

The Life-Cycle Income Analysis Model (LIAM): A Study of a Flexible Dynamic Microsimulation Modelling Computing Framework

Cathal O'Donoghue, John Lennon and Stephen Hynes

Rural Economy Research Centre (RERC), Teagasc, Ireland; email: Cathal.ODonoghue@teagasc.ie

ABSTRACT: This paper describes a flexible computing framework designed to create a dynamic microsimulation model, the Life-cycle Income Analysis Model (LIAM). The principle computing characteristics include the degree of modularisation, parameterisation, generalisation and robustness. The paper describes the decisions taken with regard to type of dynamic model used. The LIAM framework has been used to create a number of different microsimulation models, including an Irish dynamic cohort model, a spatial dynamic microsimulation model for Ireland, an indirect tax and consumption model for EU15 as part of EUROMOD and a prototype EU dynamic population microsimulation model for 5 EU countries. Particular consideration is given to issues of parameterisation, alignment and computational efficiency.

Keywords: flexible; modular; dynamic; alignment; parameterisation; computational efficiency

1. INTRODUCTION

Population-based dynamic microsimulation models are programs that are used to forecast populations into the future and to assess the impact of economic and demographic change on public policy. In particular these models have been used to analyse existing policy and to design policy reforms in inter-temporal policies such as education, pensions, long-term care and spatial policy.

The objective of this dynamic modelling framework is to incorporate a time dimension into policy analysis. Using models based on cross-section data simply allows one to look at the effect of policy at one point in time. Using cross-sectional data one is limited in the simulation of policy instruments which depend on inter-temporal factors such as pensions. A dynamic microsimulation life cycle model allows one to examine policy over time; for example life course redistribution, forecasts of cross-sectional redistribution and the simulation of pensions.

Designing dynamic microsimulation models is a large model building project involving many disciplines such as economics, social policy, statistics and computer science. This paper describes an innovative mechanism for making dynamic microsimulation models easier to construct, using a generalised method. The result is the Life-cycle Income Analysis Model (LIAM). In addition we outline a number of current applications of LIAM to further illustrate the flexibility of the approach adopted.

The paper is designed as follows. Section 2 describes the objectives of the paper, with section 3 describing the main model features. Section 4 overviews the different types of process modules, while section 5 discusses alignment. Section 6 discusses some efficiency features. Section 7 describes some of the implementations of LIAM and section 8 concludes.

2. OBJECTIVES

The construction of a dynamic model is a very large task, both in terms of grasping the types and forms of behaviour that take place over a lifetime and the effort in programming thousands of lines of code.

When DMMs were first developed, they represented advances in computer science as well as in social science methodology. However, despite dynamic microsimulation models (DMM) having existed since the 1970s (Orcutt *et al.*, 1986), developments in the field of microsimulation have progressed only slowly (O'Donoghue, 2001a). Part of the reason has been the computational resource requirements. In many countries data limitations have also significantly limited developments.

Fortunately, in recent years, both difficulties have started to be overcome. Computers have increased in speed, now allowing very powerful models to be constructed on Personal Computers. At the same time the establishment of household panel datasets in many countries, for example the European Community Household Panel Survey, the British Household Panel Survey and the German Socio-Economic Panel, allied to the increasing availability of administrative datasets, removed the barrier to the estimation of dynamic behavioural processes.

Even so, the spread of the DMM technology and the development of the field has been relatively slow. The models that are being used at present are not doing very much more than the DYNASIM model in the late 1970's. A large potential reason is the apparent benefit to cost ratio. Many institutions, when faced with the large cost of developing a dynamic model, feel the money better spent on other techniques. One significant contributor to the cost of development is the cost in producing the computing environment of the model. In addition, because the computing necessary to produce a

dynamic microsimulation model is so complicated, computing development has often taken precedence over developing better behavioural equations. It is therefore important to focus on ways of reducing the cost of building this initial framework.

O'Donoghue (2001a) surveys the dynamic microsimulation models that have been constructed, describing in particular the design choices that have been faced. While most models have been built as stand-alone efforts, a number of attempts have been made to avoid the start-up costs and learning curve in building the model by utilising the same framework for alternative applications. There were some efforts in the 1970s to write generic microsimulation computer software packages. However because of the complexity of the systems to be simulated, users typically found that they had specialist requirements that these software packages could not cater for. (See, for example, Leombruni and Richiardi, 2005.)

Although not designed with objective of constructing multiple dynamic microsimulation models, the code from CORSIM model (Caldwell, 1996) has been stripped down and used as a template in the construction of the Canadian DYNACAN, Swedish SVERIGE and the US Social Security Administration models. Subsequently there have been four examples of programs that have been written explicitly for multiple dynamic microsimulation model construction: ModGen (Wolfson and Rowe, 1998), UMDBS (Sauerbier, 2002), GENESIS (Edwards, 2004) and LIAM – the focus of this paper.

MODGEN is a computer language designed to create microsimulation models, and has been used to create a number of microsimulation models (dynamic and static) within the Canadian government, such as Lifepaths. UMDBS is a simulation system developed at Darmstadt University as part of academic research. It is implemented in the object oriented language Smalltalk and its main applications are socio-economic investigations. GENESIS is a SAS based modelling framework being used within the UK Department of Work and Pensions to create the Pensim2 pension age dynamic microsimulation model and the state pension forecasting model. LIAM, unlike MODGEN, is a closed model, in which spouses are drawn from individuals within the model population. In addition, LIAM is fully accessible to researchers, whereas GENESIS is currently available for internal government use only.

In this paper we describe the LIAM framework which was developed, ironically, because of the limited resources available to the author. Dynamic models have typically been constructed by governmental institutions (MOSART, SESIM, DYNACAN, PENSIM2) or by major research grants (SVERIGE, DYNASIM, POPSIM), although a number of models have been constructed as part

of PhDs (Harding, Baldini). Not being funded by a major research grant or by a government institution LIAM falls into the latter "low budget" category, being developed initially as part of a PhD and latterly expanded with small research grants and with the assistance of a number of PhD students. As a result, it is freely acknowledged that current implementations of LIAM are relatively basic compared to the existing proprietary models listed above. But these shortcomings can be overcome, subject to improved data and funding availability. The advantages already offered by LIAM lie in its flexibility and modularity, and its objective of providing an unconstrained model development platform, adaptable to future uses and future proofed to allow for future enhancements.

The initial application of LIAM was a single cohort life-course analysis of the redistributive impact of the Irish Tax-Benefit system, using relatively unsophisticated data and behavioural equations. Subsequent developments (described in more detail below) include improved data (2-8 years in the panel data underlying the behavioural estimations), the addition of a graphical user interface, the move to a multi-cohort population model, and the use for alternative policy analyses such as spatial, indirect taxation and international comparisons. Future developments that are planned include improving the behavioural equations to respond to changes in the policy environment such as labour supply retirement and migration.

In order to avoid in-built software limitations inhibiting future model developments, careful thought is necessary in the design of a flexible modelling framework. There are a number of features that are desirable in such a framework:

- In order to deal with new datasets with ease, using different sets of variables should not be a problem.
- It should be easy to incorporate new behavioural information.
- Ability to run on a personal computer using standard "inexpensive" software.
- It should be straightforward to make changes in the model using the framework, even if the model has not been used for a period of time, or is being used by multiple analysts. This implies transparency in the operation of the framework and also flexibility in the way in which behaviour can be incorporated in the model
- A framework that is robust to the modelling changes desired.
- Computational efficiency as, despite computing time decreasing with the availability of cheaper and faster computers, computational demand has increased at a faster rate.
- A modelling environment that allow the user to focus more on the estimation of behavioural equations rather than on computing issues.
- Ability to take account of, and examine, the feedback effects of policy reforms.

3. FRAMEWORK FEATURES

In this section we describe the main aspects of LIAM, focusing initially on the general structure of the framework and then elaborating the data structure and issues relating to modularisation and parameterisation.

Structure of Framework

A dynamic microsimulation model takes individual objects (individuals, households, farms, companies) and simulates the probabilities of various events occurring at various points in time. Dynamic events may of course occur at the same point in time as other events.

Figure 1 describes the main operations of the ageing component of the LIAM dynamic microsimulation model. In this context 'ageing' is a generic term covering any dynamic event that involves updating object characteristics over time. Here the operation of one particular ageing module at one point in time is examined. In reality this process occurs on a number of occasions as all the individuals in the database pass through a number of ageing modules at each point in time.

