0.0149 0.5163 0.4662 0.1493 0.6586 0.5055 0.8030 0.4710 0.3431 0.1121 0.0683 0.2154 0.1870 0.8481 0.7286 0.0383 0.5657 0.5456 0.3429 0.1291 0.0769}

row 12 = {0.7919 0.7095 0.9568 0.7942 0.0439 0.8656 0.8188 0.8447 0.6649 0.8580 0.6101 0.3418 0.3025 0.3749 0.2882 0.4582 0.9138 0.1359 0.8636 0.1693 0.6678 0.2028 0.5630 0.2916 0.1252 0.1824 0.9913 0.8021 0.9551 0.2274 0.6119 0.2843 0.9678 0.4838 0.7209}

row 13 = {0.9218 0.4289 0.5226 0.0592 0.0272 0.2324 0.4302 0.3678 0.3654 0.3358 0.7015 0.4018 0.8518 0.8678 0.8167 0.7032 0.2286 0.5325 0.5676 0.5247 0.0136 0.5796 0.1189 0.0974 0.1662 0.0768 0.7120 0.6683 0.6564 0.3279 0.1030 0.3708 0.4798 0.9456 0.7649}

row 14 = {0.7382 0.3046 0.8801 0.6029 0.3127 0.8049 0.8903 0.6208 0.1400 0.6802 0.0922 0.3077 0.7595 0.3722 0.9855 0.5825 0.8620 0.7258 0.9805 0.6412 0.5616 0.6665 0.1690 0.3974 0.9114 0.0074 0.8714 0.6710 0.7423 0.8995 0.1583 0.0647 0.3683 0.3677 0.6579}

row 15 = {0.1763 0.1897 0.1730 0.0503 0.0129 0.9084 0.7349 0.7313 0.5668 0.0534 0.4249 0.4116 0.9498 0.0737 0.0174 0.5092 0.6566 0.3987 0.7918 0.0162 0.4546 0.6768 0.2789 0.3333 0.1363 0.7888 0.4796 0.8206 0.3450 0.3137 0.4136 0.5448 0.7646 0.3285 0.8104}

row 16 = {0.4057 0.1934 0.9797 0.4154 0.3840 0.2319 0.6873 0.1939 0.8230 0.3567 0.3756 0.2859 0.5579 0.1998 0.8194 0.0743 0.8912 0.3584 0.1526 0.8369 0.9049 0.9425 0.5568 0.9442 0.6170 0.0178 0.4960 0.9705 0.8840 0.2517 0.5604 0.8364 0.3771 0.7729 0.3742}

row 17 = {0.9355 0.6822 0.2714 0.3050 0.6831 0.2393 0.3461 0.9048 0.6739 0.4983 0.1662 0.3941 0.0142 0.0495 0.6211 0.1932 0.4881 0.2853 0.8330 0.8035 0.2822 0.7701 0.4856 0.8386 0.2690 0.8779 0.2875 0.4869 0.3472 0.4330 0.2687 0.1453 0.9003 0.2973 0.3062}

row 18 = {0.9169 0.3028 0.2523 0.8744 0.0928 0.0498 0.1660 0.5692 0.9994 0.4344 0.8332 0.5030 0.5962 0.5667 0.5602 0.3796 0.9926 0.8686 0.1919 0.6978 0.0650 0.7374 0.9522 0.2584 0.2207 0.3525 0.0609 0.8175 0.0595 0.8424 0.7843 0.1715 0.1834 0.1779 0.3707}

row 19 = {0.4103 0.5417 0.8757 0.0150 0.0353 0.0784 0.1556 0.6318 0.9616 0.5625 0.8386 0.7220 0.8162 0.1219 0.2440 0.2764 0.3733 0.6264 0.6390 0.4619 0.4766 0.8663 0.2319 0.0429 0.7129 0.7221 0.2625 0.6416 0.7184 0.1845 0.3879 0.0680 0.3683 0.6908 0.7067}

row 20 = {0.8936 0.1509 0.7373 0.7680 0.6124 0.6408 0.1911 0.2344 0.0589 0.6166 0.4516 0.3062 0.9771 0.5221 0.8220 0.7709 0.5314 0.2412 0.6690 0.0826 0.9837 0.9909 0.4787 0.0059 0.5490 0.9685 0.1863 0.3063 0.9582 0.5082 0.0310 0.8240 0.9175 0.2639 0.1684}

row 21 = {0.0579 0.6979 0.1365 0.9708 0.6085 0.1909 0.4225 0.5488 0.3603 0.1133 0.9566 0.1122 0.2219 0.1171 0.2632 0.3139 0.1813 0.9781 0.7721 0.8207 0.9223 0.5039 0.5265 0.5744 0.9413 0.1557 0.9171 0.6609 0.1568 0.4522 0.5855 0.1340 0.5159 0.4577 0.8137}

row 22 = {0.3529 0.3784 0.0118 0.9901 0.0158 0.8439 0.8560 0.9316 0.5485 0.8983 0.1472 0.4433 0.7037 0.7699 0.7536 0.6382 0.5019 0.6405 0.3798 0.1930 0.5612 0.6291 0.7927 0.7439 0.3299 0.1630 0.1233 0.3580 0.4164 0.3256 0.5586 0.8848 0.0903 0.8437 0.4662}

row 23 = {0.8132 0.8600 0.8939 0.7889 0.0164 0.1739 0.4902 0.3352 0.2618 0.7546 0.8699 0.4668 0.5221 0.3751 0.6596 0.9866 0.4222 0.2298 0.4416 0.4454 0.6523 0.7926 0.1930 0.8068 0.7045 0.3134 0.0134 0.9382 0.0940 0.3801 0.2007 0.5147 0.7353 0.8815 0.7223}

row 24 = {0.0099 0.8537 0.1991 0.4387 0.1901 0.1708 0.8159 0.6555 0.5973 0.7911 0.7694 0.0147 0.9329 0.8234 0.2141 0.5029 0.6604 0.6813 0.4831 0.0130 0.7727 0.4486 0.9096 0.6376 0.9434 0.0294 0.3697 0.4877 0.4499 0.8865 0.0874 0.9636 0.0047 0.7000 0.9949}

row 25 = {0.1389 0.5936 0.2987 0.4983 0.5869 0.9943 0.4608 0.3919 0.0493 0.8150 0.4442 0.6641 0.7134 0.0466 0.6021 0.9477 0.6737 0.6658 0.6081 0.3087 0.1062 0.5244 0.9222 0.2513 0.5816 0.3576 0.6986 0.0910 0.8692 0.7613 0.9332 0.1205 0.6031 0.7557 0.3625}

row 26 = {0.2028 0.4966 0.6614 0.2140 0.0576 0.4398 0.4574 0.6273 0.5711 0.6700 0.6206 0.7241 0.2280 0.5979 0.6049 0.8280 0.9573 0.1347 0.1760 0.8754 0.0011 0.1715 0.0133 0.1443 0.8802 0.0272 0.8893 0.6738 0.3916 0.8838 0.2594 0.0483 0.9569 0.9745 0.7308}

row 27 = {0.1987 0.8998 0.2844 0.6435 0.3676 0.3400 0.4507 0.6991 0.7009 0.2009 0.9517 0.2816 0.4496 0.9492 0.6595 0.9176 0.1919 0.0225 0.0020 0.8353 0.5418 0.1307 0.7675 0.6516 0.7496 0.7937 0.5938 0.5149 0.2528 0.4574 0.2042 0.3802 0.3974 0.4022 0.6497}

row 28 = {0.6038 0.8216 0.4692 0.3200 0.6315 0.3142 0.4122 0.3972 0.9623 0.2731 0.6400 0.2618 0.1722 0.2888 0.1834 0.1131 0.1112 0.2622 0.7902 0.3331 0.0069 0.2188 0.9473 0.9461 0.3796 0.9992 0.1567 0.2216 0.3544 0.7992 0.0492 0.4128 0.7316 0.1313 0.6813}

row 29 = {0.2722 0.6449 0.0648 0.9601 0.7176 0.3651 0.9016 0.4136 0.7505 0.6262 0.2473 0.7085 0.9688 0.8888 0.6365 0.8121 0.5651 0.1165 0.5136 0.8807 0.4513 0.1055 0.8133 0.8159 0.7256 0.1102 0.3167 0.7250 0.7430 0.1341 0.6062 0.4014 0.6846 0.7247 0.0076}

row 30 = {0.1988 0.8180 0.9883 0.7266 0.6927 0.3932 0.0056 0.6552 0.7400 0.5369 0.3527 0.7839 0.3557 0.1016 0.1703 0.9083 0.9692 0.0693 0.2132 0.4797 0.1957 0.1414 0.9238 0.9302 0.1628 0.6226 0.2334 0.0682 0.6508 0.0653 0.5463 0.4210 0.9785 0.8995 0.6541}

row 31 = {0.0153 0.6602 0.5828 0.4120 0.0841 0.5915 0.2974 0.8376 0.4319 0.0595 0.1879 0.9862 0.0490 0.0653 0.5396 0.1564 0.0237 0.8529 0.1034 0.5608 0.7871 0.4570 0.1990 0.3099 0.9562 0.1326 0.0084 0.9641 0.9398 0.3751 0.0958 0.3770 0.2038 0.1707 0.9452}

row 32 = {0.7468 0.3420 0.4235 0.7446 0.4544 0.1197 0.0492 0.3716 0.6343 0.0890 0.4906 0.4733 0.7553 0.2343 0.6234 0.1221 0.8702 0.1803 0.1573 0.6159 0.6186 0.7881 0.6743 0.2688 0.1962 0.3100 0.3969 0.2077 0.8328 0.3735 0.6370 0.9073 0.5933 0.0430 0.6133}

row 33 = {0.4451 0.2897 0.5155 0.2679 0.4418 0.0381 0.6932 0.4253 0.8030 0.2713 0.4093 0.9028 0.8948 0.9331 0.6859 0.7627 0.0269 0.0324 0.4075 0.6619 0.0155 0.2811 0.9271 0.5365 0.7762 0.1348 0.6499 0.1611 0.4700 0.4840 0.4429 0.6702 0.9516 0.4792 0.7829}

row 34 = {0.9318 0.3412 0.3340 0.4399 0.3533 0.4586 0.6501 0.5947 0.0839 0.4091 0.4635 0.4511 0.2861 0.0631 0.6773 0.7218 0.5195 0.7339 0.4078 0.6166 0.8909 0.2248 0.3438 0.1633 0.6133 0.2233 0.0850 0.6382 0.6299 0.9695 0.0664 0.9618 0.2603 0.0939 0.0032}

row 35 = {0.4660 0.5341 0.4329 0.9334 0.1536 0.8699 0.9830 0.5657 0.9455 0.4740 0.6109 0.8045 0.2512 0.2642 0.8768 0.6516 0.1923 0.5365 0.0527 0.6851 0.7617 0.9089 0.5945 0.2110 0.1623 0.3965 0.7688 0.0002 0.0582 0.3421 0.3743 0.1630 0.5147 0.6500 0.7970}

Large hill grid #1, 50 by 50, used in problem 11

row 1 = {0.0561 0.0631 0.0707 0.0787 0.0872 0.0961 0.1054 0.1150 0.1249 0.1350 0.1451 0.1553 0.1653 0.1751 0.1845 0.1935 0.2019 0.2096 0.2165 0.2226 0.2276 0.2317 0.2346 0.2363 0.2369 0.2363 0.2346 0.2317 0.2276 0.2226 0.2165 0.2096 0.2019 0.1935 0.1845 0.1751 0.1653 0.1553 0.1451 0.1350 0.1249 0.1150 0.1054 0.0961 0.0872 0.0787 0.0707 0.0631 0.0561 0.0497}

row 2 = {0.0631 0.0710 0.0795 0.0885 0.0980 0.1081 0.1185 0.1294 0.1405 0.1518 0.1632 0.1746 0.1859 0.1969 0.2075 0.2176 0.2271 0.2357 0.2435 0.2503 0.2560 0.2605 0.2638 0.2658 0.2665 0.2658 0.2638 0.2605 0.2560 0.2503 0.2435 0.2357 0.2271 0.2176 0.2075 0.1969 0.1859 0.1746 0.1632 0.1518 0.1405 0.1294 0.1185 0.1081 0.0980 0.0885 0.0795 0.0710 0.0631 0.0559}

row 3 = {0.0707 0.0795 0.0889 0.0990 0.1097 0.1209 0.1327 0.1448 0.1572 0.1699 0.1827 0.1954 0.2080 0.2204 0.2322 0.2435 0.2541 0.2638 0.2725 0.2801 0.2865 0.2916 0.2952 0.2975 0.2982 0.2975 0.2952 0.2916 0.2865 0.2801 0.2725 0.2638 0.2541 0.2435 0.2322 0.2204 0.2080 0.1954 0.1827 0.1699 0.1572 0.1448 0.1327 0.1209 0.1097 0.0990 0.0889 0.0795 0.0707 0.0625}

row 4 = {0.0787 0.0885 0.0990 0.1103 0.1222 0.1347 0.1477 0.1612 0.1751 0.1892 0.2034 0.2176 0.2317 0.2454 0.2586 0.2712 0.2829 0.2938 0.3035 0.3119 0.3190 0.3247 0.3287 0.3312 0.3320 0.3312 0.3287 0.3247 0.3190 0.3119 0.3035 0.2938 0.2829 0.2712 0.2586 0.2454 0.2317 0.2176 0.2034 0.1892 0.1751 0.1612 0.1477 0.1347 0.1222 0.1103 0.0990 0.0885 0.0787 0.0696}

row 5 = {0.0872 0.0980 0.1097 0.1222 0.1353 0.1492 0.1637 0.1786 0.1940 0.2096 0.2254 0.2411 0.2567 0.2719 0.2865 0.3004 0.3135 0.3255 0.3362 0.3456 0.3535 0.3597 0.3642 0.3670 0.3679 0.3670 0.3642 0.3597 0.3535 0.3456 0.3362 0.3255 0.3135 0.3004 0.2865 0.2719 0.2567 0.2411 0.2254 0.2096 0.1940 0.1786 0.1637 0.1492 0.1353 0.1222 0.1097 0.0980 0.0872 0.0771}

row 6 = {0.0961 0.1081 0.1209 0.1347 0.1492 0.1645 0.1804 0.1969 0.2138 0.2311 0.2485 0.2658 0.2829 0.2997 0.3158 0.3312 0.3456 0.3588 0.3706 0.3810 0.3897 0.3965 0.4015 0.4045 0.4056 0.4045 0.4015 0.3965 0.3897 0.3810 0.3706 0.3588 0.3456 0.3312 0.3158 0.2997 0.2829 0.2658 0.2485 0.2311 0.2138 0.1969 0.1804 0.1645 0.1492 0.1347 0.1209 0.1081 0.0961 0.0850}

row 7 = {0.1054 0.1185 0.1327 0.1477 0.1637 0.1804 0.1979 0.2160 0.2346 0.2535 0.2725 0.2916 0.3104 0.3287 0.3465 0.3633 0.3791 0.3936 0.4066 0.4179 0.4274 0.4350 0.4404 0.4437 0.4449 0.4437 0.4404 0.4350 0.4274 0.4179 0.4066 0.3936 0.3791 0.3633 0.3465 0.3287 0.3104 0.2916 0.2725 0.2535 0.2346 0.2160 0.1979 0.1804 0.1637 0.1477 0.1327 0.1185 0.1054 0.0932}

row 8 = {0.1150 0.1294 0.1448 0.1612 0.1786 0.1969 0.2160 0.2357 0.2560 0.2767 0.2975 0.3182 0.3387 0.3588 0.3781 0.3965 0.4137 0.4296 0.4437 0.4561 0.4665 0.4747 0.4807 0.4843 0.4855 0.4843 0.4807 0.4747 0.4665 0.4561 0.4437 0.4296 0.4137 0.3965 0.3781 0.3588 0.3387 0.3182 0.2975 0.2767 0.2560 0.2357 0.2160 0.1969 0.1786 0.1612 0.1448 0.1294 0.1150 0.1018}

row 9 = {0.1249 0.1405 0.1572 0.1751 0.1940 0.2138 0.2346 0.2560 0.2780 0.3004 0.3230 0.3456 0.3679 0.3897 0.4107 0.4306 0.4493 0.4665 0.4819 0.4953 0.5066 0.5156 0.5220 0.5260 0.5273 0.5260 0.5220 0.5156 0.5066 0.4953 0.4819 0.4665 0.4493 0.4306 0.4107 0.3897 0.3679 0.3456 0.3230 0.3004 0.2780 0.2560 0.2346 0.2138 0.1940 0.1751 0.1572 0.1405 0.1249 0.1105}

row 10 = {0.1350 0.1518 0.1699 0.1892 0.2096 0.2311 0.2535 0.2767 0.3004 0.3247 0.3491 0.3734 0.3975 0.4211 0.4437 0.4653 0.4855 0.5041 0.5207 0.5353 0.5474 0.5571 0.5641 0.5684 0.5698 0.5684 0.5641 0.5571 0.5474 0.5353 0.5207 0.5041 0.4855 0.4653 0.4437 0.4211 0.3975 0.3734 0.3491 0.3247 0.3004 0.2767 0.2535 0.2311 0.2096 0.1892 0.1699 0.1518 0.1350 0.1194}

row 11 = {0.1451 0.1632 0.1827 0.2034 0.2254 0.2485 0.2725 0.2975 0.3230 0.3491 0.3753 0.4015 0.4274 0.4527 0.4771 0.5003 0.5220 0.5420 0.5599 0.5755 0.5886 0.5990 0.6065 0.6111 0.6126 0.6111 0.6065 0.5990 0.5886 0.5755 0.5599 0.5420 0.5220 0.5003 0.4771 0.4527 0.4274 0.4015 0.3753 0.3491 0.3230 0.2975 0.2725 0.2485 0.2254 0.2034 0.1827 0.1632 0.1451 0.1284}

row 12 = {0.1553 0.1746 0.1954 0.2176 0.2411 0.2658 0.2916 0.3182 0.3456 0.3734 0.4015 0.4296 0.4573 0.4843 0.5104 0.5353 0.5585 0.5798 0.5990 0.6157 0.6297 0.6408 0.6489 0.6538 0.6554 0.6538 0.6489 0.6408 0.6297 0.6157 0.5990 0.5798 0.5585 0.5353 0.5104 0.4843 0.4573 0.4296 0.4015 0.3734 0.3456 0.3182 0.2916 0.2658 0.2411 0.2176 0.1954 0.1746 0.1553 0.1374}

row 13 = {0.1653 0.1859 0.2080 0.2317 0.2567 0.2829 0.3104 0.3387 0.3679 0.3975 0.4274 0.4573 0.4868 0.5156 0.5434 0.5698 0.5945 0.6172 0.6376 0.6554 0.6703 0.6822 0.6907 0.6959 0.6977 0.6959 0.6907 0.6822 0.6703 0.6554 0.6376 0.6172 0.5945 0.5698 0.5434 0.5156 0.4868 0.4573 0.4274 0.3975 0.3679 0.3387 0.3104 0.2829 0.2567 0.2317 0.2080 0.1859 0.1653 0.1462}

row 14 = {0.1751 0.1969 0.2204 0.2454 0.2719 0.2997 0.3287 0.3588 0.3897 0.4211 0.4527 0.4843 0.5156 0.5461 0.5755 0.6035 0.6297 0.6538 0.6754 0.6942 0.7100 0.7225 0.7316 0.7371 0.7390 0.7371 0.7316 0.7225 0.7100 0.6942 0.6754 0.6538 0.6297 0.6035 0.5755 0.5461 0.5156 0.4843 0.4527 0.4211 0.3897 0.3588 0.3287 0.2997 0.2719 0.2454 0.2204 0.1969 0.1751 0.1549}

row 15 = {0.1845 0.2075 0.2322 0.2586 0.2865 0.3158 0.3465 0.3781 0.4107 0.4437 0.4771 0.5104 0.5434 0.5755 0.6065 0.6360 0.6637 0.6890 0.7118 0.7316 0.7483 0.7615 0.7711 0.7769 0.7788 0.7769 0.7711 0.7615 0.7483 0.7316 0.7118 0.6890 0.6637 0.6360 0.6065 0.5755 0.5434 0.5104 0.4771 0.4437 0.4107 0.3781 0.3465 0.3158 0.2865 0.2586 0.2322 0.2075 0.1845 0.1632}

row 16 = {0.1935 0.2176 0.2435 0.2712 0.3004 0.3312 0.3633 0.3965 0.4306 0.4653 0.5003 0.5353 0.5698 0.6035 0.6360 0.6670 0.6959 0.7225 0.7464 0.7672 0.7847 0.7985 0.8086 0.8146 0.8167 0.8146 0.8086 0.7985 0.7847 0.7672 0.7464 0.7225 0.6959 0.6670 0.6360 0.6035 0.5698 0.5353 0.5003 0.4653 0.4306 0.3965 0.3633 0.3312 0.3004 0.2712 0.2435 0.2176 0.1935 0.1712}

row 17 = {0.2019 0.2271 0.2541 0.2829 0.3135 0.3456 0.3791 0.4137 0.4493 0.4855 0.5220 0.5585 0.5945 0.6297 0.6637 0.6959 0.7261 0.7539 0.7788 0.8005 0.8187 0.8332 0.8437 0.8500 0.8521 0.8500 0.8437 0.8332 0.8187 0.8005 0.7788 0.7539 0.7261 0.6959 0.6637 0.6297 0.5945 0.5585 0.5220 0.4855 0.4493 0.4137 0.3791 0.3456 0.3135 0.2829 0.2541 0.2271 0.2019 0.1786}

row 18 = {0.2096 0.2357 0.2638 0.2938 0.3255 0.3588 0.3936 0.4296 0.4665 0.5041 0.5420 0.5798 0.6172 0.6538 0.6890 0.7225 0.7539 0.7827 0.8086 0.8311 0.8500 0.8650 0.8759 0.8825 0.8847

