Maximum Likelihood Estimation of the Random-Clumped Multinomial Model as Prototype Problem for Large-Scale Statistical Computing

Andrew M. Raim[a*], Matthias K. Gobbert[a], Nagaraj K. Neerchal[a] & Jorge G. Morel[b]

[a]*Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, MD, U.S.A.*

[b]*Biometrics and Statistical Sciences Department, Procter & Gamble Company, Cincinnati, OH, U.S.A.*

This is a preprint of an article submitted for consideration in the Journal of Statistical Computation and Simulation ©2012 Taylor & Francis; Journal of Statistical Computation and Simulation is available online at: www.tandfonline.com.

Abstract

Numerical methods are needed to obtain maximum likelihood estimates (MLEs) in many problems. Computation time can be an issue for some likelihoods even with modern computing power. We consider one such problem where the assumed model is the Random-Clumped Multinomial distribution. We compute MLEs for this model in parallel using the Toolkit for Advanced Optimization (TAO) software library. The computations are performed on a distributed-memory cluster with low latency interconnect. We demonstrate that for larger problems, scaling the number of processes improves wall clock time significantly. An illustrative example shows how parallel MLE computation can be useful in a large data analysis. Our experience with a direct numerical approach indicates that more substantial gains may be obtained by making use of the specific structure of the Random-Clumped model.

Key Words: parallel computing, maximum likelihood estimation, mixture distribution, multinomial

AMS Subject Classification: 65C60, 65Y05

1 Introduction

Consider a random sample of n observations $\boldsymbol{X} = (X_1, \dots, X_n)$ drawn from a probability distribution $f(x \mid \boldsymbol{\theta})$. The vector of parameters $\boldsymbol{\theta} = (\theta_1, \dots, \theta_q)$ will be considered unknown, and belongs to the space $\Theta \subseteq \mathbb{R}^q$. For a given dataset $\boldsymbol{x} = (x_1, \dots, x_n)$, the likelihood is given by

$$L(\boldsymbol{\theta} \mid \boldsymbol{x}) = \prod_{i=1}^{n} f(x_i \mid \boldsymbol{\theta}),$$

and the maximum likelihood estimate (MLE) is obtained by

$$\widehat{\boldsymbol{\theta}}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} L(\boldsymbol{\theta} \mid \boldsymbol{x}),$$

*Corresponding author. Email: araim1@umbc.edu or equivalently by maximizing the log-likelihood $\log L(\boldsymbol{\theta} \mid \boldsymbol{x})$. In some circumstances maxima can be calculated analytically, but numerical methods are often needed to carry out the optimization. Commonly used numerical methods include Newton-Raphson, Fisher Scoring, and expectation maximization (EM); see [1] for a general overview.

We investigate the computation of MLEs for the Random-Clumped Multinomial distribution, introduced by Morel and Nagaraj in [4], using a parallel architecture. In particular, we make use of a publicly available software library TAO (Toolkit for Advanced Optimization) [2], which implements a number of commonly used numerical optimization methods. TAO is a special optimization library designed for use in parallel computing environments. The objective of this work is to study the effectiveness, in terms of computing time, of applying parallel optimization to the MLE problem. We make use of the limited-memory, variable-metric (LMVM) unconstrained optimization method in TAO. Malouf [3] has demonstrated the effectiveness of LMVM in the setting of natural language processing. Using TAO to conduct maximum entropy estimation, he shows that LMVM outperforms several other methods such as conjugate gradient.

In a 2003 report describing the now-popular R package SNOW (Simple Network of Workstations), Rossini, Tierney, and Li [5] noted that parallel computing had not yet been widely adopted by statisticians. Since then, packages like SNOW and its successors have helped make parallel computing more accessible to R programmers. Many of these packages are geared toward "embarrassingly parallel" problems — those which can be easily decomposed into smaller problems that have little dependence on each other. For example SNOW provides a function called `parApply`, which evaluates a given function on each row (or column, or element) of a given matrix. Each row can be operated on independently, and the package determines how to allocate the work among available parallel processes. This is adequate for many problems in statistical computing which involve repeating the same calculation many times using randomly generated inputs. Resampling methods such as the bootstrap and Monte Carlo simulation fall into this class of applications. The methods presented in this paper look deeper into the structure of the computations and seek to improve performance within the algorithm itself. Therefore, they are best evaluated on a single complex problem rather than on problems involving repeated computations.

Our objective is to apply parallel computing to the MLE optimization problem which does not fit the mold of embarrassingly parallel. The required numerical optimization is an iterative process where each step must occur sequentially. We would like to distribute the work across many parallel processes so that the computing time can be reduced. To effectively use many processes we split the workload at each iteration, then distribute the results back to all processes to prepare for the next iteration. Therefore, efficient communication is important for good performance, and high performance computing (HPC) is especially suitable for this application. An HPC cluster provides an array of fast processors connected by a low-latency high-throughput interconnect, and optimized communication software such as the Message Passing Interface (MPI). This environment will ensure that processes can communicate efficiently, and that we can benefit from scaling the procedure to run on a large number of processes.

In Section 2 the Random-Clumped model is described, including an algorithm for drawing samples from the distribution. Section 3 discusses the approach for computing MLEs in parallel. Section 4 presents simulation studies which show how run times and solution quality are affected as problem dimensions are varied, and which verify the consistency property of the resulting maximum likelihood estimates. We find that parallel computing is effective for large problems where "large" will be quantified. In Section 5 we present a large simulated scenario from the Random Clumped Multinomial model where inference is computationally expensive. The parallel MLE approach is applied to significantly reduce the computation time of a likelihood ratio test. Finally, concluding remarks are given in Section 6.

2 The RCM model

First, let us consider the standard multinomial distribution, which arises in a natural way when a group of m people are asked a survey question with k possible responses. The sample space is the discrete set

$$\mathcal{T} = \left\{ \boldsymbol{t} = (t_1, \dots, t_k) : \ t_j \in \{0, 1, \dots, m\}, \ \sum_{j=1}^{k} t_j = m \right\},$$

where t_i denotes the number of people who gave the ith response. Let $\boldsymbol{T} = (T_1, \dots, T_k)$ denote a random vector of counts from $\mathcal{T}$. If we assume that the participants respond to the question independently of each other, with probabilities $\boldsymbol{\pi} = (\pi_1, \dots, \pi_k)$ corresponding to the possible responses, $\boldsymbol{T}$ will be distributed according to the multinomial distribution, whose density function is

$$f(\boldsymbol{t} \mid \boldsymbol{\pi}, m) = \frac{m!}{t_1!\, t_2! \cdots t_k!} \pi_1^{t_1} \pi_2^{t_2} \cdots \pi_k^{t_k}, \quad \boldsymbol{t} \in \mathcal{T}. \tag{1}$$

The parameter space is then

$$\Theta = \left\{ \boldsymbol{\pi} \in \mathbb{R}^k : \ 0 < \pi_j < 1, \ \sum_{j=1}^{k} \pi_j = 1, \right\},$$

with only $k-1$ distinct parameters since $\pi_k = 1 - \sum_{j=1}^{k-1} \pi_j$. If a random vector $\boldsymbol{T}$ follows this distribution, we write $\boldsymbol{T} \sim \text{Mult}(\boldsymbol{\pi}, m)$, and denote observed data as $\boldsymbol{t} = (t_1, \dots, t_k)$. If we repeat this survey n times, each time with a group of m people, we will obtain a sample $\boldsymbol{X} = (\boldsymbol{T}_1, \dots, \boldsymbol{T}_n)$, which can be thought of as a $k \times n$ matrix.

