12-2009

Branchings and Time Evolution of Reaction Networks

Changyuan Wang
Clemson University, cwang@alumni.clemson.edu

BRANCHINGS AND TIME EVOLUTION OF REACTION NETWORKS

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Ph.D
Physics

---

by
Changyuan Wang
December 2009

Accepted by:
Dr. Bradley S. Meyer, Committee Chair
Dr. Sean D. Brittain
Dr. Mark D. Leising
Dr. Murray S. Daw

ABSTRACT

In this thesis I analyze flows in reaction networks in terms of branchings in a digraph. If the coupled differential equations governing the rate of change of probabilities X of a state or species are finite-differenced in time, a matrix equation $(I+A\Delta t)X(t+\Delta t) = X(t)$ results, where $X(t)$ is a vector giving the probabilities at time t and $X(t+\Delta t)$ is a vector giving the probabilities at time $t+\Delta t$. I demonstrate that the matrix $(I+A\Delta t)$ may be written as the product of an incidence matrix and a weight matrix for a directed graph (digraph) representing the network. From this I demonstrate that individual diagonal element of the inverse matrix $(I+A\Delta t)^{-1}$ may be written as a sum of the exponential weight of all branchings rooted at the vertex corresponding to the root vertex in the digraph divided by the sum of the exponential weight of all branchings. I also demonstrate that the individual element of the inverse matrix at row i, column j is the sum of exponential weights of all branchings rooted at vertex i but with a path from vertex i to vertex j in the digraph divided by the sum of exponential weights of all branchings. From this I demonstrate how to compute $X(t+\Delta t)$ from $X(t)$ in terms of sums of branchings and how to compute effective transition rates. I then consider long-term solutions and demonstrate how to condense linear networks that obey detailed balance. This provides a useful connection to equilibrium analysis of the network. I then consider some implications of the branching analysis for the statistical mechanics of reaction networks, and I extend the analysis to non-linear networks. Finally I provide some example applications. I conclude that branchings in network digraphs hold promise for analyzing complicated reaction flows, and I list some future directions of possible research.

DEDICATION

I dedicate this dissertation to my parents Minbo Wang and Liying Dou and my wife Jing Zhang who have encouraged and motivated in the completion of this paper. It can not be done without their patience , understanding and support, and most of the love.

ACKNOWLEDGEMENTS

This paper is mostly done under the help and guidance from my advisor Dr. Bradley S. Meyer. I ran into numerous difficulties in carrying out the relevant research in this paper, and Dr. Meyer spent much time working with me and helped me to solve those difficulties. A lot of the key ideas directly came through my discussion with him. There were many times when I felt satisfied with a result but that was usually when he brought something deeper for me to ponder upon. His support and guidance was a very beneficial experience for me not only for the paper alone but for my future life. I would like to thank Raman Anantaraman and colleagues at Michigan State University for choosing me to attend the Rare Isotope Beams Summer School in 2007, an experience that allowed me to do experimental work on nuclear physics and thereby gain an appreciation for the nuclear data that underlie reaction networks. Finally, I would also like to thank my other committee members, Dr. Mark Leising, Dr. Sean Brittain and Dr. Murray Daw for their help.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Reaction networks are collections of states or species and the possible reactions linking them. In studying the network, one is interested in finding the time evolution of the population probabilities for the states or the abundances of the species.

For a linear network of dimension n, the time rate of change of the probabilities X_i of each state i are governed by n coupled linear differential equations:

$$\frac{dX_i}{dt} = -\Lambda_i X_i + \sum_r \sum_{j \neq i}^{n} \lambda_{ji}^{(r)} X_j, \tag{1.1}$$

where $\lambda_{ji}^{(r)}$ is the rate of transition from state j to state i due to reaction r and where the total destruction rate of i is

$$\Lambda_i = \sum_r \sum_{j \neq i}^{n} \lambda_{ij}^{(r)}. \tag{1.2}$$

The symbol $\sum_r$ indicates a sum over all reactions r in the network, and we note that $\lambda_{ij}^{(r)} = 0$ if reaction r does not link states or species i and j. The n equations of the form Eq. (1.1) may then be arranged in matrix form to read

$$\frac{dX}{dt} = -AX, \tag{1.3}$$

where the vector X comprises the individual state probabilities X_i and where the matrix elements of A are

$$A_{ii} = \Lambda_i \tag{1.4}$$

and

$$A_{ij} = -\sum_r \lambda_{ji}^{(r)}. \tag{1.5}$$

If the rates are independent of time, the solution to Eq. (1.3) is simply

$$X(t) = X(0) \exp\left(-At\right), \tag{1.6}$$

where $X(0)$ is the initial probability or abundance vector, $X(t)$ is the vector at time t, and $\exp(-At)$ is the exponential of the matrix $-At$. In many situations that arise, for example, in reaction networks in astrophysics, the rates are time dependent. Furthermore, the coupled ordinary differential equations in Eq. (1.3) are quite stiff. In such cases, it is common to finite difference in time (that is, take a finite time step Δt) and use implicit differentiation such that the derivatives are computed at time $t + \Delta t$. The resulting matrix equation is

$$(I + A\Delta t)\, X(t + \Delta t) = X(t), \tag{1.7}$$

and the solution is

$$X(t + \Delta t) = (I + A\Delta t)^{-1}\, X(t) \equiv P(t, \Delta t) X(t). \tag{1.8}$$

From the known vector at t, one thus integrates to find the vector at time $t + \Delta t$. The next time step Δt is then chosen so that no abundance or probability changes by more than a few percent.

This procedure can also accommodate non-linear networks. In such cases, instead of having a reaction $i \leftrightarrow j$, with a rate λ_{ij}, an appropriate reaction might be $i + j \leftrightarrow k + \ell$. The differential equation governing the probability or abundance of state or species i would include the terms

$$\frac{dX_i}{dt} = -f X_i X_j + r X_k X_\ell, \tag{1.9}$$

where f is the rate per interacting pair in the forward direction ($i + j \rightarrow k + \ell$) and r is the rate per interacting pair in the reverse direction ($k + \ell \rightarrow i + j$). If we finite difference this equation in time implicitly and linearize, we find

$$\frac{\Delta X}{\Delta t} = \frac{X(t + \Delta t) - X(t)}{\Delta t} =$$

$$-\frac{1}{2} f X_i(t) X_j(t+\Delta t) - \frac{1}{2} f X_i(t+\Delta t) X_j(t) + \frac{1}{2} r X_k(t) X_\ell(t+\Delta t) + \frac{1}{2} r X_k(t+\Delta t) X_\ell(t). \tag{1.10}$$

Other numbers of reactants or products can be accommodated similarly. The contribution of all such terms may then be collected into a matrix equation of the form in Eq. (1.7) with solution given by Eq. (1.8). The abundances or probabilities are then typically refined by Newton-Raphson iteration (e.g, [Hix and Meyer 2006]).

Solution of reaction networks using the above procedure is by now a mature field. It is in general possible to find the inverse matrix numerically in Eq. (1.8), though problems can arise when time steps are large or when the system nears equilibrium, in which case the matrix $(I + A\Delta t)$ tends to become singular. More generally for numerical solutions, one does not find the inverse matrix but rather solves the matrix equation Eq. (1.7) by usual matrix techniques such as Gaussian elimination or LU decomposition. Sparse matrix solution techniques, such as routines based on Krylov subspaces, permit us to evolve networks with millions of species (or states) routinely (e.g., [Jordan et al. 2005]).

What I seek in this dissertation is not better ways of solving the coupled, stiff differential equations governing the reaction network. Rather I seek a better understanding of the results. In particular, I seek a better understanding of the reaction flows. Apart from providing insight into the reaction mechanisms in the network and the underlying statistical mechanics, such understanding is essential for determining which reaction rates govern the flows and hence are important for either experimental study or more detailed theoretical evaluation. For example, reaction sensitivity studies for alpha-rich freezeouts in supernovae [The et al. 1998; Jordan et al. 2003] have already motivated nuclear physics experiments to determine key [Sonzogni et al. 2000; Vockenhuber et al. 2007] to determine key reaction cross sections that were previously only estimated theoretically. Experimental campaigns to measure reaction cross sections are costly and require solid justification for why they affect final abundance results. Such justification comes from a clear understanding of the relevant reaction's governing role in the reaction flows.

Reaction flows are straightforward to understand when they are unidirectional. In many problems of interest, however, forward and reverse flows are comparable, and complicated cycles result. For example, for the simple network $i \leftrightarrow j$, we see that if Δt is small enough, then

$$\Delta X_i = \left(-\lambda_{ij} X_i(t) + \lambda_{ji} X_j(t)\right) \Delta t. \tag{1.11}$$

Thus, the change in the abundance or probability of i is the net flow, that is, the flow in $(\lambda_{ji} X_j(t))$ minus the flow out $(\lambda_{ij} X_i(t))$, times the timestep Δt. Since X_i and X_j are typically changing more slowly than the timescales from either the forward or reverse flow

individually, however, we can expect flows to cycle many times in time Δt. The solution to eq. (1.8) of course handles this, but we would like to understanding this cycling explicitly.

One approach to understanding the behavior of reaction networks is to study the equilibria that arise. Given enough time, the reaction networks considered here establish a complete equilibrium in which all forward and reverse flows come into balance. This is a condition of maximum entropy or minimum free energy. If the external variables like temperature and density change on a timescale faster than the time required to achieve equilibrium, however, the system will not reach or maintain the complete equilibrium. Parts of the system, however, may reach an equilibrium. For example, in stellar nucleosynthesis in matter expanding from high temperature and density, the three-body reactions that assemble ^{4}He into heavier species are much slower than the reactions that convert heavy nuclei into each other. This means that the heavy nuclei can be in equilibrium under exchange of neutrons, protons, and alpha particles, but the overall abundance of heavy nuclei is not what one would find in a full equilibrium. In effect, the system is in equilibrium but now there is an extra constraint present on the number of heavy nuclei. [Meyer et al. 1998] showed that such constraints could be treated via the method of Lagrange multipliers. The Lagrange multiplier turns out to be a chemical potential associated with the heavy nuclei as a whole. This picture has proven crucial for understanding the stellar nucleosynthesis of ^{48}Ca, which is always underproduced relative to ^{66}Zn in the full equilibrium but can easily be understood as a product of a constrained equilibrium [Meyer et al. 1996].

The idea of constrained equilibria has turned out to be very useful for understanding basic features of explosive nucleosynthesis. In particular, one views explosive nucleosynthesis as a "descent of the hierarchy of statistical equilibria" (see contribution of Meyer to [Wallerstein et al. 1997]). The focus is on understanding the evolution of equilibrium clusters and the constraints that appear as time proceeds during the expansion and cooling of matter in an explosive environment. The movies at

http://www.webnucleo.org/home/movies/alpha_rich/

demonstrate this quite clearly.

While the idea of constrained equilibria has proven quite valuable for understanding reaction networks, we seek a more comprehensive framework for understanding the time evolution and flows in a reaction network. This more comprehensive framework should encompass both the equilibrium and unidirectional flows pictures. Branchings in directed graphs provide us with this framework in a straightforward, intuitive way that will greatly clarify the role of particular reactions in complex flows.

CHAPTER 2

REACTION NETWORKS AND DIGRAPHS

From Eq. (1.8) we note $P(t, \Delta t) \equiv (I + A\Delta t)^{-1}$. P is thus a transition matrix that acts on $X(t)$, the abundance or probability vector at t, to give $X(t + \Delta t)$, the abundance or probability vector at the new time $t + \Delta t$. Because $X_i(t + \Delta t) = \sum_j P_{ij}(t, \Delta t)X_j(t)$, it is clear that $P_{ij}(t, \Delta t)$ is the probability that state or species j at t contributes to state or species i at $t + \Delta t$. In other words, $P_{ij}(t, \Delta t)$ gives the net flow from species j to i over time step Δt. For our linear networks with time-independent rates, P is a right stochastic matrix and the evolution of our network is a finite state space Markov chain. For our networks with time-varying rates, we must restrict the evolution to time step Δt before again updating the rates.

To determine the individual matrix elements P_{ij}, we represent our network as a directed graph. A graph $G = (V, E)$ is a set V of vertices and a set of edges E, which are two-element subsets of V. An edge is thus a line (segment) connecting two vertices. A directed graph (digraph) $D = (V, A)$ is a set V of vertices and a set A of arcs, which are ordered pairs of vertices. In particular, an arc $a = (i, j)$ is an arrow directed from vertex i to vertex j, where i and j are both elements of V. For our problem, vertices are states or levels in our network. Arcs are related to reactions connecting those states or levels.

Given a network with n states or species, we draw a graph with n vertices. Each vertex is labeled with an index i that corresponds to the state or species number in the network and ranges from 1 to n. If a particular reaction r in the network connects state or species i to state or species j, we draw an arc in our graph from vertex j to vertex i and give the arc weight $\ln\left(\lambda_{ij}^{(r)}\Delta t\right)$. Note that the direction of the arc is in the reverse of the flow in time. This is because the arcs will represent the contribution to j *from* i in time Δt. For non-linear networks, it is important to note that there can be multiple arcs per reaction.

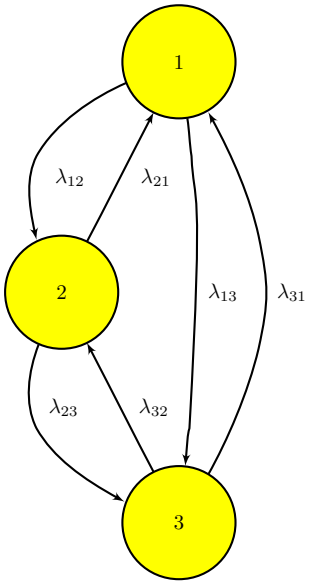

Figure 2.1 A simple 3 state network.

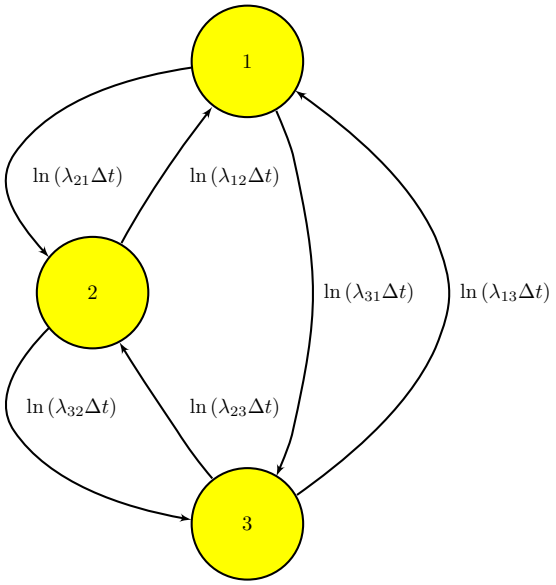

Figure 2.2 A simple 3 state network represented as a graph with arc weights. Note that the direction of the arcs has been reversed.

We now assume that we have drawn all arcs from all reactions. We denote the total number of arcs as m and a particular arc k as e_k. The out, or source, vertex of arc k is denoted $s(e_k)$, and the in, or target, vertex of arc k is $t(e_k)$. We denote the weight of arc k as $w(e_k)$. With these definitions, we note from Eqs. (1.2) and (1.4) that

$$(A\Delta t)_{ii} = \sum_k \delta_{t(e_k),i} \exp(w(e_k)), \tag{2.1}$$

where the sum runs over all arcs but the Krönecker delta picks out only those arcs with species or state i as the target. Similarly, from Eq. (1.5), we find

$$(A\Delta t)_{ij} = -\sum_k \delta_{t(e_k),j}\delta_{s(e_k),i} \exp(w(e_k)), \tag{2.2}$$

where, in this case, the sum runs over all arcs with species or state i as the source and species or state j as the target.

We may now accommodate the identity matrix in Eq. (1.7) in the following way. We extend our graph by including a fictitious vertex with label 0, and we draw a single arc from vertex 0 to each vertex i and give each of those n arcs weight zero. If we include those arcs in Eq. (2.1), the effect is to add unity to each of the diagonal terms in the matrix for $i = 1$ to n. We thus extend our matrix to include a row and column with index 0 and write

$$(I + A\Delta t)_{ii} = \sum_k \delta_{t(e_k),i} \exp(w(e_k)) \tag{2.3}$$

and

$$(I + A\Delta t)_{ij} = -\sum_k \delta_{t(e_k),j}\delta_{s(e_k),i} \exp(w(e_k)). \tag{2.4}$$

It is useful to note that $(I + A\Delta t)_{i0} = 0$ since no arc has vertex 0 as its target and that $(I + A\Delta t)_{0i} = -1$ since there is only one arc from vertex 0 to vertex i. While the extended matrix $(I + A\Delta t)$ in Eqs. (2.3) and (2.4) now has $n + 1$ rows and columns, we can return to the matrix representing our network by striking the first row and column, that is, by striking row 0 and column 0.

We may now write the matrix $I + A\Delta t$ as the product of an incidence matrix M and a weight matrix W. An incidence matrix M is a matrix with n rows and m columns, where n is the number of vertices in a digraph and m is the number of arcs. The elements

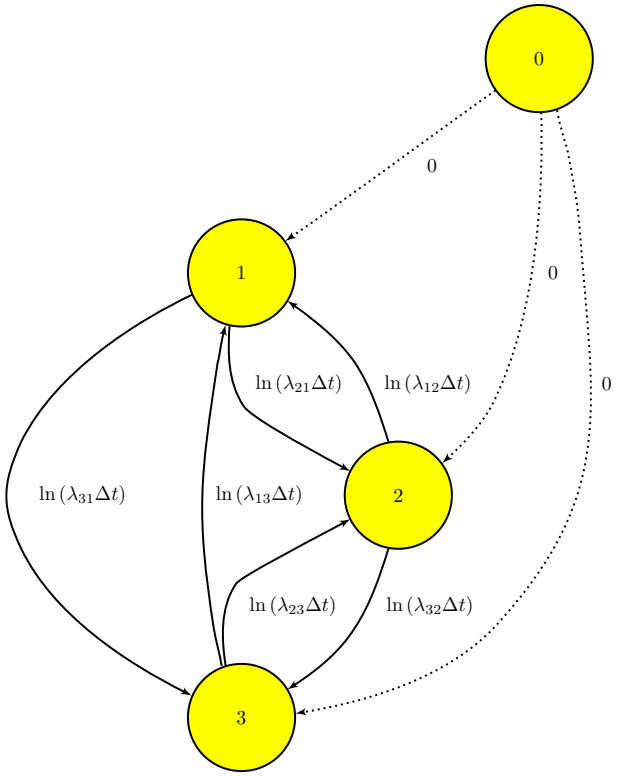

Figure 2.3 A simple 3 state network represented as a graph with arc weights. This graph now includes the fictitious vertex 0 with accompanying arcs with weight 0 to account for the identity submatrix part of the full matrix problem for the network.

of M are

$$M_{i,k} = \delta_{t(e_k),i} - \delta_{s(e_k),i}, \tag{2.5}$$

where $\delta_{i,j}$ is the usual Krönecker delta. For a column k, then, there is a -1 in the row corresponding to the source vertex of the arc e_k and a 1 in the row corresponding to the target vertex of arc e_k. The weight matrix W has m rows and n columns. The elements of W are

$$W_{k,j} = \delta_{t(e_k),j} \exp(w(e_k)) \tag{2.6}$$

In Eq. (2.6), the k-th row corresponds to the k-th arc in the graph.

For our network with n states or species plus the fictitious vertex 0, we now consider the $(n+1) \times (n+1)$ matrix MW. The (i,j) element of this matrix is

$$(MW)_{i,j} = \sum_k M_{i,k} W_{k,j} = \sum_k \delta_{t(e_k),i} \delta_{t(e_k),j} \exp(w(e_k)) - \sum_k \delta_{s(e_k),i} \delta_{t(e_k),j} \exp(w(e_k)). \tag{2.7}$$

If $i = j$, the second term in the sum in Eq. (2.7) is zero since our digraph contains no loops and, hence, the source and target of any arc must be distinct. In this case,

$$(MW)_{i,i} = \sum_k \delta_{t(e_k),i} \exp(w(e_k)). \tag{2.8}$$

If $i \neq j$, the first term in the sum in Eq. (2.7) is zero since an arc cannot have two distinct targets. In this case,

$$(MW)_{i,j} = -\sum_k \delta_{s(e_k),i} \delta_{t(e_k),j} \exp(w(e_k)). \tag{2.9}$$

Comparison of Eqs. (2.3) and (2.8) and Eqs. (2.4) and (2.9) show that

$$I + A\Delta t = MW. \tag{2.10}$$

This is true for the case of the $(n+1) \times (n+1)$ extended matrix $I + A\Delta t$ that includes row and column 0. It is also true for the case of the $n \times n$ matrix that does not include a row 0 and column 0 if we imagine striking row 0 of M and column 0 of W.

CHAPTER 3

THE INVERSE MATRIX

In this chapter I demonstrate how to compute individual elements of the matrix $P(t, \Delta t)$ from branchings in the digraph representing our reaction network. For a directed graph D, the indegree of any vertex is the number of arcs going into that vertex while the outdegree is the number of arcs leaving that vertex. A branching B on a graph is then an acyclic subgraph of D that has no vertex with indegree larger than one. If a digraph has n vertices, a spanning branching, or arborescence, has $n-1$ arcs. The underlying graph is a tree (an acyclic connected graph). A general branching has fewer than $n-1$ arcs. Its underlying graph is a forest. The weight $w(B)$ of a branching B is the sum of the weights of its arcs.

The transition matrix $P(t, \Delta t)$ for our network is the inverse of the matrix $I + A\Delta t$ (Eq. [1.8]). The (i, j) element of $(I + A\Delta t)^{-1}$ is given by the (j, i) cofactor of $I+A\Delta t$ divided by the determinant of $I + A\Delta t$. The (j, i) cofactor of a square matrix is the determinant of the submatrix obtained by removing the j-th row and i-th column multiplied by the factor $(-1)^{i+j}$.

Consider first the determinant of $I + A\Delta t$, denoted $det(I + A\Delta t)$. From Eq. (2.10), we may write

$$det(I + A\Delta t) = det(MW). \tag{3.1}$$

To calculate $det(MW)$, we use the Cauchy-Binet formula, which states that, if M is an $n \times m$ matrix and W is an $m \times n$ matrix,

$$det(MW) = \sum_S det(M_S)det(W_S), \tag{3.2}$$

where the sum runs over all subsets S of $\{1, ..., m\}$ with n elements. There are $C(n, m)$ such subsets S, where $C(n, m)$ is the usual binomial coefficient.

