SOLVING FINITE MIXTURE MODELS IN PARALLEL

Christopher Ferrall, Queen's University
http://qed.econ.queensu.ca/pub/faculty/ferrall

March 31, 2003

Abstract:
Many economic models are completed by finding a parameter vector θ that optimizes a function $f(\theta)$, a task that only be accomplished by iterating from a starting vector θ_0. Use of a generic iterative optimizer to carry out this task can waste enormous amounts of computation when applied to a class of problems defined here as finite mixture models. The finite mixture class is large and important in economics and eliminating wasted computations requires only limited changes to standard code. Further, the approach described here greatly increases gains from parallel execution and opens possibilities for re-writing objective functions to make further efficiency gains.

JEL Classification: C61, C63, D58

Keywords: Numerical Optimization, Heterogeneous Agent Models

Correspondence:

C. Ferrall, Economics, Dunning Hall, Queen's University, Kingston, Ont. K7L 3N6, CANADA,

ferrallc@post.queensu.ca

Code that implements the algorithm described is available from the author. It runs in serial and in parallel using the MPI library. Research support from the Social Sciences and Humanities Research Council of Canada and the Social Research Demonstration Corporation is gratefully acknowledged. I also benefitted from presentations at Penn State, the Federal Reserve Bank of Cleveland, NYU, Stony Brook, and the Yale meeting of the Society for Computational Economics.

I. Introduction

A basic computational task in economics is to optimize an objective $f(\theta)$ by choosing the value of a $P \times 1$ vector θ,

$$\theta^\star = \arg\max_{\theta \in \Theta} f(\theta). \tag{1}$$

When closed form solutions to (1) are not available the objective must be optimized using an iterative algorithm. Since at least Goldfeld et. al (1966) the most common strategy to solve (1) has been to:

- Write a computer procedure that takes as input the vector θ then evaluates and returns $f(\theta)$;
- Write a program that calls an optimization package, $\bar{V}(f(\cdot), \theta_0)$, that starts one of several iterative algorithms at some value θ_0 and returns as output a vector $\bar{\theta} \approx \theta^\star$.

An all-purpose or generic optimizer such as $\bar{V}(f(\cdot), \theta_0)$ typically offers a menu of algorithms designed to handle different types of objectives, for example smooth, continuous, or discontinuous objectives and constrained or unconstrained parameter vectors. Given the choice of algorithm, $f(\cdot)$ is treated as a black-box and generic optimization is applicable in virtually any context. This versatility comes at a cost when there are internal features of $f(\cdot)$ that, if exploited, would reduce the computations required to repeatedly evaluate $f(\theta)$.

This paper studies a particular class of optimization problems referred to as finite mixture models. A single evaluation of $f(\theta)$ in a finite mixture model requires solving several costly sub-problems. In addition, some parameters are dedicated to a particular sub-problem but the solutions are combined or mixed so that the contribution of parameters to $f(\theta)$ is not-separable. These are the hallmarks of a canonical economic model: estimation of parameters of an heterogeneous agent economy using generalized method of moments. Special cases of this general problem include calibrated dynamic equilibrium economies with heterogeneous agents (Rios-Rull 1999) and micro-econometric models with (finite support) unob-

served heterogeneity (Heckman-Singer 1984). A pioneering finite mixture model combining equilibrium, heterogeneity, and consistent estimation is Eckstein and Wolpin (1990).

The main point of this paper is that optimizing an objective in the finite mixture class using a generic optimizer results in many redundant calculations. This is fairly obvious in principle, but without proper notation the exact amount of redundancy is not obvious due to the combinatoric nature of optimizing a finite mixture model iteratively. With proper notation the point is made with some algebra. Further, many of these redundant calculations are avoided by making limited changes to a generic optimizer. In particular, the generic $\bar{V}(f, \theta_0)$ that makes direct calls to $f(\theta)$ is replaced by an optimizer $\tilde{V}(\tilde{f}, \theta_0, \sigma)$ that communicates with $f(\theta)$ through a user-written interface $\tilde{f}(c, b)$. The argument σ provides information on the internal structure of $f(\theta)$ and the argument c specifies the sub-task to be carried out.

The second point of this paper is that $\tilde{V}$ executes in parallel more efficiently than a generic optimizer without requiring the user of $\tilde{V}$ to write parallel code. This is important, because parallel execution is a major source of increased computing capacity over the last decade, yet use of parallel algorithms in economics has developed slowly. Writing parallel code is seen by even experienced programmers as an onerous task, and except in special cases non-linear optimization algorithms are not obviously parallel. (That is, code running on two processors may not finish much faster than code running on one processor.) The separable sub-tasks in the finite mixture model are inherently parallel, but only if they are not hidden inside a blackbox objective function. Thus, the interface $\tilde{f}$ allows the optimizer $\tilde{V}$ to exploit the returns to parallel execution inherent in finite mixture models and to do so without requiring parallel code inside the user-defined $\tilde{f}$.

Reducing the computational cost of solving finite mixture models, even if based on obvious facts and mundane modifications to code, is important because quantitative economics is a "compute bound" task. If economic models were not growing in complexity then exponential increases in computing power would result in the computational time required by every published paper to converge to zero seconds. This appears not to be the case. Instead,

the size of problems computed expands with falling computing costs indicating constrained modelling choices. Further, $\tilde{V}$ may not be a neutral technological change. It may affect which models are formulated in the first place. Finite mixture models arise in quantitative economics when a single model accounts for several simultaneous concerns. The three basic concerns most models account for are individual rationality (utility and profit maximization), internal consistency (competitive or strategic equilibrium), and external consistency (explaining or fitting data). What distinguishes the various literatures within the finite mixture class are the algorithms internal to $f(\theta)$ that account for these concerns and differing weights assigned to these concerns. For example, some work emphasizes statistically consistent estimates of parameters at the exclusion of equilibrium restrictions. Other work imposes costly equilibrium conditions but does not select parameters to conform to external data.

The third point of this paper is that re-organizing θ and $f(\theta)$ to reduce costs of computation also opens the possibility of balancing competing concerns that shape the overall objective $f(\theta)$. This point is also fairly obvious given the proper notation. Use of a generic optimizer encourages the modeler to *nest* competing concerns. For example, the optimizer $\bar{V}$ might be used to choose parameters to maximize a likelihood function given that other parameters, such as prices, are computed internally within $f(\theta)$. In this sense the concern for internal consistency is nested within a concern of external consistency. An alternative is to let the optimizer choose prices along with other parameters based on an overall objective that *balances* competing concerns. In other words, a constrained optimization problem can be converted into a unconstrained (Lagrangian) problem. What is less obvious is whether which statement of the problem is better without solving both, a self-defeating exercise. This paper provides conditions to check whether re-coding a given model to balance rather than nest competing concerns should be expected to reduce the time required to find a solution.

Section II defines the finite mixture class and introduces the parallel computing environment that $\tilde{V}$ is designed to operate under. The finite mixture class is illustrated by some simple examples and by casting the model Lee (2001) solves as a generic problem into a finite

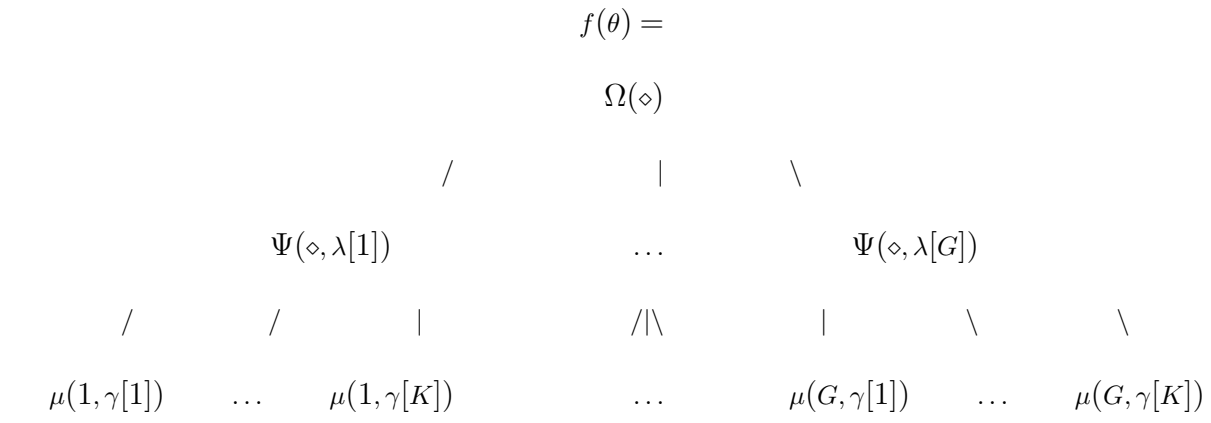
mixture model. Common problems in economics that will not gain from casting them as finite mixtures are also discussed. Section III describes three standard optimization routines to highlight that redundant calculations in finite mixture models exist across the spectrum of optimization algorithms currently used in economics. No new optimization algorithm is proposed here, although a straightforward improvement to simulated annealing arises naturally when applying it to finite mixture models. Section IV provides the main results on the computational inefficiency of generic optimization executed both in serial and in parallel. Section V considers the tradeoff in using either a nested or balanced representation of a given finite mixture model.

II. Finite Mixture Models

II.A Notation

The elements of a finite mixture model include scalars, vectors, functions, and tasks. A sub-vector or scalar element of a vector v is denoted $v[s]$ where the appropriate sub-vector indexed by the scalar s is defined somewhere in the text. The notation $v[s,t]$ is short for a sub-vector of a sub-vector, i.e. $v[s][t]$. Let $\mathcal{Z}$ denote the set of integers through Z, $\{1,2,\ldots,Z\}$, for any integer $Z>0$. Let $\mathcal{I}$ be the set of positive integers ($Z=\infty$). The length of a vector v is denoted $|v|\in\mathcal{I}$. Parameters of the problem are integers denoted with capital Roman letters, such as G and N. Arguments to a function are placed inside round brackets, ( ). The output of a function is itself a vector denoted with the same symbol. For example, the result of computing the function $\mu(g,\gamma[k])$ is a vector denoted $\mu[g,k]$. A task τ is a composite function that can implicitly require the solution of other functions as sub-tasks. The computational cost of a task τ is denoted $C\{\tau\}$. When execution is serial, computational cost are interpreted as seconds of processor time. When discussing parallel execution of a task some care will be required in interpreting costs, since total processing time is not the same as elapsed time.

Figure 1. Diagram of a Finite Mixture Model



II.B Definition

The objective in a finite mixture is a function $f : \Re^P \to \Re$. Computing f involves three layers of sub-tasks written:

$$f(\theta) = \Omega_{g=1}^{G} \left(\Psi_{k=1}^{K} \left(\mu \left(g, \gamma[k] \right), \lambda[g] \right) \right), \tag{2}$$

where

$$\mu : \mathcal{G} \times \Re^N \to \Re^M, \qquad \Psi : \Re^{KM} \times \Re^K \to \Re^Q, \qquad \text{and } \Omega : \Re^{GQ} \to \Re. \tag{3}$$

Figure 1 represents the objective function (2) as a hierarchy of tasks. The innermost task $\mu(\cdot)$ is a numerically-solved model or problem. It takes as parameters the scalar g and the vector $\gamma[k]$. The first is an index for one source of heterogeneity across problems, and g will be referred to as the observed group. It also can be thought of indexing the environments that agents may be find themselves in. The vector $\gamma[k]$ is of length N and contains parameters of μ that can be varied to optimize $f(\theta)$, which can vary for each type k, the other source of heterogeneity. Here k will be referred to as the latent or unobserved type.