A mixture of ν multinomials constructed with mixing proportions $\boldsymbol{w} = (w_1, \dots, w_\nu)$ is given by

$$f(\boldsymbol{t} \mid \boldsymbol{w}, \boldsymbol{\pi}, m) = \sum_{j=1}^{\nu} w_j \, f(\boldsymbol{t} \mid \boldsymbol{\pi}_j, m), \tag{2}$$

where $\sum_{j=1}^{\nu} w_j = 1$, $0 < w_j < 1$ for $j = 1, \dots, \nu$, and $\boldsymbol{\pi}_j = (\pi_{j1}, \dots, \pi_{jk})$ is the vector of probabilities corresponding to the jth component of the mixture. One motivation for considering a mixture distribution may be drawn from the point of view of classification. Suppose the participants in our multinomial response survey are drawn from one of ν different populations, and we are unable to record the population label for each subject. Of course, if the population label were available, we will end up with ν independently distributed multinomial count vectors. Since the labels are not available, the likelihood will be based on the mixture density given in (2). This distribution has been widely used in a number of applications including text mining, linguistics, and clustering; see [6] for a detailed review. These mixture likelihoods generally cannot be maximized in closed form, as opposed to the standard multinomial likelihood, hence numerical methods are needed for the MLE problem. Mixtures in general may not be identifiable without additional assumptions.

We consider a special multinomial mixture proposed by Morel and Nagaraj [4] as the test problem for our exploration. Following [7], we will refer to this model as the Random-Clumped Multinomial (RCM) model. The model is also described in detail in [8]. It has more recently been referred to as the Neerchal-Morel distribution by Zhou and Lange [9], who use it to help demonstrate the minorization-maximization principle. The motivation for the RCM model can be seen in the survey scenario mentioned earlier. If the m participants interact among themselves before providing their responses, then the key "independence" assumption is violated, and the multinomial distribution does not adequately model the responses. In fact, it can be shown that such data, due to lack of independence, exhibits larger variability than the multinomial distribution. This phenomenon is commonly referred to as overdispersion. Morel and Nagaraj [4] provide a model for a specific type of dependence, which turns out to be a special case of the multinomial mixture distribution in (2). In subsequent work [8, 10], they show that this model has many desirable theoretical and practical properties.

The RCM model can be obtained by correlating responses within a group by a simple logic. Imagine that the group of m respondents consists of a leader who would make his/her response public. Then, the remaining members may either follow the leader or make up their own mind independently of each other and of the leader. Thus, the distribution of the count vector $\boldsymbol{T}$ would conform to the representation $\boldsymbol{T} = \boldsymbol{Y}N + (\boldsymbol{X} \mid N)$, where

$$N \sim \text{Binomial}(\rho, m), \qquad \boldsymbol{Y} \sim \text{Mult}(\boldsymbol{\pi}, 1), \qquad (\boldsymbol{X} \mid N) \sim \text{Mult}(\boldsymbol{\pi}, m - N),$$

such that N and $\boldsymbol{Y}$ are independent, $0 < \rho < 1$, and $\boldsymbol{\pi} = (\pi_1, \dots, \pi_k)$ is a vector of category probabilities as described for the standard multinomial. It can be shown that the density for $\boldsymbol{T}$ is

$$f(\boldsymbol{t} \mid \boldsymbol{\pi}, \rho, m) = \sum_{j=1}^{k} \pi_j \, g(\boldsymbol{t} \,|\, \boldsymbol{\eta}_j, m),$$

where $g(\boldsymbol{t} \,|\, \boldsymbol{\eta}_j, m)$ is the density of a standard multinomial,

$$\boldsymbol{\eta}_j = \begin{cases} (1-\rho)\boldsymbol{\pi} + \rho\, \boldsymbol{e}_j & \text{if } j = 1, 2, \dots, k-1, \\ (1-\rho)\boldsymbol{\pi} & \text{if } j = k, \end{cases}$$

and $\boldsymbol{e}_j$ is the jth column of the identity matrix $\boldsymbol{I}_k$. We will use the notation $\boldsymbol{T} \sim \text{RCM}(\boldsymbol{\pi}, \rho, m)$ to describe the distribution of $\boldsymbol{T}$. We have noted that RCM is a special case of the mixture distribution of (2). In this mixture however, there are only k distinct parameters $\boldsymbol{\theta} = (\pi_1, \dots, \pi_{k-1}, \rho)$. Our objective will be to compute the MLE for $\boldsymbol{\theta}$ under this model. Although we will not make use of them in this paper, theoretical results are available in [10] and [6] which help to simplify MLE computations using a Fisher Scoring approach.

Neerchal and Morel [10] describe a method for generating random samples from the RCM distribution. We include this information as a convenience to the reader. We first consider generation of samples from the standard multinomial distribution. Begin with a vector

$\boldsymbol{t} = (t_1, \dots, t_k)$ of k zeroes, and known parameters $(\pi_1, \dots, \pi_k)$. Generate m observations from the uniform distribution $U_1, \dots, U_m \overset{iid}{\sim} \mathrm{U}(0,1)$. For observation U_j, determine the category c such that

$$\pi_1 + \cdots + \pi_{c-1} < U_j \leq \pi_1 + \cdots + \pi_c,$$

and add 1 to the count t_c. Repeat this process for $j = 1, \dots, m$ to obtain $\boldsymbol{t}$.

To generate samples from the RCM distribution, begin with known parameters $(\pi_1, \dots, \pi_k, \rho)$. Generate $m+1$ observations from the standard multinomial distribution $\boldsymbol{S}, \boldsymbol{S}_1^0, \dots, \boldsymbol{S}_m^0 \overset{iid}{\sim} \mathrm{Mult}(\pi_1, \dots, \pi_k, 1)$, and m observations from the uniform distribution $U_1, \dots, U_m \overset{iid}{\sim} \mathrm{U}(0,1)$. The entries of the new RCM observation are given by

$$\boldsymbol{t}_j = \boldsymbol{S}\, I(U_j \leq \rho) + \boldsymbol{S}_j^0\, I(U_j > \rho), \qquad j = 1, \dots m,$$

where $I(\cdot)$ represents the indicator function.

Availability of this simple and intuitive algorithm of generating data is one of the many reasons for the choice of RCM as our test problem. It is identifiable for all values of $(\pi_1, \dots, \pi_k, \rho)$ without requiring any additional assumptions. Furthermore, the dimension of the parameter space is proportional to the number of categories k. In a more general mixture of multinomial densities there are $q = \nu(k-1) + (\nu - 1)$ distinct parameters, including $k-1$ category probabilities for each of the ν components, and $\nu - 1$ mixing proportions. Therefore, the dimension of the parameter space will blow up quickly as the number of categories or components is increased. Thus, the RCM model encompasses many numerical issues one may face in computing the MLE of a mixture model (e.g. multiple local maxima), without having to take on the full mixture. A further desirable property of RCM is that direct numerical optimization is effective for computing its MLEs. This is not the case for all mixture models, where a specialized approach like EM may be needed. In the next section, this property will be put to use as we discuss conducting the optimization in parallel.

3 Parallel MLE Computation

The High Performance Computing Facility (HPCF, `http://www.umbc.edu/hpcf`) at the University of Maryland, Baltimore County (UMBC) is an interdisciplinary, shared campus resource for scientific computing and research on parallel algorithms. The distributed-memory cluster `tara` has 86-nodes, each with two quad-core Intel Nehalem processors (2.66 GHz, 8 MB cache) and 24 GB memory, therefore up to 8 parallel processes can run simultaneously on each node. The nodes are connected by a high performance InfiniBand network, and run 64-bit Red Hat Enterprise Linux 5 as their operating system. We make use of the GNU C/C++ compiler, and the Open MPI 1.3.3 implementation of the Message Passing Interface (MPI) standard.

