Prospectus: Graph Coloring and Managing Large Datasets in R

Michael Kane

March 14, 2008

Two unrelated topics are covered in this document. As a result, the decision was made to divide this prospectus into two chapters. The first chapter deals with a threshold phenomenon that occurs in the 3-color problem on Erdös-Rényi graphs. The second chapter deals with the **R** package *bigmemoRy*. This package, created by Jay Emerson and me, can be used as a building-block for managing and analyzing large data sets in **R**.

While the first chapter may resemble a traditional thesis prospectus, the second chapter does not. So, some justification should be given for its inclusion. The *bigmemoRy* package grew from an exploration of the the Netflix Prize data set [1]. Consequently, many of the requirements were determined by us as the need arose. While performing this exploration, we realized that the software we had written would be useful to the **R** community. As a result of this effort, **R** users are now able to do something that was previously impossible: work in parallel on large datasets in RAM and manage large data sets efficiently.

Implementation of this package is almost complete. Future work based on *bigmemoRy* will consist of designing parallel algorithms with the package as a building block, rather than extending the functionality of the package. Some of these algorithms, such as parallel k-means and linear regression, have been implemented. Chapter 2 serves as more of progress report and less as prospectus.

Finally, it should be noted that the second chapter borrows from my Statistical Computing Student Paper Competition entry [2] as well as the *bigmemoRy* package vignette[3].

[1] http://www.netflixprize.com
[2] http://stat.yale.edu/ mjk56/Research/Prospectus/bigmemoRyStudentPaper.pdf
[3] http://stat.yale.edu/ mjk56/Research/Prospectus/bigmemoRy-vignette.pdf

Chapter 1

The 3-Color Problem on an Erdös-Rényi graph

In Achlioptas and Friedgut (1999) it was shown that, for a fixed chromatic number k, a threshold phenomena exists involving the colorability of Erdös-Rényi (ER) graphs, $G(n, c/n)$, when c is allowed to vary. It was shown that, for large n, there is a value c_k such that when $c < c_k$, the graph can be colored with k colors with high probability and when $c > c_k$, the graph is uncolorable with high probability.

In Achlioptas and Molloy (1997) a greedy algorithm for coloring graphs is proposed. By analyzing the algorithm, a lower-bound on c_k is found. The paper examines the behavior of the algorithm on Erdös-Rényi graphs using differential equation to approximate list counts (described below), as well as heuristic arguments, to arrive at the conclusion that $c_k \geq 1.923$.

Some of my prospective goals in researching this threshold phenomenon are:

- Understand the use of differential equations in the graph coloring algorithm on Erdös-Rényi graphs.
- Give a more rigorous justification for the stated lower bound.
- Justify a larger lower bound.

For the rest of this chapter, it will be assumed that we are looking at the 3-color problem ($k = 3$).

1.1 The Algorithm and its Representation

For a graph with n vertices, the Achlioptas and Molloy algorithm attempts to color each vertex such that no vertex has the same color as any of its neighbors. The algorithm begins by associating each vertex with its own list of possible colorings $\{A, B, C\}$. Let L_v be the the color list for vertex v. When a vertex is colored it will be referred to as having a fixed color. When a vertex has a fixed color its list size can be regarded as zero. This way, vertices with 1 possible coloring are distinguished from vertices with a fixed color. For a graph with n vertices, the algorithm is defined as:

```
for n iterations do
    Choose uniformly at random a vertex v from vertices with smallest
      list size greater than zero;
    Pick a color ζ uniformly at random from L_v;
    Fix v with color ζ;
    foreach vertex u not in the set of fixed-color vertices do
        if u is a neighbor of v then
            Remove ζ from u's color list, that is,
              change L_u to L_u \ {ζ};
            if L_u is empty then
                The algorithm fails;
            end
        end
    end
end
```

Algorithm 1: The Greedy Graph Coloring Algorithm

If the algorithm executes for n iterations and does not reach the fail state, the algorithm succeeds.

Let $\mathcal{L}_\alpha$ (not to be confused with L_v) denote the set of α-color combinations. For example, $\mathcal{L}_2 = \{\{A, B\}, \{A, C\}, \{B, C\}\}$. Define $\mathcal{L}_1$ and $\mathcal{L}_3$ similarly. Then, let $S_\beta(t)$, with $\beta \in \mathcal{L}_2$, denote the number of vertices at time t with 2-color list β. For example, $S_{AB}(t)$ denotes the number of vertices with color list $\{A, B\}$ at time t. Denote the size of individual 1-color lists similarly. Employing a slight abuse of notation, let $S_3(t)$, $S_2(t)$, and $S_1(t)$ denote the total number of vertices at time t with, 3-color, 2-color, and 1-color lists respectively. $S_F(t)$ denotes the number of vertices with empty

