A SCORE MINIMIZING VERTEX PARTITIONING ALGORITHM USING SIMULATED ANNEALING

Nicholas S Simone, B.S. Computer Science

A Thesis Presented to the Graduate Faculty
of Saint Louis University in Partial Fulfillment
of the Requirements for the Degree of
Master of Arts

2019

COMMITTEE IN CHARGE OF CANDIDACY:

Professor Darrin Speegle,
Chairperson and Advisor

Associate Professor Brody Johnson,

Professor Erin Wolf Chambers

Dedication

This thesis is dedicated to my sister Lisa. I have never known another person with such resolve and determination. Thank you for teaching me how to go beyond what I thought I could achieve. Words cannot describe how much you are loved and missed.

TABLE OF CONTENTS

List of Figures

CHAPTER 1: THE MARKOV CHAIN

1.1. Introduction

The main objective of this paper is to explain the theoretical concepts behind simulated annealing and then implement this method on known problems. Simulated annealing is an optimization algorithm that can approximate an optimal solution in situations where the actual optimal solution cannot be found quickly by brute force methods. This algorithm has been studied at least since 1983 by S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. [**7**] We will demonstrate simulated annealing by using it on three problems: vertex covering, identifying k-partite graphs, and the Erdős–Faber–Lovász Conjecture.

In the first four chapters, we will discuss the theoretical ideas behind simulated annealing. Since the bulk of the calculations are done by the computer, it is important to understand why these calculations approximate an optimal solution. In order to understand these ideas, we will first need to study Markov chains. At its heart, simulated annealing involves traversing a Markov chain in such a way that we finish at a state that provides an optimal or near-optimal solution in a reasonable amount of time.

To understand how we should traverse the chain, we will look at the Metropolis method as well. This method introduces the general algorithm needed for simulated annealing. We can then implement the Metropolis method to suit our needs. Finally, we will also show under what conditions the simulated annealing algorithm is able to approximate an optimal solution.

1.2. The Markov Chain

Definition 1.1 ([**1**, p. 5]). A ***Markov chain*** is a sequence of random variables $\{X_0, X_1, X_2, ...\}$ that has the ***Markov property***, which will be defined shortly.

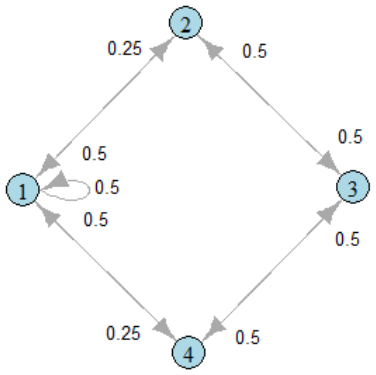


FIGURE 1.1. A Markov chain

Each random variable X_n in this sequence is the ***state*** of the Markov chain at time n. We define the set of states each X_n can take as the ***state space***.

DEFINITION 1.2 ([**1**, p. 6]). A ***state space*** $\mathcal{S}$ is a finite or countable set of states that contains all values that the random variables X_n may take. In this paper, we will only consider a finite number of states. Thus, we define $\mathcal{S} = \{1, 2, ..., N\}$ for some finite N.

We denote the probability of being in the state $i \in \mathcal{S}$ at time n as $\mathbb{P}\{X_n = i\}$. We can now define the Markov property.

DEFINITION 1.3 ([**1**, p. 8]). A sequence $\{X_0, X_1, X_2, ...\}$ has the ***Markov property*** if

$$\mathbb{P}\{X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}, ..., X_0 = i_0\} = \mathbb{P}\{X_{n+1} = i_{n+1} | X_n = i_n\}$$

In other words, the probability of moving to a new state i_{n+1} is determined solely by the state we are currently in.

To visualize a Markov chain, consider Figure 1.1. Here, our state space is $\mathcal{S} = \{1, 2, 3, 4\}$. We represent each state $i \in \mathcal{S}$ as a vertex in a directed graph. The numbers on each edge show the probability of moving from one vertex to another. In state 1, we have a 50% chance of staying at state 1, a 25% chance of moving to state 2, and a 25% chance of moving to state 4. In states 2-4, we have a 0% chance of staying in the same state, a 50% chance of moving clockwise, and a 50% chance of moving counterclockwise.

We can summarize the chain in Figure 1.1 with a ***probability transition matrix*** denoted as follows. [**1**, p. 9]

$$P = (P_{ij}) = (P(i, j))$$

Each entry in P_{ij} is the probability that the next state is j, given the current state i, so we have the following formula. [**1**, p. 9]

$$P_{ij} = \mathbb{P}\{X_{n+1} = j | X_n = i\}$$

The full probability transition matrix P for Figure 1.1 is as follows:

$$P = \begin{pmatrix} 0.5 & 0.25 & 0 & 0.25 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix}$$

Looking at the matrix above, note that the probabilities in the rows all add to 1. [**1**, p. 6] This is not a coincidence, but a requirement, since each possible state for X_{n+1} corresponds to an entry of row i. However, the columns do not need to add to 1.

As we move through the Markov chain, we can always determine the probability of moving to a given state using this matrix. [**1**, p. 8] For example, assume we know the first 5 states: $X_0 = 3$, $X_1 = 2$, $X_2 = 3$, $X_3 = 2$, and $X_4 = 1$. By the Markov property, we have that

$$\mathbb{P}\{X_5 = j | X_4 = 1, X_3 = 2, X_2 = 3, X_1 = 2, X_0 = 3\} = \mathbb{P}\{X_5 = j | X_4 = 1\}$$

Using the above matrix, we can see that

$$\mathbb{P}\{X_5 = j | X_4 = 1\} = \begin{cases} 0.5 & \text{for } j = 1 \\ 0.25 & \text{for } j = 2 \\ 0 & \text{for } j = 3 \\ 0.25 & \text{for } j = 4 \end{cases}$$

Since we have the above probabilities, we can sample the next state at time 5 in the following way. [**1**, p. 7] First, calculate a random number U_0 from the uniform distribution

between 0 and 1. Now, determine X_5 based on U_0 and the following formula

$$X_5 = \begin{cases} 1 \text{ if } 0 \leq U_0 \leq 0.5 \\ 2 \text{ if } 0.5 < U_0 \leq 0.75 \\ 4 \text{ if } 0.75 < U_0 \leq 1 \end{cases}$$

Thus, if $U_0 = 0.47$, then $X_5 = 1$. Note that there is 0% chance that we move to state 3 since states 1 and 3 are not connected. However, if we move more than one step in the Markov chain, then there is a chance we may move from state 1 to state 3. We now show how to calculate the probabilities for taking multiple steps. To do this, we first make the following definitions.

In general, we can define the probability that the chain moves to state i at time n as $\pi_n(i)$. [**1**, p. 9] Hence, we have the following formula. [**1**, p. 9]

$$\pi_n(i) = \mathbb{P}\{X_n = i\} \tag{1.1}$$

Thus, we define π_n for a finite state space $\mathcal{S} = \{1, 2, ...N\}$ as ([**1**, p. 10])

$$\pi_n = (\pi_n(1), \pi_n(2), ..., \pi_n(N))$$

Here, π_n can be thought of as a row vector giving the probability we are in each state i at time n.

Finally, we can use the law of total probability to calculate the distribution at time n given on our initial distribution π_0 (the distribution at time 0) and our probability transition matrix P. The formula is given in the following theorem. The proof of this theorem is outlined in *Stochastic Processes*, but made more rigorous here. [**1**, p. 9]

THEOREM 1.4 ([**1**, p. 9]). *Let $\mathcal{S}$ be the finite state space $\{1, 2, ..., N\}$ of a Markov chain and let P be the $N \times N$ probability transition matrix. Let π_n be the distribution of the chain at time n. Then, $\pi_n = \pi_0 P^n$.*

PROOF. First, by the Markov property, the probability transition matrix has the following property.

$$P(i, j) = \mathbb{P}\{X_{n+1} = j | X_n = i\} = \mathbb{P}\{X_1 = j | X_0 = i\}$$

We proceed by induction.

Base Case:

Let $n = 1$. Then, by the law of total probability and (1.1),

$$\begin{aligned}\pi_1(j) &= \mathbb{P}\{X_1 = j\} \\ &= \sum_{i=1}^{N} \mathbb{P}\{X_0 = i\}\mathbb{P}\{X_1 = j | X_0 = i\} \\ &= \sum_{i=1}^{N} \pi_0(i)P(i,j)\end{aligned}$$

The above equality can be expressed in matrix notation as

$$\pi_1 = \pi_0 P$$

Induction Case:

Now, assume that the following is true for all $n \leq k$. Thus,

$$\pi_k = \pi_0 P^k$$

Again, by the law of total probability and (1.1),

$$\begin{aligned}\pi_{k+1}(j) &= \mathbb{P}\{X_{k+1} = j\} \\ &= \sum_{i=1}^{N} \mathbb{P}\{X_k = i\}\mathbb{P}\{X_{k+1} = j | X_k = i\} \\ &= \sum_{i=1}^{N} \pi_k(i)P(i,j)\end{aligned}$$

The above equality can be expressed in matrix notation as

$$\pi_{k+1} = \pi_k P$$

Finally, by the induction hypothesis,

$$\pi_{k+1} = \pi_0 P^k P = \pi_0 P^{k+1}$$

□

We can now easily determine the probability matrix at any time n. For example, consider the Markov chain in Figure 1.1. We wish to know the probability distribution of the state space at time $n = 10$ given we start at state 3. To find this, we can do the following calculation. Note that we are multiplying a $1 \times N$ matrix by an $N \times N$ matrix. Thus, our result is a $1 \times N$ matrix.

$$\begin{aligned}
\pi_{10} &= \pi_0 P^{10} \\
&= (0,0,1,0) \begin{pmatrix} 0.5 & 0.25 & 0 & 0.25 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix}^{10} \\
\pi_{10} &\approx (0.43, 0.16, 0.25, 0.16)
\end{aligned}$$

Naturally, we may be curious as to what happens as n approaches infinity. For the purposes of simulated annealing, we would like the distribution π_n to approach a ***stationary distribution*** as n approaches infinity. Additionally, we would like the ***stationary distribution*** to be independent of the starting distribution π_0. We define a stationary distribution below.

DEFINITION 1.5 ([**1**, p. 12]). Given a Markov chain with probability matrix P, a ***stationary distribution*** π is a distribution such that $\pi = P\pi$.

We can think of π as the distribution π_n as n approaches infinity. However, there are some conditions we must impose on a Markov chain in order to achieve this behavior. The remainder of this chapter is dedicated to understanding them. Formally, the conditions are outlined in the Basic Limit Theorem.

1.3. The Basic Limit Theorem

THEOREM 1.6 (The Basic Limit Theorem [**1**, p. 12]). *Let* $\{X_0, X_1, X_2...\}$ *be an irreducible, aperiodic Markov chain having a stationary distribution* $\pi(\cdot)$*. Let* X_0 *have the distribution* π_0*, an arbitrary initial distribution. Then* $\lim_{n\to\infty} \pi_n(i) = \pi(i)$ *for all states* i*.*

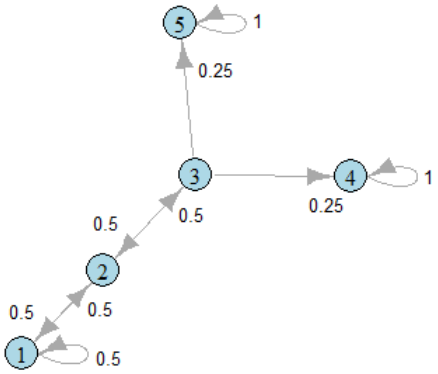


FIGURE 1.2. A Markov chain that is not irreducible

The proof of this theorem is outlined in *Stochastic Processes*. [**1**, p. 27] Note that the conditions we need to impose are that the Markov chain be ***irreducible*** and ***aperiodic***. We first look at determining if a Markov chain is irreducible. We begin by defining the terms ***accessible*** and ***communicates***.

NOTATION 1.7 ([**1**, p. 15]). We will use the shorthand "$\mathbb{P}_i$" to indicate a probability taken in a Markov chain started in state i at time 0. That is, "$\mathbb{P}_i(A)$" is shorthand for "$\mathbb{P}\{A|X_0 = i\}$." We'll also use the notation "$\mathbb{E}_i$" in an analogous way for expectation.

DEFINITION 1.8 ([**1**, p. 15]). Let i and j be two states. We say that j is ***accessible*** from i if it is possible (with positive probability) for the chain to ever visit the state j if it starts at state i. In other words:

$$\mathbb{P}_i\Big\{\bigcup_{n=0}^{\infty}\{X_n = j\}\Big\} > 0$$

DEFINITION 1.9 ([**1**, p. 16]). We say i ***communicates*** with j if j is accessible from i and i is accessible from j.

Consider the Markov chain in Figure 1.2. Note that if we are in state 1, we can move to state 3 by first stepping through state 2. Further, we can move to state 1 from state 3 by stepping through state 2 as well. Thus, we say that state 1 and state 3 communicate. However, note that if we start in state 4, we can only stay in state 4. Thus, state 4 and states 1,2,3,5 do not communicate.

We now define what it means for a Markov chain to be irreducible.

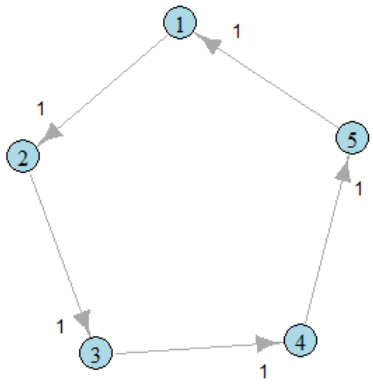


FIGURE 1.3. A Markov chain with period 5

DEFINITION 1.10 ([**1**, p. 16]). We say that the Markov chain is ***irreducible*** if all pairs of states communicate.

Thus, the Markov chain in Figure 1.2 is not irreducible. We can see why the property of being irreducible is desired. Consider what would happen as n approaches infinity in this Markov chain given different initial distributions π_0. If we start in state 4, we will stay in state 4. Likewise, if we start in state 5, we will stay in state 5. Further, if we start in states 1, 2, or 3, we have an equal chance of getting to, and then staying in, both state 4 or state 5 permanently. In fact, there is no way to tell what our final distribution would be as n approaches infinity. Thus, we cannot define a stationary distribution in this case. Hence, we will only consider those chains that are irreducible.

The second condition we require is that our Markov chains be aperiodic. To see why, consider the Markov chain in Figure 1.3. Note that all states communicate because we can start at any state and go around the chain until we reach the other. Thus, the chain is irreducible. However, we are still unable to define a stationary distribution as n approaches infinity.

The problem here is that the state we visit at time n depends on our initial distribution. [**1**, p. 16] For example, if we start at state 1, then we will only ever visit state 1 when time n is divisible by 5. Note that this repetitive visitation is called the ***period*** of a state. Recall that we want the distribution as n approaches infinity to be independent of

the initial distribution. In a Markov chain like Figure 1.3, the distribution at time n depends on the initial distribution, so we do not wish to have any periods (greater than 1) in our Markov chain. We formally define the period of a state below.

DEFINITION 1.11 ([**1**, p. 16]). Given a Markov chain $\{X_0, X_1, ...\}$ we define the ***period*** of a state i to be the greatest common divisor (gcd)

$$d_i = \gcd\{n : P^n(i,i) > 0\}$$

Now we can determine the period of a state i in Figure 1.3 by doing the following. [**1**, p. 16] First, we look at the probability transition matrix P.

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Note that P^2 is

$$P^2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

and P^5 is

$$P^5 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Thus, we have that $P^5(i,i) = 1$ for every state i. It is also easy to see this is true for any number n divisible by 5. Further, for any number m not divisible by 5, we have that $P^m(i,i) = 0$. Hence the period for all states i is 5.

It is possible that not all the states of a Markov chain have the same period. The following theorem helps solidify when they will. The proof of the theorem is in *Stochastic Processes*. [**1**, p. 17].

THEOREM 1.12 ([**1**, p. 16]). *If states i and j communicate, then $d_i = d_j$.*

Recall that we defined an irreducible Markov chain as a chain in which all pairs of states communicate with one another. Thus, by the above theorem, all states in an irreducible Markov chain have the same period. In this way, the period can be considered a property of an irreducible Markov chain as a whole, rather than just a property of a single state.

Consider the irreducible Markov chain in Figure 1.3. We have shown that all states in this chain have a period of five. By the above theorem, we now know this is not a coincidence. Further, we say the period of this irreducible Markov chain is five. This leads to our definition of aperiodic.