In computing $det(MW)$, we are computing the case for the digraph with n real vertices (no fictitious vertex 0). Nevertheless, it is useful to add the fictitious vertex 0 and

then imagine striking the first row of M (and therefore M_S) and the first column of W_S (and therefore W_S) before computing the determinants. We also, without loss of generality, imagine that M and W are sorted by the index of their invertices. There are no arcs into fictitious vertex 0; thus, the first ℓ_1 columns of M correspond to the ℓ_1 arcs that have vertex 1 as the invertex (and thus have a 1 in row 1). The first ℓ_1 rows of W thus have entries (the values $\exp(w(e_k))$) in column 1. The ℓ_1 columns in M are then followed by ℓ_2 columns in M that correspond to the ℓ_2 arcs that have vertex 2 as the invertex, and the ℓ_1 rows in W are followed by ℓ_2 rows in W with entries in column 2. This sorting proceeds until all arcs are accounted for.

We now consider the submatrices M_S and W_S. Consider first M_S. Before striking row 0, M_S is an $(n+1) \times n$ incidence matrix. It corresponds to an n-arc subgraph of the full digraph. W_S is an $n \times (n+1)$ weight matrix that has the same rows as the columns in M_S. The first column is all zeros, but, because of our sorted arrangement of the rows of W, there is one entry per row and column number of the non-zero element in each row is larger than or equal to that in the previous row. As a consequence, if any two rows in W_S have the same column number for their non-zero elements, there must be a zero in the diagonal element of one of the rows. This means that one of the columns in W_S must contain all zeros and, after striking column 0 in the $n \times (n+1)$ version of W_S, $det(W_S) = 0$. Only subgraphs of our extended digraph that have indegree equal to one for each vertex other than 0 contribute to $det(MW)$. Moreover, because of the sorted arrangement of the arcs, the contributing submatrices W_S will be diagonal; hence, for a given subset S,

$$det(W_S) = \exp\left(\sum_{k \in S} w(e_k)\right). \tag{3.3}$$

We now return to our $(n+1) \times n$ version of M_S. In appendix A I demonstrate that addition of two distinct columns in a square matrix leaves the determinant unchanged. The columns in M_S correspond to a particular subset S of arcs in the network digraph. Each arc either has vertex 0 as a source or does not. Consider a column in M_S that corresponds to an arc k_1 such that $s(e_{k_1}) \neq 0$. Since we have previously shown that each vertex other than 0 has indegree exactly one, there must be another arc k_2 in the subgraph

with $t(e_{k_2}) = s(e_{k_1})$. Add the column corresponding to arc k_2 to that corresponding to arc k_1. The column corresponding to arc k_1 now has -1 in row $s(e_{k_2})$ and 1 in row $t(e_{k_1})$ unless $s(e_{k_2}) = t(e_{k_1})$, in which case the column corresponding to arc k_1 now has all zeros and, after striking row 0 in M_S, $det(M_S) = 0$. Because $s(e_{k_2}) = t(e_{k_1})$, arcs k_1 and k_2 form a cycle; thus, the subgraph corresponding to subset S must be acyclic to contribute to $det(MW)$.

If $s(e_{k_2}) \neq 0$, we may repeat the above procedure by finding the arc k_3 whose target is the source of k_2. In this way we regress back until column operations have converted the column corresponding to arc k_1 into one in which there is -1 in row 0 and 1 in row $t(e_{k_1})$. If at any stage of the regression a cycle forms, the column corresponding to arc k_1 will have all zeros and, after striking row 0, $det(M_S) = 0$. We repeat this procedure for all columns that correspond to arcs whose source is not vertex 0. If no cycles appear, the resulting incidence matrix will have -1 in each column of row 0 and, because of the sorted arrangement of the arcs, 1 in the $(i+1, i)$ element.

The above regression procedure thus produces a new incidence matrix that has a single arc from vertex 0 to each of the other vertices, if the subgraph is acyclic. Otherwise $det(M_S) = 0$. In other words, the subgraph will only contribute to $det(MW)$ if there is an n-arc path from fictitious vertex 0 to each of the other vertices. The new incidence matrix is that for an arborescence of the extended graph in which all arcs have vertex 0 as the source. We call this graph the root graph of S. In general, an arc in this graph from vertex i to vertex j means that there is a path from vertex i to vertex j in the parent graph. In our particular case, we see that only subgraphs that have a path from vertex 0 to each of the other vertices i contribute to the overall determinant.

If we now strike row 0 from the $(n+1) \times n$ version of M_S, we are left with an $n \times n$ identity matrix. The determinant is unity. In this way, we find that

$$det(M_S)det(W_S) = \exp\left(\sum_{k \in S} w(e_k)\right) \tag{3.4}$$

if the n arcs in subset S form an acyclic subgraph of our $n+1$ digraph (that includes fictitious vertex 0) with indegree equal to zero for vertex 0 and indegree equal to one for all

other vertices. Thus, the n arcs in S form an arborescence of the digraph rooted at vertex 0. From Eq. (3.1), we thus see that

$$det(I + A\Delta t) = \sum_{S'} \exp\left(\sum_{k \in S'} w(e_k)\right) \tag{3.5}$$

where S' is a subset of arcs in the digraph that form an arborescence rooted at vertex 0.

If we now imagine removing fictitious vertex 0 and all arcs with vertex 0 as their source from the digraph, we see that the contributing subgraphs are branchings. A succinct expression of our result then is that

$$det(I + A\Delta t) = \sum_{B} \exp\left(w(B)\right), \tag{3.6}$$

where B is any branching in our digraph.

We now consider the cofactors of $I + A\Delta t$. To compute the (i, j) cofactor, we again use the Cauchy-Binet formula in Eq. (3.2). Here we are striking row j and column i in $I + A\Delta t$ (as well as row 0 and column 0 in the extended version of the matrix that includes fictitious vertex 0). Since the (j, i) minor submatrix obtained in this way has $n-1$ rows and columns, we are considering subsets S of arcs $\{1, ..., m\}$ containing $n - 1$ elements. Thus, to obtain the relevant matrix M_S, we find the $(n + 1) \times (n - 1)$ matrices that contain the set S columns of M and then strike row 0 and row j. Similarly, to compute the relevant matrix W_S, we find the $(n - 1) \times (n + 1)$ matrices that contain the set S rows of W and then strike column 0 and column i.

Consider first W_S. Since we strike column i, any subset S that includes an arc with vertex i as a target will yield a matrix W_S with all zeros in the row corresponding to that arc. As a result, only subgraphs that have indegree equal to zero for vertex i will contribute to the (i, j) cofactor. As before, the indegree of all other vertices (except vertex 0) must be one.

Consider now M_S. The relevant subgraph obtained from our subset S of arcs will have $n - 1$ total arcs, none of which has indegree larger than unity. This means that one vertex in addition to vertex 0 has indegree equal to zero. That vertex, by the previous considerations of W_S, must be vertex i. We now perform column operations. The regression

traceback will thus produce a root graph in which each vertex is connected to vertex 0 or to vertex i. As before, cycles will result in a matrix with all zeros in a column and thus $det(M_S) = 0$. The resulting incidence matrix for a contributing subgraph will have in each column -1 in either row 0 or row i. It will also have 1 in each column with the row number of that element increasing for increasing column number. Since the indegrees of vertex 0 and i are zero, however, rows 0 and i do not have 1.

We now consider striking rows 0 and j. Suppose first that $j = i$. In this case we strike the only rows that contain -1's. Because of the sorted arrangement of the arcs, the resulting matrix is the $(n-1) \times (n-1)$ identity matrix. Since $(-1)^{i+j} = 1$ for $i = j$, $det(M_S) = 1$ and the (i,i) cofactor is the weighted sum of all branchings that have indegree equal to zero for vertex i. The (i,i) element of the inverse matrix is this cofactor divided by the determinant of $I + A\Delta t$; thus,

$$P(t, \Delta t)_{ii} = \frac{\sum_{B:(i)} \exp(w(B))}{\sum_{B} \exp(w(B))}, \tag{3.7}$$

where $B : (i)$ is any branching rooted at vertex i; that is, has indegree equal to zero for vertex i.

Suppose now that $j \neq i$. First suppose that the column of the modified M_S that has 1 in row j has -1 in row 0. The subgraph composed of arcs in S then has a path from vertex 0 to vertex j. When we strike row 0 and row j, the resulting matrix has all zeros in this column and $det(M_S) = 0$. This means that there must be a -1 in row i of this column; hence, subgraphs obtained from subsets S only contribute to the (i,j) cofactor if there is a path from vertex i to vertex j.

Once we have struck rows 0 and j, we have an $(n-1) \times (n-1)$ matrix. The column in this matrix corresponding to the arc whose target is vertex j has -1 in row i and zeros in the other rows. We may compute the determinant by performing the Laplace expansion. We strike row i and this column. The resulting matrix is an $(n-2) \times (n-2)$ identity matrix since any -1's in the initial matrix are in row i. The determinant of this minor is thus unity. This is the only non-zero element in the column, so the determinant is the appropriate factor of -1. We have -1 as the factor we are expanding about. We now consider the row and column of this factor -1. Suppose first that $j < i$. In this case, the

-1 we are expanding about is in column j. Since we struck row j, the row is $i-1$; thus, we have an additional factor $(-1)^{i-1+j}$ so that $det(M_S) = (-1)^{i+j}$. If instead $j > i$, we are expanding about row i. Row j is $j-i$ units below row i; thus, the column is $i+j-i-1$, where the -1 accounts for the offset of the 1 from the diagonal in our sorted arrangement of arcs. We thus have an additional factor $(-1)^{i+j-1}$ and again $det(M_S) = (-1)^{i+j}$. The (i,j) cofactor of $I + A\Delta t$, denoted C_{ij} is thus for $i \neq j$

$$C_{ij} = (-1)^{i+j} \sum_S (-1)^{i+j} \exp\left(\sum_{k\in S} w(e_k)\right) = \sum_S \exp\left(\sum_{k\in S} w(e_k)\right), \tag{3.8}$$

where S is a subset of arcs that give a branching of the overall graph with indegree equal to zero for vertex i and a path from vertex i to vertex j. The (i,j) element of the inverse matrix is this cofactor divided by the determinant of $I + A\Delta t$; thus,

$$P(t,\Delta t)_{i,j\neq i} = \frac{\sum_{B:(i)\to j} \exp\left(w(B)\right)}{\sum_B \exp\left(w(B)\right)}, \tag{3.9}$$

where $B : (i) \to j$ is any branching rooted at vertex i that includes a path from vertex i to vertex j.

In computing the cofactors C_{ij}, we have connected them with branchings in our extended graph that are rooted at vertex 0 and vertex i. Such branchings contain $n-1$ arcs. Any such branching may be viewed as one of the n arc arborescences of the extended graph rooted at vertex 0 with the arc from vertex 0 to vertex i removed. Since the weight of this arc is zero, it does not contribute to the weight of the branching. Thus, any n arc arborescence rooted at vertex 0 may be viewed as contributing a branching rooted at vertex i if there is an arc in the arborescence from vertex 0 to vertex i.

CHAPTER 4

ENUMERATIONS

This chapter considers the number of branchings in a network digraph. We begin by considering a complete, simple digraph. A simple digraph is one for which there is at most one arc from a given vertex i to a given vertex j and there are no loops, that is, no arcs from vertex i to itself. In the language of the reaction network, there is only one reaction linking the states or species represented by vertices i and j. A complete digraph is one for which there are two arcs between every pair of vertices i and j, one arc from i to j and one arc from j to i. If there are n vertices in the complete, simple digraph, there are $C(n,2) = n!/\left((n-2)!2!\right)$ pairs of vertices and, hence, $n(n-1)$ arcs.

As we have repeatedly done, we note that branchings in an n vertex graph may be viewed as arborescences in an $n+1$ arc graph rooted at a fictitious vertex 0. To compute the number of branchings present in an n vertex complete, simple digraph, we compute the number of arborescences rooted at vertex 0 in the extended graph. To do this, we extend our graph further and add arcs from each of the vertices $\{1, 2, ..., n\}$ to 0. This makes a complete, simple digraph with $n+1$ vertices. The number of arborescences in a complete, simple digraph with n vertices is easy to compute. We use the well-known result that the number of spanning trees in a complete graph is n^{n-2} [**?**]. Since each vertex in a spanning tree can serve as the root of an arborescence, each spanning tree corresponds to n possible arborescences. The number of arborescences in the complete, simple digraph with n arcs is thus $n \times n^{(n-2)} = n^{(n-1)}$; thus, in our graph with $n+1$ vertices, the number is $(n+1)^n$. By symmetry, the number of these that are rooted at vertex 0 is $(n+1)^n/(n+1) = (n+1)^{(n-1)}$. This is the number of branchings in our n vertex complete, simple network digraph. This becomes staggeringly large even for n relatively small. For example, if $n = 9$, the number of branchings is 10^8. It will clearly not be possible to compute all branchings for large networks.

In general our networks will be sparse, not complete; that is, the number of vertices to which a given vertex will be directly connected in the digraph is a small fraction of the whole. To compute the number of branchings in this case, we assign a weight 0 to each arc in the digraph. Eqs. (3.2) and (3.6) show, then, that the number of branchings N will be

$$N = det(I + M) \tag{4.1}$$

where M is a matrix with element $M_{i,i}$ equal to the outdegree of vertex i and element $-M_{i,i\neq j}$ equal to the number of arcs in the digraph incidence on vertex j from vertex i.

Appendix B shows that the determinant of a sum of two matrices A and B is the sum of determinants of all matrices formed from the permutations of different rows of A and B. Thus, the number of branchings is $det(I)$ plus the sum of determinants of all matrices formed by replacing a row of I by the corresponding row of M plus the sum of determinants of all matrices form by replacing two rows of I by the corresponding rows of M and so forth up to $det(M)$. $det(M) = 0$ since the sum of entries across a column equals zero. Consider now the case in which we have the matrix formed by replacing row i of I by the corresponding row of M. This is the matrix that has diagonal elements equal to 1 for all entries except for row i, which has a value equal to the outdegree of vertex i. The off-diagonal element are all zero except in row i where the element $M_{i,j\neq i}$ is negative the number of the arcs from vertex i to j. We obtain this matrix from a subgraph of our extended digraph in which we remove all arcs with source not equal to vertex 0 or vertex i. We also remove the arc from 0 to i, if present. The branchings obtained from this subgraph, then, are one arc branchings with vertex i as the source. From an expansion by minors along column i, the determinant of this matrix is $M_{i,i}$, the outdegree of vertex i, which is indeed the number of one-arc branchings with vertex i as the source. Since i ranges from 1 to n, the sum of the determinants of all such matrices gives the number of one-arc branchings.

We now consider the matrices formed by replacing two rows of I by the corresponding rows of M. Suppose the two rows are i and j. We obtain this matrix by removing from the extended graph all arcs with source other than vertex 0, i, or j and the arcs from 0 to i and 0 to j, if present. The determinant of this matrix then gives the number of two-arc branchings with vertices i and j as sources. We note that these arcs may or may not form

a two-arc path depending on whether the target of i is j or vice versa or not at all. Since there are n vertices, there are $C(n,2)$ determinants that contribute two-arc branchings.

We repeat this procedure until we get to the matrix M itself. By arguments similar to those above, the determinant of M corresponds to the number of n arc branchings. An $n-1$-arc branching, however, already spans the digraph (without vertex 0). Adding another arc either forms a cycle or leads to a vertex with indegree equal to two. In either case, this is not a branching.

For a complete, simple graph with n vertices, $M_{i,i} = n-1$ and $M_{i,i\neq j} = -1$. The number of zero-arc branchings is $det(I) = 1$. The number of one-arc branchings with arc i as a source vertex is $n-1$. Since there are n choices for i, the number of one-arc branchings is $n(n-1)$, the number of arcs in the graph. For the number of two-arc branchings, the determinant reduces to that of a 2×2 matrix with $(n-1)$ on the diagonals and -1 as the off-diagonal elements. The determinant of this matrix is $n(n-2)$. When we combine with the $C(n,2)$ choices for the two rows, we obtain $(n-1)!n^2/\left((n-3)!2!\right) = C(n-1,2)n^2$.

For the general case with m rows of M replacing the corresponding rows of I, the resulting determinant is that of a $m \times m$ matrix with $n-1$ on the diagonal and -1 elsewhere. We may view this matrix as a sum of a diagonal matrix with n on each diagonal and an $m \times m$ matrix with all elements equal to -1. The determinant is again a sum of determinants of $m \times m$ matrices formed from permutations of the various rows from the two matrices. Only the diagonal matrix with n's on the diagonal and the matrices with n's on the diagonal on all rows except one row which has all -1's contribute because the other permutations will have two identical rows and, thus, zero determinant. The resulting determinant, then, is $n^m - mn^{(m-1)} = n^{(m-1)}(n-m)$ since there are m choices for the row with all -1's. In the overall matrix, there are $n!/\left((n-m)!m!\right)$ ways of taking the m rows in the $n \times n$ matrix. N_m, the number of branchings in the complete, simple graph with m arcs is thus

$$N_m = \frac{n!n^{(m-1)}\,(n-m)}{(n-m)!m!} = \frac{(n-1)!n^m}{(n-m-1)!m!} = C(n-1,m)n^m. \tag{4.2}$$

The total number of branchings in the complete graph is thus

$$N = \sum_{m=0}^{n-1} C(n-1, m) n^m = (1+n)^{(n-1)}, \tag{4.3}$$

the expected result.

As a final consideration on complete, simple graphs, consider the fraction of all branchings that are arborescences. For our n vertex complete, simple digraph, there are $n^{(n-1)}$ arborescences and $(n+1)^{(n-1)}$ branchings. The fraction of branchings that are arborescences is thus

$$\frac{N_{(n-1)}}{N} = \left(\frac{1}{1+1/n}\right)^{(n-1)}. \tag{4.4}$$

It is interesting that, in the limit that $n \to \infty$, $N_{(n-1)}/N \to 1/e$.

More generally, for a graph that is not simple and complete, one must work out the determinants for the particular case at hand.

CHAPTER 5

PROBABILITIES

The probabilities at $t + \Delta t$ are given by Eq. (1.6); thus,

$$X_i(t+\Delta t) = \sum_{j=1}^{n} P_{ij} X_j(t) = \frac{\sum_{B:(i)} \exp\{w(B)\} X_i(t)}{\sum_B \exp\{w(B)\}} + \frac{\sum_{j\neq i} \sum_{B:(i)\to j} \exp\{w(B)\} X_j(t)}{\sum_B \exp\{w(B)\}}. \tag{5.1}$$

We now consider a branching $B : (i) \to j$. This branching is rooted at i but includes a path from i to j. As argued previously, we can view this branching as derived from the n arc arborescence with the arc from vertex 0 to vertex i removed. With this observation, we interpret the branching $B : (i) \to i$ to be any branching that, when fictitious vertex 0 is added, contains an arc from vertex $0 \to i$. This allows us to write Eq. (5.1) as

$$X_i(t+\Delta t) = \frac{\sum_j \sum_{B:(i)\to j} \exp\{w(B)\} X_j(t)}{\sum_B \exp\{w(B)\}}, \tag{5.2}$$

where the sum on j in the numerator is no longer restricted to $j \neq i$.

From Eq. (5.2), the probabilities at time $t + \Delta t$ may be derived by the following algorithm. First, construct the extended graph that includes vertex 0 and has an arc with weight zero from vertex 0 to each vertex i. Construct all n arc arborescences rooted at vertex i. For each arborescence B, calculate its weight $w(B)$ and add it to the sum $\sum_B \exp\{w(B)\}$. Then, if the arborescence has an arc from vertex 0 to vertex i, add to the numerator for $X_i(t + \Delta t)$ the term $\exp\{w(B_k)\} \sum_{j\in C} X_j(t)$, where C is the subset of vertices connected to i, that is, for every vertex in C, there is a path from i to that vertex. We note that C includes i. If the arborescence has an arc from vertex 0 to m but no arc out of m, vertex m is isolated in the underlying branching. In such a case the branching contributes a term $\exp\{w(B)\} X_m(t)$ to $X_m(t + \Delta t)$.

A full implementation of the above algorithm is not practical since the number of branchings in even a modest digraph will be staggeringly large. On the other hand, we note the following property. Each arborescence (or underlying branching) occurs once in the denominator. Now consider the sum $\sum_{i=1}^{n} X_i(t+\Delta t)$. Because every vertex other than 0 has

indegree one in each arborescence, a particular branching B contributes $\exp\{w(B)\}\sum_{j=1}^{n} X_j(t) = \exp\{w(B)\}$ to the numerator. As a result, the sum over the $X_i(t+\Delta t)$ for any number of branchings included will remain unity. A modification of the above algorithm, then, is to compute the largest weight branching (B_{max}), include its contribution to the $X_i(t+\Delta t)'s$, then include the contribution of the second largest branching, and so forth, until the probabilities converge to desired accuracy. The algorithm is then practical if the number of branchings needed for convergence is a sufficiently small fraction of the total number of branchings.

It is worth noting that the largest weight (optimum) branching B_{max} may be determined from Edmonds' algorithm [Edmonds 1967]. This permits us to modify Eq. (5.2) to read

$$X_i(t+\Delta t) = \frac{\sum_j \sum_{B:(i)\to j} \exp\{w(B) - w(B_{max})\} X_j(t)}{\sum_B \exp\{w(B) - w(B_{max})\}}, \tag{5.3}$$

which keeps the exponentials small and computationally manageable as we add each successive one to the sum. It is convenient to write Eq. (5.3) as

$$X_i(t+\Delta t) = \frac{\sum_j \sum_{B:(i)\to j} \exp\{\Delta(B)\} X_j(t)}{\sum_B \exp\{\Delta(B)\}}, \tag{5.4}$$

where

$$\Delta(B) \equiv w(B) - w(B_{max}). \tag{5.5}$$

An algorithm with time complexity $\mathcal{O}(km \log n)$ and memory complexity $\mathcal{O}(k+m)$ is available for finding the k-th optimal arborescences of a digraph with n vertices and m arcs [Camerini et al. 1980].

It becomes convenient for later discussion to write Eq. (5.2) as follows:

$$X_i(t+\Delta t) = \frac{\sum_B \exp\{w(B)\} X_B^{(i)}(t)}{\sum_B \exp\{w(B)\}} \tag{5.6}$$

where we define

$$X_B^{(i)}(t) = \sum_{j:\{(i)\to j\}_B} X_j(t), \tag{5.7}$$

where the symbol $j : \{(i) \to j\}_B$ means any vertex j for which there is a path from root i to j in branching B. We emphasize that, if the branching B is not rooted at i, then we must consider $X_B^{(i)}(t) = 0$.