Data for each person are firstly taken from the database, having been transformed into the model data-structure, which is described in more detail below. The individual is then passed through each ageing module in turn. The ageing modules to be used are specified as part of a parameter list, which allows the order and the types of the transition processes to be varied. Input

parameters for each ageing module are stored in Microsoft EXCEL spreadsheets and are accessible via the user front end. Output from each ageing module is stored, in memory, in alignment storage matrices. For example, alignment regressions produce a deterministic component X_B to which is added a stochastic component, ε . These are stored in a dynamic data structure and ranked with the highest Z percent of values taken from the exogenous totals in the alignment process. If the ageing module is a transition between states, then the output will be a probability, otherwise if the ageing module is a transition between continuous amounts, for example incomes, the output is a real variable. When all individuals have been passed through the particular ageing module, alignment occurs (see section 5 below for a description). This ensures that aggregates from the micro model match macro aggregates. Finally if a variable for any individual changes then this change is registered in the database (i.e. in both the physical relational database, stored on the hard disk in ASCII and the virtual database stored in random access memory). In what follows the operations of each of these components of the ageing module are unpacked in more detail.

Data and Framework Data Structure

In this section we describe how data is handled in LIAM. We describe the database used and how data are stored within the framework. The data structure or format of the data is very important as it determines to a large extent the amount of memory required to store the data, which in turn influences the speed at which the model can run.

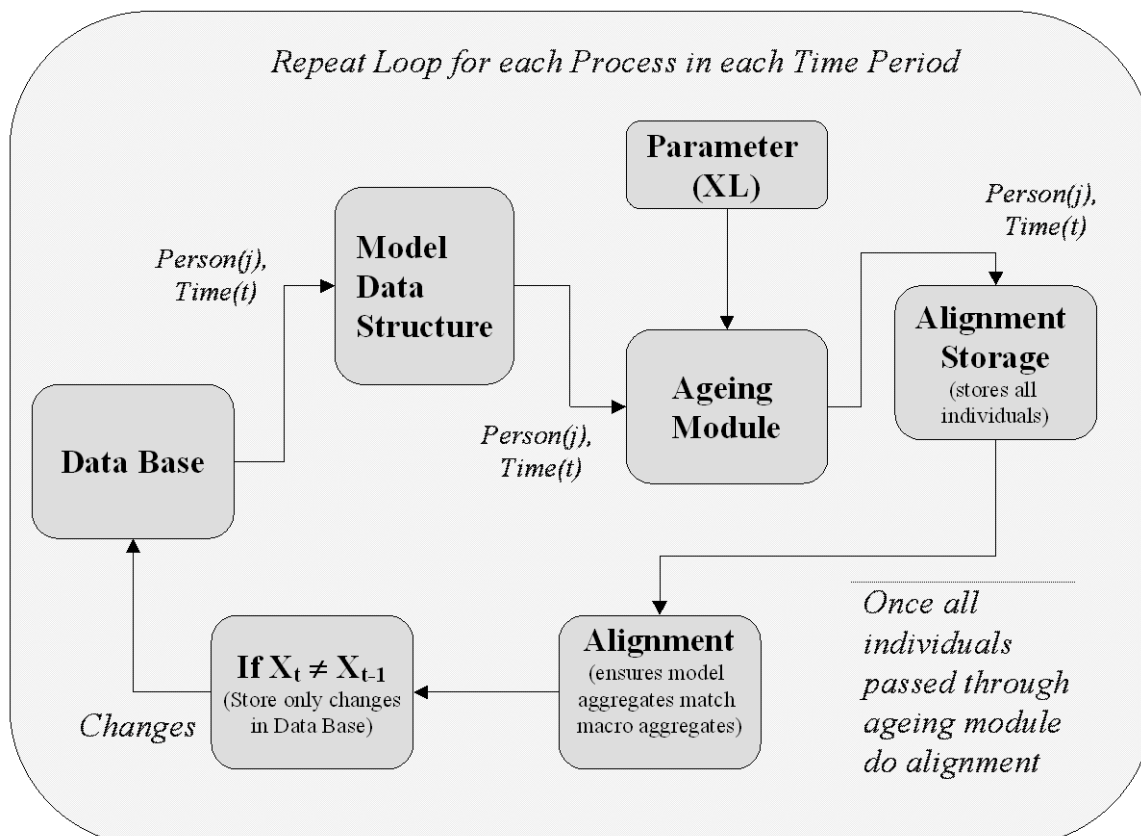


Figure 1 Description of a Dynamic Module

It also has important consequences for the flexibility of the model.

Turning first to data storage, input output data are stored in ASCII format. When storing output, variables are converted from real to integer as integers require less storage space than real numbers. This is achieved by multiplying output values by 100 and truncating the result to zero decimal places. For storage we also adopt a relational database structure due to organisation and memory handling advantages.

Figure 2 describes the data-structure used by LIAM. Structurally the data is stored in a hierarchy of object types (tobjt) such as person, household or firm. Each of these object types themselves consists of a number of objects (tobj) such as the actual incidence of a person or household. Events (tvar1) such as births, tenure status or identification number then occur to objects. Each event can have a number of incidences or values (tval). Within this data-structure persons are considered one set of objects and households another, with the IDs of the member individuals of a household stored as events that can happen to households. This is because persons can be members of a number of different households over time, breaking down the traditional nesting of persons with households associated with cross-sectional data structures.

We exploit the hierarchical nature of relational databases making data storage event driven.

Storing model output as consecutive cross-sections would result in severe inefficiencies, as each variable would be stored for each output period, so for example a person's gender would be stored for each point in time. Making data storage event driven, new data is stored only when a new event occurs and thus the data changes. Gender is therefore only stored at birth. One can make significant savings in memory as a result. Each individual variable, however, requires more information than is the case in a cross-sectional data structure. For each event it is necessary to know what event occurred (tvar1), when it occurred (tval.evtime), and the value of the event (tval.amount).

There are a number of ways in which data can be stored within LIAM itself during the simulation. If the model were open, as in the case of the DEMOGEN or LifePaths models in Canada, where new spouses are generated synthetically when needed, then all of each individual's transitions could be simulated independently of other individuals. Thus each individual could be read from the database, their life course simulated and then stored in the database one at a time. LIAM, however, uses a closed model. Except for new births or immigrants, no new individuals are generated. Marriages, for example, link individuals already in the database. This method is more straightforward to interpret as it mirrors what happens in the real world. As a result of this closed approach, individual behaviour can become dependent on the characteristics and behaviour of