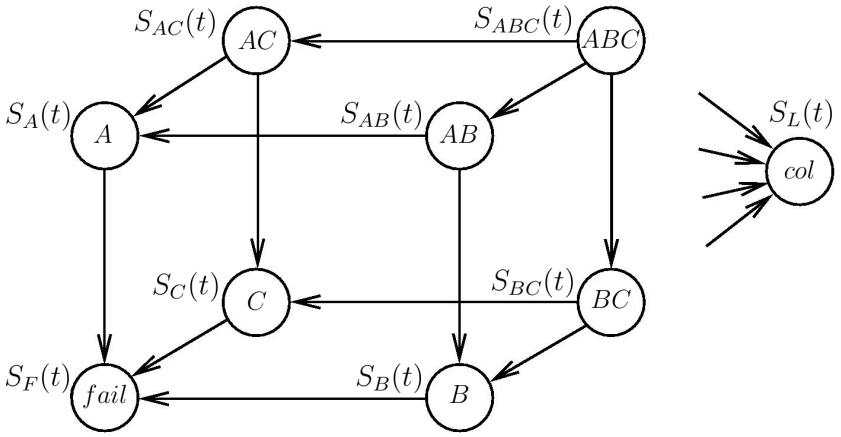

Figure 1.1: A Representation of the Progress of the Algorithm

color list and $S_L(t)$ denotes the number of vertices with fixed color.

For the ER graph on n vertices, each $\binom{n}{2}$ possible edges are included with probability c/n. We can delay the selection of edges connected to a vertex v until the neighbors of v are found in the algorithm. Viewed in this manner, the process is a Markov chain with state space given by the set of vertex counts. The rest of this report assumes an ER graph in the analysis.

A visualization of the progression of list counts is given in Figure (1.1). A vertex may begin an iteration of the algorithm with color-list $\{A, B, C\}$. If an adjacent vertex is selected to be colored with color A, the vertex list will lose A as a possible color and its new color-list is $\{B, C\}$. As a result of this change S_3 is decremented and S_2 is incremented.

1.2 Simulation Results

Before proceeding with some of the preliminary theoretical results, it may be a helpful to provide a more intuitive understanding of how the stack sizes change over time. The differential equation approximation used in the Achlioptas and Molloy paper relies on the fact that as the number of graph vertices increases, the behavior of the stack sizes approaches the approximation. As a result, one would expect that when simulating this

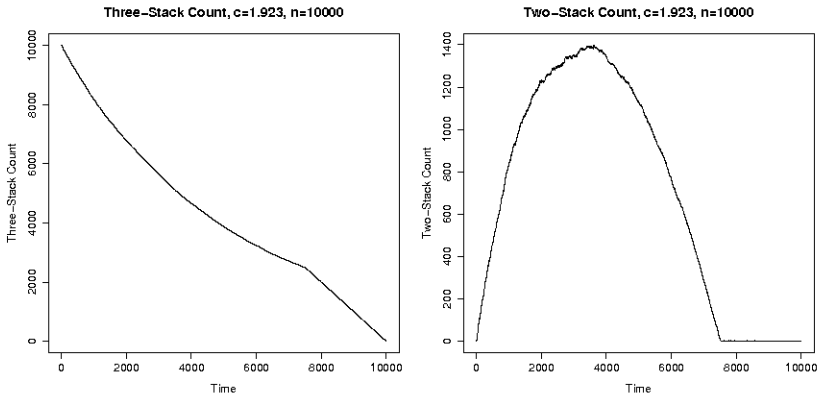

Figure 1.2: The Three-Stack and Two-Stack Counts for a single realization of the process with $c = 1.923$ and $n = 10000$.

process, the stack counts would look similar across different realizations of the process. Also, for any given realization, stack counts should appear relatively smooth.

The algorithm was implemented and the counts for $S_3(t)$ and $S_2(t)$ for a single realization are shown in Figure (1.2). For this simulation, the number of vertices is 10,000 and the value for c, 1.923, is the constant found by Achlioptas and Molloy. When compared to other realizations, this one is fairly representative and, the intuition given in the previous paragraph seems reasonable.

Figure (1.3) shows $S_1(t)$ from time 2100 to time 2400. In this case, the process does not appear to approximate a differential equation. The process is not smooth. Also, this particular realization is not representative and another realization of the process may look significantly different. It seems doubtful that a differential equation approximation of $S_1(t)$ will yield precise results.

1.3 Preliminary Results

This section covers some of the early results that have been reached while attempting to find a lower bound for the threshold of c based on the algorithm proposed by Achlioptas and Molloy.