DEFINITION 1.13 ([**1**, p. 17]). An irreducible Markov chain is said to be ***aperiodic*** if its period is 1, and periodic otherwise.

Now consider the Markov chain in Figure 1.4. Since there are only four states, it is easy to see that the each state communicates with every other state. Thus, the Markov chain is irreducible.

We can also show that it is aperiodic. Note that since the Markov chain is irreducible, it suffices to show that one state in the chain has a period of 1 by Theorem 1.12.

First, we look at the probability transition matrix P.

$$P = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix}$$

Note that P^2 is

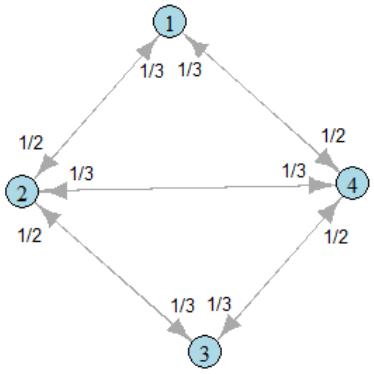


FIGURE 1.4. An irreducible, aperiodic Markov chain

$$P^2 = \begin{pmatrix} \frac{1}{3} & \frac{1}{6} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{9} & \frac{4}{9} & \frac{1}{9} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{6} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{9} & \frac{1}{3} & \frac{1}{9} & \frac{4}{9} \end{pmatrix}$$

and P^3 is

$$P^3 = \begin{pmatrix} \frac{1}{9} & \frac{7}{18} & \frac{1}{9} & \frac{7}{18} \\ \frac{7}{27} & \frac{2}{9} & \frac{7}{27} & \frac{7}{27} \\ \frac{1}{9} & \frac{7}{18} & \frac{1}{9} & \frac{7}{18} \\ \frac{7}{27} & \frac{7}{27} & \frac{7}{27} & \frac{2}{9} \end{pmatrix}$$

For both times $n = 2$ and $n = 3$, the probability of being in state 1 is greater than 0. Thus, the period of state 1 is $\gcd(2, 3) = 1$. Since the Markov chain is irreducible, all states in the chain have the same period by Theorem 1.12. Hence, the Markov chain is aperiodic.

We have now defined the two conditions the Basic Limit Theorem (Theorem 1.6) imposes on a Markov chain so that it converges to a stationary distribution. It is reasonable then to ask if an irreducible, aperiodic Markov chain always has a stationary distribution. The following theorem is proved in *Basics of Applied Stochastic Processes*. [**3**, p. 35]

THEOREM 1.14 ([**3**, p. 35]). *An irreducible Markov chain* $\{X_0, X_1, ...\}$ *has a positive stationary distribution if and only if all of its states are* ***positive recurrent****, which we will define below. In that case, the stationary distribution is unique.*

Our discussion on recurrence will be brief since the Markov chains we consider are finite; and we shall see that all finite irreducible Markov chains are positive recurrent. [**4**, p. 7] To understand the meaning of positive recurrent, we first define ***recurrent***. Informally, it means that given a state i, we will return to state i in a finite amount of time with probability 1.

As we traverse the Markov chain, we call the first time we reach a state i the "first hitting time" of state i. Formally, we define the first hitting time of state i as $T_i = \inf\{n > 0 : X_n = i\}$. [**1**, p. 19] Thus, we have the following definition.

DEFINITION 1.15 ([**1**, p. 19]). The state i is **recurrent** if $\mathbb{P}_i\{T_i < \infty\} = 1$. If i is not recurrent, it is called **transient**.

We denote the expected time at which we first come across, or "hit" state i as $\mathbb{E}_i(T_i)$.

DEFINITION 1.16 ([**1**, p. 24]). We define a recurrent state as **positive recurrent** if $\mathbb{E}_i(T_i) < \infty$ and **null recurrent** if $\mathbb{E}_i(T_i) = \infty$

For any Markov chain, if one state is transient, null recurrent, or positive recurrent, the rest of the states are as well. [**3**, p. 23] This means that we can define a Markov chain, as a whole, as transient, null recurrent, or positive recurrent. Again for our purposes, we are interested in finite Markov chains. Thus, we can make use of the following theorem.

THEOREM 1.17 ([**4**, p. 6]). *Every irreducible Markov chain with a finite state space is positive recurrent and thus has a stationary distribution.*

The proof of the theorem is in "More on Discrete-Time Markov Chains" [**4**, p. 7], but it is easy to see why we would expect this theorem to be true. If the Markov chain was not positive recurrent, there would need to be a state k in the Markov chain that is never reached even after an infinite amount of time. However, since the Markov chain is irreducible, all states communicate with positive probability. Further, every state i can reach

another state j in a finite number of steps since the Markov chain is finite and irreducible. Thus the existence of a state k that is unreachable after an infinite amount of time seems unlikely.

We now have defined all necessary conditions to apply the Basic Limit Theorem (Theorem 1.6) to a Markov chain. We now return to Figure 1.4 and discuss a stationary distribution. As we have seen, the chain in the figure is aperiodic. We can also see that all states communicate with one another, so it is irreducible. Finally, since it is irreducible and finite, it is positive recurrent. Thus, it must approach a stationary distribution π.

Specifically, we can approximate the stationary distribution of the chain in the following way. [**1**, 13] First, we start from a random state $i = 2$. Thus,

$$\pi_0 = (0, 1, 0, 0)$$

Recall that $\pi_n = \pi_0 P^n$. We then calculate the distributions at time n.

$$\begin{aligned}
\pi_1 &= (0,1,0,0)P^1 = (\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}) \\
\pi_2 &= (0,1,0,0)P^2 = (\frac{1}{9}, \frac{4}{9}, \frac{1}{9}, \frac{1}{3}) \\
&\ldots \\
\pi_{10} &= (0,1,0,0)P^{10} = (0.197, 0.303, 0.197, 0.303) \\
&\ldots \\
\pi_{1000} &= (0,1,0,0)P^{1000} = (0.2, 0.3, 0.2, 0.3)
\end{aligned}$$

Given the above result, we may suspect that the stationary distribution $\pi = (0.2, 0.3, 0.2, 0.3)$. To be certain, we can prove this is the stationary distribution by using the probability transition matrix P and Definition 1.5 and then making the following calculation.

$$\begin{aligned}
\pi &= \pi P \\
&= (0.2, 0.3, 0.2, 0.3) \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix} \\
&= (0.2, 0.3, 0.2, 0.3)
\end{aligned}$$

Looking again at Figure 1.4, this type of Markov chain is called a ***random walk***. [**1**, p. 11] A random walk is characterized by its probability transition matrix. Specifically, each entry in the matrix has the following form. [**1**, p. 53]

$$P_{rw}(i,j) = \begin{cases} \frac{1}{d(i)} \text{ if } j \in \mathcal{N}(i) \\ 0 \text{ otherwise} \end{cases} \tag{1.2}$$

Here, $d(i)$ is the degree of node i and $\mathcal{N}(i)$ is the set of all nodes that are a neighbors to i. We say two nodes are neighbors if they are joined by an edge; however, i is not considered a neighbor of itself, so $i \notin \mathcal{N}(i)$. [**1**, p. 53]

Further, the stationary distribution of the random walk has the following form. [**1**, p. 53]

$$\pi_{rw} = \frac{d(i)}{\sum_{j \in \mathcal{S}} d(j)}$$

We can see this in Figure 1.4. The number of neighbors at each node is $(2, 3, 2, 3)$ and the sum of those numbers is 10. Thus, the stationary distribution is

$$\pi = (\frac{2}{10}, \frac{3}{10}, \frac{2}{10}, \frac{3}{10}) = (0.2, 0.3, 0.2, 0.3)$$

This is the same stationary distribution that we found above. It is worth highlighting that Theorem 1.14 states that this stationary distribution is unique. Hence, we should expect the same result even though we calculated the stationary distribution using with two different methods.

In the next chapter, we will see how using a random walk can allow us to sample from certain distributions. We will then extend this by the Metropolis method and sample from any distribution. Finally, in Chapter 4, we will show that the Markov chain used in simulated annealing meets the criteria of the Basic Limit Theorem (Theorem 1.6). This allows us to use the Metropolis method to sample from such a Markov chain.

CHAPTER 2: THE METROPOLIS METHOD

2.1. The Random Walk

The Metropolis method is an algorithm for sampling from probability distributions using Markov chains pioneered by Nicholas Metropolis along with A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller. [**2**] To understand how it works, we first look at how to sample from a random walk Markov chain. Specifically, we will use the Markov chain in Figure 1.4. By using the following algorithm, we can randomly draw from the numbers $\{1, 2, 3, 4\}$ with probabilities $\{0.2, 0.3, 0.2, 0.3\}$. Thus we have a 20% chance to draw a 1, a 30% chance to draw a 2, ect. Note that these probabilities are the stationary distribution for the Markov chain in Figure 1.4. The general algorithm for sampling from a random walk is outlined below.

(1) i = a random initial state
(2) for 10,000 iterations:
(3) choose a random neighbor i with each neighbor having an equal chance of being chosen.
(4) $i = j$
(5) return(i)

Using the above algorithm is excessive for such a small state space, but it helps demonstrate the concept for larger examples.

In the above algorithm, P_{rw} is of the same form as the probability transition matrix in (1.2). To run the algorithm, we start at a random initial position. Then, we choose a random neighbor with each neighbor having an equal probability of being chosen. This is because the probability of moving to a neighbor at state i is define as $\frac{1}{d(i)}$ in (1.2). Thus, if

a state has 3 neighbors, each has a $\frac{1}{3}$ chance of being chosen. We then repeat this process for many iterations and return the final state we are in.

It is important to remember that the output of this algorithm is only *one* sample from the stationary distribution $\{0.2, 0.3, 0.2, 0.3\}$. Why did we run the loop 10,000 times? Recall the distribution π_n converges to the stationary distribution π as n approaches infinity. Thus, we need to run the algorithm for a sufficiently long period of time. For this particular example, we know the distribution we are sampling from, so we can check if 10,000 iterations is sufficient.

In the general case, we may not know what distribution we are sampling from, so we cannot determine the accuracy of running a given number of iterations. On the other hand, we can say that if a Markov chain meets the conditions of the Basic Limit Theorem (Theorem 1.6), then the longer the algorithm runs, the more closely it resembles the stationary distribution. [**1**, p. 53]

2.2. The Metroplis Method

Suppose now we wish to draw the same four numbers, but from a different distribution, $\alpha = \{0.3, 0.3, 0.1, 0.3\}$. To sample from this distribution, we need to find a new Markov chain that has this stationary distribution. The new Markov chain would still have 4 states since there are 4 numbers to sample from. However, we would need to recalculate all 16 probabilities for moving between each pair of states so that these probabilities form a 4×4 matrix P such that $\alpha = P\alpha$.

While this may seem like a difficult task, the Metropolis method describes how to modify our existing Markov chain to sample from α. Recall that the stationary distribution of the random walk has the following form. [**1**, p. 53]

$$\pi_{rw} = \frac{d(i)}{\sum_{j \in \mathcal{S}} d(j)} \tag{2.1}$$

Note that the denominator is just a normalization constant, so we can write the above equation as follows. [**1**, p. 53]

$$\pi_{rw}(i) \propto d(i)$$

To modify the distribution we introduce a multiplicative function $f(i)$ that can be evaluated for each state i. [**1**, p. 53]

$$\pi(i) \propto d(i)f(i) \tag{2.2}$$

Finally, we define the probability transition matrix.

$$P(i,j) = \begin{cases} \frac{1}{d(i)} \min\{1, \frac{f(j)}{f(i)}\} & \text{if } j \in \mathcal{N}(i) \\ 1 - \sum_{k \in \mathcal{N}(i)} P(i,k) & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Looking at the above equations, we may ask if the distribution defined in (2.2) is truly a stationary distribution for the probability matrix defined in (2.3). The following proposition and its proof are in *Stochastic Processes*. [**1**, p. 54]

PROPOSITION 2.1. *For π defined by (2.2) and $(P(i,j))$ defined by (2.3), we have $\pi P = \pi$.*

Note that the new probability transition matrix in (2.3) is different than the random walk probability transition matrix in (1.2). Thus, we must adjust our algorithm to sample from (2.3).

(1) i = a random initial state
(2) for 10,000 iterations:
(3) choose a random neighbor of i and call it j with each neighbor having an equal chance of being chosen.
(4) u=a random number from the uniform distribution [0,1]
(5) if u<min$\{1, \frac{f(j)}{f(i)}\}$
(6) $i = j$
(7) return(i)

The adjustments made to the probability transition matrix and above algorithm are the backbone of the Metropolis method. The key difference between it and the random walk algorithm is the extra step after choosing a neighbor. Call the neighbor we choose j and our current state i. Instead of simply moving to j, we decide whether or not to move with probability $\min\{1, \frac{f(j)}{f(i)}\}$. If we do not decide to move, we remain in state i. This too is a major difference since, in a random walk, we always move to a new state.

Note that if we are in state i, we do not calculate its degree $d(i)$ anywhere in the algorithm. Thus, we may worry that this process does not use the probability transition matrix as defined in (2.3). Similarly, we might be concerned that the process does not calculate the probability of staying in the same state.

The first concern can be allayed by making a simple calculation. Let P_a be the probability we choose a random neighbor j with each neighbor having equal probability of being chosen. Thus, $P_a = \frac{1}{d(i)}$. Further, let P_b be the probability that we choose to move to j. Then $P_b = \min\{1, \frac{f(j)}{f(i)}\}$. Since we do both calculations independently, the probability they both happen is

$$P_a * P_b = \frac{1}{d(i)} * \min\{1, \frac{f(j)}{f(i)}\} = P(i, j)$$

We address the second concern by making the observation that the sum in the formula for the case $j = i$ is the total probability that we move to any new state. We can call this probability P_c. Thus, the probability we stay is simply $1 - P_c$.

We now return to sampling from the distribution $\pi = \{0.3, 0.3, 0.1, 0.3\}$ using the Metropolis method. To do so, we first need to determine the function $f(i)$. This can be done by using the random walk stationary distribution $\pi_{rw} = \{0.2, 0.3, 0.2, 0.3\}$. To find $f(i)$, we divide the i^{th} entry $\pi(i)$ by $\pi_{rw}(i)$. [**1**, p. 55] Thus, we define $f(i)$ as the following.

$$f(i) = (1.5, 1, 0.5, 1)$$

We can now run the second algorithm and sample from the stationary distribution $\alpha = \{0.3, 0.3, 0.1, 0.3\}$. In Chapter 4, we will revisit the Metropolis Method and modify it slightly for our purposes. However, first we take a short detour in the next chapter to define several important graph theory concepts.

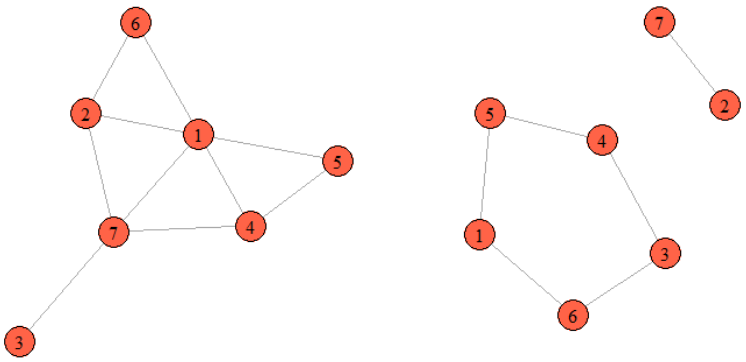
CHAPTER 3: GRAPH THEORY

3.1. States and Vertex Groups

Our ultimate goal is to run simulated annealing on vertex partitioning problems. Thus, we first introduce some essential graph theory definitions and notations. Consider an undirected graph G with a set of vertices $V(G) = \{v_1, v_2, ..., v_n\}$ and a set of edges $E(G) = \{e_1, e_2, ..., e_m\}$.

DEFINITION 3.1 ([**5**, p. 5]). A graph G is ***connected*** if for every partition of its vertex set $V(G)$ into two nonempty sets X and Y, there is an edge with one end in X and one end in Y; otherwise, the graph is ***disconnected***.

The two graphs G_1 and G_2 in Figure 3.1 show a connected graph and a disconnected graph respectively. For our purposes, we will only consider graphs that are finite, connected, and undirected.