The Toolkit for Advanced Optimization (TAO, `http://www.mcs.anl.gov/research/projects/tao`) is an optimization library for both single-processor and massively-parallel architectures. It is built on top of the Portable, Extensible Toolkit for Scientific Computation (PETSc, `http://www.mcs.anl.gov/petsc`), a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. Both libraries are open source and were developed at Argonne National Laboratory. Both use

MPI for handling interprocess communications. We make use of TAO 1.10 and PETSc 3.0.0 on the `tara` cluster, with programming carried out in the C++ language. TAO and PETSc help to remove the burden of writing distributed code from the programmer. Management of distributed data structures can be left up to the libraries, allowing the programmer to focus on solving the problem at hand. This convenience also implies a loss of control, which may mean relinquishing the best possible performance.

TAO provides a suite of optimization algorithms and a framework to use them; these are described in detail in the user manual [2]. The programmer provides several key ingredients such as the objective function $h(\boldsymbol{\theta})$ to be optimized, the code to evaluate the gradient vector $\nabla h(\boldsymbol{\theta}) = \partial h(\boldsymbol{\theta})/\partial \boldsymbol{\theta}$, and the code to evaluate the Hessian matrix $\boldsymbol{H}(\boldsymbol{\theta}) = \partial^2 h(\boldsymbol{\theta})/\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T$. These three ingredients are used by the TAO algorithms to conduct a search for the optimal solution. Several algorithm choices are available for unconstrained optimization. The Nelder-Mead method is typically the worst performer, but requires only the objective function. The nonlinear conjugate gradient method and limited-memory, variable-metric (LMVM) method require both an objective and gradient function to be implemented. The Newton line search method uses the objective, the gradient, and also the Hessian. For this work we have considered only LMVM because it performed well, yet does not require explicit formation of the Hessian.

We now give a brief description of the LMVM method. Given a current solution $\boldsymbol{\theta}^{(i)} \in \mathbb{R}^q$, LMVM consists of two main steps to find the next solution $\boldsymbol{\theta}^{(i+1)}$. First the direction $\boldsymbol{d}$ of the next step is found by solving the linear system

$$\boldsymbol{H}^{(i)} \boldsymbol{d} = -\nabla h(\boldsymbol{\theta}^{(i)}),$$

where $\boldsymbol{H}^{(i)}$ is an approximation to the Hessian, which is computed within the method. Computation of the approximation utilizes a limited amount of information coming from previous steps. After the direction is obtained, a line search is performed to compute the size of the next step τ, so that

$$h(\boldsymbol{\theta}^{(i)} + \tau \boldsymbol{d})$$

is minimized. The next iterate is then given by $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \tau \boldsymbol{d}$. There are several tuning parameters available for TAO's LMVM method, but we leave them at their default settings, except that the iteration limit was set to a very large number to ensure convergence.

The objective function for the RCM MLE problem is the negative of the log-likelihood

$$h(\boldsymbol{\theta}) := -\log L(\boldsymbol{\theta} \mid \boldsymbol{X}) = -\sum_{i=1}^{n} \log \left\{ \sum_{j=1}^{k} \pi_j \left[\frac{m!}{t_{i1}! \cdots t_{ik}!} \eta_{j1}^{t_{i1}} \cdots \eta_{jk}^{t_{ik}} \right] \right\}, \tag{3}$$

based on the distribution discussed in Section 2. Recall that $\eta_{j\ell}$'s are functions of $\boldsymbol{\pi}$ and ρ. The log-likelihood is generally preferred over the likelihood in computations, because it involves a summation of many moderately-sized negative numbers rather than a product of many numbers of very small magnitude. TAO methods are set up to solve minimization problems; taking the objective to be the negative log-likelihood accomplishes maximization.

To use an unconstrained optimization method in this problem, we must address the natural constraints in the parameter space. That is, $\boldsymbol{\theta} = (\pi_1, \ldots, \pi_k, \rho)$ are all probabilities which must lie between 0 and 1, and π_i's must sum to 1. To allow the optimization to work in $\mathbb{R}^{k+1}$, two transformations are applied to any point $\boldsymbol{\theta}^* \in \mathbb{R}^{k+1}$ proposed by the optimizer. The logistic cumulative distribution function $e^x/(1+e^x)$ is first applied to each coordinate of $\boldsymbol{\theta}^*$ to enforce the $(0,1)$ range constraint. The summation constraint is then enforced by scaling the first k coordinates by their sum. Note that because of the scaling step, we do not use the fact that $\pi_k = 1 - \sum_{i=1}^{k-1} \pi_i$ to infer π_k. Therefore $q = k+1$ total parameters are under consideration. To compute the gradient vector needed for LMVM, a finite difference approximation

$$\frac{\partial h(\boldsymbol{\theta})}{\partial \theta_j} \approx \frac{h(\boldsymbol{\theta} + \delta \boldsymbol{e}_j) - h(\boldsymbol{\theta})}{\delta}, \qquad j = 1, \dots, q$$

is used with $\delta = 10^{-8}$. Notice that two objective function evaluations are needed to compute each component of the gradient.

There are several possible ways to achieve parallelization within the framework we have discussed. Observe that the log-likelihood in (3) can be quite expensive to evaluate, because it requires iterating over every component of the mixture for each observation in the sample. Each step of LMVM requires evaluation of the gradient, and each evaluation of the gradient requires $2q$ evaluations of the objective function. Additional evaluations of the log-likelihood may be required by LMVM as well, to carry out the line search, for example. Notice that the log-likelihood is a sum over n terms. One idea is to evaluate this sum over multiple parallel processes, and for large sample sizes we would expect good performance. We have chosen a different approach, however, which is to compute the q components $\partial h(\boldsymbol{\theta})/\partial \theta_j$ of the gradient in parallel. Notice that these components can be computed independently. This approach limits the number of parallel processes to the dimension of the problem, but has the advantage that it generalizes to any objective function. In this scheme, all sample data must be available on all processes in order to evaluate the log-likelihood. This could also be seen as a drawback if the sample data is very large, and perhaps may not fit in the memory of a single process. For the MLE problem described in this paper, memory requirements are on the order of kn to store a sample and k to store the parameters. Memory will not be an issue here, but may be an important consideration in other problems.

We are now prepared to write down the parallel MLE algorithm.

1. Split the indices $\{1, \dots, q\}$ as evenly as possible among the p available processes. Denote $\mathrm{Ind}(s)$ as the set of indices assigned to process s.

2. Start with an initial guess $\boldsymbol{\theta}^{(0)} = (\theta_1^{(0)}, \dots, \theta_q^{(0)})$.

3. Run an LMVM iteration, which will require evaluation of the gradient vector.

4. To evaluate the gradient, each process $s = 1, \dots, p$ computes

$$\frac{\partial h(\boldsymbol{\theta}^{(0)})}{\partial \theta_j}, \quad \forall j \in \mathrm{Ind}(s).$$

   At this point processes $1, \dots, p$ work in parallel. (Now each process has a fragment of the gradient vector in its local memory. To continue with the algorithm, we must make the entire vector available on all processes).

5. Make the entire vector $\nabla h(\boldsymbol{\theta}^{(0)})$ available on all processes. This is accomplished in MPI with a single command `MPI_Allgather`.

6. LMVM continues simultaneously on all processes, and a new guess $\boldsymbol{\theta}^{(1)}$ is obtained.

7. This process repeats, giving $\boldsymbol{\theta}^{(0)}, \boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(g)}$, until stopping criteria are met. Finally $\boldsymbol{\theta}^{(g)}$ is returned as the MLE.