The model is re-solved for each combination of k and g.[1] The symbols Ψ and Ω denote functions that transform their arguments and pass the result to the next level. The

[1] When certain combinations of k and g are impossible the model μ can do nothing.

task Ψ uses the concatenated output vectors of the model, $\mu[g,1] \quad \mu[g,2] \quad \cdots \quad \mu[g,K]$, to evaluate the result for observed type g. The weight of types is determined by the K-vector $\lambda[g]$. The outermost task Ω aggregates the concatenated contributions of each group, $[\,\Psi[1] \quad \Psi[2] \quad \ldots \quad \Psi[G]\,]$, into a real number, $\Omega[1] = f(\theta)$.

Assumptions and Requirements.

Ra. Five positive integers define the organization of $f(\theta)$:

$$\sigma = (\,G \quad K \quad N \quad M \quad Q\,), \tag{4}$$

where G and K are the number of observed and unobserved groups, N is the length of $\gamma[k]$, and M and Q are finite bounds on the output produced by sub-tasks μ and Ψ. That is, $|\mu[g,k]| \leq M$ and $|\Psi[g]| \leq Q$ for all g and k.

Rb. The parameter vector θ contains λ and γ in a pre-specified order:

$$\theta = \begin{pmatrix} \lambda \\ \gamma \end{pmatrix} = \begin{pmatrix} \lambda[1] \\ \lambda[2] \\ \vdots \\ \lambda[G] \\ \gamma[1] \\ \gamma[2] \\ \vdots \\ \gamma[K] \end{pmatrix}. \tag{5}$$

It follows that $P = |\theta| = K(G+N)$. The domain of the objective is $\Theta \subset \Re^P$.[2]

Rc. $\Psi_{k=1}^{K}$ is homogeneous of degree 0 in $\lambda[g]$, so without loss of generality $\sum_k \lambda[g,k] = 1$. The number of unconstrained is less than or equal to $P - G = K(G+N) - G \equiv P^U$.

Rd. Aggregation is costless: $C\left\{\Omega_{g=1}^{G}\right\} = 0$; and evaluation is subject to constant returns to scale: $C\left\{\Psi_{k=1}^{K}\right\} = KC\{\Psi_1\}$.

Some of the requirements simply choose one of many equivalent ways to describe θ and $f(\cdot)$. Assuming that the aggregation task is costless simplifies many of the expressions later

[2] Constrained iterative optimization can be carried out using transformations of the parameter vector or using augmented (penalized) objective functions. For example, see Judd (1998). The key results about solving finite mixture models developed here survive the presence of such constraints.

on, but otherwise is not required. Typically, $\Omega = \sum$ or $\Omega = \sum \ln(\cdot)$, and in either case the assumption is essentially true relative to the cost of solving a complex underlying model. If it were not true, then the cost of computing the underlying model is so small that the issues addressed here are not important.

Definition 1. Finite Mixture Model. Let $f(\theta) \colon \Theta \to \Re$ take the form (2) and let $\mathcal{C}_P^0$ denote the set of such functions. The task $\tilde{V} \colon \mathcal{C}_P^0 \times \Re^P \times \mathcal{I}^5 \to \Re^P$ is a finite mixture model when

$$\tilde{\theta} \equiv \tilde{V}\left(f(\cdot), \theta_0, \sigma\right) \approx \arg\max_{\theta \in \Theta} f(\theta). \tag{6}$$

There is a simple but critical implication of identifying a finite mixture model not with the objective itself but the choice of parameters that optimize the objective. The finite mixture model is not solved just by evaluating $f(\theta)$ but only when the optimal parameter vector is found. Therefore, any evaluations of $f(\theta)$ during iterative optimization are properly seen as intermediate results required to compute $\tilde{\theta}$. Once $\tilde{\theta}$ is found the model may be applied in completely different ways. For example, policy experiments are to be carried out or evidence for competing theories is to be assessed.

What matters then is the total cost of computing $\tilde{\theta}$ and not simply the cost of computing $f(\theta)$. The idea is not uncommon. Any algorithm akin to the "EM" algorithm exploits the fact that function evaluations during optimization are simply means to an end that need not embody all the restrictions ultimately imposed at the final parameter vector. As an example of an important finite mixture model, consider the estimation of a discrete choice dynamic program. Aguirregabiria and Mira (2002) propose an extension of the algorithm of Hotz and Miller (1993), which itself was motivated in part by avoiding calculations embedded in Rust's (1994, 1996) nested fix point algorithm. The Aguirregabiria and Mira procedure uses the fact that a joint concern for consistency and individual rationality does not imply that it is optimal to impose individual rationality exactly while searching for consistent parameters. Recently, Arcidiacono and Jones (2002) consider a balanced approach to these problems in

the presence of unobserved heterogeneity. Imai et al. (2002) describe a Bayesian approach to balancing the concern for individual rationality. The point of this paper is that similar results apply to the much broader class of problems in computational economics in which one solution of the model requires two or more objectives to be optimized.

Definition 2. Related Problems and Special Cases. Let $\tilde{V}\left(f(\cdot), \theta_0, \sigma\right)$ be a finite mixture model.

1. *Blackbox optimization* is a finite mixture model with $G = K = M = Q = 1$:

$$\theta = \begin{pmatrix} 1.0 \\ \gamma[1,1] \end{pmatrix}$$
$$f(\theta) = \Omega\left(\Psi\left(\mu\left(1, \gamma[1]\right)\right)\right) = \mu\left[1,1\right]. \tag{7}$$

   where $\lambda[1,1] = 1.0$ is the redundant weight of the single latent type in the single observed group[3], and Ψ and Ω are redundant tasks.

2. The *blackbox representation* of a finite mixture model is:

$$B\{\tilde{V}\} = \tilde{V}(\tilde{f}(\cdot), \theta_0, \sigma^B[\sigma]), \tag{8}$$

   where $\sigma^B[\sigma] = (\,1 \quad 1 \quad K(N+G) \quad 1 \quad 1\,)$.

3. Optimization of a *partially separable objective* is the case: $Q = M = G = 1 < K$; $\Psi = \sum$; and λ redundant:

$$f(\theta) = \sum_{k=1}^{K} \mu\left(1, \gamma[k]\right). \tag{9}$$

4. Let $\tilde{V}_n\left(f_n(\cdot), \theta_0, \sigma_n\right)$ and $\tilde{V}_b\left(f_b(\cdot), \theta_0, \sigma_b\right)$ be two representations of a particular finite mixture model. Then f_b is said to be *balanced* and f_n is said to be *nested* objectives relative to each other if: $\quad N_b \geq N_n; \quad C\left\{\Psi_b\right\} \leq C\left\{\Psi_n\right\}; \quad$ and $C\left\{\mu_b\right\} \leq C\left\{\mu_n\right\}$.

The first case formalizes the obvious statement that a blackbox objective is a special type of finite mixture model. The second case also formalizes an obvious fact, that any finite mixture model can be solved as a blackbox optimization problem. This is useful because the

[3] This can be avoided by allowing the option of setting $G = 0$ to indicate that no weights are necessary.

cost of solving a particular problem using $\tilde{V}(\cdot)$ versus a generic optimizer $\bar{V}(\cdot)$ is simply the difference in the cost of solving $\tilde{V}$ and $B(\tilde{V})$, written as $C\left\{\tilde{V}\right\}$ and $C\left\{B(\tilde{V})\right\}$, respectively. The third case illustrates that additive objectives are very special cases of the broad finite mixture class. In a nested objective (case 4), certain parameters and one or more concerns are not under the direct to control of $\tilde{v}$. Imposing these concerns inside $f(\theta)$ uses calculations to impose them only as intermediate inputs to the final solution $\theta^{\star}$. (This view of iterative optimization is elaborated in section IV.)

II.C Implementing $\tilde{f}$

One way to code (2) is to write different functions for (μ, Ψ, Ω). In some respects it is more convenient to ask for an interface function, $\tilde{f}(c, b)$, that stands between these tasks and $\tilde{V}$.[4] The first argument is a vector describing the call being made,

$$c = (\tau \quad k \quad g \quad \theta \quad j), \tag{10}$$

which allows $\tilde{V}$ to control the execution of tasks and to split $f(\theta)$ into components that can be computed separately. The first element is the task to carry out, $\tau \in \{\mu, \Psi, \Omega\}$. The last element j is a tag which $\tilde{V}$ uses to keep track of asynchronous (parallel) tasks. The argument b is a buffer that handles both input and output. A two-way buffer allows $\tilde{V}$ to send all the input necessary to carry out the specify task and then to receive back the results. Thus, pseudo-code for $\tilde{f}$ is given in Example 1.

The tag j is not used by $\tilde{f}$, but it becomes an important ingredient to efficient parallel execution, because the processor assigned to calculate a given task may not have carried out the requisite sub-tasks. Separating the call type from the input buffer then frees $\tilde{V}$ from keeping track of what information already has been sent to the processor. The input to Ψ is of size KM, and the input to Ω is of size GQ. The buffer must be able to hold the largest message passed between $\tilde{V}$ and $\tilde{f}$: $|b| = \max\{KM, GQ\}$. Example 2 - Example 4 in the

[4] For example, initializations common to each task can be programmed once inside $\tilde{f}$ rather than separately in each sub-task.

Example 1. Pseudo-code for $\tilde{f}$

```
myprogram()
   local σ, θ0
   σ = ( G   K   N   M   Q )
   put starting values in θ0
   Ṽ(f̃, σ, θ0)                          call optimizer

f̃(c, b)                                 Definition of interface
  local allmu, allv
  if c.τ=μ {                             task is to solve model
     b = μ(c.g,γ[c.k]) }                 solve model, return results in buffer γ[c.k]=θ[GK+1+(k-1)N:GK+kN]
  if c.τ=Ψ {                             task is to evaluate
     allmu = b                           use model output in buffer on input
     b = Ψ(λ[c.g],allmu) }                 call evaluation procedure, send result back in buffer
  if c.τ=Ω {                             task is to aggregate
     allv = b                              use evaluate output in buffer on input
     b = Ω(allv) }                       call aggregation procedure to compute f(θ)
return
```

Appendix provide pseudo-code for modifying standard optimization code in order to exploit the interface $\tilde{f}$.