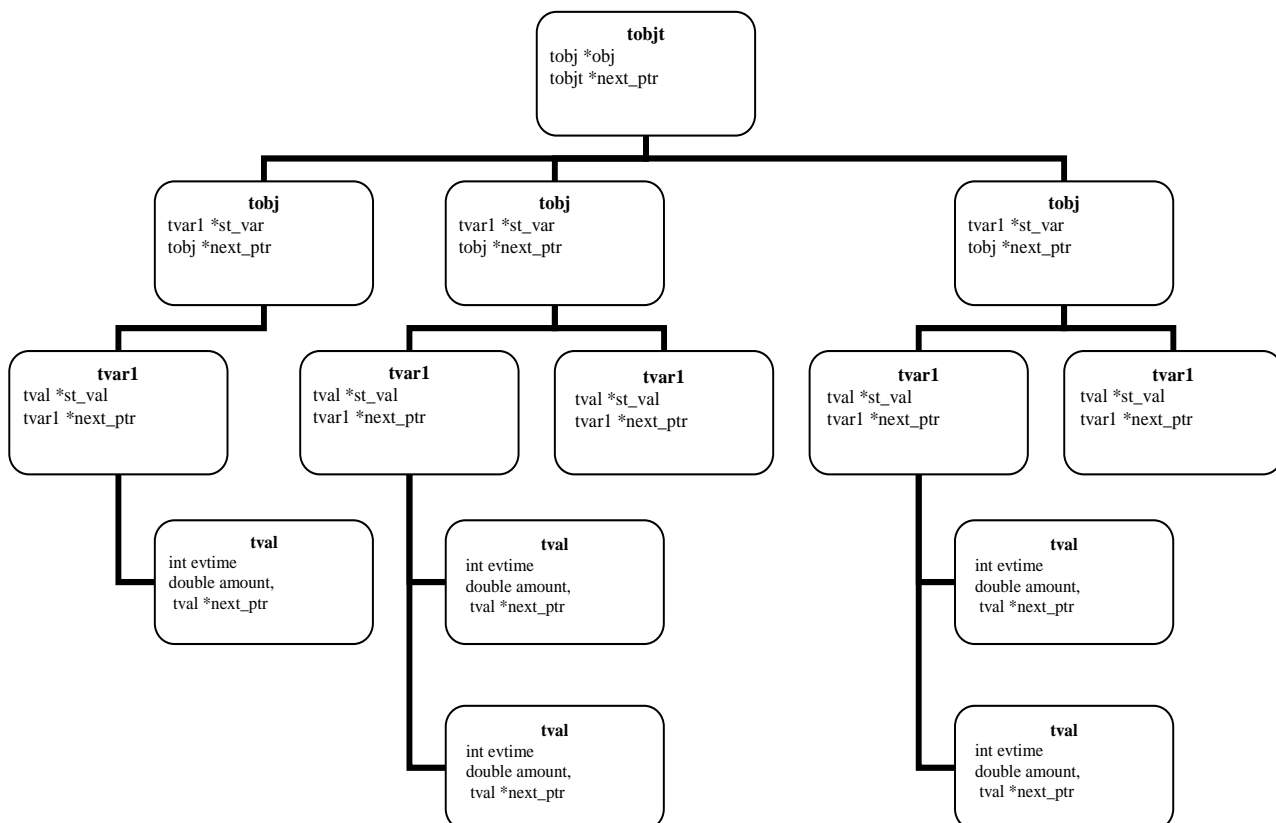


Figure 2 Model data structure
Notes: See text for explanation of object and event types

other members of the sample. For example, alignment to required global totals means that the chance of employment for an individual is not independent of the rest of the population, but depends upon the aggregate outcomes of the whole labour market. Other operations in the model that may depend on other individuals include the marriage market and other processes which depend on spousal information and alignment routines.

Although the model is not solely individual based (it is a multilevel model comprising Regions, Counties, Districts, Households and Persons), a side effect of the inter-person dependency engendered by the closed model approach is that it is necessary to store all individuals in memory during the simulation. To this end, therefore, during a model run the virtual database stored in memory during the operation of the program mimics the structure of the relational database stored on the hard disk. Once the data have been read from the database into memory, LIAM runs through each object type (person, family and so on), in turn simulating the life course events desired for each object of that type. Simulation processes are therefore object-type specific

Typically variables which are components of the household data structure are declared in long lists within a dynamic model. They may be initialised elsewhere and have other operations carried out in other parts of the program. As the modelling framework is so large and complicated, it rapidly becomes difficult to keep track of all the places in LIAM which need to be altered when a new variable is included. As a result, instead of declaring variables within LIAM, we declare the list of variables to be used separately in a parameter sheet (dyvardesc) .LIAM then creates space for the variable, initialises the data and carries out all necessary transformations and operations automatically. This minimises the number of model alterations necessary when a new variable is introduced, keeping the framework entirely flexible with regard to the set of variables used whilst maintaining its robustness. For example, if the user wishes to introduce a new instrument with an output variable such as health status, then the user simply needs to introduce the variable into the parameter sheet and LIAM will do the rest, without the user having to recode the framework itself. Another advantage of the flexible declaration of variables is that, because variables are stored in vectors, new composite variables can be produced easily. For example, a complex variable like disposable income, if not simulated directly, can be generated from the vector of its components such as employment income and capital income.

Another important advantage of the hierarchical method of data storage is the ease with which duration information can be accessed. As the date and value of each event is stored it is possible to determine such information as duration, duration in the last 12 months, date an event first

occurred, date an event ended, duration in a particular state and so on. Information of this kind is frequently required by tax-benefit systems and other policy analysis. Additionally it is easy to access earlier values of an event such as previous earnings.

Population versus Cohort

The initial database depends on the purpose of the simulation. One of the key distinctions in the literature is between longitudinal or single cohort models and population or cross-section/multi-cohort models. However, this distinction is now largely redundant due to advances in computing power. From a computing perspective, a cohort can simply be seen as an initial sample of unrelated individuals aged 0, while the population contains a sample of individuals of different ages, some of whom are related. As a result, the computing framework has been developed to handle both types of analysis. Running LIAM as a dynamic population model requires that the initial cross-section is stored in the required manner, while running the model as a dynamic cohort model requires the model to first generate an initial cohort.

Modularisation

The use of modularisation is an important technique that helps achieve the objectives of flexibility, transparency and robustness that LIAM requires. Modularisation means that components within the LIAM are designed to be as autonomous as possible. Modules are the components where calculations take place, each with its own parameters, variable definitions and self-contained structure, with fixed inputs and outputs. The result is a set of independent components that do not interact with each other directly, allowing the framework to operate as a collection of independent building blocks. Because each process module is entirely self contained, each can be run independently, left out or replaced by an alternative module. Constructing a program in this way allows for the model to be easily expanded to deal with new behavioural equations or functions. Also, because it allows the user to focus on individual components one at a time, without interaction with the rest of the program, the robustness of the model can be improved.

Linkages

Many policy instruments depend upon multiple units of analysis. So, for example, pensions may depend upon individual characteristics such as contribution histories, age and so on, taxation may depend upon family characteristics such as both spouses' incomes, and social protection instruments and welfare measures upon the household unit. Similarly sociological analyses and long-term care analysis may depend upon wider kinship networks. These linkages are not strictly hierarchical (e.g. region, household, family, individual). They may, in fact, consist of a web of linkages (region, firm, household, family, individual, mother, father, partner, children and so on). This multi-level structure with its complex

interactions between levels is one of the main complications of microsimulation models that make it difficult to use person-based modelling frameworks. While it is not infeasible to simulate using non-hierarchical linkages such as those between relatives across households using other software packages such as social simulation and statistical packages, the non-hierarchical structure often requires one to be "creative" in designing the model due to inflexibilities in the model as they are often non-standard requirements. In contrast purpose designed microsimulation packages such as LIAM can have these data structures in-built, improving the transparency and flexibility of the modelling environment.

In the LIAM framework, the mechanism of linking objects has been automated as a relational database. Potentially any object can be linked to an object of either the same or a different object type. For example individuals of object type person (p) will be linked to their household of object type (h), while in turn the household is linked with the individuals in the household. Therefore calculation of the number of persons in a household is a process carried out at the household level. This new household level variable, `npers`, can be accessed by individual processes using the prefix `h_npers`. (The actual prefix used is user-defined.) In similar fashion linkages can be made between objects of the same type. For example a child can be linked to parent's information, accessing mother's education level using `m_edlevel` and father's using `f_edlevel`.

In the initial framework, there are no predefined linkages as the objects can be of any type defined by the user. The user pre-defines all linkages using the parameterisation described below to essentially create a web of linkages between objects; essentially defining keys to link tables. As long as the nature of the linkage is defined, it is then possible at any level of the model to access information from another level. This is quite a powerful feature of the data-structure, saving both time and memory. In the absence of these linkages, such as `h_npers`, a new process would have to be simulated which would store this number of persons in a household as a person level variable `p_hnpers` (say), which is analogous to a flat file, where household level variables are stored at the person level. The use of linkages or keys provides the space saving advantages of a relational database and avoids the simulation of an extra process to convert the household variable to the person level.

Creating and Killing Objects

While creating a new object (person, family, household, enterprise) is itself not a very complicated task, creating space and assigning initial default values, creating a new object to mimic the birth of a new person is rather more complicated. As a result we have had to develop a specific as opposed to generic `new_birth` function. The assignment of variable values such as single,

age zero, no education and so on is straightforward. However it is also desirable that the new child inherits the linkages of the parent. So, for example, the partner (if any) of the mother at birth becomes the child's father. Similarly other children of the mother become siblings and the hierarchical linkages such as the household, family and region of the mother are also inherited by the child.

Analogously, "killing" a person is also more difficult than killing an object. The individual needs to be extracted from the web of kinship networks and other linkages of which they form part. For example, the number of persons in a household decreases by one, whilst the spouse becomes a widow. At the same time bequest of accumulated wealth may need to be transferred and, in the case of pensions systems, contributions or entitlements may need to be transferred to surviving dependants. Again the possible complexity of the operation is far too difficult to generalise, so again object-specific routines have been required.