It is clear that

$$\sum_i X_B^{(i)}(t) = 1 \tag{5.8}$$

because, for any given branching, a vertex is connected by a path to one and only one root of the branching; thus, the sum in Eq. (5.8) is simply the sum of the probabilities of each vertex at time t. We then easily reconfirm that

$$\sum_i X_i(t + \Delta t) = \frac{\sum_B \exp\{w(B)\} \sum_i X_B^{(i)}(t)}{\sum_B \exp\{w(B)\}} = 1; \tag{5.9}$$

that is, the sum of the probabilities at later time is unity independent of the number of branchings included in the sum (as long as at least one branching is included).

CHAPTER 6

EFFECTIVE RATES

Eq. (1.1) gives the rate of change of state i. If we finite difference in time, we find

$$\frac{dX_i}{dt} \to \frac{X_i(t+\Delta t) - X_i(t)}{\Delta t}. \tag{6.1}$$

From Eq. (5.1), we thus find

$$\frac{X_i(t+\Delta t) - X_i(t)}{\Delta t} = -\frac{1}{\Delta t}\frac{\sum_{B:\overline{(i)}} \exp\{w(B)\}}{\sum_B \exp\{w(B)\}} X_i(t) + \frac{1}{\Delta t}\sum_{j\neq i} \frac{\sum_{B:(i)\to j} \exp\{w(B)\}}{\sum_B \exp\{w(B)\}} X_j(t), \tag{6.2}$$

where $B : \overline{(i)}$ is any branching *not* rooted at i. From Eqs. (1.1) and (6.2), we thus infer the effective transition rate from j to i:

$$\lambda_{ji}^{eff} = \frac{1}{\Delta t}\frac{\sum_{B:(i)\to j} \exp\{w(B)\}}{\sum_B \exp\{w(B)\}}. \tag{6.3}$$

The total effective destruction rate of i is

$$\Lambda_i^{eff} = \sum_{j\neq i} \lambda_{ij}^{eff} = \frac{1}{\Delta t}\sum_{j\neq i} \frac{\sum_{B:(j)\to i} \exp\{w(B)\}}{\sum_B \exp\{w(B)\}}. \tag{6.4}$$

Because the sum in Eq. (6.4) is over all branchings not rooted at i, this becomes

$$\Lambda_i^{eff} = \frac{1}{\Delta t}\frac{\sum_{B:\overline{(i)}} \exp\{w(B)\}}{\sum_B \exp\{w(B)\}}, \tag{6.5}$$

as in Eq. (6.2).

CHAPTER 7

LONG-TIME BEHAVIOR OF NETWORKS

In this chapter, I consider the long-time behavior of the network, that is, the solutions for the probabilities when $\Delta t \to \infty$. Because the arc weights for our graph are given by terms of the form $\ln(\lambda \Delta t)$, for sufficiently large Δt, all arc weights will be positive. The largest weight branchings, then, are ones that span the network. Spanning branchings, or arborescences, of an n vertex digraph have $n-1$ arcs; thus, spanning branchings will have a weight given by some number plus $(n-1)\ln \Delta t$.

We begin by considering an undirected graph $G = (V, E)$. A connected component of G is a maximal set of vertices $U \subseteq V$, such that for every pair of vertices i and j in U, there is a path between i and j. The graph G itself is connected if $U = V$, that is, there is a path between each pair of vertices in the graph. Equivalently, we may say that the graph is connected if there is at least one spanning tree in the graph.

We now consider a digraph $D = (V, A)$. A *strongly-connected component* of D is a maximal set of vertices $U \subseteq V$, such that for every pair of vertices i and j in U, we have a path from i to j and a path from j to i. A digraph itself is strongly connected if $U = V$, that is, every vertex in the digraph is reachable from every other vertex.

If our network (more correctly, the digraph representing our network) is not strongly connected, then at least one vertex j is not reachable from at least one other vertex i in the digraph. It is thus clear that we cannot have an arborescence in our network rooted at i. From Eq. (5.6), we see that $X_i(t + \Delta t)$ will be a weighted sum of branchings rooted at i divided by the weighted sum of all branchings. Because the branchings rooted at i do not span the digraph, the maximum weighted sum will be proportional to $\exp\{(n-2)\ln \Delta t\}$ at best. If there are spanning branchings present in the network, however, the denominator in Eq. (5.6) will include terms proportional to $\exp\{(n-1)\ln \Delta t\}$. As $\Delta t \to \infty$, then, it is clear that $X_i(t + \Delta t) \to 0$. We thus see that if the underlying graph of our network is

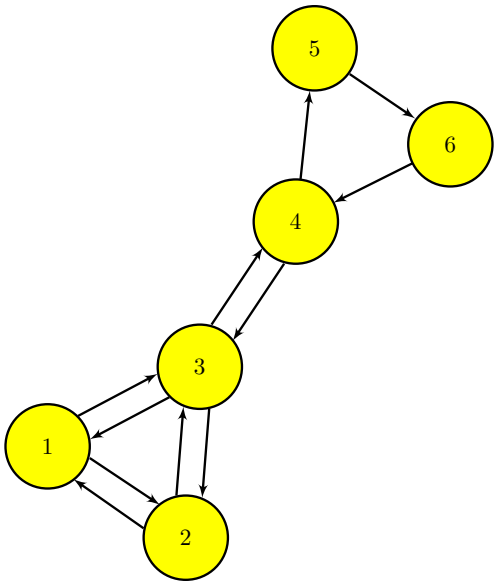

Figure 7.1 A simple, strongly-connected six vertex network. Note that, while the network is strongly connected, it is not reversible.

connected, any vertex that cannot serve as the root of an arborescence will have a probability that goes to zero as $\Delta t \to \infty$.

With this consideration in mind, we restrict our attention to strongly connected networks in which every vertex is reachable from every other vertex. Again from Eq. (5.6) we see that, as $\Delta t \to \infty$, the weighted sums are dominated by the spanning branchings. Furthermore, since these branchings are spanning, there is a path from the single root in the branching to every other vertex; thus,

$$X^{(i)}_{B_{\text{span}}}(t) = 1. \tag{7.1}$$

From Eq. (5.6) we see that

$$X_{i,\infty} = \frac{\sum_{B:(i),\text{span}} \exp\{w(B)\}}{\sum_{B:\text{span}} \exp\{w(B)\}}, \tag{7.2}$$

where the subscript ∞ denotes the limit $\Delta t \to \infty$ and the notation B:span denotes branchings B that span the graph. We note in Eq. (7.2) that the term $\exp\{(n-1)\Delta t\}$ is present in

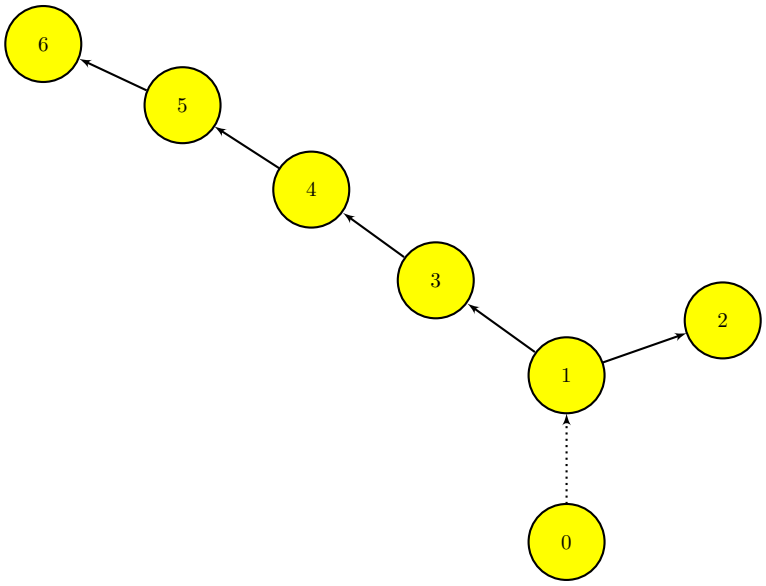

Figure 7.2 A spanning branching, or arborescence, of the six-vertex network in Fig. 7.1 .

both the numerator and denominator and cancels out. We may thus compute the branching weights without the Δt terms.

Eq. (7.2) shows that the long-time solution for the probability of state i in a reaction network will be the weighted sum of arborescences rooted at i divided by the weighted sum of all arborescences in the network. In many cases in physics, this long-time solution represents an equilibrium, especially if the network is strongly connected. The equilibrium probability in a network for the state represented by vertex i in our digraph then is the weighted fraction of branchings rooted at vertex i.

At this point we now consider networks in which the reaction rates derive from quantum mechanics. In such a case, all individual reaction flows are equal and the reaction rate λ_{ji} may be derived by detailed balance from λ_{ij}. In particular, we find

$$\frac{\lambda_{ji}}{\lambda_{ij}} = \frac{X_i^{eq}}{X_j^{eq}}. \tag{7.3}$$

where X_i^{eq} is the equilibrium probability of state i. It is convenient to define

$$R_{ij} \equiv \frac{X_i^{eq}}{X_j^{eq}}. \tag{7.4}$$

When detailed balance holds, we may derive an extremely useful result for the equilibrium probability of a state in the linear network. To do this, we note that the underlying graph of an arborescence is a spanning tree. We further note that an n vertex spanning tree is the underlying graph for n arborescences. For example, consider the spanning tree. Choose a particular vertex i of the tree as the root of the arborescence. Change all edges of the tree incident on i into arcs with vertex i as the out vertex. Consider any vertex j that is the target of the newly created arcs. An edge incident on j but not on i must then be changed into an arc directed out from vertex j. The procedure continues until we reach a leaf of the tree and all edges have been turned into arcs. The result is a branching rooted at vertex i.

We assume now that all arcs in our digraph are reversible, that is, that if there is an arc from vertex i to vertex j, there is also an arc from vertex j to vertex i. If the rates in the network derive from quantum mechanics, the weights of these two arcs are related by detailed balance, as described above. Now consider two branchings that have the same spanning tree as their underlying graph but have different roots. The weights of these branchings are related. Suppose the first branching B_i is rooted at i and the second branching B_j is rooted at j. Since the two branchings derive from the same tree, there is a directed path P_j in B_j from vertex j to vertex i that is the reverse of the directed path P_i in B_i from vertex i to vertex j. To convert B_i into B_j, then, we simply reverse all arcs in P_i to produce P_j. This means that

$$w(B_j) = \ln\left(\frac{\lambda_{ii_1}}{\lambda_{i_1 i}}\right) + \ln\left(\frac{\lambda_{i_1 i_2}}{\lambda_{i_2 i_1}}\right) + ... + \ln\left(\frac{\lambda_{i_k j}}{\lambda_{j i_k}}\right) + w(B_i) = \ln\left(R_{i_1 i} R_{i_2 i_1} ... R_{i_k j}\right) + w(B_i), \tag{7.5}$$

where $i_1, i_2, ..., i_k$ are the intermediate vertices in paths P_i and P_j. Because of Eqs. (7.3) and (7.4) we thus find

$$\exp\{w(B_j)\} = R_{ji} \exp\{w(B_i)\}. \tag{7.6}$$

We now consider $X_{i,\infty}$. Since every spanning branching is rooted at some vertex j, we may write

$$X_{i,\infty} = \frac{\sum_{B:(i),\text{span}} \exp\{w(B)\}}{\sum_j \sum_{B:(j),\text{span}} \exp\{w(B)\}}, \tag{7.7}$$

From Eq. (7.6), we may now write

$$X_{i,\infty} = \frac{\sum_{B:(i),\text{span}} \exp\{w(B)\}}{\sum_j R_{ji} \sum_{B:(i),\text{span}} \exp\{w(B)\}}, \tag{7.8}$$

which becomes

$$X_{i,\infty} = \frac{1}{\sum_j R_{ji}}. \tag{7.9}$$

Eq. (7.9) may be expanded via Eq. (7.4) to read

$$X_{i,\infty} = \frac{X_i^{eq}}{\sum_j X_j^{eq}} = X_i^{eq} \tag{7.10}$$

since the probabilities are normalized. For the case of a strongly-connected, linear, reversible network governed by detailed balance, the long-time probabilities are those in equilibrium, as expected.

We now consider a particular branching $B_T^{(i)}$ derived from a spanning tree T and rooted at vertex i. If we multiply numerator and denominator of Eq. (7.9) by $\exp\{w(B^{(i)})\}$, we find

$$X_{i,\infty} = \frac{\exp\{w(B_T^{(i)})\}}{\sum_j \exp\{w(B_T^{(i)}\} R_{ji}}, \tag{7.11}$$

which, by Eq. (7.6), becomes

$$X_{i,\infty} = \frac{\exp\left\{w(B_T^{(i)})\right\}}{\sum_j \exp\left\{w(B_T^{(j)})\right\}}. \tag{7.12}$$

It is important to note that the branchings $B_T^{(j)}$ are all derived from $B_T^{(i)}$, that is, all branchings in the denominator in Eq. (7.12) are from the same tree. The valuable result is that, for a strongly-connected, reversible linear network governed by detailed balance, we may find the long-time probability of a state by finding any spanning tree T, computing the exponential weight of the arborescence rooted at i derived from T, and dividing this result by the sum of the exponential weights of all arborescences derived from T. Kruskal's algorithm [**?**] for finding the minimum spanning tree has time complexity $\mathcal{O}(m \ln m)$, where m is the

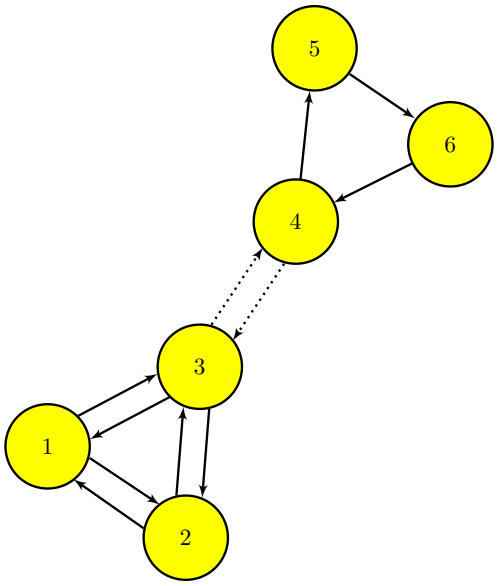

Figure 7.3 The same network as in Fig. 7.1 but now the arcs (3,4) and (4,3) have negligible weights such that the network breaks into two clusters.

number of edges in the tree, while Prim's algorithm [**?**] for finding the minimum spanning tree has time complexity $\mathcal{O}(m \ln n)$. Both algorithms are clearly quite efficient. Alternatively, we may find the maximum weight arborescence in the network and use the underlying graph as T. In any event, such a procedure greatly reduces the computational effort. For example, in a complete, simple network with n vertices, one computes n branchings to find $X_{i,\infty}$ for all i in Eq. (7.12) while one computes $n^{(n-1)}$ branchings in Eq. (7.2).

We now consider a network for which some subset of reactions is too slow to maintain full equilibrium. The simplest scenario to consider is one in which the slow reactions cause the network to break up into two distinct equilibrium clusters. We assume that the two clusters themselves are strongly connected but that there is no arc between the two clusters so that the full network is no longer strongly connected. Fig. 7.3 illustrates such a scenario. In this case, if the arcs between vertices 3 and 4 have negligible weights, the network breaks up into two clusters, one cluster, C_1, comprising vertices 1, 2, and 3 and the other cluster, C_2, comprising vertices 4, 5, and 6. Clusters C_1 and C_2 are themselves strongly connected, but C_1 and C_2 are not connected to each other.

As long as the arc weights remain sufficiently strong within the two clusters (because Δt is sufficiently large), the dominant branchings will separately span the two clusters but will lack arcs between the two clusters. Fig. 7.4 illustrates such a branching for the six-vertex network of Fig. 7.3. The branching, call it B, has two roots, one a vertex in C_1 and the other a vertex in C_2. The weight of this branching is then the weight of the arborescence rooted within cluster C_1 plus the weight of the arborescence rooted within cluster C_2. We may thus write

$$\exp w(B) = \exp\left\{w(B^{(C_1)})\right\} \exp\left\{w(B^{(C_2)})\right\}. \tag{7.13}$$

Since the arborescences within C_1 are independent of those in C_2, it thus clear that, if we neglect arcs whose source is in C_1 and target is in C_2 or whose source is in C_2 and target is in C_1, the exponential sum over branchings is

$$\sum_B \exp\{w(B)\} = \sum_{B_1:[(C_1)]} \sum_{B_2:[(C_2)]} \exp\{w(B_1)\} \exp\{w(B_2)\} \tag{7.14}$$

where $B : [(C_m)]$ indicates a branching B rooted in (as indicated by the parentheses) and completely contained within (as indicated by the square brackets) cluster C_m. Similarly, we will use the notation $B : [(i \in C_m)]$ to denote a branching B rooted at a vertex $i \in C_m$ such that the branching is also completely contained within C_m. The probability of state $i \in C_1$ at $t + \Delta t$ is given by

$$X_i(t + \Delta t) = \frac{\sum_{B_1:[(i\in C_1)]} \exp\{w(B_1)\} X^{(i)}_{B_1} \sum_{B_2:[(C_2)]} \exp\{w(B_2)\}}{\sum_{B_1:[(C_1)]} \exp\{w(B_1)\} \sum_{B_2:[(C_2)]} \exp\{w(B_2)\}} \tag{7.15}$$

from which it is clear that

$$X_i(t + \Delta t) = \frac{\sum_{B:[(i\in C_1)]} \exp\{w(B)\} X^{(i)}_B}{\sum_{B:[(C_1)]} \exp\{w(B)\}}. \tag{7.16}$$

If we further assume that only the branchings $B : [C_1]$ that span C_1 are included because Δt is large enough, then we find

$$X_i(t + \Delta t) = \frac{X_{C_1} \sum_{B:[(i\in C_1)]} \exp\{w(B)\}}{\sum_{B:[(C_1)]} \exp\{w(B)\}} = f^{(i)}_{C_1} X_{C_1} \tag{7.17}$$

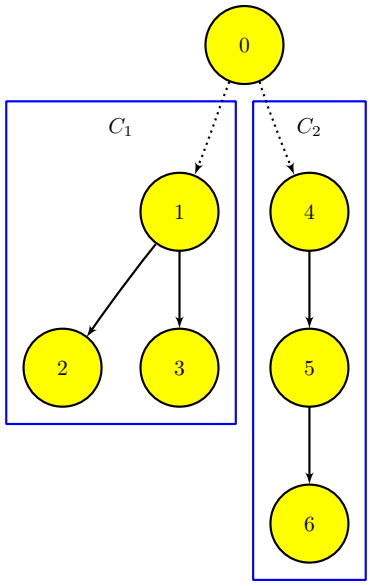

Figure 7.4 A branching for the network in Fig. 7.3.

where in general for state i and cluster C_m

$$X_{C_m} = \sum_{i \in C_m} X_i(t) \tag{7.18}$$

and

$$f_{C_m}^{(i)} = \frac{\sum_{B:[(i \in C_m)]} \exp\{w(B)\}}{\sum_{B:[(C_m)]} \exp\{w(B)\}}. \tag{7.19}$$

In this way, C_m acts as its own equilibrium cluster, and the probability of state i is the fraction $f_{C_m}^{(i)}$ of exponentially weighted branchings in C_m rooted at i multiplied by the total probability of the cluster.

We construct spanning branchings of the full network from the two branchings that span C_1 and C_2. A spanning branching of the full network will be rooted at a single vertex. Suppose $i \in C_1$. The branching will then have a single arc from a vertex in C_1 to a vertex in C_2. Since that arc leaves the vertex in C_1, the source vertex can be any vertex in C_1. On the other hand, the arc goes to a vertex in C_2. As a consequence, the target vertex of the arc must be the root of the spanning branching in C_2. Call that vertex k. In effect, to construct a spanning branching rooted at i of the full network, we remove the arc from the

fictitious vertex 0 to k and add an arc from vertex $j \in C_1$ to $k \in C_2$. The weight of the arc (j,k) is $\ln(\lambda_{kj}\Delta t)$; thus, the exponential weight of the branching B so constructed is

$$\exp\{w(B:(i\in C_1))\} = \exp\{w(B_1:[(i\in C_1)])\}\exp\{w(B_2:[(k\in C_2)])\}\,\lambda_{kj}\Delta t. \quad (7.20)$$

We obtain the sum of all exponential weights of branchings rooted at i by summing over each branching rooted at k and then over all possible values of j and k. The result is

$$\sum_{B:(i)} \exp\{w(B)\} = \sum_{B_1:[(i\in C_1)]} \exp\{w(B_1)\} \sum_{j\in C_1}\sum_{k\in C_2} \lambda_{kj}\Delta t. \sum_{B_2:[(k\in C_2)]} \exp\{w(B_2)\}. \quad (7.21)$$

Finally, to get the sum of all spanning branchings rooted in cluster C_1, we sum over i, which results in a sum over all arborescences rooted in and contained within C_1. Thus,

$$\sum_{B:(C_1)} \exp\{w(B)\} = \sum_{B_1:[(C_1)]} \exp\{w(B_1)\} \sum_{j\in C_1}\sum_{k\in C_2} \lambda_{kj}\Delta t \sum_{B_2:[(k\in C_2)]} \exp\{w(B_2)\}. \quad (7.22)$$

A similar analysis follows for the remaining spanning branchings, which are rooted in C_2. The result is

$$\sum_{B:(C_2)} \exp\{w(B)\} = \sum_{B_2:[(C_2)]} \exp\{w(B_2)\} \sum_{j\in C_1}\sum_{k\in C_2} \lambda_{jk}\Delta t \sum_{B_1:[(j\in C_1]} \exp\{w(B_1)\}. \quad (7.23)$$

From these results and Eq. (7.19) we find

$$\sum_{B:\text{span}} \exp\{w(B)\} =$$

$$\Delta t \sum_{B_1:[(C_1)]} \exp\{w(B_1)\} \sum_{B_2:[(C_2)]} \exp\{w(B_2)\} \sum_{j\in C_1}\sum_{k\in C_2} \left\{\lambda_{jk} f_{C_1}^{(j)} + \lambda_{kj} f_{C_2}^{(k)}\right\}. \quad (7.24)$$

We now compute the effective rate for transition from C_2 to C_1. From §6, and with the assumptions 1) that only $n-2$ and $n-1$ arc branchings dominate and 2) that the $n-2$ arc branchings are those dominated by the union of spanning branchings within C_1 and C_2, we find

$$\lambda_{C_2C_1}^{eff} = \frac{1}{\Delta t} \frac{\sum_{B:(C_1)} \exp\{w(B)\}}{\sum_{B_1:[(C_1)]} \exp\{w(B_1)\} \sum_{B_2:[(C_2)]} \exp\{w(B_2)\} + \sum_{B:span} \exp\{w(B)\}}, \quad (7.25)$$

which becomes

$$\lambda_{C_2C_1}^{eff} = \frac{\sum_{j\in C_1}\sum_{k\in C_2} \lambda_{kj} f_{C_2}^{(k)}}{1 + \Delta t \sum_{j\in C_1}\sum_{k\in C_2} \left\{\lambda_{jk} f_{C_1}^{(j)} + \lambda_{kj} f_{C_2}^{(k)}\right\}}. \quad (7.26)$$

If we define

$$\lambda_{C_m C_n} \equiv \sum_{j \in C_m} \sum_{k \in C_n} \lambda_{kj} f_{C_n}^{(k)}, \tag{7.27}$$

we find

$$\lambda_{C_2 C_1}^{eff} = \frac{\lambda_{C_2 C_1}}{1 + (\lambda_{C_1 C_2} + \lambda_{C_2 C_1})\,\Delta t}, \tag{7.28}$$

which is the effective rate for a two-state system given transition rates $\lambda_{C_1 C_2}$ and $\lambda_{C_2 C_1}$ between the two states C_1 and C_2. Similarly, we may find

$$\lambda_{C_1 C_2}^{eff} = \frac{\lambda_{C_1 C_2}}{1 + (\lambda_{C_1 C_2} + \lambda_{C_2 C_1})\,\Delta t}. \tag{7.29}$$

We have thus reduced the full system to a two-state problem and found the effective transition rates.