0.8825 0.8759 0.8650 0.8500 0.8311 0.8086 0.7827 0.7539 0.7225 0.6890 0.6538 0.6172 0.5798 0.5420 0.5041 0.4665 0.4296 0.3936 0.3588 0.3255 0.2938 0.2638 0.2357 0.2096 0.1854}

row 19 = {0.2165 0.2435 0.2725 0.3035 0.3362 0.3706 0.4066 0.4437 0.4819 0.5207 0.5599 0.5990 0.6376 0.6754 0.7118 0.7464 0.7788 0.8086 0.8353 0.8586 0.8781 0.8936 0.9048 0.9116 0.9139 0.9116 0.9048 0.8936 0.8781 0.8586 0.8353 0.8086 0.7788 0.7464 0.7118 0.6754 0.6376 0.5990 0.5599 0.5207 0.4819 0.4437 0.4066 0.3706 0.3362 0.3035 0.2725 0.2435 0.2165 0.1916}

row 20 = {0.2226 0.2503 0.2801 0.3119 0.3456 0.3810 0.4179 0.4561 0.4953 0.5353 0.5755 0.6157 0.6554 0.6942 0.7316 0.7672 0.8005 0.8311 0.8586 0.8825 0.9026 0.9185 0.9301 0.9371 0.9394 0.9371 0.9301 0.9185 0.9026 0.8825 0.8586 0.8311 0.8005 0.7672 0.7316 0.6942 0.6554 0.6157 0.5755 0.5353 0.4953 0.4561 0.4179 0.3810 0.3456 0.3119 0.2801 0.2503 0.2226 0.1969}

row 21 = {0.2276 0.2560 0.2865 0.3190 0.3535 0.3897 0.4274 0.4665 0.5066 0.5474 0.5886 0.6297 0.6703 0.7100 0.7483 0.7847 0.8187 0.8500 0.8781 0.9026 0.9231 0.9394 0.9512 0.9584 0.9608 0.9584 0.9512 0.9394 0.9231 0.9026 0.8781 0.8500 0.8187 0.7847 0.7483 0.7100 0.6703 0.6297 0.5886 0.5474 0.5066 0.4665 0.4274 0.3897 0.3535 0.3190 0.2865 0.2560 0.2276 0.2014}

row 22 = {0.2317 0.2605 0.2916 0.3247 0.3597 0.3965 0.4350 0.4747 0.5156 0.5571 0.5990 0.6408 0.6822 0.7225 0.7615 0.7985 0.8332 0.8650 0.8936 0.9185 0.9394 0.9560 0.9680 0.9753 0.9778 0.9753 0.9680 0.9560 0.9394 0.9185 0.8936 0.8650 0.8332 0.7985 0.7615 0.7225 0.6822 0.6408 0.5990 0.5571 0.5156 0.4747 0.4350 0.3965 0.3597 0.3247 0.2916 0.2605 0.2317 0.2049}

row 23 = {0.2346 0.2638 0.2952 0.3287 0.3642 0.4015 0.4404 0.4807 0.5220 0.5641 0.6065 0.6489 0.6907 0.7316 0.7711 0.8086 0.8437 0.8759 0.9048 0.9301 0.9512 0.9680 0.9802 0.9876 0.9900 0.9876 0.9802 0.9680 0.9512 0.9301 0.9048 0.8759 0.8437 0.8086 0.7711 0.7316 0.6907 0.6489 0.6065 0.5641 0.5220 0.4807 0.4404 0.4015 0.3642 0.3287 0.2952 0.2638 0.2346 0.2075}

row 24 = {0.2363 0.2658 0.2975 0.3312 0.3670 0.4045 0.4437 0.4843 0.5260 0.5684 0.6111 0.6538 0.6959 0.7371 0.7769 0.8146 0.8500 0.8825 0.9116 0.9371 0.9584 0.9753 0.9876 0.9950 0.9975 0.9950 0.9876 0.9753 0.9584 0.9371 0.9116 0.8825 0.8500 0.8146 0.7769 0.7371 0.6959 0.6538 0.6111 0.5684 0.5260 0.4843 0.4437 0.4045 0.3670 0.3312 0.2975 0.2658 0.2363 0.2091}

row 25 = {0.2369 0.2665 0.2982 0.3320 0.3679 0.4056 0.4449 0.4855 0.5273 0.5698 0.6126 0.6554 0.6977 0.7390 0.7788 0.8167 0.8521 0.8847 0.9139 0.9394 0.9608 0.9778 0.9900 0.9975 1.0000 0.9975 0.9900 0.9778 0.9608 0.9394 0.9139 0.8847 0.8521 0.8167 0.7788 0.7390 0.6977 0.6554 0.6126 0.5698 0.5273 0.4855 0.4449 0.4056 0.3679 0.3320 0.2982 0.2665 0.2369 0.2096}

row 26 = {0.2363 0.2658 0.2975 0.3312 0.3670 0.4045 0.4437 0.4843 0.5260 0.5684 0.6111 0.6538 0.6959 0.7371 0.7769 0.8146 0.8500 0.8825 0.9116 0.9371 0.9584 0.9753 0.9876 0.9950 0.9975 0.9950 0.9876 0.9753 0.9584 0.9371 0.9116 0.8825 0.8500 0.8146 0.7769 0.7371 0.6959 0.6538 0.6111 0.5684 0.5260 0.4843 0.4437 0.4045 0.3670 0.3312 0.2975 0.2658 0.2363 0.2091}

row 27 = {0.2346 0.2638 0.2952 0.3287 0.3642 0.4015 0.4404 0.4807 0.5220 0.5641 0.6065 0.6489 0.6907 0.7316 0.7711 0.8086 0.8437 0.8759 0.9048 0.9301 0.9512 0.9680 0.9802 0.9876 0.9900 0.9876 0.9802 0.9680 0.9512 0.9301 0.9048 0.8759 0.8437 0.8086 0.7711 0.7316 0.6907 0.6489 0.6065 0.5641 0.5220 0.4807 0.4404 0.4015 0.3642 0.3287 0.2952 0.2638 0.2346 0.2075}

row 28 = {0.2317 0.2605 0.2916 0.3247 0.3597 0.3965 0.4350 0.4747 0.5156 0.5571 0.5990 0.6408 0.6822 0.7225 0.7615 0.7985 0.8332 0.8650 0.8936 0.9185 0.9394 0.9560 0.9680 0.9753 0.9778 0.9753 0.9680 0.9560 0.9394 0.9185 0.8936 0.8650 0.8332 0.7985 0.7615 0.7225 0.6822 0.6408 0.5990 0.5571 0.5156 0.4747 0.4350 0.3965 0.3597 0.3247 0.2916 0.2605 0.2317 0.2049}

row 29 = {0.2276 0.2560 0.2865 0.3190 0.3535 0.3897 0.4274 0.4665 0.5066 0.5474 0.5886 0.6297 0.6703 0.7100 0.7483 0.7847 0.8187 0.8500 0.8781 0.9026 0.9231 0.9394 0.9512 0.9584 0.9608 0.9584 0.9512 0.9394 0.9231 0.9026 0.8781 0.8500 0.8187 0.7847 0.7483 0.7100 0.6703 0.6297 0.5886 0.5474 0.5066 0.4665 0.4274 0.3897 0.3535 0.3190 0.2865 0.2560 0.2276 0.2014}

row 30 = {0.2226 0.2503 0.2801 0.3119 0.3456 0.3810 0.4179 0.4561 0.4953 0.5353 0.5755 0.6157 0.6554 0.6942 0.7316 0.7672 0.8005 0.8311 0.8586 0.8825 0.9026 0.9185 0.9301 0.9371 0.9394 0.9371 0.9301 0.9185 0.9026 0.8825 0.8586 0.8311 0.8005 0.7672 0.7316 0.6942 0.6554 0.6157 0.5755 0.5353 0.4953 0.4561 0.4179 0.3810 0.3456 0.3119 0.2801 0.2503 0.2226 0.1969}

row 31 = {0.2165 0.2435 0.2725 0.3035 0.3362 0.3706 0.4066 0.4437 0.4819 0.5207 0.5599 0.5990 0.6376 0.6754 0.7118 0.7464 0.7788 0.8086 0.8353 0.8586 0.8781 0.8936 0.9048 0.9116 0.9139 0.9116 0.9048 0.8936 0.8781 0.8586 0.8353 0.8086 0.7788 0.7464 0.7118 0.6754 0.6376 0.5990 0.5599 0.5207 0.4819 0.4437 0.4066 0.3706 0.3362 0.3035 0.2725 0.2435 0.2165 0.1916}

row 32 = {0.2096 0.2357 0.2638 0.2938 0.3255 0.3588 0.3936 0.4296 0.4665 0.5041 0.5420 0.5798 0.6172 0.6538 0.6890 0.7225 0.7539 0.7827 0.8086 0.8311 0.8500 0.8650 0.8759 0.8825 0.8847 0.8825 0.8759 0.8650 0.8500 0.8311 0.8086 0.7827 0.7539 0.7225 0.6890 0.6538 0.6172 0.5798 0.5420 0.5041 0.4665 0.4296 0.3936 0.3588 0.3255 0.2938 0.2638 0.2357 0.2096 0.1854}

row 33 = {0.2019 0.2271 0.2541 0.2829 0.3135 0.3456 0.3791 0.4137 0.4493 0.4855 0.5220 0.5585 0.5945 0.6297 0.6637 0.6959 0.7261 0.7539 0.7788 0.8005 0.8187 0.8332 0.8437 0.8500 0.8521 0.8500 0.8437 0.8332 0.8187 0.8005 0.7788 0.7539 0.7261 0.6959 0.6637 0.6297 0.5945 0.5585 0.5220 0.4855 0.4493 0.4137 0.3791 0.3456 0.3135 0.2829 0.2541 0.2271 0.2019 0.1786}

row 34 = {0.1935 0.2176 0.2435 0.2712 0.3004 0.3312 0.3633 0.3965 0.4306 0.4653 0.5003 0.5353 0.5698 0.6035 0.6360 0.6670 0.6959 0.7225 0.7464 0.7672 0.7847 0.7985 0.8086 0.8146 0.8167 0.8146 0.8086 0.7985 0.7847 0.7672 0.7464 0.7225 0.6959 0.6670 0.6360 0.6035 0.5698 0.5353 0.5003 0.4653 0.4306 0.3965 0.3633 0.3312 0.3004 0.2712 0.2435 0.2176 0.1935 0.1712}

row 35 = {0.1845 0.2075 0.2322 0.2586 0.2865 0.3158 0.3465 0.3781 0.4107 0.4437 0.4771 0.5104 0.5434 0.5755 0.6065 0.6360 0.6637 0.6890 0.7118 0.7316 0.7483 0.7615 0.7711 0.7769 0.7788 0.7769 0.7711 0.7615 0.7483 0.7316 0.7118 0.6890 0.6637 0.6360 0.6065 0.5755 0.5434 0.5104 0.4771 0.4437 0.4107 0.3781 0.3465 0.3158 0.2865 0.2586 0.2322 0.2075 0.1845 0.1632}

row 36 = {0.1751 0.1969 0.2204 0.2454 0.2719 0.2997 0.3287 0.3588 0.3897 0.4211 0.4527 0.4843 0.5156 0.5461 0.5755 0.6035 0.6297 0.6538 0.6754 0.6942 0.7100 0.7225 0.7316 0.7371 0.7390 0.7371 0.7316 0.7225 0.7100 0.6942 0.6754 0.6538 0.6297 0.6035 0.5755 0.5461 0.5156 0.4843 0.4527 0.4211 0.3897 0.3588 0.3287 0.2997 0.2719 0.2454 0.2204 0.1969 0.1751 0.1549}

row 37 = {0.1653 0.1859 0.2080 0.2317 0.2567 0.2829 0.3104 0.3387 0.3679 0.3975 0.4274 0.4573 0.4868 0.5156 0.5434 0.5698 0.5945 0.6172 0.6376 0.6554 0.6703 0.6822 0.6907 0.6959 0.6977 0.6959 0.6907 0.6822 0.6703 0.6554 0.6376 0.6172 0.5945 0.5698 0.5434 0.5156 0.4868 0.4573 0.4274 0.3975 0.3679 0.3387 0.3104 0.2829 0.2567 0.2317 0.2080 0.1859 0.1653 0.1462}

row 38 = {0.1553 0.1746 0.1954 0.2176 0.2411 0.2658 0.2916 0.3182 0.3456 0.3734 0.4015 0.4296 0.4573 0.4843 0.5104 0.5353 0.5585 0.5798 0.5990 0.6157 0.6297 0.6408 0.6489 0.6538 0.6554 0.6538 0.6489 0.6408 0.6297 0.6157 0.5990 0.5798 0.5585 0.5353 0.5104 0.4843 0.4573 0.4296 0.4015 0.3734 0.3456 0.3182 0.2916 0.2658 0.2411 0.2176 0.1954 0.1746 0.1553 0.1374}

row 39 = {0.1451 0.1632 0.1827 0.2034 0.2254 0.2485 0.2725 0.2975 0.3230 0.3491 0.3753 0.4015 0.4274 0.4527 0.4771 0.5003 0.5220 0.5420 0.5599 0.5755 0.5886 0.5990 0.6065 0.6111 0.6126 0.6111 0.6065 0.5990 0.5886 0.5755 0.5599 0.5420 0.5220 0.5003 0.4771 0.4527 0.4274 0.4015 0.3753 0.3491 0.3230 0.2975 0.2725 0.2485 0.2254 0.2034 0.1827 0.1632 0.1451 0.1284}

row 40 = {0.1350 0.1518 0.1699 0.1892 0.2096 0.2311 0.2535 0.2767 0.3004 0.3247 0.3491 0.3734 0.3975 0.4211 0.4437 0.4653 0.4855 0.5041 0.5207 0.5353 0.5474 0.5571 0.5641 0.5684 0.5698 0.5684 0.5641 0.5571 0.5474 0.5353 0.5207 0.5041 0.4855 0.4653 0.4437 0.4211 0.3975 0.3734 0.3491 0.3247 0.3004 0.2767 0.2535 0.2311 0.2096 0.1892 0.1699 0.1518 0.1350 0.1194}

row 41 = {0.1249 0.1405 0.1572 0.1751 0.1940 0.2138 0.2346 0.2560 0.2780 0.3004 0.3230 0.3456 0.3679 0.3897 0.4107 0.4306 0.4493 0.4665 0.4819 0.4953 0.5066 0.5156 0.5220 0.5260 0.5273 0.5260 0.5220 0.5156 0.5066 0.4953 0.4819 0.4665 0.4493 0.4306 0.4107 0.3897 0.3679 0.3456 0.3230 0.3004 0.2780 0.2560 0.2346 0.2138 0.1940 0.1751 0.1572 0.1405 0.1249 0.1105}

row 42 = {0.1150 0.1294 0.1448 0.1612 0.1786 0.1969 0.2160 0.2357 0.2560 0.2767 0.2975 0.3182 0.3387 0.3588 0.3781 0.3965 0.4137 0.4296 0.4437 0.4561 0.4665 0.4747 0.4807 0.4843 0.4855 0.4843 0.4807 0.4747 0.4665 0.4561 0.4437 0.4296 0.4137 0.3965 0.3781 0.3588 0.3387 0.3182 0.2975 0.2767 0.2560 0.2357 0.2160 0.1969 0.1786 0.1612 0.1448 0.1294 0.1150 0.1018}

row 43 = {0.1054 0.1185 0.1327 0.1477 0.1637 0.1804 0.1979 0.2160 0.2346 0.2535 0.2725 0.2916 0.3104 0.3287 0.3465 0.3633 0.3791 0.3936 0.4066 0.4179 0.4274 0.4350 0.4404 0.4437 0.4449 0.4437 0.4404 0.4350 0.4274 0.4179 0.4066 0.3936 0.3791 0.3633 0.3465 0.3287 0.3104 0.2916 0.2725 0.2535 0.2346 0.2160 0.1979 0.1804 0.1637 0.1477 0.1327 0.1185 0.1054 0.0932}

row 44 = {0.0961 0.1081 0.1209 0.1347 0.1492 0.1645 0.1804 0.1969 0.2138 0.2311 0.2485 0.2658 0.2829 0.2997 0.3158 0.3312 0.3456 0.3588 0.3706 0.3810 0.3897 0.3965 0.4015 0.4045 0.4056 0.4045 0.4015 0.3965 0.3897 0.3810 0.3706 0.3588 0.3456 0.3312 0.3158 0.2997 0.2829 0.2658 0.2485 0.2311 0.2138 0.1969 0.1804 0.1645 0.1492 0.1347 0.1209 0.1081 0.0961 0.0850}

row 45 = {0.0872 0.0980 0.1097 0.1222 0.1353 0.1492 0.1637 0.1786 0.1940 0.2096 0.2254 0.2411 0.2567 0.2719 0.2865 0.3004 0.3135 0.3255 0.3362 0.3456 0.3535 0.3597 0.3642 0.3670 0.3679

0.3670 0.3642 0.3597 0.3535 0.3456 0.3362 0.3255 0.3135 0.3004 0.2865 0.2719 0.2567 0.2411 0.2254 0.2096 0.1940 0.1786 0.1637 0.1492 0.1353 0.1222 0.1097 0.0980 0.0872 0.0771}

row 46 = {0.0787 0.0885 0.0990 0.1103 0.1222 0.1347 0.1477 0.1612 0.1751 0.1892 0.2034 0.2176 0.2317 0.2454 0.2586 0.2712 0.2829 0.2938 0.3035 0.3119 0.3190 0.3247 0.3287 0.3312 0.3320 0.3312 0.3287 0.3247 0.3190 0.3119 0.3035 0.2938 0.2829 0.2712 0.2586 0.2454 0.2317 0.2176 0.2034 0.1892 0.1751 0.1612 0.1477 0.1347 0.1222 0.1103 0.0990 0.0885 0.0787 0.0696}

row 47 = {0.0707 0.0795 0.0889 0.0990 0.1097 0.1209 0.1327 0.1448 0.1572 0.1699 0.1827 0.1954 0.2080 0.2204 0.2322 0.2435 0.2541 0.2638 0.2725 0.2801 0.2865 0.2916 0.2952 0.2975 0.2982 0.2975 0.2952 0.2916 0.2865 0.2801 0.2725 0.2638 0.2541 0.2435 0.2322 0.2204 0.2080 0.1954 0.1827 0.1699 0.1572 0.1448 0.1327 0.1209 0.1097 0.0990 0.0889 0.0795 0.0707 0.0625}

row 48 = {0.0631 0.0710 0.0795 0.0885 0.0980 0.1081 0.1185 0.1294 0.1405 0.1518 0.1632 0.1746 0.1859 0.1969 0.2075 0.2176 0.2271 0.2357 0.2435 0.2503 0.2560 0.2605 0.2638 0.2658 0.2665 0.2658 0.2638 0.2605 0.2560 0.2503 0.2435 0.2357 0.2271 0.2176 0.2075 0.1969 0.1859 0.1746 0.1632 0.1518 0.1405 0.1294 0.1185 0.1081 0.0980 0.0885 0.0795 0.0710 0.0631 0.0559}

row 49 = {0.0561 0.0631 0.0707 0.0787 0.0872 0.0961 0.1054 0.1150 0.1249 0.1350 0.1451 0.1553 0.1653 0.1751 0.1845 0.1935 0.2019 0.2096 0.2165 0.2226 0.2276 0.2317 0.2346 0.2363 0.2369 0.2363 0.2346 0.2317 0.2276 0.2226 0.2165 0.2096 0.2019 0.1935 0.1845 0.1751 0.1653 0.1553 0.1451 0.1350 0.1249 0.1150 0.1054 0.0961 0.0872 0.0787 0.0707 0.0631 0.0561 0.0497}

row 50 = {0.0497 0.0559 0.0625 0.0696 0.0771 0.0850 0.0932 0.1018 0.1105 0.1194 0.1284 0.1374 0.1462 0.1549 0.1632 0.1712 0.1786 0.1854 0.1916 0.1969 0.2014 0.2049 0.2075 0.2091 0.2096 0.2091 0.2075 0.2049 0.2014 0.1969 0.1916 0.1854 0.1786 0.1712 0.1632 0.1549 0.1462 0.1374 0.1284 0.1194 0.1105 0.1018 0.0932 0.0850 0.0771 0.0696 0.0625 0.0559 0.0497 0.0439}

Large 2-pit grid, 50 by 50, used in problem 12

row 1 = {0.9991 0.9989 0.9985 0.9981 0.9975 0.9967 0.9957 0.9943 0.9926 0.9903 0.9875 0.9838 0.9793 0.9737 0.9668 0.9585 0.9486 0.9369 0.9232 0.9074 0.8895 0.8695 0.8473 0.8230 0.7970 0.7695 0.7409 0.7117 0.6825 0.6538 0.6263 0.6007 0.5777 0.5578 0.5418 0.5300 0.5227 0.5203 0.5228 0.5300 0.5419 0.5579 0.5778 0.6008 0.6264 0.6540 0.6827 0.7120 0.7413 0.7699}

row 2 = {0.9989 0.9986 0.9982 0.9976 0.9969 0.9960 0.9949 0.9934 0.9914 0.9888 0.9856 0.9815 0.9764 0.9701 0.9624 0.9531 0.9419 0.9287 0.9134 0.8957 0.8756 0.8530 0.8281 0.8009 0.7716 0.7407 0.7086 0.6757 0.6428 0.6106 0.5797 0.5509 0.5250 0.5028 0.4847 0.4714 0.4633 0.4606 0.4633 0.4715 0.4848 0.5029 0.5252 0.5511 0.5799 0.6109 0.6432 0.6762 0.7091 0.7413}

row 3 = {0.9985 0.9982 0.9977 0.9971 0.9963 0.9953 0.9939 0.9922 0.9900 0.9872 0.9836 0.9790 0.9733 0.9663 0.9577 0.9473 0.9349 0.9203 0.9032 0.8835 0.8612 0.8361 0.8084 0.7781 0.7456 0.7112 0.6755 0.6389 0.6023 0.5664 0.5321 0.5001 0.4713 0.4465 0.4264 0.4116 0.4026 0.3995 0.4026 0.4117 0.4265 0.4467 0.4715 0.5004 0.5324 0.5669 0.6029 0.6395 0.6762 0.7120}

row 4 = {0.9981 0.9976 0.9971 0.9964 0.9955 0.9943 0.9928 0.9909 0.9884 0.9853 0.9813 0.9762 0.9700 0.9623 0.9528 0.9414 0.9277 0.9116 0.8929 0.8712 0.8466 0.8190 0.7885 0.7552 0.7194 0.6816 0.6422 0.6020 0.5617 0.5222 0.4843 0.4491 0.4174 0.3901 0.3680 0.3517 0.3417 0.3384 0.3418 0.3518 0.3682 0.3903 0.4177 0.4495 0.4848 0.5228 0.5624 0.6029 0.6432 0.6827}

row 5 = {0.9975 0.9969 0.9963 0.9955 0.9944 0.9931 0.9914 0.9893 0.9866 0.9831 0.9787 0.9732 0.9664 0.9580 0.9477 0.9353 0.9204 0.9029 0.8824 0.8589 0.8321 0.8020 0.7688 0.7325 0.6936 0.6523 0.6095 0.5657 0.5217 0.4787 0.4374 0.3990 0.3645 0.3347 0.3106 0.2929 0.2820 0.2784 0.2821 0.2931 0.3109 0.3351 0.3649 0.3996 0.4381 0.4795 0.5228 0.5669 0.6109 0.6540}

row 6 = {0.9967 0.9960 0.9953 0.9943 0.9931 0.9916 0.9897 0.9874 0.9844 0.9806 0.9758 0.9699 0.9625 0.9534 0.9423 0.9289 0.9130 0.8941 0.8721 0.8467 0.8179 0.7855 0.7497 0.7106 0.6686 0.6242 0.5780 0.5307 0.4834 0.4369 0.3924 0.3510 0.3138 0.2817 0.2557 0.2366 0.2249 0.2210 0.2250 0.2368 0.2561 0.2822 0.3144 0.3518 0.3934 0.4381 0.4848 0.5324 0.5799 0.6264}

row 7 = {0.9957 0.9949 0.9939 0.9928 0.9914 0.9897 0.9876 0.9850 0.9818 0.9777 0.9725 0.9662 0.9583 0.9486 0.9367 0.9225 0.9054 0.8853 0.8619 0.8349 0.8042 0.7697 0.7315 0.6898 0.6451 0.5977 0.5484 0.4980 0.4475 0.3979 0.3505 0.3063 0.2665 0.2322 0.2045 0.1841 0.1716 0.1675 0.1718 0.1844 0.2050 0.2329 0.2674 0.3074 0.3518 0.3996 0.4495 0.5004 0.5511 0.6008}

row 8 = {0.9943 0.9934 0.9922 0.9909 0.9893 0.9874 0.9850 0.9822 0.9786 0.9742 0.9687 0.9619 0.9536 0.9433 0.9309 0.9158 0.8979 0.8767 0.8520 0.8236 0.7912 0.7549 0.7146 0.6707 0.6235