For all simulations presented below, we select the total number of parameters q to be evenly divisible by p. This selection is taken for convenience and to demonstrate the ideal case of parallel performance with equal load balancing, but is not a limitation of the method itself. Some of the internals of LMVM potentially work in parallel as well (e.g., linear solves) which would further improve performance, but it is not immediately apparent if they are implemented this way. Stopping criteria are left to the TAO defaults, except that we have raised the limit on number of iterations, as mentioned earlier.

To summarize, the RCM MLE problem has been formulated as a TAO optimization problem. TAO was not strictly necessary to implement the idea, but is convenient because it features a variety of optimization routines and other utilities, and it has been designed to work in parallel. Our method of choice is LMVM; although this is an iterative method, we have identified pieces (components of the gradient vector) which can be computed in parallel within an iteration. We elected to use a finite difference approximation to the gradient, but a closed-form expression could be used if available. As mentioned earlier, standard numerical optimization works well for RCM, hence more specialized algorithms did not need to be considered. The chosen approach does not take into consideration very large datasets, instead it focuses on reduced computing time as the main goal.

It is worth making an important distinction between the task of computing the MLE for a single sample, and the task of approximating, say, its sampling variance using a bootstrap with 1000 repetitions. As mentioned earlier, the bootstrap is an embarrassingly parallel procedure, and each repetition can run independently so that results are combined only at the very end. Splitting the 1000 repetitions among the p available processes would eliminate almost all communication overhead, and would therefore be the preferred parallelization method for this case. Our approach is intended for problems where estimation for a single sample may be prohibitively expensive. As always, it is important to evaluate the overall computing task before deciding how parallelization should be handled.

4 Simulations

A series of simulations was carried out, first to verify that the estimation is working correctly ("consistency check"), and then to study parallel performance as problem sizes are varied ("performance experiments"). Problem size is determined by the following dimensions: sample size n, cluster size m, number of multinomial categories k (total parameters: $q = k + 1$), number of repetitions r, and number of MPI processes p.

To select the true parameters in a simulation, a symmetric vector $\boldsymbol{v} = (1, 2, 3, \ldots, 3, 2, 1)$ is generated which contains $k \in \{1, 2, 3, \ldots\}$ elements. Let $\boldsymbol{\pi} = \boldsymbol{v} / \sum_i v_i$, and $\rho = 1/4$ so that the true parameters are $\boldsymbol{\theta} = (\boldsymbol{\pi}, \rho)$. This construction provides a quick but deterministic construction of $\boldsymbol{\theta}$ for any valid choice of k. Given $\boldsymbol{\theta}$, a random sample of n observations $\boldsymbol{X} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$ is drawn from $\text{RCM}(\boldsymbol{\pi}, \rho, m)$ using the procedure described in Section 2. The objective function $h(\boldsymbol{\theta})$ given in (3) is constructed with this sample data. The TAO framework is then invoked with an initial guess $\boldsymbol{\theta}^{(0)} = (\boldsymbol{\pi}^{(0)}, \rho^{(0)})$, where $\boldsymbol{\pi}^{(0)} = (1/k, \ldots, 1/k)$ and $\rho^{(0)} = 1/2$. The selected optimization routine (LMVM) runs until stopping criteria are reached. If the optimization is successful, a maximum likelihood estimate $\widehat{\boldsymbol{\theta}}_{\text{MLE}}$ is obtained.

Repeating a simulation r times given a fixed $\boldsymbol{\theta}$ yields iid RCM samples $\boldsymbol{X}_1, \dots, \boldsymbol{X}_r$, and resulting estimates $\widehat{\boldsymbol{\theta}}^{(1)}_{\text{MLE}}, \dots, \widehat{\boldsymbol{\theta}}^{(r)}_{\text{MLE}}$.

4.1 Consistency Check for MLE

The consistency property can be used to provide some assurance that our parallel MLE program is computing estimates correctly. Morel and Nagaraj verify in [4] that the MLE is consistent and asymptotically normal for RCM. For any $\varepsilon > 0$,

$$\mathrm{P}\left(\|\widehat{\boldsymbol{\theta}}_{\text{MLE}} - \boldsymbol{\theta}\| \geq \varepsilon\right) \to 0 \quad \text{as } n \to \infty,$$

where we will take $\|\cdot\|$ to be the Euclidean norm. By Chebyshev's inequality,

$$\mathrm{P}\left(\|\widehat{\boldsymbol{\theta}}_{\text{MLE}} - \boldsymbol{\theta}\| \geq \varepsilon\right) \leq \frac{\mathrm{E}\|\widehat{\boldsymbol{\theta}}_{\text{MLE}} - \boldsymbol{\theta}\|}{\varepsilon},$$

and therefore it is sufficient to show that $\mathrm{E}\|\widehat{\boldsymbol{\theta}}_{\text{MLE}} - \boldsymbol{\theta}\| \to 0$ to verify consistency. We let

$$Q_{rn} = Q_{rn}(\boldsymbol{X}_1, \dots, \boldsymbol{X}_r) := \frac{1}{r}\sum_{j=1}^{r} \|\widehat{\boldsymbol{\theta}}^{(j)}_{\text{MLE}} - \boldsymbol{\theta}\|, \tag{4}$$

which is a Monte Carlo estimate for the expectation $\mathrm{E}\|\widehat{\boldsymbol{\theta}}_{\text{MLE}} - \boldsymbol{\theta}\|$. When r is fixed sufficiently large, a decrease in Q_{rn} for increasing n will be an indication that consistency is holding. Demonstrating consistency does not guarantee that the estimates are global maxima of the likelihood function, but it at least provides some evidence that the program yields a consistent solution to the MLE problem.

Sample data was generated and stored for each distinct setting of (n, k, m), for the maximum number of repetitions needed at that setting. This ensures that any two runs using the same dimensions (n, k, m, r) use exactly the same data, and thus their results are directly comparable. All runs are distributed in the same way across the HPCF cluster. For $p \leq 8$, a single compute node is used (each process will then run on its own dedicated core). For $p > 8$, we only consider p as a multiple of 8, and choose the number of nodes as $p/8$ so that all eight cores are utilized on each machine. Raim and Gobbert [11] demonstrate that the use of multiple cores per node is an effective strategy for distributing workloads on the HPCF cluster. All nodes used in each simulation are reserved exclusively for us by the scheduler, ensuring the results are free of interference from other cluster users' jobs.

Table 1 is intended to demonstrate consistency of the computed estimates, with k, m, p fixed and n varying from a moderate to large sample size. For each row of the table $r = 512$ repetitions of the simulation were carried out. The value of Q_{rn} is displayed, along with average number of LMVM iterations and average walltime (in seconds) per iteration. Table 2 shows similar results, but with n held fixed and m, k, and p changing individually. The quality of the solutions (but not necessarily their consistency) can be seen from this table as dimensions of the simulation is varied. Increasing r leads to the expected increase in walltime and an improvement of Q_{rn} in approximating $\mathrm{E}\|\widehat{\boldsymbol{\theta}}_{\text{MLE}} - \boldsymbol{\theta}\|$; these results are not shown.

In Table 1, it can be seen that Q_{rn} decreases as the sample size n increases. The quantity $\sqrt{n}Q_{rn}$ appears to stabilize around the value of 2.2 as n is increased, corresponding to the well-known property of the MLE bring root-n convergent. Therefore, the computed estimates appear to be consistent. Furthermore, the average walltime per repetition roughly doubles as the sample size is doubled, while the average number of LMVM iterations increases slowly. Table 2(a) shows that Q_{rn} is also decreased as the cluster size m is increased. The average walltime and number of iterations increases for large m, but also when the cluster size is very small ($m = 1$ or 2). Note that m enters the computations through factorials in the log-likelihood. These calculations were implemented with the optimized `lgamma_r` function in C rather than the naive recursive formula. Table 2(b) shows a dramatic increase in computational time as the number of categories k is doubled. This seems intuitive since the dimension of the optimization problem has dimension proportional to k. The average number of LMVM iterations per repetition also increases, although not as dramatically, while Q_{rn} does not appear to be affected in a systematic way. Comparing entries in Table 2(c), we see that changing the number of processes p does not have a significant effect on solutions — individual estimates across settings of p indeed agree to 4–5 decimal places — but increasing p decreases the required runtime as expected.