We now assume that the reduced two state system thus produced is strongly connected, that is, that neither $\lambda_{C_1 C_2}$ nor $\lambda_{C_2 C_1}$ is zero. In this case, we may consider the ratio

$$\frac{\lambda_{C_1 C_2}^{eff}}{\lambda_{C_2 C_1}^{eff}} = \frac{\lambda_{C_1 C_2}}{\lambda_{C_2 C_1}}. \tag{7.30}$$

We note, however, that this ratio is that of the sum of weighted spanning branchings rooted in C_2 to that in C_1. It is thus clear that

$$\frac{\lambda_{C_1 C_2}}{\lambda_{C_2 C_1}} = \frac{X_{C_2,\infty}}{X_{C_1,\infty}}. \tag{7.31}$$

The two-state linear system automatically satisfies its own detailed balance.

Although the above analysis considered only two strongly-connected components, it holds for any number of strongly-connected components that are not connected to each other. This reduction process then presents a valuable possible algorithm for computing the evolution of the state probabilities. The steps would be the following:

1. Construct the digraph D representing the network and assign weights w to the arcs.
2. Choose a threshold weight (for example, $w = 0$) and construct a copy D' of D that excludes arcs in D whose weight falls below the threshold weight. Arcs with weights above the threshold will change state probabilities on a timescale fast compared to Δt.

3. Find the strongly-connected components of D'. Tarjan's algorithm for finding the strongly-connected components of a digraph has time complexity $\mathcal{O}(n+m)$, where n is the number of vertices and m is the number of arcs [**?**].

4. Treat each strongly-connected component as a cluster C_m and compute the probability for the cluster by Eq. (7.18).

5. Compute the cluster fractions $f_{C_m}^{(i)}$ for each state i by Eq. (7.19). If the rates obey detailed balance, this process is greatly simplified because one may find the cluster fractions for each cluster from arborescences derived from a single spanning tree of the cluster.

6. Compute cluster transition rates from Eq. (7.27). This requires individual rates between species in different clusters, which may not be in D' but are in D.

7. Compute the evolution of the reduced system by treating the clusters as states and using the normal branching rules previously derived.

If the entire digraph is strongly connected, the above algorithm yields the full equilibrium since there will only be one cluster with no transitions; that is, the reduced graph will be the trivial graph (one vertex, no arcs). The cluster probability will be unity. The individual state probabilities can then be determined from Eq. (7.2).

More generally, there will be more than one strongly-connected component. Given time, the clusters will merge. On the other hand, if the individual rates are decreasing with time due, for example, to material expanding and cooling, larger clusters may break up into smaller clusters. Such evolution can continue until the whole network breaks up into n separate clusters. If this continues until one single branching, that with no arcs (or, perhaps more properly, with only arcs from fictitious vertex 0 to each vertex i), dominates, then the network will experience full reaction freezeout. This process of cluster breakup would be the "descent of the hierarchy of statistical equilibria" discussed in §1.

CHAPTER 8

REACTION NETWORK THERMODYNAMICS

In this chapter, I consider network thermodynamics in the context of branchings in digraphs. To begin, it is useful to note that a particle with mass m that obeys Maxwell-Boltzmann statistics typically has a chemical potentials μ given by

$$\mu = mc^2 + kT \ln \left\{ \frac{n}{g} \left(\frac{2\pi\hbar^2}{mkT} \right)^{3/2} \right\}, \tag{8.1}$$

where c is the speed of light in vacuum, k is Boltzmann's constant, n is the number density of the particles, g is a multiplicity (spin factor) for the particle, and $\hbar$ is Planck's constant divided by π. Suppose we are considering atoms of a particular isotope. At temperature T, these atoms will be distributed among their excited states. Let i be a particular state in the atom. The number density of atoms at time t in state i will be $n_i(t) = nX_i(t)$, where $X_i(t)$ is the probability to find an atom in state i at t. The different states will have different masses m_i, which can be related to the ground state mass m_0 by $m_i c^2 = m_0 c^2 + E_i$, where E_i is the excitation energy of state i above the ground state. From this we may find,

$$\frac{\mu_i}{kT} = \frac{E_i}{kT} + \ln \left\{ \frac{X_i}{g_i} \right\} + m_0 c^2 + kT \ln \left\{ n \left(\frac{2\pi\hbar^2}{mkT} \right)^{3/2} \right\}, \tag{8.2}$$

where we neglect the small difference between masses inside the logarithm. At "chemical" equilibrium, all states have the same chemical potential ($\mu_i^{eq} = \mu_j^{eq}$ for all i and j); thus,

$$\frac{X_j^{eq}}{X_i^{eq}} = \frac{g_j}{g_i} \exp \left\{ - \left(E_j - E_i \right) / kT \right\}, \tag{8.3}$$

which is the usual Boltzmann distribution.

In general, the distribution of atoms among their excited states will not be the equilibrium one; thus, it is convenient to consider the distance from equilibrium for a given state i:

$$\frac{\mu_i}{kT} - \frac{\mu_i^{eq}}{kT} = \ln \left(\frac{X_i}{X_i^{eq}} \right). \tag{8.4}$$

Eq. (8.4) shows that, if the probability for state i is bigger than that in equilibrium, $\mu_i > \mu_i^{eq}$, and it is energetically favorable to reduce the probability of state i. The reverse is true if $X_i < X_i^{eq}$. When $X_i = X_i^{eq}$, both sides of Eq. (8.4) are zero, as expected.

All this motivates consideration of the chemical potential as defined in Eq. (8.4). We note that, when states i may be considered as a linear network and when Eq. (8.4) holds, we may find

$$\frac{\mu_i}{kT} - \frac{\mu_i^{eq}}{kT} = \ln\left(\frac{\sum_{B:(i)} \exp\{w(B)\} X_B^{(i)}}{\sum_B \exp\{w(B)\}}\right) - \ln\left(\frac{\sum_{B:(i),\text{span}} \exp\{w(B)\}}{\sum_{B:\text{span}} \exp\{w(B)\}}\right). \tag{8.5}$$

Thus, the chemical potential difference from equilibrium is the logarithm of the ratio of the fraction of the weighted sum of branchings rooted at i compared to the same fraction in equilibrium.

From Eq. (8.5), it makes sense to define μ_i/kT as

$$\frac{\mu_i}{kT} = \ln\left(\frac{\sum_{B:(i)} \exp\{w(B)\} X_B^{(i)}}{\sum_B \exp\{w(B)\}}\right) + \frac{A_i}{kT}, \tag{8.6}$$

where A_i is a function of quantities specific to i (such as the multiplicity) and to the system as a whole (such as the volume). We make the further key assumption that we may write

$$A_i = A + \Gamma_i, \tag{8.7}$$

that is, that we may separate out from A_i a part that depends on i and a part that does not. This holds in may physical situations [recall Eq. (8.2)]. With this definition, we find

$$\frac{\mu_i}{kT} - \frac{\Gamma_i}{kT} = \ln\left(\frac{\sum_{B:(i)} \exp\{w(B)\} X_B^{(i)}}{\sum_B \exp\{w(B)\}}\right) + \frac{A}{kT}, \tag{8.8}$$

Finally, it becomes convenient to define

$$\frac{\mu_i'}{kT} = \frac{\mu_i}{kT} - \frac{A}{kT}. \tag{8.9}$$

We see that μ_i' is a chemical potential with the A part removed. With this definition, we see that

$$\frac{\mu_i'}{kT} - \frac{\Gamma_i}{kT} = \ln\left(\frac{\sum_{B:(i)} \exp\{w(B)\} X_B^{(i)}}{\sum_B \exp\{w(B)\}}\right). \tag{8.10}$$

We compute Γ_i by considering the long-time limit. We then take the difference between μ_i' and μ_j', which goes to zero. As a consequence, we find

$$\Gamma_j - \Gamma_i = kT \ln \left(\frac{\sum_{B:(i),span} \exp\{w(B)\}}{\sum_{B:(j),\text{span}} \exp\{w(B)\}} \right). \tag{8.11}$$

From this we infer

$$\Gamma_i = E_0 - kT \ln \left(\sum_{B:(i),\text{span}} \exp\{w(B)\} \right), \tag{8.12}$$

where E_0 is an overall energy scale. It makes sense to define

$$E_0 = kT \ln \left(\sum_{B:span} \exp\{w(B)\} \right). \tag{8.13}$$

With this definition, Γ_i will be independent of Δt because the spanning branchings all have the same power of Δt which then cancel out. Γ_i is an energy. When the network is strongly connected and reversible, the weighted sum of the branchings rooted at vertex i is proportional to the equilibrium probability. Eq. (8.11) and (8.3) give

$$\Gamma_j - \Gamma_i = kT \ln \left(\frac{g_i}{g_j} \right) + E_j - E_j. \tag{8.14}$$

Thus, in this case, the Γ's are related to the physical energy of the various states. When the network is not reversible, the Γ's are a different energy.

Let us now consider the general quantity

$$\mathcal{Z} = \exp\left\{ \frac{A}{kT} \right\}. \tag{8.15}$$

Since

$$\frac{\sum_B \exp\{w(B)\}}{\sum_B \exp\{w(B)\}} = \frac{\sum_i \sum_{B:(i)} \exp\{w(B)\} X_B^{(i)}}{\sum_B \exp\{w(B)\}} = 1, \tag{8.16}$$

we find

$$\mathcal{Z} = \exp\left\{ \frac{A}{kT} \right\} \frac{\sum_i \sum_{B:(i)} \exp\{w(B)\} X_B^{(i)}}{\sum_B \exp\{w(B)\}}. \tag{8.17}$$

By Eq. (8.8), we thus find

$$\mathcal{Z} = \exp\left\{ \frac{A}{kT} \right\} \sum_i \exp\left\{ \frac{(\mu_i' - \Gamma_i)}{kT} \right\}. \tag{8.18}$$

We compute

$$\Omega = -kT \ln \mathcal{Z} = -\ln A - kT \ln \left(\sum_i \exp \left\{ \frac{(\mu_i' - \Gamma_i)}{kT} \right\} \right). \tag{8.19}$$

At this point, we identify Ω as the grand potential and $\mathcal{Z}$ as the grand partition function. Since A depends only on the system, for example, the temperature T, the set of external variables $\{Y\}$ (such as the volume V, which we will take to be the only external parameter), and possibly the total number chemical potential μ, we thus identify $A = -\Omega_{\text{system}}(T, V, \mu)$. The second term is a grand potential associated simply with the probabilities $\{X_i\}$. In this way it is a "network" grand potential that is not dependent on the external variables:

$$\Omega_{\text{net}} = -kT \ln \left(\sum_i \exp \left\{ \frac{(\mu_i' - \Gamma_i)}{kT} \right\} \right). \tag{8.20}$$

With these definitions

$$\Omega(T, V, \mu, \{\mu_i'\}) = \Omega_{\text{system}}(T, V, \mu) + \Omega_{\text{net}}(T, \{\mu_i'\}). \tag{8.21}$$

A change in the grand potential is given by

$$d\Omega = \left(\frac{\partial \Omega}{\partial T} \right)_{V,\mu,\{X_i\}} dT + \left(\frac{\partial \Omega}{\partial V} \right)_{T,\mu,\{X_i\}} dV + \left(\frac{\partial \Omega}{\partial \mu} \right)_{T,V,\{X_i\}} dN + \sum_i \left(\frac{\partial \Omega}{\partial \mu_i'} \right)_{T,V,\mu,\{X_{j\neq i}\}} dX_i. \tag{8.22}$$

This becomes

$$d\Omega = \left(\frac{\partial \left(\Omega_{\text{system}} + \Omega_{\text{net}} \right)}{\partial T} \right)_{V,\mu,\{X_i\}} dT + \left(\frac{\partial \Omega_{\text{system}}}{\partial V} \right)_{T,\mu,\{X_i\}} dV + \left(\frac{\partial \Omega_{\text{system}}}{\partial \mu} \right)_{T,V,\{X_i\}} d\mu + \sum_i \left(\frac{\partial \Omega_{\text{net}}}{\partial \mu_i'} \right)_{T,V,\mu,\{\mu_{j\neq i}'\}} d\mu_i'. \tag{8.23}$$

From standard thermodynamics, we recall

$$d\Omega = -SdT - PdV - Nd\mu, \tag{8.24}$$

so we recognize the entropy

$$S = -\left(\frac{\partial \left(\Omega_{\text{system}} + \Omega_{\text{net}} \right)}{\partial T} \right)_{V,\mu,\{X_i\}}, \tag{8.25}$$

the pressure

$$P = -\left(\frac{\partial \Omega_{\text{system}}}{\partial V}\right)_{T,\mu,\{X_i\}}, \tag{8.26}$$

and the total number

$$N = -\left(\frac{\partial \Omega_{\text{system}}}{\partial \mu}\right)_{T,V,\{X_i\}}. \tag{8.27}$$

We further identify

$$X_i = \left(\frac{\partial \Omega_{\text{net}}}{\partial \mu_i'}\right)_{T,V,\mu,\{\mu'_{j\neq i}\}}. \tag{8.28}$$

To consider these results, we begin by computing X_i. The result is

$$X_i(t+\Delta t) = \frac{\exp\left\{\frac{(\mu_i'-\Gamma_i)}{kT}\right\}}{\sum_j \exp\left\{\frac{(\mu_j'-\Gamma_j)}{kT}\right\}} = \frac{\sum_{B:(i)} \exp\{w(B)\}\, X_B^{(i)}}{\sum_B \exp\{w(B)\}}, \tag{8.29}$$

where the second equation follows from Eq. (8.10). This is the expected result. We next compute the entropy. We first note from Eq. (8.25) that we can write

$$S = S_{\text{system}} + S_{\text{net}}. \tag{8.30}$$

The system entropy is the normal entropy for a system with temperature T, volume V, and total chemical potential μ. Of more interest is the network entropy

$$S_{\text{net}} = -\left(\frac{\partial \Omega_{\text{net}}}{\partial T}\right)_{V,\mu,\{\mu_i\}} = -k\sum_i \frac{(\mu_i'-\Gamma_i)}{kT}\exp\left\{\frac{(\mu_i'-\Gamma_i)}{kT}\right\}. \tag{8.31}$$

From this we see

$$S_{\text{net}} = -k\sum_i X_i(t+\Delta t)\ln X_i(t+\Delta t). \tag{8.32}$$

The above result is not surprising in that it is simply the Shannon entropy [**?**]. What is interesting about it is that we may substitute in Eq. (5.4) to find

$$S_{\text{net}} = k\ln\left(\sum_B \exp\{\Delta(w(B)\}\right) - k\sum_i X_i(t+\Delta t)\ln\left(\sum_{B:(i)} \exp\{\Delta(B)\}\, X_B^{(i)}\right), \tag{8.33}$$

where we recall the sum of probabilities is unity. Let us now define a "complexity" associated with vertex i as $\mathcal{C}_i$:

$$\mathcal{C}_i = \sum_{B:(i)} \exp\{\Delta(B)\}\, X_B^{(i)}. \tag{8.34}$$

The total complexity

$$\mathcal{C} = \sum_i \mathcal{C}_i. \tag{8.35}$$

The result is

$$S_{\rm net} = k \ln \mathcal{C} - k \sum_i X(t + \Delta t) \ln \mathcal{C}_i = k \ln \mathcal{C} - k \ln \langle \mathcal{C}_i \rangle, \tag{8.36}$$

where $\langle \mathcal{C}_i \rangle$ is the average complexity associated with a single vertex. The total complexity is the weighted sum of branchings in the network compared to the maximum branching. For example, if only one branching dominates, $\mathcal{C} = 1$. If all branchings have equal weight, $\mathcal{C} = N$, where N is the total number of branchings. Since $X_B^{(i)}$ is the fraction of a branching B that "belongs" to root vertex i, the network entropy is given by the logarithm of the full complexity less the logarithm of the average complexity belonging to any given node.

As a final point, we consider the change in entropy in the system. In the absence of an external heating, the entropy is given by

$$T dS = -\sum_i \mu_i dN_i, \tag{8.37}$$

where $N_i = X_i N$, with N the total number of atoms. In terms of the specific entropy $s = S/N$, we may find

$$d(s/k) = -\sum_i \frac{\mu_i}{kT} dX_i. \tag{8.38}$$

From Eq. (8.9), we obtain

$$d(s/k) = -\sum_i \left(\frac{\mu_i'}{kT} + \frac{\Gamma_i}{kT} \right) dX_i, \tag{8.39}$$

where the part dependent on A drops out because $\sum_i dX_i = 0$. From Eqs. (8.8) and (8.11) we then find

$$d(s/k) = -\sum_i \ln \left(\frac{\sum_{B:(i)} \exp\{w(B)\} X_B^{(i)}}{\sum_{B:(i),span} \exp\{w(B)\}} \right) dX_i. \tag{8.40}$$

This result gives us a useful way of computing the entropy change in a system from the branchings. Notice in particular that for long-time solutions in a strongly-connected network that $d(s/k) \to 0$ because spanning branchings dominate and the logarithm goes to $\ln(1)$.

CHAPTER 9

NON-LINEAR NETWORK TREATMENT

Thus far I have only treated linear reaction networks. This chapter extends the analysis to non-linear networks. For definiteness, I will use the example of nuclear reaction networks in astrophysical plasmas (generally stellar interiors) in which the reacting species are fully-ionized nuclei.

To begin, consider the number density n_i of a nuclear species i in an astrophysical plasma. If we ignore small corrections in mass due to binding energies, the number density of nucleons is $N_A\rho$, where N_A is Avogadro's number and ρ is the mass density. We thus define Y_i, the abundance of i per nucleon, to be

$$Y_i = \frac{n_i}{N_A\rho}. \tag{9.1}$$

Let us now suppose i participates in a reaction $i + j + ... + k \leftrightarrow \ell + m + ...n$. The convention will be that the "forward" direction for the reaction is the exothermic direction while the "reverse" direction is the endothermic direction. By convention, then, the reactants in the reaction are those species combining in the exothermic direction to produce the products. Let R be the set of reactants and P be the subset of products. There are N_R reactants and N_P products. The time rate of change of a species $q \in R$ due to this reaction is thus

$$\frac{dY_{q\in R}}{dt} = -\lambda_{for}\prod_{j\in R} Y_j + \lambda_{rev}\prod_{k\in P} Y_k, \tag{9.2}$$

where λ_{for} is the rate per interacting set of nuclei in the forward direction and λ_{rev} is the rate per interacting set of nuclei in the reverse direction. The time rate of change of a species $s \in P$ due to this reaction, then, is

$$\frac{dY_{s\in P}}{dt} = \lambda_{for}\prod_{j\in R} Y_j - \lambda_{rev}\prod_{k\in P} Y_k, \tag{9.3}$$

We now define the mass fraction X_i of species i. This is the number of grams of i per gram of total material. Since species i has Z_i protons and A_i nucleons, the mass

fraction of species i is

$$X_i = A_i Y_i. \tag{9.4}$$

By conservation of mass, we then note

$$\sum_i X_i = 1. \tag{9.5}$$

X_i thus is the probability that a gram of nucleons will be in the form of species i. With these definitions, Eqs. (9.2) and (9.3) become

$$\frac{dX_{q\in R}}{dt} = -\lambda_{for} A_q \prod_{j\in R} \frac{X_j}{A_j} + \lambda_{rev} A_q \prod_{k\in P} \frac{X_k}{A_k}, \tag{9.6}$$

and

$$\frac{dX_{s\in P}}{dt} = \lambda_{for} A_s \prod_{j\in R} \frac{X_j}{A_j} - \lambda_{rev} A_s \prod_{k\in P} \frac{X_k}{A_k}. \tag{9.7}$$

To evolve the mass fractions, we now finite difference in time and linearize. We first define modified forward and reverse rates as

$$\tilde{\lambda}_{for} \equiv \frac{\lambda_{for}}{N_R} \prod_{j\in R} \frac{X_j(t)}{A_j} \tag{9.8}$$

and

$$\tilde{\lambda}_{rev} \equiv \frac{\lambda_{rev}}{N_P} \prod_{k\in P} \frac{X_k(t)}{A_k}. \tag{9.9}$$

For $q \in R$, the result is then

$$\frac{X_q(t+\Delta t) - X_q(t)}{\Delta t} = -\tilde{\lambda}_{for} \sum_{\ell\in R} \frac{A_q}{X_\ell(t)} X_\ell(t+\Delta t) + \tilde{\lambda}_{rev} \sum_{m\in P} \frac{A_q}{X_m(t)} X_m(t+\Delta t). \tag{9.10}$$

For $s \in P$, the result is

$$\frac{X_s(t+\Delta t) - X_s(t)}{\Delta t} = \tilde{\lambda}_{for} \sum_{\ell\in R} \frac{A_s}{X_\ell(t)} X_\ell(t+\Delta t) - \tilde{\lambda}_{rev} \sum_{m\in P} \frac{A_s}{X_m(t)} X_m(t+\Delta t). \tag{9.11}$$

As with the linear network, we may thus write

$$(I + M\Delta t)\, X(t+\Delta t) = X(t), \tag{9.12}$$

where $X(t)$ and $X(t+\Delta t)$ are vectors whose elements are the mass fractions of all species at time t and $t+\Delta t$, respectively.