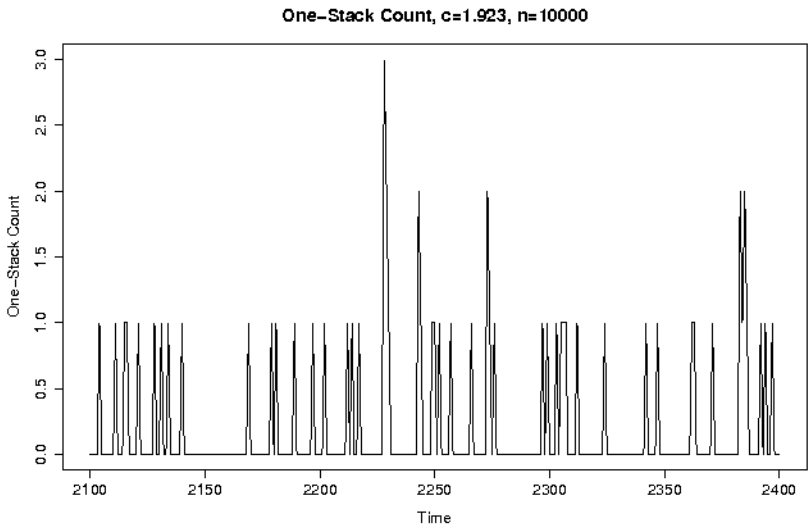

Figure 1.3: The One-Stack Counts for a single realization of the process with $c = 1.923$ and $n = 10000$.

1.3.1 When $c < 1$ the Algorithm Succeeds with High Probability

The algorithm starts by selecting a vertex in the set of vertices with color-list of size 3. This set will be referred to as the 3-stack. Since the fixed-color vertex's neighbors lose a possible coloring, they move to a 2-stack. There is a c/n chance that a vertex is a neighbor of another vertex in the graph. Since there are at most n uncolored vertices, at time t the number of 3-color vertices that become 2-color vertices is stochastically dominated by a random variable with $\text{Bin}(n, c/n)$ distribution.

For the case where $c < 1$, the expected number of vertices which go from the 3-stack to a 2-stack is less than 1 at each time step. Since the algorithm moves 1 vertex to the fixed color stack at any time, when a vertex is moved to the 2-color stack it is quickly moved to the fixed color stack in the next time step. This implies that, with high probability, when $c < 1$ the algorithm will succeed.

1.3.2 1-Stacks Stay Small Until Order $n^{1/3}$

For a vertex to be a member of a 1-stack at time t it must have been chosen twice to have a color from its color-list removed. Each of the other $t-2$ times it was not selected to be colored and was not adjacent to a vertex being colored. This means that the probability of a given vertex being in a 1-stack at time t is at least

$$p_1(t) = \binom{t}{2}\left(\frac{c}{n}\right)^2. \tag{1.3.1}$$

This implies that the expected number of 1-stack vertices up to time m_1 is less than

$$\sum_{t=1}^{m_1} np_1(t) \leq \sum_{t=1}^{m_1} \frac{t^2}{2}\frac{c^2}{n} = \frac{c^2}{6n}m_1^3. \tag{1.3.2}$$

For a given $\varepsilon_1 > 0$, if an m_1 is chosen as a suitably small multiple of $n^{1/3}$, we can ensure that $\mathbb{P}\{S_1(m_1) > 0\} < \varepsilon_1$.

1.3.3 The Probability of Failure is Small Until Order $n^{2/3}$

For a vertex to reach the fail state at time t, it must have been chosen three times to have a color from its color-list removed. This means that the probability of a vertex reaching the fail state at time t is at least

$$p_f(t) = \binom{t}{3}\left(\frac{c}{n}\right)^3.$$

Then, the expected number of vertices in the fail state at time t can be bound by

$$\mathbb{P}S_f(t) \leq n\binom{t}{3}\left(\frac{c}{n}\right)^3 \leq \frac{t^3}{6}\frac{c^3}{n^2}. \tag{1.3.3}$$

For a given $\varepsilon_f > 0$, if an m_f is chosen as a suitably small multiple of $n^{2/3}$, we can ensure that $\mathbb{P}\{S_f(m_f) > 0\} < \varepsilon_f$.

1.3.4 An Upper Bound on the Expected Maximum of $S_2(t)$

At each step in the algorithm one of the vertices is removed for coloring. This means that, if we start with n vertices

$$n - t = S_3(t) + S_2(t) + S_1(t) + S_f(t). \tag{1.3.4}$$

Therefore, $n - t \geq S_3(t)$. Using this fact it follows that the number of vertices going from the 3-stack to a 2-stack is stochastically dominated by $b_t^{(2)} \sim \text{Bin}(t(t+1)/2, c/n)$. Therefore, it follows that $S_3(t)$ is stochastically dominated by $\sum_t b_t^{(2)}$. To get an upper bound on the biggest $S_2(t)$ it is sufficient to get a bound on the size of the sum of $b_t^{(2)}$. This can be done by centering each of the b_t values and then applying the Bennett Inequality.