II.D Parallel Execution

Nagurney (1996) surveys applications of parallel processing in economics, and Doornik et al. (2002) discuss reasons and methods for making transparent parallel execution more common in computational economics. The term 'parallel processing' refers to several overlapping computer architectures and software environments, which is understandable because computations can be done in parallel at widely different levels of sophistication. At the highest level economics can be done in parallel by having two co-authors work on two projects separately. At a slightly lower level a single researcher can run two different programs on different computers. Some tasks are "embarrassingly" parallel and can also be solved simultaneously on separate processors. A Monte Carlo experiment can be conducted twice as fast simply by starting the same program on two different computers with different initial random seeds. Working down we reach perhaps the the lowest level of parallel computation,

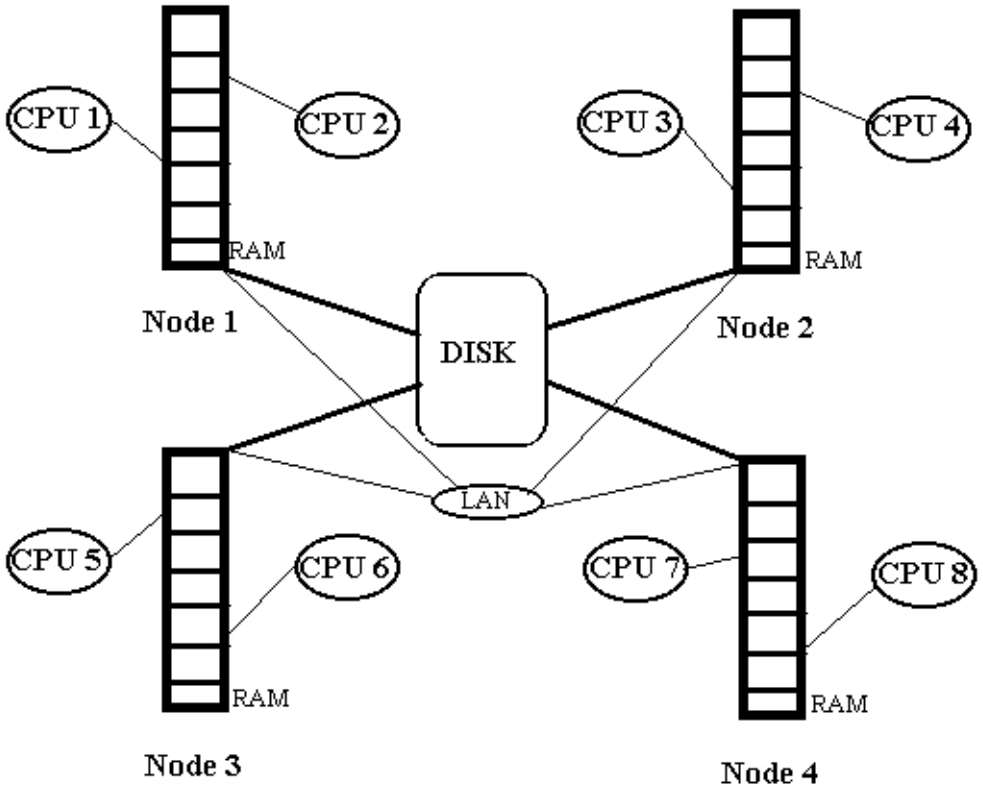
vector-processing. A vector processor can carry out a single operation simultaneously on multiple memory cells, thus parallelizing tasks such as matrix arithmetic.

These levels of parallel processing are described in decreasing order of sophistication or flexibility of the independent processors (co-authors down to vector hardware) and in increasing order of coordination (from separate tasks down to a single operation on specific memory cells). Modes of parallel processing differ in their efficiency when applied to different tasks. A vector processor cannot effectively compute two different rational expectations equilibria at once. By the same token two co-authors are not particularly effective at multiplying two matrices together.

In between vector processing and running multiple jobs on independent processors lies a range of parallel processing paradigms. A mixture of two currently common ones is illustrated by Figure 2. Eight processors share one long-term storage unit (disk). They are organized in pairs into four nodes. Processors in the same node share a memory unit (RAM). The nodes can communicate with each other through a network, typically at a slower speed than with RAM and, depending on the architecture, faster or slower than with disk.

Two processors in the same node can operate simultaneously on different segments of memory that are assigned to a single program. This is 'single instruction, multiple data stream' or SIMD architecture. For example, consider a loop that adds up a vector of numbers. A compiler aware of the SIMD architecture can cause the loop to execute up to twice as fast by assigning each processor to sum half of the vector and ensuring that one processor adds the two partial results to complete the operation. The next segment of the user's program may 'parallelize' as well, but it might just as well include a complex operation that the compiler cannot cause to execute on several processors with predictable results. For example, a loop over k that calls a user-written function $\mu(g, \gamma[k])$ will not parallelize using SIMD architecture. The compiler cannot be sure that separate function calls do not refer to and modify shared memory locations. If they do then the order in which separate processors operate helps determine the results, making them unreliable.

Figure 2. A Mixed Parallel Architecture



The form of parallel execution important for solving finite mixture models is running copies of a single program on multiple data (SPMD). Multiple copies may run on the same physical node in Figure 2 but their memory is segmented. A Monte Carlo exercise is an obvious example of a SPMD-ready problem. Iterative optimization has some elements that can benefit from simple SPMD execution by starting the algorithms at different initial vectors or with other options. However, the gains from independent optimizations is not particularly large, because nothing revealed about the objective in one copy can be shared with the others. To do better, the copies of the program must communicate with each other to coordinate their actions. Unlike execution in SIMD architecture, which can be transparent to the

programmer, parallel programming using SPMD requires calls to special library routines that send messages back and forth across processors.[5]

III. Examples of Finite Mixture Models

III.A Simple Examples

An example of a finite mixture model is the random-effects probit model estimated on panel data (McFadden and Train 2000). The model can be written

$$
\begin{aligned}
y_{it}^{\star} &= X_{it}\beta_i + \epsilon_{it} \\
y_{it} &= (y_{it}^{\star} > 0) \\
\epsilon_i &\equiv \begin{pmatrix} \epsilon_{i1} \\ \vdots \\ \epsilon_{iT} \end{pmatrix} \sim N(0, \Sigma(\sigma_i)) \\
Prob\left[\beta_i = \beta^k, \nu_i = \nu^k\right] &= \lambda[k]
\end{aligned}
\tag{11}
$$

for $i = 1, 2, \ldots, I$, $t = 1, 2, \ldots, T$, and $k = 1, 2, \ldots, K$. Typically only one set of mixture weights over the random effects is specified, so $G = 1$. Also, only the intercept term typically varies across types, but the notation allows the whole coefficient vector and the variance parameters ν^k to be type-specific. The model parameter vector takes the form $\gamma[k] = (\,\beta^k \quad \sigma^k\,)$. If there are N_1 coefficients and N_2 parameters in ν^k, then $N = N_1 + N_2$. The likelihood of an observation conditional on being type k is

$$
\mu[i,k] = \int_{R(y_{i1};X_{i1}\beta^k)} \cdots \int_{R(y_{iT};X_{iT}\beta^k)} f\left(\epsilon_i; \Sigma\left(\sigma^k\right)\right) d\epsilon_i, \tag{12}
$$

where

$$
R(y_{it}; X_{it}\beta^k + \nu^k) = \begin{cases} (-\infty, -(X_{it}\beta^k + \nu^k)] & \text{for } y_{it} = 0 \\ [-(X_{it}\beta^k + \nu^k), \infty) & \text{for } y_{it} = 1. \end{cases} \tag{13}
$$

[5] Swann (2001, 2002) concisely describes the principles of programming with message passing across processors and illustrates them using parallel optimization of an additively separable log-likelihood function, a special case of a finite mixture model. The straightforward parallel execution across observations he describes is precluded from working efficiently by the presence of unobserved heterogeneity or equilibrium restrictions.

Unless T is small or the variance matrix $\Sigma(\nu^k)$ is highly restricted, the integral in (12) is expensive to compute. As a finite mixture model the conditional likelihood for each observation is reported by μ. Thus, $M = I$. Once the K vectors have been computed, unobserved heterogeneity is integrated out:

$$f_{\text{panel}}(\theta) = \Omega = \sum_{i=1}^{I} \Psi[i] = \sum_{i=1}^{I} \ln \left(\sum_{k=1}^{K} \lambda[k]\mu[i,k] \right). \tag{14}$$

The vector of individual log-likelihoods is reported by Ψ, thus $Q = I$.[6] The parameter vector for a random-effects probit is:

$$\sigma_{\text{panel}} = (1 \quad K \quad N \quad I \quad I). \tag{15}$$

Thus, $\tilde{\theta} = \tilde{V}(f_{\text{panel}}(), \theta_0, \sigma_{\text{panel}})$. The gain from re-casting the model as a finite mixture comes through two channels. First, the mixture probabilities $\lambda[k]$ are treated differently than the parameters in $\gamma[k]$. None of the K costly integrals is re-computed when only the weights are varied. Second, when only one of the type-specific parameters is changed the other $K - 1$ costly integrals are not re-computed.

Not all numerical optimization in economics gains from casting the problem as a finite mixture model. For example, the log-likelihood function for an ordinary logit model can be written

$$f_{\text{logit}}(\theta) = \sum_{i=1} \ln \left(\exp\{y_i(-X_i\theta)\}/1 + \exp\{-X_i\theta\} \right), \tag{16}$$

where X_i is the vector of exogenous controls for observation i, y_i is the endogenous binary indicator, and θ is the vector of estimated coefficients, usually denoted $\hat{\beta}$. Computing $f_{\text{logit}}(\theta)$ requires no intermediate results, and results from past evaluations of the likelihood do not reduce the cost of computing $f_{\text{logit}}(\theta)$. In addition, the cost of computing the likelihood for each observation is so small that any gains in treating i as an index of heterogeneity would

[6] Setting $Q = I$ makes it straightforward for $\tilde{V}$ to implement the BHHH approximation to the Hessian matrix. A flag is set that indicates the evaluation task is an individual log-likelihood. When a gradient in θ is being computed these intermediate values can be stored and used to compute the outer-product of the gradient vector.

be swamped by additional communication costs. For these very reasons the cost of solving the logit model has been negligible for two decades.

A non-trivial example of a problem that is not amenable to a finite mixture representation is the fixed-effect linear regression, written:

$$Y_{it} = X_{it}\beta + \eta_i + \epsilon_{it}, \tag{17}$$

for $t = 1, \ldots, T_i$, $i = 1, \ldots, N$, and $\epsilon_i \sim N(0, \Sigma(\nu))$ the vector of disturbance terms for observation i. The variance matrix is determined by a vector of estimated parameters ν. The term η_i is an intercept term specific to observation i. Note that $G = K = 1$ and the parameter vector takes the form:

$$\gamma[1] = (\,\beta \quad \nu \quad \nu_1 \quad \nu_2 \quad \cdots \quad \nu_I\,). \tag{18}$$

The computational problem here is a long parameter vector and a sparse Hessian matrix, because η_i and η_j do not interact with each other (their cross-partial in the residual sum of squares are zero). To cast this as a finite mixture model the 'unobserved' type must be associated with the index i, because there are parameters specific to each of individual. As with the logit model there are no intermediate results worth storing, and the extra communication costs of sending the full parameter vector back and forth would swamp any gains in separating the problems.

III.B An Extended Example

To further fix ideas a finite mixture model current in the literature is mapped into the form (2). Lee (2001) estimates an equilibrium model of schooling, occupational choice, wages and cohort size using 25 years of data from the Current Population Survey along with other sources. The economy is a perfect foresight overlapping generations environment. Non-stationarity is caused by the exogenous pre-World War II baby-bust and the post-war baby-boom. At each age between 16 and 65 individuals choose to be in one of four distinct occupations indexed by m: at home, at school, at work in a blue collar job, and at work

in a white collar job. The payoff of each occupation depends on the person's current state, denoted S. The probability of possible states next year depends on the current state and the current occupation. The core of the model is therefore a dynamic programming problem:

$$\max_{m\in\{1,2,3,4\}} u(m,S) + \delta E[V(S';S,m)]. \tag{19}$$

Observed heterogeneity is associated with sex and birth cohort. Thus each index g would be associated with a pair $t(g), f(g)$, where $t(g) \in \{1865,\ldots,2065\}$ is the birth year of group g and $f(g) \in \{0,1\}$ is an indicator for female members of group g. The total number of groups is $G = 2 \times 200 = 400$. Unobserved heterogeneity enters (19) by assuming that within each demographic group there are two types. This might suggest $K = 2$, but Lee estimates some sex-specific parameters. Within the finite mixture format this requires $K = 4$. Types 1 and 2 would be specific to men and types 3 and 4 specific to women. The type proportions are assumed to differ across groups only between sexes and not cohorts. Based on this information we can determine that the weighting vector λ is of size 1600×1 vector satisfying a set of 1598 linear restrictions

$$\begin{aligned}\lambda[k,g] &= \lambda[k,f(g)]\\ \lambda[4,g] &= 1 - \sum_{k=1}^{3} \lambda[k,g]\end{aligned}$$

for all g.