Migration

Migration is another complex operation. From the point of view of LIAM, emigration is analogous to killing someone. The reason for this is that, in a national model, we do not track individuals while they are abroad. (Technically, as pension rights can be accumulated overseas, one may need to simulate their life-histories while overseas, but this has been beyond the capacity of any existing model.) A variation of the pageant algorithm (Chénard, 2000) is used to ensure that the migration of family units results in national individual level aggregates being achieved.

Immigration, however, is a more difficult situation. An immigrant differs from a new birth in that they have an accumulated set of characteristics and potentially are accompanied by other family members. One solution is to simulate the range of characteristics of new immigrants on arrival. However the range of characteristics is very broad and variable and so it would be very difficult to retain the correct multi-dimensional distribution of characteristics. To avoid this problem we sample (with replacement) from a set of immigrant households in the data. Thus whenever we need an immigrant household we simply select a "real" (data-dependent) family with the actual characteristics of a new immigrant family. In addition to preserving the multi-dimensional distribution, it saves substantial computing time compared to having to simulate all of the relevant immigrant characteristics.

Parameterisation

In order that modules and other components of LIAM can be changed with ease, it is necessary to store model parameters externally. Therefore, where possible, parameters are not hard coded within the framework. Figure 3 details the set of parameters used by the modelling framework. The sets of parameters, representing the *flow of*

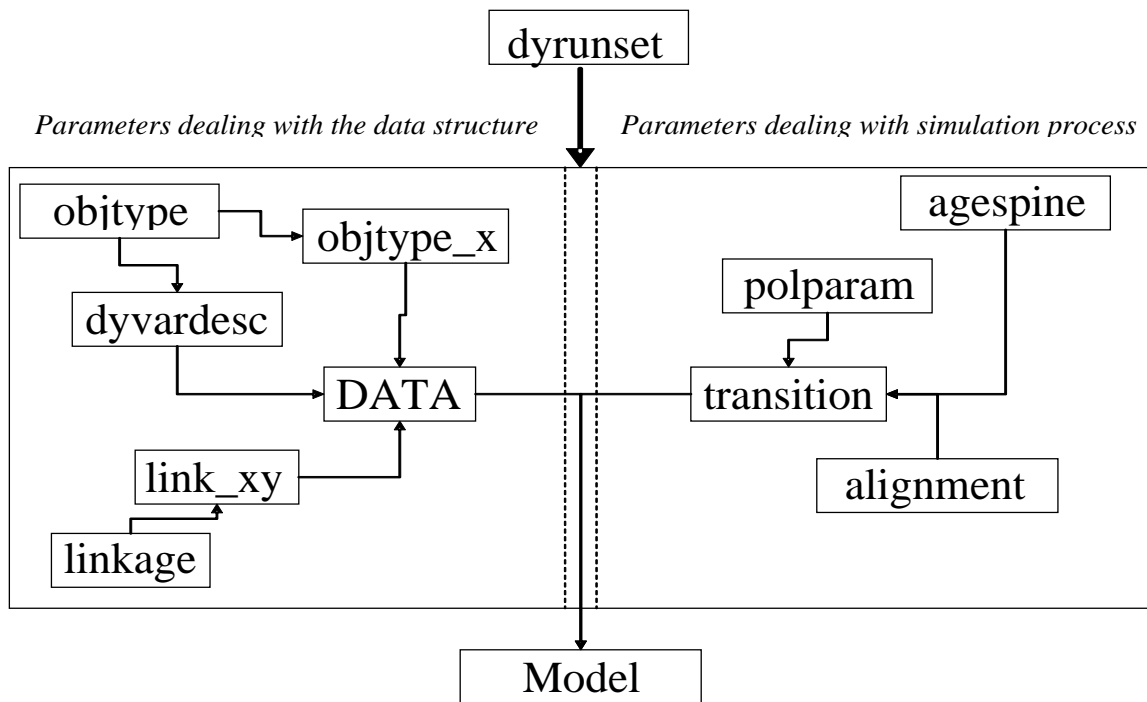


Figure 3 Parameter Sheet Hierarchy

Notes: See text for explanation of parameter types

control in the model, are in some sense hierarchical.

At the top level we have *dyrunset* parameters which contain the parameters necessary to run the model, detailing directories (location of input and output files), time period to be run and so on. The remaining parameters deal with either the data structure or the simulation process. Dashed lines in Figure 3 indicate this division.

On the data side the highest level parameters are contained in *objtype*. This file tells the model how many object types there are (region, household, person and so forth). LIAM creates each object type based upon the list defined here and assigns user-defined prefixes (for example, r,h and p). The framework then looks for files *objtype_x* containing the incidences of each of the object types (r, h, p), which in turn contain the identification numbers (IDs) of each object of that particular object type. So *objtype_p* would contain the IDs of all persons.

Related to the set of object types is a set of variables associated with each type in the *dyvardesc* file. In this file, all variables used in LIAM are declared and described. (In the front end, the parameter files have equivalent menus.) It is, in essence a data dictionary, for the model. This file contains information on the following attributes of each variable:

- variable name
- variable type (binary, multi-category, continuous)
- whether an income variable (monetary amount that can be added to other monetary amounts) – this prevents categorical data

being summed; for example, adding gender to employment

- limits of the variable (upper and lower bounds) for debugging and validation
- whether or not a categorical variables (if so how many categories and the list of categories – for tabulation purposes),
- whether or not needs to be updated during the simulation to account for inflation
- default values to be taken by new persons
- data description (describes variable names)

Associated with each variable, there is a data table containing information about the object(s) associated with the variable (who), the time the event occurred (when) and the value of the variable (what).

While each data table within an object type is linked by the key or object ID, we need further information to link objects of the same or other type. The *linkage* parameters define the set of possible links between objects. The user needs to define a name for the linkage (for example, ph – person to household; hp – household to person) and the equivalent origin and destination types (p for link ph; h for link ph). These linkages between objects are stored in the linkage file *link_xy*. Subsequently, for each linkage listed in *link_xy*, LIAM pairs the relevant origin objects and destination objects (for example children to parents) and stores the resulting origin and destination IDs in a link-specific file (for example pc for children and parents).

We now consider the set of parameters that define the simulation processes. The highest level is the *agespine* or process spine/list. In any simulation

there is an implicit ordering, and events are triggered through conditions. The process spine contains the list of modules to be run in the dynamic model, so that by varying the order of the modules and varying the content of the list, one can vary the types of processes that can be run in the model. This feature exploits the modularisation inherent in LIAM, where because each process is seen as a separate building block, the number, type and order of processes can vary without having to change the programme code. Each process or module has a corresponding parameter sheet in the parameter file "Transitions". These parameter files tell the model the output variables of each process, the type of process, whether a process needs to be aligned and the actual process parameters themselves, such as the transition rates, regression equation and policy rules and so on. If a particular process is to be aligned, then LIAM will look for an appropriate set of *alignment* parameters. Sometimes individual parameters may be required to be changed between runs without any change to the set of processes. A reform to a pension simulation module where the replacement rate was changed is an example. The *polparam* parameter set contains information associated with each parameter for each possible "system", where the system to be run is defined in the *dyrunset* parameters.

4. PROCESS MODULES

This section describes the main process types that can be used by LIAM. This refers to the collection of operations that are simulated on objects during a simulation. These include demographic processes such as birth, marriage, having children and death, education, labour market processes such as employment and unemployment, the simulation of incomes and interactions with the tax-benefit system.

In order to aid flexibility, we classify processes under a number of headings. In this way, instead of programming each module separately, we only need to program the module type once. In order to run a module, we then only need a module name (which is included in the process spine), a module type to determine which program to run and a set of parameters which is fixed for every process type. At present there are 6 module types:

- transition matrices, in the form of a log linear model (*trap*)
- transformations (*tran*)
- regressions, both with continuous and limited dependent variable (*regr*)
- macro alignment (discussed in section 5) (*macro*)
- marriage market (*mmkt*)
- tax-benefit system (*tb*) (actually a collection of modules; we have linked LIAM to the EUROMOD EU15 tax-benefit model and to other tax-benefit routines)

The first component of a parameter file contains details about what conditions need to hold for the process to be run. At each point (step) in time, each individual is passed through the module. If the conditions hold, then the module calculations are carried out and the output passed to the alignment component of the module. The output for each individual is stored until all individuals have passed through the module. The alignment component then ensures that the aggregates correspond with external control totals.