Following the justification for the Bennett inequality given in (Pollard 2001, Chapter 11), if $X \sim \text{Bin}(n, p)$, then

$$\mathbb{P}\{X \geq \varepsilon\} \leq \exp\left(-\frac{(\varepsilon - np)^2}{2np(1-p)}\psi\left(\frac{n(\varepsilon - np)}{np(1-p)}\right)\right) \tag{1.3.5}$$

where

$$\psi(x) := \begin{cases} 2\left((1+x)\log(1+x) - x\right)/x^2 & \text{for } x \geq -1 \text{ and } x \neq 0 \\ 1 & \text{for } x = 0. \end{cases}$$

If $x \leq 17$ then $\psi(x) > 1/4$. For the binomial case this implies that when $\varepsilon + np \leq 17$, Equation (1.3.5) becomes

$$\mathbb{P}\{X \geq \varepsilon + np\} \leq \exp\left(-\frac{(\varepsilon - np)^2}{4np(1-p)}\right)$$

between $\varepsilon = np$ and $\varepsilon - np \leq 17$ there is a sharp decrease in the probability that X will be larger than ε. In other words, the probability that X will is larger than its expected becomes small quickly in ε.

Getting back to the case of $S_2(t)$, it can be seen that the sum of the binomial increments b_t is distributed as $\text{Bin}(nt - t(t+1)/2, c/n)$. This implies that at time t_1

$$\mathbb{P}S_2(t_1) \leq ct\left(1 - \frac{(t+1)}{2n}\right). \tag{1.3.6}$$

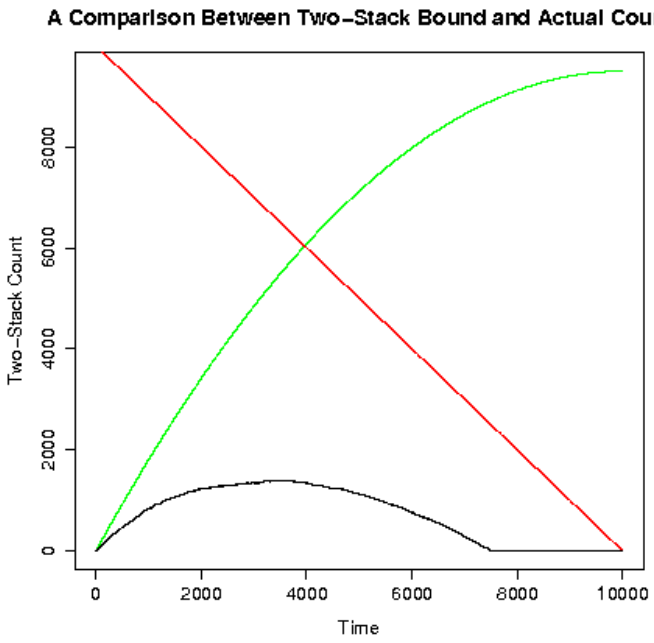

Figure 1.4: The bound for the expected value of $S_2(t)$ (green), $n - t$ (red), and the actual $S_2(t)$ counts (black).

This bound is crude. The first problem is that it does not take into account the fact that when $S_1(t)$ is zero and $S_2(t) > 0$ one of the members of the two-stack will be colored and subsequently the size will decrease by 1. As a result, the bound on the expected size of $S_2(t)$ is strictly increasing. However, by taking the minimum of Equations 1.3.6 and 1.3.4 a better bound is reached. Figure 1.4 shows this bound vs. a simulated $S_2(t)$.

From the figure it looks like the derived bound is a poor upper-bound for the expected size of $S_2(t)$. However, before dismissing this bound completely, it should be noted that the maximum size of $S_2(t)$ increases very quickly in c while the given bound on the expected value increases linearly. This bound may prove more useful when looking at larger values of c.

1.3.5 Clearing out 1-Stacks

In the last two sections it was shown that at early stages of the algorithm, all vertices have color list sizes of either 2 or 3. Equation (1.3.2) indicates that at some time of order $n^{1/3}$ some vertices begin to have color list sizes of 1. Since a vertex gets to a 1-stack from a 2-stack and the size of a 2-stack is relatively small at this time, it seems reasonable that for the first few times the 1-stack becomes non-empty, the number of vertices going from a 2-stack to a 1-stack is relatively small. By a time of order $n^{2/3}$ there is a danger that a vertex's color list will become empty and the algorithm will fail. This section examines the behavior of the 1-stacks at times of order $n^{2/3}$.