Of the parameters specifying the problem facing an agent of type k,g, some would be traditionally defined as (exogenous) structural parameters and others as (endogenous) prices. We partition $\gamma[k]$ accordingly:

$$\gamma[k] = (\,\gamma_x[k] \quad \gamma_p[k]\,). \tag{20}$$

A total of 29 parameters belong to γ_x because they are at least partially under the control of the optimization package.[7] They include a discount factor, a rate of random retirement between ages 60 and 65, and a correlation between contemporaneous wage shocks across

[7] The stock of schooling at age 16 and the number of children by age differ across groups but are held fixed throughout the optimization process and can be excluded from $\gamma_x[k]$.

occupations. The two occupations each has a wage equation with seven parameters: coefficients on education, experience and its square along with the standard deviation of i.i.d occupation-specific wage shocks. Each occupation also has a non-pecuniary benefit, although the blue-collar benefit is normalized to zero. Agents face a stream of benefits from attending school defined by six parameters, including tuition costs, re-entry costs, and a standard deviation of the shock of the random component of the non-pecuniary component of school attendance. Three parameters define the stream of rewards from staying at home. Because Lee's original two types have been expanded to four, and there are restrictions that $\gamma_x[1] = \gamma_x[3]$, $\gamma_x[2] = \gamma_x[4]$, and $\gamma_x[1,s] = \gamma_x[2,s]$ for all parameters except 12 sex-specific parameters.

The sequence of endogenous occupation-specific skill prices that an agent faces are specific to the calendar year, y. In Lee's formulation of the overall objective, prices are not under the control of the optimization package. Instead, at each evaluation of the objective, prices are computed out of sight of $\bar{V}(\cdot)$. Agents make decisions over fifty periods (ages 16 to 65), so this leads to 100 prices effective to a given group. To ensure each agent's problem is completely defined by $\gamma[k]$ the full vector $\gamma_p[k]$ of 200 prices indexed by y would be included in $\gamma[k]$.[8] Agents in cohort t at age a see as the current the prices for year $y = t + a - 16$. Because equilibrium prices do not differ across unobserved type, the model imposes the set of 600 linear restrictions $\gamma_p[1] = \gamma_p[2] = \gamma_p[3] = \gamma_p[4]$.

One solution of the model generates a large vector of conditional choice probabilities that come from solving (19):

$$\mu[g,k] = [\text{Prob}(m|S)]\,. \tag{21}$$

Based on a mixture of these output vectors, parameters are chosen to address two concerns. First, the predictions are matched to data and prices are chosen to satisfy equilibrium restrictions. Let $Y[m,S]$ denote a 12×1 vector of moments implied by the current state and

[8] To avoid multiple copies of common parameters, θ could be defined to include λ, γ and a third vector that is shared by all problems.

occupational choice. For the first concern the state space is partitioned into subsets based on values of the last two elements of $Y[m,s]$, current years of schooling and presence of young children in the household:

$$\{C[d] : C[d] \subset S, Y[m,S][11:12] = d \forall\, S \in C[d]\}.$$

Then predicted conditional moments:

$$\Psi\hat{Y}[d,g] = \sum_{k=1}^{4} \lambda[k,g] \sum_{s \in C[d]} \sum_{m=1}^{4} P(m|S)P(s|C[d],k,g)Y[m,s]. \tag{22}$$

Of a possible 208,000 moments, 140,400 were matched because in some cases $Y[y,g]$ was further averaged over groups. For the second concern, each calendar year had four equilibrium conditions derived from a pre-defined aggregate Cobb-Douglas production function. Given current prices the quantity of each skill demanded equals the quantity supplied, and the price of each skill equals its marginal revenue at quantities demanded. Total skills in calendar year y equal the cohort-size weighted sum of the skill supplies within each demographic group. Let $ND(y)$ denote the vector of net skill demands in year y and $MR(y)$ denote the vector of skill marginal revenue. The equilibrium restriction can be expressed as a smooth function $\Delta :^4 \rightarrow$ written $\Delta\left(ND(y), \gamma_p[y] - MR(y)\right)$ such that $\Delta(\cdot,\cdot) \geq 0$ and $\Delta(x,z) = 0$ if and only if $x = z = (0,0)$.

In Lee's formulation prices are not under the control of the optimization problem but rather are computed internally within a blackbox objective. Using the notation defined here, his parameter vector and objective can be written:

$$\begin{aligned} \theta &= \begin{pmatrix} \lambda[1,f] \\ \lambda[1,m] \\ \gamma_x \end{pmatrix} \\ \tilde{\theta} &= \arg\min_{\theta} \left(Y[d,g] - \hat{Y}\left(d,g;\theta,\gamma_p^\star(\theta)\right)\right)' A \left(Y - \hat{Y}\left(d,g;\theta,\gamma_p^\star(\theta)\right)\right) \\ &\quad \text{subject to} \quad \gamma_p^\star(\theta) = \arg\min_{\gamma_p} \sum_y \Delta(ND(y;\theta,\gamma_p(\theta)), \gamma_p[y] - MR(y;\theta,\gamma_p(\theta))). \end{aligned} \tag{23}$$

The external consistency of the model is measured by the weighted distance between observed and predicted moments. The vector $\hat{Y}$ stacks all predicted moments and Y stacks all

observed moments from the data in a corresponding vector. The matrix A is a restricted weighting matrix described in Lee (2001). The internal consistency of the model is measured by the sum of distances from equilibrium across years. Using the definitions introduced earlier this constrained form of the objective is said to be *nested*, because the concern for internal consistency (equilibrium prices) is nested within the concern for external consistency. The optimization algorithm has no direct control over the price vector γ_p. By choosing the nested form the researcher specifies a lexicographic ordering of concerns. In addition, the parameter vector θ contains no separable subsets because the solutions of all agents' problems enter into $\sum_y \Delta(\cdot,\cdot)$. Thus, the objective (23) is only trivially a finite mixture model.

If finding equilibrium prices were inexpensive, then it is likely that use of a constrained optimization algorithms to solve (23) would be efficient. However, the set of equilibrium prices is both large and complex, making Lee's use of a specialized iterative procedure to solve for equilibrium (thereby nesting the concern) a practical solution. The tradeoff is that equilibrium is solved repeatedly for parameter values that are never used, so one might consider de-nesting the equilibrium concern to balance it with the concern for external consistency. For example, consider the parameter vector and model

$$\theta = \begin{pmatrix} \lambda \\ \gamma \end{pmatrix} = \begin{pmatrix} \lambda & \gamma_x[1] & \gamma_p[1] & \cdots & \gamma_x[K] & \gamma_p[K] \end{pmatrix}' \tag{24}$$

$$f(\theta) = (Y - \hat{Y})' A (Y - \hat{Y}) + \omega_i \sum_y \Delta(ND(y;, \gamma_p[y] - MR(y)). \tag{25}$$

The weight ω_i is chosen by the researcher. Now the objective can be broken down into 4×100 separate individual problems that each depend on a vector of parameters $\gamma[k]$. The result of these 400 problems are then 'mixed' using weights Λ to evaluate the model's predictions. These predictions are then assessed for external consistency and internal consistency at the final stage. The optimal parameter vector for an objective of the form (25) balances two of the concerns within Lee's analysis: internal and external consistency. Setting ω_i very large approximates the lexicographic ordering of concerns implied by final parameters of a nested objective.

IV. Iterative Optimization of Finite Mixture Models

IV.A Algorithms

Press et al. (1987), Judd (1998) and many other texts provide excellent descriptions of numerical optimization techniques. Here the details of an algorithm are important only that they imply a sequencing of function evaluations. Let the n^{th} iteration ($n \geq 0$) of an iterative optimization algorithm begin after $H(n)$ evaluations of the objective. Define (Θ_H, F_H) as the current history of an iterative optimization, where $\Theta_H = \{\theta_h\}_{h=0}^{H}$ is the sequence of parameter vectors already evaluated, and $F_H = \{f_h\}_{h=0}^{H}$ is the corresponding sequence of real numbers $f_h = f(\theta_h)$. At the start of iteration n, one of the vectors previously evaluated has been designated the 'current vector', $\theta^n \in \Theta_{H(n)}$. For our purposes, the essential elements of an iterative optimization algorithm are the current vector, the next vector $\theta_{H+1} = \mathcal{N}(\Theta_H, F_H)$, and a stopping rule which determines, $H^\star$, the number of evaluations before convergence to $\theta^\star$ is declared. The total cost of blackbox iterative optimization can be written

$$C\left\{B(\tilde{V})\right\} = C\left\{f(\theta_0)\right\} + \sum_{h=0}^{H^\star} C\left\{f\Big(\mathcal{N}\left(\Theta_H, F_H\right)\Big)\right\} = H^\star C\left\{f\right\}. \tag{26}$$

From (26) it becomes obvious that all of the features of iterative optimization typically emphasized—the size of θ, the choice of θ_0, the choice of algorithm $N(h_n)$, the properties of $f(\theta)$, and the desired precision in approximating $\theta^\star$—affect the total cost of solving (1) only through the scalar $H^\star$. They have no direct effect on $C\{f\}$, although they may lead the researcher to chose models with $C\{f\}$ small in order to make $C\left\{B(\tilde{V})\right\}$ acceptably low. When the objective is not a blackbox, then a more general cost of optimization obtains:

$$C\left\{\tilde{V}\right\} = \sum_{h=0}^{H^\star} C_h\left\{f\Big(\mathcal{N}(\Theta_H, F_H)\Big)\right\}, \tag{27}$$

where $C_0\{f\} = C\{f\}$ and $C_h\{f\} \leq C\{f\}$, because at some points in the history it might be possible to evaluate $f\Big(\mathcal{N}(\Theta_H, F_H)\Big)$ with fewer new computations than (27) requires. When $f(\theta)$ is a blackbox there is usually no way to tell $f(\theta)$ which intermediate results to keep.

There is also no way for $f(\theta)$ to send intermediate results back to $\bar{V}(\cdot)$ for keeping. Storing and recalling all past intermediate results is itself not feasible when literally thousands of function evaluations may be required to reach a desired level of precision.

Three standard optimization algorithms are now described with the purpose of illuminating the difference between $C\{V\}$ and $C\left\{\tilde{V}\right\}$.

Quasi-Newton with Numerical Gradients. The focus on numerical gradients is motivated by the observation that it is a rare event indeed when a heterogeneous agent economy involving dynamics, equilibrium, uncertainty or strategic choice yields analytical first or second derivatives for the overall objective. Newton's method with numerical gradients and Hessians would only amplify the gains in computation described for Quasi-Newton methods.

Q0. Begin Quasi-Newton iteration n with θ^n, an associated $P^U \times 1$ gradient vector $\nabla f(\theta^n)$, and a current approximation to the $P^U \times P^U$ inverse Hessian matrix $[\nabla^2 f^{-1}]^n$.

Q1. Carry out a line maximization in a direction indicated by the current information: $\theta_{h+m'} = \theta^n + t_{m'} D^n$, where $D^n = -[\nabla^2 f^{-1}]^n \nabla f(\theta^n)$, and $t_{m'}$ are scalars $(m' = 1, 2, \ldots, n_L)$ that depend on the line-maximization algorithm and the computed values $f(\theta_{h+m'})$.