0.5735 0.5215 0.4683 0.4150 0.3626 0.3125 0.2658 0.2238 0.1877 0.1584 0.1369 0.1237 0.1193 0.1239 0.1373 0.1591 0.1886 0.2251 0.2674 0.3144 0.3649 0.4177 0.4715 0.5252 0.5778}

row 9 = {0.9926 0.9914 0.9900 0.9884 0.9866 0.9844 0.9818 0.9786 0.9748 0.9700 0.9642 0.9571 0.9483 0.9376 0.9246 0.9089 0.8902 0.8682 0.8425 0.8129 0.7792 0.7413 0.6994 0.6536 0.6043 0.5522 0.4979 0.4423 0.3867 0.3320 0.2796 0.2309 0.1870 0.1492 0.1186 0.0961 0.0824 0.0778 0.0826 0.0967 0.1195 0.1504 0.1886 0.2329 0.2822 0.3351 0.3903 0.4467 0.5029 0.5579}

row 10 = {0.9903 0.9888 0.9872 0.9853 0.9831 0.9806 0.9777 0.9742 0.9700 0.9650 0.9589 0.9514 0.9423 0.9312 0.9178 0.9017 0.8825 0.8598 0.8333 0.8029 0.7682 0.7292 0.6860 0.6388 0.5880 0.5341 0.4781 0.4208 0.3632 0.3068 0.2526 0.2022 0.1569 0.1178 0.0862 0.0629 0.0488 0.0441 0.0492 0.0637 0.0874 0.1195 0.1591 0.2050 0.2561 0.3109 0.3682 0.4265 0.4848 0.5419}

row 11 = {0.9875 0.9856 0.9836 0.9813 0.9787 0.9758 0.9725 0.9687 0.9642 0.9589 0.9525 0.9447 0.9354 0.9240 0.9103 0.8939 0.8744 0.8514 0.8245 0.7935 0.7582 0.7186 0.6746 0.6265 0.5747 0.5198 0.4626 0.4041 0.3453 0.2876 0.2323 0.1808 0.1344 0.0945 0.0621 0.0384 0.0239 0.0192 0.0244 0.0394 0.0637 0.0967 0.1373 0.1844 0.2368 0.2931 0.3518 0.4117 0.4715 0.5300}

row 12 = {0.9838 0.9815 0.9790 0.9762 0.9732 0.9699 0.9662 0.9619 0.9571 0.9514 0.9447 0.9368 0.9272 0.9157 0.9019 0.8854 0.8659 0.8428 0.8158 0.7848 0.7493 0.7095 0.6653 0.6169 0.5647 0.5094 0.4517 0.3926 0.3333 0.2750 0.2191 0.1670 0.1201 0.0797 0.0471 0.0230 0.0085 0.0038 0.0091 0.0244 0.0492 0.0826 0.1239 0.1718 0.2250 0.2821 0.3418 0.4026 0.4633 0.5228}

row 13 = {0.9793 0.9764 0.9733 0.9700 0.9664 0.9625 0.9583 0.9536 0.9483 0.9423 0.9354 0.9272 0.9176 0.9061 0.8923 0.8760 0.8566 0.8338 0.8072 0.7764 0.7414 0.7019 0.6580 0.6099 0.5580 0.5029 0.4455 0.3865 0.3273 0.2691 0.2133 0.1613 0.1144 0.0740 0.0414 0.0174 0.0029 -0.0017 0.0038 0.0192 0.0441 0.0778 0.1193 0.1675 0.2210 0.2784 0.3384 0.3995 0.4606 0.5203}

row 14 = {0.9737 0.9701 0.9663 0.9623 0.9580 0.9534 0.9486 0.9433 0.9376 0.9312 0.9240 0.9157 0.9061 0.8947 0.8812 0.8652 0.8464 0.8241 0.7982 0.7683 0.7341 0.6955 0.6526 0.6054 0.5545 0.5004 0.4439 0.3859 0.3275 0.2701 0.2150 0.1636 0.1174 0.0775 0.0453 0.0216 0.0074 0.0029 0.0085 0.0239 0.0488 0.0824 0.1237 0.1716 0.2249 0.2820 0.3417 0.4026 0.4633 0.5227}

row 15 = {0.9668 0.9624 0.9577 0.9528 0.9477 0.9423 0.9367 0.9309 0.9246 0.9178 0.9103 0.9019 0.8923 0.8812 0.8682 0.8528 0.8347 0.8135 0.7887 0.7600 0.7272 0.6901 0.6487 0.6033 0.5540 0.5016 0.4468 0.3904 0.3337 0.2778 0.2241 0.1739 0.1288 0.0899 0.0585 0.0355 0.0216 0.0174 0.0230 0.0384 0.0629 0.0961 0.1369 0.1841 0.2366 0.2929 0.3517 0.4116 0.4714 0.5300}

row 16 = {0.9585 0.9531 0.9473 0.9414 0.9353 0.9289 0.9225 0.9158 0.9089 0.9017 0.8939 0.8854 0.8760 0.8652 0.8528 0.8383 0.8213 0.8014 0.7782 0.7512 0.7203 0.6853 0.6461 0.6030 0.5561 0.5061 0.4537 0.3997 0.3453 0.2916 0.2399 0.1917 0.1483 0.1108 0.0806 0.0585 0.0453 0.0414 0.0471 0.0621 0.0862 0.1186 0.1584 0.2045 0.2557 0.3106 0.3680 0.4264 0.4847 0.5418}

row 17 = {0.9486 0.9419 0.9349 0.9277 0.9204 0.9130 0.9054 0.8979 0.8902 0.8825 0.8744 0.8659 0.8566 0.8464 0.8347 0.8213 0.8057 0.7875 0.7662 0.7415 0.7130 0.6806 0.6443 0.6041 0.5603 0.5134 0.4641 0.4132 0.3618 0.3110 0.2621 0.2163 0.1751 0.1395 0.1108 0.0899 0.0775 0.0740 0.0797 0.0945 0.1178 0.1492 0.1877 0.2322 0.2817 0.3347 0.3901 0.4465 0.5028 0.5578}

row 18 = {0.9369 0.9287 0.9203 0.9116 0.9029 0.8941 0.8853 0.8767 0.8682 0.8598 0.8514 0.8428 0.8338 0.8241 0.8135 0.8014 0.7875 0.7713 0.7523 0.7303 0.7048 0.6756 0.6427 0.6061 0.5660 0.5229 0.4774 0.4303 0.3825 0.3352 0.2896 0.2469 0.2083 0.1751 0.1483 0.1288 0.1174 0.1144 0.1201 0.1344 0.1569 0.1870 0.2238 0.2665 0.3138 0.3645 0.4174 0.4713 0.5250 0.5777}

row 19 = {0.9232 0.9134 0.9032 0.8929 0.8824 0.8721 0.8619 0.8520 0.8425 0.8333 0.8245 0.8158 0.8072 0.7982 0.7887 0.7782 0.7662 0.7523 0.7361 0.7172 0.6951 0.6697 0.6408 0.6083 0.5726 0.5339 0.4928 0.4501 0.4066 0.3633 0.3215 0.2823 0.2469 0.2163 0.1917 0.1739 0.1636 0.1613 0.1670 0.1808 0.2022 0.2309 0.2658 0.3063 0.3510 0.3990 0.4491 0.5001 0.5509 0.6007}

row 20 = {0.9074 0.8957 0.8835 0.8712 0.8589 0.8467 0.8349 0.8236 0.8129 0.8029 0.7935 0.7848 0.7764 0.7683 0.7600 0.7512 0.7415 0.7303 0.7172 0.7018 0.6836 0.6624 0.6380 0.6102 0.5793 0.5456 0.5095 0.4717 0.4330 0.3943 0.3568 0.3215 0.2896 0.2621 0.2399 0.2241 0.2150 0.2133 0.2191 0.2323 0.2526 0.2796 0.3125 0.3505 0.3924 0.4374 0.4843 0.5321 0.5797 0.6263}

row 21 = {0.8895 0.8756 0.8612 0.8466 0.8321 0.8179 0.8042 0.7912 0.7792 0.7682 0.7582 0.7493 0.7414 0.7341 0.7272 0.7203 0.7130 0.7048 0.6951 0.6836 0.6697 0.6532 0.6337 0.6111 0.5856 0.5573 0.5267 0.4943 0.4608 0.4272 0.3943 0.3633 0.3352 0.3110 0.2916 0.2778 0.2701 0.2691 0.2750 0.2876 0.3068 0.3320 0.3626 0.3979 0.4369 0.4787 0.5222 0.5664 0.6106 0.6538}

row 22 = {0.8695 0.8530 0.8361 0.8190 0.8020 0.7855 0.7697 0.7549 0.7413 0.7292 0.7186 0.7095 0.7019 0.6955 0.6901 0.6853 0.6806 0.6756 0.6697 0.6624 0.6532 0.6416 0.6275 0.6106 0.5908 0.5684 0.5436 0.5169 0.4891 0.4608 0.4330 0.4066 0.3825 0.3618 0.3453 0.3337 0.3275 0.3273 0.3333 0.3453 0.3632 0.3867 0.4150 0.4475 0.4834 0.5217 0.5617 0.6023 0.6428 0.6825}

row 23 = {0.8473 0.8281 0.8084 0.7885 0.7688 0.7497 0.7315 0.7146 0.6994 0.6860 0.6746 0.6653 0.6580 0.6526 0.6487 0.6461 0.6443 0.6427 0.6408 0.6380 0.6337 0.6275 0.6190 0.6080 0.5943 0.5781 0.5595 0.5389 0.5169 0.4943 0.4717 0.4501 0.4303 0.4132 0.3997 0.3904 0.3859 0.3865 0.3926 0.4041 0.4208 0.4423 0.4683 0.4980 0.5307 0.5657 0.6020 0.6389 0.6757 0.7117}

row 24 = {0.8230 0.8009 0.7781 0.7552 0.7325 0.7106 0.6898 0.6707 0.6536 0.6388 0.6265 0.6169 0.6099 0.6054 0.6033 0.6030 0.6041 0.6061 0.6083 0.6102 0.6111 0.6106 0.6080 0.6032 0.5958 0.5860 0.5737 0.5595 0.5436 0.5267 0.5095 0.4928 0.4774 0.4641 0.4537 0.4468 0.4439 0.4455 0.4517 0.4626 0.4781 0.4979 0.5215 0.5484 0.5780 0.6095 0.6422 0.6755 0.7086 0.7409}

row 25 = {0.7970 0.7716 0.7456 0.7194 0.6936 0.6686 0.6451 0.6235 0.6043 0.5880 0.5747 0.5647 0.5580 0.5545 0.5540 0.5561 0.5603 0.5660 0.5726 0.5793 0.5856 0.5908 0.5943 0.5958 0.5950 0.5917 0.5860 0.5781 0.5684 0.5573 0.5456 0.5339 0.5229 0.5134 0.5061 0.5016 0.5004 0.5029 0.5094 0.5198 0.5341 0.5522 0.5735 0.5977 0.6242 0.6523 0.6816 0.7112 0.7407 0.7695}

row 26 = {0.7695 0.7407 0.7112 0.6816 0.6523 0.6242 0.5977 0.5735 0.5522 0.5341 0.5198 0.5094 0.5029 0.5004 0.5016 0.5061 0.5134 0.5229 0.5339 0.5456 0.5573 0.5684 0.5781 0.5860 0.5917 0.5950 0.5958 0.5943 0.5908 0.5856 0.5793 0.5726 0.5660 0.5603 0.5561 0.5540 0.5545 0.5580 0.5647 0.5747 0.5880 0.6043 0.6235 0.6451 0.6686 0.6936 0.7194 0.7456 0.7716 0.7970}

row 27 = {0.7409 0.7086 0.6755 0.6422 0.6095 0.5780 0.5484 0.5215 0.4979 0.4781 0.4626 0.4517 0.4455 0.4439 0.4468 0.4537 0.4641 0.4774 0.4928 0.5095 0.5267 0.5436 0.5595 0.5737 0.5860 0.5958 0.6032 0.6080 0.6106 0.6111 0.6102 0.6083 0.6061 0.6041 0.6030 0.6033 0.6054 0.6099 0.6169 0.6265 0.6388 0.6536 0.6707 0.6898 0.7106 0.7325 0.7552 0.7781 0.8009 0.8230}

row 28 = {0.7117 0.6757 0.6389 0.6020 0.5657 0.5307 0.4980 0.4683 0.4423 0.4208 0.4041 0.3926 0.3865 0.3859 0.3904 0.3997 0.4132 0.4303 0.4501 0.4717 0.4943 0.5169 0.5389 0.5595 0.5781 0.5943 0.6080 0.6190 0.6275 0.6337 0.6380 0.6408 0.6427 0.6443 0.6461 0.6487 0.6526 0.6580 0.6653 0.6746 0.6860 0.6994 0.7146 0.7315 0.7497 0.7688 0.7885 0.8084 0.8281 0.8473}

row 29 = {0.6825 0.6428 0.6023 0.5617 0.5217 0.4834 0.4475 0.4150 0.3867 0.3632 0.3453 0.3333 0.3273 0.3275 0.3337 0.3453 0.3618 0.3825 0.4066 0.4330 0.4608 0.4891 0.5169 0.5436 0.5684 0.5908 0.6106 0.6275 0.6416 0.6532 0.6624 0.6697 0.6756 0.6806 0.6853 0.6901 0.6955 0.7019 0.7095 0.7186 0.7292 0.7413 0.7549 0.7697 0.7855 0.8020 0.8190 0.8361 0.8530 0.8695}

row 30 = {0.6538 0.6106 0.5664 0.5222 0.4787 0.4369 0.3979 0.3626 0.3320 0.3068 0.2876 0.2750 0.2691 0.2701 0.2778 0.2916 0.3110 0.3352 0.3633 0.3943 0.4272 0.4608 0.4943 0.5267 0.5573 0.5856 0.6111 0.6337 0.6532 0.6697 0.6836 0.6951 0.7048 0.7130 0.7203 0.7272 0.7341 0.7414 0.7493 0.7582 0.7682 0.7792 0.7912 0.8042 0.8179 0.8321 0.8466 0.8612 0.8756 0.8895}

row 31 = {0.6263 0.5797 0.5321 0.4843 0.4374 0.3924 0.3505 0.3125 0.2796 0.2526 0.2323 0.2191 0.2133 0.2150 0.2241 0.2399 0.2621 0.2896 0.3215 0.3568 0.3943 0.4330 0.4717 0.5095 0.5456 0.5793 0.6102 0.6380 0.6624 0.6836 0.7018 0.7172 0.7303 0.7415 0.7512 0.7600 0.7683 0.7764 0.7848 0.7935 0.8029 0.8129 0.8236 0.8349 0.8467 0.8589 0.8712 0.8835 0.8957 0.9074}

row 32 = {0.6007 0.5509 0.5001 0.4491 0.3990 0.3510 0.3063 0.2658 0.2309 0.2022 0.1808 0.1670 0.1613 0.1636 0.1739 0.1917 0.2163 0.2469 0.2823 0.3215 0.3633 0.4066 0.4501 0.4928 0.5339 0.5726 0.6083 0.6408 0.6697 0.6951 0.7172 0.7361 0.7523 0.7662 0.7782 0.7887 0.7982 0.8072 0.8158 0.8245 0.8333 0.8425 0.8520 0.8619 0.8721 0.8824 0.8929 0.9032 0.9134 0.9232}

row 33 = {0.5777 0.5250 0.4713 0.4174 0.3645 0.3138 0.2665 0.2238 0.1870 0.1569 0.1344 0.1201 0.1144 0.1174 0.1288 0.1483 0.1751 0.2083 0.2469 0.2896 0.3352 0.3825 0.4303 0.4774 0.5229 0.5660 0.6061 0.6427 0.6756 0.7048 0.7303 0.7523 0.7713 0.7875 0.8014 0.8135 0.8241 0.8338 0.8428 0.8514 0.8598 0.8682 0.8767 0.8853 0.8941 0.9029 0.9116 0.9203 0.9287 0.9369}

row 34 = {0.5578 0.5028 0.4465 0.3901 0.3347 0.2817 0.2322 0.1877 0.1492 0.1178 0.0945 0.0797 0.0740 0.0775 0.0899 0.1108 0.1395 0.1751 0.2163 0.2621 0.3110 0.3618 0.4132 0.4641 0.5134 0.5603 0.6041 0.6443 0.6806 0.7130 0.7415 0.7662 0.7875 0.8057 0.8213 0.8347 0.8464 0.8566 0.8659 0.8744 0.8825 0.8902 0.8979 0.9054 0.9130 0.9204 0.9277 0.9349 0.9419 0.9486}

row 35 = {0.5418 0.4847 0.4264 0.3680 0.3106 0.2557 0.2045 0.1584 0.1186 0.0862 0.0621 0.0471 0.0414 0.0453 0.0585 0.0806 0.1108 0.1483 0.1917 0.2399 0.2916 0.3453 0.3997 0.4537 0.5061

0.5561 0.6030 0.6461 0.6853 0.7203 0.7512 0.7782 0.8014 0.8213 0.8383 0.8528 0.8652 0.8760 0.8854 0.8939 0.9017 0.9089 0.9158 0.9225 0.9289 0.9353 0.9414 0.9473 0.9531 0.9585}

row 36 = {0.5300 0.4714 0.4116 0.3517 0.2929 0.2366 0.1841 0.1369 0.0961 0.0629 0.0384 0.0230 0.0174 0.0216 0.0355 0.0585 0.0899 0.1288 0.1739 0.2241 0.2778 0.3337 0.3904 0.4468 0.5016 0.5540 0.6033 0.6487 0.6901 0.7272 0.7600 0.7887 0.8135 0.8347 0.8528 0.8682 0.8812 0.8923 0.9019 0.9103 0.9178 0.9246 0.9309 0.9367 0.9423 0.9477 0.9528 0.9577 0.9624 0.9668}

row 37 = {0.5227 0.4633 0.4026 0.3417 0.2820 0.2249 0.1716 0.1237 0.0824 0.0488 0.0239 0.0085 0.0029 0.0074 0.0216 0.0453 0.0775 0.1174 0.1636 0.2150 0.2701 0.3275 0.3859 0.4439 0.5004 0.5545 0.6054 0.6526 0.6955 0.7341 0.7683 0.7982 0.8241 0.8464 0.8652 0.8812 0.8947 0.9061 0.9157 0.9240 0.9312 0.9376 0.9433 0.9486 0.9534 0.9580 0.9623 0.9663 0.9701 0.9737}

row 38 = {0.5203 0.4606 0.3995 0.3384 0.2784 0.2210 0.1675 0.1193 0.0778 0.0441 0.0192 0.0038 -0.0017 0.0029 0.0174 0.0414 0.0740 0.1144 0.1613 0.2133 0.2691 0.3273 0.3865 0.4455 0.5029 0.5580 0.6099 0.6580 0.7019 0.7414 0.7764 0.8072 0.8338 0.8566 0.8760 0.8923 0.9061 0.9176 0.9272 0.9354 0.9423 0.9483 0.9536 0.9583 0.9625 0.9664 0.9700 0.9733 0.9764 0.9793}

row 39 = {0.5228 0.4633 0.4026 0.3418 0.2821 0.2250 0.1718 0.1239 0.0826 0.0492 0.0244 0.0091 0.0038 0.0085 0.0230 0.0471 0.0797 0.1201 0.1670 0.2191 0.2750 0.3333 0.3926 0.4517 0.5094 0.5647 0.6169 0.6653 0.7095 0.7493 0.7848 0.8158 0.8428 0.8659 0.8854 0.9019 0.9157 0.9272 0.9368 0.9447 0.9514 0.9571 0.9619 0.9662 0.9699 0.9732 0.9762 0.9790 0.9815 0.9838}

row 40 = {0.5300 0.4715 0.4117 0.3518 0.2931 0.2368 0.1844 0.1373 0.0967 0.0637 0.0394 0.0244 0.0192 0.0239 0.0384 0.0621 0.0945 0.1344 0.1808 0.2323 0.2876 0.3453 0.4041 0.4626 0.5198 0.5747 0.6265 0.6746 0.7186 0.7582 0.7935 0.8245 0.8514 0.8744 0.8939 0.9103 0.9240 0.9354 0.9447 0.9525 0.9589 0.9642 0.9687 0.9725 0.9758 0.9787 0.9813 0.9836 0.9856 0.9875}

row 41 = {0.5419 0.4848 0.4265 0.3682 0.3109 0.2561 0.2050 0.1591 0.1195 0.0874 0.0637 0.0492 0.0441 0.0488 0.0629 0.0862 0.1178 0.1569 0.2022 0.2526 0.3068 0.3632 0.4208 0.4781 0.5341 0.5880 0.6388 0.6860 0.7292 0.7682 0.8029 0.8333 0.8598 0.8825 0.9017 0.9178 0.9312 0.9423 0.9514 0.9589 0.9650 0.9700 0.9742 0.9777 0.9806 0.9831 0.9853 0.9872 0.9888 0.9903}

row 42 = {0.5579 0.5029 0.4467 0.3903 0.3351 0.2822 0.2329 0.1886 0.1504 0.1195 0.0967 0.0826 0.0778 0.0824 0.0961 0.1186 0.1492 0.1870 0.2309 0.2796 0.3320 0.3867 0.4423 0.4979 0.5522 0.6043 0.6536 0.6994 0.7413 0.7792 0.8129 0.8425 0.8682 0.8902 0.9089 0.9246 0.9376 0.9483 0.9571 0.9642 0.9700 0.9748 0.9786 0.9818 0.9844 0.9866 0.9884 0.9900 0.9914 0.9926}

row 43 = {0.5778 0.5252 0.4715 0.4177 0.3649 0.3144 0.2674 0.2251 0.1886 0.1591 0.1373 0.1239 0.1193 0.1237 0.1369 0.1584 0.1877 0.2238 0.2658 0.3125 0.3626 0.4150 0.4683 0.5215 0.5735 0.6235 0.6707 0.7146 0.7549 0.7912 0.8236 0.8520 0.8767 0.8979 0.9158 0.9309 0.9433 0.9536 0.9619 0.9687 0.9742 0.9786 0.9822 0.9850 0.9874 0.9893 0.9909 0.9922 0.9934 0.9943}

row 44 = {0.6008 0.5511 0.5004 0.4495 0.3996 0.3518 0.3074 0.2674 0.2329 0.2050 0.1844 0.1718 0.1675 0.1716 0.1841 0.2045 0.2322 0.2665 0.3063 0.3505 0.3979 0.4475 0.4980 0.5484 0.5977 0.6451 0.6898 0.7315 0.7697 0.8042 0.8349 0.8619 0.8853 0.9054 0.9225 0.9367 0.9486 0.9583 0.9662 0.9725 0.9777 0.9818 0.9850 0.9876 0.9897 0.9914 0.9928 0.9939 0.9949 0.9957}