Transition Matrices

One of the most important processes in a dynamic model is the transition between different discrete states. Transition Matrices are often used to perform these operations. They specify the probability of an individual with particular circumstances moving from state A to state B. In this framework, transition matrices can be stored as log-linear models (See, for example, Dobson, 1990). In this way transition rates are decomposed into average and relative transition rates. As a result extra-relative transition rates can be added with ease. For example, if a mortality rate on average fell by 0.1% every 10 years, then a time-dependent relative probability parameter of 0.999 could be added. Similarly, this approach allows the model builder to combine information from different sources. So, for example, we combine actual age-gender specific mortality rates for 1991 taken from life-tables and use relative mortality rates taken from Nolan (1990) that incorporate socio-economic relative mortality rates.

Regressions

The second type of transition process used is based upon standard regression models. At present, this type of module allows four types of dependent variable

- standard continuous dependent variable
- log dependent variable, allowing for use of the log normal distribution.
- logit discrete choice dependent variable
- probit discrete choice dependent variable

Any variable in the model can be used as a dependent variable and any variable can be used as an explanatory variable. The error term can also vary. The default error term takes a normal distribution with independent disturbances. Following Pudney (1992), LIAM also allows for the error term to be decomposed into individual specific (u_n) random effects and general error components (v_{nt}). However, more complicated error decompositions are also possible. This allows some degree of heterogeneity to be assigned specifically to individuals. So, for example, in determining earnings the individual specific error may represent some difference in innate ability, while the general error term represents random variation over time. Breaking up the variation in this manner will tend to reduce within lifetime variation and prevent to some degree the existence of very unusual life paths.

In the LIAM modelling framework, transitions occur at discrete time intervals because of the weakness of the data and because of the desire to be able to align the data. As Galler (1997) points out, the statistical shortcomings associated with the use of discrete time models make it is desirable to use short term discrete time periods such as a month. As the computing requirements can be substantial for monthly simulations, LIAM is sufficiently flexible to allow the user as the ability to specify the time period to be used and so monthly or annual periods can be simulated. For a fuller review of the advantages and disadvantages of continuous time versus discrete time, see O'Donoghue (2001a).

Transformations

While regression models and transition matrices are stochastic processes, involving a random component, some processes are deterministic. Examples include age, which depends on the date of birth, widowhood, which depends on the death of a spouse and so on. Likewise if an individual moves from year 6 in education to year 7, years of education increase by 1.

Within the transformation-types handled by LIAM there are two types of deterministic transformation, *gen* and *fgen*. The *gen* functions are of simpler types, utilising calculation routines combining sets of variables using standard operations ($\{+, -, *, /, \max, \min, ^, (,)\}$). The *fgen* set of functions represent ad-hoc programs. It is where we exploit, for example, the relational database structure of the data in operations such as the number of persons in a household, where the function counts the number of objects of type person linked to the object household as defined by the *link_hp* link file. Similarly it is where ad-hoc functions such as *new_birth* and *killperson* are defined. While *gen* functions and predefined *fgen* functions are pre-coded and parameterised so that new users can employ them, new *fgen* functions such as the pension system of a new country need to be programmed by the user if the existing functionality does not allow it.

Marriage Market

If an individual is selected to marry or form a partnership a process is needed to determine which spouse they will take. The process used here is to take the characteristics of the individual chosen to marry and the characteristics of each possible spouse and determine the likelihood of a match. Similar to the method used in other models such as the CORSIM model, this is done using a logit model that estimates the probability of marriage between pairs of individuals. The parameter file therefore is identical to that used in the regression process type. The module itself forms a matrix of the characteristics of the n men and n women selected to marry. It estimates a probability for each pairing and assigns a match to the couples with the highest probability of marrying.

Bouffard *et al.* (2001) have identified some

problematical issues associated with the marriage market, in particular with strange matches occurring amongst the last people to be married in a particular simulation. In order to avoid these issues, we allow the user to create a super-set of potential male partners, so that rather the last female in the marriage pool being forced to select an unlikely match, there remain a number of males to choose from. In addition we employ the Order of Decreasing Differences algorithm proposed by Howard Redway at the UK Department for Work and Pensions, which creates a measure of the distance of an individual from the centre of the population (or the average characteristic of the population) and first selects for matching the females with the most unusual characteristics, who are likely to be the most difficult to match. The logic is that those in the centre of the data are "average people" who are more likely to find a good match than someone at the extremes.

Policy Processes

The fourth process type is the simulation of the tax-benefit system. Re-emphasising the desire to reuse code and, wherever possible to avoid duplication, the dynamic framework has been made flexible enough to link with other specialist programs such as pre-existing tax-benefit models. As a result tax-benefit routines from other models, such as EUROMOD, can be seamlessly accessed and thus used as module components of the dynamic model.

Behavioural Response

A desirable feature often ignored in dynamic microsimulation models is the ability to include feedback loops so that behaviour can respond to changes in public policy. A criticism made by Citro and Hanushek (1997) is that dynamic models are insufficiently flexible to incorporate the demands of behavioural response. In order to be able to simulate behaviour, typically the model needs to be able to call a policy simulation routine a number of times to quantify the financial impact of alternative choices on the decision in questions such as the choice to work or retire. As a result the software framework has been designed to be able to incorporate feedback loops. The degree of modularisation that exists in the framework allows any number or order of modules to be run and for modules to be able to be run a number of times.

For example, in O'Donoghue (2001b) we implemented a simple labour supply model where labour supply depended on the tax-benefit system. In order to have labour supply depend on tax-benefit policy, the tax-benefit system needed to be run once as an input into the labour supply module and again, after labour supply has been determined, in order to recalculate taxes and benefits in the light of the resulting behavioural decision. In O'Donoghue (2001b) the model used the tax-benefit system as an input into decisions to work, decisions to seek part-time employment versus full-time employment and to become self-employed. The tax-benefit system therefore

needed to be run 5 times to examine the impact of the system on the choice faced by an individual. In the case of spouses, whose behaviour can be inter-dependent, the tax-benefit system needed to be simulated 17 times (4 decisions for each spouse, plus one run on the basis of resulting joint behaviour). As a result incorporating behavioural response, although possible, can be computationally expensive.

Other possible behavioural routines that could be included are retirement decisions, consumption and benefit take-up. Although computationally expensive, the framework is sufficiently flexible should the user require and the computing power becomes available.

Robustness

Finally, in order to avoid robustness problems due to modules being incorrectly specified, the model contains a debug device which ensures that all inputs required by a module are actually available (i.e. have either been generated in the model or read from the database) before each module can be run.

5. ALIGNMENT

The section describes the alignment function contained in LIAM. The objective of alignment is to ensure that output aggregates match external control totals. The reason this is done is that micro behaviour (both social and economic) is extremely complex and, micro-theory being limited, the full variability of the system (in this case the life paths of individuals) cannot be accurately predicted. In addition, a household model only makes forecasts about a small part of the economy and largely ignores interactions with the rest of the world economy. Finally, data taken from relatively short periods of time may not fully reflect dynamics over time. For all of the reasons dynamic micro-models may not be able forecast aggregate characteristics of the population well.

Discrete choice models

In discrete choice models the output for each individual is a probability. In order to use these models for predictive purposes, a decision rule is necessary. In other words, what forecasted probability (or higher) will produce an event. In order to predict a state with a logit (or probit) model, one draws a random number uniformly distributed number u_i . When

$$u_i < \text{logit}^{-1}(\alpha + \beta X_i)$$

or

$$u_i < \text{probit}^{-1}(\alpha + \beta X_i),$$

then a state is predicted to occur.

Another use of alignment is in correcting for predictive failures of econometric models. For

example, when using discrete choice models such as logit or probit models, the predictive power is often poor. Duncan and Weeks (2000:292) highlight that "even in functionally well-specified models, the predictive performance is poor, particularly where some states are relatively densely or sparsely represented in the data". Greene (1997:894) attributes this to the fact that "the maximum likelihood estimator is not chosen to maximise a fitting criterion based on prediction of y , as it is in the classical regression (which maximises R^2). It is chosen to maximise the joint density of the observed dependent variable." Thus the further the probability of an event occurring is from 0.5, the less effective these decision rules are at producing the desired result. As a result models may under or over predict the number of events. So, for example, if 5% of individuals of individuals should have the event, then the logit model may not necessarily produce 5% of events. Alignment, however, will constrain the event to occur to 5% of individuals. This is effectively a calibration mechanism and will produce the correct proportion of events. Of course, care must be in its use as alignment may lead to the disguising of errors in model specification.