Q2. One of the n_L new vectors becomes θ^{n+1}. Approximate $\nabla f(\theta^{k+1})$ numerically, perhaps with a two-sided (central) numerical gradient,

$$\nabla f(\theta) \approx \frac{\overrightarrow{f}(\theta + \epsilon) - \overrightarrow{f}(\theta - \epsilon)}{2\text{diag}(\epsilon)}. \tag{28}$$

The right side is shorthand for evaluating $f(\cdot)$ at $2P^U$ different vectors, sitting in two $P \times P^U$ matrices, $\theta + \epsilon$ and $\theta - \epsilon$. The matrix ϵ is diagonal so each column includes a small change in a single element of θ. The division in (28) is as an element-by-element operation on two vectors. Thus after the last line-maximization evaluation, the quasi-Newton algorithm will specify the next $2P^U$

evaluations to be added to the history:

$$\begin{aligned}
\theta_{h(n)+n_L+1}[1] &= \theta^{h(n)+1}[1] + \text{diag}(\epsilon[1]) \\
\theta_{h(n)+n_L+2}[1] &= \theta^{h(n)+1}[1] - \text{diag}(\epsilon[1]) \\
\theta_{h(n)+n_L+3}[2] &= \theta^{h(n)+1}[2] + \text{diag}(\epsilon[2]) \\
\theta_{h(n)+n_L+4}[2] &= \theta^{h(n)+1}[1] - \text{diag}(\epsilon[2]) \\
&\vdots \\
\theta_{h(n)+n_L+2P^U-1}[P^U] &= \theta^{h(n)+1}[P^U] + \text{diag}(\epsilon[P^U]) \\
\theta_{h(n)+n_L+2P^U}[P^U] &= \theta^{h(n)+1}[P^U] - \text{diag}(\epsilon[P^U]).
\end{aligned} \tag{29}$$

Q3. Update $\left[\nabla^2 f^{-1}\right]^{n+1}$ without further function evaluations (for example using the BFGS formula).

Line maximization in Q1 is not inherently parallel, because the sequence of values $t_{m'}$ is determined by realized values of the objective. On the other hand, the gradient computation in Q2 is parallel because the sequence of evaluations can be determined at once and distributed to independent processors. Thus, a Quasi-Newton algorithm contains some elements that scale with the number of processors available and some that do not. The ratio of time spent in these two parts of the algorithm is to a large extent under the control of the researcher. The researcher can choose to use, say, $2D$ finite deviations to approximate the gradient, where Q2 sets $D = 1$. Increasing D improves the direction of search. The researcher can also choose the precision in the line maximization to effect n_L. The fraction of evaluations per iteration that can be done in parallel is $2DP^U/(n_L + 2DP^U)$. Note that even with $D = 1$ the ratio increases automatically with the number of free parameters.

Nelder-Mead Simplex. The NM Simplex or Amoeba algorithm is a non-gradient method very commonly used in economics, sometimes as a robust and intelligent method to find starting values for a Quasi-Newton routine.

S0. Begin iteration n of the Nelder-Mead algorithm with a set of $P^U + 1$ distinct

evaluations $\mathcal{S}^n \subseteq \Theta^{P^U+1}$ that together form a simplex in $\Re^{P^U}$. The current vector is the best valued point: $\theta^n \equiv \max_{\theta \in \mathcal{S}^n} f(\theta)$. Without loss of generality, θ^n is element $P^U + 1$ of the S^n.

S1. Evaluate $f(\theta)$ at two trial vectors, $\theta_{h(n)+1}$ and $\theta_{h(n)+2}$, both functions of the current simplex. Either or both vectors might replace elements of $\mathcal{S}^n$ to form $\mathcal{S}^{n+1}$. If so, iteration n is complete.

S2. Otherwise, P^U further evaluations are required to collapse the simplex around θ^n:

$$\begin{aligned} \theta_{h(n)+2+1} &= \frac{1}{2}\left(\theta^n + \mathcal{S}^n[1]\right) \\ \theta_{h(n)+2+2} &= \frac{1}{2}\left(\theta^n + \mathcal{S}^n[2]\right) \\ &\vdots \\ \theta_{h(n)+2+P^U} &= \frac{1}{2}\left(\theta^n + \mathcal{S}^n[P^U]\right). \end{aligned} \tag{30}$$

Like the line maximization step of a Quasi-Newton iteration, the two trial evaluations in S1 are not inherently parallel, since which point to try the second time depends on the outcome of the first. Like the gradient calculation Q2, shrinking the simplex in S2 is a classic parallel operation, because the evaluations can be done in any order. In the case where the simplex is collapsed the ratio determining the scaling of an iteration across parallel processors is $P^U/(P^U + 2)$. Many iterations may not result in a collapse so the effective scaling of the whole algorithm is possibly much lower than this.

A key difference between S2 and Q2 when applied to a finite mixture model is that many if not all the parameter values in S2 differ from those in θ^n. It is therefore not possible to isolate the sub-tasks that need to be resolved. However, the finite mixture form of (2) allows the sub-tasks to be carried out in parallel. Thus parallel execution of the simplex collapse in $\tilde{V}(\cdot)$ is less 'granular' than in $\bar{V}(\cdot)$. And the initial simplex $\mathcal{S}^0$ can be constructed in parallel using steps in each of the free parameters starting from θ^0: $\theta_1 = \theta^0 + \epsilon[1]$, $\theta_2 = \theta^0 + \epsilon[2]$, etc.

Blackbox Simulated Annealing. Discontinuities and multiple local optima in $f(\theta)$ both severely limit the reliability of gradient and simplex algorithms. Metropolis's simulated

annealing algorithm attempts to overcome these problems by making random mistakes with decreasing probability as the algorithm plays out.

A0. Begin iteration n with θ^n and a current 'temperature' T^n. Draw a $P^U \times 1$ pseudo-random vector z and a pseudo-random variable ν.

A1. Evaluate $f^r = f(\theta_{h(n)+1})$, where $\theta_{h(n)+1} = \theta^n + z$.

A2. Update the current vector using the rule:

$$\theta^{n+1} = M(\theta^n, \theta_{h(n)+1}) \equiv \begin{cases} \theta_{h(n)+1} & \text{if } (f^n > f^r)\&(\nu < T^n) \\ \theta^n & \text{otherwise.} \end{cases} \tag{31}$$

Adjust T^{n+1} toward zero.

As with line maximization and simplex collapsing, evaluating the objective at the trial vector typically involves resolving all sub-tasks. In a finite mixture model this amount work can made much more productive.

Finite Mixture Simulated Annealing

Å0. Begin with θ^n, T^n, and z.

Å1. Let $z[i:j]$ denote the $P^U \times 1$ vector which contains 0 except in the contiguous elements i-j which equal the corresponding elements of z. Construct up to 2^{G+K} separate trial vectors that retain the current vector except within a subset set of structured components of the parameter vector:

$$\begin{aligned}
\theta_1^\dagger &= \theta^n + z[1:K] \\
\theta_2^\dagger &= \theta^n + z[(K+1):2K] \\
&\vdots \\
\theta_G^\dagger &= \theta^n + z[(G-1)K+1:GK] \\
\theta_{G+1}^\dagger &= \theta^n + z[GK+1:GK+N] \\
\theta_{G+2}^\dagger &= \theta^n + z[GK+N+1:GK+2N] \\
&\vdots
\end{aligned}$$

$$
\begin{aligned}
\theta^{\dagger}_{G+K} &= \theta^n + z[GK + N(K-1) + 1 : P^U] \\
\theta^{\dagger}_{G+K+1} &= \theta^n + z[1:K] + z[(K+1):2K] \\
&\vdots \\
\theta^{\dagger}_{G+K+G} &= \theta^n + z[1:K] + z[(G-1)K + 1 : GK] \\
&\vdots \\
\theta^{\dagger}_{2^{G+K}} &= \theta^n + z.
\end{aligned} \tag{32}
$$

The last such vector is the same as θ^r in blackbox simulated annealing. In others sub-vectors of θ^n are retained. Let $R \leq 2^{G+K}$ be the number of trial vectors chosen to consider and let j_r be a sequence of R indices into the possible combinations in (32).

Å2. Proceed under one of two options:

Option a: define θ^r as the best of these selected objectives:

$$
\theta^{n+1} = M\left(\theta^n, \max_{r=1,\ldots,R} \theta^{\dagger}_{j_r}\right). \tag{33}
$$

Option b: evaluate outcomes sequentially according to rule (31):

$$
\begin{aligned}
\theta^{\ddagger}_0 &= \theta^n \\
\theta^{\ddagger}_1 &= M\left(\theta^{\ddagger}_0, \theta^{\dagger}_1\right) \\
&\vdots \\
\theta^{\ddagger}_R &= M\left(\theta^{\ddagger}_{R-1}, \theta^{\dagger}_R\right) \\
\theta^{n+1} &= \theta^{\ddagger}_R
\end{aligned} \tag{34}
$$

The 2^{G+K} possible combinations use the same $2GK$ solutions of μ related to θ^n and $\theta^n + z$. To avoid recomputing these values requires storing more than one intermediate solution. When the evaluation task is cheap then all 2^{G+K} options can be evaluated at little marginal cost compared to the one trial vector θ^r. This re-use of solutions is possible only

because of the finite mixture form of the objective and the corresponding structure in the parameter vector.

To summarize, describing these leading optimization algorithms as sequences of related function evaluations demonstrates three points:

- Many vectors in an algorithm's history relate closely to the current vector. When combined with the partitioning of the parameter vector in a finite mixture model this dependence implies that many intermediate calculations are replicated exactly during the history of the algorithm.
- Storing intermediate results for the current vector, and possibly a small number of other results, can eliminate these replicated calculations. However, a blackbox objective cannot be told by the optimizer, and cannot infer on its own, where in the stream of function evaluations the algorithm is currently located. Storage of intermediate results for the current vector alone is not possible in blackbox optimization.
- A potentially large proportion of evaluations can be done in parallel because the subtasks of a finite mixture model are separate and the sequence of parameter vectors is at times completely independent of the results of current calculations. The parallel execution must be coordinated, for example as in the client/server model described in Section II and exhibited in the Appendix.

IV.B Precision and Avoided Calculations

Let δ denote the task of gaining a unit of precision in solving (2). The conventional notion of precision typically concerns the value $\|\theta^n - \theta^\star\| / \|\theta^{n-1} - \theta^\star\|$, which measures how much improvement is made per iteration of an algorithm. As is well known, the precision gained per iteration depends crucially on the match between the objective and the algorithm. For example, a quasi-Newton iteration applied to a (concave) quadratic function will essentially arrive at $\theta^\star$ in one iteration. However, the same algorithm applied to a discontinuous function has no guarantee of any improvement. Simulated annealing starting from the same point may

make significant gains after the same number of function evaluations. But when applied to a smooth concave function simulated annealing falls hopelessly behind in precision compared to quasi-Newton. The simplex algorithm falls in between in terms of robustness and efficiency.

The purpose here is not to consider the matching of algorithms to a given objective, so the usual formulas for rates of convergence are not particularly relevant. Rather, we wish to compare the performance of a given algorithm when applied to:

- a given objective represented as a blackbox or a finite mixture model.
- a given objective optimized under parallel or serial execution.
- a given finite mixture model represented by a balanced versus nested objective.

For these purposes it is reasonable to define precision not in terms of θ but with a certain amount of computation that would be expected to move the iterative algorithm towards $\theta^\star$, taking into account that a longer parameter vector will ultimately require more iterations and more function evaluations. Put another way, the measure of costs should take into account that $H^\star$ in (27) is affected by the size of the model.