row 45 = {0.6264 0.5799 0.5324 0.4848 0.4381 0.3934 0.3518 0.3144 0.2822 0.2561 0.2368 0.2250 0.2210 0.2249 0.2366 0.2557 0.2817 0.3138 0.3510 0.3924 0.4369 0.4834 0.5307 0.5780 0.6242 0.6686 0.7106 0.7497 0.7855 0.8179 0.8467 0.8721 0.8941 0.9130 0.9289 0.9423 0.9534 0.9625 0.9699 0.9758 0.9806 0.9844 0.9874 0.9897 0.9916 0.9931 0.9943 0.9953 0.9960 0.9967}

row 46 = {0.6540 0.6109 0.5669 0.5228 0.4795 0.4381 0.3996 0.3649 0.3351 0.3109 0.2931 0.2821 0.2784 0.2820 0.2929 0.3106 0.3347 0.3645 0.3990 0.4374 0.4787 0.5217 0.5657 0.6095 0.6523 0.6936 0.7325 0.7688 0.8020 0.8321 0.8589 0.8824 0.9029 0.9204 0.9353 0.9477 0.9580 0.9664 0.9732 0.9787 0.9831 0.9866 0.9893 0.9914 0.9931 0.9944 0.9955 0.9963 0.9969 0.9975}

row 47 = {0.6827 0.6432 0.6029 0.5624 0.5228 0.4848 0.4495 0.4177 0.3903 0.3682 0.3518 0.3418 0.3384 0.3417 0.3517 0.3680 0.3901 0.4174 0.4491 0.4843 0.5222 0.5617 0.6020 0.6422 0.6816 0.7194 0.7552 0.7885 0.8190 0.8466 0.8712 0.8929 0.9116 0.9277 0.9414 0.9528 0.9623 0.9700 0.9762 0.9813 0.9853 0.9884 0.9909 0.9928 0.9943 0.9955 0.9964 0.9971 0.9976 0.9981}

row 48 = {0.7120 0.6762 0.6395 0.6029 0.5669 0.5324 0.5004 0.4715 0.4467 0.4265 0.4117 0.4026 0.3995 0.4026 0.4116 0.4264 0.4465 0.4713 0.5001 0.5321 0.5664 0.6023 0.6389 0.6755 0.7112 0.7456 0.7781 0.8084 0.8361 0.8612 0.8835 0.9032 0.9203 0.9349 0.9473 0.9577 0.9663 0.9733 0.9790 0.9836 0.9872 0.9900 0.9922 0.9939 0.9953 0.9963 0.9971 0.9977 0.9982 0.9985}

row 49 = {0.7413 0.7091 0.6762 0.6432 0.6109 0.5799 0.5511 0.5252 0.5029 0.4848 0.4715 0.4633 0.4606 0.4633 0.4714 0.4847 0.5028 0.5250 0.5509 0.5797 0.6106 0.6428 0.6757 0.7086 0.7407 0.7716 0.8009 0.8281 0.8530 0.8756 0.8957 0.9134 0.9287 0.9419 0.9531 0.9624 0.9701 0.9764 0.9815 0.9856 0.9888 0.9914 0.9934 0.9949 0.9960 0.9969 0.9976 0.9982 0.9986 0.9989}

row 50 = {0.7699 0.7413 0.7120 0.6827 0.6540 0.6264 0.6008 0.5778 0.5579 0.5419 0.5300 0.5228 0.5203 0.5227 0.5300 0.5418 0.5578 0.5777 0.6007 0.6263 0.6538 0.6825 0.7117 0.7409 0.7695 0.7970 0.8230 0.8473 0.8695 0.8895 0.9074 0.9232 0.9369 0.9486 0.9585 0.9668 0.9737 0.9793 0.9838 0.9875 0.9903 0.9926 0.9943 0.9957 0.9967 0.9975 0.9981 0.9985 0.9989 0.9991}

Large hill grid #2, 80 by 80, used in problem 13

row 1 = {0.0340 0.0371 0.0403 0.0437 0.0473 0.0511 0.0550 0.0591 0.0634 0.0679 0.0725 0.0772 0.0821 0.0871 0.0921 0.0973 0.1025 0.1078 0.1130 0.1183 0.1235 0.1287 0.1338 0.1388 0.1437 0.1484 0.1529 0.1572 0.1613 0.1651 0.1686 0.1719 0.1747 0.1773 0.1795 0.1813 0.1827 0.1837 0.1843 0.1845 0.1843 0.1837 0.1827 0.1813 0.1795 0.1773 0.1747 0.1719 0.1686 0.1651 0.1613 0.1572 0.1529 0.1484 0.1437 0.1388 0.1338 0.1287 0.1235 0.1183 0.1130 0.1078 0.1025 0.0973 0.0921 0.0871 0.0821 0.0772 0.0725 0.0679 0.0634 0.0591 0.0550 0.0511 0.0473 0.0437 0.0403 0.0371 0.0340 0.0312}

row 2 = {0.0371 0.0404 0.0439 0.0476 0.0515 0.0556 0.0599 0.0644 0.0691 0.0739 0.0790 0.0841 0.0894 0.0948 0.1004 0.1060 0.1117 0.1174 0.1231 0.1289 0.1346 0.1402 0.1458 0.1512 0.1565 0.1617 0.1666 0.1713 0.1757 0.1799 0.1837 0.1872 0.1904 0.1931 0.1955 0.1975 0.1990 0.2001 0.2008 0.2010 0.2008 0.2001 0.1990 0.1975 0.1955 0.1931 0.1904 0.1872 0.1837 0.1799 0.1757 0.1713 0.1666 0.1617 0.1565 0.1512 0.1458 0.1402 0.1346 0.1289 0.1231 0.1174 0.1117 0.1060 0.1004 0.0948 0.0894 0.0841 0.0790 0.0739 0.0691 0.0644 0.0599 0.0556 0.0515 0.0476 0.0439 0.0404 0.0371 0.0340}

row 3 = {0.0403 0.0439 0.0477 0.0518 0.0560 0.0605 0.0651 0.0700 0.0751 0.0804 0.0858 0.0914 0.0972 0.1031 0.1091 0.1152 0.1214 0.1276 0.1338 0.1401 0.1463 0.1524 0.1585 0.1644 0.1701 0.1757 0.1811 0.1862 0.1910 0.1955 0.1997 0.2035 0.2069 0.2099 0.2125 0.2146 0.2163 0.2175 0.2182 0.2185 0.2182 0.2175 0.2163 0.2146 0.2125 0.2099 0.2069 0.2035 0.1997 0.1955 0.1910 0.1862 0.1811 0.1757 0.1701 0.1644 0.1585 0.1524 0.1463 0.1401 0.1338 0.1276 0.1214 0.1152 0.1091 0.1031 0.0972 0.0914 0.0858 0.0804 0.0751 0.0700 0.0651 0.0605 0.0560 0.0518 0.0477 0.0439 0.0403 0.0369}

row 4 = {0.0437 0.0476 0.0518 0.0561 0.0607 0.0656 0.0707 0.0759 0.0814 0.0872 0.0931 0.0992 0.1054 0.1118 0.1183 0.1249 0.1316 0.1384 0.1451 0.1519 0.1586 0.1653 0.1719 0.1783 0.1845 0.1906 0.1964 0.2019 0.2071 0.2120 0.2165 0.2207 0.2244 0.2276 0.2304 0.2328 0.2346 0.2359 0.2367 0.2369 0.2367 0.2359 0.2346 0.2328 0.2304 0.2276 0.2244 0.2207 0.2165 0.2120 0.2071 0.2019 0.1964 0.1906 0.1845 0.1783 0.1719 0.1653 0.1586 0.1519 0.1451 0.1384 0.1316 0.1249 0.1183 0.1118 0.1054 0.0992 0.0931 0.0872 0.0814 0.0759 0.0707 0.0656 0.0607 0.0561 0.0518 0.0476 0.0437 0.0400}

row 5 = {0.0473 0.0515 0.0560 0.0607 0.0657 0.0710 0.0765 0.0822 0.0881 0.0943 0.1007 0.1073 0.1141 0.1210 0.1280 0.1352 0.1424 0.1497 0.1571 0.1644 0.1717 0.1789 0.1860 0.1929 0.1997 0.2062 0.2125 0.2185 0.2241 0.2294 0.2343 0.2388 0.2428 0.2463 0.2494 0.2519 0.2538 0.2552 0.2561 0.2564 0.2561 0.2552 0.2538 0.2519 0.2494 0.2463 0.2428 0.2388 0.2343 0.2294 0.2241 0.2185 0.2125 0.2062 0.1997 0.1929 0.1860 0.1789 0.1717 0.1644 0.1571 0.1497 0.1424 0.1352 0.1280 0.1210 0.1141 0.1073 0.1007 0.0943 0.0881 0.0822 0.0765 0.0710 0.0657 0.0607 0.0560 0.0515 0.0473 0.0433}

row 6 = {0.0511 0.0556 0.0605 0.0656 0.0710 0.0766 0.0825 0.0887 0.0952 0.1018 0.1087 0.1158 0.1231 0.1306 0.1382 0.1460 0.1538 0.1617 0.1696 0.1775 0.1853 0.1931 0.2008 0.2083 0.2156 0.2226 0.2294 0.2359 0.2420 0.2477 0.2530 0.2578 0.2621 0.2660 0.2692 0.2719 0.2741 0.2756 0.2765 0.2768 0.2765 0.2756 0.2741 0.2719 0.2692 0.2660 0.2621 0.2578 0.2530 0.2477 0.2420 0.2359 0.2294 0.2226 0.2156 0.2083 0.2008 0.1931 0.1853 0.1775 0.1696 0.1617 0.1538 0.1460 0.1382 0.1306 0.1231 0.1158 0.1087 0.1018 0.0952 0.0887 0.0825 0.0766 0.0710 0.0656 0.0605 0.0556 0.0511 0.0468}

row 7 = {0.0550 0.0599 0.0651 0.0707 0.0765 0.0825 0.0889 0.0956 0.1025 0.1097 0.1171 0.1248 0.1327 0.1407 0.1489 0.1572 0.1657 0.1742 0.1827 0.1912 0.1997 0.2080 0.2163 0.2244 0.2322 0.2398 0.2471 0.2541 0.2607 0.2668 0.2725 0.2777 0.2824 0.2865 0.2900 0.2929 0.2952 0.2969 0.2979 0.2982 0.2979 0.2969 0.2952 0.2929 0.2900 0.2865 0.2824 0.2777 0.2725 0.2668 0.2607 0.2541 0.2471 0.2398 0.2322 0.2244 0.2163 0.2080 0.1997 0.1912 0.1827 0.1742 0.1657 0.1572 0.1489 0.1407 0.1327 0.1248 0.1171 0.1097 0.1025 0.0956 0.0889 0.0825 0.0765 0.0707 0.0651 0.0599 0.0550 0.0504}

row 8 = {0.0591 0.0644 0.0700 0.0759 0.0822 0.0887 0.0956 0.1027 0.1102 0.1179 0.1259 0.1341 0.1426 0.1512 0.1601 0.1690 0.1781 0.1872 0.1964 0.2055 0.2146 0.2236 0.2325 0.2412 0.2496

0.2578 0.2657 0.2731 0.2802 0.2868 0.2929 0.2985 0.3035 0.3080 0.3117 0.3149 0.3173 0.3191 0.3202 0.3205 0.3202 0.3191 0.3173 0.3149 0.3117 0.3080 0.3035 0.2985 0.2929 0.2868 0.2802 0.2731 0.2657 0.2578 0.2496 0.2412 0.2325 0.2236 0.2146 0.2055 0.1964 0.1872 0.1781 0.1690 0.1601 0.1512 0.1426 0.1341 0.1259 0.1179 0.1102 0.1027 0.0956 0.0887 0.0822 0.0759 0.0700 0.0644 0.0591 0.0542}

row 9 = {0.0634 0.0691 0.0751 0.0814 0.0881 0.0952 0.1025 0.1102 0.1182 0.1265 0.1350 0.1439 0.1529 0.1622 0.1717 0.1813 0.1910 0.2008 0.2106 0.2204 0.2302 0.2398 0.2494 0.2587 0.2677 0.2765 0.2849 0.2929 0.3005 0.3076 0.3142 0.3202 0.3256 0.3303 0.3344 0.3377 0.3404 0.3422 0.3434 0.3438 0.3434 0.3422 0.3404 0.3377 0.3344 0.3303 0.3256 0.3202 0.3142 0.3076 0.3005 0.2929 0.2849 0.2765 0.2677 0.2587 0.2494 0.2398 0.2302 0.2204 0.2106 0.2008 0.1910 0.1813 0.1717 0.1622 0.1529 0.1439 0.1350 0.1265 0.1182 0.1102 0.1025 0.0952 0.0881 0.0814 0.0751 0.0691 0.0634 0.0581}

row 10 = {0.0679 0.0739 0.0804 0.0872 0.0943 0.1018 0.1097 0.1179 0.1265 0.1353 0.1445 0.1540 0.1637 0.1736 0.1837 0.1940 0.2044 0.2149 0.2254 0.2359 0.2463 0.2567 0.2668 0.2768 0.2865 0.2959 0.3049 0.3135 0.3216 0.3292 0.3362 0.3426 0.3484 0.3535 0.3578 0.3614 0.3642 0.3662 0.3675 0.3679 0.3675 0.3662 0.3642 0.3614 0.3578 0.3535 0.3484 0.3426 0.3362 0.3292 0.3216 0.3135 0.3049 0.2959 0.2865 0.2768 0.2668 0.2567 0.2463 0.2359 0.2254 0.2149 0.2044 0.1940 0.1837 0.1736 0.1637 0.1540 0.1445 0.1353 0.1265 0.1179 0.1097 0.1018 0.0943 0.0872 0.0804 0.0739 0.0679 0.0622}

row 11 = {0.0725 0.0790 0.0858 0.0931 0.1007 0.1087 0.1171 0.1259 0.1350 0.1445 0.1543 0.1644 0.1747 0.1853 0.1961 0.2071 0.2182 0.2294 0.2406 0.2519 0.2630 0.2741 0.2849 0.2956 0.3059 0.3159 0.3256 0.3347 0.3434 0.3515 0.3590 0.3658 0.3720 0.3774 0.3820 0.3859 0.3889 0.3911 0.3924 0.3928 0.3924 0.3911 0.3889 0.3859 0.3820 0.3774 0.3720 0.3658 0.3590 0.3515 0.3434 0.3347 0.3256 0.3159 0.3059 0.2956 0.2849 0.2741 0.2630 0.2519 0.2406 0.2294 0.2182 0.2071 0.1961 0.1853 0.1747 0.1644 0.1543 0.1445 0.1350 0.1259 0.1171 0.1087 0.1007 0.0931 0.0858 0.0790 0.0725 0.0664}

row 12 = {0.0772 0.0841 0.0914 0.0992 0.1073 0.1158 0.1248 0.1341 0.1439 0.1540 0.1644 0.1751 0.1862 0.1975 0.2090 0.2207 0.2325 0.2444 0.2564 0.2683 0.2802 0.2920 0.3035 0.3149 0.3259 0.3366 0.3468 0.3566 0.3658 0.3745 0.3825 0.3898 0.3963 0.4021 0.4070 0.4111 0.4143 0.4166 0.4180 0.4185 0.4180 0.4166 0.4143 0.4111 0.4070 0.4021 0.3963 0.3898 0.3825 0.3745 0.3658 0.3566 0.3468 0.3366 0.3259 0.3149 0.3035 0.2920 0.2802 0.2683 0.2564 0.2444 0.2325 0.2207 0.2090 0.1975 0.1862 0.1751 0.1644 0.1540 0.1439 0.1341 0.1248 0.1158 0.1073 0.0992 0.0914 0.0841 0.0772 0.0707}

row 13 = {0.0821 0.0894 0.0972 0.1054 0.1141 0.1231 0.1327 0.1426 0.1529 0.1637 0.1747 0.1862 0.1979 0.2099 0.2221 0.2346 0.2471 0.2598 0.2725 0.2852 0.2979 0.3104 0.3227 0.3347 0.3465 0.3578 0.3687 0.3791 0.3889 0.3981 0.4066 0.4143 0.4213 0.4274 0.4327 0.4370 0.4404 0.4429 0.4444 0.4449 0.4444 0.4429 0.4404 0.4370 0.4327 0.4274 0.4213 0.4143 0.4066 0.3981 0.3889 0.3791 0.3687 0.3578 0.3465 0.3347 0.3227 0.3104 0.2979 0.2852 0.2725 0.2598 0.2471 0.2346 0.2221 0.2099 0.1979 0.1862 0.1747 0.1637 0.1529 0.1426 0.1327 0.1231 0.1141 0.1054 0.0972 0.0894 0.0821 0.0752}

row 14 = {0.0871 0.0948 0.1031 0.1118 0.1210 0.1306 0.1407 0.1512 0.1622 0.1736 0.1853 0.1975 0.2099 0.2226 0.2356 0.2488 0.2621 0.2756 0.2891 0.3025 0.3159 0.3292 0.3422 0.3550 0.3675 0.3795 0.3911 0.4021 0.4125 0.4222 0.4312 0.4395 0.4468 0.4533 0.4589 0.4635 0.4671 0.4697 0.4713 0.4718 0.4713 0.4697 0.4671 0.4635 0.4589 0.4533 0.4468 0.4395 0.4312 0.4222 0.4125 0.4021 0.3911 0.3795 0.3675 0.3550 0.3422 0.3292 0.3159 0.3025 0.2891 0.2756 0.2621 0.2488 0.2356 0.2226 0.2099 0.1975 0.1853 0.1736 0.1622 0.1512 0.1407 0.1306 0.1210 0.1118 0.1031 0.0948 0.0871 0.0797}

row 15 = {0.0921 0.1004 0.1091 0.1183 0.1280 0.1382 0.1489 0.1601 0.1717 0.1837 0.1961 0.2090 0.2221 0.2356 0.2494 0.2633 0.2774 0.2916 0.3059 0.3202 0.3344 0.3484 0.3622 0.3757 0.3889 0.4016 0.4139 0.4255 0.4365 0.4468 0.4564 0.4651 0.4729 0.4798 0.4857 0.4906 0.4944 0.4971 0.4988 0.4994 0.4988 0.4971 0.4944 0.4906 0.4857 0.4798 0.4729 0.4651 0.4564 0.4468 0.4365 0.4255 0.4139 0.4016 0.3889 0.3757 0.3622 0.3484 0.3344 0.3202 0.3059 0.2916 0.2774 0.2633 0.2494 0.2356 0.2221 0.2090 0.1961 0.1837 0.1717 0.1601 0.1489 0.1382 0.1280 0.1183 0.1091 0.1004 0.0921 0.0844}

row 16 = {0.0973 0.1060 0.1152 0.1249 0.1352 0.1460 0.1572 0.1690 0.1813 0.1940 0.2071 0.2207 0.2346 0.2488 0.2633 0.2780 0.2929 0.3080 0.3230 0.3381 0.3531 0.3679 0.3825 0.3968 0.4107 0.4241 0.4370 0.4493 0.4610 0.4718 0.4819 0.4911 0.4994 0.5066 0.5128 0.5180 0.5220 0.5250 0.5267 0.5273 0.5267 0.5250 0.5220 0.5180 0.5128 0.5066 0.4994 0.4911 0.4819 0.4718 0.4610 0.4493 0.4370 0.4241 0.4107 0.3968 0.3825 0.3679 0.3531 0.3381 0.3230 0.3080 0.2929 0.2780 0.2633 0.2488 0.2346 0.2207 0.2071 0.1940 0.1813 0.1690 0.1572 0.1460 0.1352 0.1249 0.1152 0.1060 0.0973 0.0891}

row 17 = {0.1025 0.1117 0.1214 0.1316 0.1424 0.1538 0.1657 0.1781 0.1910 0.2044 0.2182 0.2325 0.2471 0.2621 0.2774 0.2929 0.3086 0.3245 0.3404 0.3562 0.3720 0.3876 0.4030 0.4180 0.4327

0.4468 0.4604 0.4734 0.4857 0.4971 0.5077 0.5174 0.5261 0.5338 0.5403 0.5458 0.5500 0.5531 0.5549 0.5556 0.5549 0.5531 0.5500 0.5458 0.5403 0.5338 0.5261 0.5174 0.5077 0.4971 0.4857 0.4734 0.4604 0.4468 0.4327 0.4180 0.4030 0.3876 0.3720 0.3562 0.3404 0.3245 0.3086 0.2929 0.2774 0.2621 0.2471 0.2325 0.2182 0.2044 0.1910 0.1781 0.1657 0.1538 0.1424 0.1316 0.1214 0.1117 0.1025 0.0939}

row 18 = {0.1078 0.1174 0.1276 0.1384 0.1497 0.1617 0.1742 0.1872 0.2008 0.2149 0.2294 0.2444 0.2598 0.2756 0.2916 0.3080 0.3245 0.3411 0.3578 0.3745 0.3911 0.4075 0.4236 0.4395 0.4549 0.4697 0.4841 0.4977 0.5106 0.5226 0.5338 0.5440 0.5531 0.5611 0.5680 0.5738 0.5782 0.5815 0.5834 0.5840 0.5834 0.5815 0.5782 0.5738 0.5680 0.5611 0.5531 0.5440 0.5338 0.5226 0.5106 0.4977 0.4841 0.4697 0.4549 0.4395 0.4236 0.4075 0.3911 0.3745 0.3578 0.3411 0.3245 0.3080 0.2916 0.2756 0.2598 0.2444 0.2294 0.2149 0.2008 0.1872 0.1742 0.1617 0.1497 0.1384 0.1276 0.1174 0.1078 0.0987}

row 19 = {0.1130 0.1231 0.1338 0.1451 0.1571 0.1696 0.1827 0.1964 0.2106 0.2254 0.2406 0.2564 0.2725 0.2891 0.3059 0.3230 0.3404 0.3578 0.3753 0.3928 0.4102 0.4274 0.4444 0.4610 0.4771 0.4927 0.5077 0.5220 0.5356 0.5482 0.5599 0.5706 0.5802 0.5886 0.5958 0.6018 0.6065 0.6099 0.6119 0.6126 0.6119 0.6099 0.6065 0.6018 0.5958 0.5886 0.5802 0.5706 0.5599 0.5482 0.5356 0.5220 0.5077 0.4927 0.4771 0.4610 0.4444 0.4274 0.4102 0.3928 0.3753 0.3578 0.3404 0.3230 0.3059 0.2891 0.2725 0.2564 0.2406 0.2254 0.2106 0.1964 0.1827 0.1696 0.1571 0.1451 0.1338 0.1231 0.1130 0.1035}

row 20 = {0.1183 0.1289 0.1401 0.1519 0.1644 0.1775 0.1912 0.2055 0.2204 0.2359 0.2519 0.2683 0.2852 0.3025 0.3202 0.3381 0.3562 0.3745 0.3928 0.4111 0.4293 0.4473 0.4651 0.4824 0.4994 0.5157 0.5314 0.5464 0.5605 0.5738 0.5860 0.5972 0.6072 0.6160 0.6236 0.6299 0.6348 0.6383 0.6405 0.6412 0.6405 0.6383 0.6348 0.6299 0.6236 0.6160 0.6072 0.5972 0.5860 0.5738 0.5605 0.5464 0.5314 0.5157 0.4994 0.4824 0.4651 0.4473 0.4293 0.4111 0.3928 0.3745 0.3562 0.3381 0.3202 0.3025 0.2852 0.2683 0.2519 0.2359 0.2204 0.2055 0.1912 0.1775 0.1644 0.1519 0.1401 0.1289 0.1183 0.1084}