We now identify the contribution of the particular reaction to the matrix elements. For $u \in R$ and $v \in R$, we find

$$M_{u\in R,v\in R}\Delta t = \tilde{\lambda}_{for}\Delta t\frac{A_u}{X_v(t)}. \tag{9.13}$$

For $u \in R$ and $v \in P$, we find

$$M_{u\in R,v\in P}\Delta t = -\tilde{\lambda}_{rev}\Delta t\frac{A_u}{X_v(t)}. \tag{9.14}$$

For $u \in P$ and $v \in R$,

$$M_{u\in P,v\in R}\Delta t = -\tilde{\lambda}_{for}\Delta t\frac{A_u}{X_v(t)}. \tag{9.15}$$

Finally, for $u \in P$ and $v \in P$,

$$M_{u\in P,v\in P}\Delta t = \tilde{\lambda}_{rev}\Delta t\frac{A_u}{X_v(t)}. \tag{9.16}$$

We now consider the column sum of matrix elements from a particular reaction. Consider first a column v such that $v \in R$. We sum over rows. The result is

$$\begin{aligned}\sum_u M_{u,v\in R} &= \sum_{u_1\in R} M_{u_1\in R,v\in R} + \sum_{u_2\in P} M_{u_2\in P,v\in R} \\ &= \frac{\tilde{\lambda}_{for}}{X_v(t)}\left(\sum_{u_1\in R} A_{u_1} - \sum_{u_2\in P} A_{u_2}\right) \\ &= 0. \end{aligned} \tag{9.17}$$

The last line follows from the fact that the number of nucleons in the reaction is conserved. If $v \in P$, then

$$\begin{aligned}\sum_u M_{u,v\in P} &= \sum_{u_1\in R} M_{u_1\in R,v\in P} + \sum_{u_2\in P} M_{u_2\in P,v\in P} \\ &= \frac{\tilde{\lambda}_{rev}}{X_v(t)}\left(-\sum_{u_1\in R} A_{u_1} + \sum_{u_2\in P} A_{u_2}\right) \\ &= 0. \end{aligned} \tag{9.18}$$

The matrix defined in this way thus has the crucial property that column sums are zero. This means that the diagonal element is the negative of the sum of the other elements in

the same column. The network can then be represented as a digraph as we have previously discussed.

As we have previously seen, the (u, v) matrix element corresponds to an arc from vertex u to vertex v. We thus can identify arc weights. If u and v are distinct vertices, and if $u \in R$ and $v \in P$, then then arc weight w_{uv} is

$$w_{u\in R,v\in P} = \ln\left[\tilde{\lambda}_{rev}\Delta t\frac{A_u}{X_v(t)}\right]. \tag{9.19}$$

Similarly

$$w_{u\in P,v\in R} = \ln\left[\tilde{\lambda}_{for}\Delta t\frac{A_u}{X_v(t)}\right]. \tag{9.20}$$

These are exactly analogous to the linear network case except that we use the modified rates and the arc weight is modified by the ratio $A_u/X_v(t)$. Thus one must multiply the modified rate times Δt by the mass number of the outvertex and divide by the mass fraction at time t of the invertex.

If u and v are distinct, and if $u \in R$ and $v \in R$ or $u \in P$ and $v \in P$, the matrix element M_{uv} is positive. By the network digraph rules, then, the arc weight is the logarithm of a negative number. This is incorporated by adding $i\pi$ to the arc weight, where $i = \sqrt{-1}$, as usual. The result is

$$w_{u\in R,v\in R} = i\pi + \ln\left[\tilde{\lambda}_{for}\Delta t\frac{A_u}{X_v(t)}\right] \tag{9.21}$$

and

$$w_{u\in P,v\in P} = i\pi + \ln\left[\tilde{\lambda}_{rev}\Delta t\frac{A_u}{X_v(t)}\right] \tag{9.22}$$

A complex arc weight adds a complication to the problem since branching weights are now possibly complex; however, this complication is minor since the imaginary part of the branching weight will be an integer multiple of $i\pi$. In comparing exponential sums of branching weights, then, we must simply compare absolute magnitudes to determine a branching's relative contribution.

A mnemonic may help in remembering when an arc requires the $i\pi$ addend. Consider first, for example, the reaction $i \leftrightarrow j$. This is a linear reaction. In the network digraph, this reaction contributes an arc $i \to j$ and an arc $j \to i$. Consider now the non-linear reaction $i + j \leftrightarrow k$. This reaction thus contributes normal arcs $i \to k$, $j \to k$, $k \to i$, and $k \to j$ since

these vertices all lie on opposite sides of the reaction. There are also arcs between reactants i and j. We can accommodate this by writing $i \to k - j$ and $j \to k - i$. The negative sign in front of the i or j means arc $i \to j$ and the arc $j \to i$ thus includes the logarithm of a negative one. Alternatively, we may imagine a pivot point attached to the $\leftrightarrow$ in $i + j \leftrightarrow k$. To get a arc from i to j, we must "rotate" j by π radians about the pivot point to get it on the other side of the reaction. This π then can be thought of as the $i\pi$ in the arc weight.

The matrix derived from a single reaction is singular–each row in the matrix is a multiple of the other rows. Consider the matrix elements for a given row u_1. The elements have the form of $\tilde{\lambda}\Delta t$ times the ratio $A_{u_1}/X_v(t)$, where v is the column number. For another row u_2, the element is the same $\tilde{\lambda}\Delta t$ times the ratio $A_{u_2}/X_v(t)$. Thus all elements of row u_2 are a multiple A_{u_2}/A_{u_1} of the corresponding elements of row u_1.

An important consequence of the fact that the rows of the matrix are multiples of each other is the result that $det\,(I + M\Delta t) = 1 + Tr(M\Delta t)$. This is because the determinant is the sum of all determinants of matrices formed from permutations of rows of I and $M\Delta t$. Any matrix with two or more rows from $M\Delta t$ will give determinant zero because those two rows are multiples of each other; thus, the only permutations giving non-zero values are I and the matrices formed from one row of $M\Delta t$ and all the other rows from I. The determinant of these latter matrices is simply the product of the diagonal elements, and the sum of these determinants is thus the trace of $M\Delta t$. As we have seen, a diagonal element of $M\Delta t$ is the negative of the sum of the other elements in the same column. These other elements are the negative of the exponential arc weights. The consequence is that branchings arising from a single non-linear reaction are simply the sum of the individual arcs. In other words, branchings from a single reaction can have at most one arc.

We now consider many reactions. For each reaction r we construct a matrix M_r. We may then write Eq. (9.12) as

$$(I + M\Delta t)\, X(t + \Delta t) = \left(I + \sum_r M_r \Delta t \right) X(t + \Delta t) = X(t). \tag{9.23}$$

To compute $det\,(I + M\Delta t)$, we again note that this is the sum of all determinants of matrices formed from permutations of rows of I and the various M_r. Because the rows of each M_r are multiples of each other, however, any matrix that has two or more rows from a given M_r

will yield determinant zero. A determinant is a sum of terms, each of which contains only one element from each row in the matrix [see Eq. (B.1)]. We also note that the terms that survive in this sum are each exponentials of branching weights in our network digraph; thus, we find the result that a reaction can contribute no more than one arc to any branching.

Unlike linear reactions, each non-linear reaction contributes multiple arcs to the network digraph. For example, the forward reaction $i + j \rightarrow k$ contributes four arcs: $i \rightarrow j$, $i \rightarrow k$, $j \rightarrow i$, and $j \rightarrow k$. Nevertheless, only one of these arcs may appear in any branching in the network. A branching that includes, for example, two arcs from the same reaction will be canceled out by one or more branchings with two arcs from the same reaction in the sum of branchings that gives rise to the determinant. Normal computation of a determinant handles these calculations; however, since these cancellations are often between large numbers, numerical inaccuracies can creep in. By explicitly excluding terms in the determinant that will cancel out, we can not only reduce the size of our calculation but also increase its accuracy.

As a final note, the useful result for the linear network that long-term solutions can be obtained from a single spanning tree does not hold for the non-linear network when detailed balance applies. This is because there is not a simple relationship between $\tilde{\lambda}_{for}$ and $\tilde{\lambda}_{rev}$. These rates also depend on the abundances of the interacting species at time t which may or may not have their equilibrium values.

In summary, non-linear reaction networks may also be analyzed with branchings in network digraphs. For nuclear reaction networks, we must first consider mass fractions in place of abundances. This permits the matrix M for the reaction to have the crucial property that the elements in a give column sum to zero. We must then modify reaction rates by including the abundances of the reactants and dividing by the number of reactants to account for the linearization. Then we must assign arcs weights in the network digraph. To do this, we must multiply the modified rate by Δt and by the ratio of the mass number of the outvertex to the mass fraction at time t of the invertex. The logarithm of the absolute value of this quantity is the arc weight. We finally add $i\pi$ to the arc weight if the arc is between two reactants (or two products) in the reaction. At this point we may apply all the previous rules we developed for analyzing linear reactions with branchings in digraphs

except that detailed balance must be treated more carefully. An important added constraint on our calculation of branchings is that a reaction may contribute no more than one arc to any branching.

While the above analysis considered nuclear reaction networks, it should readily to any non-linear network. The key is to identify the quantity conserved in reactions (in the above case, the number of nucleons) and the effective probability (in the above case, the mass fraction of species).

CHAPTER 10
EXAMPLES AND APPLICATIONS

The analysis in the previous chapters is rather abstract. Here I present some examples to illustrate the key ideas of applications of branchings to reaction networks.

10.1 Two-State System

The simplest non-trivial system we can analyze is the two-state linear network shown in Fig. 10.1. The transition rate from state 1 to 2 is λ_{12} and from state 2 to 1 is λ_{21}. We create a network digraph by assigning states 1 and 2 to vertices 1 and 2 in the digraph, respectively, and then, for time step Δt, adding arc $(1,2)$ with weight $\ln(\lambda_{21}\Delta t)$ and arc $(2,1)$ with weight $\ln(\lambda_{12}\Delta t)$. We then add fictitious vertex 0 and arcs $(0,1)$ and $(0,2)$ with weights 0 each. This results in the network digraph shown in Fig. 10.2.

To analyze the flows from t to $t+\Delta t$, we must find the branchings. There are three, as shown in Fig. 10.3. Shown are the two-arc arborescences including fictitious vertex 0 and the arcs out of vertex 0 to one or more of vertices 1 and 2. If we focus on the actual graphs without the fictitious vertex, the branchings are the digraphs with only vertices 1 and 2 and the solid arcs; thus, the branchings are 1) the graph with vertices 1 and 2 and no arc [with branching weight $w(B_1) = 0$], 2) the graph with vertices 1 and 2 and arc $(1,2)$ [with branching weight $w(B_2) = \ln(\lambda_{21}\Delta t)$], and 3) the graph with vertices 1 and 2 and arc $(2,1)$ [with branching weight $w(B_3) = \ln(\lambda_{12}\Delta t)$].

With the branchings and their weights now available, it is possible to compute the transition matrix, that is, the inverse matrix we apply to the probabilities at t to get those at $t+\Delta t$. probabilities $X(t+\Delta t)$ from $X(t)$. From Eqs. (3.7) and (3.9), we find

$$\begin{pmatrix} X_1(t+\Delta t) \\ X_2(t+\Delta t) \end{pmatrix} = \frac{1}{1+(\lambda_{12}+\lambda_{21})\,\Delta t} \begin{pmatrix} 1+\lambda_{21}\Delta t & \lambda_{21}\Delta t \\ \lambda_{12}\Delta t & 1+\lambda_{12}\Delta t \end{pmatrix} \begin{pmatrix} X_1(t) \\ X_2(t) \end{pmatrix} \tag{10.1}$$

In Eq. (10.1, the factor in front of the matrix is one over the sum of the exponentials of the all the branchings weights. The $(1,1)$ element of the matrix is then this overall factor

Figure 10.1 Raw network for the two-state system.

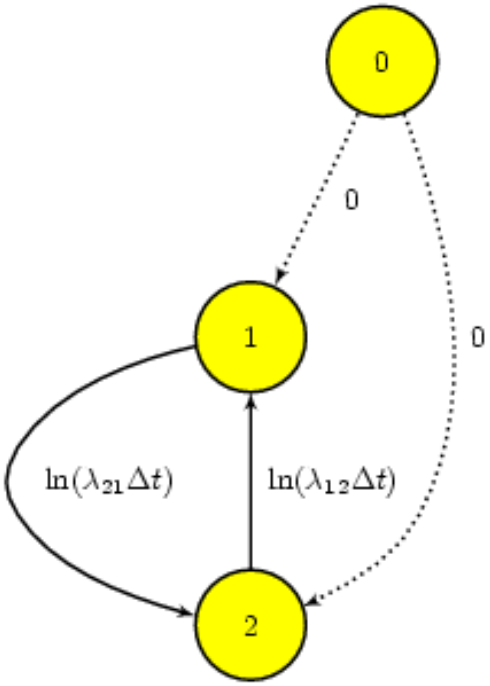

Figure 10.2 Network digraph for the two-state system.

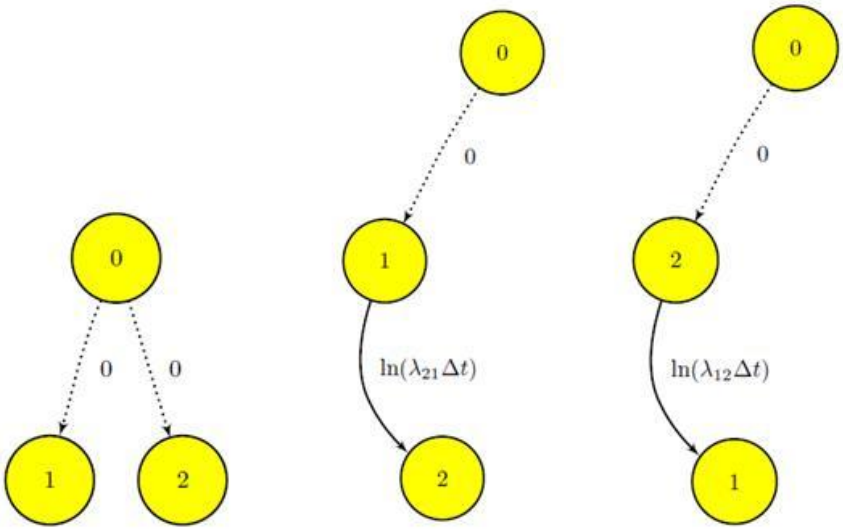

Figure 10.3 Branchings for the two-state system.

times the sum of the exponential weights of all branchings rooted at vertex 1; thus, it is $\exp(w(B_1)) + \exp(w(B_2))$. The $(1,2)$ element of the matrix is this overall factor times the sum of the sum of the exponential weights of all branchings rooted at vertex 1 but with a path from vertex 1 to vertex 2; thus, it is $\exp(w(B_2))$. The $(2,1)$ and the $(2,2)$ matrix elements are similar except for branchings rooted at vertex 2.

We may compute the probabilities at $t + \Delta t$ directly from Eq. (10.1); however, it is instructive to compute them via the sorted branching algorithm. Let us assume the branching weights are such that $w(B_2) > w(B_1) > w(B_3)$, as determined by an implementation of the Camerini et al. algorithm. We would construct the probabilities at time $t + \Delta t$ as follows. We begin with $X_1(t + \Delta t) = X_2(t + \Delta t) = 0$. Now consider B_2. It is rooted at vertex 1, so it contributes to $X_1(t + \Delta t)$. Thus, from the first term in Eq. (5.6), we would find $\Delta(B_2) = 0$ and

$$X_1(t + \Delta t) = \frac{e^0 X^{(1)}_{B_2}(t)}{e^0} = X_1(t) + X_2(t) \tag{10.2}$$

and

$$X_2(t + \Delta t) = 0, \tag{10.3}$$

since B_2 is not rooted at vertex 2. Notice that in this step $X_1(t+\Delta t) + X_2(t+\Delta t) = X_1(t) + X_2(t) = 1$ since the probabilities at t are normalized. We next apply B_1. Here $\Delta(B_1) = 0 - \ln(\lambda_{21}\Delta t) = -\ln(\lambda_{21}\Delta t)$ and, hence,

$$X_1(t+\Delta t) = \frac{X_1(t) + X_2(t) + \frac{X_1(t)}{\lambda_{21}\Delta t}}{1 + \frac{1}{\lambda_{21}\Delta t}} = \frac{X_1(t)\,(1+\lambda_{21}\Delta t) + \lambda_{21}\Delta t X_2(t)}{1+\lambda_{21}\Delta t} \tag{10.4}$$

and

$$X_2(t+\Delta t) = \frac{\frac{X_2(t)}{\lambda_{21}\Delta t}}{1 + \frac{1}{\lambda_{21}\Delta t}} = \frac{X_2(t)}{1+\lambda_{21}\Delta t}. \tag{10.5}$$

If we sum, we find that $X_1(t+\Delta t) + X_2(t+\Delta t) = 1$. Finally, we consider B_3. Here $\Delta(B_3) = \ln(\lambda_{12}\Delta t) - \ln(\lambda_{21}\Delta t)$. If this branching were negligible [$\exp\{\Delta(B_3)\}$ sufficiently less than one], then we could stop, and the probabilities in Eq. (10.4) and (10.5) would be satisfactory. If not, we would include this branching to find

$$X_1(t+\Delta t) = \frac{X_1(t) + X_2(t) + \frac{X_1(t)}{\lambda_{21}\Delta t}}{1 + \frac{1}{\lambda_{21}\Delta t} + \frac{\lambda_{12}\Delta t}{\lambda_{21}\Delta t}} \tag{10.6}$$

and

$$X_2(t+\Delta t) = \frac{\frac{X_2(t)}{\lambda_{21}\Delta t} + \frac{\lambda_{12}\Delta t}{\lambda_{21}\Delta t}\,(X_1(t) + X_2(t))}{1 + \frac{1}{\lambda_{21}\Delta t} + \frac{\lambda_{12}\Delta t}{\lambda_{21}\Delta t}}. \tag{10.7}$$

After minor manipulations, these probabilities become

$$X_1(t+\Delta t) = \frac{(1+\lambda_{21}\Delta t)\,X_1(t) + \lambda_{21}\Delta t X_2(t)}{1 + (\lambda_{12}+\lambda_{21})\,\Delta t} \tag{10.8}$$

and

$$X_2(t+\Delta t) = \frac{\lambda_{12}\Delta t X_1(t) + (1+\lambda_{12}\Delta t)\,X_2(t)}{1 + (\lambda_{12}+\lambda_{21})\,\Delta t} \tag{10.9}$$

These are the results expected from Eq. (10.1).

In the long time limit ($\Delta t \to \infty$), $w(B_3)$ would become larger than $w(B_1)$. This means that we would add brancing B_3 before B_1 in computing the $X(t+\Delta t)$'s. In any event, the long-time limit of the probabilities are

$$X_1(t+\Delta t) = \frac{\lambda_{21}\Delta t}{(\lambda_{12}+\lambda_{21})\,\Delta t} = \frac{\lambda_{21}}{\lambda_{12}+\lambda_{21}} \tag{10.10}$$

and

$$X_2(t+\Delta t) = \frac{\lambda_{12}\Delta t}{(\lambda_{12}+\lambda_{21})\,\Delta t} = \frac{\lambda_{12}}{\lambda_{12}+\lambda_{21}}. \tag{10.11}$$

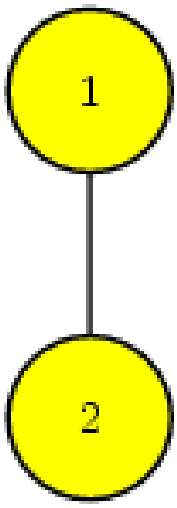

Figure 10.4 The single spanning tree in the simple two-state network.

Notice that these are from all spanning branchings (B_2 and B_3), which themselves are derived from the one spanning tree in the problem, shown in Fig. (10.4). Interestingly, the probabilities in Eq. (10.10) and (10.11) are the equilibrium values though we have not required detailed balance explicitly. That the equilibrium values derive from a single spanning tree is simply a consequence of the fact that there is only one spanning tree in the network.

The above procedure shows that as we add each branching, the solutions tend to converge on the correct result. What is also useful is that the sum of the probabilities always stays unity during each iteration; thus, if we stop the iteration before convergence, we nevertheless still have normalized probabilities.

From Eq. (6.3), we find, assuming $w(B_2)$ is the largest weight branching,

$$\lambda_{12}^{eff} = \frac{1}{\Delta t} \frac{\frac{\lambda_{12}}{\lambda_{21}}}{1 + \frac{1}{\lambda_{21}\Delta t} + \frac{\lambda_{12}}{\lambda_{21}}} \tag{10.12}$$

and

$$\lambda_{21}^{eff} = \frac{1}{\Delta t} \frac{1}{1 + \frac{1}{\lambda_{21}\Delta t} + \frac{\lambda_{12}}{\lambda_{21}}}. \tag{10.13}$$

We could then compute the probabilities at $t + \Delta t$ as

$$\frac{X_1(t+\Delta t) - X_1(t)}{\Delta t} = -\lambda_{12}^{eff} X_1(t) + \lambda_{21}^{eff} X_2(t) \tag{10.14}$$

and

$$\frac{X_2(t+\Delta t) - X_2(t)}{\Delta t} = \lambda_{12}^{eff} X_1(t) - \lambda_{21}^{eff} X_2(t) \tag{10.15}$$

Simple manipulation of these equations leads back to Eqs. (10.6) and (10.7). In general, this method of computing the probabilities is less useful than directly from the branchings, however, because there will be effective rates between every species; thus, we will need to compute and store $\mathcal{O}(n^2)$ effective rates, which may not be practical for a large network. Nevertheless, the effective rates are useful for analyzing flows between particular species over time step Δt.

It is worth noting that the total complexity of the network (again assuming that $w(B_2) > w(B_1) > w(B_3)$) is

$$C = 1 + \frac{1}{\lambda_{21}\Delta t} + \frac{\lambda_{12}}{\lambda_{21}}. \tag{10.16}$$

Depending on the relative magnitudes of the terms, it is clear that $1 \leq C \leq 3$. A complexity near 3 means all branchings are contributing. A complexity near 1 means that B_2 dominates. We may also note that the network entropy is

$$\begin{aligned} S_{net}/k &= \ln\left(1 + \frac{1}{\lambda_{21}\Delta t} + \frac{\lambda_{12}}{\lambda_{21}}\right) \\ &\quad - \frac{\left(1+\lambda_{21}\Delta t\right) X_1(t) + \lambda_{21}\Delta t X_2(t)}{1 + \left(\lambda_{12}+\lambda_{21}\right)\Delta t} \ln\left(1 + \frac{X_1(t)}{\lambda_{21}\Delta t}\right) \\ &\quad - \frac{\lambda_{12}\Delta t X_1(t) + \left(1+\lambda_{21}\Delta t\right) X_2(t)}{1 + \left(\lambda_{12}+\lambda_{21}\right)\Delta t} \ln\left(\frac{X_2(t)}{\lambda_{21}\Delta t} + \frac{\lambda_{12}}{\lambda_{21}}\right). \end{aligned} \tag{10.17}$$

At this point it is worth noting that the network entropy can decrease in the evolution of the system. Imagine, for instance, that $\lambda_{12} \ll \lambda_{21}$. The long-time (equilibrium) solution will be $X_1(t+\Delta t) \gg X_2(t+\Delta t)$. Suppose $X_2(t) = 1$ and $X_1(t) = 0$. If we imagine evolving the system by simply dialing up Δt, the network entropy will begin at zero. As probability shifts from state 2 to 1, however, there will be some point at which the two states will have equal probability. Here $S_{net} = k \ln 2$. Further evolution, however, will lead to the case that $X_1 \gg X_2$. In this case, the network entropy will decrease towards one

again. This is a consequence of the entropy being for a system with a finite number of states. Nevertheless, the total entropy $S = S_{system} + S_{net}$ is likely increasing with time because the energy released by the de-excitation is going into other degrees of freedom (say, the translational degrees of the atoms) that are accounted for in S_{system}.