Definition: Precision in $\theta^\star$. The unit cost of increased precision in computing $\theta^\star$ is defined as the cost of evaluating $f(\theta)$ and $\nabla f(\theta)$ using central differences as many times as there are free parameters, P^U:

$$C\{\delta\} \equiv C\{f\} + P^U C\{\nabla f\}. \tag{35}$$

This definition accounts for the fact that a model with more parameters is a more difficult optimization problem requiring more iterations to converge. For example, gaining the same precision in a model with double the number of parameters is assumed to take twice as many iterations and each iteration can take essentially twice as long. Thus a unit precision requires quadruple time. This serves as a benchmark for assessing the potential practical impact in computational time from using $\tilde{V}$ rather than $\bar{V}$ to solve a finite mixture model.

Implication 2: Blackbox Costs. Let $B\{\tilde{V}\}$ be the blackbox representation of an optimization problem. The costs of evaluating the objective, the gradient, and a unit of precision are:

$$\bar{C}\{f\} = G\left(KC\{\mu\} + KC\{\Psi_1\}\right) \tag{36}$$

$$\bar{C}\{\nabla f\} = 2P^U\bar{C}\{f\} = 2\left(G(K-1) + KN\right)\bar{C}\{f\} \tag{37}$$

$$\bar{C}\{\delta\} = P^U(1+2P^U)\bar{C}\{f\}. \tag{38}$$

This is a direct implication of the set-up of the problem, and as the notation suggests these are upper bounds on the costs of the task.

Implication 3: Finite Mixture Costs Let V be a finite mixture model, and suppose $f(\theta^n)$ has been computed and the GK output vectors $\mu[g,\gamma[k]]$ have been stored. Then

I3a.

$$\tilde{C}\{\nabla f\} = 2KG\left(NC\{\mu\} + (NK+K-1)\,C\{\Psi_1\}\right). \tag{39}$$

I3b. Algebra reduces the difference between (37) and (39) to

$$\begin{aligned} d^c(\nabla f) &= \bar{C}\{\nabla f\} - \tilde{C}\{\nabla f\} \\ &= 2KG\Big[\Big[K(G-1) + N(K-1)\Big]C\{\mu\} + (K-1)(G-1)C\{\Psi_1\}\Big], \end{aligned} \tag{40}$$

which is non-negative and a fourth-order expression in G, K, and N. It is quadratic in G and K and linear in N, all else constant.

I3c. For $K>1$ and $G>1$ more algebra leads to a relative efficiency gain of

$$\%d^c(\nabla f) = 100\times\frac{d^c(\nabla f)}{\tilde{C}\{\nabla f\}} = 100\times\frac{(K-1)(G-1)\left(1+\left(\frac{K}{K-1}+\frac{N}{G-1}\right)\rho\right)}{N(\rho+K)+K-1} \tag{41}$$

where $\rho = C\{\mu\}/C\{\Psi_1\}$ is the ratio of solution costs to evaluation costs.

Table 1 presents some examples of finite mixture models and their costs. Consider a model of modest size, $G=K=N=4$. For example, an individual's problem is characterized

Table 1. Problem Size and Differences in Computational Costs

σ				c{τ}				1 processor			8 processors		max. scale	
K	G	N	P^U	F	Ψ	ρ	f	d^c(Mf)	d^c(δ)	%d^c(Mf)	d^c(f)	d^c(Mf)	d(χ)	d^c(Mf)
1	1	1	1	3	1	3.00	4	0	0	0	0	0	0	0
2	1	10	21	3	1	1.50	8	120	2,520	42	3	7,920	-2	3
2	2	10	22	3	1	1.50	16	296	6,512	150	11	20,424	36	11
4	4	4	28	3	1	0.75	64	2,592	72,576	982	51	225,504	72	57
4	4	4	28	1	3	0.08	64	1,632	45,696	910	49	141,984	72	51
4	4	15	72	3	1	0.75	64	5,760	414,720	958	51	1,261,440	336	57
10	1	15	159	3	1	0.30	40	8,100	1,287,900	25	24	3,888,000	-18	27
10	10	15	240	3	1	0.30	400	151,200	36,288,000	8141	341	109,317,600	2520	387
10	10	15	240	1	3	0.03	400	93,600	22,464,000	8105	327	67,672,800	2520	369

Costs measured in elapsed time, not processor time.

by four parameters, there are four demographic groups (say, race × sex), and each demographic group is made up of a different mixture of four unobserved types. Then the difference in the cost of computing a numerical gradient is $768C\{\mu\} + 288C\{\Psi_1\}$. If the model takes 3 seconds to solve and 1 seconds to evaluate each case, this amounts to a 51 *minute* difference between $C\{\nabla f_{\tilde{V}}\}$ and $C\{\nabla f_v\}$. So $\tilde{V}$ would reach an additional unit of precision in solving (1) roughly 24 hours before $\bar{V}$ would. This reduction in computational time may result in solving exactly the same model in an afternoon rather than overnight. Or it may spur the development of a larger and better model that can be solved overnight. This difference in total time reflects a percentage difference in costs of 240: calculating ∇f takes nearly 3.5 times as long using $\bar{V}$.

Table 1 illustrates that the inefficiency in using generic optimization to solve a finite mixture model is explosive in terms of problem size. The compounding problem in models with unobserved heterogeneity problem is so big that very good reasons must be present to consider setting $K >> 1$ when using blackbox optimization. An alternative method to improve a model's performance is to solve a different model, requiring some combination of

increased N, $C\{\mu\}$, and $C\{\Psi\}$. It would seem typical that development of a bigger model is expensive (in terms of development costs) relative to adding more heterogeneity to an existing model. By relaxing the constraint on increasing K it is possible that much richer results can be generated without the cost of greatly complicating the underlying model.

IV.C Parallel Execution

Once a user has written the interface $\tilde{f}$ between $f(\theta)$ and of $\tilde{V}$, they are spared the onerous task of converting code that works in serial to work in parallel, because $\tilde{V}$ exploits the independent nature of the sub-tasks (see Figure 1). The performance of $\tilde{V}$ relative to $\bar{V}$ is greatly modified when running in parallel on several independent but coordinated processors. There is no way for $\bar{V}$ to distribute the sub-tasks across different processors, but $\tilde{V}$ can assign each of the GK model solutions $\mu(g, \gamma[k])$ to a separate processor or as many as are available. The G solutions to $\Psi(g)$ can also be conducted in parallel once the corresponding input vector is complete.[9]

Definition 3. Costs under Parallel Execution. Define:

- **a.** $C_{\|Z}\{\tau\}$ as elapsed time to perform task τ when executed on Z equally responsive and independent processors;
- **b.** $\chi\{\tau\}$ as the maximum number of processors that can be used in parallel for a given task: $\forall\ Z > \chi\{\tau\}, \quad C_{\|Z}\{\tau\} = C_{\|\chi\{\tau\}}\{\tau\}$;
- **c.** $ceil\{x\}$ as the integer ceiling of x (e.g. $ceil\{1.1\} = 2$).

Note that $C_{\|1}\{\tau\} = C\{\tau\}$ and that if τ is a standalone task not made up of sub-tasks then $\chi\{\tau\} = 1$.

Implication 4: Maximal Scaling of $f(\theta)$. Let communication time between processors be negligible relative to $C\{f(\theta)\}$.

[9] Schittkowski (2001) describes the optimizer NLPQLP which exploits distributed (parallel) execution of the sequential quadratic algorithm. As a generic optimizer NLPQLP cannot keep track of intermediate results required to reduce the granularity of a finite mixture model.

I4a. For all $Z > 0$, elapsed time to calculate $B(\tilde{f})$ is $C_{\|Z}\left\{B(\tilde{f})\right\} = \bar{C}\{f\}$ in (36). That is, $\chi\left\{B(\tilde{f})\right\} = 1$.

I4b. Elapsed time to calculate $\tilde{f}(\theta)$ is

$$C_{\|Z}\left\{\tilde{f}\right\} = ceil\{GK/Z\}C\{\mu\} + ceil\{G/Z\}KC\{\Psi_1\}. \tag{42}$$

That is, $\chi\left\{\tilde{f}\right\} = GK$ and in this case the cost of computing $\tilde{f}$ is simply $C\{\mu\} + KC\{\Psi_1\}$.

I4c. For $Z \geq GK$ the percentage difference in elapsed time for a single function evaluation is

$$\%d_{\|GK}(f) = 100 \times \frac{C_{\|Z}\left\{B(\tilde{f})\right\} - C_{\|Z}\left\{\tilde{f}\right\}}{C_{\|Z}\left\{B(\tilde{f})\right\}} = 100 \times \frac{KG - 1 + (G-1)K\rho}{1+\rho}. \tag{43}$$

The terms $\bar{C}\{f\}$ and $C_{\|GK}\left\{\tilde{f}\right\}$ are the best and worst costs of evaluating an objective in isolation (without any stored intermediate calculations). As discussed in Section III, some aspects of optimization are not subject to strong gains from parallel execution. Implication I4 states the limits to gains from parallel execution for these evaluations. Table 1 summarizes $d^c(f) = C\left\{B(\tilde{f})\right\} - C\left\{\tilde{f}\right\}$, the difference in elapse time to compute the objective starting from scratch, for various problems under parallel execution with 8 processors. Total *processing* time is the same for each algorithm as it was under serial execution. Elapsed time for $B(\tilde{f})$ is also the same, because a blackbox presents no sub-tasks that can be done in parallel. When $f(\theta)$ is to be evaluated at related parameter vectors the marginal cost of another evaluation can be even lower than $C_{\|GK}\left\{\tilde{f}\right\}$. The next implication covers the inherently parallel phases of an optimization algorithms.

Implication 5: Maximal Scaling of $\nabla f(\theta)$. Define $\tilde{f}$ and $B(\tilde{f})$ as in the previous implication.

I5a. $\chi\left\{\nabla B(\tilde{f}))\right\} = 2P^U$ and $C_{\|2P^U}\left\{\nabla B(\tilde{f})\right\} = \bar{C}\{f\}$.

I5b. $\chi\left\{\nabla\tilde{f}\right\} = 2NKG$ and $C_{\|2NKG}\left\{\nabla\tilde{f}\right\} = C\{\mu\} + KC\{\Psi_1\}$.

Calculating the gradient of a finite mixture objective can scale up to $2NKG$ parallel processors and become as cheap to compute as the model for one type. Table 1 also reports $d^c(\nabla f)$. Calculating a blackbox gradient does benefit from more processors because of the $2P^U$ function evaluations can be done simultaneously. As under serial execution redundant calculations are made, and the fall in elapsed time comes with an inefficiency borne by other users of the same processors. When N is relatively large calculation of $\nabla B(\tilde{f})$ can complete faster than $\nabla \tilde{f}$, but it would still be slowing down the system as a whole as under serial execution.

Table 1 also reports $d(\chi) = \chi\left\{\nabla \tilde{f}\right\} - \chi\left\{\nabla B(\tilde{f})\right\}$, the difference in the number of processors that can be kept busy all at once while calculating the gradient in a finite mixture and blackbox objective. There are cases when $d(\chi) < 0$, which are situations in which N is relatively large. While $\nabla B(\tilde{f})$ can use more processors simultaneously in these cases, it is still doing so inefficiently and will not complete the gradient any faster in elapsed time than $\nabla \tilde{f}$ running on fewer processors. The elapsed time difference under maximum scaling is not explosive like it is under a fixed number of processors because implicitly computing capacity is added to keep up with the scale of the problem. Yet a finite mixture gradient is always faster to compute.

V. Balanced versus Nested Objectives

The objective $f(\theta)$ in a finite mixture model is based on three tasks (μ, Ω, Ψ): model solution, evaluation, and aggregation. These tasks are treated as black-boxes that are connected to each other as in Figure 1. Since the black-box is a special case there is no loss of generality. Is there a gain to exploiting this special structure? There are three aspects to that question.