row 21 = {0.1235 0.1346 0.1463 0.1586 0.1717 0.1853 0.1997 0.2146 0.2302 0.2463 0.2630 0.2802 0.2979 0.3159 0.3344 0.3531 0.3720 0.3911 0.4102 0.4293 0.4483 0.4671 0.4857 0.5038 0.5215 0.5385 0.5549 0.5706 0.5853 0.5992 0.6119 0.6236 0.6341 0.6433 0.6512 0.6578 0.6629 0.6666 0.6688 0.6696 0.6688 0.6666 0.6629 0.6578 0.6512 0.6433 0.6341 0.6236 0.6119 0.5992 0.5853 0.5706 0.5549 0.5385 0.5215 0.5038 0.4857 0.4671 0.4483 0.4293 0.4102 0.3911 0.3720 0.3531 0.3344 0.3159 0.2979 0.2802 0.2630 0.2463 0.2302 0.2146 0.1997 0.1853 0.1717 0.1586 0.1463 0.1346 0.1235 0.1132}

row 22 = {0.1287 0.1402 0.1524 0.1653 0.1789 0.1931 0.2080 0.2236 0.2398 0.2567 0.2741 0.2920 0.3104 0.3292 0.3484 0.3679 0.3876 0.4075 0.4274 0.4473 0.4671 0.4868 0.5061 0.5250 0.5434 0.5611 0.5782 0.5945 0.6099 0.6243 0.6376 0.6498 0.6607 0.6703 0.6786 0.6854 0.6907 0.6946 0.6969 0.6977 0.6969 0.6946 0.6907 0.6854 0.6786 0.6703 0.6607 0.6498 0.6376 0.6243 0.6099 0.5945 0.5782 0.5611 0.5434 0.5250 0.5061 0.4868 0.4671 0.4473 0.4274 0.4075 0.3876 0.3679 0.3484 0.3292 0.3104 0.2920 0.2741 0.2567 0.2398 0.2236 0.2080 0.1931 0.1789 0.1653 0.1524 0.1402 0.1287 0.1179}

row 23 = {0.1338 0.1458 0.1585 0.1719 0.1860 0.2008 0.2163 0.2325 0.2494 0.2668 0.2849 0.3035 0.3227 0.3422 0.3622 0.3825 0.4030 0.4236 0.4444 0.4651 0.4857 0.5061 0.5261 0.5458 0.5649 0.5834 0.6012 0.6181 0.6341 0.6491 0.6629 0.6756 0.6869 0.6969 0.7055 0.7126 0.7181 0.7221 0.7245 0.7253 0.7245 0.7221 0.7181 0.7126 0.7055 0.6969 0.6869 0.6756 0.6629 0.6491 0.6341 0.6181 0.6012 0.5834 0.5649 0.5458 0.5261 0.5061 0.4857 0.4651 0.4444 0.4236 0.4030 0.3825 0.3622 0.3422 0.3227 0.3035 0.2849 0.2668 0.2494 0.2325 0.2163 0.2008 0.1860 0.1719 0.1585 0.1458 0.1338 0.1226}

row 24 = {0.1388 0.1512 0.1644 0.1783 0.1929 0.2083 0.2244 0.2412 0.2587 0.2768 0.2956 0.3149 0.3347 0.3550 0.3757 0.3968 0.4180 0.4395 0.4610 0.4824 0.5038 0.5250 0.5458 0.5662 0.5860 0.6052 0.6236 0.6412 0.6578 0.6733 0.6877 0.7008 0.7126 0.7229 0.7318 0.7392 0.7449 0.7491 0.7516 0.7524 0.7516 0.7491 0.7449 0.7392 0.7318 0.7229 0.7126 0.7008 0.6877 0.6733 0.6578 0.6412 0.6236 0.6052 0.5860 0.5662 0.5458 0.5250 0.5038 0.4824 0.4610 0.4395 0.4180 0.3968 0.3757 0.3550 0.3347 0.3149 0.2956 0.2768 0.2587 0.2412 0.2244 0.2083 0.1929 0.1783 0.1644 0.1512 0.1388 0.1272}

row 25 = {0.1437 0.1565 0.1701 0.1845 0.1997 0.2156 0.2322 0.2496 0.2677 0.2865 0.3059 0.3259 0.3465 0.3675 0.3889 0.4107 0.4327 0.4549 0.4771 0.4994 0.5215 0.5434 0.5649 0.5860 0.6065 0.6264 0.6455 0.6637 0.6808 0.6969 0.7118 0.7253 0.7375 0.7483 0.7575 0.7651 0.7711 0.7753 0.7779 0.7788 0.7779 0.7753 0.7711 0.7651 0.7575 0.7483 0.7375 0.7253 0.7118 0.6969 0.6808 0.6637 0.6455 0.6264 0.6065 0.5860 0.5649 0.5434 0.5215 0.4994 0.4771 0.4549 0.4327 0.4107 0.3889 0.3675 0.3465 0.3259 0.3059 0.2865 0.2677 0.2496 0.2322 0.2156 0.1997 0.1845 0.1701 0.1565 0.1437 0.1316}

row 26 = {0.1484 0.1617 0.1757 0.1906 0.2062 0.2226 0.2398 0.2578 0.2765 0.2959 0.3159 0.3366 0.3578 0.3795 0.4016 0.4241 0.4468 0.4697 0.4927 0.5157 0.5385 0.5611 0.5834 0.6052 0.6264

0.6469 0.6666 0.6854 0.7031 0.7197 0.7351 0.7491 0.7617 0.7728 0.7823 0.7901 0.7963 0.8007 0.8034
0.8043 0.8034 0.8007 0.7963 0.7901 0.7823 0.7728 0.7617 0.7491 0.7351 0.7197 0.7031 0.6854 0.6666
0.6469 0.6264 0.6052 0.5834 0.5611 0.5385 0.5157 0.4927 0.4697 0.4468 0.4241 0.4016 0.3795 0.3578
0.3366 0.3159 0.2959 0.2765 0.2578 0.2398 0.2226 0.2062 0.1906 0.1757 0.1617 0.1484 0.1359}

row 27 = {0.1529 0.1666 0.1811 0.1964 0.2125 0.2294 0.2471 0.2657 0.2849 0.3049 0.3256
0.3468 0.3687 0.3911 0.4139 0.4370 0.4604 0.4841 0.5077 0.5314 0.5549 0.5782 0.6012 0.6236 0.6455
0.6666 0.6869 0.7063 0.7245 0.7416 0.7575 0.7719 0.7849 0.7963 0.8061 0.8142 0.8206 0.8251 0.8279
0.8288 0.8279 0.8251 0.8206 0.8142 0.8061 0.7963 0.7849 0.7719 0.7575 0.7416 0.7245 0.7063 0.6869
0.6666 0.6455 0.6236 0.6012 0.5782 0.5549 0.5314 0.5077 0.4841 0.4604 0.4370 0.4139 0.3911 0.3687
0.3468 0.3256 0.3049 0.2849 0.2657 0.2471 0.2294 0.2125 0.1964 0.1811 0.1666 0.1529 0.1401}

row 28 = {0.1572 0.1713 0.1862 0.2019 0.2185 0.2359 0.2541 0.2731 0.2929 0.3135 0.3347
0.3566 0.3791 0.4021 0.4255 0.4493 0.4734 0.4977 0.5220 0.5464 0.5706 0.5945 0.6181 0.6412 0.6637
0.6854 0.7063 0.7261 0.7449 0.7625 0.7788 0.7937 0.8070 0.8187 0.8288 0.8371 0.8437 0.8484 0.8512
0.8521 0.8512 0.8484 0.8437 0.8371 0.8288 0.8187 0.8070 0.7937 0.7788 0.7625 0.7449 0.7261 0.7063
0.6854 0.6637 0.6412 0.6181 0.5945 0.5706 0.5464 0.5220 0.4977 0.4734 0.4493 0.4255 0.4021 0.3791
0.3566 0.3347 0.3135 0.2929 0.2731 0.2541 0.2359 0.2185 0.2019 0.1862 0.1713 0.1572 0.1440}

row 29 = {0.1613 0.1757 0.1910 0.2071 0.2241 0.2420 0.2607 0.2802 0.3005 0.3216 0.3434
0.3658 0.3889 0.4125 0.4365 0.4610 0.4857 0.5106 0.5356 0.5605 0.5853 0.6099 0.6341 0.6578 0.6808
0.7031 0.7245 0.7449 0.7642 0.7823 0.7990 0.8142 0.8279 0.8399 0.8503 0.8588 0.8655 0.8703 0.8732
0.8742 0.8732 0.8703 0.8655 0.8588 0.8503 0.8399 0.8279 0.8142 0.7990 0.7823 0.7642 0.7449 0.7245
0.7031 0.6808 0.6578 0.6341 0.6099 0.5853 0.5605 0.5356 0.5106 0.4857 0.4610 0.4365 0.4125 0.3889
0.3658 0.3434 0.3216 0.3005 0.2802 0.2607 0.2420 0.2241 0.2071 0.1910 0.1757 0.1613 0.1478}

row 30 = {0.1651 0.1799 0.1955 0.2120 0.2294 0.2477 0.2668 0.2868 0.3076 0.3292 0.3515
0.3745 0.3981 0.4222 0.4468 0.4718 0.4971 0.5226 0.5482 0.5738 0.5992 0.6243 0.6491 0.6733 0.6969
0.7197 0.7416 0.7625 0.7823 0.8007 0.8178 0.8334 0.8474 0.8598 0.8703 0.8791 0.8859 0.8909 0.8938
0.8948 0.8938 0.8909 0.8859 0.8791 0.8703 0.8598 0.8474 0.8334 0.8178 0.8007 0.7823 0.7625 0.7416
0.7197 0.6969 0.6733 0.6491 0.6243 0.5992 0.5738 0.5482 0.5226 0.4971 0.4718 0.4468 0.4222 0.3981
0.3745 0.3515 0.3292 0.3076 0.2868 0.2668 0.2477 0.2294 0.2120 0.1955 0.1799 0.1651 0.1512}

row 31 = {0.1686 0.1837 0.1997 0.2165 0.2343 0.2530 0.2725 0.2929 0.3142 0.3362 0.3590
0.3825 0.4066 0.4312 0.4564 0.4819 0.5077 0.5338 0.5599 0.5860 0.6119 0.6376 0.6629 0.6877 0.7118
0.7351 0.7575 0.7788 0.7990 0.8178 0.8353 0.8512 0.8655 0.8781 0.8889 0.8978 0.9048 0.9099 0.9129
0.9139 0.9129 0.9099 0.9048 0.8978 0.8889 0.8781 0.8655 0.8512 0.8353 0.8178 0.7990 0.7788 0.7575
0.7351 0.7118 0.6877 0.6629 0.6376 0.6119 0.5860 0.5599 0.5338 0.5077 0.4819 0.4564 0.4312 0.4066
0.3825 0.3590 0.3362 0.3142 0.2929 0.2725 0.2530 0.2343 0.2165 0.1997 0.1837 0.1686 0.1545}

row 32 = {0.1719 0.1872 0.2035 0.2207 0.2388 0.2578 0.2777 0.2985 0.3202 0.3426 0.3658
0.3898 0.4143 0.4395 0.4651 0.4911 0.5174 0.5440 0.5706 0.5972 0.6236 0.6498 0.6756 0.7008 0.7253
0.7491 0.7719 0.7937 0.8142 0.8334 0.8512 0.8674 0.8820 0.8948 0.9058 0.9149 0.9221 0.9272 0.9303
0.9314 0.9303 0.9272 0.9221 0.9149 0.9058 0.8948 0.8820 0.8674 0.8512 0.8334 0.8142 0.7937 0.7719
0.7491 0.7253 0.7008 0.6756 0.6498 0.6236 0.5972 0.5706 0.5440 0.5174 0.4911 0.4651 0.4395 0.4143
0.3898 0.3658 0.3426 0.3202 0.2985 0.2777 0.2578 0.2388 0.2207 0.2035 0.1872 0.1719 0.1574}

row 33 = {0.1747 0.1904 0.2069 0.2244 0.2428 0.2621 0.2824 0.3035 0.3256 0.3484 0.3720
0.3963 0.4213 0.4468 0.4729 0.4994 0.5261 0.5531 0.5802 0.6072 0.6341 0.6607 0.6869 0.7126 0.7375
0.7617 0.7849 0.8070 0.8279 0.8474 0.8655 0.8820 0.8968 0.9099 0.9211 0.9303 0.9376 0.9428 0.9460
0.9470 0.9460 0.9428 0.9376 0.9303 0.9211 0.9099 0.8968 0.8820 0.8655 0.8474 0.8279 0.8070 0.7849
0.7617 0.7375 0.7126 0.6869 0.6607 0.6341 0.6072 0.5802 0.5531 0.5261 0.4994 0.4729 0.4468 0.4213
0.3963 0.3720 0.3484 0.3256 0.3035 0.2824 0.2621 0.2428 0.2244 0.2069 0.1904 0.1747 0.1601}

row 34 = {0.1773 0.1931 0.2099 0.2276 0.2463 0.2660 0.2865 0.3080 0.3303 0.3535 0.3774
0.4021 0.4274 0.4533 0.4798 0.5066 0.5338 0.5611 0.5886 0.6160 0.6433 0.6703 0.6969 0.7229 0.7483
0.7728 0.7963 0.8187 0.8399 0.8598 0.8781 0.8948 0.9099 0.9231 0.9345 0.9439 0.9512 0.9565 0.9597
0.9608 0.9597 0.9565 0.9512 0.9439 0.9345 0.9231 0.9099 0.8948 0.8781 0.8598 0.8399 0.8187 0.7963
0.7728 0.7483 0.7229 0.6969 0.6703 0.6433 0.6160 0.5886 0.5611 0.5338 0.5066 0.4798 0.4533 0.4274
0.4021 0.3774 0.3535 0.3303 0.3080 0.2865 0.2660 0.2463 0.2276 0.2099 0.1931 0.1773 0.1624}

row 35 = {0.1795 0.1955 0.2125 0.2304 0.2494 0.2692 0.2900 0.3117 0.3344 0.3578 0.3820
0.4070 0.4327 0.4589 0.4857 0.5128 0.5403 0.5680 0.5958 0.6236 0.6512 0.6786 0.7055 0.7318 0.7575

0.7823 0.8061 0.8288 0.8503 0.8703 0.8889 0.9058 0.9211 0.9345 0.9460 0.9555 0.9629 0.9683 0.9715
0.9726 0.9715 0.9683 0.9629 0.9555 0.9460 0.9345 0.9211 0.9058 0.8889 0.8703 0.8503 0.8288 0.8061
0.7823 0.7575 0.7318 0.7055 0.6786 0.6512 0.6236 0.5958 0.5680 0.5403 0.5128 0.4857 0.4589 0.4327
0.4070 0.3820 0.3578 0.3344 0.3117 0.2900 0.2692 0.2494 0.2304 0.2125 0.1955 0.1795 0.1644}

row 36 = {0.1813 0.1975 0.2146 0.2328 0.2519 0.2719 0.2929 0.3149 0.3377 0.3614 0.3859
0.4111 0.4370 0.4635 0.4906 0.5180 0.5458 0.5738 0.6018 0.6299 0.6578 0.6854 0.7126 0.7392 0.7651
0.7901 0.8142 0.8371 0.8588 0.8791 0.8978 0.9149 0.9303 0.9439 0.9555 0.9651 0.9726 0.9780 0.9813
0.9824 0.9813 0.9780 0.9726 0.9651 0.9555 0.9439 0.9303 0.9149 0.8978 0.8791 0.8588 0.8371 0.8142
0.7901 0.7651 0.7392 0.7126 0.6854 0.6578 0.6299 0.6018 0.5738 0.5458 0.5180 0.4906 0.4635 0.4370
0.4111 0.3859 0.3614 0.3377 0.3149 0.2929 0.2719 0.2519 0.2328 0.2146 0.1975 0.1813 0.1660}

row 37 = {0.1827 0.1990 0.2163 0.2346 0.2538 0.2741 0.2952 0.3173 0.3404 0.3642 0.3889
0.4143 0.4404 0.4671 0.4944 0.5220 0.5500 0.5782 0.6065 0.6348 0.6629 0.6907 0.7181 0.7449 0.7711
0.7963 0.8206 0.8437 0.8655 0.8859 0.9048 0.9221 0.9376 0.9512 0.9629 0.9726 0.9802 0.9857 0.9890
0.9900 0.9890 0.9857 0.9802 0.9726 0.9629 0.9512 0.9376 0.9221 0.9048 0.8859 0.8655 0.8437 0.8206
0.7963 0.7711 0.7449 0.7181 0.6907 0.6629 0.6348 0.6065 0.5782 0.5500 0.5220 0.4944 0.4671 0.4404
0.4143 0.3889 0.3642 0.3404 0.3173 0.2952 0.2741 0.2538 0.2346 0.2163 0.1990 0.1827 0.1673}

row 38 = {0.1837 0.2001 0.2175 0.2359 0.2552 0.2756 0.2969 0.3191 0.3422 0.3662 0.3911
0.4166 0.4429 0.4697 0.4971 0.5250 0.5531 0.5815 0.6099 0.6383 0.6666 0.6946 0.7221 0.7491 0.7753
0.8007 0.8251 0.8484 0.8703 0.8909 0.9099 0.9272 0.9428 0.9565 0.9683 0.9780 0.9857 0.9912 0.9945
0.9956 0.9945 0.9912 0.9857 0.9780 0.9683 0.9565 0.9428 0.9272 0.9099 0.8909 0.8703 0.8484 0.8251
0.8007 0.7753 0.7491 0.7221 0.6946 0.6666 0.6383 0.6099 0.5815 0.5531 0.5250 0.4971 0.4697 0.4429
0.4166 0.3911 0.3662 0.3422 0.3191 0.2969 0.2756 0.2552 0.2359 0.2175 0.2001 0.1837 0.1683}

row 39 = {0.1843 0.2008 0.2182 0.2367 0.2561 0.2765 0.2979 0.3202 0.3434 0.3675 0.3924
0.4180 0.4444 0.4713 0.4988 0.5267 0.5549 0.5834 0.6119 0.6405 0.6688 0.6969 0.7245 0.7516 0.7779
0.8034 0.8279 0.8512 0.8732 0.8938 0.9129 0.9303 0.9460 0.9597 0.9715 0.9813 0.9890 0.9945 0.9978
0.9989 0.9978 0.9945 0.9890 0.9813 0.9715 0.9597 0.9460 0.9303 0.9129 0.8938 0.8732 0.8512 0.8279
0.8034 0.7779 0.7516 0.7245 0.6969 0.6688 0.6405 0.6119 0.5834 0.5549 0.5267 0.4988 0.4713 0.4444
0.4180 0.3924 0.3675 0.3434 0.3202 0.2979 0.2765 0.2561 0.2367 0.2182 0.2008 0.1843 0.1688}

row 40 = {0.1845 0.2010 0.2185 0.2369 0.2564 0.2768 0.2982 0.3205 0.3438 0.3679 0.3928
0.4185 0.4449 0.4718 0.4994 0.5273 0.5556 0.5840 0.6126 0.6412 0.6696 0.6977 0.7253 0.7524 0.7788
0.8043 0.8288 0.8521 0.8742 0.8948 0.9139 0.9314 0.9470 0.9608 0.9726 0.9824 0.9900 0.9956 0.9989
1.0000 0.9989 0.9956 0.9900 0.9824 0.9726 0.9608 0.9470 0.9314 0.9139 0.8948 0.8742 0.8521 0.8288
0.8043 0.7788 0.7524 0.7253 0.6977 0.6696 0.6412 0.6126 0.5840 0.5556 0.5273 0.4994 0.4718 0.4449
0.4185 0.3928 0.3679 0.3438 0.3205 0.2982 0.2768 0.2564 0.2369 0.2185 0.2010 0.1845 0.1690}

row 41 = {0.1843 0.2008 0.2182 0.2367 0.2561 0.2765 0.2979 0.3202 0.3434 0.3675 0.3924
0.4180 0.4444 0.4713 0.4988 0.5267 0.5549 0.5834 0.6119 0.6405 0.6688 0.6969 0.7245 0.7516 0.7779
0.8034 0.8279 0.8512 0.8732 0.8938 0.9129 0.9303 0.9460 0.9597 0.9715 0.9813 0.9890 0.9945 0.9978
0.9989 0.9978 0.9945 0.9890 0.9813 0.9715 0.9597 0.9460 0.9303 0.9129 0.8938 0.8732 0.8512 0.8279
0.8034 0.7779 0.7516 0.7245 0.6969 0.6688 0.6405 0.6119 0.5834 0.5549 0.5267 0.4988 0.4713 0.4444
0.4180 0.3924 0.3675 0.3434 0.3202 0.2979 0.2765 0.2561 0.2367 0.2182 0.2008 0.1843 0.1688}

row 42 = {0.1837 0.2001 0.2175 0.2359 0.2552 0.2756 0.2969 0.3191 0.3422 0.3662 0.3911
0.4166 0.4429 0.4697 0.4971 0.5250 0.5531 0.5815 0.6099 0.6383 0.6666 0.6946 0.7221 0.7491 0.7753
0.8007 0.8251 0.8484 0.8703 0.8909 0.9099 0.9272 0.9428 0.9565 0.9683 0.9780 0.9857 0.9912 0.9945
0.9956 0.9945 0.9912 0.9857 0.9780 0.9683 0.9565 0.9428 0.9272 0.9099 0.8909 0.8703 0.8484 0.8251
0.8007 0.7753 0.7491 0.7221 0.6946 0.6666 0.6383 0.6099 0.5815 0.5531 0.5250 0.4971 0.4697 0.4429
0.4166 0.3911 0.3662 0.3422 0.3191 0.2969 0.2756 0.2552 0.2359 0.2175 0.2001 0.1837 0.1683}

row 43 = {0.1827 0.1990 0.2163 0.2346 0.2538 0.2741 0.2952 0.3173 0.3404 0.3642 0.3889
0.4143 0.4404 0.4671 0.4944 0.5220 0.5500 0.5782 0.6065 0.6348 0.6629 0.6907 0.7181 0.7449 0.7711
0.7963 0.8206 0.8437 0.8655 0.8859 0.9048 0.9221 0.9376 0.9512 0.9629 0.9726 0.9802 0.9857 0.9890
0.9900 0.9890 0.9857 0.9802 0.9726 0.9629 0.9512 0.9376 0.9221 0.9048 0.8859 0.8655 0.8437 0.8206
0.7963 0.7711 0.7449 0.7181 0.6907 0.6629 0.6348 0.6065 0.5782 0.5500 0.5220 0.4944 0.4671 0.4404
0.4143 0.3889 0.3642 0.3404 0.3173 0.2952 0.2741 0.2538 0.2346 0.2163 0.1990 0.1827 0.1673}

row 44 = {0.1813 0.1975 0.2146 0.2328 0.2519 0.2719 0.2929 0.3149 0.3377 0.3614 0.3859
0.4111 0.4370 0.4635 0.4906 0.5180 0.5458 0.5738 0.6018 0.6299 0.6578 0.6854 0.7126 0.7392 0.7651