(A) G_1, connected (B) G_2, disconnected

FIGURE 3.1. Connected and disconnected graphs

DEFINITION 3.2 ([**5**, p. 3]). Two vertices v_1 and v_2 are ***adjacent*** if they share a common edge.

For example, the vertices 1 and 2 are adjacent in graph G_1.

In the next chapter, we will also be talking about the ***state*** of a graph G. Recall that the state space $\mathcal{S}$ of a Markov chain is all possible values the random variable X_n may take. In Chapter 1 we used numbers to label each state, i.e. $\{1, 2, 3, 4\}$. In Chapter 4, each vertex will be assigned to one and only one ***vertex group*** (as defined below). Each state will be one possible permutation of this assignment. Note that the following definition is for this paper and may differ from other literature on the subject.

DEFINITION 3.3. Let G be a finite connected undirected graph and let $0 < m < \infty$. Let $\{A_1, A_2, ..., A_m\}$ be a partition of $V(G)$ such that every vertex belongs to one and only one set A_i, $0 < i \leq m$. Thus, the sets $\{A_1, A_2, ..., A_m\}$ are mutually exclusive. Each A_i is called a ***vertex group***.

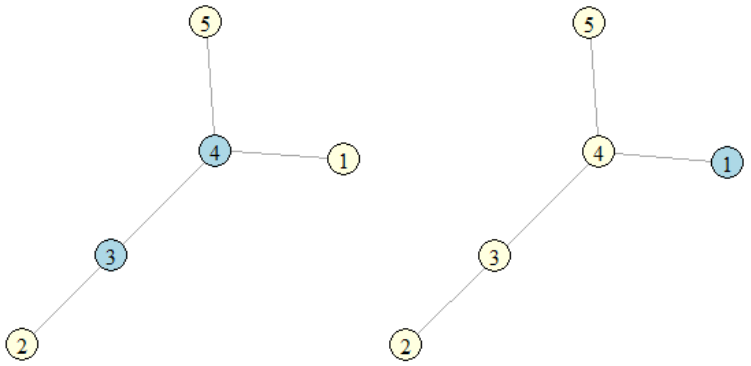
If we fix m for a graph G, then an indexing set of length m can be used for all partitions of V(G) into m mutually exclusive sets. We denote the indexing set of the vertex groups as $C(G)$.

For the sake of brevity, we call the vertex group with index $i \in C(G)$ "vertex group i."

3.2. Small Examples

In the small examples below, we partition the vertices of a graph G into a given number of colors. Thus, the indexing set, $C(G)$, is a set of finite colors. Coloring the groups helps visualize them. Consider the graph G_3 in Figure 3.2a. The blue vertices are in vertex group b and the yellow vertices are in vertex group y. These are the only two colors so $C(G_3) = \{b, y\}$ and $|C(G)| = 2$.

It is important to note that the graphs in both figures are identical except for the coloring. Together, they illustrate 2 different states of the same graph. We denote individual states by listing the vertex groups in order of the vertex number. For example, the state of graph G_3 in Figure 3.2a is (y, y, b, b, y) since vertices 1, 2, and 5 are in vertex group y and vertices 3 and 4 are in vertex group b. Similarly, the state of graph G_3 in Figure 3.2b is (b, y, y, y, y).



(A) G_3 in state (y, y, b, b, y) (B) G_3 *in state* (b, y, y, y, y)

FIGURE 3.2. Two states of G_3

Finally, note that zero vertices may be assigned to a vertex group in a given state. For example, let G be a graph such that $C(G) = \{a, b, c\}$ and $|V(G)| = 5$. It is possible G may enter the state (a, c, c, a, c) with no vertices in vertex group b.

We now define ***step*** and ***adjacent*** so that we can talk about moving between two different states. Note that the following definitions are for this paper and may differ from other literature on the subject.

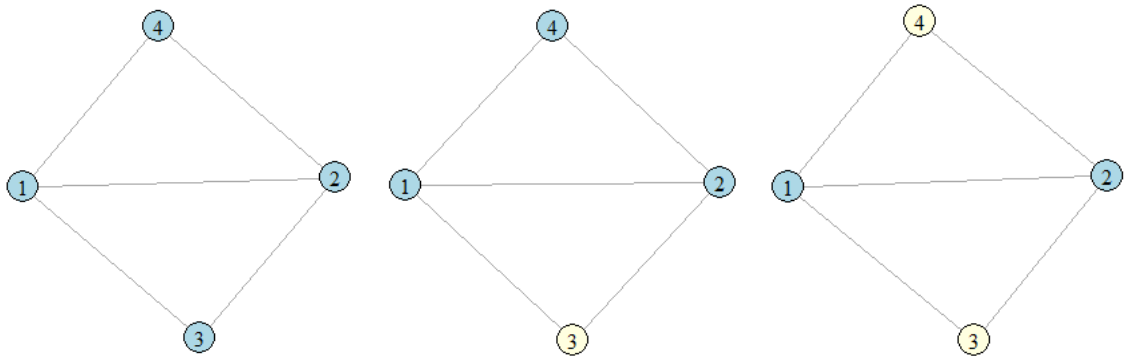
DEFINITION 3.4. Let state $i = (a_1, a_2, ..., a_n)$ and state $j = (b_1, b_2, ..., b_n)$ be two different states of a graph G where each $a_k, b_k \in C(G)$ and $1 \leq k \leq n$. The number of ***steps*** between state i and state j is the minimum number of vertex group reassignments needed so that state i = state j.
If two states are zero steps or one step away from each other, then they are ***adjacent***.

Note that we have defined the term ***adjacent*** for both two vertices and two states. We will always be explicit as to which definition we are using.

Now consider the graph G_5 where $C(G_5) = \{b, y\}$ and $V(G) = \{1, 2, 3, 4\}$. In Figure 3.3, we see three different states of G_5. If we are in state (b, b, b, b) (Figure 3.3a), then we may move to state (b, b, y, b) (Figure 3.3b) in one step. This is because we only need to reassign the vertex group of one vertex to make the state (b, b, b, b) equal to the state (b, b, y, b). Note that this also means the two states are adjacent.

In fact, we may move to states (y, b, b, b), (b, y, b, b), (b, b, y, b), or (b, b, b, y) in one step from state (b, b, b, b). Additionally, we may move to state (b, b, b, b) in 0 steps. Thus, the states (y, b, b, b), (b, y, b, b), (b, b, y, b), (b, b, b, y), and (b, b, b, b) are all adjacent to state (b, b, b, b).

However, moving to the state (b, b, y, y) (Figure 3.3c) from state (b, b, b, b) (Figure 3.3a) requires us to change the vertex group of two vertices. Thus, we say state (b, b, y, y) is two steps away from state (b, b, b, b) and the two states are not adjacent.



(A) G_5 in state (b, b, b, b) (B) G_5 in state (b, b, y, b) (C) G_5 in state (b, b, y, y)

FIGURE 3.3. Three states of G_5

Finally, it is important to note that the number of steps is the *minimum* number of reassignments required. Note that we may reassign vertex groups in the following way to move from state (b, b, b, b) to state (b, b, b, y).

(1) $(b, b, b, b) \to (b, b, y, b)$

(2) $(b, b, y, b) \to (b, b, y, y)$

(3) $(b, b, y, y) \to (b, b, b, y)$

Thus, we may be tempted to say the two states are 3 steps apart. However, the minimum number of reassignments needed is 1, so the two states are 1 step apart.

CHAPTER 4: SIMULATED ANNEALING

4.1. Admissible Markov Chains

In order to perform simulated annealing, we must first define the Markov chain on which it will act. In the previous chapter, we defined the states of a graph G and what it means to step between them. We shall now use those definitions to define an ***admissible*** Markov chain of a graph G.

Note that the following definition is local to this paper and may differ from other literature on the subject.

Definition 4.1. Let G be a finite, connected, undirected graph with a set of vertices $V(G)$. Let $|V(G)| = N$. Additionally, let $C(G)$ be a fixed indexing set for the partitions of $V(G)$ such that $2 \leq |C(G)| < \infty$. Then an ***admissible*** Markov chain of graph G is a Markov chain such that:

(1) The state space of the Markov chain is the set of all possible states of graph G given $C(G)$. We denote this as the set $S(G)$. All states $s \in S(G)$ are of the form:

$$s = (c_1, c_2, ..., c_N) \text{ where } c_i \in C(G), 1 \leq i \leq N$$

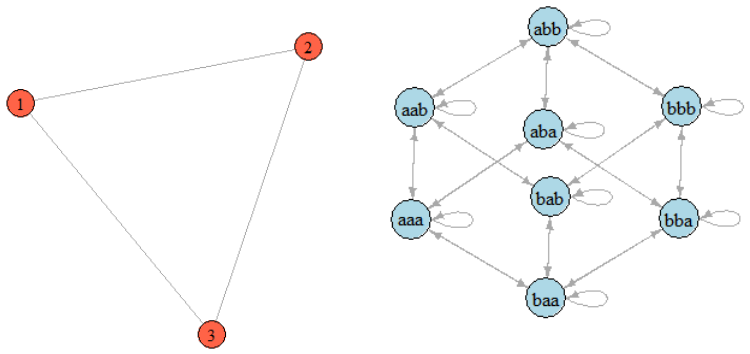
(2) If states i and j are adjacent, then there is positive probability that they will move between each other. Thus, the entry $P(i, j) > 0$ in the probability transition matrix.

(3) If states i and j are not adjacent, then there is 0 probability that they will move between each other. Thus, the entry $P(i, j) = 0$ in the probability transition matrix.

For example, consider the graph G in Figure 4.1a. Note that $|V(G)| = 3$ and $C(G) = \{a, b\}$. Thus,

$$S(G) = \{(a,a,a),(a,a,b),(a,b,a),(a,b,b),(b,a,a),(b,a,b),(b,b,a),(b,b,b)\}$$

Given this state space, we can then visualize its Markov chain in Figure 4.1b. Note that the two states (a,a,a) and (a,a,b) are connected by a double arrow since they are adjacent. However, (a,a,a) and (a,b,b) are not connected since they are two steps away from each other. It should be noted that since all states are adjacent to themselves, each state has a loop back on itself. Note that we have not labeled the probabilities of moving between states of the Markov chain for readability. However, if two states are connected by an arrow there is a nonzero positive probability of moving between those states.



(A) G

(B) *An admissible Markov chain of* G

FIGURE 4.1. Graph G and an admissible Markov chain of G

The number of states the graph in Figure 4.1a can take is $|S(G)| = 2^3 = 8$ states. In general, $|S(G)|$ is equal to the number of vertex groups raised to the number of vertices.

Our ultimate goal is to show that we can run simulated annealing on an admissible Markov chain of a graph. However, we must first show that an admissible Markov chain meets the criteria of the Basic Limit Theorem (Theorem 1.6). This leads to the following theorem and corollary. Similar results are proven in *Stochastic Processes* using different methods. [**1**, p. 71]

THEOREM 4.2. *An admissible Markov chain of graph G as defined in Definition 4.1 is irreducible, aperiodic, and positive recurrent.*

PROOF. First, we determine if an admissible Markov chain of a graph G is irreducible. Note that G is finite, so $|V(G)|$ is finite. Additionally, the number of groups $|C(G)|$ is finite by definition. Finally, the number of states $|S(G)|$ in the Markov chain is

$$|S(G)| = |C(G)|^{|V(G)|}$$

Hence, $|S(G)|$ is finite.

Let $|V(G)| = N$. We will now show how to move from one state to another in a finite number of steps.

Suppose we start at state $s_1 = (g_1, g_2, g_3, ..., g_N)$ and we wish to move to state $s_2 = (h_1, h_2, ..., h_N)$ where $g_i, h_i \in C(G)$. We can move to the state $(h_1, g_2, g_3, ..., g_N)$ from s_1 in one step by switching the group of the first vertex. We can then move to $(h_1, h_2, g_3, ..., g_N)$ in one step by switching the group of the second vertex. Continuing this process, it is clear we can move from state s_1 to state s_2 in at most N steps. Note all intermediary steps are one step apart, so there is a nonzero positive probability of moving between them.

Thus, s_1, s_2, and all the states between them communicate. Since states s_1 and s_2 are arbitrary, all states communicate. Hence, an admissible Markov chain is irreducible.

Since an admissible Markov chain is irreducible and its state space $S(G)$ is finite, then an admissible Markov chain is also positive recurrent by Theorem 1.17.

Finally, we determine if an admissible Markov chain is aperiodic. Recall that to show an irreducible Markov chain is aperiodic, we only need to prove it for one state by Theorem 1.12.

Let $|C(G)| = m$ and let $C(G) = (c_1, c_2, c_3, ..., c_m)$. We now look at state $s_0 = (c_1, c_1, c_1, ..., c_1)$. Note that the length of s_0 is $|V(G)|$, which is finite. Further, note that this particular state is in all graphs G where $|C(G)| \geq 2$ and $|V(G)| \geq 0$.

Let P be the probability transition matrix of the Markov chain. We wish to know $P(s_0, s_0)$. By definition, there is a nonzero positive probability that the s_0 remains in its current state. So let $P(s_0, s_0) = p_0$. We now wish to find $P^2(s_0, s_0)$. Note that only nonnegative numbers make up P by definition. Thus, we know that $P^2(s_0, s_0) \geq (p_0)^2 > 0$. Similarly, we know that $P^3(s_0, s_0) \geq (p_0)^3 > 0$.

By Definition 1.11, the period of state s_0 is at most $\gcd(2,3) = 1$. Thus, the period of state s_0 is 1. Hence, an admissible Markov chain is aperiodic. □

COROLLARY 4.3. *An admissible Markov chain of a graph G as defined in Definition 4.1 meets the conditions of the Basic Limit Theorem (Theorem 1.6).*

PROOF. By Theorem 4.2, the Markov chain is positive recurrent. Thus, by Theorem 1.14, the Markov chain has a positive stationary distribution. By Theorem 4.2, the Markov chain is irreducible and aperiodic as well. Thus, it satisfies the conditions for the Basic Limit Theorem (Theorem 1.6). □

Our main goal is to use this Markov chain to solve optimization problems. To do this, we first define the ***score*** of a state.

DEFINITION 4.4. [**1**, p. 57] The ***scoring function*** (or ***cost function***) $c(i)$ maps every state $i \in S(G)$ to a real finite nonnegative number. This number is the ***score*** of state i.

The main purpose of a scoring function is to evaluate a given state and determine if it is optimal. In our case, we will say state $i \in S(G)$ is optimal if $c(i) = \min_{j \in S(G)} c(j)$. If the state is not optimal, we would like the value $c(i)$ to reflect how "close" the state is to being optimal. In other words, a state with a score of 2 should be "closer" to the optimal state than a state with a score of 100. This behavior is desired, but not required.

So far, we have shown that given a finite connected undirected graph G, we can define an admissible Markov chain on G and a scoring function on the state space $S(G)$. Further, we have shown that this admissible Markov chain meets the requirements of the Basic Limit Theorem (Theorem 1.6).

4.2. Approaching the Special Distribution

Now we define the stationary distribution that we wish to sample from. Let $\mathcal{S}^*$ be a subset of $S(G)$ that contains all elements of $S(G)$ the score $\min_{i \in S(G)} c(i)$. Formally, it is as follows. [**1**, p. 58]

$$\mathcal{S}^* = \{s \in \mathcal{S} : c(s) = \min_{i \in S(G)} c(i)\}$$

We call the desired stationary distribution the "special distribution" π^*. [**1**, p. 58]

$$\pi^* = \begin{cases} \dfrac{d(i)}{\sum_{j \in \mathcal{S}^*} d(j)} & \text{if } i \in \mathcal{S}^* \\ 0 & \text{otherwise} \end{cases}$$

This distribution puts a positive probability only on states with the global minimum score. So, if we sample from π^*, we are guaranteed to draw a state with the optimal score. [**1**, p. 59] The probability transition matrix for the Markov chain would be defined the same way as in (2.3).

Sampling from this special distribution is not easy. This is because we want to move between all states, with no probability of sampling from the states that are non-optimal. Recall in Chapter 2 that we drew from the distribution $\alpha = \{0.3, 0.3, 0.1, 0.3\}$. To do this, we modified the random walk distribution $\pi_{rw} = \{0.2, 0.3, 0.2, 0.3\}$ by a function $f(i)$. This function was obtained by dividing the i^{th} entry of α by the i^{th} entry of π_{rw}. Additionally, we modified the probability transition matrix of our Markov chain to be of the form defined in (2.3).