This consideration leads to another thought experiment. Suppose the system starts at $t = 0$ at low temperature with equal probabilities for the two states: $X_1(t) = X_2(t) = 0.5$. Suppose further that at low temperature, the rates $\lambda_{12} = \lambda_{21} = 0$ so that the system is not evolving. Now suppose the system is suddenly heated up to high temperature such that $\lambda_{12} = \lambda_{21} = 1$. The states already have their equilibrium probabilities, so they will not evolve with time. Nevertheless, we have the sense that, before $\Delta t \approx 1$, the states have not "earned" their equilibrium, so something must be evolving. I propose that, in fact, what is evolving is the complexity. For small Δt, the complexity is simply unity because branching B_2 will dominate. As Δt increases, however, the complexity will first increase until it attains a maximum of three at $\Delta t = 1$ where all three branchings have equal weight. As Δt increases further, however, branchings B_1 and B_3 will dominate B_2, and the complexity will decrease to two, which is its long-time limit. In this way, the complexity, as I have defined it, may be a particulaly valuable diagnostic of the evolution of a reaction network.

10.2 Effective De-Excitation Rates

In this section, I address a question of excitation and de-excitation of a multi-level atom suggested by Prof. Sean Brittain of Clemson University. Fig. 10.5 shows three levels in an atom. There are transitions among all three levels. The transition rates are all governed by detailed balance so that the reverse and forward rates are related by a Boltzmann factor. Suppose now that the transitions between levels 1 and 2 are very weak. The problem is to demonstrate that the long-time limit is still such that levels 1 and 2 have their equilibrium probability ratio.

A standard approach is to compute a 3×3 rate matrix, set the right-hand side vector to zero, and solve. I will approach the problem from the digraph branching perspective. I begin by noting that if I exclude the transitions between levels 1 and 2, my network has a single spanning tree. I can compute the long-time probabilities for levels 1 and 2 from this

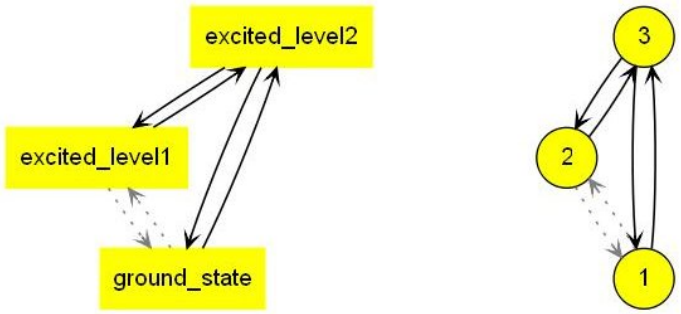

Figure 10.5 Excitation of a three-level system.

spanning tree by, in the case of level 1, a branching rooted at vertex 1 with arcs $(1,3)$ and $(3,2)$ and, in the case of level 2, a branching rooted at vertex 2 with arcs $(2,3)$ and $(3,1)$. Since the denominators of my branching result for $X_1(t+\Delta t)$ and $X_2(t+\Delta t)$ are the same, I find

$$\frac{X_{2,\infty}}{X_{1,\infty}} = \frac{\lambda_{32}\lambda_{13}}{\lambda_{23}\lambda_{31}}. \tag{10.18}$$

By detailed balance, this becomes

$$\frac{X_{2,\infty}}{X_{1,\infty}} = \frac{X_{2,eq}X_{3,eq}}{X_{3,eq}X_{1,eq}} = \frac{X_{2,eq}}{X_{1,eq}}, \tag{10.19}$$

as required.

It is also possible to compute an effective excitation rate from level 1 to level 2 (via level 3). To do this, we compute λ_{12}^{eff}. This rate is the exponential of the branching weight with the path from vertex 2 to 1 divided by the sum of the exponentials of all branching weights:

$$\lambda_{12}^{eff} = \frac{1}{\Delta t}\frac{\lambda_{32}\lambda_{13}\Delta t^2}{1+(\lambda_{13}+\lambda_{31}+\lambda_{23}+\lambda_{32})\,\Delta t+(\lambda_{13}\lambda_{32}+\lambda_{13}\lambda_{23}+\lambda_{31}\lambda_{23})\,\Delta t^2}. \tag{10.20}$$

For Δt much larger than the smallest $1/\lambda$ (but not ∞), the effective excitation rate is

$$\lambda_{12}^{eff} = \frac{\lambda_{32}\lambda_{13}}{\lambda_{13} + \lambda_{31} + \lambda_{23} + \lambda_{32} + (\lambda_{13}\lambda_{32} + \lambda_{13}\lambda_{23} + \lambda_{31}\lambda_{23})\,\Delta t}. \tag{10.21}$$

The effective de-excitation rate is similarly easy to find. The essence of these effective rates is that they are the sum of exponential weights of spanning branchings with paths from 1 to 2 or 2 to 1 divided by the exponential weights of all branchings with one fewer arc than a spanning branching.

If we extend the atom to more than three levels, the above procedures also apply. We simply need to find a single spanning tree and consider only branchings derived from it. Alternatively, we could apply a Camerini et al. based algorithm. For the effective excitation rate, we would constrain the branchings to be those with the appropriate paths.

10.3 Silicon Disintegration

As a final example, I consider a simple non-linear network for silicon disintegration. The network consists of six species (^{4}He, ^{12}C, ^{16}O, ^{20}Ne, ^{24}Mg, and ^{28}Si) and the reactions among them [$^4\mathrm{He} + ^4\mathrm{He} + ^4\mathrm{He} \leftrightarrow ^{12}\mathrm{C} + \gamma$, $^{12}\mathrm{C} + ^4\mathrm{He} \leftrightarrow ^{16}\mathrm{O} + \gamma$, $^{16}\mathrm{O} + ^4\mathrm{He} \leftrightarrow ^{20}\mathrm{Ne} + \gamma$, $^{20}\mathrm{Ne} + ^4\mathrm{He} \leftrightarrow ^{24}\mathrm{Mg} + \gamma$, $^{24}\mathrm{Mg} + ^4\mathrm{He} \leftrightarrow ^{28}\mathrm{Si} + \gamma$– the γ's represent energy released in the reaction, which is typically in the form of one or more γ rays.]. The network digraph is shown in Fig. 10.6. In this figure, the arcs coming from different reactions have different colors. For example, the arcs for the triple-α reaction are shown in purple. Notice that there are three arcs from ^{4}He to ^{12}C and back because we treat each of the three ^{4}He's as a separate species–we could combine each of the set of three arcs into single arcs.

To analyze the branchings in this network, I employ the branching code described in Appendix C. To generate the arcs, I use a code built on top of libnucnet, a library of codes for storing and managing nuclear reactions networks written at Clemson University [Meyer and Adams 2007]. For definiteness, I started with mass fractions $X(^4\mathrm{He}) = X(^{12}\mathrm{C}) = X(^{16}\mathrm{O}) = X(^{20}\mathrm{Ne}) = X(^{24}\mathrm{Mg}) = 0.1$ and $X(^{28}\mathrm{Si}) = 0.5$. I computed the rates for reactions at a temperature of 4×10^9 K and a density of 1 g/cc. I then computed all the branchings at time steps $\Delta t = 10^{-3}$ s and $\Delta t = 10^3$ s.

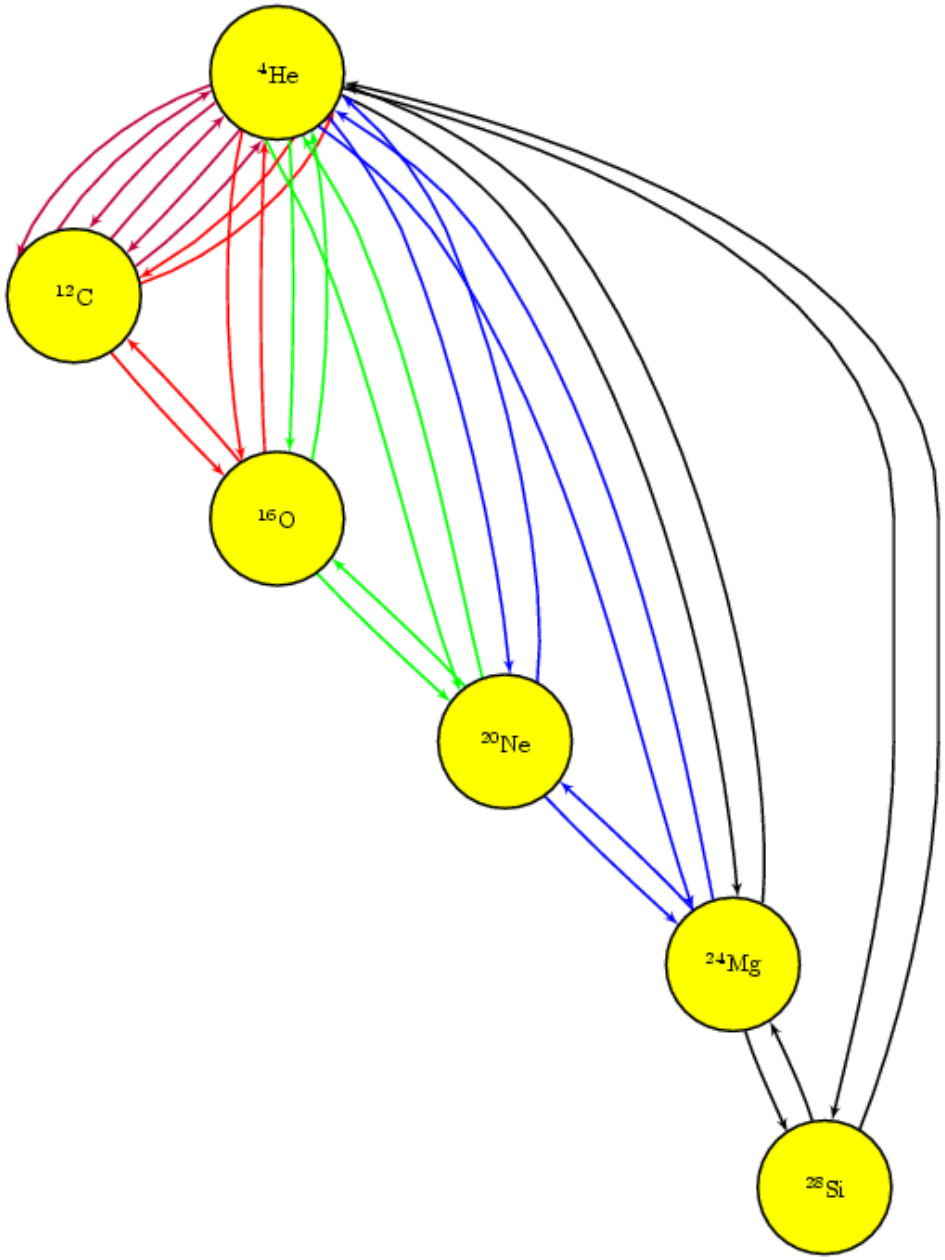

Figure 10.6 Digraph for the alpha network. Arcs deriving from different reactions have different colors.

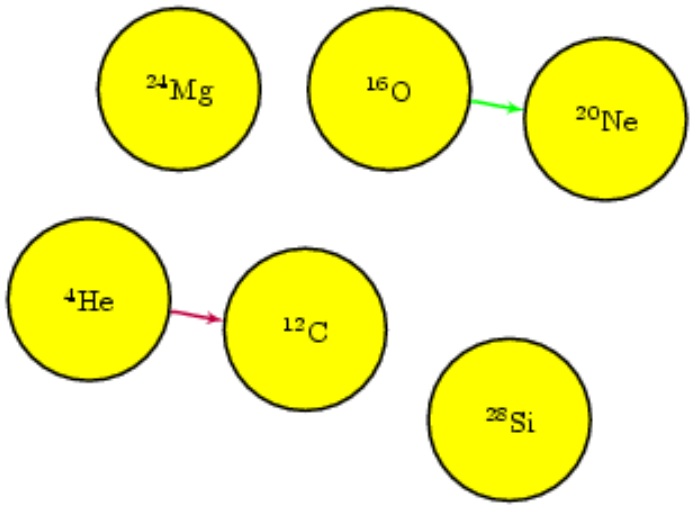

Figure 10.7 Optimal branching for the $\Delta t = 10^{-3}$ s case.

For $\Delta t = 10^{-3}$, there are a total of 2,949 branchings. The complexity is $C = 5.6731$, which is considerably less than the total number of branchings. The above numbers include the constraint that no reaction should contribute more than one arc to a branching. Had I not included this constraint, there would be a total of 8,046 branchings. The total complexity is the same, of course, due to the cancellations that are avoided by the constraint on the number of arcs from each reaction. Fig. 10.7 shows the optimal branching in this case–it has weight $w = 8.1331$.

The mass fractions at $t+\Delta t$ are $X(^4\mathrm{He}) = 0.213363$, $X(^{12}\mathrm{C}) = 0.0175353$, $X(^{16}\mathrm{O}) = 0.186649$, $X(^{20}\mathrm{Ne}) = 4.37987e-05$, $X(^{24}\mathrm{Mg}) = 0.086772$, and $X(^{28}\mathrm{Si}) = 0.495636$. The network entropy is $S_{net}/k = \ln(3.57598)$. If I constrain the branchings such that only those with $\Delta \geq 10^{-10}$, I find the same results (to the presented number of significant figures) with only 1,024 branchings. Perhaps more usefully, if I combine the three arcs for the triple-α reaction, there are only a total of 1,613 branchings and the complexity drops to 1.87903 although the entropy and final abundances are the same. This result shows that strategies to combine arcs where possible should be employed.

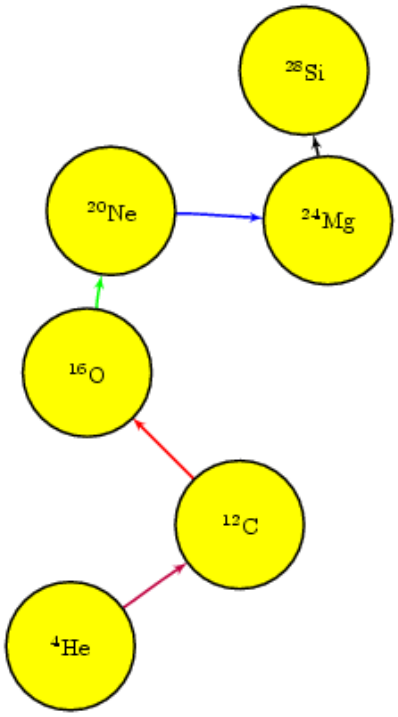

Figure 10.8 Optimal branching for the $\Delta t = 10^3$ s case.

I now consider the case $\Delta t = 10^3$ s. Here there are again a total of 2,949 branchings. The total complexity is $C = 7.88516$. This is larger than for the $\Delta t = 10^{-3}$ case because the longer time step allows a larger fraction of the arcs to contribute. The optimal branching is that shown in Fig. 10.8. This branching is spanning because all the arc weights are positive because Δt is sufficiently large.

The final mass fractions are $X(^4\mathrm{He}) = 0.88781$, $X(^{12}\mathrm{C}) = 9.18441e-08$, $X(^{16}\mathrm{O}) = 5.67553e-05$, $X(^{20}\mathrm{Ne}) = 2.89249e-08$, $X(^{24}\mathrm{Mg}) = 0.0151745$, and $X(^{28}\mathrm{Si}) = 0.0969591$. The network entropy is $S_{net}/k = 1.48588$. This is less than for the $\Delta t = 10^{-3}$ s case because the ^{28}Si and other species have largely disintegrated into ^{4}He, and this single species dominates the abundances. Interestingly, the initial network entropy is $S_{net}/k = 1.49787$, so, as in the thought experiment in §10.1, the network entropy grows and then declines again, ultimately to a value less than its starting value. If I consider only branchings

with $\Delta \geq 10^{-10}$, then there are only 437 contributing branchings; however, the $X(^{12}\mathrm{C}) = 9.18395e-08$ (the other mass fractions are the same to the number of significant figures presented). The ^{12}C mass fraction deviates in the third decimal place, as expected for cutting off the branchings at $\Delta \geq 10^{-10}$. It may be necessary to develop strategies to include at least one branching rooted at a low-probability vertex, even if it would be fall below the cut-off Δ.

As with the $\Delta t = 10^{-3}$ s case, combining the triple-α arcs into single arcs reduces the number of branchings to 1,613. The total complexity is $C = 2.62839$. This again emphasizes the desirability of combining arcs where possible.

Finally, I note that $\Delta t = 10^{3}$ s is large enough that the network has nearly achieved its long-time solution. For completeness, if I run the calculation at $\Delta t = 10^{30}$ s, I find $C = 7.88407$. This is slightly less than for the $\Delta t = 10^{3}$ s case, which shows a few four arc (sub-spanning) branchings still contributed at that time step. The final mass fractions are $X(^{4}\mathrm{He}) = 0.887866$, $X(^{12}\mathrm{C}) = 1.8853e-16$, $X(^{16}\mathrm{O}) = 3.79323e-05$, $X(^{20}\mathrm{Ne}) = 2.87501e-08$, $X(^{24}\mathrm{Mg}) = 0.0151733$, and $X(^{28}\mathrm{Si}) = 0.0969229$. The long-time network entropy is $S_{net}/k = 1.48548$, slightly down from the $\Delta t = 10^{3}$ s case. It is worth emphasizing that the long-time abundances above are not necessarily the equilibrium abundances because the treatment linearizes the problem. We could attain the equilibrium abundances by iterating with a Newton-Raphson procedure, as briefly outlined in §1.

CHAPTER 11
CONCLUSIONS

In this thesis, I have considered reaction networks in terms of branchings in directed graphs. In particular, I considered linearized networks that have been finite-differenced in time implicitly. I demonstrated that the resulting matrix can be written as incidence matrix for a directed graph times a weight matrix. The directed graph has vertices given by the various states or species in the network, plus a fictitious vertex (labeled with index "0"). The arcs in the graph are all the connections between the states (essentially the reactions), although the arcs point in the opposite direction compared to the actual flow in time since they represent the contribution of the invertex to the outvertx in time Δt. The arc weights are given by the logarithm of the reaction rate times the Δt. There are also arcs from vertex 0 to all the other vertices (but none from the other vertices to 0), all with weight zero.

With the network graph so constructed, I demonstrated that individual elements of the inverse matrix are given by the ratio of the sum of exponential weights of branchings in the digraph. For example, the (i, j) inverse matrix element is given by the sum of exponential weights of all branchings that have a path from vertex i to vertex j divided by the sum of all branchings. This let me define an effective rate for transition from j to i as this same ratio divided by Δt. Finally, the probability of state or species i at time $t + \Delta t$ is the sum of terms given by the exponential weight of a branching rooted at vertex i times the sum of all probabilities of vertices linked to i by a path going from i to that vertex, with this sum then divided by the sum of all exponential weights of branchings.

With these results, I then demonstrated how to condense strongly-connected components of clusters within the digraph. These condensed clusters then represented long-time "equilibrium" or steady-state components of the network. I demonstrated how to compute effective rates of transitions between these clusters. I then demonstrated how to view the non-equilibrium entropy of a multi-state network in terms of branchings. This led to my

definition of network complexity as the sum of exponential weights of branchings relative to the maximum weight branching.

The above results were derived for linear networks. I then demonstrated how to treat non-linear networks (in the case of nuclear reaction networks). The key is to evolve not abundances, as is traditionally done, but rather mass fractions, which are probabilities. By modifying arc weights accordingly, I showed the results for the linear network held for the non-linear one. An added result is that a particular reaction cannot contribute more than one arc to a branching. This leads to a tremendous reduction in the number of branchings that contribute and to greater accuracy in any calculation. Finally, I demonstrated my results with some examples and other applications.

At this point it is worth considering future directions in this research. Here is an incomplete list of future possible projects.

- Construct a code to compute the k-th largest branchings in a digraph. This code should be able to account for complex weights and to include no more than one arc from a given reaction in any branching. Prof. Meyer is consulting with Prof. M. Saltzman in the Department of Mathematical Sciences at Clemson to build a code based on the Camerini et al. algorithm. It will need to be extended to allow for the non-linear network constraints.

- Develop strategies for computing branchings rooted at a particular vertex. In particular, it is necessary to develop a version of the Camerini et al. algorithm that requires the arc from vertex 0 to vertex i to be included in all computed branchings. This is necessary to compute relative flows over time Δt efficiently.

- Develop strategies for computing branchings that necessarily include a path from vertex i to vertex j. This is necessary to compute effective rates efficiently.

- Study potential applications to iterative solvers for matrix equations. Such solvers require only matrix multiples, but they converge slowly if the matrix is not diagonally dominant. Convergence can be speeded up if the matrix is pre-conditioned, that is, multiplied by an approximate inverse matrix. Since branchings provide a systematic way of computing such an inverse matrix, it may be possible to compute a good preconditioner by, say, only including the leading few branchings in a calculation of the inverse matrix.

- Once the above are developed, implement parallel versions of these algorithms.

Branchings hold promise for a better understanding of reaction networks. Analysis of branchings may bridge the gap between long-time (equilibrium) solutions and kinetic

calculations both conceptually and computationally. Progress on the above tasks would help clarify this potential promise.