The types of control totals that can be used to align to include:

- The aggregate proportion/number in a state or moving between states
- The average event value
- The distribution of values
- The average growth rate in the value of an event

In this paper we shall deal specifically only with the first type

A simple analogy for the relationship between alignment and the process modules is that the process modules, such as logit models, produce a ranking variable, while the alignment mechanism selects the number of transitions. For example, in our econometric model we may have an equation of the probability of dying as described in equation (1), that depends on age, gender and whether an individual is disabled or not. Assuming that disabled people have a higher mortality rate, then given the same age and gender and distribution, the mortality distribution for disabled people will be higher:

$$\text{logit}(p_i) = \alpha + \beta_1 \times \text{Disabled}_i + \beta_2 \times \text{Age}_i + \beta_3 \times \text{Gender}_i + \beta_4 \times \text{Disabled}_i \times \text{Age}_i + \varepsilon_i \quad (1)$$

The deterministic component of the model will result in those with a higher risk having a better chance of the event occurring, while the stochastic part (ε_i) will ensure that there is some variability (so that not only those with high risk are selected). This model therefore produces the person-specific risk of dying.

In order to select the number of people that die, we use the alignment probabilities. Firstly individuals are grouped into the appropriate age

and gender groups. As everyone in the relevant group will have the same age and gender, they only differ by the deterministic component for disabled people, $\beta_1 \times Disabled_i + \beta_4 \times Disabled_i \times Age_i$, and the stochastic component ε_i . The object then is to select to die the people in the group with the highest probabilities of dying. If β_1 is positive, proportionally more disabled will die than non-disabled. As a result we see that the output of the model equation is used to rank the individuals to whom the event occurs, but to leave the decision of experiences the event to the alignment process.

Implementation

In this section we describe a practical method for ranking individuals for alignment. We take as our reference point a logistic model:

$$p_i = \text{logit}^{-1}(\alpha + \beta X_i + \varepsilon_i) \quad (2)$$

Utilising the model

$$\text{logit}(p^*_i) = \alpha + \beta X_i$$

will result in those with the highest risk always being selected for the event. Continuing our example above, the disabled, all other things being equal, would be selected to have a die. In reality those with the highest risk will on average be selected more than those with lower risk, rather than simply selected those with the highest risk. As a result some variability needs to be introduced.

Models based on the CORSIM framework such as the DYNACAN model (Chénard, 2000) utilise the following method. Firstly, predicted probability is produced using our econometric model:

$$p^*_i = \text{logit}^{-1}(\alpha + \beta X_i).$$

Next, a random number, u_i , is drawn from a uniform distribution, and subtracted from the predicted probability, p^*_i , to produce a ranking variable, $r_i = p^*_i - u_i$. This value is then used to rank individuals so that the top x% of values are selected.

A concern about this method is that the range of possible ranking values is not the same for each point. In other words, because the random number $u_i \in [0,1]$ is subtracted from the deterministically predicted p^*_i , then the ranking value takes the range $r_i \in [-1,1]$. However the ranking value for each individual will only take a possible range $r_i \in [u_i-1, u_i]$. So, for example, if p^*_i is small, say = 0.1, the range of possible ranking values is [-0.9, 0.1]. At the other extreme if p^*_i is large, say = 0.9, then the range of possible ranking values is [-0.1, 0.9]. Thus because there is only a small overlap for these extreme points, even if a very low random variable is selected, then an individual with a small p^*_i will have a very low chance of being selected.

Ideally the range of possible ranking values should be the same, so that for each individual, $r_i \in [a,b]$, with individuals with a low p^*_i being clustered

towards the bottom and those with a high p^*_i being clustered towards the top.

We now consider an alternative method. This method takes a predicted logistic variable:

$$\text{logit}(p_i) = \alpha + \beta X_i.$$

Next, a random number is drawn taken from the logistic distribution. This is added to the prediction, giving

$$\text{logit}(p_i) + \varepsilon_i.$$

The resulting inverse logit,

$$p_i = \text{logit}^{-1}(\alpha + \beta X_i + \varepsilon_i)$$

is then used to rank individuals and similarly the top x% of households are selected.

The rank produced by the two methods is not the same. The second method will be more likely to select cases at extreme points than the first, while first method will select more points with central values of p^*_i .

Macro Alignment

There are a number of levels at which alignment can occur. At the lowest level, alignment refers to the decision rule used in a discrete choice model. The next level, described above in our mortality example, which is called the meso-level, concerns the idea that the aggregates for particular groups (in this case gender and age) should match desired external totals. Meso-level alignment and the use of alignment as a decision rule can however be combined into one stage.

Sometimes the desired targets are narrower than the alignment targets we use. Extending our mortality alignment example, suppose we align mortality by age, gender and occupation. We wish to include occupation in the alignment as occupational structure is very important for other characteristics in the model, and because there are known to be significant differentials in occupational mortality rates. However if one of the core targets in the model is to achieve the mortality distribution supplied by external sources such as official population projections, which may be disaggregated only by age and gender, then our meso-alignment may produce different aggregates. This will happen if our underlying occupation distribution is different to the one implicit in the official forecasts. It may therefore be desirable to adjust the results again to achieve these targets. This process is known as macro alignment. Another example follows on from the meso-alignment in the simulation of transitions between employment states. Macro alignment is then used to constrain total employment rates. See Appendix 1 for the steps involved in the macro alignment process.

Behavioural Change

Handling behavioural interactions in the model resulting from alternative scenarios is another issue one needs to consider when deciding on an

alignment strategy.

One potential solution is to compare the average (pre-alignment) event value, such as the average transition rate or average earnings in the baseline scenario, with the average in the alternative scenario. One potential method is to increase alignment values by proportional difference. This is a method utilised in some dynamic models. However, this approach assumes that all processes are *unconstrained*. In some cases, such as mortality rates, this may be a realistic assumption. One may expect that an exogenous increase in human capital will reduce total mortality rates. In this case shifting down the alignment totals is appropriate.

Other processes face market or other institutional *constraints*, issues that are only partially simulated in the model. One example involves the labour market, where there is a behavioural change in labour participation in response to a tax change. If labour supply increases, then wages would fall and employment increase. This is similar to shifting the alignment probabilities. However one would have to shift earnings as well. Unfortunately, due to rigidities in the labour market, this may not necessarily happen. Labour demand may be fixed, in which case we may just simply see that as more women supply labour, they simply replace people in the labour market who are less "employable". This is similar to not shifting alignment at all. In cases where there are market interactions such as this, it may be useful to incorporate a model of the market that would inform the response of alignment totals to economic and demographic totals. At present the framework makes no explicit incorporation of behavioural change in the alignment structure. Future work on macro-micro linkages will attempt to address this.

6. EFFICIENCY

In initial versions of the LIAM framework, developments focused on functionality and not speed. So for each process the model passed through all the objects of the object type, checked to see if a condition was true and, if true, performed the calculation $f(X\beta + \varepsilon_i)$ before, if necessary, using alignment to produce the predicted value of the process. In this section we describe a number of improvements in computational efficiency, relative to this initial approach, that have been made recently.

A first efficiency saving relies on the fact that most processes are relatively stable and so do not change much year on year. Because of this, eligibility conditions are unlikely to change much year on year. For example, for lone parent births the model used to check to see if an object was female, single and of child bearing age. It did this for each year and for every person. This is inefficient as gender doesn't change, so there is no need to repeatedly check whether a male can

have a child. Similarly marital status eligibility does not need to be retested annually, given that marital status changes only infrequently. In the same way, the age range condition (child bearing age) only changes twice over a lifetime. A key speed improvement, therefore, has been to calculate the conditions for all people in the first year of the simulation and then only to recalculate the condition if an input variable to the condition changed.

The same is true when calculating regressions. Most regressions are of the form

$$f\left(\sum_{i,j} X_{ij}\beta_j + \varepsilon_i\right) = f(A + \varepsilon_i)$$

Again, by calculating the value of the expression in the first year, when X_{ij} changes to X_{ij}^* one only needs to apply the following transformation

$$f(A + -X_{ij}\beta_j + X_{ij}^*\beta_j + \varepsilon_i).$$

The same speed efficiencies can be found for transformations, alignment and tabulations. For example, when aligning by age-group, sex and education level, or creating output tables by these components, most of these categories do not change much if at all during the simulation. It is therefore computationally quite expensive to identify, say, all 20-30 year old males with university education each period of the simulation to perform an alignment. Rather, by computing the group membership at the outset and only changing group membership when a characteristic changes, we significantly reduce the computational costs.