To see the difficulty, suppose we wish to sample from the distribution $\beta = \{0, 0.3, 0.1, 0.3\}$. Note that we have set one of the probabilities to 0. Using the method from Chapter 2, we obtain $f(i) = \{0, 1, 0.5, 1\}$. However, note that $P(1, j) = 0$ if $j \neq i$ by (2.3). Thus, the new Markov chain is not irreducible since no states communicate with state 1. This means it no longer meets the conditions of the Basic Limit Theorem (Theorem 1.6). Thus drawing from a distribution using the Metropolis method where some probabilities are 0 presents some difficulty.

We will work around this by defining a function that approaches the special distribution π^* as n approaches infinity.

Let $0 < T < \infty$. We define the probability distribution $\pi_T = \{\pi_T(i) : i \in S(G)\}$ on $S(G)$ by

$$\pi_T = \frac{d(i)e^{-c(i)/T}}{G(T)} \tag{4.1}$$

where $d(i)$ is the degree of state i and

$$G(T) = \sum_{j \in S(G)} d(j) e^{-c(j)/T} \tag{4.2}$$

is a normalization constant to make it a probability distribution. [**1**, p. 58] The letter "T" stands for "temperature." It is easy to see that the definition of π_T is similar to the definition of π_{rw} in (2.1). In fact, π_T is the random walk distribution multiplied by $f(i) = e^{-c(j)/T}$ as in (2.2). Thus, the probability transition matrix is similar to the one defined in (2.3).

$$P_T(i,j) = \begin{cases} \frac{1}{d(i)} \min\{1, \frac{e^{-c(j)/T}}{e^{-c(i)/T}}\} & \text{if } j \in \mathcal{N}(i) \\ 1 - \sum_{k \in \mathcal{N}(i)} P(i,k) & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Since we will be traversing a Markov chain with the above probability transition matrix, we need to show that this Markov chain is admissible given a graph G. Additionally, note that as T changes, so does the Markov chain. Thus, we must see if the Markov chain is admissible for every $T > 0$. This leads to the following theorem. A similar result is proven in *Stochastic Processes* using different methods. [**1**, p. 71]

THEOREM 4.5. *Let G be a finite, connected, undirected graph where $|C(G)| = m$. Let $T > 0$ where $T \in \mathbb{R}$. Assume that the states of the graph G are the states in the probability transition matrix as defined in (4.3). Further, assume if state i is adjacent to state j in G and $i \neq j$, then $j \in \mathcal{N}(i)$. The Markov chain defined is admissible.*

PROOF. The first condition is met as we have defined the states of the Markov chain to be exactly the states of graph G.

To show the second condition, note that $T > 0$. Thus, we have that $P_T(i,i) > 0$ and $P_T(i,j) > 0$ if $j \in \mathcal{N}(i)$. Hence, if two states are adjacent they can move to each other with positive probability.

Further, note that if state $j \notin \mathcal{N}(i)$ and $j \neq i$, then $P_T(i,j) = 0$. Thus, the third condition is also met. □

Now we can apply Theorem 4.2 to every Markov chain when $T > 0$ and see that they follow the Basic Limit Theorem (Theorem 1.6). Thus we have the following theorem which is stated, but not proven, in *Stochastic Processes.* [**1**, p. 59]

THEOREM 4.6. *As $T \downarrow 0$, we have that π_T converges in distribution to π^*. That is, $\pi_T(i) \to \pi^*(i)$ for all $i \in S(G)$.*

PROOF. Consider the stationary distribution at temperature T given in (4.1) and (4.2). Combining them, we get the following.

$$\begin{aligned}\pi_T &= \frac{d(i)e^{-c(i)/T}}{\sum_{j\in S(G)} d(j)e^{-c(j)/T}} \\ &= \frac{d(i)}{\sum_{j\in S(G)} d(j)e^{c(i)/T-c(j)/T}}\end{aligned}$$

Next, split $S(G)$ into two disjoint sets $\mathcal{S}^*$ and $(\mathcal{S}^*)^c$. Thus, we have two sums on the bottom:

$$\lim_{T\to 0} \pi_T = \lim_{T\to 0} \frac{d(i)}{\sum_{j\in\mathcal{S}^*} d(j)e^{c(i)-c(j)/T} + \sum_{j\notin\mathcal{S}^*} d(j)e^{c(i)-c(j)/T}}$$

We now split into two cases.

Case 1: $i \in \mathcal{S}^*$

Consider the bottom left sum. Since $i, j \in \mathcal{S}^*$, then $c(i) = c(j) = \min_{k\in S(G)} c(k)$. Thus, we have $c(i) - c(j) = 0 \ \forall j \in \mathcal{S}^*$.

$$\lim_{T\to 0} \sum_{j\in\mathcal{S}^*} d(j)e^{c(i)-c(j)/T} = \lim_{T\to 0} \sum_{j\in\mathcal{S}^*} d(j)e^{0/T} = \lim_{T\to 0} \sum_{j\in\mathcal{S}^*} d(j) = \sum_{j\in\mathcal{S}^*} d(j)$$

Now, consider the bottom right sum. Note that $c(i) < c(j) \ \forall j \notin \mathcal{S}^*$. Let $n_j = c(i) - c(j)$. Thus, n_j is a real nonzero negative number.

$$\lim_{T\to 0} \sum_{j\notin\mathcal{S}^*} d(j)e^{c(i)-c(j)/T} = \lim_{T\to 0} \sum_{j\notin\mathcal{S}^*} d(j)e^{n_j/T} = \sum_{j\notin\mathcal{S}^*} d(j)e^{-\infty} = \sum_{j\notin\mathcal{S}^*} 0 = 0$$

Hence, when $i \in \mathcal{S}^*$

$$\lim_{T\to 0} \pi_T = \frac{d(i)}{\sum_{j\in\mathcal{S}^*} d(j)}$$

Case 2: $i \notin \mathcal{S}^*$

First, consider the bottom left sum. Note that $\forall j \in \mathcal{S}^*$, $c(i) > c(j)$. Define $p_j = c(i) - c(j)$. Note that p_j is a real positive nonzero number. Thus, we have the following.

$$\lim_{T\to 0} \sum_{j\in\mathcal{S}^*} d(j)e^{c(i)-c(j)/T} = \lim_{T\to 0} \sum_{j\in\mathcal{S}^*} d(j)e^{p_j/T} = \sum_{j\in\mathcal{S}^*} d(j)e^{\infty} = \sum_{j\in\mathcal{S}^*} \infty = \infty$$

Now consider the bottom right sum. Since $d(j)$ is a finite positive integer and e^x is a positive function, we can define the bottom right sum as some nonnegative number s. Hence, when $i \notin \mathcal{S}^*$

$$\lim_{T\to 0} \pi_T = \lim_{T\to 0} \frac{d(i)}{\sum\limits_{j\in\mathcal{S}^*} d(j)e^{c(i)-c(j)/T} + \sum\limits_{j\in\mathcal{S}^{*`}} d(j)e^{c(i)-c(j)/T}}$$

$$\lim_{T\to 0} \pi_T = \frac{d(i)}{\infty + s}$$

$$\lim_{T\to 0} \pi_T = 0$$

Putting both cases together, we have that $\lim_{T\to 0} \pi_T = \dfrac{d(i)}{\sum_{j\in\mathcal{S}^*} d(j)}$ when $i \in \mathcal{S}^*$ and $\lim_{T\to 0} \pi_T = 0$ otherwise. □

We have now shown that the stationary distributions π_T converge in distribution to π^* as T approaches 0. However, in practice, we do not expect to sample directly from the stationary distribution π_T since π_T is the limit of an infinite sequence of distributions. Thus, we still need to show that we can converge to the special distribution π^* without actually ever reaching any π_T distributions. We prove this in the following theorem. Note that a similar result is proven in *Stochastic Processes* through different methods. [**1**, p. 71]

THEOREM 4.7. *There exists a sequence $T_1, T_2, ...$ such that $T_k > 0$, $n_k < \infty$ $\forall k$, and*

$$\lim_{k\to\infty} (\pi_{T_k})_{n_k}(i) = \pi^*$$

PROOF. Let $\varepsilon > 0$. By Theorem 4.6 and the definition of convergence, we may choose a $T_1 > 0$ such that $\forall i \in S(G)$,

$$|\pi_{T_1}(i) - \pi^*(i)| < \frac{\varepsilon}{4}$$

By the Basic Limit Theorem (Theorem 1.6) and the definition of convergence, $\exists n_1 \in \mathbb{N}$ such that $\forall i \in S(G)$,

$$|(\pi_{T_1})_{n_1}(i) - \pi_{T_1}(i)| < \frac{\varepsilon}{4}$$

By the triangle inequality,

$$\begin{aligned}|(\pi_{T_1})_{n_1}(i) - \pi_{T_1}(i) + \pi_{T_1}(i) - \pi^*(i)| &< |(\pi_{T_1})_{n_1}(i) - \pi_{T_1}(i)| + |\pi_{T_1}(i) - \pi^*(i)| \\ &< \frac{\varepsilon}{4} + \frac{\varepsilon}{4}\end{aligned}$$

Thus, for $k = 1$, there is a $T_1 > 0$ and $n_1 < \infty$ such that

$$|(\pi_{T_1})_{n_1}(i) - \pi^*(i)| < \frac{\varepsilon}{2}$$

Now, by Theorem 4.6 and the definition of convergence, $\exists T_2$ such that $T_1 > T_2 > 0$ and $\forall i \in S(G)$,

$$|\pi_{T_2}(i) - \pi^*(i)| < \frac{\varepsilon}{8}$$

By the Basic Limit Theorem (Theorem 1.6) and the definition of convergence, $\exists n_2 \in \mathbb{N}$ such that $n_1 < n_2 < \infty$ and $\forall i \in S(G)$,

$$|(\pi_{T_2})_{n_2}(i) - \pi_{T_2}(i)| < \frac{\varepsilon}{8}$$

By the triangle inequality,

$$\begin{aligned}|(\pi_{T_2})_{n_2}(i) - \pi_{T_2}(i) + \pi_{T_2}(i) - \pi^*(i)| &< |(\pi_{T_2})_{n_2}(i) - \pi_{T_2}(i)| + |\pi_{T_2}(i) - \pi^*(i)| \\ &< \frac{\varepsilon}{8} + \frac{\varepsilon}{8}\end{aligned}$$

Thus, for $k = 2$, $\exists T_2$ and n_2 such that $T_1 > T_2 > 0$ and $n_1 < n_2 < \infty$ and

$$|(\pi_{T_2})_{n_2}(i) - \pi^*(i)| < \frac{\varepsilon}{4}$$

If we repeat this process k number of times, then we make a sequence $T_1, T_2, .., T_k$ such that $T_1 > ... > T_{k-1} > T_k > 0$ and $n_1 < ... < n_{k-1} < n_k < \infty$ and

$$|(\pi_{T_k})_{n_k}(i) - \pi^*(i)| < \frac{\varepsilon}{2^k}$$

Finally, as $k \to \infty$, $\frac{\varepsilon}{2^k} \to 0$. Thus,

$$\lim_{k\to\infty} (\pi_{T_k})_{n_k}(i) = \pi^*$$

□

The above theorem proves there exists of a sequence of temperatures, but does not provide a method for finding such a sequence. It does however, provide an outline for the simulated annealing process. The name annealing come from the real life process of slowly cooling metal to remove internal stresses. The idea is that at high temperatures, the atoms have enough energy to even out their structure. As the temperature cools, the atoms settle in a new position that is free of internal stress.

Simulated annealing is a similar process. At high values of T, the algorithm steps through the Markov chain with ease, looking for an optimal solution. As T lowers, the algorithm begins to settle near or upon an optimal solution. The sequence of temperatures we choose is called the ***cooling schedule***. We will see in the next chapter how the choice of cooling schedule affects the performance of the algorithm.

4.3. The Simulated Annealing Algorithm

In general we can consider the cooling schedule to be a function $T(k)$ where k is the iteration the algorithm is currently on. With this defined, we can alter the Metropolis method algorithm from Chapter 2 to sample from the special distribution S^*. [**1**, p. 62]

(1) Define the function $T(k)$
(2) $k = 1$
(3) $i =$ a random initial state
(4) while $k < 10,000$
(5) choose a random neighbor j with each neighbor having an equal chance of being chosen
(6) u=a random number from the uniform distribution [0,1]
(7) if u<min$\{1, \exp^{\frac{c(i)-c(j)}{T(k)}}\}$
(8) $i = j$

(9) k=k+1

(10) return(i)

The two major changes we have made are in defining a cooling schedule $T(k)$ and using the scoring functions in step 7. We do the latter to follow the probability matrix defined in (4.3). Note that is possible to optimize by finding the highest score. However, the above algorithm, and savi in particular, is designed to find the lowest score. In the next three chapters, we will apply this algorithm to three vertex partitioning problems in graph theory.

CHAPTER 5: SAVI AND VERTEX COVERING

5.1. Introduction

The next three chapters outline three different vertex partitioning problems. For each problem, we use simulated annealing to find an approximate solutions. The R-package savi was written in order to perform different cooling schedules for simulated annealing. [**13**] Our goal is to show that using simulated annealing will net better results than a greedy algorithm. Recall that during each step, simulated annealing makes a choice of whether or not to move to a state with a higher score. The greedy algorithm is a simple process that never moves to a state with a higher score and only moves to a state with an equal or lower score.

The main reason that a greedy algorithm may fail is due to the existence of "strict local minimum states" in the a given graph G. These are states such that all adjacent states have a higher score, but whose score is not the global minimum. Such local minima cause the greedy algorithm to "get stuck" in this state and not progress. We shall see that the flexibility provided by simulated annealing allows us to escape strict local minimum states.

Note that the scoring functions and theorems stated in the next three chapters are for this paper and may differ from other literature on the subject. Further, their proofs are the original work of this paper unless otherwise cited or stated.

5.2. The Scoring Function

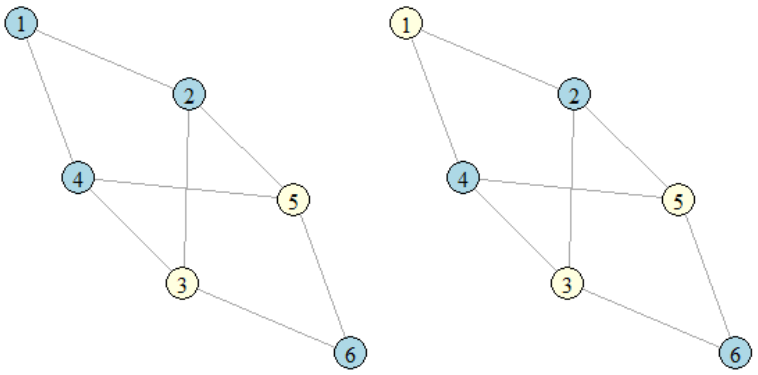
In this chapter, we are interested in finding the minimum ***covering*** for a graph G. We will show how simulated annealing helps with this process and why the greedy algorithm is insufficient. First, we define the problem.

Definition 5.1 ([**5**, p. 201]). A ***covering*** of a graph G is a subset $V_c \subseteq V(G)$ such that every edge of G has at least one end in V_c.

We wish to find the minimum cardinality of V_c. The complexity of this problem was first proven to be NP-complete by R. M. Karp. [**8**, p. 97] This makes it an excellent candidate for approximate optimization since finding the true optimal solution is very difficult.

An easy way to think of this problem is to imagine a maze of hallways. [**6**] A company wishes to know the minimum number of lights needed to illuminate all the hallways. Thus, we consider one group the "on" group and the other group the "off" group. If at least one edge is between two vertices that are "off," we say the graph is not covered since there is a hallway that is not lit.

Consider the two states of graph G in Figure 5.1. Here, we color the "on" vertices as yellow and the "off" vertices as blue. G is not covered in Figure 5.1a since both vertices 1 and 2 are "off" and there is an edge between them. The edge between vertices 1 and 4 has the same issue. However, in Figure 5.1b, every edge has at least one vertex that is "on." Thus, G is covered.