Let $t_0 = \varepsilon_c n^{2/3}$ be a random time. Let

$$\tau_1 = \min\{t \geq t_0 : S_1(t) = 0\}.$$

This is the first time a 1-stack becomes empty after a random time of order $n^{2/3}$. Let $s > 1$ be a number of steps after t_0. Let N be the number of vertices that drop to a 1-stack between time t_0 and $t_1 = t_0 + s$. Then

$$\{\tau_1 > t_0 + s\} \leq \{N \geq 1\} + \{S_1(t_0) \geq s\}.$$

The number of time steps to clear out the 1-stack is less than the size of the 1-stack at time t_0 plus the number of vertices that go to the 1-stack between time t_0 and t_1.

By Equation (1.3.1), the probability a vertex ends up in a 1-stack at time t_0 is less than $p_1(t_0)$. Since at time t_0 there are less than n vertices which have not been colored, the distribution of $S_1(t_0)$ is stochastically dominated by a $\text{Bin}(n, (ct_0)^2/2n)$ distribution. Using the Bennett bound results from the previous section, $\mathbb{P}\{S_1(t_0) \geq s\}$ will be close to 1 in s until

$$s = (ct_0)^2/2 \tag{1.3.7}$$

at which point the probability will decrease quickly.

From time t_0 to t_1, vertices can get to the 1-stack from either the 3-stack or the 2-stack. Let $\Delta_{t_0} B_{v,t_1}$ be the number of times vertex v is chosen to lose a color from its color from t_0 to t_1. Let V, $V_3(t)$ and $V_2(t)$ be the set of all vertices and the set of vertices with color list sizes 3 and 2 respectively.

Then,

$$\begin{aligned} N &= \sum_{t=t_0}^{t_1} \left(\sum_{v \in V} \{v \in V_3(t)\}\{\Delta_{t_0} B_{v,t_1} = 2\} + \{v \in V_2(t)\}\{\Delta_{t_0} B_{v,t_1} = 1\} \right) \\ &\leq \sum_{t=t_0}^{t_1} \left(\frac{c}{n}\right)^2 S_3(t) + \frac{c}{n} S_2(t) \\ &\leq \frac{sc}{n} \left(c + S_2(t_1)\right). \end{aligned}$$

This means that

$$\mathbb{P}\{N \geq 1\} \leq \mathbb{P}\left\{ S_2(t_2) \geq \frac{n}{sc} - c \right\}. \tag{1.3.8}$$

Using a $\text{Bin}(n, c/n)$ distribution to stochastically bound $S_3(t)$ and the Bennett bound results from the last section, when the expected value of binomial becomes greater than its mean, this probability goes to zero quickly. This means Equation (1.3.8) decreases sharply at

$$\frac{ct_0^2}{2} = \frac{n}{sc} - c$$

or

$$s = \frac{n}{c^2} \left(\frac{t_0^2}{2} + 1 \right). \tag{1.3.9}$$

It may be noted that in general the bound found in Equation (1.3.7) is smaller than Equation (1.3.9). This means that there is a sharp decrease in the probability of hitting τ_1 after $t_0 + s$ when $s = (ct_0)/2$.

1.3.6 Color-Balance for a Simplified Process

It has already been shown that the algorithm will have empty 1-stacks until a time of order $n^{1/3}$. At this time, a 1-stack will become non-empty but it will quickly return to being empty. During this time, the 1-stack is emptied because the number of vertices going from a 2-stack to a 1-stack is small. As the algorithm continues, the number of vertices in the 2-stacks increases. As a result, the number of vertices going from 2-stacks to 1-stacks increases. At a time of order $n^{2/3}$ it is possible for the algorithm to fail. Failure will occur because, as the number of vertices in the 1-stack increases, it becomes more likely that a 1-stack vertex will move to the fail stack.

Until now, the analysis has been "color-blind". The proportions of colors within the 2-stacks have not been considered. When 2-stacks are being colored, the most prevalent color in the 2-stack color lists is the most likely to be picked. Therefore, it seems reasonable that there is a feedback mechanism which balances the color proportions. This section gives some justification to this idea by considering a simplified model.

Consider a model with a 3-stack and 2-stacks only and colored vertices remain unchanged. At the beginning of the algorithm all vertices are in the 3-stack. Then, one of the 3-stack vertices is selected and a color from its color list is chosen. Adjacent vertices lose the selected color and are moved to the appropriate 2-stack. At the second iteration one of the 2-stack vertices is selected and a color is chosen at random from its color list. As a further simplification, only 3-stack vertices may lose a color from its color list. Then, 3-stack vertices which are adjacent to the selected vertex lose the selected color. The algorithm continues until the 3-stack is empty.