These improvements were applied by creating a data structure that links every variable to the processes which utilise the variable. When the value of the variable changes, then the related conditions, regressions, transformations etc are updated.

The move from periodic simulations to initialisation plus simulation only when inputs change transfers some of the computing cost from the simulation period to the start. Initialisation becomes a good deal longer as, rather than simulating equations where conditions are met (for example, only simulating work equations for those who are in-work the previous period), we must now simulate all equations (i.e. simulate the equations conditional on working and conditional on not working in the previous period). On the other hand, as all that is being calculated is

$$\sum_{i,j} X_{ij}\beta_j,$$

no alignment needs to occur at this stage. The net result is much less looping through the data and significant overall economies of scale.

As development of the program occurred incrementally and different versions have been run on different machines and with different specifications, it has been quite difficult to gauge

the impact of the speed improvements. However a conservative estimate is that the run time is 10% of what it was and potentially as low as 5%. So the qualitative conclusion is that the gains are highly significant.

Another, although less significant, speed improvement was obtained by storing variables in a static rather than in a dynamic data structure (i.e. as an array rather than a list). As the set of variables does not vary within a run of the model, there is no gain to using a dynamic list; instead a speed penalty is imposed as the list needs to be traversed to find a particular variable as opposed to simply using the array index to identify the variable.

While attention was paid in the original data-structure to the memory efficiency of the data-structure by storing only new events, little attention was paid to the space taken within this structure, which proved very costly. For example all incidences of values were stored as *doubles* or real numbers, even though the majority of variables were binary variables that could be stored as a *char*. To improve this we introduced a new category in the data dictionary which specified what type of variable to be used, so that binary variables only took up a fraction of the space required to store a *double*. Also we stored real and categorical numbers as integers. Real numbers were converted to integers by multiplying by 1000 and retaining only the integer element of the number. Integers take half as much space as a double. We also conducted an audit of the entire data-structure, stripping out as much superfluous memory requirements as possible. This has been particularly important for allowing for much bigger datasets to be simulated on a laptop with limited RAM.

Something we have not explored yet is the further speed improvement that could be found by creating sub-sets of objects associated with each condition. At present, the model needs to scroll down through the whole dataset for each process. If conditions are updated dynamically, then the sub-groups where the condition is true can be updated dynamically, resulting in calculations only taking place on the subset. When the condition changes then, this updates the set of objects where the condition is true. Therefore in doing simulations, the model will only simulate over the set of objects eligible to be simulated.

At present all processes are run in series. In other words each process is completely simulated before moving on to the next process. This requires a data pass for each process. This is necessary for processes that are aligned as the decision about who makes a transition will depend upon all objects and not on individual objects. However some processes, such as transformations and unaligned processes, do not need to be done serially. Efficiencies could be gained by simulating these processes in parallel. For example, if age is simulated for an individual, then age squared and

age band could be calculated using age as input for each individual before continuing on to the next individual, cutting the number of data passes and improving the speed of the model. Other examples include the calculation of durations and lagged values of variables.

As always in microsimulation models, there is a trade-off between flexibility, complexity and performance. Parameterisation may sometimes result in enhanced complexity and thus reduced transparency and ease of use of the model. In the LIAM structure, this has been less of an issue as the parameterisation allows for the same code to be reused over and over without recoding, so to some extent improving the transparency. There are, however, some performance overheads noted elsewhere in the paper, where the degree of parameterisation and generalisation may increase the number of operations required and thus increase the time to run a simulation. This increase must be weighed up against the complexity of creating a dynamic model without an existing framework.

7. IMPLEMENTATIONS OF THE FRAMEWORK

In this section we describe a number of implementations of the LIAM framework. Thus far there have been four implementations of the model

- a. Life-cycle redistribution in the Irish Tax-Benefit system - Irish Dynamic Cohort Microsimulation Model
- b. Redistributive impact of Indirect Taxes in Europe - dynamic microsimulation of expenditures in the EUROMOD framework.
- c. Spatial Policy Analyses - Simulation Model of the Irish Local Economy (SMILE)
- d. Cross-national comparisons of the distributional impact of pensions and the incentive to retire - multi-country dynamic microsimulation model - EU 6th Framework project, Old Age Income Maintenance Policies (AIM).

Model (a) was the basis of the author's PhD and has been used to examine the life-course redistributive impact of the Irish Tax-Benefit System (O'Donoghue, 2002) and the redistributive impact of pension reform (O'Donoghue, 2005). The implementation was a single cohort model taking 1000 people aged 0 and simulating their entire life-history before linking with the EUROMOD tax-benefit model to simulate the tax-benefit system. A feedback loop was used to incorporate the impact of tax-benefit policy on labour supply decisions.

While model (a) utilised an external tax-benefit microsimulation model to provide tax-benefit simulations for use in the dynamic microsimulation model, model (b) takes inputs from the dynamic microsimulation framework into a static tax-benefit model. As part of the EU tax-benefit model EUROMOD, there was a desire to

examine the impact of indirect taxation on redistribution (O'Donoghue *et al.*, 2004). However, most of the databases used as inputs into the model did not contain expenditure information. The LIAM framework was used to simulate a system of equations simulating total expenditure and budget shares of twenty groups of goods on the basis of information contained in the income surveys used in the model. Indirect taxes were then simulated using the EUROMOD framework. This model used datasets of up to 50,000 households simulating indirect taxes for one fiscal year.

In recent years parallel microsimulation modelling has been used for geographical and spatial analysis (See Clarke, 1996 and Holm *et al.*, 1996). Since 2002, a team comprising the University of Leeds, the National University of Ireland Galway and Teagasc have been developing model (c), the Simulation Model of the Irish Local Economy (SMILE), using the LIAM framework with the principle objective of carrying out spatial analysis in Ireland (O'Donoghue *et al.*, 2005). Examples include modelling the impact of local area demographic changes on welfare, modelling the spatial impact of rural policy reforms, identifying agri-tourism hotspots. A future goal is to model the spatial behavioural impact of public infrastructure developments such as road building programs. The first component of the model is developed outside the framework, requiring the statistical matching of individual tabular local area census information with micro-level household data to produce the base dataset. This is done using a statistical matching algorithm. The LIAM framework is used to simulate typical dynamic microsimulation variables such as demographic and labour market variables. Particular advancements from this model include regional labour markets, micro-farm level production functions and spatial behavioural models. The model is currently under development. At present it is split into around 30 county models of about 70,000 persons each and simulates spatial-based policy at the local level.

The fourth implementation of LIAM, also in development, is being used to carry out cross-national comparisons of the distributional impact of pensions in a selection of countries in the EU (Belgium, Germany, Ireland, Italy and Sweden). In addition a comparative analysis will be carried using a semi-structural retirement decision module based upon discounted income and pension wealth streams. This model simulates over the 50 year horizon 2000-2050, using cross-sections of about 10-15,000 individuals.

8. CONCLUSIONS

While there have been substantial numbers of papers describing analyses carried out by models developed using LIAM, relatively little has been written on the technical development of the models. In this paper we have discussed some of

the methodological innovations underpinning the LIAM dynamic microsimulation framework.

The key feature of LIAM is the extensive use of parameterisation. Parameterisation of the main features of the dynamic model allows for the programme code which runs transitions, alignment and transformations to be reused for different purposes. When adding new variables to the model, alterations need to be made only in one place, in a parameter file, making the model easier to change and reducing the possibility of error. This aids model flexibility, as code does not need to be reprogrammed when parameters change. This in turn improves the durability of models developed using LIAM, as it allows new parameters to be included when better information becomes available. This is perhaps the most important feature of the LIAM, allowing the framework to be used to develop model applications for a wide variety of purposes. It also allows for ease of expansion as improved data and micro-behaviour become available. In addition, by defining the data structure outside the model, in a parameter file, model transparency and robustness is improved.