First, will an optimizer $\tilde{V}(f, \theta_0, \sigma)$ based on this structure out-perform the black-box optimizer $\bar{V}(f, \theta_0)$? The previous section provided an affirmative answer. As long as storage

and communication costs are small relative to computational costs then $\tilde{V}$ will reduce the time it takes to optimize $f(\theta)$. In certain problems the reduction can easily make a model feasible to solve using $\tilde{V}$ when it would not be feasible using $\bar{V}$.

Second, can a broad spectrum of problems encountered in economics be broken down into these tasks without resorting to the trivial case of $G = K = 1$? As Table 1 indicates, problems that are mixture models in a trivial sense will not benefit from $\tilde{V}$. The example in section III attempted to show by analogy that models that allow for heterogeneity and impose some combination of individual rationality, equilibrium (internal consistency), and consistent estimation (external consistency) are non-trivial examples of this framework. Much larger models (greater amounts of observed and unobserved heterogeneity) may become feasible to solve when using $\tilde{V}$ rather than $\bar{V}$.

The third aspect of the question is whether reformulating the objective function to exploit the finite mixture form of a problem leads to any further gains. The particular change considered here is choosing between the nested and balanced representations of a given finite mixture model defined earlier in D2.

By definition unnecessary calculations can be avoided by solving nested problems simultaneously (in balance) so that only at the ultimate answer are both concerns satisfied. However, more balance requires a longer parameter vector. For example, suppose one of the concerns in the model is that agents use optimal decision rules. In a nested representation the objective might call a procedure that computes optimal decision rules each time the optimizer evaluates the objective. In a balanced representation the optimizer would be asked to choose decision rules for agents along with other parameters. The objective must assess how far the rules sent in θ are from being optimal, which is presumably cheaper than actually finding the optimal rules. Thus the tradeoff between the nested and balanced representation of a finite mixture model is one of more parameters to optimize in return for lower costs of solving the model during iterative optimization. Although all concerns that enter f would enter both representations, their relative importance will likely differ. The results of

optimizing f_n and f_b are therefore not necessarily equivalent.

The balanced representation of an objective relates to the approach of penalizing an objective when side constraints are not met. The nested representation relates to imposing side constraints directly, such as choosing optimal decision rules for agents at each evaluation of a likelihood function. The balanced or penalized-objective approach would augment $f(\theta)$ with a penalty for non-optimal decision rules. The difference here is that explicit allowance for the finite mixture form of the objective gives the modeler more control over these tradeoffs. The issue is when will the cost of optimizing over more parameters outweigh the benefit of not imposing nested concerns at each evaluation of the objective.

Implication 6: Balanced versus Nested Objectives. Let f_b and f_n be balanced and nested representations of a given finite mixture model and for $x = b, n$ define $D_x N_x C\{\mu_x\} + (N_x K + K - 1)C\{\Psi_x\}$. For $\tau = \nabla, \delta$, define $d^c(\tau_n) \equiv C\{\tau(f_n)\} - C\{\tau(f_b)\}$. Then:

I6a. $d^c(\delta_n) > 0 \quad \rightarrow \quad d^c(\nabla_n) > 0 \quad \leftrightarrow \quad D_n > D_b.$

I6b. $N_n C\{\mu_n\} - C\{\Psi_n\} > N_b C\{\mu_b\} - C\{\Psi_b\} \quad \rightarrow \quad d^c(\nabla_n) > 0$ for all K and all G.

I6c. $D_n > D_b$ & $G(K-1)/K > (A_b N_b - A_n N_n)/(A_b - A_n) \quad \rightarrow \quad d^c(\delta_n) > 0.$

All of these results are based on algebraic manipulations of previously defined costs of computing tasks. The expression D_x determines whether a gradient calculation is faster using f_b or f_n, taking into account changes in solution costs, numbers of parameters, and redundant calculations. Because the balanced approach adds model parameters to the optimization problem a necessary condition for a unit of precision to be cheaper is that the balanced gradient is cheaper. I6b gives a sufficient condition for the balanced gradient to be cheaper to compute for any value of K and G. Finally, a cheaper gradient and large values of G and K together imply that δ is cheaper in the balanced representation. The values of G and K have to be large enough to make up for the increase in dimensionality as measured in the expression $(A_b N_b - A_n N_n)/(A_b - A_n)$, which may be positive or negative. These results allow a researcher to attempt various approaches to balance their objective and check for potential

gains or losses in computational efficiency without having to carry out a full evaluation of the objective and its gradient (let alone to solve the finite mixture model by optimizing both problems).

Table 2 illustrates Implication 6. First, consider a model without heterogeneity and with no difference in costs between the balanced and nested representations. However, the balanced version has $N_b = 2$ model parameters versus $N_n = 1$ in the nested version. The result is that a unit of precision costs $d^c(\delta_n) = 12$ seconds more in the balanced version. We get to zero difference in a case where the nested version has a model that costs five times more to compute and a evaluation that costs three times as much. The next segment of the table considers a balanced representation that is very efficient. It cuts model and evaluation costs to one-tenth and one-fifth their values in the nested version, respectively. It takes 11 model parameters versus 4 to achieve these savings. Without heterogeneity the nested solution is slightly cheaper to solve. Adding an unobserved type ($K = 2$) makes the difference larger, but then adding an observed type ($G = 2$) results in the cost of precision under the balanced objective being 272 seconds lower.

Finally, in the bottom of Table 2 the model and evaluation costs are the same as the middle rows, but now there are four observed and unobserved types. With $N_n = 11$ a unit of precision is 17,000 seconds cheaper than the balanced objective. However, adding a single model parameter brings the difference to exactly zero. Thus, differences between nested and balanced objectives are very sensitive to difference in all dimensions, particularly when heterogeneity is a major element of the analysis. Rules of thumb based on previous experience may be very misleading. By the same token, the comparisons in Implication I6 are themselves only rough guides to the choice of how to represent a given model. The realized cost of optimizing f_n compared to f_b depends on the properties of the sub-tasks, the quality of starting values, and so forth. Some models may lend themselves to a balanced objectives, others not. For example, if the underlying model can use specialize solution methods with fast convergence it might prove inefficient to solve the model using ordinary

Table 2. Balanced versus Nested Objectives

Shared		Balanced					Nested					
K	G	N	P^u	c{F}	c{Ψ}	D	N	P^u	c{F}	c{Ψ}	D	$d^c\delta_n$
1	1	2	2	1	1	4	1	1	1	1	2	-12
1	1	2	2	1	1	4	1	1	5	3	8	0
1	1	11	11	1	1	22	4	4	10	5	60	-4
2	1	11	23	1	1	34	4	9	10	5	85	-68
2	2	11	24	1	1	34	4	10	10	5	85	272
4	4	11	56	1	1	58	4	28	10	5	135	17,024
4	4	12	60	1	1	63	4	28	10	5	135	0

Costs measured in elapsed time.

non-linear optimization methods. Even in that case the balanced optimizer can assess partial solutions to the model on other grounds (such as distance from equilibrium) and replace many cases of solving quickly the model at bad parameters values with fewer cases of solving slowly the model with good parameter values.

V.A Interpreting Results

Depending on the type of finite mixture model being solved there will exist external standards with which to assess the ultimate parameter vector $\theta^\star$. For example, sufficient conditions for maximum likelihood estimates of a model's parameters to be statistically consistent, efficient, and asymptotically normal are well known. A likelihood as in Rust (1994, 1996) that nests other concerns inside it can appeal to such results. When choices (and in equilibrium models, prices) are optimized simultaneously with underlying parameters, the standards for assessing results are not so clear. What can be said is that the final parameter

vector balanced the concerns specified by the researcher, and that a balanced objective can approximate a nested objective with appropriate weights on each concern. At the minimum, a modeler may find a balanced objective to be a good way to get estimates of all free variables before they re-nest the model to impose exactly some concerns at each objective evaluation.

One might hope for a sounder basis for making changes in overall solution strategies. As an analogy for such a standard, consider Debreu's (1983) vanguard proof that the Walrasian competitive equilibrium is a Nash equilibrium in a game played among agents and a fictitious *auctioneer*. The auctioneer's objective is to minimize the value of the excess demand function by choosing prices taking as given the quantities demanded by the agents, who in turn take as given the prices called out by the auctioneer. The first and third terms in objective (25) reflect exactly these strategies. Under usual regularity conditions there would exist a price that simultaneously sets all marginal utilities to zero and equates aggregate demand to aggregate supply. In Debreu's notion of a social equilibrium the primitive preferences and endowments of agents are taken as given by all players.

When preference and technology parameters vary along with choices and prices, the notion of a social equilibrium does not apply. No player is concerned with a choice of these parameters because no player is concerned with external consistency of the equilibrium. Consider introducing a second fictitious player, *the econometrician*, with the objective of maximizing external consistency. The strategies for each type of player are:

- the econometrician calls out preferences of K agent types, each denoted $\gamma_p[k]$, and population weights $\lambda[k, g]$, taking the choices of other players as given;
- the auctioneer calls out equilibrium outcomes denoted $\gamma_e[g]$, taking the choices of other players as given;
- K agent types call out conditional decision rules $\gamma_d[k, g]$ taking as given the choices of other players.

The payoff to agents is their utility or profit given their preferences, the equilibrium outcomes, and decision rules. The payoff to the auctioneer is the negative distance from

equilibrium given decision rules and population weights. The payoff to the econometrician is the distance between externally generated data (moments) and the model's predicted values given the decision rules, equilibrium outcomes, and population proportions.

Obviously a Nash equilibrium to this game is one in which agents choices are optimal given preferences and prices, equilibrium is achieved, and exogenous data are best explained by the estimated preferences, choices and prices. A social planner's problem which puts some weight on each player's objective is equivalent to a balanced objective. The weights are chosen by the researcher in order to balance the three competing concerns. The typical nested formulation is akin to a Stackelberg game with the econometrician moving first followed by the auctioneer and the agents. This solution appears to be easier to interpret, but in some models the computational advantage of a balanced objective may be large enough to displace the more manageable nested objective as the default approach to specifying the finite mixture model. It will then be important to assess results from a balanced objective on their own terms, perhaps using a game-theoretic framework as suggested here.

VI. Conclusion

If Laocoon were alive today he might warn us to "beware of *geeks* bearing gifts." The road to current methods of numerical optimization in economics is littered with hardware (vector processors, connection machines, etc.) and software (neural networks, genetic algorithms) that promised general breakthroughs in computational practice. These breakthroughs were just over a horizon requiring only some investments in expensive hardware, dedicated software, or new modelling approaches. Many of these gifts turned out to be Trojan horses. Costs were paid in advance for future benefits. Meanwhile general technological advance allowed researchers who had kept using generic resources to overtake and then surpass those who had invested in specific investments which did not catch on. Thus, a call to re-formulate optimization problems in economics should be made and received with a great deal of skep-

ticism. There are five reasons why a switch from $\bar{V}$ to $\tilde{V}$ may not turn out to be a Trojan horse.

First, $\tilde{V}$ includes as a special case the long-standing preferred method of coding objective functions. Thus, a function f implemented through the interface $\tilde{f}$ can still be optimized using a generic optimizer.

Second, the structure of finite mixture models is exploited by $\tilde{V}$ without any attempt to tweak the performance of the underlying optimization algorithm. Tried-and-true algorithms still form the basis of the optimization.

Third, the only special assumption about the computing environment for the advantages of $\tilde{V}$ to outweigh its overhead is that communication and storage costs are small relative to processing costs. To the extent that the underlying tasks in finite mixture models remain non-linear and reliant on iterative algorithms, this assumption appears innocuous. The present and the future of intensive computing include parallel processing. Generically, optimization of well-behaved functions does not lend itself to parallel processing. However, using $\tilde{V}$ exploits all the returns to parallel execution inherent in the class of finite mixture models.