In this simplified model, let $X_{AB}(t)$ denote the number of vertices with color list $\{A, B\}$ at time t. $X_{AC}(t)$ and $X_{BC}(t)$ are defined similarly. Let $X(t)$ be the vector $(X_{AB}(t), X_{AC}(t), X_{BC}(t))$. Again, using a slight abuse of notation, $X_2(t)$ will denote the number of 2-stack vertices at time t.

After the first step, the probability a vertex with color list $\{A, B\}$ is colored is its proportion to the total 2-stack size. Define

$$\delta_{AB}(t) = \{v \in V_{AB}(t) \text{ colored at } t+1 | X(t)\}.$$

Then,

$$\mathbb{P}\delta_{AB}(t) = \frac{X_{AB}(t)}{X(t)}.$$

Let b_{t+1} be the number of vertices that go from 3-color lists to 2-color lists at time $t+1$. The probability that a vertex $v \in V_{AB}(t)$ is selected at time $t+1$ is the probability that a vertex with color list $\{A, B\}$ or $\{B, C\}$ is picked to be colored times the probability that the color picked is B. Then, letting $\mathbb{P}_t$ denote the probability conditioned on information through time t,

$$\begin{aligned}\mathbb{P}_t\{X_{AC}(t+1) = X_{AC}(t) + b_{t+1}\} &= \frac{1}{2}\left(\frac{X_{AB}(t) + X_{BC}(t)}{X_2(t)}\right) \\ &= \frac{1}{2}\left(1 - \frac{X_{AB}(t)}{X_2(t)}\right).\end{aligned}$$

At each step of the algorithm, there is a c/n chance that a vertex in a 3 stack will move to a 2-stack. Therefore,

$$b_{t+1} \sim \text{Bin}(S_3(t), c/n).$$

In order to show that $X_{AB}(t) \approx X_{AC}(t) \approx_{BC} (t)$, consider the differences between 2-color stack sizes. The behavior of

$$Z_t = X_{AB}(t) - X_{BC}(t)$$

is typical. Let

$$\Delta_{t+1} Z = Z_{t+1} - Z_t.$$

Then,

$$\begin{aligned}
\mathbb{P}_t Z_t \Delta_{t+1} Z &= Z_t \mathbb{P}_t b_{t+1} \mathbb{P}_t(\delta_{AB}(t+1) - \delta_{BC}(t+1)) \\
&= -\frac{c}{2n} \frac{X_3(t)}{X_2(t)} Z_t^2 \\
&\leq 0.
\end{aligned}$$

This implies that if, at some time, the difference $|X_{AB}(t) - X_{BC}(t)|$ gets large, the stack selection process ensures that subsequent iterations of the algorithm will reduce this difference. Also, this negative feedback mechanism exists between all two-color pairs of elements in X_t. Furthermore,

$$\begin{aligned}
\mathbb{P}_t Z_{t+1}^2 &\leq Z_t^2 + \mathbb{P}_t b_{t+1}^2 \mathbb{P}_t(\delta_{AB}(t+1) - \delta_{BC}(t+1)^2 \\
&\leq Z_t^2 + \mathbb{P}_t b_{t+1} \\
&\leq Z_t^2 + c + c^2.
\end{aligned}$$

From this we can conclude

$$\mathbb{P} Z_{t+1}^2 \leq \mathbb{P} Z_0 + t(c + c^2)$$

and the second moment increases, at most, linearly in time.

If we change the simplified process so that colored vertices are sent to a colored-stack the feedback phenomena still occurs. In fact, feedback becomes more pronounced. However, bounding the second moment of the increments becomes difficult. The problem is that the conditional second moment of Z_{t+1} is a function of $Z_t/X_2(t)$ and, since a 2-stack size of zero may occur with positive probability, we cannot use the same method used previously to bound $1/X_2(t)$.

1.4 Issues and Conjectures

In analyzing the greedy graph-coloring algorithm, one of the challenges is understanding of how coloring affects stack sizes. I expect that for the first couple steps of the algorithm, all vertices are members of the 3-stack. During this time a few vertices are colored. Next, vertices begin to move to 2-stacks. During this time most of the vertices which are colored come from the 2-stacks. Then, a small number of vertices move to 1-stacks.

At this time, if the rate at which vertices are colored in the 1-stack becomes greater than the rate of vertices from the 2-stacks move to the 1-stacks then the 1-stacks will continue to be cleared out. Eventually, the two-stacks will become smaller and no more vertices will move to 1-stacks. In this case, the algorithm will succeed.

On the other hand, if the rate at which vertices are colored in the 1-stack is less than the rate of vertices moving from 2-stacks to 1-stacks, the 1-stack size will increase. Eventually, 2 of the 1-stack vertices will be adjacent in the graph and have the same color. In this case the algorithm will fail.