A second key feature of LIAM is modularisation. Six generic module types have been identified and developed, which between them are designed to cover the full range of model functionality required in a dynamic microsimulation model. Because these modules are fully parameterised, in LIAM one can declare a new module by simply taking an existing module as a template and changing the parameters as required. Similarly, because module run-order and type is parameterised, the model can handle any number of modules in any order without the need for extra programming, provided the functionality of the new module maps onto one of the existing generic module types. New modules can be added to a model without affecting its integrity as, in LIAM, all modules are designed to work independently of each other. This further adds to the robustness of the model. Also, by allowing the user to focus on small sections of code at time, modularisation improves the transparency of the model.

The third aspect of LIAM highlighted in this paper is its handling of alignment. As well reviewing the rationale for alignment, we have set out the circumstances in which the use of generic or event specific functions are most appropriate for the implementation of alignment. We have also set out in detail the process required to align aggregate outcomes to the proportion or number of objects in, or moving between, given states, including consideration of hierarchical (meso/macro) alignment.

The final aspect of LIAM highlighted in this paper is the range of efficiency improvements, in both speed and memory usage, that have been implemented as a result of lessons learnt whilst developing the framework. The main improvement has been move from the evaluation of functions

on a universal, periodic, basis, to an initial universal evaluation followed by reevaluation only for those objects to whom the function applies, and for whom values in relevant elements of their attribute set have changed. Secondary improvements have been linked to more efficient data storage, including the conversion of all data, including floating point data, to integer format.

We do not claim that LIAM is either the fastest or most efficient dynamic microsimulation model currently in use. But unfortunately the methodological underpinnings of many models remain undocumented or are set out only in in-house documentation. In this paper we have attempted to publicly document some of the issues that have arisen in the creation of the LIAM framework, with the hope that other model builders can learn from our experiences. Although the framework is not an attempt at writing a microsimulation programming language, it has allowed for a variety of different applications to be constructed without the need for extensive recoding. In addition, as we have shown, it has been possible to use this framework as a template for other dynamic models, because the model itself is entirely independent of data and behavioural equations to be used. To promote collaboration and further development, LIAM is available to researchers on request.

Acknowledgements

The authors gratefully acknowledge financial assistance from the Postgraduate Fellowship of the Economic and Social Research Institute, Dublin, the Irish Department of Agriculture Rural Stimulus Fund, Center for Research on Pensions and Welfare Policies (CERP), University of Turin, the NUIG Millennium fund, the Combat Poverty Research Fund and the *Targeted Socio-Economic Research* programme of the European Commission (CT97-3060), the AIM project financed under the 6th Framework Research Program of the European Commission and the IDARI project financed under the 5th Framework Research Program of the European Commission. We are very grateful for the assistance of Donal Kelly, who substantially improved the code of this model. We are also grateful for comments provided by Geert Bryon, Jane Falkingham and seminar participants in LSE, Nordic Microsimulation Workshop Copenhagen, AIM workshop Madrid and colleagues in Teagasc, the DWP and NUIG. The authors remain responsible for all remaining errors.

REFERENCES

- Bouffard N, Easther R, Johnson T Morrison R J and Vink J (2001) 'Matchmaker, matchmaker, make me a match', *Brazilian Electronic Journal of Economics*, 4(2).
- Caldwell S B (1996) 'Health, wealth, pensions and life paths: the CORSIM dynamic microsimulation model', in Harding A (Ed.) *Microsimulation and Public Policy*, Amsterdam: Elsevier, 505-522.
- Chénard D (2000) 'Individual alignment and group processing: an application to migration processes in DYNACAN', in Mitton L, H Sutherland and M Weeks (Eds.) *Microsimulation modelling for policy analysis: challenges and innovations*, Cambridge: Cambridge University Press, 238-247.
- Citro C F and Hanushek E A (Eds.) (1997) *Assessing Policies for Retirement Income: Needs for Data, Research, and Models*, Washington DC: National Research Council.
- Clarke G P (1996) (Ed.) *Microsimulation for Urban and Regional Policy Analysis*, London: Pion.
- Dobson A (1990) *Generalised Linear Models*. London: Chapman and Hall.
- Duncan A and Weeks M (2000) 'Simulating transitions using discrete choice models', in Mitton L, H Sutherland and M Weeks (Eds.), *Microsimulation Modelling for Policy Analysis: Challenges and Innovations*, Cambridge: Cambridge University Press, 292-308.
- Edwards S (2004) 'GENESIS: SAS based computing environment for dynamic microsimulation models', Mimeo, Department of Work and Pensions, London.
- Galler H (1997) 'Discrete-time and continuous-time approaches to dynamic microsimulation reconsidered', NATSEM Technical Paper 13, National Centre for Social and Economic Modelling, University of Canberra, Australia.
- Holm E, Lindgren U, Makila K and Malmberg G (1996) 'Simulating an entire nation', in Clarke G P (Ed.), *Microsimulation for Urban and Regional Policy Analysis*, London: Pion, 164-186.
- Immervoll H and O'Donoghue D (2001) 'Towards a multi-purpose framework for tax-benefit microsimulation: a discussion by reference to EUROMOD, a European tax-benefit model', *EUROMOD Working Paper*, EM2/01, Department of Applied Economics, University of Cambridge.
- Leombruni R and Richiardi M (2005) 'An agent-based microsimulation of labour force participation. Some results for Italy', mimeo, LABORatorio R. Revelli, University of Turin.
- Nolan B (1990) 'Inequity in the Financing and Delivery of Health Care in Ireland', ESRI Working Paper 16, Economic and Social Research Institute, Dublin.
- O'Donoghue C (2001a) 'Dynamic Microsimulation: A Survey', *Brazilian Electronic Journal of Economics*, 4(2).
- O'Donoghue C (2001b) 'Redistribution in the Irish Tax-Benefit System', Unpublished PhD thesis, London School of Economics, UK.
- O'Donoghue C (2002) 'Redistribution over the lifetime in the Irish tax-benefit system: an application of a prototype dynamic microsimulation model for Ireland', *Economic and Social Review*, 32(3), 191-216.
- O'Donoghue C (2005) 'Assessing the impact of pensions policy reform in Ireland: the case of increasing the pension age' in Fornero E and P Sestito (Eds.) *Pension Systems: Beyond Mandatory Retirement*, Cheltenham: Edward Elgar, 107-143.

O'Donoghue C, Baldini M and Mantovani D (2004) 'Modelling the redistributive impact of indirect taxes in Europe: an application of EUROMOD', *EUROMOD Working Paper no. 7/01*, Department of Applied Economics, University of Cambridge.

O'Donoghue C, Ballas D, Clarke G and Lennon J (2005) 'Location choice decisions in Ireland', paper presented to the conference Modelling Urban Social Demographics, University of Surrey, Guildford.

Orcutt G, Merz J and Quinke H (Eds.) (1986) *Microanalytic Simulation Models to Support Social and Financial Policy*, Amsterdam: North-Holland.

Pudney S E (1992) 'Dynamic simulation of pensioner's incomes: methodological issues and a model design for Great Britain', Dept. of Applied Economics Discussion paper, MSPMU 9201, University of Cambridge.

Sauerbier T (2002) 'UMDBS - a new tool for dynamic microsimulation', *Journal of Artificial Societies and Social Simulation*, 5(2).

Wolfson M and Rowe G (1998) 'LifePaths - toward an integrated microanalytic framework for socio-economic statistics', paper presented to the 26th General Conference of the International Association for Research in Income and Wealth, Cambridge, UK.

Appendix 1 Macro alignment in LIAM

In LIAM, macro alignment occurs as follows:

1. Specify alignment sheets, that need to be the same shape for each process (but macro can be a subset), as does the predictor

2. Create a temporary set of alignment structures of type talign (= n+1, where n is the number of processes to be macro aligned - structure (0) is to store the macro level)
3. For each sub process, run through conditions and count the number of people (level.nPer) who meet conditions who are in each alignment cell (we don't store predicted probability at this point as we don't know it - maybe simply assign zero and use the existing code)
4. For macro process, do the same
5. Multiply the cell p times the number in cell N = np, the number to be selected in cell
6. If the sub-processes are more disaggregated than the macro level, collapse to the lower level by summing N over the higher level (ie across education levels)
7. Now we have the N's for the 2 dimensional table for macro and each sub-process. Sum over sub-processes to get expected overall N_t and compare with the Macro N_m
8. To adjust multiply the highest level of the macro sheet (in this case level 2) in each of the sub-process by N_m/N_t
9. Backup original Alignment numbers (to be used in the following year)
10. Store new Alignment totals in the sub-process alignment structures