(A) *G is not covered* (B) *G is covered*

FIGURE 5.1. Vertex covering examples

We now define a scoring function for all possible states of graph G. Let every vertex in G belong to either the "on" group or the "off" group. If an edge is between two "off" vertices, then we say that edge is not covered. The score of a state i is the sum of the number of vertices in the "on" group and the number of edges not covered times 1.1. Formally,

$$c(i) = |\text{vertices "on"}| + 1.1(|\text{edges not covered}|) \tag{5.1}$$

LEMMA 5.2. *Let G be a finite connected undirected graph where $C(G) = \{on, off\}$. Let $V_{on} \subseteq V(G)$ be the set of vertices in the "on" vertex group and $V_{off} \subseteq V(G)$ be the set of vertices in the "off" vertex group. Let $c(i)$ be the scoring function defined in (5.1). Let $c(s_0) = min_{j \in S(G)} c(j)$. Then when G is in state s_0, V_{on} covers G.*

PROOF. Let G be in state s_0 where $c(s_0) = \min_{j \in S(G)} c(j)$. Assume to the contrary that V_{on} is not a cover of G. Then there is an edge between two vertices in V_{off}. If we move one of these vertices to V_{on}, then we have a new state s_1.

Since s_1 has one less uncovered edge than s_0 and one more vertex in V_{on} than s_0,

$$c(s_1) = c(s_0) - 1.1 + 1$$

However, this means that $c(s_1) < c(s_0)$, a contradiction. □

THEOREM 5.3. *Let G, V_{on}, V_{off}, $c(i)$, and s_0 be as defined in Lemma 5.2. When G is in state s_0, V_{on} has minimum cardinality.*

PROOF. Assume to the contrary that there exists a set V_1 that covers G and $|V_1| < |V_{on}|$. Call this new state where V_1 is the set of all *on* vertices s_1.

If V_1 covers all of G, then $c(s_1) = |V_1|$ since there are 0 edges not covered. By 5.2, V_{on} covers all of G as well. Thus, $c(s_0) = |V_{on}|$.

Since $|V_1| < |V_{on}|$, then $c(s_1) < c(s_0)$. Since s_0 is the minimal score, this is a contradiction. □

The two above proofs show that this scoring function maps the optimal state of G to the lowest score. The implementation of this scoring function is included in the "Savi and Vertex Covering" vignette. [**13**]

5.3. Strict Local Minimum States

We now examine why simulated annealing is superior to the greedy algorithm for this problem. Recall that one of properties of annealing is that the algorithm may choose to move to a state with a higher score. This is important since it is possible that a state exists such that all states 1 step away have a strictly lower score.

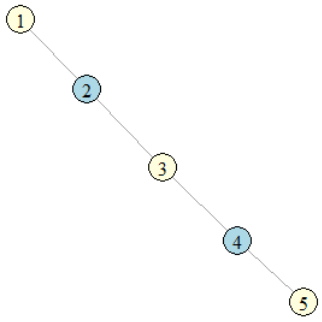


FIGURE 5.2. A strict local minimum state with score 3

Take for instance the graph in Figure 5.2. Here, yellow is the "on" group, while blue is the "off" group.

Using our scoring function (5.1), we calculate that the global minimum state has a score of 2 and the score of its current state is 3. However, it is impossible to move to a state with a lower score in one step. Note switching one of the terminal yellow vertices blue would result in a score of 3.1. Further, turning on one of the blue vertices yellow would result in a score of 4. Finally, turning the middle vertex blue increases the score to 4.2.

If we start in this state, the greedy algorithm will never move to another state since all states within one step have higher scores. We can use savi to see this in action. In Figure 5.3 is a plot of the score after running the greedy algorithm on the above graph. As you can see, the score of the current state does not change since the algorithm cannot leave the current state without moving to a higher score.

However, in Figure 5.4 we used simulated annealing. We can see from this plot that the true global minimum state has a score of 2. Note that at higher temperatures, the algorithm was more accepting of "worse" scores. However, as the temperature approached 0, it settled onto the optimal solution. For this example, the simulated annealing followed the adaptive cooling schedule. We will discuss the various cooling schedules in more detail at the end of the chapter.

DEFINITION 5.4. Let i be a state in an Markov chain of a graph G. Let $c(k)$ be a scoring function. If $c(i) < c(j)$ for every state j one step away from i, then the state i is a called a

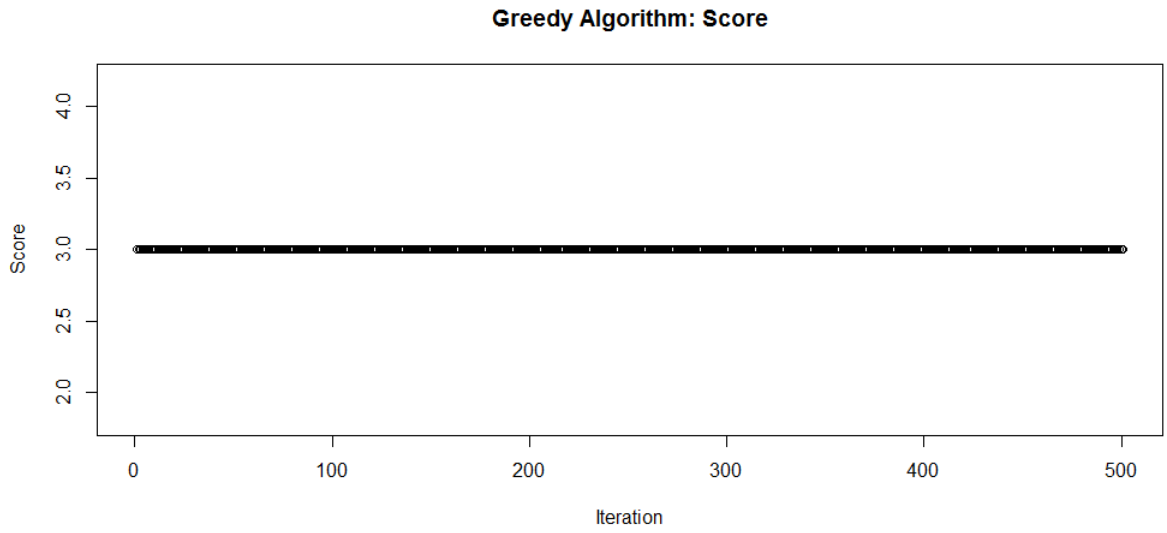


FIGURE 5.3. The greedy algorithm remains at a score of 3

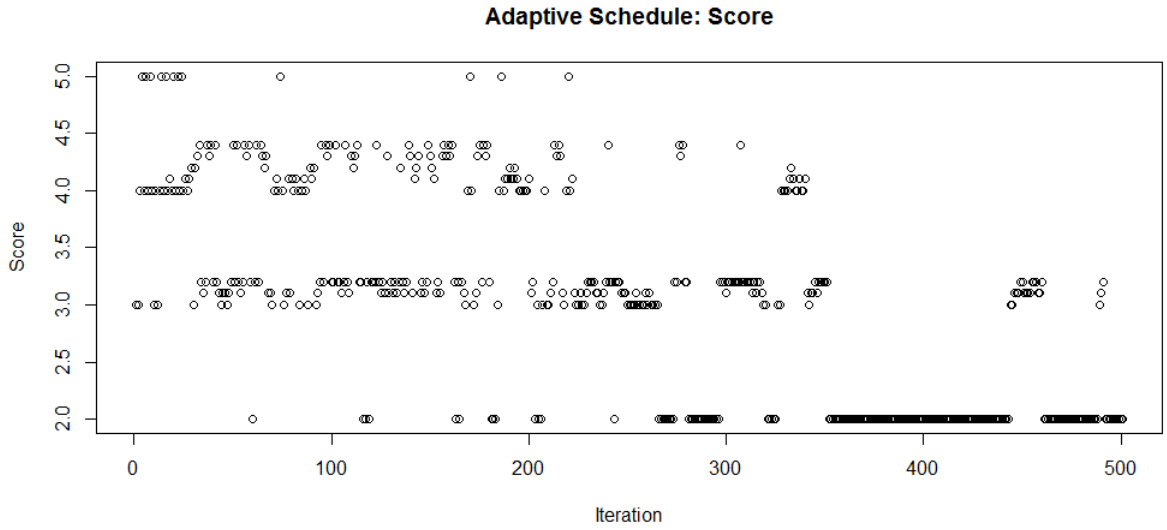


FIGURE 5.4. Simulated annealing moves out of the strict local minimum state

strict local minimum state. If $c(i) \leq c(j)$ for every state j one step away from i, then the state i is a called a ***local minimum state***. If $c(i) \leq c(j)$ for all $j \in S(G)$ the state i is called a ***global minimum state***.

The motivation behind defining a strict local minimum state in addition to a local minimum state will become apparent as we continue our discussion. Specifically, we will look at graphs which have local minimum states, but no strict local minimum states.

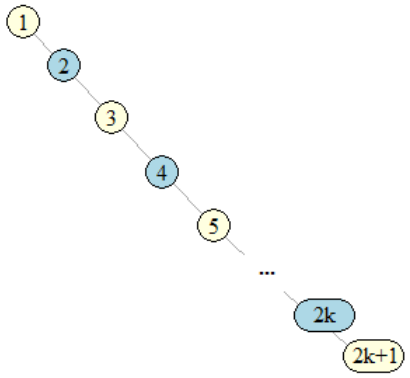


FIGURE 5.5. G in a strict local minimum state with $2k + 1$ vertices

Note that in the previous example we could not move to a state with a score *strictly* less than the score of the current state. We will now make a general statement about the existence of strict local minimum states for the vector covering problem.

THEOREM 5.5. *An infinite number of graphs contain at least one strict local minimum state for the vertex covering problem.*

PROOF. This is an extension of the example in Figure 5.2. Consider the graph G in Figure 5.5. Here, $|C(G)| = 2$ and $|V(G)| = 2k + 1$ where $k \geq 2$, $k \in \mathbb{N}$. Additionally, let $c(i)$ be the scoring function defined in (5.1).

Finally, let the coloring in Figure 5.2 be the state s_l. We will show s_l is a strict local minimum state.

First, note that if either terminal vertex is switched to blue, then the new state will have a score that is 0.1 higher than $c(s_l)$. This is because the new state has one less yellow vertex, but one more uncovered edge.

Additionally, switching any blue vertex to yellow results in a state with a score that is 1 more than $c(s_1)$, since the new state has one more yellow vertex.

Finally, turning one of the yellow vertices blue in between two blue vertices increases the score by 1.2. This is because the new state has one less yellow vertex, but two more uncovered edges.

Thus, we cannot move to a state with a strictly lower score in one step. Hence, the graph is in a strict local minimum state.

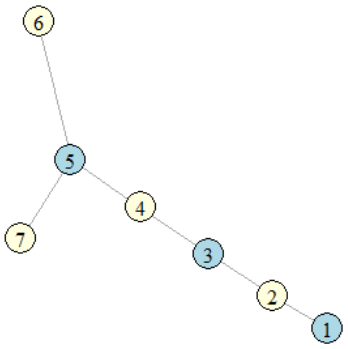


FIGURE 5.6. A graph in a strict local minimum state

Since k is arbitrary, there are an infinite number of graphs with a strict local minimum state. □

Note that not all graphs with strict local minimum states have this particular form. Consider the graph in Figure 5.6.

This graph is clearly in a strict local minimum state, but is not a straight line like the other graphs above. However, it should be noted that such problematic states were rarely found when running the examples in the next section.

5.4. Large Examples

We are now ready to look at simulated annealing on a random graph of 1000 vertices. This graph has two different vertex groups which are assigned at random. Further, the graph is finite, connected, and undirected as desired. In this case, savi attempts to find the smallest vertex cover of the graph. Listed below are the results of running the greedy algorithm and simulated annealing with 4 different cooling schedules. The same graph and initial state was used in all 5 cases. Each case was run for 10,000 iterations. The code to run these examples is in the "Savi and Vertex Covering" vignette. [**13**]

Below is a plot of the score for each iteration. Recall that at every iteration, the algorithm tries to move to a new state. Whether it does depends on the algorithm and the score of that state. Since the greedy algorithm only chooses lower scores, we see the score of the current state quickly drop, then remain the nearly constant.

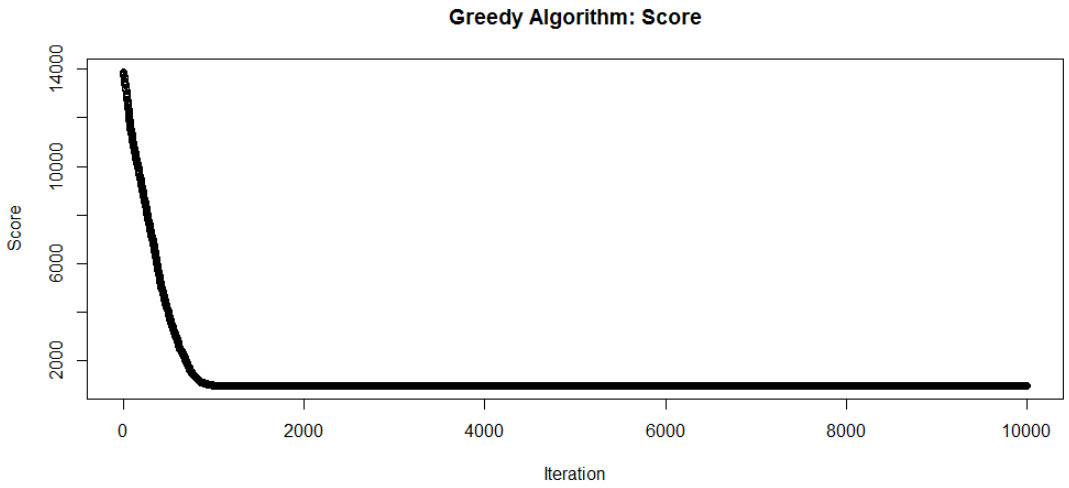


FIGURE 5.7. The greedy algorithm

The minimum score found by the greedy algorithm in the graph above was 959. Further, the minimum state found was a valid vertex cover for the graph. However, it is possible that the algorithm was trapped in a strict local minimum state and a state with a lower score exists. It is also possible that this is the minimum cover. To help determine this, we can look at the results of simulated annealing with different cooling schedules.

When running simulated annealing, we wish to slowly decrease the temperature over time so as to approach the special distribution π^*. While the process of simulated annealing is straightforward, there are many different ways we can lower the temperature. We first look at the "logarithmic" cooling schedule. Here, the temperature at each step is given by the following formula. Note that i is the current iteration, so the temperature lowers with each iteration.

$$T = \frac{\alpha}{\beta + i} \tag{5.2}$$

Specifically, we set $\alpha = 1000$ and $\beta = 1$. These parameters were chosen based on past performance and can be adjusted. The program ran for 10,000 iterations.

The lowest score that found by the logarithmic schedule was 989.2. Additionally, the minimum state found was not a valid vertex cover as two edges were left uncovered. As seen in the graph, the score dropped slower than the greedy algorithm. It also went up several times before settling on a solution. This is due to the small probability that the

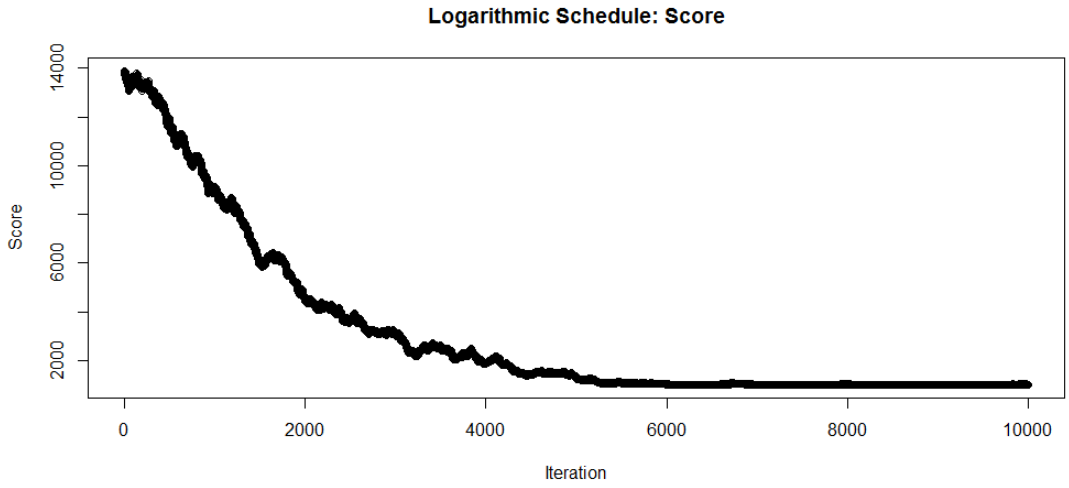


FIGURE 5.8. The logarithmic schedule

algorithm will move to states with higher scores. However, the more iterations ran, the lower the temperature. Thus, there is lower the probability of moving to a higher score as more iterations run.