0.7901 0.8142 0.8371 0.8588 0.8791 0.8978 0.9149 0.9303 0.9439 0.9555 0.9651 0.9726 0.9780 0.9813
0.9824 0.9813 0.9780 0.9726 0.9651 0.9555 0.9439 0.9303 0.9149 0.8978 0.8791 0.8588 0.8371 0.8142
0.7901 0.7651 0.7392 0.7126 0.6854 0.6578 0.6299 0.6018 0.5738 0.5458 0.5180 0.4906 0.4635 0.4370
0.4111 0.3859 0.3614 0.3377 0.3149 0.2929 0.2719 0.2519 0.2328 0.2146 0.1975 0.1813 0.1660}

row 45 = {0.1795 0.1955 0.2125 0.2304 0.2494 0.2692 0.2900 0.3117 0.3344 0.3578 0.3820
0.4070 0.4327 0.4589 0.4857 0.5128 0.5403 0.5680 0.5958 0.6236 0.6512 0.6786 0.7055 0.7318 0.7575
0.7823 0.8061 0.8288 0.8503 0.8703 0.8889 0.9058 0.9211 0.9345 0.9460 0.9555 0.9629 0.9683 0.9715
0.9726 0.9715 0.9683 0.9629 0.9555 0.9460 0.9345 0.9211 0.9058 0.8889 0.8703 0.8503 0.8288 0.8061
0.7823 0.7575 0.7318 0.7055 0.6786 0.6512 0.6236 0.5958 0.5680 0.5403 0.5128 0.4857 0.4589 0.4327
0.4070 0.3820 0.3578 0.3344 0.3117 0.2900 0.2692 0.2494 0.2304 0.2125 0.1955 0.1795 0.1644}

row 46 = {0.1773 0.1931 0.2099 0.2276 0.2463 0.2660 0.2865 0.3080 0.3303 0.3535 0.3774
0.4021 0.4274 0.4533 0.4798 0.5066 0.5338 0.5611 0.5886 0.6160 0.6433 0.6703 0.6969 0.7229 0.7483
0.7728 0.7963 0.8187 0.8399 0.8598 0.8781 0.8948 0.9099 0.9231 0.9345 0.9439 0.9512 0.9565 0.9597
0.9608 0.9597 0.9565 0.9512 0.9439 0.9345 0.9231 0.9099 0.8948 0.8781 0.8598 0.8399 0.8187 0.7963
0.7728 0.7483 0.7229 0.6969 0.6703 0.6433 0.6160 0.5886 0.5611 0.5338 0.5066 0.4798 0.4533 0.4274
0.4021 0.3774 0.3535 0.3303 0.3080 0.2865 0.2660 0.2463 0.2276 0.2099 0.1931 0.1773 0.1624}

row 47 = {0.1747 0.1904 0.2069 0.2244 0.2428 0.2621 0.2824 0.3035 0.3256 0.3484 0.3720
0.3963 0.4213 0.4468 0.4729 0.4994 0.5261 0.5531 0.5802 0.6072 0.6341 0.6607 0.6869 0.7126 0.7375
0.7617 0.7849 0.8070 0.8279 0.8474 0.8655 0.8820 0.8968 0.9099 0.9211 0.9303 0.9376 0.9428 0.9460
0.9470 0.9460 0.9428 0.9376 0.9303 0.9211 0.9099 0.8968 0.8820 0.8655 0.8474 0.8279 0.8070 0.7849
0.7617 0.7375 0.7126 0.6869 0.6607 0.6341 0.6072 0.5802 0.5531 0.5261 0.4994 0.4729 0.4468 0.4213
0.3963 0.3720 0.3484 0.3256 0.3035 0.2824 0.2621 0.2428 0.2244 0.2069 0.1904 0.1747 0.1601}

row 48 = {0.1719 0.1872 0.2035 0.2207 0.2388 0.2578 0.2777 0.2985 0.3202 0.3426 0.3658
0.3898 0.4143 0.4395 0.4651 0.4911 0.5174 0.5440 0.5706 0.5972 0.6236 0.6498 0.6756 0.7008 0.7253
0.7491 0.7719 0.7937 0.8142 0.8334 0.8512 0.8674 0.8820 0.8948 0.9058 0.9149 0.9221 0.9272 0.9303
0.9314 0.9303 0.9272 0.9221 0.9149 0.9058 0.8948 0.8820 0.8674 0.8512 0.8334 0.8142 0.7937 0.7719
0.7491 0.7253 0.7008 0.6756 0.6498 0.6236 0.5972 0.5706 0.5440 0.5174 0.4911 0.4651 0.4395 0.4143
0.3898 0.3658 0.3426 0.3202 0.2985 0.2777 0.2578 0.2388 0.2207 0.2035 0.1872 0.1719 0.1574}

row 49 = {0.1686 0.1837 0.1997 0.2165 0.2343 0.2530 0.2725 0.2929 0.3142 0.3362 0.3590
0.3825 0.4066 0.4312 0.4564 0.4819 0.5077 0.5338 0.5599 0.5860 0.6119 0.6376 0.6629 0.6877 0.7118
0.7351 0.7575 0.7788 0.7990 0.8178 0.8353 0.8512 0.8655 0.8781 0.8889 0.8978 0.9048 0.9099 0.9129
0.9139 0.9129 0.9099 0.9048 0.8978 0.8889 0.8781 0.8655 0.8512 0.8353 0.8178 0.7990 0.7788 0.7575
0.7351 0.7118 0.6877 0.6629 0.6376 0.6119 0.5860 0.5599 0.5338 0.5077 0.4819 0.4564 0.4312 0.4066
0.3825 0.3590 0.3362 0.3142 0.2929 0.2725 0.2530 0.2343 0.2165 0.1997 0.1837 0.1686 0.1545}

row 50 = {0.1651 0.1799 0.1955 0.2120 0.2294 0.2477 0.2668 0.2868 0.3076 0.3292 0.3515
0.3745 0.3981 0.4222 0.4468 0.4718 0.4971 0.5226 0.5482 0.5738 0.5992 0.6243 0.6491 0.6733 0.6969
0.7197 0.7416 0.7625 0.7823 0.8007 0.8178 0.8334 0.8474 0.8598 0.8703 0.8791 0.8859 0.8909 0.8938
0.8948 0.8938 0.8909 0.8859 0.8791 0.8703 0.8598 0.8474 0.8334 0.8178 0.8007 0.7823 0.7625 0.7416
0.7197 0.6969 0.6733 0.6491 0.6243 0.5992 0.5738 0.5482 0.5226 0.4971 0.4718 0.4468 0.4222 0.3981
0.3745 0.3515 0.3292 0.3076 0.2868 0.2668 0.2477 0.2294 0.2120 0.1955 0.1799 0.1651 0.1512}

row 51 = {0.1613 0.1757 0.1910 0.2071 0.2241 0.2420 0.2607 0.2802 0.3005 0.3216 0.3434
0.3658 0.3889 0.4125 0.4365 0.4610 0.4857 0.5106 0.5356 0.5605 0.5853 0.6099 0.6341 0.6578 0.6808
0.7031 0.7245 0.7449 0.7642 0.7823 0.7990 0.8142 0.8279 0.8399 0.8503 0.8588 0.8655 0.8703 0.8732
0.8742 0.8732 0.8703 0.8655 0.8588 0.8503 0.8399 0.8279 0.8142 0.7990 0.7823 0.7642 0.7449 0.7245
0.7031 0.6808 0.6578 0.6341 0.6099 0.5853 0.5605 0.5356 0.5106 0.4857 0.4610 0.4365 0.4125 0.3889
0.3658 0.3434 0.3216 0.3005 0.2802 0.2607 0.2420 0.2241 0.2071 0.1910 0.1757 0.1613 0.1478}

row 52 = {0.1572 0.1713 0.1862 0.2019 0.2185 0.2359 0.2541 0.2731 0.2929 0.3135 0.3347
0.3566 0.3791 0.4021 0.4255 0.4493 0.4734 0.4977 0.5220 0.5464 0.5706 0.5945 0.6181 0.6412 0.6637
0.6854 0.7063 0.7261 0.7449 0.7625 0.7788 0.7937 0.8070 0.8187 0.8288 0.8371 0.8437 0.8484 0.8512
0.8521 0.8512 0.8484 0.8437 0.8371 0.8288 0.8187 0.8070 0.7937 0.7788 0.7625 0.7449 0.7261 0.7063
0.6854 0.6637 0.6412 0.6181 0.5945 0.5706 0.5464 0.5220 0.4977 0.4734 0.4493 0.4255 0.4021 0.3791
0.3566 0.3347 0.3135 0.2929 0.2731 0.2541 0.2359 0.2185 0.2019 0.1862 0.1713 0.1572 0.1440}

row 53 = {0.1529 0.1666 0.1811 0.1964 0.2125 0.2294 0.2471 0.2657 0.2849 0.3049 0.3256
0.3468 0.3687 0.3911 0.4139 0.4370 0.4604 0.4841 0.5077 0.5314 0.5549 0.5782 0.6012 0.6236 0.6455

0.6666 0.6869 0.7063 0.7245 0.7416 0.7575 0.7719 0.7849 0.7963 0.8061 0.8142 0.8206 0.8251 0.8279
0.8288 0.8279 0.8251 0.8206 0.8142 0.8061 0.7963 0.7849 0.7719 0.7575 0.7416 0.7245 0.7063 0.6869
0.6666 0.6455 0.6236 0.6012 0.5782 0.5549 0.5314 0.5077 0.4841 0.4604 0.4370 0.4139 0.3911 0.3687
0.3468 0.3256 0.3049 0.2849 0.2657 0.2471 0.2294 0.2125 0.1964 0.1811 0.1666 0.1529 0.1401}

row 54 = {0.1484 0.1617 0.1757 0.1906 0.2062 0.2226 0.2398 0.2578 0.2765 0.2959 0.3159
0.3366 0.3578 0.3795 0.4016 0.4241 0.4468 0.4697 0.4927 0.5157 0.5385 0.5611 0.5834 0.6052 0.6264
0.6469 0.6666 0.6854 0.7031 0.7197 0.7351 0.7491 0.7617 0.7728 0.7823 0.7901 0.7963 0.8007 0.8034
0.8043 0.8034 0.8007 0.7963 0.7901 0.7823 0.7728 0.7617 0.7491 0.7351 0.7197 0.7031 0.6854 0.6666
0.6469 0.6264 0.6052 0.5834 0.5611 0.5385 0.5157 0.4927 0.4697 0.4468 0.4241 0.4016 0.3795 0.3578
0.3366 0.3159 0.2959 0.2765 0.2578 0.2398 0.2226 0.2062 0.1906 0.1757 0.1617 0.1484 0.1359}

row 55 = {0.1437 0.1565 0.1701 0.1845 0.1997 0.2156 0.2322 0.2496 0.2677 0.2865 0.3059
0.3259 0.3465 0.3675 0.3889 0.4107 0.4327 0.4549 0.4771 0.4994 0.5215 0.5434 0.5649 0.5860 0.6065
0.6264 0.6455 0.6637 0.6808 0.6969 0.7118 0.7253 0.7375 0.7483 0.7575 0.7651 0.7711 0.7753 0.7779
0.7788 0.7779 0.7753 0.7711 0.7651 0.7575 0.7483 0.7375 0.7253 0.7118 0.6969 0.6808 0.6637 0.6455
0.6264 0.6065 0.5860 0.5649 0.5434 0.5215 0.4994 0.4771 0.4549 0.4327 0.4107 0.3889 0.3675 0.3465
0.3259 0.3059 0.2865 0.2677 0.2496 0.2322 0.2156 0.1997 0.1845 0.1701 0.1565 0.1437 0.1316}

row 56 = {0.1388 0.1512 0.1644 0.1783 0.1929 0.2083 0.2244 0.2412 0.2587 0.2768 0.2956
0.3149 0.3347 0.3550 0.3757 0.3968 0.4180 0.4395 0.4610 0.4824 0.5038 0.5250 0.5458 0.5662 0.5860
0.6052 0.6236 0.6412 0.6578 0.6733 0.6877 0.7008 0.7126 0.7229 0.7318 0.7392 0.7449 0.7491 0.7516
0.7524 0.7516 0.7491 0.7449 0.7392 0.7318 0.7229 0.7126 0.7008 0.6877 0.6733 0.6578 0.6412 0.6236
0.6052 0.5860 0.5662 0.5458 0.5250 0.5038 0.4824 0.4610 0.4395 0.4180 0.3968 0.3757 0.3550 0.3347
0.3149 0.2956 0.2768 0.2587 0.2412 0.2244 0.2083 0.1929 0.1783 0.1644 0.1512 0.1388 0.1272}

row 57 = {0.1338 0.1458 0.1585 0.1719 0.1860 0.2008 0.2163 0.2325 0.2494 0.2668 0.2849
0.3035 0.3227 0.3422 0.3622 0.3825 0.4030 0.4236 0.4444 0.4651 0.4857 0.5061 0.5261 0.5458 0.5649
0.5834 0.6012 0.6181 0.6341 0.6491 0.6629 0.6756 0.6869 0.6969 0.7055 0.7126 0.7181 0.7221 0.7245
0.7253 0.7245 0.7221 0.7181 0.7126 0.7055 0.6969 0.6869 0.6756 0.6629 0.6491 0.6341 0.6181 0.6012
0.5834 0.5649 0.5458 0.5261 0.5061 0.4857 0.4651 0.4444 0.4236 0.4030 0.3825 0.3622 0.3422 0.3227
0.3035 0.2849 0.2668 0.2494 0.2325 0.2163 0.2008 0.1860 0.1719 0.1585 0.1458 0.1338 0.1226}

row 58 = {0.1287 0.1402 0.1524 0.1653 0.1789 0.1931 0.2080 0.2236 0.2398 0.2567 0.2741
0.2920 0.3104 0.3292 0.3484 0.3679 0.3876 0.4075 0.4274 0.4473 0.4671 0.4868 0.5061 0.5250 0.5434
0.5611 0.5782 0.5945 0.6099 0.6243 0.6376 0.6498 0.6607 0.6703 0.6786 0.6854 0.6907 0.6946 0.6969
0.6977 0.6969 0.6946 0.6907 0.6854 0.6786 0.6703 0.6607 0.6498 0.6376 0.6243 0.6099 0.5945 0.5782
0.5611 0.5434 0.5250 0.5061 0.4868 0.4671 0.4473 0.4274 0.4075 0.3876 0.3679 0.3484 0.3292 0.3104
0.2920 0.2741 0.2567 0.2398 0.2236 0.2080 0.1931 0.1789 0.1653 0.1524 0.1402 0.1287 0.1179}

row 59 = {0.1235 0.1346 0.1463 0.1586 0.1717 0.1853 0.1997 0.2146 0.2302 0.2463 0.2630
0.2802 0.2979 0.3159 0.3344 0.3531 0.3720 0.3911 0.4102 0.4293 0.4483 0.4671 0.4857 0.5038 0.5215
0.5385 0.5549 0.5706 0.5853 0.5992 0.6119 0.6236 0.6341 0.6433 0.6512 0.6578 0.6629 0.6666 0.6688
0.6696 0.6688 0.6666 0.6629 0.6578 0.6512 0.6433 0.6341 0.6236 0.6119 0.5992 0.5853 0.5706 0.5549
0.5385 0.5215 0.5038 0.4857 0.4671 0.4483 0.4293 0.4102 0.3911 0.3720 0.3531 0.3344 0.3159 0.2979
0.2802 0.2630 0.2463 0.2302 0.2146 0.1997 0.1853 0.1717 0.1586 0.1463 0.1346 0.1235 0.1132}

row 60 = {0.1183 0.1289 0.1401 0.1519 0.1644 0.1775 0.1912 0.2055 0.2204 0.2359 0.2519
0.2683 0.2852 0.3025 0.3202 0.3381 0.3562 0.3745 0.3928 0.4111 0.4293 0.4473 0.4651 0.4824 0.4994
0.5157 0.5314 0.5464 0.5605 0.5738 0.5860 0.5972 0.6072 0.6160 0.6236 0.6299 0.6348 0.6383 0.6405
0.6412 0.6405 0.6383 0.6348 0.6299 0.6236 0.6160 0.6072 0.5972 0.5860 0.5738 0.5605 0.5464 0.5314
0.5157 0.4994 0.4824 0.4651 0.4473 0.4293 0.4111 0.3928 0.3745 0.3562 0.3381 0.3202 0.3025 0.2852
0.2683 0.2519 0.2359 0.2204 0.2055 0.1912 0.1775 0.1644 0.1519 0.1401 0.1289 0.1183 0.1084}

row 61 = {0.1130 0.1231 0.1338 0.1451 0.1571 0.1696 0.1827 0.1964 0.2106 0.2254 0.2406
0.2564 0.2725 0.2891 0.3059 0.3230 0.3404 0.3578 0.3753 0.3928 0.4102 0.4274 0.4444 0.4610 0.4771
0.4927 0.5077 0.5220 0.5356 0.5482 0.5599 0.5706 0.5802 0.5886 0.5958 0.6018 0.6065 0.6099 0.6119
0.6126 0.6119 0.6099 0.6065 0.6018 0.5958 0.5886 0.5802 0.5706 0.5599 0.5482 0.5356 0.5220 0.5077
0.4927 0.4771 0.4610 0.4444 0.4274 0.4102 0.3928 0.3753 0.3578 0.3404 0.3230 0.3059 0.2891 0.2725
0.2564 0.2406 0.2254 0.2106 0.1964 0.1827 0.1696 0.1571 0.1451 0.1338 0.1231 0.1130 0.1035}

row 62 = {0.1078 0.1174 0.1276 0.1384 0.1497 0.1617 0.1742 0.1872 0.2008 0.2149 0.2294
0.2444 0.2598 0.2756 0.2916 0.3080 0.3245 0.3411 0.3578 0.3745 0.3911 0.4075 0.4236 0.4395 0.4549

0.4697 0.4841 0.4977 0.5106 0.5226 0.5338 0.5440 0.5531 0.5611 0.5680 0.5738 0.5782 0.5815 0.5834
0.5840 0.5834 0.5815 0.5782 0.5738 0.5680 0.5611 0.5531 0.5440 0.5338 0.5226 0.5106 0.4977 0.4841
0.4697 0.4549 0.4395 0.4236 0.4075 0.3911 0.3745 0.3578 0.3411 0.3245 0.3080 0.2916 0.2756 0.2598
0.2444 0.2294 0.2149 0.2008 0.1872 0.1742 0.1617 0.1497 0.1384 0.1276 0.1174 0.1078 0.0987}

row 63 = {0.1025 0.1117 0.1214 0.1316 0.1424 0.1538 0.1657 0.1781 0.1910 0.2044 0.2182
0.2325 0.2471 0.2621 0.2774 0.2929 0.3086 0.3245 0.3404 0.3562 0.3720 0.3876 0.4030 0.4180 0.4327
0.4468 0.4604 0.4734 0.4857 0.4971 0.5077 0.5174 0.5261 0.5338 0.5403 0.5458 0.5500 0.5531 0.5549
0.5556 0.5549 0.5531 0.5500 0.5458 0.5403 0.5338 0.5261 0.5174 0.5077 0.4971 0.4857 0.4734 0.4604
0.4468 0.4327 0.4180 0.4030 0.3876 0.3720 0.3562 0.3404 0.3245 0.3086 0.2929 0.2774 0.2621 0.2471
0.2325 0.2182 0.2044 0.1910 0.1781 0.1657 0.1538 0.1424 0.1316 0.1214 0.1117 0.1025 0.0939}

row 64 = {0.0973 0.1060 0.1152 0.1249 0.1352 0.1460 0.1572 0.1690 0.1813 0.1940 0.2071
0.2207 0.2346 0.2488 0.2633 0.2780 0.2929 0.3080 0.3230 0.3381 0.3531 0.3679 0.3825 0.3968 0.4107
0.4241 0.4370 0.4493 0.4610 0.4718 0.4819 0.4911 0.4994 0.5066 0.5128 0.5180 0.5220 0.5250 0.5267
0.5273 0.5267 0.5250 0.5220 0.5180 0.5128 0.5066 0.4994 0.4911 0.4819 0.4718 0.4610 0.4493 0.4370
0.4241 0.4107 0.3968 0.3825 0.3679 0.3531 0.3381 0.3230 0.3080 0.2929 0.2780 0.2633 0.2488 0.2346
0.2207 0.2071 0.1940 0.1813 0.1690 0.1572 0.1460 0.1352 0.1249 0.1152 0.1060 0.0973 0.0891}

row 65 = {0.0921 0.1004 0.1091 0.1183 0.1280 0.1382 0.1489 0.1601 0.1717 0.1837 0.1961
0.2090 0.2221 0.2356 0.2494 0.2633 0.2774 0.2916 0.3059 0.3202 0.3344 0.3484 0.3622 0.3757 0.3889
0.4016 0.4139 0.4255 0.4365 0.4468 0.4564 0.4651 0.4729 0.4798 0.4857 0.4906 0.4944 0.4971 0.4988
0.4994 0.4988 0.4971 0.4944 0.4906 0.4857 0.4798 0.4729 0.4651 0.4564 0.4468 0.4365 0.4255 0.4139
0.4016 0.3889 0.3757 0.3622 0.3484 0.3344 0.3202 0.3059 0.2916 0.2774 0.2633 0.2494 0.2356 0.2221
0.2090 0.1961 0.1837 0.1717 0.1601 0.1489 0.1382 0.1280 0.1183 0.1091 0.1004 0.0921 0.0844}

row 66 = {0.0871 0.0948 0.1031 0.1118 0.1210 0.1306 0.1407 0.1512 0.1622 0.1736 0.1853
0.1975 0.2099 0.2226 0.2356 0.2488 0.2621 0.2756 0.2891 0.3025 0.3159 0.3292 0.3422 0.3550 0.3675
0.3795 0.3911 0.4021 0.4125 0.4222 0.4312 0.4395 0.4468 0.4533 0.4589 0.4635 0.4671 0.4697 0.4713
0.4718 0.4713 0.4697 0.4671 0.4635 0.4589 0.4533 0.4468 0.4395 0.4312 0.4222 0.4125 0.4021 0.3911
0.3795 0.3675 0.3550 0.3422 0.3292 0.3159 0.3025 0.2891 0.2756 0.2621 0.2488 0.2356 0.2226 0.2099
0.1975 0.1853 0.1736 0.1622 0.1512 0.1407 0.1306 0.1210 0.1118 0.1031 0.0948 0.0871 0.0797}

row 67 = {0.0821 0.0894 0.0972 0.1054 0.1141 0.1231 0.1327 0.1426 0.1529 0.1637 0.1747
0.1862 0.1979 0.2099 0.2221 0.2346 0.2471 0.2598 0.2725 0.2852 0.2979 0.3104 0.3227 0.3347 0.3465
0.3578 0.3687 0.3791 0.3889 0.3981 0.4066 0.4143 0.4213 0.4274 0.4327 0.4370 0.4404 0.4429 0.4444
0.4449 0.4444 0.4429 0.4404 0.4370 0.4327 0.4274 0.4213 0.4143 0.4066 0.3981 0.3889 0.3791 0.3687
0.3578 0.3465 0.3347 0.3227 0.3104 0.2979 0.2852 0.2725 0.2598 0.2471 0.2346 0.2221 0.2099 0.1979
0.1862 0.1747 0.1637 0.1529 0.1426 0.1327 0.1231 0.1141 0.1054 0.0972 0.0894 0.0821 0.0752}

row 68 = {0.0772 0.0841 0.0914 0.0992 0.1073 0.1158 0.1248 0.1341 0.1439 0.1540 0.1644
0.1751 0.1862 0.1975 0.2090 0.2207 0.2325 0.2444 0.2564 0.2683 0.2802 0.2920 0.3035 0.3149 0.3259
0.3366 0.3468 0.3566 0.3658 0.3745 0.3825 0.3898 0.3963 0.4021 0.4070 0.4111 0.4143 0.4166 0.4180
0.4185 0.4180 0.4166 0.4143 0.4111 0.4070 0.4021 0.3963 0.3898 0.3825 0.3745 0.3658 0.3566 0.3468
0.3366 0.3259 0.3149 0.3035 0.2920 0.2802 0.2683 0.2564 0.2444 0.2325 0.2207 0.2090 0.1975 0.1862
0.1751 0.1644 0.1540 0.1439 0.1341 0.1248 0.1158 0.1073 0.0992 0.0914 0.0841 0.0772 0.0707}

row 69 = {0.0725 0.0790 0.0858 0.0931 0.1007 0.1087 0.1171 0.1259 0.1350 0.1445 0.1543
0.1644 0.1747 0.1853 0.1961 0.2071 0.2182 0.2294 0.2406 0.2519 0.2630 0.2741 0.2849 0.2956 0.3059
0.3159 0.3256 0.3347 0.3434 0.3515 0.3590 0.3658 0.3720 0.3774 0.3820 0.3859 0.3889 0.3911 0.3924
0.3928 0.3924 0.3911 0.3889 0.3859 0.3820 0.3774 0.3720 0.3658 0.3590 0.3515 0.3434 0.3347 0.3256
0.3159 0.3059 0.2956 0.2849 0.2741 0.2630 0.2519 0.2406 0.2294 0.2182 0.2071 0.1961 0.1853 0.1747
0.1644 0.1543 0.1445 0.1350 0.1259 0.1171 0.1087 0.1007 0.0931 0.0858 0.0790 0.0725 0.0664}

row 70 = {0.0679 0.0739 0.0804 0.0872 0.0943 0.1018 0.1097 0.1179 0.1265 0.1353 0.1445
0.1540 0.1637 0.1736 0.1837 0.1940 0.2044 0.2149 0.2254 0.2359 0.2463 0.2567 0.2668 0.2768 0.2865
0.2959 0.3049 0.3135 0.3216 0.3292 0.3362 0.3426 0.3484 0.3535 0.3578 0.3614 0.3642 0.3662 0.3675
0.3679 0.3675 0.3662 0.3642 0.3614 0.3578 0.3535 0.3484 0.3426 0.3362 0.3292 0.3216 0.3135 0.3049
0.2959 0.2865 0.2768 0.2668 0.2567 0.2463 0.2359 0.2254 0.2149 0.2044 0.1940 0.1837 0.1736 0.1637
0.1540 0.1445 0.1353 0.1265 0.1179 0.1097 0.1018 0.0943 0.0872 0.0804 0.0739 0.0679 0.0622}

row 71 = {0.0634 0.0691 0.0751 0.0814 0.0881 0.0952 0.1025 0.1102 0.1182 0.1265 0.1350
0.1439 0.1529 0.1622 0.1717 0.1813 0.1910 0.2008 0.2106 0.2204 0.2302 0.2398 0.2494 0.2587 0.2677