APPENDICES

Appendix A

Column addition in a square matrix

In this appendix, I demonstrate the well-known result that column addition in a square matrix leaves the determinant unchanged. To begin, consider the $n \times n$ square matrix B with elements

$$b_{kj} = \delta_{k,j} + \delta_{k,\ell}\delta_{j,m}. \tag{A.1}$$

Now consider the matrix product $C = AB$, where the elements of the $n \times n$ square matrix A are a_{ij} and those of the result matrix C are c_{ij}. We may thus write

$$c_{ij} = \sum_{k=1,n} a_{ik}b_{kj}, \tag{A.2}$$

If we apply Eq. (A.1), we find

$$c_{ij} = a_{ij} + a_{i\ell}\delta_{j,m}. \tag{A.3}$$

Matrix C is therefore the result of adding column ℓ to column m in matrix A. From Eq. (A.1), we see that matrix B is the identity matrix with an extra one at row ℓ and column m.

First consider the case $\ell = m$. This is the case of adding column ℓ to itself or, equivalently, doubling all elements in column ℓ. In this case, matrix B is diagonal with ones in each element except for a two in row ℓ, column ℓ. Now the determinant of C is

$$det(C) = det(AB) = det(A) \cdot det(B). \tag{A.4}$$

For a diagonal matrix, the determinant is the product of the diagonal elements; thus, in this case $det(B) = 2$ and $det(C) = 2det(A)$. Addition of a column to itself in a matrix doubles the matrix's determinant, a well-known result.

Now consider the case $\ell \neq m$. Matrix B is thus a matrix with ones down the diagonal and a single other matrix element with value one at row ℓ, column m. The determinant of B can be determined by Laplace expansion by minors. We expand along column ℓ, which has a single non-zero element, namely, the one on the diagonal. Striking row ℓ and column ℓ leaves a $(n-1) \times (n-1)$ square identity matrix as the only contributing minor to the

expansion. The determinant is unity; thus, for $\ell \neq m$, $det(B) = 1$ and $det(C) = det(A)$. Column addition in a square matrix leaves the determinant unchanged.

Appendix B

Determinant of the Sum of Two Matrices

In this appendix, I consider some well-known properties of the determinant of a sum of two matrices. First, one notes that the determinant of an $n \times n$ matrix A may be written

$$det(A) = \sum_{i_1,i_2,i_3,...,i_n} \epsilon_{i_1 i_2 i_3 ... i_n} a_{1 i_1} a_{2 i_2} a_{3 i_3} ... a_{n i_n}, \tag{B.1}$$

where $a_{j i_j}$ is the matrix element of A in row j and column i_j and where $\epsilon_{i_1 i_2 i_3 ... i_n}$ is the completely anti-symmetric Levi-Cività symbol, which is $+1$ for even permutations of its indices, -1 for odd permutations, and zero if any index is repeated (e.g., [Arfken and Weber 1995]). The sum in Eq. (B.1) runs from 1 to n for each index i_j. The determinant of the sum of two $n \times n$ square matrices A and B is thus

$$det(A+B) = \sum_{i_1,i_2,i_3,...,i_n} \epsilon_{i_1 i_2 i_3 ... i_n} (a_{1 i_1} + b_{1 i_1})(a_{2 i_2} + b_{2 i_2})(a_{3 i_3} + b_{3 i_3}) ... (a_{n i_n} + b_{n i_n}), \tag{B.2}$$

where $b_{j i_j}$ is the matrix element of B in row j and column i_j. Expansion of Eq. (B.2) results in a series of 2^n sums of the form

$$det(A+B) = \sum_{ab} \sum_{i_1,i_2,i_3,...,i_n} \epsilon_{i_1 i_2 i_3 ... i_n} c_{1 i_1} \times c_{2 i_2} \times c_{3 i_3} \times ... \times c_{n i_n}, \tag{B.3}$$

where each $c_{j i_j}$ is either $a_{j i_j}$ or $b_{j i_j}$ and the symbol $\sum_{ab}$ denotes the sum of the various choices. Since each individual sum in Eq. (B.3) is a determinant, it is evident that the determinant of $A + B$ is the sum of determinants of all matrices formed by replacing some number of rows of A by their corresponding rows in B. Thus, the determinant of $A + B$ is the determinant of A + the sum of the determinants of matrices formed by replacing one row of A by its corresponding row in B + the sum of determinants of matrices formed by replacing two rows of A by the two corresponding rows in B + ... + the sum of determinants of all matrices with $n - 1$ rows from B and one row from A + the determinant of B.

The following simple example demonstrates this result. Consider the matrix

$$A = \begin{pmatrix} a & -b \\ -a & b \end{pmatrix}. \tag{B.4}$$

We seek the determinant $|I + A|$, where I is the identity matrix. Direct calculation yields $|I + A| = 1 + a + b$. Using the permutation of rows approach, however, yields

$$|I+A| = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 0 \\ -a & b \end{vmatrix} + \begin{vmatrix} a & -b \\ 0 & 1 \end{vmatrix} + \begin{vmatrix} a & -b \\ -a & b \end{vmatrix} = 1+a+b+0 = 1+a+b \quad \text{(B.5)}$$

as expected.

Appendix C

Branching Code

In this appendix, I present the branching code written by myself and Professor Bradley S. Meyer (Clemson University). It computes all branchings and their weightings for the input digraph and the complexity (as defined in Eq. (8.35)). If initial probabilities $X_i(t)$ are provided in another input file, the code will also compute the network entropy and the probabilities $X_i(t+\Delta t)$. The code requires a "fictitious" vertex and at least one arc from that vertex to another one. It depends on the webnucleo.org library module wn_matrix (see http://www.webnucleo.org). wn_matrix itself depends on libxml, the GNOME XML toolkit and parser, and gsl, the GNU Scientific Library.

The code works by creating an array of arcs (sorted by in vertex) and a weight matrix. It then builds a tree. It constructs a root node. At this point the only "leaf" is the root node. It then considers the first arc. It adds it to the current leaves of the tree–at this point only the root node. It then considers the next arc. It adds that arc as a child of any leaf if it maintains the branching condition, namely, the arc should not create a cycle or a vertex with indegree larger than one. This is determined by recursing through the leaf and its parents up to the root node and adding the arc. This subset of arcs is then a possible branching. The code computes the determinant of the incidence submatrix of these arcs and checks that the corresponding weight submatrix has non-zero values on the diagonal. Once a leaf has $n-2$, where n is the number of vertices in the full digraph (including the fictitious vertex), it is no longer considered as a possible leaf to which one may add an arc since it already spans the digraph. That node of the tree is then added to a branching list. After all arcs have been considered, the branching list is sorted using qsort() and then output.

If the network is non-linear, the added "sign" tag keeps track of whether the arc is real or "virtual". Also, the sum of the number of reactants and products in a given reaction is stored in a hash. As possible branchings are considered, the hash is queried to determine if the branching would span the subset of arcs involved in the reaction. If it would, the arc is not added to the leaf under consideration.

This program is useful for studying digraphs up to about ten vertices, depending on the sparseness of the digraph. Despite the efficiency of the tree for storing the possible branchings, however, the required memory and execution time grow dramatically for increasingly large graphs. A new code built on the Camerini et al. [Camerini et al. 1980] algorithm is desirable. We are consulting with Professor Matthew Saltzman of the Department of Mathematical Sciences about implementing such a code.

To execute the code (executable named "branching"), one would type:

branching reacs.txt arcs.txt (probs.txt)

where the probs.txt (the probabilities file) is optional. For a three species non-linear network with two reactions ($1 \leftrightarrow 2$ called "12" and $1 + 2 \leftrightarrow 3$ called "123"), the reacs.txt file would be:

```
root  5
12    2
123   3
```

The root reaction is the one from the fictitious vertex to each of the other vertices. The current implementation of the code requires the number following the "root" to be larger than or equal to $n + 2$. The reaction "12" has one reactant and one product, hence the 2. The reaction "123" has two reactants and one product, hence the 3.

The arcs.txt file is

```
0  1  0    1  root
0  2  0    1  root
0  3  0    1  root
1  2  5    1  12
2  1  3    1  12
1  2  10. -1  123
2  1  11. -1  123
1  3  20.  1  123
```

```
3  1  21.  1  123
2  3  32.  1  123
3  2  41.  1  123
```

The first column is the outvertex of the given arc, the second is the invertex, the third is the arc weight, the fourth is the sign (1 for real arc, -1 for virtual arc), and the fifth is the reaction to which the arc belongs.

The probs.txt file looks simply like:

```
1  0.3
2  0.4
3  0.3
```

and gives the initial probability for each vertex.

The output from such input is:

```
(0,3) (3,2) (2,1)
w(B) = 42.0000     Delta(B) =  0.0000  exp{Delta(B)} = 1.0000e+00


(0,3) (3,2) (0,1)
w(B) = 41.0000     Delta(B) = -1.0000  exp{Delta(B)} = 3.6788e-01


(2,3) (1,2) (0,1)
w(B) = 37.0000     Delta(B) = -5.0000  exp{Delta(B)} = 6.7379e-03


(2,3) (0,2) (2,1)
w(B) = 33.0000     Delta(B) = -9.0000  exp{Delta(B)} = 1.2341e-04


(2,3) (0,2) (0,1)
w(B) = 32.0000     Delta(B) = -10.0000  exp{Delta(B)} = 4.5400e-05


(0,3) (1,2) (3,1)
```

```
w(B) = 26.0000     Delta(B) = -16.0000  exp{Delta(B)} = 1.1254e-07

(1,3) (1,2) (0,1)
w(B) = 25.0000     Delta(B) = -17.0000  exp{Delta(B)} = 4.1399e-08

(1,3) (0,2) (2,1)
w(B) = 21.0000     Delta(B) = -21.0000  exp{Delta(B)} = 7.5826e-10

(0,3) (0,2) (3,1)
w(B) = 21.0000     Delta(B) = -21.0000  exp{Delta(B)} = 7.5826e-10

(1,3) (0,2) (0,1)
w(B) = 20.0000     Delta(B) = -22.0000  exp{Delta(B)} = 2.7895e-10

(0,3) (0,2) (2,1)
w(B) = 11.0000     Delta(B) = -31.0000  exp{Delta(B)} = -3.4425e-14

(0,3) (1,2) (0,1)
w(B) = 10.0000     Delta(B) = -32.0000  exp{Delta(B)} = -1.2664e-14

(0,3) (1,2) (0,1)
w(B) =  5.0000     Delta(B) = -37.0000  exp{Delta(B)} = 8.5330e-17

(0,3) (0,2) (2,1)
w(B) =  1.0000     Delta(B) = -41.0000  exp{Delta(B)} = 1.5629e-18

(0,3) (0,2) (0,1)
w(B) =  0.0000     Delta(B) = -42.0000  exp{Delta(B)} = 5.7495e-19

Number of branchings = 15
```

```
Number of nodes = 3
Complexity = ln(1.37479)
1 0.0851881
2 0.000112884
3 0.914699
Entropy = ln(1.33963).
```