A better understanding of the effect of coloring on stack sizes will translate to a better understanding of the size of $S_3(t)$ and $S_2(t)$. From this, I should be able to get better bound on the failure probability.

Based a heuristic argument for the effect of coloring on stack sizes, I would conjecture that the lower bound on c (the ER graph parameter), is at least 2.885. Simulation results indicate that the threshold may be as high as 3.5.

Chapter 2

bigmemoRy

The package *bigmemoRy* originally grew from the need to deal with large data sets in RAM interactively in **R**. While working on the Netflix Prize competition, it was quickly realized that although **R** excels at examining and analyzing small data sets, as soon as the data set size is of the order of the amount of RAM on a machine, computations become unwieldy. The reason for this inefficiency, which is discussed below, is that in some cases **R** makes unnecessary copies of a data set and **R** does not always represent data as efficiently as possible. *bigmemoRy* provides a more efficient representation, both in terms of disk usage and manipulation, thereby allowing users to deal with larger data sets in RAM than what was previously possible.

In addition, *bigmemoRy* has been implemented to use shared memory on *nix machines which allows users to share data across **R** sessions. This means that multiple users can access the same data set at the same time. Even more exciting is the fact that this also allows a user to implement algorithms which work in parallel on the same data set. Until *bigmemoRy* this was not possible in **R**.

2.1 The Problem with Large Data Sets in R

Many **R** users are not aware that 4-byte integers are implemented in **R**. So if memory is at a premium and only integers are required, a user would avoid the 229 MB memory consumption of

```
> x <- matrix(0, 1e+07, 3)
> round(object.size(x)/(1024^2))

[1] 229
```

in favor of the 114 MB memory consumption of an integer matrix:

```
> x <- matrix(as.integer(0), 1e+07, 3)
> round(object.size(x)/(1024^2))

[1] 114
```

Similar attention is needed in subsequent arithmetic, because

```
> x <- x + 1
```

coerces `x` into a matrix of 8-byte real numbers. The memory used is then back up to 229 MB, and the peak memory usage of this operation is approximately 344 MB (114 MB from the original `x` and a new 229 MB vector). In contrast,

```
> x <- matrix(as.integer(0), 1e+07, 3)
> x <- x + as.integer(1)
```

has the desired consequence, adding 1 to every element of `x` while maintaining the integer type. The memory usage remains at 114 MB, and the operation requires temporary memory overhead of an additional 114 MB. It's tempting to be critical of *any* memory overhead for such a simple operation, but this is often necessary in a high-level, interactive programming environment. In fact, **R** is being efficient in the previous example, with peak memory consumption of 2.24 MB; the peak memory consumption of

```
> x <- matrix(as.integer(0), 1e+07, 3)
> x <- x + matrix(as.integer(1), 1e+07, 3)
```

is 344 MB. Some overhead is necessary in order to provide flexibility in the programming environment, freeing the user from the explicit structures of programming languages like **C**. However, this overhead becomes cumbersome with massive data sets.

Similar challenges arise in function calls, where **R** uses the *call-by-value* convention instead of *call-by-reference*. Fortunately, **R** creates physical copies of objects only when apparently necessary (called *copy-on-demand*). So there is no extra memory overhead for

```
> x <- matrix(as.integer(0), 1e+07, 3)
> myfunc1 <- function(z) return(c(z[1, 1], nrow(z), ncol(z)))
> myfunc1(x)

[1]        0 10000000        3
```

which uses only 114 MB for the matrix `x`, while

```
> x <- matrix(as.integer(0), 1e+07, 3)
> myfunc2 <- function(z) {
+     z[1, 1] <- as.integer(5)
+     return(c(z[1, 1], nrow(z), ncol(z)))
+ }
> myfunc2(x)

[1]        5 10000000        3
```

temporarily creates a second 114 MB matrix, for a total peak memory usage of 229 MB.

These examples demonstrate the simultaneous strengths and weaknesses of **R**. As a programming environment, it frees the user from the rigidity of a programming language like **C**. The resulting power and flexibility has a cost: data structures can be inefficient, and even the simplest manipulations can create unexpected memory overhead. For most day-to-day use, the strengths far outweigh the weaknesses. But when working with massive data sets, even the best efforts can grind to a halt.

2.2 Design and Consequences

`BigMatrix` is defined using an `S4` class containing an external pointer (**R** type `externalptr`) and a type (a character string describing the type of the atomic element). The pointer gives the memory location of a **C** structure such as `BigIntMatrix` (the only structure yet implemented). The

`BigIntMatrix` structure contains the number of rows and columns of the matrix, a vector of strings corresponding to matrix column names, and a pointer to an array of pointers to integers (of length equal to the number of columns of the matrix). The structure, including the vectors of data (one for each column of the matrix), is dynamically allocated using `malloc()`. This is not the typical representation of a matrix as a vector but it has some natural advantages for data exploration and analysis.