Finally, recall Theorem 4.7 states that a cooling schedule for converging to the special distribution π^* always exists. It is interesting to note that for any problem, a cooling schedule in the form of (5.2) also always exists. [**1**, p. 60] However, as seen above, it performed worse than the greedy algorithm. In fact, while an interesting theoretical result, the schedule itself often does not perform well in practice.

Next, we look at a "step" cooling schedule. For this schedule we begin at a set temperature, hold on that temperature for a set number of iterations, then move down to the next temperature by a given percentage. In the following implementation, we started the temperature at 1000 and held each temperature for 100 iterations. Then we decreased the temperature by 20%. This process is done for 10,000 iterations.

The lowest score found by the step schedule was 955. Thus, this schedule performed better than the greedy algorithm and the logarithmic schedule. Further, the minimum state found was indeed a valid vertex cover.

The last schedule is the "Adaptive" cooling schedule. This schedule is similar to the step schedule. We begin at a given temperature, then remain there for a given number of iterations. However, instead of stepping down right away, the algorithm can choose to stay

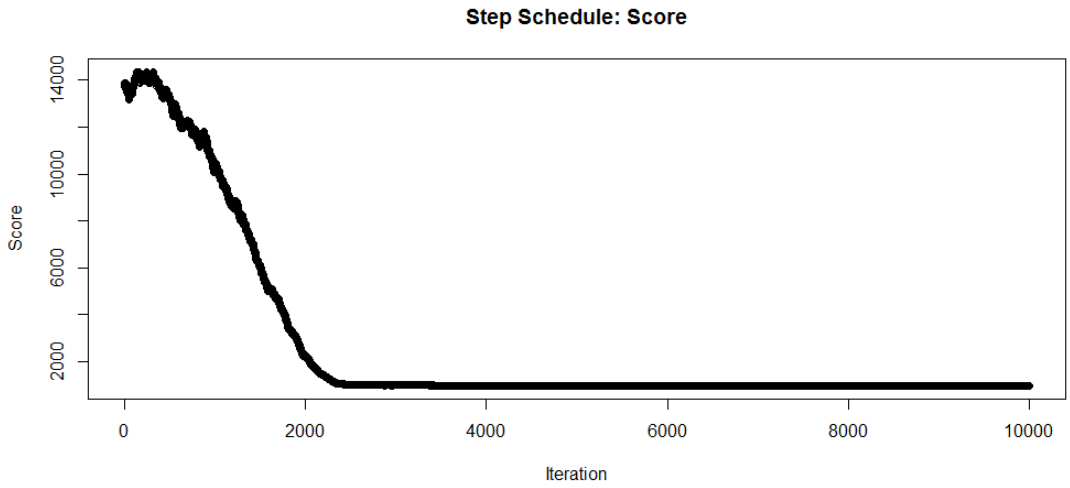


FIGURE 5.9. The step schedule

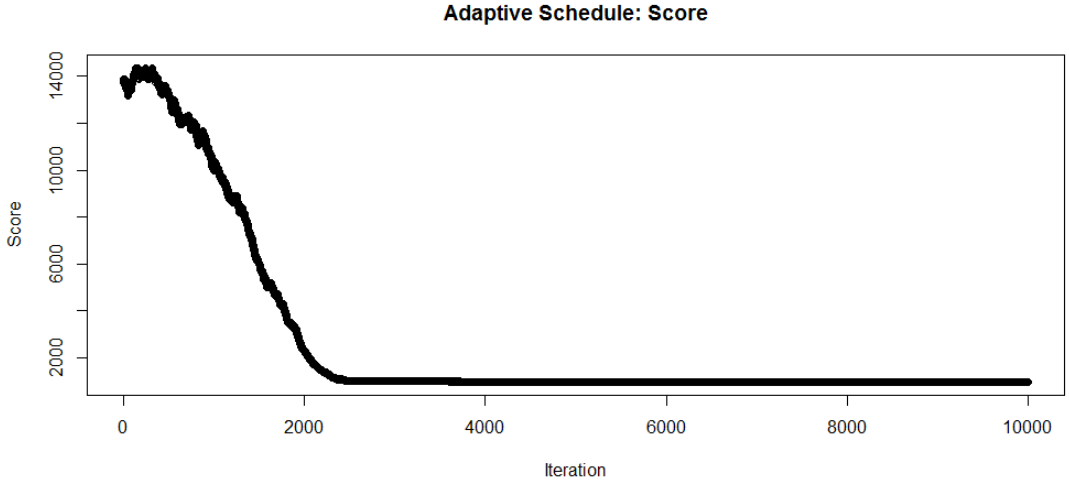


FIGURE 5.10. The adaptive schedule

at the current temperature if the score is not improving. If this is the case, we want to stay at the current temperature so the algorithm has a greater chance of moving to different and better states. To determine whether or not the score is improving, the algorithm looks at the scores of recent iterations. In this implementation, it looks at the previous 100 iterations to determine if it should lower or stay at the current temperature. For a more in-depth explanation, see the "Basics of Running Savi" vignette. [**13**] As with the step schedule, the initial temperature is 1000 and when the temperature decreased, it decreased by 20%.

The lowest score found was 953, which beats all of the above scores. Additionally, the minimum state found was a valid vertex cover. However, it should be noted that we only ran each simulation for 10,000 iterations. It is possible a state with a lower score may be found after more.

Based on the above performances on the same large graph, the adaptive cooling schedule seems to work the best out of all the cooling schedules implemented. For the next two chapters, we will focus on comparing the greedy algorithm to the adaptive cooling schedule.

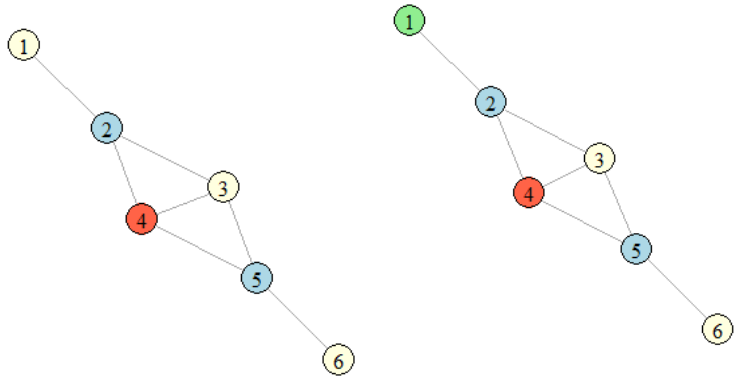
CHAPTER 6: SAVI AND MULTIPARTITE GRAPHS

6.1. The Scoring Function

The second partitioning problem is determining if a given graph G is ***k-partite***. The complexity of this problem was first proven to be NP-complete by R. M. Karp as well. [**8**, p. 97] Again, this is a good candidate for approximate optimization since finding the true optimal solution is difficult.

DEFINITION 6.1 ([**5**, p. 10]). A ***k-partite graph*** G is one whose vertex set can be partitioned into k subsets, or parts, in such a way that no edge has both ends in the same part.

Consider the graph G_1 in Figure 6.1a. Its vertices are colored with three colors: blue, yellow, and red. Note there is no edge between any two vertices of the same color. Thus, this graph is 3-partite. Additionally, you could color the graph with four colors by changing vertex 1 to green as in Figure 6.1b. Thus, this graph is also 4-partite. In general, we wish to partition the vertices into the fewest possible vertex groups.



(A) A 3-partite graph G_1 (B) A 4-partite coloring of G_1

FIGURE 6.1. Colorings of G_1

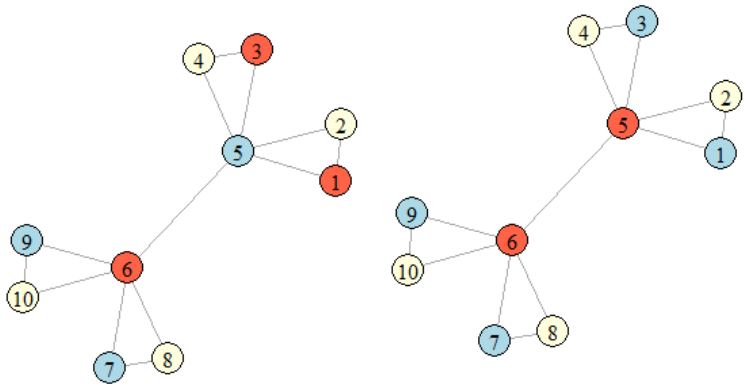
To define the scoring function, let G be a finite connected undirected graph and let $C(G)$ be the set of k vertex groups $\{g_1, g_2, g_3, .., g_k\}$ that we wish to partition $V(G)$ into. Let the set of edges connecting 2 vertices of the same group be $E^* \subseteq E(G)$. Then the scoring function of a state i in G is defined as the cardinality of E^*.

$$c(i) = |E^*| \tag{6.1}$$

It is trivial to see that when $c(i) = 0$ the graph is k-partite. It is important to note that, in practice, we run the program for a limited amount of time. Thus, if $c(i) \neq 0$ after running for a finite amount of time, we have not shown that the graph is not k-partite. Further, it is also important to remember that we can only test if a graph is k-partite for one particular k at a time. In other words, we may determine that a graph is 5-partite, but we would need to run the algorithm again to examine whether it might be 4-partite.

6.2. Strict Local Minimum States

As with the vertex covering problem, there exist strict local minimum states. In particular, the graph in Figure 6.2a is a 3-partite graph and is in a state where the score is 0. Thus, a proper coloring exists. However, the same graph in Figure 6.2b is in a state such that the score is 1 and the score cannot be lowered in one step.



(A) A 3-partite graph G_2

(B) A strict local minimum state of G_2 with a score of 1

FIGURE 6.2. States of graph G_2

Since this is a small example, it is easy to see that changing the color of any vertex in Figure 6.2b will result in a score increase of at least 1. Hence, the graph is in a strict local minimum state. Here, using simulated annealing is preferable over the greedy algorithm since there is a chance the greedy algorithm will get trapped in a strict local minimum state.

To illustrate this, we now run the greedy algorithm on the graph in Figure 6.2b. Note that in Figure 6.3, the score stays at 1. This is because the greedy algorithm started in a state such that a lower score cannot be found in one step. Thus, the algorithm does not progress.

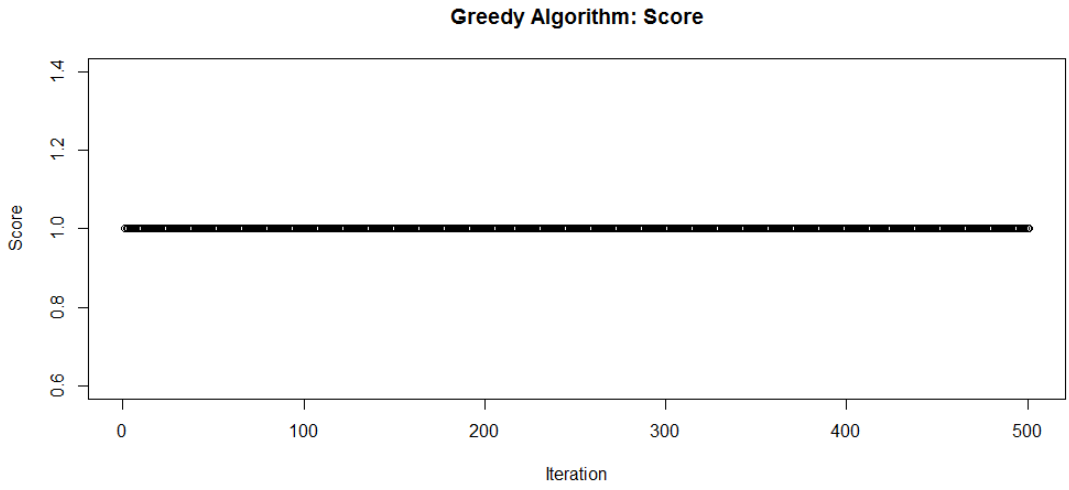


FIGURE 6.3. The greedy algorithm remains at a score of 1

We can also run simulated annealing with the adaptive cooling schedule on the graph to show that this algorithm will progress even when starting in a strict local minimum state. This progress is possible because, at higher temperatures, we have a good chance of moving to a state with a higher score. However, as the temperature gradually cools, this chance decreases and the score becomes less volatile, eventually resting around 0.

Note that the score moves up during some of the last iterations in Figure 6.4. This is because we never truly reach temperature 0. Thus, there is always a small chance of moving out of the optimal state. To account for this, we keep track of the lowest score found rather than looking at the last score found.

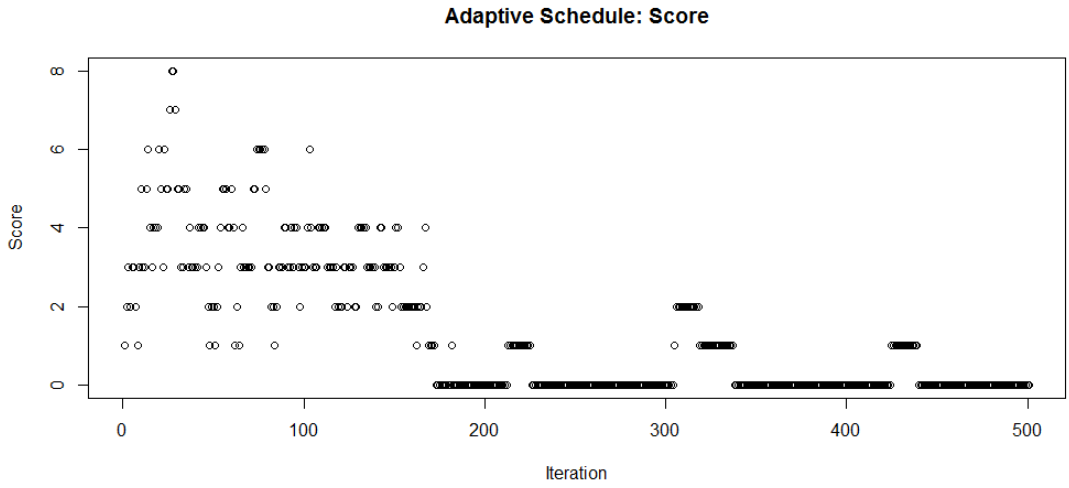


FIGURE 6.4. Simulated annealing moves out of the strict local minimum state

As with vertex covering, there are a infinite number of k-partite graphs with at least one strict local minimum state. To show this, we first prove that there exists a k-partite graph with such a state for every $k > 2$. Then, given a $k > 2$, we prove that there are an infinite number of graphs with such a state.

LEMMA 6.2. *For every $k > 2$, there exists a k-partite graph with at least 1 strict local minimum state.*

PROOF. The graph G in Figure 6.5 is the general form of the graph in Figure 6.2b.

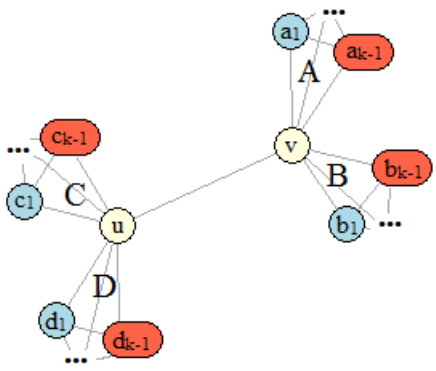


FIGURE 6.5. A k-partite graph with 4-complete subgraphs

Here, there are four k-complete subgraphs A, B, C, and D such that:

$$V(A) = \{v, a_1, a_2, ..., a_{k-1}\}$$

$$V(B) = \{v, b_1, b_2, ..., b_{k-1}\}$$

$$V(C) = \{u, c_1, c_2, ..., c_{k-1}\}$$

$$V(D) = \{u, d_1, d_2, ..., d_{k-1}\}$$

Since $k > 2$, we may assume there are at least three colors. Thus, we have that vertices u and v are yellow; vertices a_1, b_1, c_1, and d_1 are blue; and vertices $a_{k-1}, b_{k-1}, c_{k-1}$, and d_{k-1} are red. There are $k-3$ vertices remaining in each k-complete subgraph. We assign the remaining $k-3$ colors to those vertices such that every vertex in each k-complete subgraph has a unique color.

Hence, each of the 4 k-complete subgraphs are colored so that every vertex has a unique color. However, vertices v and u are the same color. Thus, the score of the state of graph G in Figure 6.5 is 1.

Without loss of generality, consider subgraph A. Note each of its k vertices is a different color. However, there are only k colors. Thus, changing the color of any one vertex will result in A having at least two vertices of the same color. Since A is a k-complete graph, at least 2 vertices of the same color will share an edge and increase the score by at least 1. Thus, we cannot change the color of any vertex in A without increasing the score.