The code is:

```
/*////////////////////////////////////////////////////////////////////////////
// <file
//   repository_path  = "$Source: /pub/cvsprojects/nucleo/local/texfiles/thesis/wang_dis
//   revision         = "$Revision: 1.4 $"
//   date             = "$Date: 2009/11/30 15:18:37 $"
//   tag              = "$Name:  $"
//   template_version = "cp_template.0.12"
// >
//
//   <description>
//     <abstract>
//       Code to compute branchings in a general non-linear network.
//     </abstract>
//   </description>
//   <license>
//     This file was originally written by Changyuan Wang and Bradley S. Meyer.
//
//     This is free software; you can redistribute it and//or modify it
//     under the terms of the GNU General Public License as published by
//     the Free Software Foundation; either version 2 of the License, or
//     (at your option) any later version.
//
```

```
//     This software is distributed in the hope that it will be useful,
//     but WITHOUT ANY WARRANTY; without even the implied warranty of
//     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//     GNU General Public License for more details.
//   </license>
//
// </file>
////////////////////////////////////////////////////////////////////////////*/
#include <stdlib.h>
#include <stdio.h>
#include <WnMatrix.h>

#define BUF_SIZE 256
#define CUT -100.

/*============================================================================
// Define types.
//==========================================================================*/

typedef struct {
  xmlNodePtr pNode;
  double dWeight;
  double dWeightSign;
  double dDiff;
  double dDiffSign;
} branching;

typedef struct {
  size_t iInVertex;
  size_t iOutVertex;
```

```
  double dWeight;
  double dSign;
  xmlChar *sxReaction;
} arc;

typedef struct {
  xmlHashTablePtr pReactionHash;
  arc **pArcs;
} graph;

/*============================================================================
// Prototypes.
//============================================================================*/

int print_list( branching *, graph * );
void print_branching( xmlNodePtr, graph * );
double compute_weight( xmlNodePtr, graph * );
double compute_sign( xmlNodePtr, graph * );
double compute_complexity( xmlListPtr );
int complexity_walker( branching *, double * );
gsl_vector *compute_vertex_complexities( xmlListPtr, graph *, gsl_vector * );
void populate_branching_hash( xmlNodePtr, xmlHashTablePtr, graph * );
int vertex_complexities_walker( branching *, void * );
int assign_vertex_complexities( size_t *, void * );
void
add_vertex_complexities(
  const xmlChar *,
  void *,
  const xmlChar *,
  const xmlChar *,
```

```
  const xmlChar *
);
void get_branching_root_list( xmlNodePtr, xmlListPtr, graph * );
static void
add_node(xmlNodePtr, size_t, size_t, size_t, WnMatrix *, graph *, xmlListPtr );
xmlNodePtr add_child( xmlNodePtr, size_t );
void assign_matrix( xmlNodePtr, size_t, WnMatrix *, WnMatrix * );
void branching_free( xmlLinkPtr );
void arc_free( arc * );
double compute_determinant( WnMatrix * );
int is_valid_branching( xmlNodePtr, graph *, size_t, WnMatrix * );
int has_distinct_reactions( xmlNodePtr, graph * );
void distinct_reactions( xmlNodePtr, graph *, xmlHashTablePtr, int * );
int compare_branchings( const void *, const void * );
int arc_sort( const void *, const void * );
void branching_sort( xmlListPtr );
int populate_branching_array( branching *, void * );

/*============================================================================
// print_list().
//============================================================================*/

int
print_list( branching *p_branching, graph *p_graph )
{

  if( p_branching->dDiff < CUT ) return 0;

  print_branching( p_branching->pNode, p_graph );
  printf(
```

```
    "w(B) = %7.4f     Delta(B) = %7.4f  exp{Delta(B)} = %.4e\n",
    p_branching->dWeight,
    p_branching->dDiff,
    exp( p_branching->dDiff ) * p_branching->dDiffSign
  );
  printf( "\n" );

  return 1;

}

/*============================================================================
// print_branching().
//============================================================================*/

void
print_branching( xmlNodePtr p_node, graph *p_graph )
{

  xmlChar *sx_arc;
  size_t i;

  if( p_node == xmlDocGetRootElement( p_node->doc ) )
    fprintf( stdout, "\n" );
  else
  {
    sx_arc = xmlGetProp( p_node, (const xmlChar *) "arc" );
    i = (size_t) atol( (const char *) sx_arc ) - 1;
    xmlFree( sx_arc );
    fprintf(
```

```
      stdout,
      "(%lu,%lu) ",
      (unsigned long) p_graph->pArcs[i]->iOutVertex,
      (unsigned long) p_graph->pArcs[i]->iInVertex
    );
    print_branching( p_node->parent, p_graph );
  }

}

/*============================================================================
// compute_weight().
//============================================================================*/

double
compute_weight( xmlNodePtr p_node, graph *p_graph )
{

  xmlChar *sx_arc;
  size_t i;

  if( p_node == xmlDocGetRootElement( p_node->doc ) )
    return 0.;
  else
  {
    sx_arc = xmlGetProp( p_node, (const xmlChar *) "arc" );
    i = (size_t) atol( (const char *) sx_arc ) - 1;
    xmlFree( sx_arc );
    return
      compute_weight( p_node->parent, p_graph ) + p_graph->pArcs[i]->dWeight;
```

```
  }

}

/*============================================================================
// compute_sign().
//============================================================================*/

double
compute_sign( xmlNodePtr p_node, graph *p_graph )
{

  xmlChar *sx_arc;
  size_t i;

  if( p_node == xmlDocGetRootElement( p_node->doc ) )
    return 1.;
  else
  {
    sx_arc = xmlGetProp( p_node, (const xmlChar *) "arc" );
    i = (size_t) atol( (const char *) sx_arc ) - 1;
    xmlFree( sx_arc );
    return compute_sign( p_node->parent, p_graph ) * p_graph->pArcs[i]->dSign;
  }

}

/*============================================================================
// get_branching_root_list().
//============================================================================*/
```

```
void
get_branching_root_list( xmlNodePtr p_node, xmlListPtr p_list, graph *p_graph )
{

  xmlChar *sx_arc;
  size_t i;

  if( p_node == xmlDocGetRootElement( p_node->doc ) )
    return;
  else
  {
    sx_arc = xmlGetProp( p_node, (const xmlChar *) "arc" );
    i = (size_t) atol( (const char *) sx_arc ) - 1;
    xmlFree( sx_arc );
    if( p_graph->pArcs[i]->iOutVertex == 0 )
      xmlListPushBack( p_list, &p_graph->pArcs[i]->iInVertex );
    get_branching_root_list( p_node->parent, p_list, p_graph );
  }

}

/*============================================================================
// compute_complexity().
//============================================================================*/

double
compute_complexity( xmlListPtr p_list )
{
```

```
  double d_result = 0;

  xmlListWalk(
    p_list,
    (xmlListWalker) complexity_walker,
    &d_result
  );

  return d_result;

}

/*=========================================================================
// complexity_walker().
//=========================================================================*/

int
complexity_walker(
  branching *p_branching, double *p_result
)
{

  if( p_branching->dDiff < CUT ) return 0;

  *p_result += exp( p_branching->dDiff ) * p_branching->dDiffSign;

  return 1;

}
```

```
/*============================================================================
// compute_vertex_complexities().
//============================================================================*/

gsl_vector *
compute_vertex_complexities(
  xmlListPtr p_list,
  graph *p_graph,
  gsl_vector *p_probabilities
)
{

  struct {
    double dWeight;
    double dSum;
    graph *pGraph;
    gsl_vector *pProbabilities;
    gsl_vector *pVector;
    xmlHashTablePtr pHash;
  } work;

  size_t i;
  gsl_vector *p_return_vector;

  work.pGraph = p_graph;

  work.pProbabilities = p_probabilities;

  work.pVector = gsl_vector_calloc( p_probabilities->size );
```

```
  xmlListWalk(
    p_list,
    (xmlListWalker) vertex_complexities_walker,
    &work
  );

  p_return_vector = gsl_vector_calloc( p_probabilities->size );

  for( i = 0; i < p_return_vector->size; i++ )
    if( !WnMatrix__value_is_zero( gsl_vector_get( work.pVector, i ) ) )
      gsl_vector_set(
        p_return_vector,
        i,
        gsl_vector_get( work.pVector, i )
      );

  gsl_vector_free( work.pVector );

  return p_return_vector;

}

/*==========================================================================
// vertex_complexities_walker().
//==========================================================================*/

int
vertex_complexities_walker(
  branching *p_branching, void *p_data
)
```

```
{

  typedef struct {
    double dWeight;
    double dSum;
    graph *pGraph;
    gsl_vector *pProbabilities;
    gsl_vector *pVector;
    xmlHashTablePtr pHash;
  } work;

  xmlListPtr p_root_list;
  work *p_work = ( work * ) p_data;

  if( p_branching->dDiff < CUT ) return 1;

  p_work->dWeight = exp( p_branching->dDiff ) * p_branching->dDiffSign;

  p_work->pHash =
    xmlHashCreate( 0 );

  populate_branching_hash(
    p_branching->pNode,
    p_work->pHash,
    p_work->pGraph
  );

  p_root_list = xmlListCreate( NULL, NULL );

  get_branching_root_list(
```

```
    p_branching->pNode,
    p_root_list,
    p_work->pGraph
  );

  xmlListWalk(
    p_root_list,
    (xmlListWalker) assign_vertex_complexities,
    p_work
  );

  xmlListDelete( p_root_list );

  xmlHashFree( p_work->pHash, (xmlHashDeallocator) xmlFree );

  return 1;

}

/*=============================================================================
// populate_branching_hash().
//============================================================================*/

void
populate_branching_hash(
  xmlNodePtr p_node,
  xmlHashTablePtr p_hash,
  graph *p_graph
)
{
```

```
xmlChar *sx_arc, *sx_valid, sx_out[BUF_SIZE], sx_in[BUF_SIZE];
size_t i;

if( p_node != xmlDocGetRootElement( p_node->doc ) )
{
  sx_arc = xmlGetProp( p_node, (const xmlChar *) "arc" );
  i = (size_t) atol( (const char *) sx_arc ) - 1;
  xmlFree( sx_arc );

  xmlStrPrintf(
    sx_out,
    BUF_SIZE,
    (const xmlChar *) "%lu",
    (unsigned long) p_graph->pArcs[i]->iOutVertex
  );

  xmlStrPrintf(
    sx_in,
    BUF_SIZE,
    (const xmlChar *) "%lu",
    (unsigned long) p_graph->pArcs[i]->iInVertex
  );

  sx_valid = xmlCharStrdup( "valid" );
  xmlHashAddEntry3( p_hash, sx_out, sx_in, NULL, sx_valid );

  populate_branching_hash( p_node->parent, p_hash, p_graph );

}
```

```
}

/*============================================================================
// assign_vertex_complexities().
//============================================================================*/

int
assign_vertex_complexities( size_t *p_vertex, void *p_data )
{

  typedef struct {
    double dWeight;
    double dSum;
    graph *pGraph;
    gsl_vector *pProbabilities;
    gsl_vector *pVector;
    xmlHashTablePtr pHash;
  } work;

  xmlChar sx_vertex[BUF_SIZE];
  work *p_work = ( work * ) p_data;

  xmlStrPrintf( sx_vertex, BUF_SIZE, (const xmlChar *) "%lu", *p_vertex );

  p_work->dSum =
    gsl_vector_get( p_work->pProbabilities, *p_vertex );

  xmlHashScanFull3(
    p_work->pHash,
```

```
    sx_vertex,
    NULL,
    NULL,
    (xmlHashScannerFull) add_vertex_complexities,
    p_work
  );

  p_work->pVector->data[*p_vertex] += p_work->dWeight * p_work->dSum;

  return 1;

}

/*============================================================================
// add_vertex_complexities().
//============================================================================*/

void
add_vertex_complexities(
  const xmlChar *sx_valid,
  void *p_data,
  const xmlChar *sx_out,
  const xmlChar *sx_in,
  const xmlChar *sx_extra
)
{

  typedef struct {
    double dWeight;
    double dSum;
```

```
    graph *pGraph;
    gsl_vector *pProbabilities;
    gsl_vector *pVector;
    xmlHashTablePtr pHash;
  } work;

  work *p_work = (work *) p_data;

  if( !sx_valid || !sx_out || sx_extra )
  {
    fprintf( stderr, "Not valid input.\n" );
    exit(EXIT_FAILURE);
  }

  p_work->dSum +=
    gsl_vector_get(
      p_work->pProbabilities,
      (size_t) atol( (const char *) sx_in )
    );

  xmlHashScanFull3(
    p_work->pHash,
    sx_in,
    NULL,
    NULL,
    (xmlHashScannerFull) add_vertex_complexities,
    p_work
  );

}
```

```
/*============================================================================
// add_child().
//============================================================================*/

xmlNodePtr
add_child( xmlNodePtr p_parent, size_t i_arc )
{

  xmlChar sx_arc[BUF_SIZE];
  xmlNodePtr p_child;

  xmlStrPrintf(
    sx_arc,
    BUF_SIZE,
    (const xmlChar *) "%lu",
    (unsigned long) i_arc
  );

  p_child =
    xmlNewChild(
      p_parent,
      NULL,
      (const xmlChar *) "node",
      NULL
    );

  xmlNewProp( p_child, (const xmlChar *) "arc", sx_arc );

  return p_child;
```

```
}

/*============================================================================
// add_node().
//============================================================================*/

static void
add_node(
  xmlNode * p_node,
  size_t i_arc,
  size_t i_level,
  size_t i_nodes,
  WnMatrix *p_incidence,
  graph *p_graph,
  xmlListPtr p_list
)
{

  xmlNodePtr p_cur_node = NULL, p_child;
  branching *p_branching;

  if( p_node->type == XML_ELEMENT_NODE )
  {
    if( i_level <  i_nodes )
    {
      p_child = add_child( p_node, i_arc );
      if( !is_valid_branching( p_child, p_graph, i_level, p_incidence ) )
      {
        xmlUnlinkNode( p_child );
```

```
    xmlFreeNode( p_child );
    p_child = NULL;
  }
  for(
    p_cur_node = p_node->children;
    p_cur_node;
    p_cur_node = p_cur_node->next
  )
    if( p_cur_node != p_child )
      add_node(
        p_cur_node,
        i_arc,
        i_level + 1,
        i_nodes,
        p_incidence,
        p_graph,
        p_list
      );
}
if( i_level == i_nodes - 1 )
{
  if( p_child )
  {
    p_branching = ( branching * ) malloc( sizeof( branching ) );
    p_branching->pNode = p_child;
    p_branching->dWeight = compute_weight( p_child, p_graph );
    p_branching->dWeightSign = compute_sign( p_child, p_graph );
    xmlListPushBack( p_list, p_branching );
  }
}
```

```
  }

}

/*============================================================================
// branching_free().
//============================================================================*/

void
branching_free( xmlLinkPtr p_link )
{

  branching *p_branching;

  p_branching = (branching *) xmlLinkGetData( p_link );

  free( p_branching );
}

/*============================================================================
// arc_free().
//============================================================================*/

void
arc_free( arc *p_arc )
{

  xmlFree( p_arc->sxReaction );
  free( p_arc );
```

```
}

/*============================================================================
// compute_determinant().
//============================================================================*/

double
compute_determinant( WnMatrix *self )
{

  gsl_matrix *p_matrix;
  gsl_permutation *p_perm;
  int i_signum = 0;
  double d_result;

  p_perm = gsl_permutation_alloc( WnMatrix__getNumberOfRows( self ) );

  p_matrix = WnMatrix__getGslMatrix( self );

  gsl_linalg_LU_decomp( p_matrix, p_perm, &i_signum );

  d_result = gsl_linalg_LU_det( p_matrix, i_signum );

  gsl_matrix_free( p_matrix );
  gsl_permutation_free( p_perm );

  return d_result;

}
```

```
/*============================================================================
// assign_matrix().
//============================================================================*/

void
assign_matrix(
  xmlNodePtr p_node,
  size_t i_col,
  WnMatrix *p_parent_matrix,
  WnMatrix *p_matrix
)
{

  WnMatrix__Line *p_col;
  xmlChar *sx_arc;
  size_t i;

  if( i_col > 0 )
  {
    sx_arc = xmlGetProp( p_node, (const xmlChar *) "arc" );
    p_col =
      WnMatrix__getColumn( p_parent_matrix, (size_t) atol( (char *) sx_arc ) );
    xmlFree( sx_arc );
    for( i = 0; i < WnMatrix__Line__getNumberOfElements( p_col ); i++ )
    {
      WnMatrix__assignElement(
          p_matrix,
          WnMatrix__Line__getNonZeroIndices( p_col )[i],
          i_col,
          WnMatrix__Line__getNonZeroElements( p_col )[i]
```

```
        );
    }
    WnMatrix__Line__free( p_col );
    assign_matrix( p_node->parent, i_col - 1, p_parent_matrix, p_matrix );
  }

}

/*============================================================================
// has_distinct_reactions().
//============================================================================*/

int
has_distinct_reactions( xmlNodePtr p_node, graph *p_graph )
{

  int i_result = 1;
  xmlHashTablePtr p_hash;

  p_hash = xmlHashCreate( 0 );

  distinct_reactions( p_node, p_graph, p_hash, &i_result );

  xmlHashFree( p_hash, (xmlHashDeallocator) xmlFree );

  return i_result;

}

/*============================================================================
```

```
// distinct_reactions().
//============================================================================*/

void
distinct_reactions(
  xmlNodePtr p_node,
  graph *p_graph,
  xmlHashTablePtr p_hash,
  int *p_result
)
{

  xmlChar *sx_arc, *sx_current, *sx_max, *sx_new_current, sx_tmp[BUF_SIZE];
  size_t i;
  int i_max, i_current;

  if( p_node == xmlDocGetRootElement( p_node->doc ) )
    return;

  sx_arc = xmlGetProp( p_node, (const xmlChar *) "arc" );
  i = (size_t) atol( (const char *) sx_arc ) - 1;
  xmlFree( sx_arc );

  sx_max =
      xmlHashLookup(
        p_graph->pReactionHash,
        p_graph->pArcs[i]->sxReaction
      );

  if( !sx_max )
```

```
{
  fprintf(
    stderr,
    "Reaction %s not present.\n", p_graph->pArcs[i]->sxReaction
  );
  exit( EXIT_FAILURE );
}

i_max = atoi( (char *) sx_max );

sx_current =
  xmlHashLookup(
    p_hash,
    p_graph->pArcs[i]->sxReaction
  );

if( !sx_current )
  i_current = 0;
else
  i_current = atoi( (char *) sx_current );

if( ++i_current < i_max - 1 || i_max == 2 )
{
  xmlStrPrintf(
    sx_tmp,
    BUF_SIZE, (const xmlChar *) "%d",
    i_current
  );
  sx_new_current = xmlStrdup( sx_tmp );
  xmlHashUpdateEntry(
```

```
      p_hash,
      p_graph->pArcs[i]->sxReaction,
      sx_new_current,
      (xmlHashDeallocator) xmlFree
    );
    distinct_reactions( p_node->parent, p_graph, p_hash, p_result );
  }
  else
  {
    *p_result = 0;
    return;
  }

}

/*============================================================================
// is_valid_branching().
//============================================================================*/

int
is_valid_branching(
  xmlNodePtr p_node,
  graph *p_graph,
  size_t i_level,
  WnMatrix *p_incidence
)
{

  WnMatrix *p_work, *p_submatrix;
  size_t i;
```

```
if( !has_distinct_reactions( p_node, p_graph ) )
  return 0;

p_work =
  WnMatrix__new(
    WnMatrix__getNumberOfRows( p_incidence ),
    i_level
  );

assign_matrix(
  p_node,
  i_level,
  p_incidence,
  p_work
);

p_submatrix = WnMatrix__extractMatrix( p_work, 1L, 1L, i_level, i_level );

WnMatrix__free( p_work );

for( i = 1; i <= WnMatrix__getNumberOfRows( p_submatrix ); i++ )
   if( WnMatrix__getElement( p_submatrix, i, i ) != 1. )
   {
     WnMatrix__free( p_submatrix );
     return 0;
   }

if( WnMatrix__value_is_zero( compute_determinant( p_submatrix ) ) )
{
```

```
    WnMatrix__free( p_submatrix );
    return 0;
  }
  else
  {
    WnMatrix__free( p_submatrix );
    return 1;
  }

}

/*============================================================================
// compare_branchings().
//============================================================================*/

int
compare_branchings( const void *p_data1, const void *p_data2 )
{

  const branching * p_branching1 = *( branching * const * ) p_data1;
  const branching * p_branching2 = *( branching * const * ) p_data2;

  if( p_branching1->dWeight < p_branching2->dWeight )
    return -1;
  else if( p_branching1->dWeight > p_branching2->dWeight )
    return 1;
  else
    return 0;

}
```

```
/*============================================================================
// branching_sort().
//============================================================================*/

void
branching_sort( xmlListPtr p_list )
{

  struct {
    branching **pBranchings;
    size_t iIndex;
  } work;

  size_t i, i_count;

  i_count = xmlListSize( p_list );

  work.pBranchings =
    ( branching ** )
    malloc( sizeof( branching * ) * i_count );

  work.iIndex = 0;

  xmlListWalk(
    p_list,
    (xmlListWalker) populate_branching_array,
    &work
  );
```

```
  xmlListClear( p_list );

  qsort(
    work.pBranchings,
    i_count,
    sizeof( branching * ),
    compare_branchings
  );

  for( i = 0; i < i_count; i++ )
  {
    work.pBranchings[i]->dDiff =
      work.pBranchings[i]->dWeight -
      work.pBranchings[i_count-1]->dWeight;
    work.pBranchings[i]->dDiffSign =
      work.pBranchings[i]->dWeightSign *
      work.pBranchings[i_count-1]->dWeightSign;
  }

  for( i = 0; i < i_count; i++ )
    xmlListPushFront( p_list, work.pBranchings[i] );

  free( work.pBranchings );

}

/*============================================================================
// populate_branching_array().
//============================================================================*/
```

```
int
populate_branching_array(
  branching *p_branching,
  void *p_data
)
{

  typedef struct {
    branching **pBranchings;
    size_t iIndex;
  } work;

  branching *p_new_branching;
  work *p_work = ( work * ) p_data;

  p_new_branching = (branching *) malloc( sizeof( branching ) );

  if( !p_new_branching )
  {
    fprintf( stderr, "Couldn't allocate memory.\n" );
    exit( EXIT_FAILURE );
  }

  p_new_branching->pNode = p_branching->pNode;
  p_new_branching->dWeight = p_branching->dWeight;
  p_new_branching->dWeightSign = p_branching->dWeightSign;

  p_work->pBranchings[p_work->iIndex++] = p_new_branching;

  return 1;
```

```
}

/*============================================================================
// arc_sort().
//============================================================================*/

int
arc_sort( const void *p_data1, const void *p_data2 )
{

  const arc * p_arc1 = *( arc * const * ) p_data1;
  const arc * p_arc2 = *( arc * const * ) p_data2;

  if( p_arc1->iInVertex < p_arc2->iInVertex )
    return -1;
  else if( p_arc1->iInVertex > p_arc2->iInVertex )
    return 1;
  else
    return 0;

}

/*============================================================================
// main().
//============================================================================*/

int
main(int argc, char **argv)
{
```

```
xmlDocPtr doc = NULL;       /* document pointer */
xmlNodePtr root_node = NULL; /* node pointers */
xmlListPtr p_branching_list;
graph *p_graph;
size_t i, i_level, i_arcs, i_nodes;
size_t i_1, i_2;
double d_tmp, d_tmp_2, d_total_complexity, d_entropy;
char s_reaction[BUF_SIZE], s_reactants[BUF_SIZE];
xmlChar *sx_reactants;
WnMatrix *p_incidence_matrix;
gsl_vector *p_complexities, *p_probabilities;
FILE *p_file;

if ( argc < 3 || argc > 4 ) {
  fprintf(
    stderr, "\nUsage: %s reac_file arc_file prob_file\n\n", argv[0]
  );
  fprintf(
    stderr, "  reac_file = input arc text file\n\n"
  );
  fprintf(
    stderr, "  arc_file = input arc text file\n\n"
  );
  fprintf(
    stderr,
    "  prob_file = input vectex probability text file (optional)\n\n"
  );
}
```

```
LIBXML_TEST_VERSION;

/*
 * Creates a new document, a node and set it as a root node
 */
doc = xmlNewDoc( (const xmlChar *) "1.0");
root_node = xmlNewNode(NULL, (const xmlChar *) "root");
xmlDocSetRootElement(doc, root_node);

/*=============================================================================
// Create graph.
//============================================================================*/

p_graph = (graph *) malloc( sizeof( graph ) );

if( !p_graph )
{
  fprintf( stderr, "Couldn't allocate memory for graph.\n" );
  return EXIT_FAILURE;
}

/*=============================================================================
// Store reactions.
//============================================================================*/

p_graph->pReactionHash = xmlHashCreate( 0 );

p_file = fopen( argv[1], "r" );

if( !p_file )
```

```
{
  fprintf( stderr, "Unable to open file.\n" );
  return EXIT_FAILURE;
}

while( !feof( p_file ) )
{
  fscanf( p_file, "%s  %s\n", s_reaction, s_reactants );
  sx_reactants = xmlCharStrdup( s_reactants );
  if(
      xmlHashAddEntry(
        p_graph->pReactionHash,
        (const xmlChar *) s_reaction,
        sx_reactants
      )
  )
  {
    fprintf( stderr, "Couldn't add entry.\n" );
    return EXIT_FAILURE;
  }
}

fclose( p_file );

/*============================================================================
// Store arcs.
//============================================================================*/

p_file = fopen( argv[2], "r" );
```

```
if( !p_file )
{
  fprintf( stderr, "Unable to open file.\n" );
  return EXIT_FAILURE;
}

i_arcs = 0;
i_nodes = 0;

while( !feof( p_file ) )
{
  fscanf(
    p_file,
    "%lu  %lu  %lf  %lf  %s\n",
    (unsigned long *) &i_1,
    (unsigned long *) &i_2,
    &d_tmp,
    &d_tmp_2,
    s_reaction
  );
  if( i_1 > i_nodes ) i_nodes = i_1;
  if( i_2 > i_nodes ) i_nodes = i_2;
  i_arcs++;
}

p_graph->pArcs =
  (arc **) malloc( sizeof( arc ) * i_arcs );

if( !p_graph->pArcs )
{
```

```
  fprintf( stderr, "Couldn't allocate memory for arcs.\n" );
  return EXIT_FAILURE;
}

for( i = 0; i < i_arcs; i++ )
  p_graph->pArcs[i] = (arc *) malloc( sizeof( arc ) );

p_incidence_matrix = WnMatrix__new( i_nodes, i_arcs );

rewind( p_file );

i_arcs = 0;

while( !feof( p_file ) )
{
  fscanf(
    p_file,
    "%lu  %lu  %lf  %lf  %s\n",
    (unsigned long *) &i_1,
    (unsigned long *) &i_2,
    &d_tmp,
    &d_tmp_2,
    s_reaction
  );
  p_graph->pArcs[i_arcs]->iOutVertex = i_1;
  p_graph->pArcs[i_arcs]->iInVertex = i_2;
  p_graph->pArcs[i_arcs]->dWeight = d_tmp;
  p_graph->pArcs[i_arcs]->dSign = d_tmp_2;
  p_graph->pArcs[i_arcs]->sxReaction =
     xmlCharStrdup( s_reaction );
```

```
  i_arcs++;
}

fclose( p_file );

/*=========================================================================
// Sort arcs.
//=========================================================================*/

qsort( p_graph->pArcs, i_arcs, sizeof( arc * ), arc_sort );

/*=========================================================================
// Assign incidence matrix.
//=========================================================================*/

for( i = 1; i <= i_arcs; i++ )
{

  if( p_graph->pArcs[i - 1]->iOutVertex != 0 )
    WnMatrix__assignElement(
      p_incidence_matrix,
      p_graph->pArcs[i-1]->iOutVertex,
      i,
      -1.
    );

  WnMatrix__assignElement(
    p_incidence_matrix,
    p_graph->pArcs[i-1]->iInVertex,
    i,
```

```
    1.
  );
}

p_branching_list =
  xmlListCreate(
    (xmlListDeallocator) branching_free,
    NULL
  );

for( i = 1; i <= i_arcs; i++ )
{
  i_level = 1;
  add_node(
    root_node,
    i,
    i_level,
    i_nodes + 1,
    p_incidence_matrix,
    p_graph,
    p_branching_list
  );
}

/*=========================================================================
// Sort branchings and print out.
//=========================================================================*/

branching_sort( p_branching_list );
```

```
xmlListWalk(
  p_branching_list,
  (xmlListWalker) print_list,
  p_graph
);

printf( "Number of branchings = %d\n", xmlListSize( p_branching_list ) );
printf( "Number of nodes = %d\n", (int) i_nodes );

d_total_complexity = compute_complexity( p_branching_list );

printf(
  "Complexity = ln(%g)\n",
  d_total_complexity
);

/*============================================================================
// Here compute entropy if initial probabilities provided.
//============================================================================*/

if( argc == 4 )
{

  p_file = fopen( argv[3], "r" );

  if( !p_file )
  {
     fprintf( stderr, "Unable to open file.\n" );
    return EXIT_FAILURE;
  }
```

```
if( !( p_probabilities = gsl_vector_calloc( i_nodes + 1 ) ) )
{
  fprintf( stderr, "Unable to allocate vector.\n" );
  return EXIT_FAILURE;
}

while( !feof( p_file ) )
{
  fscanf(
    p_file,
    "%lu  %lf\n",
    (unsigned long *) &i_1,
    &d_tmp
  );
  if( i_1 >= i_nodes + 1 )
  {
    fprintf( stderr, "Too large a vertex number.\n" );
    return EXIT_FAILURE;
  }
  gsl_vector_set( p_probabilities, i_1, d_tmp );
}

fclose( p_file );

p_complexities =
  compute_vertex_complexities( p_branching_list, p_graph, p_probabilities );

d_entropy = 0.;
for( i = 1;  i <= i_nodes; i++ )
```

```
  {

    d_tmp = p_complexities->data[i] / d_total_complexity;

    fprintf(
      stdout,
      "%lu %g\n",
      (unsigned long)i,
      d_tmp
    );

    d_entropy -= d_tmp * log( d_tmp );

  }

  fprintf( stdout, "Entropy = ln(%g).\n", exp( d_entropy ) );

  gsl_vector_free( p_complexities );

  gsl_vector_free( p_probabilities );

}

/*============================================================================
// Clean up.
//============================================================================*/

for( i = 0; i < i_arcs; i++ )
  arc_free( p_graph->pArcs[i] );
```

```
  free( p_graph->pArcs );

  xmlHashFree( p_graph->pReactionHash, (xmlHashDeallocator) xmlFree );

  free( p_graph );

  xmlListDelete( p_branching_list );
  WnMatrix__free( p_incidence_matrix );
  xmlFreeDoc(doc);

  xmlCleanupParser();
  xmlMemoryDump();

  return EXIT_SUCCESS;

}
```

BIBLIOGRAPHY

Arfken, G. B. and Weber, H. J. 1995. *Mathematical Methods for Physicists*. Academic Press, San Diego, CA.

Camerini, P. M., Fratta, L., and Maffioli, F. 1980. The k best spanning arborescences of a network. *Networks 10*, 91–109.

Edmonds, J. 1967. Optimum Branchings. *J. Res. Nat. Bur. Standards 71B*, 233–240.

Hix, W. R. and Meyer, B. S. 2006. Thermonuclear kinetics in astrophysics. *Nuclear Physics A 777*, 188–207.

Jordan, G. C., Gupta, S. S., and Meyer, B. S. 2003. Nuclear reactions important in α -rich freeze-outs. *Phys. Rev. C 68,* 6 (Dec.), 065801.

Jordan, G. C., Meyer, B. S., and D'Azevedo, E. 2005. Toward *in situ* Calculation of Nucleosynthesis in Supernova Models. In *Open Issues in Supernovae*, A. Mezzacappa and G. M. Fuller, Eds. World Scientific.

Meyer, B. S. and Adams, D. C. 2007. Libnucnet: A Tool for Understanding Nucleosynthesis. *70th Annual Meteoritical Society Meeting, held in August 13-17, 2007, Tucson, Arizona. Meteoritics and Planetary Science Supplement, Vol. 42, p.5215 42*, 5215–+.

Meyer, B. S., Krishnan, T. D., and Clayton, D. D. 1996. 48ca production in matter expanding from high temperature and density. *Astrophys. J. 462*, 825–838.

Meyer, B. S., Krishnan, T. D., and Clayton, D. D. 1998. Theory of quasi-equilibrium nucleosynthesis and applications to matter expanding from high temperature and density. *Astrophys. J. 498*, 808–830.

Sonzogni, A. A., Rehm, K. E., Ahmad, I., Borasi, F., Bowers, D. L., Brumwell, F., Caggiano, J., Davids, C. N., Greene, J. P., Harss, B., Heinz, A., Henderson, D., Janssens, R. V., Jiang, C. L., McMichael, G., Nolen, J., Pardo, R. C., Paul, M., Schiffer, J. P., Segel, R. E., Seweryniak, D., Siemssen, R. H., Truran, J. W., Uusitalo, J., Wiedenhöver, I., and Zabransky, B. 2000. The ^{44}Ti\(α,p\) Reaction and its Implication on the ^{44}Ti Yield in Supernovae. *Physical Review Letters 84*, 1651–1654.

The, L.-S., Clayton, D. D., Jin, L., and Meyer, B. S. 1998. Nuclear reactions governing the nucleosynthesis of 44ti. *Astrophys. J. 504*, 500–515.

Vockenhuber, C., Ouellet, C. O., The, L.-S., Buchmann, L., Caggiano, J., Chen, A. A., Crawford, H., D'Auria, J. M., Davids, B., Fogarty, L., Frekers, D., Hussein, A., Hutcheon, D. A., Kutschera, W., Laird, A. M., Lewis, R., O'Connor, E., Ottewell, D., Paul, M., Pavan, M. M., Pearson, J., Ruiz, C., Ruprecht, G., Trinczek, M., Wales, B., and

Wallner, A. 2007. Measurement of the Ca40(α,γ)Ti44 reaction relevant for supernova nucleosynthesis. *Phys. Rev. C 76,* 3 (Sept.), 035801–+.

Wallerstein, G., Iben, I., Parker, P., Boesgaard, A. M., Hale, G. M., Champagne, A. E., Barnes, C. A., Kaeppeler, F., Smith, V. V., Hoffman, R. D., Timmes, F. X., Sneden, C., Boyd, R. N., Meyer, B. S., and Lambert, D. L. 1997. Synthesis of the elements in stars : forty years of progress. *Rev. Mod. Phys. 69*, 995–1084.