A collection of functions are provided for the class `BigMatrix`. They are designed to mimic traditional matrix functionality. For example, the **R** function `ncol()` automatically associates a `BigMatrix` with a new method function, returning the number of columns of the matrix via the **C** function `CGetIntNcol()`. Other examples are more subtle. For example, if `x` is a `BigMatrix` and `a` and `b` are traditional **R** vectors of row and column indices, then `x[a,b]` can be used to retrieve the appropriate submatrix of `x` (and is a traditional **R** matrix, not a `BigMatrix`). The same notation can be used to assign values to the submatrix, if the dimensionality of the assignment conforms to **R**'s set of permissible matrix assignments.

An object of class `BigMatrix` contains a *reference* to a data structure. This has some noteworthy consequences. First, if a `BigMatrix` is passed to a function which changes its values, those changes will be reflected outside the function scope. Thus, *side-effects* have been introduced for `BigMatrix` objects. When an object of class `BigMatrix` is passed to a function, it should be thought of as being *called-by-reference*.

The second consequence is that copies are not made by simple assignment. For example:

```
> library(bigmemoRy)
> x <- BigMatrix(nrow = 2, ncol = 2, init = 0)
> x[, ]

     [,1] [,2]
[1,]    0    0
[2,]    0    0

> y <- x
> y[1, 1] <- 1
> x[, ]
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    0
```

The assignment `y <- x` does not copy the contents of the `BigMatrix` object; it only copies the type information and memory location of the data. As a result, the variables `x` and `y` refer to the same data in memory. If an actual copy is required, the programmer must first create a new object of class `BigMatrix` and then copy the contents.

The core functions supporting `BigMatrix` objects are:

`BigMatrix()`	`is.BigMatrix()`	`as.BigMatrix()`	`nrow()`	`deepcopy.bm()`
`ncol()`	`dim()`	`head()`	`tail()`	`colnames()`
`print()`	`which.bm()`	`read.bm()`	`write.bm()`	`rownames()`
`rm.cols.bm()`	`add.cols.bm()`	`hash.mat.bm()`	`dimnames()`	`"["` and `"[<-]"`

Other basic functions are included, serving as templates for the development of new functions. These include:

`colmin()`	`min()`	`max()`	`colmax()`
`colrange()`	`range()`	`colmean()`	`mean()`
`colvar()`	`colsd()`	`summary()`	`biglm.bm()`
`bigglm.bm()`	`kmeans.bm()`		

2.3 Development Based on `bigmemoRy`

Since the completion of the basic functionality of ***bigmemoRy***, the emphasis has switched to designing and implementing algorithms that take advantage of the packages specific strengths.

A wrapper has been written for Thomas Lumley's ***biglm*** package. This package allows a user to create a linear or generalized linear model based on chunks, or subsets of the rows of a data set. An initial model is created based on an initial chunk or subset of the rows of a data set. The model is then updated with other chunks from the data set. This iterative updating allows the user to perform regressions on data sets that are too large for the `lm` and `glm` functions. The ***bigmemoRy*** wrapper allows a user to perform a regression on a `BigMatrix` object using a calling convention similar to that

of `lm`. The wrapper function calculates the appropriate chunk size and calls the appropriate regression functions.

Jay Emerson has implemented a parallel k-means cluster analysis based on Lloyd's scheme. Like the linear regression wrapper, it uses *bigmemoRy* for its space efficiency. However, parallel k-means also takes advantage of shared memory. Parallel k-means spawns multiple **R** sessions which perform analyses on the data concurrently. Gains are seen not only in the amount of memory needed to perform the analysis, but also in how long the analysis takes to run.

2.4 Conclusion

bigmemoRy supports double, integer, short, and char data types; in Unix environments, the package optionally implements the data structures in shared memory. Previously, parallel use of **R** required redundant copies of data for each **R** session. The shared memory version of a `BigMatrix` object now allows separate R sessions on the same computer to share access to a single copy of the data set. *bigmemoRy* extends and augments the **R** statistical programming environment, opening the door for more powerful parallel analyses and data mining of large data sets.

Bibliography

Achlioptas, D. and E. Friedgut (1999). A sharp threshold for k-colorability. *Random Structures and Algorithms 14*, 63–70.

Achlioptas, D. and M. Molloy (1997). The analysis of a list-coloring algorithm on a random graph (extended abstract). *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, 204–212.

Pollard, D. (2001). *A User's Guide to Measure Theoretic Probability*. Cambridge University Press.