4.2 Performance Experiments

We consider the parallel performance of the RCM estimation problem when varying sample size n, cluster size m, and number of categories k. These dimensions are altered along with the number of processes p. Now that we have made sure the estimates are consistent solutions to the MLE problem, the number of repetitions r will be fixed at 1. We examine walltime as well as the metrics "speedup" and "efficiency", which are conventionally given in parallel performance studies. Let $c \in \{n, m, k\}$ be an experiment variable under observation. Define $T_p(c)$ as the walltime in seconds to compute a problem of size c using p processes. The speedup is defined as $S_p(c) = T_1(c)/T_p(c)$, where $S_p(c)$ close to p suggests ideal parallel performance. The efficiency is defined as $E_p(c) = S_p(c)/p$, where $E_p(c)$ close to 1 suggests ideal parallel performance. Whenever c is held constant and the number of processes p varies, the same sample data has been used. This helps to simplify comparisons between different settings of p. Parallel runs were compared to corresponding serial runs to ensure that results matched.

Table 3 and Figure 1 show the results of simulations varying n. We can see that for each fixed n, doubling the number of processes p shows a strong halving effect of the walltime, which weakens at about $p = 64$. The pattern is similar for all settings of n, except that larger n have slightly better scaling. This can readily be seen from the speedup and efficiency plots. Table 4 and Figure 2 show the results of simulations where m is varied. Again we see the definite halving effect in walltime as p is doubled, which starts to weaken around $p = 64$. In the speedup and efficiency plots, we can see that all settings of m show almost exactly the same scaling pattern.

Table 5 and Figure 3 show the results of simulations where k is varied. Recall in our parallelization scheme that the p processes divide the work of computing the $k + 1$ entries of the gradient vector. When $p \geq k$ some processes will be left with no useful work; therefore, these results have been omitted. Notice that for small k, the run time is too quick to justify parallelization. As k increases, run time drastically increases. For a fixed large k such as $k = 127$, doubling the number of processes p shows a strong halving effect of the walltime, which weakens as p approaches $k + 1$. This effect is also reflected in the speedup

Table 1: Results for consistency check with $m = 32$, $k = 7$, $p = 1$. using $r = 512$ repetitions.

n	Avg Walltime (Per Repetition)	Q_{rn}	Avg Iterations (Per Repetition)
32	0.0953	3.9723×10^{-2}	22.3027
64	0.1925	2.6488×10^{-2}	22.3418
128	0.3930	1.9318×10^{-2}	22.5469
256	0.7722	1.3410×10^{-2}	22.8633
512	1.5759	9.7332×10^{-3}	23.5664
1024	3.2304	6.6835×10^{-3}	23.8613
2048	6.5058	4.7491×10^{-3}	24.5527
4096	13.8054	3.4338×10^{-3}	25.7402

Table 2: Check for solution quality, varying m, k, p. For each row, $r = 512$ repetitions of the simulation were run.

(a) Vary m, using $n = 128$, $k = 31$, $p = 1$			
m	Avg Walltime	Q_{rn}	Avg Iterations
1	7.0399	3.8119×10^{-1}	11.3281
2	5.6237	2.5764×10^{-1}	8.6621
4	5.1445	2.5436×10^{-1}	7.8223
8	5.5690	2.5262×10^{-1}	8.0977
16	6.3344	2.5009×10^{-1}	8.7402
32	8.6227	1.4740×10^{-1}	10.9590
64	10.8879	1.3485×10^{-2}	12.1816
128	12.4226	9.5934×10^{-3}	12.5469
256	16.8323	6.7424×10^{-3}	16.2383
512	16.6591	4.8318×10^{-3}	15.7773
(b) Vary k, using $n = 64$, $m = 256$, $p = 1$			
k	Avg Walltime	Q_{rn}	Avg Iterations
3	0.0102	9.4829×10^{-3}	7.9297
7	0.0904	9.3881×10^{-3}	10.2695
15	0.8962	9.6047×10^{-3}	13.3301
31	8.2345	9.5747×10^{-3}	15.9141
63	59.6599	9.5986×10^{-3}	15.4277
(c) Vary p, using $n = 256$, $m = 256$, $k = 31$			
p	Avg Walltime	Q_{rn}	Avg Iterations
1	34.4564	4.7939×10^{-3}	16.5430
2	17.5231	4.7939×10^{-3}	16.5371
4	9.2531	4.7939×10^{-3}	16.5391
8	4.9527	4.7940×10^{-3}	16.5352
16	2.7603	4.7939×10^{-3}	16.5391
32	1.6717	4.7940×10^{-3}	16.5391

and efficiency plots. The most notable observation is the decrease in run time for $k = 127$ from about 12.7 minutes serially, to about 10 seconds when using all 128 processes. Similar results are obtained in Tables 3 and 4 where k is fixed at 127. Thus, for large enough problem sizes, adding parallel processes drastically reduces the walltime needed to compute MLEs.

5 An LRT application

Our performance studies have demonstrated the effectiveness of parallel computing for the RCM MLE problem when many parameters need to be estimated. But where might we encounter such problems in a data analysis? To answer this question, we will next consider hypothesis testing in a fixed effects model embedded into RCM. Even a fairly simple problem in this framework can be computationally expensive if there are a large number of multinomial categories and/or covariates. To create an effective demonstration, we will consider a scenario which is ideally suited to RCM. It will feature both a large number of categories and a simple fixed effects model. We will generate data for this scenario and show that computation of the likelihood ratio test (LRT) is one instance where practical use can be made of our parallel MLE idea.

Suppose there are n guidance counselors who advise high school students in selecting a college from k possibilities. For simplicity, suppose m students are assigned to each counselor, and no student is assigned to more than one counselor. A student visits his or her counselor zero or more times for advice until they have chosen a college, and may or may not be influenced by their counselor. Let $\boldsymbol{x} = (x_1, \dots, x_n)$, where x_i is the total number of visits to the ith counselor by their students. Intuitively, we might expect that a more heavily utilized counselor will have a greater influence on their students. One can imagine each student choosing between the counselor's recommendation and his/her own personal choice, as in the generation of RCM described in Section 2. This scenario is ideally modeled by RCM with parameter ρ capturing the degree of influence of the counselor.

Let $\boldsymbol{T}_i = (T_{i1}, \dots, T_{ik})$ denote the vector of counts for counselor i, for each of the k possible colleges. We can also suppose that the first $k-1$ categories represent specific college choices, and the kth category represents a catch-all for all other possibilities, such as attending an unlisted college or not attending any college at all. We will suppose that $\boldsymbol{T}_1, \dots, \boldsymbol{T}_n$ are independent, and

$$\boldsymbol{T}_i \sim \text{RCM}(\boldsymbol{\pi}, \rho_i, m), \qquad \log\left(\frac{\rho_i}{1-\rho_i}\right) = \alpha + \beta x_i. \tag{5}$$

Recall that ρ_i is the probability of "following the leader" from Section 2. In this scenario, following the leader means choosing the preferred college of the counselor. In 5 we have expressed the log-odds of ρ_i by a linear function, where α is the common baseline effect of a counselor's influence on students, and β is a common slope which incorporates how heavily students have utilized their counselor. Here $\boldsymbol{\pi}$ is constant across all counselors. Therefore aside from counselor influence, the probability distribution of choosing among the k colleges is the same for all students. For this problem, the unknown parameter $\boldsymbol{\theta}$ is contained in the space

$$\Theta = \left\{ (\pi_1, \dots, \pi_k, \alpha, \beta) \in \mathbb{R}^q : \; 0 < \pi_j < 1, \; \sum_{j=1}^{k} \pi_j = 1 \right\}, \quad q = k+2.$$

Table 3: Walltime, speedup, and efficiency varying n, for $k = 127$, $m = 64$, $r = 1$. Tests were performed with 8 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 4$ which uses 4 processes per node.