Note that all vertices of G are part of a k-complete subgraph. Thus, changing the color of any one vertex in G will increase the score by at least 1. Hence, we cannot move to a state with a strictly lower score in one step.

However, if we change the color of a_1 and b_1 to yellow and change the color of vertex v to blue, then the score is 0. Note that this shows that G is indeed a k-partite graph.

Since we must change the vertex group of at least 2 vertices to lower the score, the state of G in Figure 6.5 is a strict local minimum state.

Since k is an arbitrary integer, there is a k-partite graph for every $k > 2$ that has at least 1 strict local minimum state. □

THEOREM 6.3. *For every $k > 2$, there are an infinite number of k-partite graphs with at least 1 strict local minimum state.*

PROOF. Fix an arbitrary $k > 2$ and consider the graph G in Figure 6.6. Note that n k-partite complete subgraphs share vertex u. When $n = 2$, the graph in Figure 6.6 is the same as the graph in Figure 6.5.

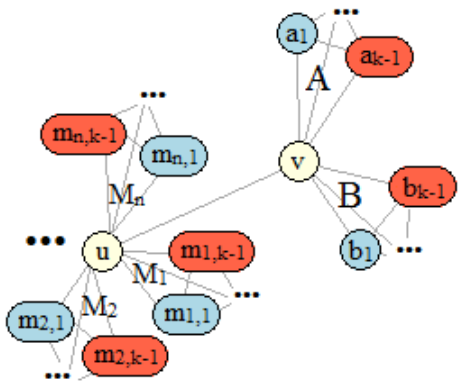


FIGURE 6.6. A k-partite graph where n k-complete subgraphs share vertex u.

Note that A and B are 2 k-complete subgraphs of G that share vertex v. Again, there are also n k-complete subgraphs of G that share vertex u. They are labeled as $M_1, M_2, ...,$ and M_n. The vertices are assigned as follows:

$$V(A) = \{v, a_1, a_2, ..., a_{k-1}\}$$

$$V(B) = \{v, b_1, b_2, ..., b_{k-1}\}$$

$$V(M_1) = \{u, m_{1,1}, m_{1,2}, ..., m_{1,k-1}\}$$

$$V(M_2) = \{u, m_{2,1}, m_{2,2}, ..., m_{2,k-1}\}$$

$$...$$

$$V(M_n) = \{u, m_{n,1}, m_{n,2}, ..., m_{n,k-1}\}$$

As with Lemma 6.2, we may assign 3 colors to the vertices in Figure 6.6 since $k > 2$. Thus, vertices u and v are yellow; vertices $a_1, b_1, m_{1,1}, m_{2,1}, ...,$ and $m_{n,1}$ are blue; and vertices $a_{k-1}, b_{k-1}, m_{1,k-1}, m_{2,k-1}, ...,$ and $m_{n,k-1}$ are red.

We will induct on the number of k-complete subgraphs attached to vertex u and show that adding another k-complete subgraph to vertex u creates a new graph H that is also k-partite and in a strict local minimum state.

Base Case:

We have shown that there exists a k-partite graph with at least 1 strict local minimum state for all $k > 2$. Particularly, we have shown that such a graph exists when there are 2 k-complete subgraphs attached to vertex u. Thus, by Lemma 6.2, Theorem 6.3 is true for $n = 2$.

Induction:

Now, assume that Theorem 6.3 is true for all $2 < n \leq t$. Let the graph G be the graph in Figure 6.6 with t k-complete subgraphs sharing vertex u. By the induction hypothesis, G is currently in a strict local minimum state.

Define graph H as the graph in Figure 6.6 with $t + 1$ k-complete subgraphs. Essentially, H is graph G with one additional k-complete subgraph sharing vertex u. Call this additional subgraph M_{t+1}.

Note that the k-complete subgraph M_{t+1} shares vertex u, which is colored yellow. Additionally, $M_{t+1,1}$ is blue and $M_{t+1,k-1}$ is red. We assign the remaining $k - 3$ colors to the other $k - 3$ vertices in M_{t+1} so that every vertex in M_{t+1} has a unique color.

Note that there are only k colors. Thus, changing the color of one vertex will result in M_{t+1} having at least two vertices of the same color. Since M_{t+1} is a k-complete graph, at least 2 vertices of the same color will share an edge and increase the score by at least 1. Thus, we cannot change the color of any vertex in M_{t+1} without increasing the score.

With the addition of M_{t+1}, all vertices of H are part of a k-complete subgraph. Thus, changing the color of any one vertex in H will increase the score by at least 1. Hence, we cannot move to a state with a strictly lower score in one step.

However, if we change the color of a_1 and b_1 to yellow and change the color of vertex v to blue, then the score is 0. Note that this shows that H is indeed a k-partite graph.

Since we must change the vertex group of at least 2 vertices to lower the score, the state of H in Figure 6.6 is a strict local minimum state.

Thus for all $n \geq 2$, the graph in Figure 6.6 is in a strict local minimum stats. So, there are infinite k-partite graphs that have at least 1 strict local minimum state.

Since k is arbitrary, there are an infinite number of such k-partite graphs for every $k > 2$. □

The results of the above proof show why the greedy algorithm is insufficient when attempting to find an optimal solution. The tendency for it to stay trapped in a strict local minimum state does not allow it to find better states further away from that strict local minimum state. Though it may seem counter-intuitive at first, moving to higher scores at certain points can eventually lower the score overall.

6.3. Large Examples

To see this in action, we will now compare the greedy algorithm and simulated annealing with the adaptive cooling schedule on a known 8-partite graph with 50 vertices. Note that the same initial state was used in both cases. Below is a plot of the score array after running the greedy algorithm for 10,000 iterations. A score of 0 was reached after about 3100 iterations, confirming the graph as 8-partite.

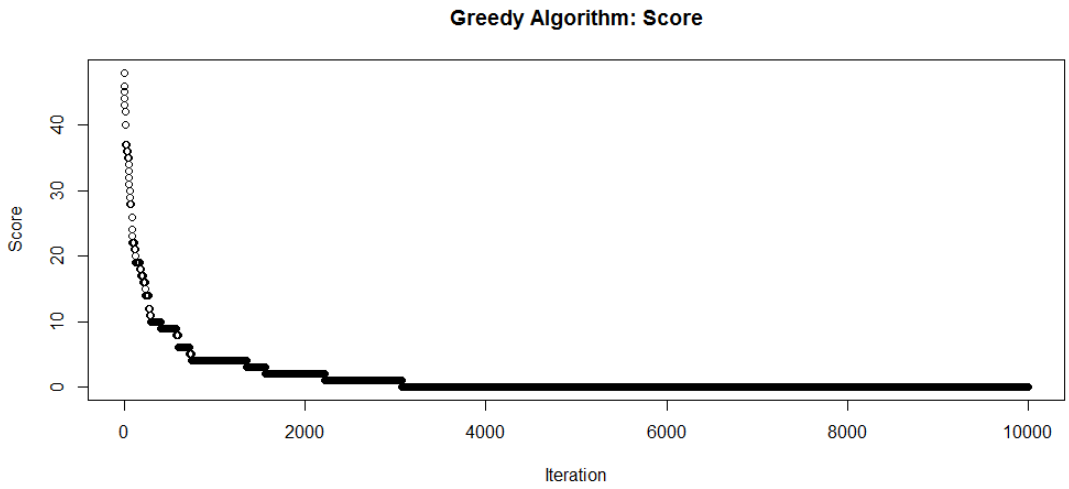


FIGURE 6.7. The greedy algorithm

Now we run simulated annealing using the adaptive cooling schedule. Below is the graph of the score array. A final score of 0 was reached after about 8600 iterations. Thus,

the greedy algorithm performed slightly better. It is important to note that there are cases like this where the greedy algorithm will outperform other methods. However, it is safer to use simulated annealing.

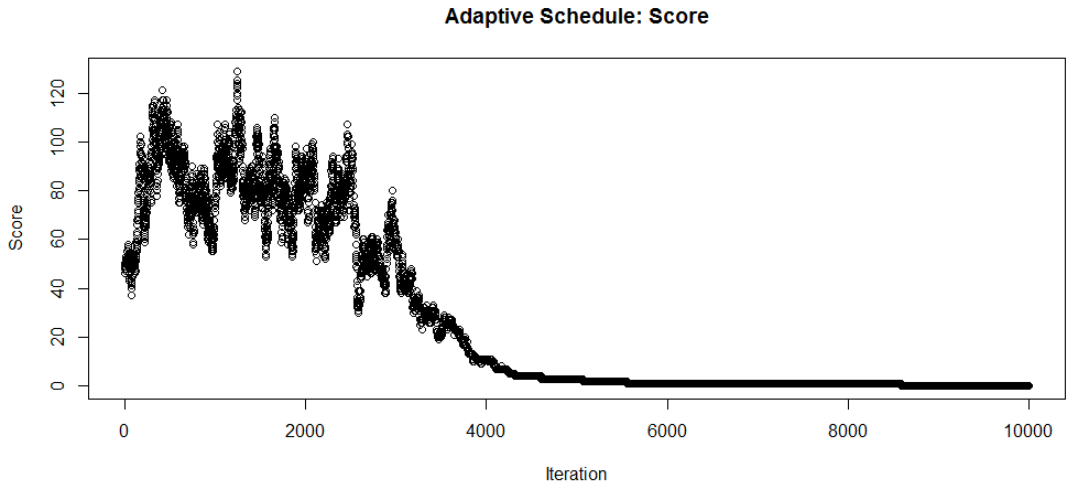


FIGURE 6.8. The adaptive schedule

CHAPTER 7: SAVI AND THE E.F.L. CONJECTURE

7.1. The Erdős–Faber–Lovász Conjecture

The Erdős–Faber–Lovász (EFL) conjecture is a special case of the previous problem and uses the same scoring function. Paul Erdős first presented the problem in his the paper "On the Combinatorial Problems I Would Most Like to See Solved" as an extension of a problem thought up by Vance Faber, László Lovász, and himself at a party. [**9**, p. 26] Formally, the EFL conjecture states the following.

Conjecture 7.1 (EFL Conjecture). *Let $k \in \mathbb{N}$ and let G be a graph such that G is connected and composed of k k-complete graphs, any pair of which share at most one vertex. Then the graph can be colored with k colors.*

If we let $k = 5$, then Figure 7.1 shows an example of a graph that conforms to the conditions of the EFL conjecture. We call such a graph an EFL-5 graph. The graph in Figure 7.1 has been colored by savi so that the score is 0. However, this is only one possible configuration of an EFL-5 graph. In order to prove the conjecture for $k = 5$, we would need to show such a coloring existed for all possible EFL-5 graphs. On the other hand, we can use savi to find a counterexample to the conjecture by searching for an EFL graph whose score is greater than 0. However, even if such a graph is found, we would need to investigate it further since savi is used to find a score *close* to the optimal score, but may not find the lowest one possible.

7.2. Local Minimum States

In the previous two chapters, we showed that simulated annealing is a better method than the greedy algorithm due to the existence of strict local minimum states. We may ask if such states exist for any EFL-k graphs.

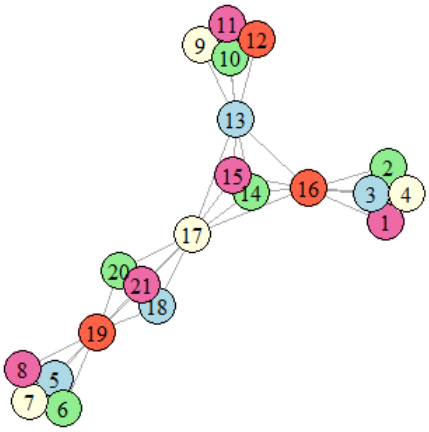
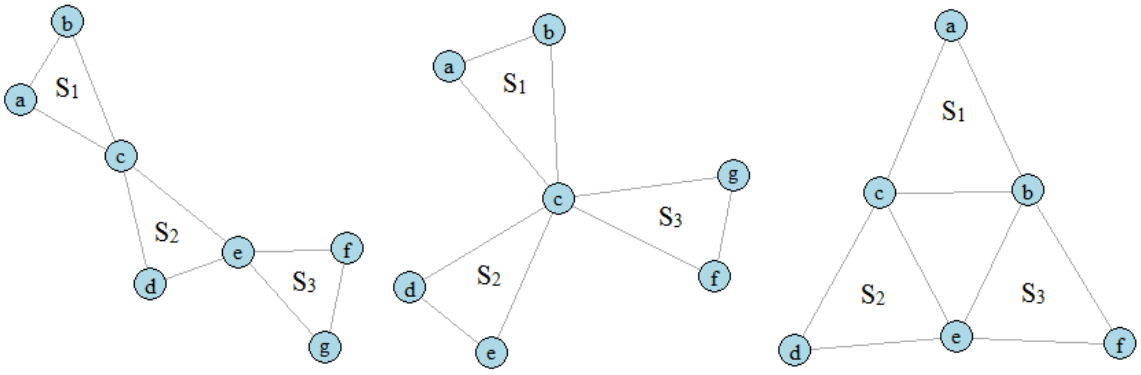


FIGURE 7.1. An EFL-5 graph colored by savi

To answer this, we first note that the proofs in the previous chapter do not apply to any EFL-k graph. Recall that the argument relies on the existence of a single edge between several complete subgraphs. However, there are no such edges between the k-complete subgraphs in an EFL-k graph.

With the following proofs, we will show that no *strict* local minimum states exist for any EFL-3 graph. However, we will see that local minimum states do exist. Additionally, we will examine some EFL-4 graphs and show that there are *probably* no strict local minimum states for those graphs either.

However, showing there are no strict local minimum states for a general EFL-k graph is beyond the scope of this thesis.



(A) EFL-3a (B) EFL-3b (C) EFL-3c

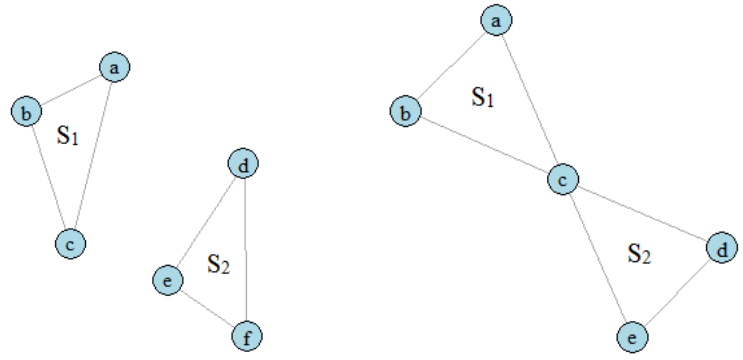
FIGURE 7.2. All EFL-3 graphs

Now, consider the three EFL-3 graphs in Figure 7.2. Note that each of the 3-complete subgraphs in the EFL-3 graphs are labeled S_1, S_2, and S_3. We will now prove the following lemma.

LEMMA 7.2. *All EFL-3 graphs are isomorphic to one of the three EFL-3 graphs in Figure 7.2.*

PROOF. We will prove this by construction. Consider the two 3-complete graphs below in Figure 7.3a. An EFL graph must be connected and two 3-complete graphs can only share one vertex. Thus, we must choose two vertices to connect.

Here, we choose vertex c from S_1 and f from S_2 to create the graph in Figure 7.3b. It does not matter which pair of vertices we choose, since any other choice will result in a graph isomorphic to the graph we created in Figure 7.3b.



(A) Two 3-complete graphs (B) The 3-complete graphs connected by vertex c

FIGURE 7.3. Merging two 3-complete graphs

Now, consider our choices for connecting another 3-complete graph, S_3, with vertices $\{f, g, h\}$ as shown in Figure 7.4. We split into two cases.

Case 1:

In this case, suppose S_3 shares one of its vertices. Without loss of generality choose vertex h. Connecting h to a, b, d, or e will result in a graph isomorphic to the graph EFL-3a. Further, connecting vertex h to c will result in the graph EFL-3b.

We have now exhausted all the possibilities of subgraph S_3 sharing one of its vertices.

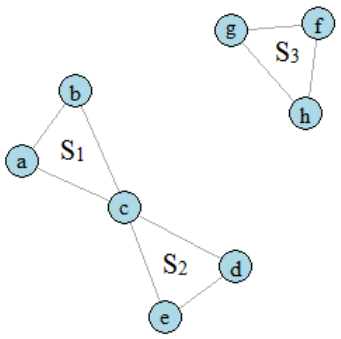


FIGURE 7.4. Connecting a third 3-complete graph.