0.2765 0.2849 0.2929 0.3005 0.3076 0.3142 0.3202 0.3256 0.3303 0.3344 0.3377 0.3404 0.3422 0.3434
0.3438 0.3434 0.3422 0.3404 0.3377 0.3344 0.3303 0.3256 0.3202 0.3142 0.3076 0.3005 0.2929 0.2849
0.2765 0.2677 0.2587 0.2494 0.2398 0.2302 0.2204 0.2106 0.2008 0.1910 0.1813 0.1717 0.1622 0.1529
0.1439 0.1350 0.1265 0.1182 0.1102 0.1025 0.0952 0.0881 0.0814 0.0751 0.0691 0.0634 0.0581}

row 72 = {0.0591 0.0644 0.0700 0.0759 0.0822 0.0887 0.0956 0.1027 0.1102 0.1179 0.1259
0.1341 0.1426 0.1512 0.1601 0.1690 0.1781 0.1872 0.1964 0.2055 0.2146 0.2236 0.2325 0.2412 0.2496
0.2578 0.2657 0.2731 0.2802 0.2868 0.2929 0.2985 0.3035 0.3080 0.3117 0.3149 0.3173 0.3191 0.3202
0.3205 0.3202 0.3191 0.3173 0.3149 0.3117 0.3080 0.3035 0.2985 0.2929 0.2868 0.2802 0.2731 0.2657
0.2578 0.2496 0.2412 0.2325 0.2236 0.2146 0.2055 0.1964 0.1872 0.1781 0.1690 0.1601 0.1512 0.1426
0.1341 0.1259 0.1179 0.1102 0.1027 0.0956 0.0887 0.0822 0.0759 0.0700 0.0644 0.0591 0.0542}

row 73 = {0.0550 0.0599 0.0651 0.0707 0.0765 0.0825 0.0889 0.0956 0.1025 0.1097 0.1171
0.1248 0.1327 0.1407 0.1489 0.1572 0.1657 0.1742 0.1827 0.1912 0.1997 0.2080 0.2163 0.2244 0.2322
0.2398 0.2471 0.2541 0.2607 0.2668 0.2725 0.2777 0.2824 0.2865 0.2900 0.2929 0.2952 0.2969 0.2979
0.2982 0.2979 0.2969 0.2952 0.2929 0.2900 0.2865 0.2824 0.2777 0.2725 0.2668 0.2607 0.2541 0.2471
0.2398 0.2322 0.2244 0.2163 0.2080 0.1997 0.1912 0.1827 0.1742 0.1657 0.1572 0.1489 0.1407 0.1327
0.1248 0.1171 0.1097 0.1025 0.0956 0.0889 0.0825 0.0765 0.0707 0.0651 0.0599 0.0550 0.0504}

row 74 = {0.0511 0.0556 0.0605 0.0656 0.0710 0.0766 0.0825 0.0887 0.0952 0.1018 0.1087
0.1158 0.1231 0.1306 0.1382 0.1460 0.1538 0.1617 0.1696 0.1775 0.1853 0.1931 0.2008 0.2083 0.2156
0.2226 0.2294 0.2359 0.2420 0.2477 0.2530 0.2578 0.2621 0.2660 0.2692 0.2719 0.2741 0.2756 0.2765
0.2768 0.2765 0.2756 0.2741 0.2719 0.2692 0.2660 0.2621 0.2578 0.2530 0.2477 0.2420 0.2359 0.2294
0.2226 0.2156 0.2083 0.2008 0.1931 0.1853 0.1775 0.1696 0.1617 0.1538 0.1460 0.1382 0.1306 0.1231
0.1158 0.1087 0.1018 0.0952 0.0887 0.0825 0.0766 0.0710 0.0656 0.0605 0.0556 0.0511 0.0468}

row 75 = {0.0473 0.0515 0.0560 0.0607 0.0657 0.0710 0.0765 0.0822 0.0881 0.0943 0.1007
0.1073 0.1141 0.1210 0.1280 0.1352 0.1424 0.1497 0.1571 0.1644 0.1717 0.1789 0.1860 0.1929 0.1997
0.2062 0.2125 0.2185 0.2241 0.2294 0.2343 0.2388 0.2428 0.2463 0.2494 0.2519 0.2538 0.2552 0.2561
0.2564 0.2561 0.2552 0.2538 0.2519 0.2494 0.2463 0.2428 0.2388 0.2343 0.2294 0.2241 0.2185 0.2125
0.2062 0.1997 0.1929 0.1860 0.1789 0.1717 0.1644 0.1571 0.1497 0.1424 0.1352 0.1280 0.1210 0.1141
0.1073 0.1007 0.0943 0.0881 0.0822 0.0765 0.0710 0.0657 0.0607 0.0560 0.0515 0.0473 0.0433}

row 76 = {0.0437 0.0476 0.0518 0.0561 0.0607 0.0656 0.0707 0.0759 0.0814 0.0872 0.0931
0.0992 0.1054 0.1118 0.1183 0.1249 0.1316 0.1384 0.1451 0.1519 0.1586 0.1653 0.1719 0.1783 0.1845
0.1906 0.1964 0.2019 0.2071 0.2120 0.2165 0.2207 0.2244 0.2276 0.2304 0.2328 0.2346 0.2359 0.2367
0.2369 0.2367 0.2359 0.2346 0.2328 0.2304 0.2276 0.2244 0.2207 0.2165 0.2120 0.2071 0.2019 0.1964
0.1906 0.1845 0.1783 0.1719 0.1653 0.1586 0.1519 0.1451 0.1384 0.1316 0.1249 0.1183 0.1118 0.1054
0.0992 0.0931 0.0872 0.0814 0.0759 0.0707 0.0656 0.0607 0.0561 0.0518 0.0476 0.0437 0.0400}

row 77 = {0.0403 0.0439 0.0477 0.0518 0.0560 0.0605 0.0651 0.0700 0.0751 0.0804 0.0858
0.0914 0.0972 0.1031 0.1091 0.1152 0.1214 0.1276 0.1338 0.1401 0.1463 0.1524 0.1585 0.1644 0.1701
0.1757 0.1811 0.1862 0.1910 0.1955 0.1997 0.2035 0.2069 0.2099 0.2125 0.2146 0.2163 0.2175 0.2182
0.2185 0.2182 0.2175 0.2163 0.2146 0.2125 0.2099 0.2069 0.2035 0.1997 0.1955 0.1910 0.1862 0.1811
0.1757 0.1701 0.1644 0.1585 0.1524 0.1463 0.1401 0.1338 0.1276 0.1214 0.1152 0.1091 0.1031 0.0972
0.0914 0.0858 0.0804 0.0751 0.0700 0.0651 0.0605 0.0560 0.0518 0.0477 0.0439 0.0403 0.0369}

row 78 = {0.0371 0.0404 0.0439 0.0476 0.0515 0.0556 0.0599 0.0644 0.0691 0.0739 0.0790
0.0841 0.0894 0.0948 0.1004 0.1060 0.1117 0.1174 0.1231 0.1289 0.1346 0.1402 0.1458 0.1512 0.1565
0.1617 0.1666 0.1713 0.1757 0.1799 0.1837 0.1872 0.1904 0.1931 0.1955 0.1975 0.1990 0.2001 0.2008
0.2010 0.2008 0.2001 0.1990 0.1975 0.1955 0.1931 0.1904 0.1872 0.1837 0.1799 0.1757 0.1713 0.1666
0.1617 0.1565 0.1512 0.1458 0.1402 0.1346 0.1289 0.1231 0.1174 0.1117 0.1060 0.1004 0.0948 0.0894
0.0841 0.0790 0.0739 0.0691 0.0644 0.0599 0.0556 0.0515 0.0476 0.0439 0.0404 0.0371 0.0340}

row 79 = {0.0340 0.0371 0.0403 0.0437 0.0473 0.0511 0.0550 0.0591 0.0634 0.0679 0.0725
0.0772 0.0821 0.0871 0.0921 0.0973 0.1025 0.1078 0.1130 0.1183 0.1235 0.1287 0.1338 0.1388 0.1437
0.1484 0.1529 0.1572 0.1613 0.1651 0.1686 0.1719 0.1747 0.1773 0.1795 0.1813 0.1827 0.1837 0.1843
0.1845 0.1843 0.1837 0.1827 0.1813 0.1795 0.1773 0.1747 0.1719 0.1686 0.1651 0.1613 0.1572 0.1529
0.1484 0.1437 0.1388 0.1338 0.1287 0.1235 0.1183 0.1130 0.1078 0.1025 0.0973 0.0921 0.0871 0.0821
0.0772 0.0725 0.0679 0.0634 0.0591 0.0550 0.0511 0.0473 0.0437 0.0403 0.0371 0.0340 0.0312}

row 80 = {0.0312 0.0340 0.0369 0.0400 0.0433 0.0468 0.0504 0.0542 0.0581 0.0622 0.0664
0.0707 0.0752 0.0797 0.0844 0.0891 0.0939 0.0987 0.1035 0.1084 0.1132 0.1179 0.1226 0.1272 0.1316

0.1359 0.1401 0.1440 0.1478 0.1512 0.1545 0.1574 0.1601 0.1624 0.1644 0.1660 0.1673 0.1683 0.1688 0.1690 0.1688 0.1683 0.1673 0.1660 0.1644 0.1624 0.1601 0.1574 0.1545 0.1512 0.1478 0.1440 0.1401 0.1359 0.1316 0.1272 0.1226 0.1179 0.1132 0.1084 0.1035 0.0987 0.0939 0.0891 0.0844 0.0797 0.0752 0.0707 0.0664 0.0622 0.0581 0.0542 0.0504 0.0468 0.0433 0.0400 0.0369 0.0340 0.0312 0.0286}

A.2 Terminal Node Sets

The node set locations are defined by a row and a column location corresponding to the hexagonal grid as in Fig. A.1.

Small random terminal node set #1, 10 nodes, used in problems 1 and 3
(8,4) (20,4) (15,5) (17,7) (4,10) (10,10) (14,10) (21,11) (10,14) (18,14)

Small ring terminal node set, 7 nodes, used in problems 2 and 4
(7,5) (17,5) (3,9) (11,9) (21,9) (7,13) (17,13)

Small random terminal node set #2, 7 nodes, used in problem 5
(22,4) (12,6) (5,9) (16,10) (9,11) (12,14) (11,15)

Medium random terminal node set #1, 15 nodes, used in problems 6 and 7
(27,3) (31,3) (15,9) (7,11) (35,11) (1,13) (17,13) (31,13) (10,14) (14,16) (24,18) (32,18) (3,27) (23,27) (26,30)

Medium ring terminal node set, 15 nodes, used in problems 8 and 9
(13,3) (23,3) (8,8) (28,8) (14,12) (4,14) (34,14) (18,18) (4,20) (34,20) (26,22) (9,27) (27,27) (15,33) (21,33)

Medium random terminal node set #2, 15 nodes, used in problem 10
(24,4) (30,4) (3,7) (5,7) (4,16) (12,16) (28,18) (29,19) (5,23) (15,23) (33,23) (22,30) (25,31) (23,33) (27,33)

Large random terminal node set #1, 20 nodes, used in problem 11
(44,2) (49,7) (45,11) (22,12) (1,13) (11,13) (47,13) (49,15) (46,18) (31,21) (33,23) (38,40) (50,40) (19,41) (3,45) (11,45) (39,45) (18,46) (13,47) (15,49)

Large ring terminal node set, 20 nodes, used in problem 12
(16,12) (22,12) (30,12) (12,16) (34,16) (37,19) (8,20) (5,23) (41,23) (2,26) (22,26) (44,26) (6,30) (42,30) (38,34) (11,35) (15,39) (33,39) (19,43) (27,43)

Large random terminal node set #2, 32 nodes, used in problem 13
(35,5) (42,6) (64,6) (4,8) (66,8) (46,16) (15,19) (43,19) (49,19) (55,19) (71,19) (77,19) (2,24) (72,24) (22,26) (66,26) (74,26) (36,32) (40,34) (75,45) (54,46) (30,48) (72,48) (31,59) (6,62) (68,62) (31,65) (47,65) (55,65) (31,69) (77,69) (25,71)

A.3 Solutions

A.3.1 Our GA Steiner Node Locations

The Steiner tree solutions can be reconstructed by finding the minimal spanning tree on the set of terminal and Steiner nodes as described in Sec. 4.3.2.

Problem 1
(11,1) (19,1) (21,3) (19,5) (20,6) (19,7) (21,9) (20,10)

Problem 2
(3,5) (19,5) (1,7) (21,7) (20,8) (20,10) (1,11) (21,11) (3,13) (19,13) (1,15) (21,15) (3,17) (19,17)

Problem 3
(3,1) (19,1) (1,3) (21,3) (2,4) (1,5) (19,5) (21,5) (2,6) (20,6) (1,7) (19,7) (21,7) (2,8) (20,8) (1,9) (21,9) (2,10) (20,10) (20,12) (21,13) (20,14)

Problem 4
(3,1) (19,1) (1,3) (21,3) (2,4) (6,4) (20,4) (1,5) (19,5) (21,5) (2,6) (20,6) (1,7) (21,7) (2,8) (20,8) (1,9) (2,10) (20,10) (1,11) (21,11) (20,12) (3,13) (21,13) (18,14) (20,14)

Problem 5
(20,6) (11,7) (12,8) (11,9) (14,12)

Problem 6
(21,3) (33,3) (35,5) (34,6) (35,7) (34,8) (35,9) (34,10) (3,11) (17,11) (16,12) (33,13) (2,14) (1,15) (35,15) (2,16) (12,16) (1,17) (2,18) (1,19) (2,20) (1,21) (2,22) (1,23) (2,24) (1,25) (1,29) (21,29) (2,30) (22,30) (1,31) (2,32) (1,33) (3,35) (17,35)

Problem 7
(21,3) (35,7) (34,8) (35,9) (8,10) (14,10) (34,10) (3,11) (33,13) (2,14) (1,15) (35,15) (2,16) (12,16) (1,17) (2,18) (1,19) (2,20) (1,21) (35,21) (2,22) (34,22) (1,23) (35,23) (2,24) (34,24) (1,25) (35,25) (30,30)

Problem 8
(3,1) (15,1) (25,1) (33,1) (1,3) (35,3) (2,4) (34,4) (1,5) (35,5) (2,6) (34,6) (1,7) (35,7) (2,8) (34,8) (1,9) (35,9) (2,10) (34,10) (1,11) (35,11) (34,12) (35,13) (12,14) (35,15) (34,16) (1,17) (35,17) (16,18) (34,18) (35,19) (28,20) (35,21) (34,22) (35,23) (34,24) (35,25) (34,26) (35,27) (34,28) (35,29) (34,30) (35,31) (34,32) (35,33) (23,35) (33,35)

Problem 9
(30,8) (5,11) (6,12) (35,13) (35,15) (34,16) (1,17) (35,17) (22,18) (34,18) (35,19) (32,22)

Problem 10
(7,5) (15,5) (19,5) (23,5) (16,6) (18,6) (6,8) (5,9) (6,10) (5,11) (9,15) (11,15) (5,17) (7,17) (13,17) (12,18) (13,19) (25,19) (27,19) (2,20) (12,20) (20,20) (24,20) (13,21) (19,21) (12,22) (26,22) (30,22) (13,23) (31,23) (12,24) (14,26) (16,26) (17,27) (19,27) (23,31) (26,32) (25,33)

Problem 11
(7,1) (43,1) (1,7) (2,8) (1,9) (2,10) (1,11) (2,12) (12,12) (50,22) (49,23) (50,24) (49,25) (50,26) (49,27) (50,28) (49,29) (50,30) (49,31) (50,32) (49,33) (50,34) (49,35) (50,36) (49,37) (50,38) (49,39) (16,44) (42,44) (41,45) (45,45) (16,50) (34,50)

Problem 12
(36,12) (38,14) (36,16) (38,18) (41,29) (12,36) (31,39)

Problem 13
(9,3) (33,3) (41,5) (74,8) (1,11) (2,12) (1,13) (80,14) (79,15) (80,16) (7,19) (30,26) (38,32) (80,32) (79,33) (80,34) (79,35) (80,36) (79,37) (80,38) (79,39) (80,40) (74,46) (25,53) (80,56) (79,57) (80,58) (79,59) (80,60) (79,61) (28,62) (80,62) (79,63) (4,64) (80,64) (65,65) (79,65) (80,66) (27,69) (11,71)

A.3.2 Optimal Steiner Node Locations

Problem 1
(11,1) (19,1) (21,3) (19,5) (20,6) (19,7) (21,9) (20,10)

Problem 2
(3,1) (19,1) (1,3) (21,3) (3,5) (19,5) (1,7) (21,7) (20,8) (20,10) (1,11) (21,11) (3,13) (19,13)

Problem 3
(1,1) (21,1) (2,2) (20,2) (1,3) (21,3) (2,4) (1,5) (19,5) (21,5) (2,6) (20,6) (1,7) (19,7) (21,7) (2,8) (20,8) (1,9) (21,9) (2,10) (20,10) (20,12) (21,13) (20,14)

Problem 4
(1,1) (21,1) (2,2) (20,2) (1,3) (21,3) (2,4) (6,4) (20,4) (1,5) (19,5) (21,5) (2,6) (20,6) (1,7) (21,7) (2,8) (20,8) (1,9) (2,10) (20,10) (1,11) (21,11) (20,12) (3,13) (21,13) (18,14) (20,14)

Problem 5
(20,6) (11,7) (12,8) (11,9) (14,12)

Problem 6
(21,3) (33,3) (35,5) (34,6) (35,7) (34,8) (35,9) (34,10) (3,11) (17,11) (16,12) (33,13) (2,14) (1,15) (35,15) (2,16) (8,16) (1,17) (2,18) (1,19) (2,20) (1,21) (2,22) (1,23) (2,24) (1,25) (1,29) (21,29) (2,30) (22,30) (1,31) (2,32) (1,33) (3,35) (17,35)

Problem 7
(21,3) (35,7) (34,8) (9,9) (35,9) (34,10) (3,11) (13,13) (33,13) (2,14) (12,14) (1,15) (35,15) (2,16) (1,17) (2,18) (1,19) (2,20) (1,21) (35,21) (2,22) (34,22) (1,23) (35,23) (2,24) (34,24) (1,25) (35,25) (30,30)

Problem 8
(3,1) (13,1) (23,1) (33,1) (14,2) (22,2) (1,3) (35,3) (2,4) (34,4) (1,5) (35,5) (2,6) (34,6) (1,7) (35,7) (2,8) (34,8) (1,9) (35,9) (2,10) (34,10) (1,11) (35,11) (2,12) (34,12) (1,13) (35,13) (2,14) (12,14) (1,15) (35,15) (2,16) (34,16) (1,17) (35,17) (16,18) (34,18) (35,19) (35,21) (34,22) (35,23) (34,24) (35,25) (33,27) (35,29) (34,30) (35,31) (34,32) (35,33) (23,35) (33,35)

Problem 9
(30,8) (5,11) (6,12) (35,13) (35,15) (34,16) (1,17) (35,17) (22,18) (34,18) (35,19) (32,22)

Problem 10
(7,5) (15,5) (19,5) (23,5) (16,6) (18,6) (6,8) (5,9) (6,10) (5,11) (9,15) (11,15) (5,17) (7,17) (13,17) (12,18) (13,19) (25,19) (27,19) (12,20) (20,20) (24,20) (13,21) (19,21) (12,22) (26,22) (30,22) (13,23) (31,23) (11,25) (13,25) (8,26) (10,26) (14,26) (16,26) (17,27) (19,27) (23,31) (26,32) (25,33)

A.3.3 Solution Plots

In this section we display the optimal Steiner trees and the lowest cost Steiner trees found by our GA for problems 1 through 10. The optimal solutions are on the left side of each page, while our GA solutions are on the right (see Figures A.2 through A.21). In each plot, triangles are Steiner nodes, black circles are terminal nodes, and the lighter the shading of a cell, the more costly the cell. In Figures A.22 through A.24 we display the lowest cost solutions found by our GA for the large problems (11, 12, and 13). The optimal solutions for these problems were not obtained due to the length of running time required. All tree costs may be found in Table 4.2. Recall from that table that our best GA solution is optimal for problems $1 - 5$ and 9. The discrepancies in the solutions for problems 2, 3, and 4 indicate that there is more than one optimal solution in each of these problems.