(a) Wall clock time in seconds

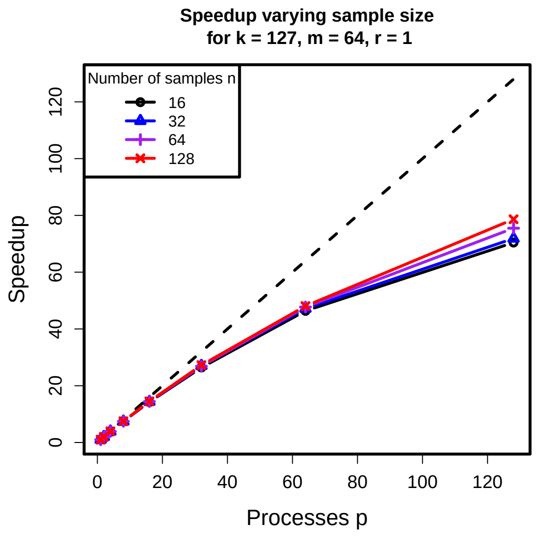
n	$p = 1$	2	4	8	16	32	64	128
16	150.20	81.16	40.22	20.13	10.53	5.69	3.24	2.13
32	217.70	109.42	55.86	29.36	15.17	8.06	4.62	3.03
64	323.73	165.90	82.59	43.30	22.43	12.05	6.78	4.29
128	646.91	325.20	164.33	86.37	44.64	23.73	13.46	8.23

(b) Observed speedup S_p

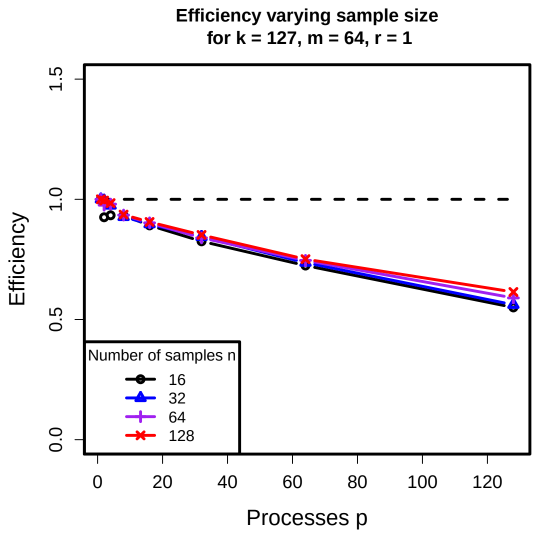
n	$p = 1$	2	4	8	16	32	64	128
16	1.00	1.85	3.73	7.46	14.26	26.41	46.36	70.43
32	1.00	1.99	3.90	7.42	14.35	27.00	47.14	71.86
64	1.00	1.95	3.92	7.48	14.43	26.87	47.74	75.48
128	1.00	1.99	3.94	7.49	14.49	27.26	48.07	78.65

(c) Observed efficiency E_p

n	$p = 1$	2	4	8	16	32	64	128
16	1.00	0.93	0.93	0.93	0.89	0.83	0.72	0.55
32	1.00	0.99	0.97	0.93	0.90	0.84	0.74	0.56
64	1.00	0.98	0.98	0.93	0.90	0.84	0.75	0.59
128	1.00	0.99	0.98	0.94	0.91	0.85	0.75	0.61



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 1: Scalability (speedup and efficiency) as n varies.

Table 4: Walltime, speedup, and efficiency varying m, for $n = 128$, $k = 127$, $r = 1$. Tests were performed with 8 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 4$ which uses 4 processes per node.

(a) Wall clock time in seconds

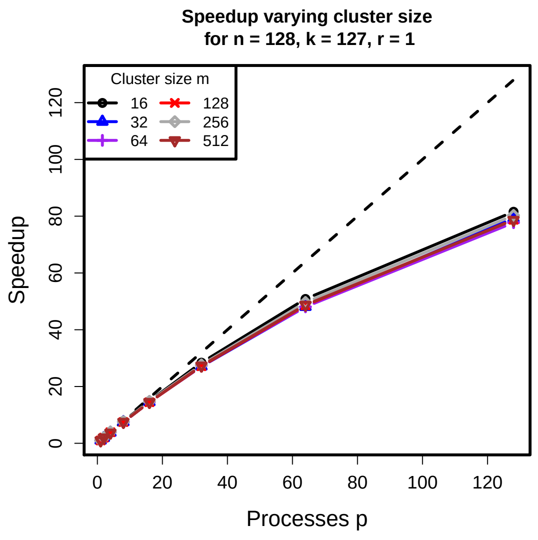
m	$p = 1$	2	4	8	16	32	64	128
16	539.51	263.01	132.24	68.90	36.31	19.00	10.61	6.61
32	697.73	366.01	185.50	93.34	48.04	25.77	14.51	8.80
64	646.84	331.35	164.44	86.34	44.56	23.61	13.46	8.32
128	616.36	310.00	158.39	82.35	42.56	22.70	12.69	7.83
256	777.63	384.21	194.77	102.01	52.62	28.15	15.67	9.69
512	1056.65	531.78	285.14	141.10	72.81	38.69	21.68	13.44

(b) Observed speedup S_p

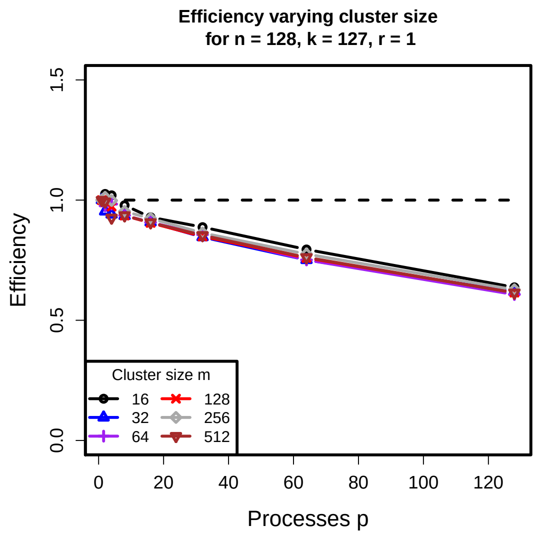
m	$p = 1$	2	4	8	16	32	64	128
16	1.00	2.05	4.08	7.83	14.86	28.40	50.85	81.56
32	1.00	1.91	3.76	7.48	14.52	27.07	48.07	79.25
64	1.00	1.95	3.93	7.49	14.52	27.40	48.07	77.72
128	1.00	1.99	3.89	7.48	14.48	27.16	48.58	78.73
256	1.00	2.02	3.99	7.62	14.78	27.62	49.63	80.28
512	1.00	1.99	3.71	7.49	14.51	27.31	48.75	78.63

(c) Observed efficiency E_p

m	$p = 1$	2	4	8	16	32	64	128
16	1.00	1.03	1.02	0.98	0.93	0.89	0.79	0.64
32	1.00	0.95	0.94	0.93	0.91	0.85	0.75	0.62
64	1.00	0.98	0.98	0.94	0.91	0.86	0.75	0.61
128	1.00	0.99	0.97	0.93	0.91	0.85	0.76	0.62
256	1.00	1.01	1.00	0.95	0.92	0.86	0.78	0.63
512	1.00	0.99	0.93	0.94	0.91	0.85	0.76	0.61



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 2: Scalability (speedup and efficiency) as m varies.