Case 2:

Suppose S_3 shares two of its vertices. Since each pair of complete graphs may only share one vertex, S_3 must share one vertex with S_1 and another with S_2. Without loss of generality, choose vertices g and h.

Note that connecting vertex g to c is not possible since doing so would prohibit h from being attached. This is because S_3 would then share vertex c with both S_1 and S_2. Similarly, vertex h cannot connect to c either.

Thus, vertices h and g must attached to a, b, d, or e. Let h be connected e and g be attached to b. This configuration results in the graph EFL-3c.

In general, h needs to connect to one of $\{a, b\}$, or one of $\{e, f\}$. Further, once h's connections are chosen, g must choose from the opposite pair. Doing so will result in a graph isomorphic to the graph EFL-3c.

Finally, note that S_3 cannot share 3 of its vertices as there are only two 3-complete graphs and each pair may only share one vertex. Thus, the two cases have exhausted all possible configurations for an EFL-3 graph. Hence, the only possible EFL-3 graphs are those isomorphic to the graphs in Figure 7.2. □

We now return to the question of the existence of strict local minimum states for an EFL-3 graph. This leads to the following theorem.

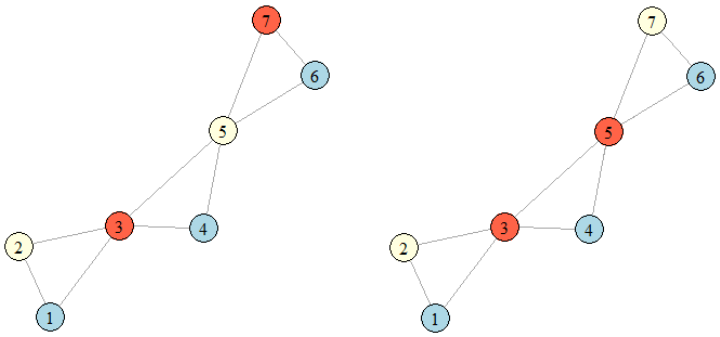
THEOREM 7.3. *There are no strict local minimum states for any EFL-3 graph.*

PROOF. By Lemma 7.2, there are only 3 different graphs we need to check. Thus, it is feasible to check every possible state to find a strict local minimum state.

Recall that the number of states in an admissible Markov chain of graph G is $|S(G)| = |C(G)|^{|V(G)|}$. Thus, the total number of states to check is $3^7 = 2187$ for EFL-3a, $3^7 = 2187$ for EFL-3b, and $3^6 = 729$ for EFL-3c.

Custom code was written in order to check every state and can be found in the vignette "EFL-3-Test." [**13**] No strict local minimum states were found. □

However, it is easy to construct local minimum states. Consider the graphs in Figure 7.5. The state of the graph in Figure 7.5a has a score of 0. Thus this EFL-3 graph does not contradict the EFL conjecture. However, consider the state of the graph in Figure 7.5b. Here there is an edge between two vertices of the same color. Note that there are no states one step away with a lower score, but there are states one step away that have the same score. Thus, this is a local minimum state, but not a strict local minimum state.



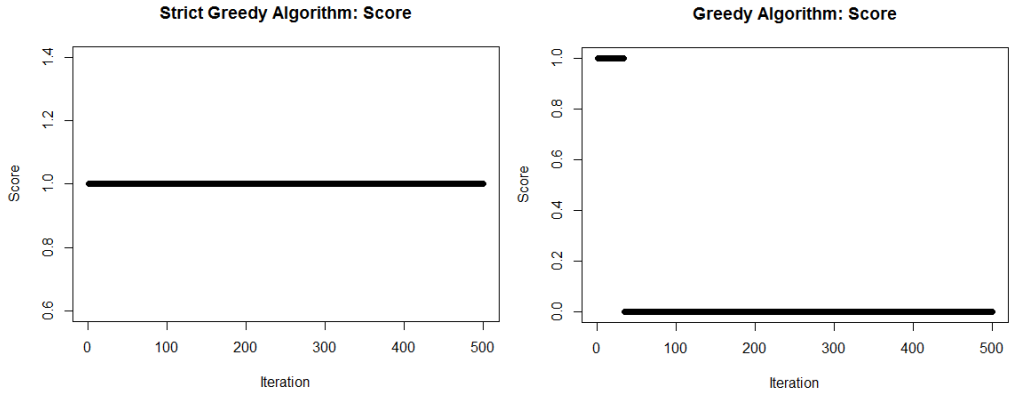
(A) Global minimum state where score=0

(B) Local Minimum State where score=1

FIGURE 7.5. Minimum States

Specifically, the score stays the same if we switch vertex 3, 4, or 5 to yellow. Changing vertex 4 to yellow won't help the situation, but changing vertex 5 to yellow will help. Note that once vertex 5 is yellow, changing vertex 7 to red lowers the score to 0 and we arrive at the state in Figure 7.5a. Thus, we have a situation where moving to a state of the same score can help find a lower score.

To see this in action, we run two versions of the greedy algorithm on the graph in Figure 7.5b. The first algorithm will only move to scores strictly less than the current score. The second algorithm will move to score less than or equal to the current score. The results can be seen in Figure 7.6.



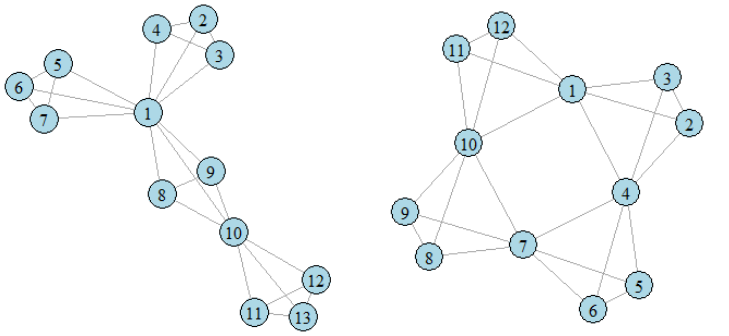
(A) The greedy algorithm (B) The strict greedy algorithm

FIGURE 7.6. Greedy algorithm vs strict greedy algorithm

Note that the strict greedy algorithm failed to find a solution while the regular greedy algorithm was able to find one. It is for this reason that we allow the greedy algorithm to move to a state with the same score by default. In fact, since there are no strict local minimums for $k = 3$, the greedy algorithm is sufficient for correctly coloring any EFL-3 graph and we do not need to use simulated annealing.

One final note we wish to make is that this strict vs non-strict issue does not affect simulated annealing. This is because there is a small probability of moving to a state with a higher score already.

We now look at EFL-4 graphs. While we cannot show that no strict local minimum states exist in any EFL-4 graph, we can show that they probably don't exist. To show this, we ran the greedy algorithm on nine EFL-4 graphs. Two such graphs are in Figure 7.7. The remaining graphs can be viewed in the vignette "EFL-4-Graphs." Each graph was ran 20 times and the score reached 0 in every case. Thus, we can reasonably conclude there are likely no graphs with a strict local minimum state when k=4. The code for this can be found in the vignette "EFL-4-Test." [**13**]



(A) *An EFL-4 graph* (B) *A second EFL-4 graph*

FIGURE 7.7. EFL-4 graph examples

7.3. Large Examples

Finally, we tested a larger graph. Here, we let $k = 14$ and made a random EFL-14 graph in savi. Note that the same initial state was used in both cases. We first ran the greedy algorithm to see if the EFL conjecture held for this graph. The score per iteration is in Figure 7.8. The greedy algorithm found a minimum score of 0 after 10,000 iterations. Thus, the conjecture is not contradicted by this EFL-14 graph.

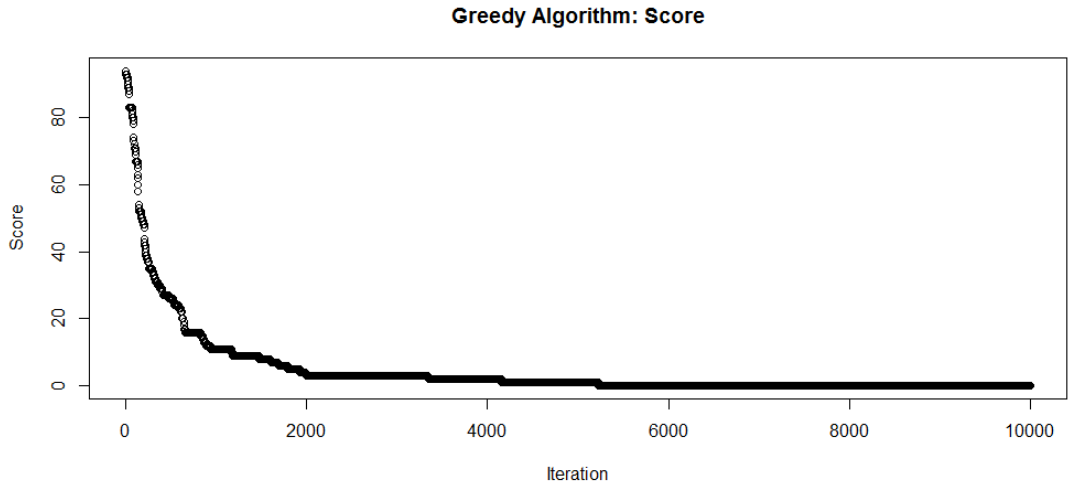


FIGURE 7.8. The greedy algorithm

Next we used simulated annealing with the adaptive cooling schedule on the same graph. After 10,000 iterations, the annealing found a score of 4. Knowing that the greedy

algorithm had found a lower score, the annealing was ran again. This time it ran for 15,000 iterations. A state with a score of 0 was found after about 14,300 iterations.

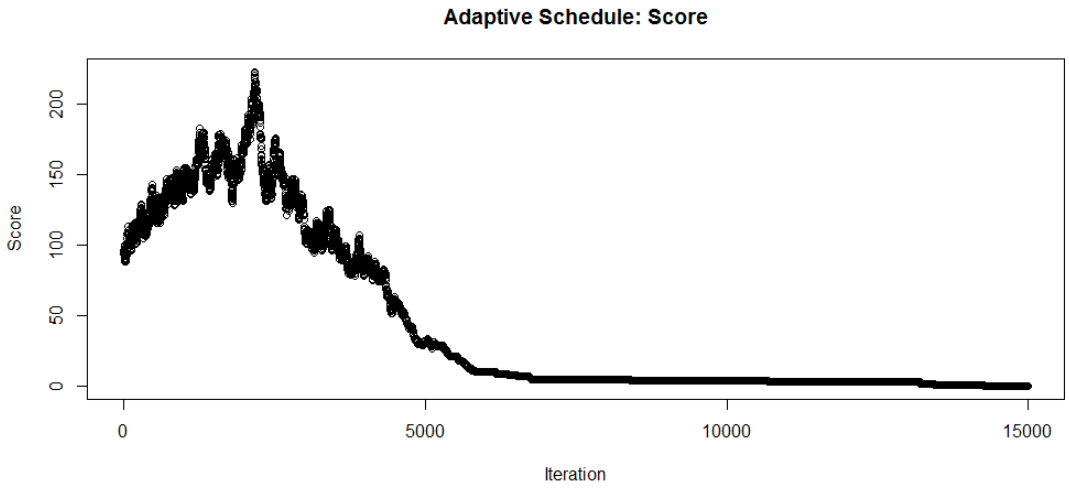


FIGURE 7.9. The adaptive schedule

Conclusion

The first goal of this paper was to understand the process of simulated annealing and show that the process can stand up to mathematical rigor. To do this, we first defined the properties of an admissible Markov chain. Then, we showed that we can apply the Basic Limit Theorem (Theorem 1.6) to such a chain by showing it was irreducible, aperiodic, and positive recurrent. Once we could apply the theorem, we then tailored the Metropolis method to our needs by having the process approach a special distribution. This tailored process was simulated annealing. Recall that the special distribution was one where only the optimal score was nonzero. Thus we showed that, eventually, the simulated annealing process would have no choice but move to a state with the optimal solution as the probability of moving to a state with a non-optimal solution would be 0.

After showing the theoretical results, we wished to apply the simulated annealing process. Thus, the second goal was to create a package in R so that the process could be carried out on different problems. Specifically we looked at vertex partitioning problems where it is not feasible to test every possible combination to find the optimal result. Further, we introduced the idea of different cooling schedules, which would determine how long the process stayed at each temperature. The testing of different schedules is important as the theory only states that such a schedule exists, but does not define one. The package savi was coded so that we could study these various cooling schedules and see how they perform on finding approximate solutions for vertex partitioning problems.

Thus, the final goal was the comparison of the cooling schedules against themselves and the greedy algorithm. To this end, it was shown that simulated annealing was superior to the greedy algorithm because of the existence of strict local minimum states. These states appeared both in the vertex covering problem and the k-partite graph problem.

Further, it was shown that an infinite number of graphs contain these strict local minimum states. Thus, we recommend using simulated annealing on such problems.

In the final chapter we looked at the Erdős–Faber–Lovász Conjecture and showed how simulated annealing could be helpful in finding a counter example to the conjecture. Additionally, we also studied the interesting group of graphs defined by the Erdos-Faber-Lovas conjecture. Letting $k = 3$, we saw that no strict local minimum states existed, but local minimum states did exist. Here we showed that it is possible for the strict greedy algorithm, which only moves to states with a strictly lower score, to get stuck in a local minimum state. However, the regular greedy algorithm, which moves to states with a lower or equal score, will not get stuck in a local minimum state. Hence, we recommend coding the greedy algorithm so that it moves between states of the same score.

Finally, throughout this thesis, the greedy algorithm was constrained to looking for a state with a higher score one step away from its current state. Because of this constraint, we were able to show that the algorithm may fail in some optimization problems. However, the proofs provided would not work with a greedy algorithm able to search for higher scores two or more steps away. Thus, a future study could explore the minimum amount of steps needed for the greedy algorithm to avoid strict local minimum states for a given problem.

Bibliography

[1] Joseph T Chang. *Stochastic Processes*. Yale University, 2007.

[2] Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. (1953). "Equation of State Calculations by Fast Computing Machines". Journal of Chemical Physics. 21 (6): 1087–1092.

[3] R. Serfozo. *Basics of Applied Stochastic Processes*. Springer-Verlag Berlin Heidelberg, 2009.

[4] Sigman, Karl. "More on Discrete-Time Markov Chains." Columbia University, 2006, http://www.columbia.edu/~ww2040/4701Sum07/4701-06-Notes-MCII.pdf

[5] Bondy, J.A. and Murty, U.S.R. *Graph Theory*. Springer, 2008.

[6] Vertex Cover. Brilliant.org. Retrieved 09:49, December 14, 2019, from https://brilliant.org/wiki/vertex-cover/

[7] Kirkpatrick, S.; Gelatt Jr, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". Science. 220 (4598): 671–680

[8] Karp R.M. (1972) Reducibility among Combinatorial Problems. In: Miller R.E., Thatcher J.W., Bohlinger J.D. (eds) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston, MA

[9] Erdős, P. Combinatorica (1981) 1: 25. https://doi.org/10.1007/BF02579174

[10] R Core Team (2019). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

[11] Csardi G, Nepusz T: The igraph software package for complex network research, InterJournal, Complex Systems 1695. 2006. http://igraph.org

[12] Hadley Wickham, Jim Hester and Winston Chang (2019). devtools: Tools to Make Developing R Packages Easier. R package version 2.2.1. https://CRAN.R-project.org/package=devtools

[13] Simone, Nick. savi, (2019), GitHub repository, https://github.com/nsimone101/savi

[14] Ognyanova, K. (2016) Network analysis with R and igraph: NetSci X Tutorial. Retrieved from www.kateto.net/networks-r-igraph.

[15] Broman, Karl. "Putting your R package on GitHub." *Karl Broman*, https://kbroman.org/pkg_primer/pages/github.html

[16] Wickham, Hadley. "R packages." *R packages*, http://r-pkgs.had.co.nz/

[17] pomber. "Select random element in a list of R?" *Stack Overflow*, 26 December 2014 https://stackoverflow.com/questions/9390965/select-random-element-in-a-list-of-r/9391911

Vita Auctoris

Nicholas Scott Simone was born in Chesterfield, Missouri in 1989. He attended high school at Rockwood Summit High School. He then attended the University of Missouri where he earned a Bachelor of Science degree in Computer Science with Engineering Honors along with a Minor in Engineering and Mathematics.