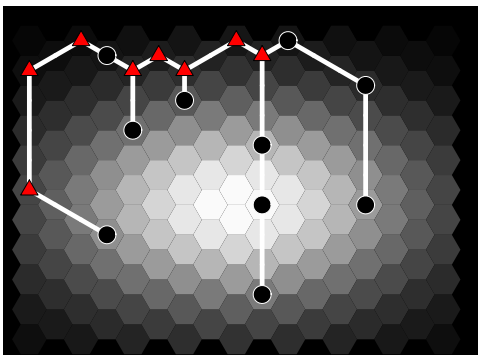

Figure A.2: Optimal tree for Problem 1

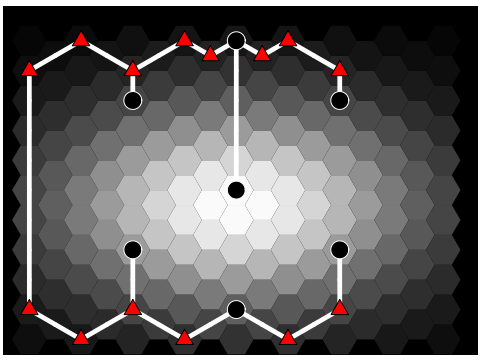

Figure A.3: Optimal tree for Problem 2

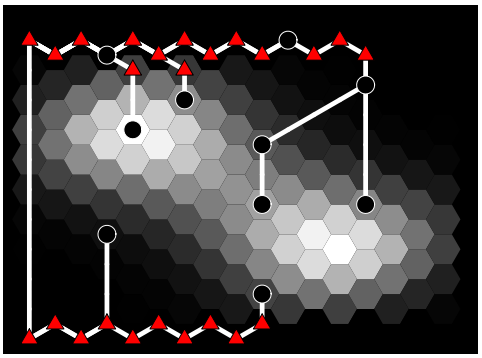

Figure A.4: Optimal tree for Problem 3

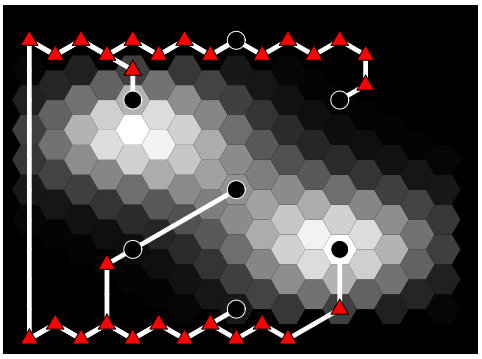

Figure A.5: Optimal tree for Problem 4

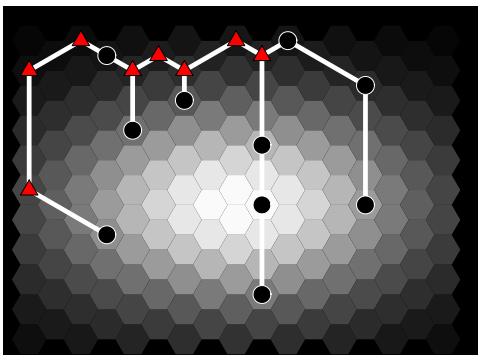

Figure A.6: Our best GA tree for Prob. 1

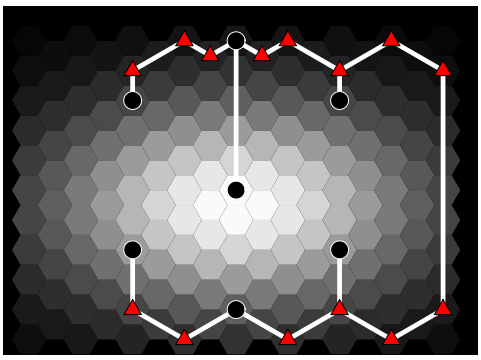

Figure A.7: Our best GA tree for Prob. 2

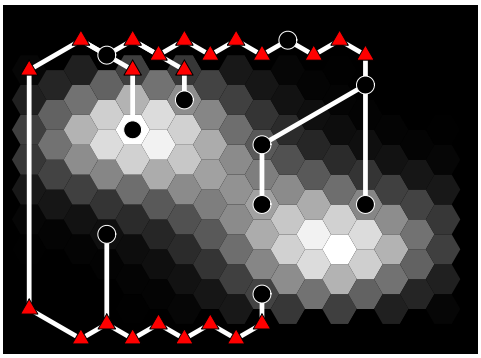

Figure A.8: Our best GA tree for Prob. 3

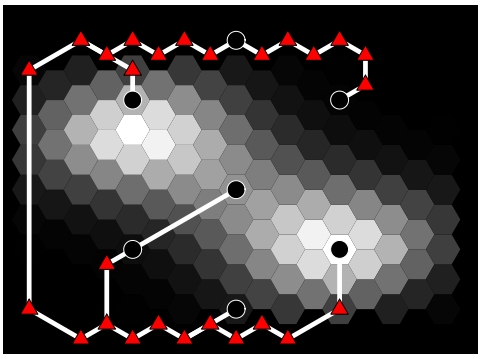

Figure A.9: Our best GA tree for Prob. 4

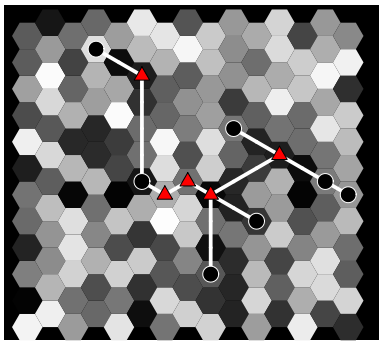

Figure A.10: Optimal tree for Problem 5

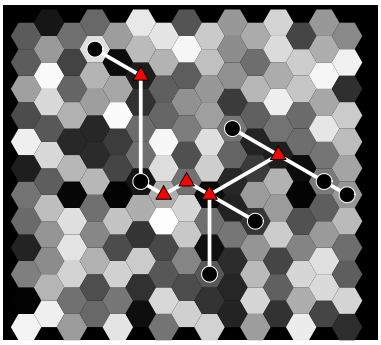

Figure A.14: Our best GA tree for Prob. 5

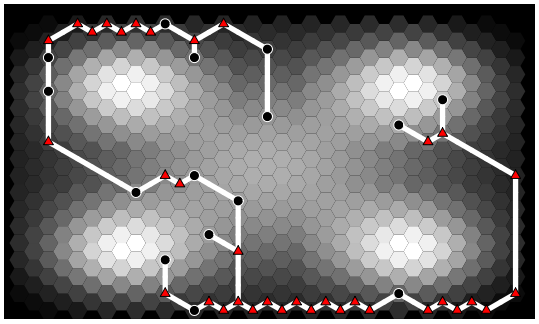

Figure A.11: Optimal tree for Problem 6

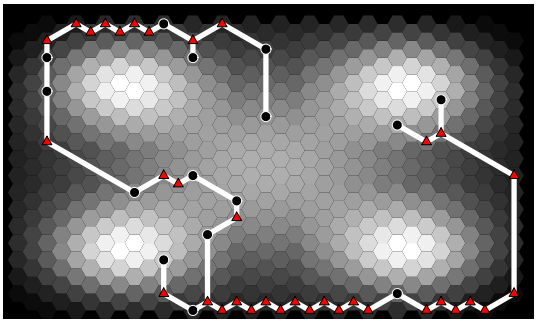

Figure A.15: Our best GA tree for Prob. 6

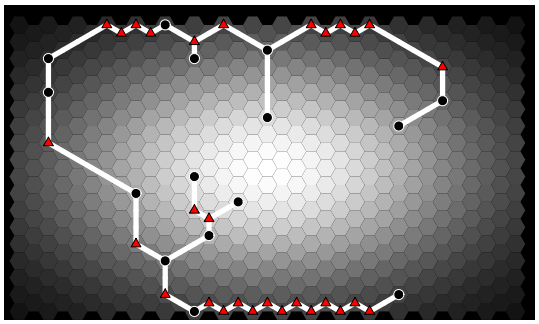

Figure A.12: Optimal tree for Problem 7

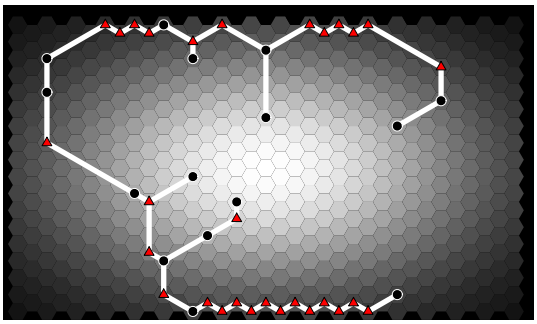

Figure A.16: Our best GA tree for Prob. 7

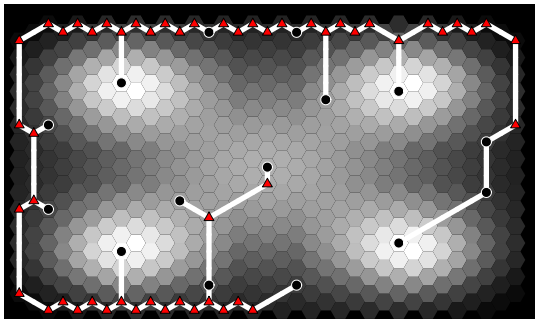

Figure A.13: Optimal tree for Problem 8

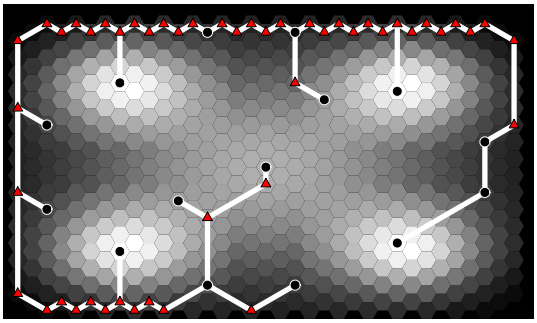

Figure A.17: Our best GA tree for Prob. 8

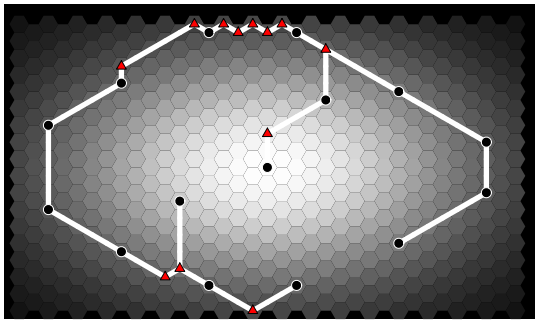

Figure A.18: Optimal tree for Problem 9

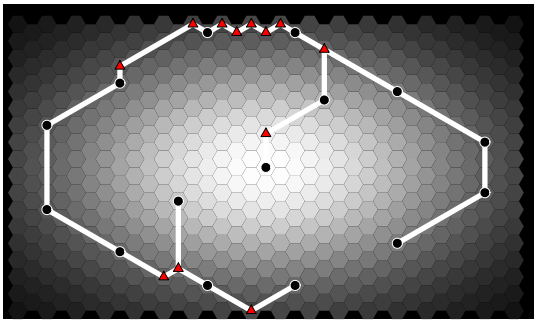

Figure A.20: Our best GA tree for Prob. 9

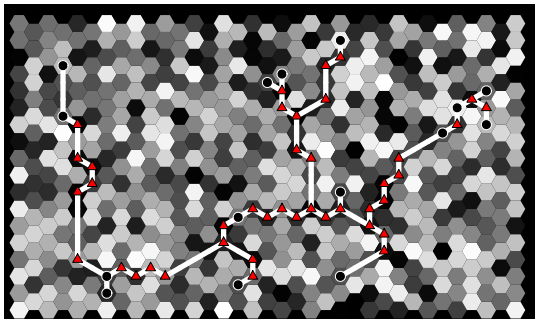

Figure A.19: Optimal tree for Problem 10

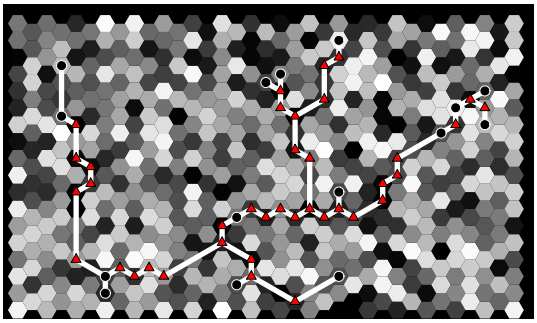

Figure A.21: Our best GA tree for Prob. 10

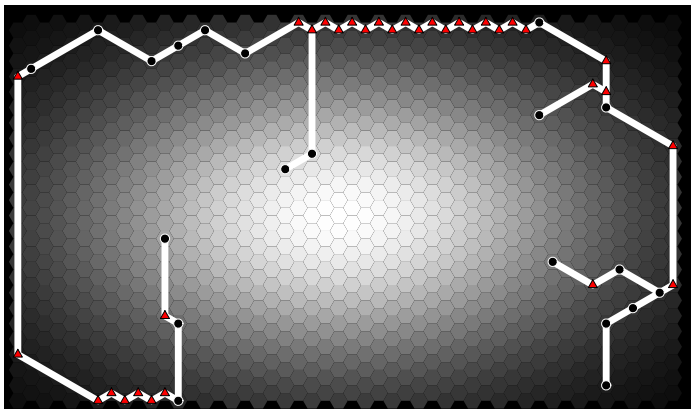

Figure A.22: Our best GA tree for Prob. 11

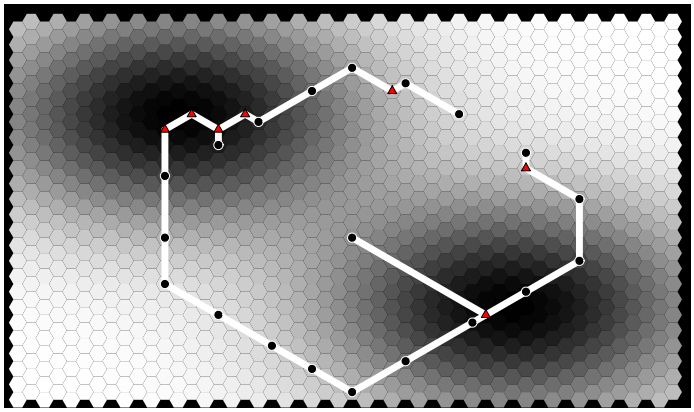

Figure A.23: Our best GA tree for Prob. 12

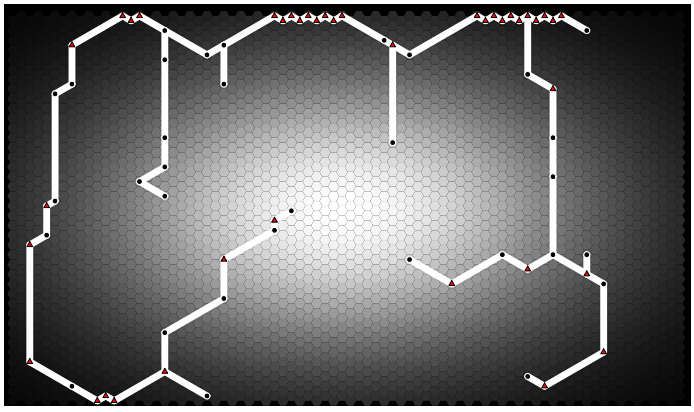

Figure A.24: Our best GA tree for Prob. 13

BIBLIOGRAPHY

[1] I. Frommer, E. Harder, B. Hunt, R. Lance, E. Ott, and J. Yorke. Modeling congested Internet connections. In *Proceedings of IASTED CCN*, pages 319–324, 2004.

[2] I. Frommer, B. Golden, and G. Pundoor. Heuristic methods for solving Euclidean non-uniform Steiner tree problems. In *Proceedings of INFORMS ICS*, pages 133–148, 2005.

[3] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet. Preprint, available from `http://www.icir.org/tbit`.

[4] M. Fomenkov, K. Keys, D. Moore, and k claffy. Longitudinal study of Internet traffic in 1998-2003. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), 2003.

[5] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE Trans. on Networking*, 1(7):397–413, 1993.

[6] V. Misra, W. Gong, and D. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of ACM SIGCOMM*, pages 151–160, 2000.

[7] V. Firoiu and M. Borden. Study of active queue management for congestion control. In *Proceedings of IEEE INFOCOM*, pages 1435–1444, 2000.

[8] P. Ranjan, E. Abed, and R. La. Nonlinear instabilities in TCP-RED. In *Proceedings of IEEE INFOCOM*, pages 249–258, 2002.

[9] S.H. Low, F. Paganini, J. Wang, S. Adlakha, and J. Doyle. Dynamics of TCP/RED and a scalable control. In *Proceedings of IEEE INFOCOM*, pages 239–248, 2002.

[10] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3):67–82, 1997.

[11] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM*, pages 303–314, 1998.

[12] J. Hespanha, S. Bohacek, K. Obraczka, and J. Lee. *Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, Volume 2034*, chapter Hybrid Modeling of TCP Congestion Control, pages 291–304. Springer-Verlag, 2001.

[13] S. Bohacek, J. Hespanha, J. Lee, and K. Obraczka. A hybrid systems modeling framework for fast and accurate simulation of data communication networks. In *Proceedings of ACM SIGMETRICS*, pages 58–69, 2003.

[14] Y. Liu, F. Lo Presti, V. Misra, D. Towsley, and Y. Gu. Fluid models and solutions for large-scale IP networks. In *Proceedings of ACM SIGMETRICS*, pages 91–101, 2003.

[15] `http://www.isi.edu/nsnam/ns/`.

[16] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F*, 140(2):107–113, 1993.

[17] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer, New York, 2001.

[18] Adtech AX/4000 10 Gbps Generator/Analyzer Test Module with OC-192 POS/BERT/10GBASE-W Ethernet Interface, `http://www.spirentcom.com/documents/189.pdf`.

[19] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of IEEE INFOCOM*, pages 996–1003, 1998.

[20] `http://www.thefengs.com/wuchang/work/cstrike/CSE-02-005.pdf`.

[21] A. Kuzmanovic and E. Knightly. Low-rate TCP-targeted denial of service attacks. In *Proceedings of ACM SIGCOMM*, pages 75–86, 2003.

[22] A. Veres and M. Boda. The chaotic nature of TCP congestion control. In *Proceedings of IEEE INFOCOM*, pages 1715–1723, 2000.

[23] A. Fekete, M. Marodi, and G. Vattay. On the prospects of chaos aware traffic modeling. *CoRR*, cond-mat/0208488, 2002.

[24] M. Liu, H. Zhang, and L. Trajkovic. Stroboscopic model and bifurcations in TCP/RED. In *Proceedings of IEEE International Symposium on Circuits and Systems*, 2005.

[25] J. Gao and N. Rao. TCP AIMD dynamics over Internet connections. *IEEE Communications Letters*, 9(1):4–6, 2005.

[26] P. Jain and S. Banerjee. Border-collision bifurcations in one-dimensional discontinuous maps. *International Journal on Bifurcation and Chaos*, 13(11):3341–3352, 2003.

[27] P. Winter. The Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.

[28] P. A. Thurber and G. Xue. Computing hexagonal Steiner trees using PCX. In *Proceedings of the International Conference on Electronics, Circuits and Systems*, pages 381–384, 1999.

[29] C. Coulston. Steiner minimal trees in a hexagonally partitioned space. *International Journal of Smart Engineering System Design*, 5:1–6, 2003.

[30] J. M. Smith and J. S. Liebman. Steiner trees, Steiner circuits, and the interference problem in building design. *Engineering Design*, 4:15–36, 1979.

[31] C. Alpert, G. Gandham, J. Hu, J. Neves, S. Quay, and S. Sapatnekar. Steiner tree optimization for buffers, blockages and bays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(4):556–562, 2001.

[32] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Number 53 in Annals of Discrete Mathematics. North-Holland, 1992.

[33] F. K. Hwang. A primer of the Euclidean Steiner tree problem. *Annals of operations research*, 33:73–84, 1991.

[34] S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[35] H. J. Prömel and A. Steger. *The Steiner Tree Problem: a tour through graphs, algorithms, and complexity*. Vieweg, Braunschweig/Wiesbaden, Germany, 2002.

[36] M. Zachariasen and P. Winter. *Workshop on Algorithm Engineering and Experimentation, Lecture Notes in Computer Science, Volume 1619*, chapter Obstacle-Avoiding Euclidean Steiner Trees in the Plane: An Exact Algorithm, pages 282–295. Springer-Verlag GmbH, 1999.

[37] Y. Kanemoto, R. Sugawara, and M. Ohmura. A genetic algorithm for the rectilinear Steiner tree in 3-D VLSI layout design. In *Proceedings of the 47th IEEE International Midwest Symposium on Circuits and Systems*, pages I–465 – I–468, 2004.

[38] C. Stanton and J. M. Smith. Steiner trees and 3-D macromolecular conformation. *INFORMS Journal on Computing*, 16(4):470–485, 2004.

[39] J. L. Ganley and J. P. Cohoon. A faster dynamic programming algorithm for exact rectilinear Steiner minimal trees. In *Proceedings of the Fourth Great Lakes Symposium on VLSI*, pages 238–241, 1994.

[40] C. Gröpl, S. Hougardy, T. Nierhoff, and H. J. Prömel. *Steiner Trees in Industry*, chapter Approximation algorithms for the Steiner tree problem in graphs. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.

[41] J. Barreiros. An hierarchic genetic algorithm for computing (near) optimal Euclidean Steiner trees. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 56–65, 2003.

[42] C. C. Ribeiro and M. C. de Souza. Tabu search for the Steiner problem in graphs. *Networks*, 36(2):138–146, 2000.

[43] L. J. Osborne and B. E. Gillett. A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA Journal on Computing*, 3:213–225, 1991.

[44] B. A. Julstrom. A scalable genetic algorithm for the rectilinear Steiner problem. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1169–1173, 2002.

[45] B. A. Julstrom. A hybrid evolutionary algorithm for the rectilinear Steiner problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 49–55, 2003.

[46] S. Dreyfus and R. Wagner. The Steiner tree problem in graphs. *Networks*, 1:195–207, 1972.

[47] J. Cheriyan and R. Ravi. *Lecture Notes on Approximation Algorithms for Network Problems*, chapter 7, Approximation algorithms for Steiner trees, pages 92–95. `http://www.gsia.cmu.edu/afs/andrew/gsia/ravi/WWW/new-lecnotes.html`.

[48] R. Floyd. Algorithm 97 (SHORTEST PATH). *Communications of the Association for Computing Machinery*, 5:345, 1962.

[49] R. Prim. Shortest connection networks and some generalizations. *Bell systems technical journal*, 36:1389–1401, 1957.

[50] S. H. Jung. Queen-bee evolution in genetic algorithms. *IEEE Electronic Letters*, 39:575–576, 2003.

[51] A. Mitchell. *The ESRI Guide to GIS, Volume 1: Geographic Patterns and Relationships*. ESRI Press, Redlands, California, 1999.

[52] B. Huang, R. L. Cheu, and Y. S. Liew. GIS and genetic algorithms for HAZMAT route planning with security considerations. *International Journal of Geographical Information Science*, 18(8):769–787, 2004.