Table 5: Walltime, speedup, and efficiency varying k, for $n = 128$, $m = 256$, $r = 1$. Tests were performed with 8 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 4$ which uses 4 processes per node.

(a) Wall clock time in seconds

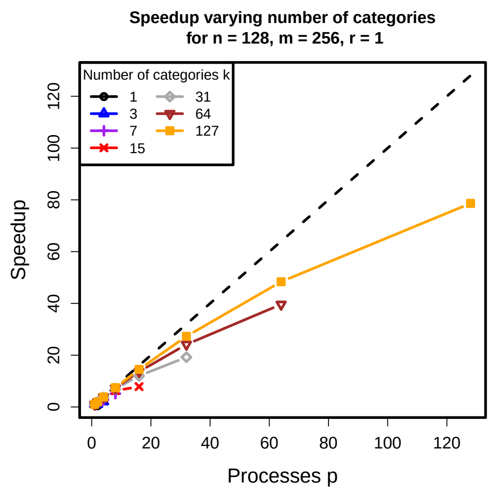
k	$p = 1$	2	4	8	16	32	64	128
1	0.001	0.003	—	—	—	—	—	—
3	0.025	0.025	0.012	—	—	—	—	—
7	0.175	0.121	0.063	0.034	—	—	—	—
15	1.873	1.006	0.543	0.297	0.238	—	—	—
31	16.714	8.538	4.556	2.397	1.389	0.870	—	—
63	131.692	67.555	33.924	18.076	9.660	5.452	3.324	—
127	763.255	384.019	197.899	102.115	52.572	27.962	15.777	9.706

(b) Observed speedup S_p

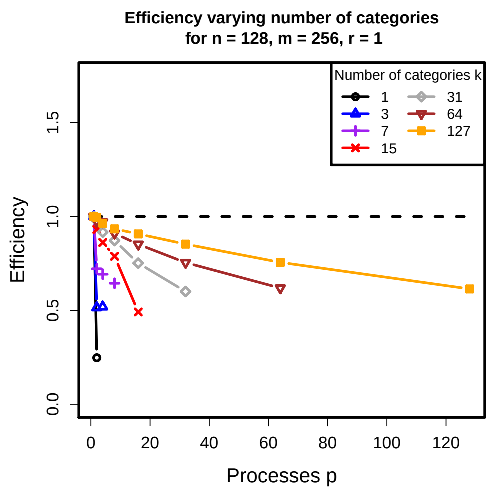
k	$p = 1$	2	4	8	16	32	64	128
1	1.00	0.50	—	—	—	—	—	—
3	1.00	1.03	2.08	—	—	—	—	—
7	1.00	1.44	2.77	5.16	—	—	—	—
15	1.00	1.86	3.45	6.30	7.87	—	—	—
31	1.00	1.96	3.67	6.97	12.03	19.22	—	—
64	1.00	1.95	3.88	7.29	13.63	24.15	39.62	—
127	1.00	1.99	3.86	7.47	14.52	27.30	48.38	78.64

(c) Observed efficiency E_p

k	$p = 1$	2	4	8	16	32	64	128
1	1.00	0.25	—	—	—	—	—	—
3	1.00	0.51	0.52	—	—	—	—	—
7	1.00	0.72	0.69	0.64	—	—	—	—
15	1.00	0.93	0.86	0.79	0.49	—	—	—
31	1.00	0.98	0.92	0.87	0.75	0.60	—	—
64	1.00	0.97	0.97	0.91	0.85	0.75	0.62	—
127	1.00	0.99	0.96	0.93	0.91	0.85	0.76	0.61



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 3: Scalability (speedup and efficiency) as k varies.

We will consider a testing problem for the significance of the slope,

$$H_0 : \beta = 0 \qquad \text{vs.} \qquad H_1 : \beta \neq 0.$$

Two MLE computations are needed for the LRT: the unrestricted MLE $\widehat{\boldsymbol{\theta}}$, and the MLE $\widehat{\boldsymbol{\theta}}_0$ under the restriction H_0. The LRT statistic can then be computed as

$$-2\log\Lambda = -2\log\frac{L(\widehat{\boldsymbol{\theta}}_0)}{L(\widehat{\boldsymbol{\theta}})} = -2\left\{\log L(\widehat{\boldsymbol{\theta}}_0) - \log L(\widehat{\boldsymbol{\theta}})\right\}, \tag{6}$$

where the likelihood is given by

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(\boldsymbol{t}_i \mid \boldsymbol{\pi}, \rho_i(\alpha,\beta), m) \tag{7}$$

and f is the density of RCM.

As before, solving the likelihood equation in closed form is not practical, and we turn to numerical computation of the MLE. This can be accomplished in parallel by applying the method from Section 3 and embedding the new likelihood (7) into an objective function. For this application we use the initial guess $\boldsymbol{\theta}^{(0)} = (\boldsymbol{\pi}^{(0)}, \alpha^{(0)}, \beta^{(0)})$, where $\boldsymbol{\pi}^{(0)} = (1/k, \ldots, 1/k)$, $\alpha^{(0)} = 0$, and $\beta^{(0)} = 0$.

To generate data from this scenario, we select a number of categories k and sample size n, and let $m = 100$ students per counselor. The category probabilities are generated by drawing a random sample $U_1, \ldots, U_k$ from $U(0,1)$ and then letting $\pi_j = U_j / \sum_i U_i$. To generate the covariate $\boldsymbol{x}$, we suppose $x_{i1}, \ldots, x_{im} \overset{iid}{\sim} \text{Geometric}(\phi)$, where x_{ij} represents the number of visits of the jth student of counselor i until a college decision is made. We choose $\phi = 0.9$ so that the expected number of visits per student $\text{E}(x_{ij}) = (1-\phi)/\phi = 1/9$ is small. The total number of visits x_i to counselor i are then drawn independently from $\text{NegBin}(m, \phi)$. We let $\alpha = -5$ and $\beta = 0.3$, so that

$$\log\left(\frac{\rho_i}{1-\rho_i}\right) = -5 + 0.3\,x_i \qquad \Longleftrightarrow \qquad \frac{\rho_i}{1-\rho_i} = e^{-5}(e^{0.3})^{x_i},$$

and thus the odds of "following the leader" will be multiplied by $e^{0.3} \approx 1.35$ for each visit to the counselor. In this scenario, utilization of the counselor has a fairly strong influence over college choice, but students do not tend to make much use of their counselor. Now that all of the parameters and covariates have realized values, RCM responses are generated according to (5) using the algorithm given in Section 2.

Table 6 shows the results of our computations on two problems: $(k = 50, n = 500)$ and $(k = 98, n = 1000)$. For each problem, two cases are shown which correspond to the two likelihood maximizations that need to be computed. In each case results from three runs are shown: one for TAO code in serial, one for TAO code in parallel, and one for a simple implementation in R using the `optim` function with the built-in BFGS optimization method. The maximized log-likelihood, walltime in hours:minutes:seconds format, and number of iterations are shown for each run. The number of iterations for serial R was limited to 100, which is the default setting. For each parallel TAO run, we choose a moderately sized p which evenly divides the number of parameters, and used the smallest number of compute

Table 6: Results for LRT computations of generated application problems. For problem size (a), the parallel full space run used 13 processes across 2 nodes, and the parallel restricted run used 17 processes across 2 nodes. For problem size (b), the parallel full space run used 20 processes across 3 nodes, and the parallel restricted run used 11 processes across 2 nodes.