Fourth, a focus on finite mixture models is justified because their elements relate to long-standing touchstones of economics: concerns for individual rationality, internal consistency, and external consistency. It is not argued that economics should in the future compute different problems, and that $\tilde{V}$ makes it feasible to compute those problems. Rather, it is argued that most current problems computed in economics are already non-trivial forms of the finite mixture model. The seemingly diverse problems currently solved numerically have more in common than generally recognized, because their common features are less important when using generic optimization.

Finally, using a nested (or sequential) algorithm to optimize a finite mixture model has the potential to waste a great number of calculations by imposing certain concerns exactly on each candidate parameter vector, although only one such parameter vector will ultimately be used. Use of $\tilde{V}$ rather than $\bar{V}$ allows the researcher to control the balance among all

concerns. The iterative algorithm can speed to a final result by avoiding exact solutions to some concerns during iteration. Conditions have been provided that a researcher can check beforehand to have some confidence whether switching to a balanced objective will in fact decrease the cost of solving the overall objective.

VII. Appendix

VII.A Serial Execution

Steps to convert $\bar{V}$ into $\tilde{V}$

S1. Create space for holding values for one previous evaluation of θ, denoted θ^h, μ^h, Ψ^h, and Ω^h.

S2. Replace direct calls to $f(\theta)$ with a function $ff(\theta, r, h)$ that is internal to $\tilde{V}$. The last two arguments are boolean values. When h is true the optimization algorithm is requesting that $ff(\cdot)$ hold results for this evaluation. When r is true the optimization algorithm is modifying elements of γ. Therefore $\mu(\cdot)$ may need to be resolved for some types. The evaluation of $f(\cdot)$ at the beginning of an iteration would typically have $r = h = 1$. All steps taken in a numerical gradient routine would have $h = 0$ and would have $r = 1$ if the index of the incremented parameter is greater than KG.

S3. Write code for $ff(\cdot)$ as in Example 2, which is the function that calls the user's $\tilde{f}$ function. Whether a particular model is resolved depends on whether $\gamma[k] = \gamma_k^h$. If so, then the held values of Ψ_{kg}^h are still valid for all g and they can be sent to $\tilde{f}$ when evaluating the model. When $r = 0$ only elements of λ are being modified so no calls to μ need be made.

VII.B Parallel Execution

The pseudo code for ff given in Example 2 calls tasks sequentially. It will not work efficiently under parallel (distributed) execution. To do this a list of tasks must be set up and performed all at once. To keep track of asynchronous results, each sub-task is assigned an integer tag j. The total number of separate tasks in one evaluation is $T^u = GK + G + 1$. So, for example, tag 1 is associated with task Ω, tags 2 through $G + 1$ are associated with the task Ψ and values of g between 1 and G. The tags between $T^l = G + 2$ and T^u are

Example 2. Pseudo serial code calling $\tilde{f}$ to evaluate $f(\theta)$

```
ff(θ,r,h)
 local k, g, c, μ^f, Ψ^f, b
 for k = 1,...,K
     if (r)+(γ[k]≠γ^h[k]) {                  γ[k] is the sub-vector θ[GK+1+(k-1)N:GK+kN]
         for g = 1,...,G {
             c = (μ,k,g,θ)                   set the call information
             f̃(c,b)
             μ^f[k,g] = b[1:M]               store model results locally
             if h {                              and hold if asked to do so
                μ^h[k,g]=μ^f[k,g] } }
     else {
         for g = 1,...,G {
             μ^f[k,g] = μ^h[k,g] } }
 for g = 1,...,G {
     b = μ^f[g]                              pass model results in b
     c = (Ψ,0,g,θ)
     f̃(c,b)
     Ψ^f[g] = b[Q]                           store and hold
     if h {
        Ψ^h[k,g]=Ψ^f[k,g]}}
 b = Ψ^f                                     pass evaluation output in b
 c = (Ω,0,0,θ)
 f̃(c,b)                                     aggregate to compute objective
 ff = b[1]
 end
```

associated with μ and all the combinations of k and g. A look-up table can be constructed that associates each tag j with a task and values of k and g.

Using these tags $\tilde{V}$ can be coded so that parallel execution is transparent to the user. Example 3 illustrates this. The parallel environment is set up and maintained within $\tilde{V}$, which is organized as a client-server system. That is, one processor takes on the role of the client and carries out the optimization routine. All other processors take on the role of server. They wait for instructions sent by the client. Two special tags must be defined. The tag j = `STOP` = -1 is sent by the client to servers when optimization is complete and they should exit. Tag j = `NEW` = 0 is sent to the client along with a new vector in the message buffer to be used for all tasks until a new parameter vector is sent.

Example 2 provides pseudo-code for using such a coding of tasks to compute objective in

parallel. The code refers to two routines, `receive` and `send`, that carry out communication between processors. These routines are pseudo versions of `MPI_Recv` and `MPI_Send`. The routine `initialize` is a pseudo version of `MPI_Initialize`. See Swann (2002) for more information.

To compute a gradient or set up a simplex in parallel requires setting up a list of evaluations to perform in parallel. Represent the list as a vector L, where $L[d]$ is one of D parameter vectors to evaluate in parallel. For example, in a central difference numerical gradient $D = 2P^U$. Positive and negative changes to parameter n might be given indices $d = 2(n-1)$ and $d = 2(n-1)+1$, respectively. Let `manyf(L,h,r)` be a routine that processes a vector of evaluations in parallel. Tags sent to and from processors will have the form $t = (d-1)T^u + j$. Once the whole list of parameter vectors have been processed the results are stored in a vector F where $F[d] = f(L[d])$. For example, the gradient routine inside a parallel finite mixture optimizer would call `F=manyf(L,r,h)` and then construct the gradient in a simple loop: $\nabla f[n] = (F[2(n-1)] - F[2(n-1)+1])/(2\epsilon[n])$.

Example 3. Pseudo-code for parallel client-server optimization code

```
global Z, myid, tagtable
Ṽ(f̃,σ,θ0)
  initialize(σ,myid,Z)
  if myid=0 {                                  processor 0 is the client
     client() }
  else {
     server() }                                  the others are servers
return

server()
  local b, θ, c
  j=0
  while j≠ STOP {
     receive(b,0,j)
     if j=NEW {                  client is updating the parameter vector
           θ=b }
     else {
        if j≠ STOP {
           c = (tagtable[j],θ)          look up the task for tag j
           f̃(c,b)
           send(b,0,j)}}}                    send result back to client
return

client()
  carry out optimization algorithm selected by user
  when done send STOP tag to all servers
return
```

Example 4. Pseudo-code parallel for evaluating $f(\theta)$

```
vf(θ,r,h)
 local j, d[T], gdone[G], ngdone, ntorecv, ntosend, icall, b, busy
 b = θ
 for z = 1, ..., Z {
      send(θ,z,NEW)}            update parameter vector
 ntorecv = GK+G+1
 ntosend = GK+G+1
 icall = max(T^l,T^u-Z-1)              get icall nodes busy right away
 for j = T^u, T^u-1, ..., icall {
      send(b,z,j)
      --ntosend}
 while ntorecv > 0 {
      receive(b,ANY_SOURCE,j,z)             wait for any node to respond
      --ntorecv
      d[j]=1
      if j≥ T^l {                      one more μ[k,g] done
         ++gdone[c[j].g]}
      process task j, buffer b              see code for ff in Example 2
      busy = 0
      ngdone = 0
      if ntosend>0 then {                   more tasks to send
         j = T^l
         while (busy=0) & (jj≥ icall) {
             if d[j]=0 {
                send(b,z,,j)                send task j to node z
                --ntosend
                busy=1 }
             --j}
         if busy=0 {                        all model tasks sent
            for j = T^l-1,...,2 {
                if (not d[j])*(gdone[c[j].g]=K) {
                   send(b,z,j)
                   --ntosend
                   busy=1}
                else {
                   ++ngdone}}
         if (busy=0) * (ngdone=G) {         ready to aggregate
            send(b,z,1)
            --ntosend}}
return
```

VIII. References

Arcidiacono, Peter and John B. Jones 2002. "Finite Mixture Distributions, Sequential Likelihood and the EM Algorithm," *Econometrica* (forthcoming), `http://-www.econ.duke.edu/ psarcidi/em10.pdf.`

Aguirregabiria, Victor and Pedro Mira (2002). "Swapping the Nested Fixed Point Algorithm: A Class of Estimators for Discrete Markov Decision Models," *Econometrica* 70, 4, July, 1519-1543.

Debreu, Gerard 1983. "A Social Equilibrium Existence Theorem," in *Mathematical Economics: Twenty Papers of Gerard Debreu*, Econometric Society Monographs 4, Cambridge University Press.

Doornik, Jurgen A., David F. Hendry, and Neil Shephard 2002. "Computationally-intensive Econometrics using a Distributed Matrix-programming Language," *Philosophical Transactions of the Royal Society of London, Series A*, 360, 1245-1266.

Eckstein, Zvi and K. I. Wolpin (1990). "Estimating a Market Equilibrium Search Model from Panel Data on Individuals," *Econometrica 58*, 4, pp 783-808.

Goldfeld, Stephen M., Richard E. Quandt, and Hale F. Trotter 1966. "Maximization by Quadratic Hill-Climbing," *Econometrica* 34, 3, 541-551.

Heckman, James J. and Burton Singer. 1984. "A Method for Minimizing the Impact of Distributional Assumptions in Econometric Models for Duration Data," *Econometrica* 52

Hotz, Joseph and Robert A. Miller (1993). "Conditional Choice Probabilities and the estimation of dynamic models," *Review of Economic Studies* 60, 497-529.

Imai, Susumu, Neelam Jain and Andrew Ching 2002. "Bayesian Estimation of Dynamic Discrete Choice Models," manuscript, Ohio State, April, `http://www`.econ.ohio-state.edu/aching/research/bayes2c.pdf

Judd, Kenneth L. 1998. *Numerical Methods In Economics*, Cambridge, Mit Press.

Mcfadden, Daniel and Kenneth Train 2000. "Mixed MNL Models for Discrete Response," *Journal of Applied Econometrics* 15, 447-470.

Nagurney, Anna 1996. "Parallel computation," in *Handbook of Computational Economics*, D. Kendrick, J. Rust, H.M. Amman, (eds.), North-Holland, Amsterdam, 331-400.

Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling 1987. *Numerical Recipes: The Art of Scientific Computing*, New York, Cambridge.

Rios-Rull, Victor 1999. "Computation of Equilibria in Heterogenous Agent Models," in *Computational Methods for the Study of Dynamic Economies: An Introduction*, Ramon Marimon and Andrew Scott (eds.): Oxford.

Rust, John 1994. "Structural Estimation of Markov Decision Processes," in *Handbook of Econometrics*, Volume 4, R. Engle and D. McFadden (eds.), 30823139, North Holland.

—— 1996. "Numerical Dynamic Programming in Economics," in *Handbook of Computational Economics*, H. Amman, D. Kendrick and J. Rust (eds.), Elsevier, North Holland.

Schittkowski, Klaus 2001. "NLPQLP: A New Fortran Implementation of a Sequential Quadratic Programming Algorithm for Parallel Computing," manuscript, `http://www.uni-bayreuth.de/departments/math/ kschittkowski/nlpqlp_rep.htm`

Swann, Christopher A. 2001. "Software for parallel computing: the LAM implementation of MPI, *Journal of Applied Econometrics* 16, 2, 185-194.

—— 2002. "Maximum Likelihood Estimation Using Parallel Computing: An Introduction to MPI," *Computational Economics 19*, 2, April, 145-178.