(a) Results for $k = 50, n = 500$

case	#params	run	log-lik	walltime	#iters
Under H_0	51	serial R	-40791.06	02:53:43	57
		serial TAO	-40791.06	00:03:14	14
		parallel TAO	-40791.06	00:00:16	14
Full space	52	serial R	-37284.64	03:21:25	60
		serial TAO	-37284.63	00:05:53	25
		parallel TAO	-37284.63	00:00:35	25

(b) Results for $k = 98, n = 1000$

case	#params	run	log-lik	walltime	#iters
Under H_0	99	serial R	-111241.20	48:54:17	100
		serial TAO	-111241.23	01:12:15	24
		parallel TAO	-111241.23	00:07:43	24
Full space	100	serial R	-104468.55	49:10:42	100
		serial TAO	-104467.38	02:02:48	40
		parallel TAO	-104467.38	00:07:20	40

nodes possible to run each job. All parallel TAO jobs shown in Table 6 were run on either two or three nodes, and 11 to 20 processes overall. As in Section 4, the number of parameters for both problems here were chosen deliberately for convenience so that the work could be split evenly across a moderate number of processes.

Notice first that for each case, the values of the log-likelihoods attained across all three runs are nearly the same. This gives a cross-check between the TAO and R codes that correct solutions to the MLE problem are computed. The iteration counts match between serial and parallel TAO runs, but R required significantly more iterations. In fact, in the larger problem the iteration limit of 100 has been reached, hence further improvement may have been possible. Also, notice that in the restricted case of the larger problem, R has managed to find a slightly better solution than TAO. These issues are not necessarily cause for alarm, since there may be differences between the two optimization methods and their implementations.

We see that the R code is dramatically slower than the serial TAO code. For instance, the larger problem required over two days in R to solve either case, whereas the serial TAO code required only about 1 to 2 hours. This observation should be viewed in light of the fact that neither the R nor TAO codes were carefully tuned for performance. Moving from serial TAO to parallel TAO, we see that either case of the larger problem can now be solved in about 7 to 8 minutes using at most 20 processes on 3 nodes. Therefore computing the entire LRT for the larger problem has taken about 4 days in serial R, compared to about 3:15 hours in serial TAO and about 15 minutes in parallel TAO. From Section 4, we would expect performance to scale well with additional parallel processes.

Finally, we may compute the LRT for the two problems using the results in Table 6. For the $(k = 50, n = 500)$ problem we have $-2 \log \Lambda = 7012.87$. For the $(k = 98, n = 1000)$ problem the test statistic evaluates to $-2 \log \Lambda = 13547.70$. Under the $-2 \log \Lambda \simeq \chi^2_1$ approximation, we may correctly reject H_0 in both cases and conclude that the number of visits x_i to a counselor has a significant effect on the log-odds of "following the leader".

In this problem we considered a simple model with a single covariate but a large number of categories in the response. Notice that a model with many covariates and perhaps a smaller number of categories also leads to a many-parameter situation, where parallel computing would be useful for obtaining MLEs. Notice that covariates can easily be added for the counselor, or perhaps at a less granular level such as the high school which employs the counselor or their geographical region. Covariates at the student level are more granular however, and would complicate the model significantly. Adding covariates at the student level would require the $\boldsymbol{T}_i$'s to be split into smaller observations sharing common counselors, and hence the observations would no longer be independent.

6 Conclusions

We have demonstrated the effectiveness of computing MLEs in parallel using the Random-Clumped model as a test problem. TAO provided an environment to conduct numerical optimizations in parallel, requiring only an objective function, gradient vector, and Hessian matrix. The MLE procedure is just one example of an application that can benefit from this kind of parallel optimization.

The effects of adjusting sample size, cluster size, and number of categories were studied by simulation. Increasing the sample size verified that the computed estimates were consistent. Increasing the number of categories quickly caused run time to increase. Increasing the cluster size increased run time to a lesser extent, and it also improved the quality of estimates. The parallel performance was also studied, varying the number of processes along with sample size, number of multinomial categories, and cluster size. We observed excellent parallel performance when varying sample size and cluster size. The best parallel performance is possible when the number of categories is large. However, increasing the number of categories causes run time to increase faster than linearly. Therefore, RCM likelihoods with a very large number of categories will be infeasible to maximize, even on a large cluster, using the basic approach presented here. Smaller problems involving many repetitions do not require a high performance computing cluster, since repetitions are computationally independent and require little communication. This kind of parallelism can be accomplished with less elaborate programming using tools like the SNOW package for R.

Finally, we saw how the parallel MLE approach could be used in a more realistic RCM analysis. We used this method to compute the numerator and denominator of the LRT, and we conducted a test for the slope in a linked model. This yielded a significant improvement in performance using only a few compute nodes.

The approach used here can be applied to statistical computations in general; particularly, we have exploited it to compute MLEs for the Random-Clumped model. For this model, useful theoretical results are available to vastly improve the performance of MLE computation. For example, Neerchal and Morel [8] suggest a block diagonal approximation for the Fisher information matrix, which can be used to effectively carry out Fisher Scoring iterations. Further improvements are proposed in [6], in the context of EM. Incorporating these results could yield vastly improved performance, and perhaps new opportunities for applications of the RCM model.

Acknowledgments

The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant no. CNS–0821258) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See http://www.umbc.edu/hpcf for more information on HPCF and the projects using its resources. The first author additionally acknowledges financial support as HPCF RA.

References

[1] G.H. Givens and J.A. Hoeting, *Computational Statistics*, Wiley-Interscience, 2005.

[2] S. Benson, L.C. McInnes, J. Moré, T. Munson, and J. Sarich, *TAO User Manual (Revision 1.9)*, ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2007 http://www.mcs.anl.gov/tao.

[3] R. Malouf, *A comparison of algorithms for maximum entropy parameter estimation*, in *COLING-02: Proceedings of the 6th Conference on Natural Language Learning*, Association for Computational Linguistics, 2002, pp. 1–7.

[4] J.G. Morel and N.K. Nagaraj, *A finite mixture distribution for modelling multinomial extra variation*, Biometrika 80 (1993), pp. 363–371.

[5] A. Rossini, L. Tierney, and N. Li, *Simple Parallel Statistical Computing in R*, Working Paper 193, UW Biostatistics Working Paper Series, 2003 http://www.bepress.com/uwbiostat/paper193.

[6] M. Liu, *Estimation for Finite Mixture Multinomial Models*, PhD Thesis, University of Maryland, Baltimore County, Department of Mathematics and Statistics, 2005.

[7] T. Banerjee and S. Paul, *Miscellanea. An extension of Morel-Nagaraj's finite mixture distribution for modelling multinomial clustered data*, Biometrika 86 (1999), pp. 723–727.

[8] N.K. Neerchal and J.G. Morel, *Large Cluster Results for Two Parametric Multinomial Extra Variation Models*, Journal of the American Statistical Association 93 (1998), pp. 1078–1087.

[9] H. Zhou and K. Lange, *MM Algorithms for Some Discrete Multivariate Distributions*, Journal of Computational and Graphical Statistics 19 (2010), pp. 645–665.

[10] N.K. Neerchal and J.G. Morel, *An improved method for the computation of maximum likelihood estimates for multinomial overdispersion models*, Computational Statistics & Data Analysis 49 (2005), pp. 33–43.

[11] A. M. Raim and M. K. Gobbert, *Parallel performance studies for an elliptic test problem on the cluster tara*, HPCF–2010–2, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2010.