Full Paper for ISGC 2010 submitted for peer review

GENESIS Social Simulation Modelling Progress

Andy Turner
CCG, School of Geography, University of Leeds, UK
http://www.geog.leeds.ac.uk/people/a.turner

A.G.D.Turner@leeds.ac.uk

1. Introduction

GENESIS[1] is a UK project funded by its Economic and Social Research Council through the National Centre for e-Social Science research node program. Some GENESIS work aims to develop simulation models that represent individual humans and their organisations and how they change their location and influence over time. For this I am developing two models that operate at different temporal resolutions over different time scales. The models are implemented as open source in the Java programming language[2] and share a common set of packages, classes and methods. One model works with time steps of seconds and runs for days, which I call a *traffic model*. The other works with time steps of a day and runs for years, which I call a *demographic model*. More detailed descriptions of these models and their development is described in Sections 2 and 3 respectively.

My GENESIS dynamic simulation models were developed from first principles, aiming to be as simple or basic as possible in the first instance and develop by adding new things to improve realism step by step. The development has formed an iterative cycle and I move from one model back to the next implementing similar enhancements.

At first, the basic models were developed and tested for small populations and regions and these were run to provide some graphical outputs. A basic demographic model was reasonably straightforward to develop, whereas a basic traffic model was more challenging and was developed in several steps.

Having developed and run basic models, the next major steps were:

1. to make it easy for others to configure and run the models; and,
2. to scale up, so that the models could be applied to reasonable large city sized regions with millions of Agents.

With Alex Voss[3], I began to develop a Repast[4] implementation of the basic demographic model at the International Symposium for Grid Computing 2009 and we presented this at specially arranged meetings at the Academia Sinica Centre for Survey Research at that time[5]. The aim of this was to encourage collaboration and get others involved in our social simulation efforts. This pioneering work was progressed by Alex who with David Fergusson[6] developed materials for and organised a Social Simulation Tutorial which we ran together at the International Conference on e-Social Science in Cologne, Germany, in July 2009[7]. An updated Social Simulation Tutorial developed mainly by Alex, but also by Rob Procter[8] and myself was run at the International Symposium for Grid Computing 2010[9]. The use of the agent simulation framework Repast has made it easier for non-programmers to engage with a more graphical user interface.

The crux of this paper is on scaling up the models, a computational challenge which can only be partially addressed by using bigger machines. To run city sized simulations without finding a massive shared memory machine, what was done was to develop the model source code so that much of the data for a simulation can be stored and swapped with slower access persistent memory held in a database or directly as files on the file system. To achieve this: some utility methods were created to test the amount of available memory and swap data as appropriate; and, wrapper like methods were implemented for each method which allowed for the handling of Java OutOfMemoryErrors if they were encountered. These methods are outlined in Section 4 where some other ideas for scaling up the social simulation models are outlined.

At present the visual outputs of the simulation models I have created are basic. They were initially only producible as the model runs. However, the implementation has now progressed so that visualisation can be done independently although clearly it is not possible to visualise data for time periods that a simulation has yet to reach. In order to decouple the visualisation, the model needs to output data from which these visualisations can be generated. As a simulation model becomes more computationally expensive to run, it becomes more useful to store data to generate visualisations from, than to produce new visualisations by re-running the simulation model. Also it is useful to be able checkpoint or freeze-dry a simulation such that it can be restarted from a specified point rather than having to run everything from the start to get to that specific point. Once there is the ability to restart a simulation model, it is only a small effort to develop the source code to allow data outputs from one simulation to be input to other simulations. In Section 6 some ways to develop the models in the next few development iterations are considered. Section 7 provides a short summary and concluding remarks.

2. Developing a traffic model

To begin with traffic simulation was considered from first principles. To model peoples movements on an individual level requires a way to store the location of each agent. As a first step, agents were positioned in a confined region on a Euclidean 2D plane and made to move around this randomly by repositioning at each time tick. A maximum range for movement was used to constrain Agent movements. A basic visualisation was developed which marked Agent movements as lines on an image. Next the concept of a destinations was developed, so that rather than necessarily having a different destination at each time tick, an Agent might be assigned a destination beyond its maximum range for movement in a time tick. By initially clustering Agents origin and destination locations into various sub-regions, some interesting images could be created. However by studying the visualisations it was noticed that Agents rarely shared the same routes which is something that tends to happen in organised societies. It was understood that the way to encourage route sharing was by some form of Agent-Environment and/or Agent-Agent interaction. Developing a model such that there is a small benefit to each Agent of using an existing route, manifest by an improved capacity and flow of a route in the Environment from its use, and/or a preference for Agents to combine and share journeys should result in route sharing emerging. However, modelling the emergence of roads and encouraging route sharing became less of a priority compared to constrain movement so that it only took place on an existing network:

For a contemporary UK city model, the movements considered most important are commuting journeys whereby most Agents are assigned to move from home to work and back again in a daily cycle. Initially, to keep the model simple, all Agents were given the same time (shift) to be at the work location. As the distance to the work locations was variable, their journeys were not to start necessarily at the same time. Some data about commuting journeys in the UK has been captured in its Census data, particularly the Special Transport Statistics as described and can be accessed via the UK Centre for Interaction Data Estimation and Research (CIDER)[10]. The Special Transport Statistics data is incomplete, but crucially it provides flow information about where people live and work at a reasonably high level of spatial detail. There are details of these commuting journey flows (breakdowns by mode of transport and job type variables), but there is no direct linkage to data about the usual times of work of people represented in the flows. In reality, the times and locations people work may vary considerably, but in my experience, there is a clear general pattern of rush hours to and from work at the beginning and end of a day around 8am and 5pm.

At this stage it was clear that another two types of model were being developed: Those that were entirely synthetically made up; and, those which were to be seeded from available data. Also for those models seeded with available data there was a further distinction: those seeded from publicly available data; and, those seeded (at least in part) from data which is not publicly available and is more use restricted.

Various synthetic city models were created focussing on modelling and visualising commuting journeys. A motivation factor to concentrate on developing constrained movement on an existing network was that visualisation of some completely made up synthetic cities seemed more realistic (compared with models seeded with census data). The most appealing simulations were from models where Agent home locations were unique, but clustered into residential zones and Agent work locations were shared and clustered into business districts. Also at this stage of development, visualisation of lines on blank background was becoming too limited and the images were enhanced using a background depicting population density. Population density raster grids were output for each time tick and aggregated population density raster grids were also output. Compiling the outputs into animations served a key dual purpose of demonstrating progress to project collaborators and in helping debug and develop the models.

Now, consider how to develop a model so that Agents try to arrive at work on time for their shift. An estimate of the time needed for their journey can be precomputed based on distance, or it can be learned. Learning it by running the model is perhaps easiest, but it is also potentially useful for other journeys an Agent might make to enable them to estimate journey times. The learning was initially implemented as follows: On Day 1 of the simulation, each Agent set off to work at the time they were due to be at work (late). After they arrived at work, they recorded their lateness. On Day 2 of the simulation each agent set off at a time - however much earlier they were late on Day 1. Without any constraints on traffic, each Agent made it to work on time on Day 2. Visualisations of these models although interesting, were still hard to relate to reality. Agents rarely shared routes and there was no capacity like constraints that are characteristic of traffic.

Two things to do to constrain Agent movement were considered. One was to route agents along known transport infrastructure and apply capacity constraints on this infrastructure. Another involved restricting Agent movement to a high resolution regular network and encouraging a transport infrastructure to emerge/evolve by appropriately modelling Agents-Environment and Agent-Agent interaction. To start with I developed code to restricting Agent movements to a high resolution regular network, but before I began to develop code to model the evolution of transport infrastructure through use, I was guided by my GENESIS collaborators to focus on developing constrained use of existing transport infrastructure. For this two options were considered:

1. Use the OpenStreetMap[11] data which is publicly available for the entire world; and
2. Use UK Ordnance Survey[12] data available under an academic license via Edina[13].

Option 1 was made more attractive by the existence of the TravelingSalesman routing API[14][15]. TravelingSalesman provided a means to route Agents via the OpenStreetMap road network. So, I started with this and hit a problem that meant I needed to scale down the simulations used to test the models. The issue was that Agents needed reference to the route they had planned for a journey and this could be massive compared with simply knowing a destination location. To get results I needed to scale back from a city model to something for a small town. I was diverted into developing code so as to handle the data about Agents better and indeed allow for collections of Agents to be moved or swapped to and from fast access memory and persistent memory. This memory handling is really the crux of the paper and is detailed in Section 4 after Section 3 which describes the GENESIS demographic model which I used to help develop the memory handling code.

3. GENESIS Demographic Model

There are a number of dynamic demographic simulation models being developed in GENESIS. This section details one that works with time steps of a day. My colleagues Belinda Wu[16] and Mark Birkin[17] are developing others which work with time steps of a year[18]. I decided to develop a model with daily time steps because many things happen day to day, people migrate (i.e move usual residence or home location), get married, are born or die and celebrate specific days. Anniversaries and religious festivals and other things that draw groups of people together can be organised into daily activity calendars. There can be general activity calendars and each person can have their own. Much human activity can be modelled in daily chunks and the levels of these activities is constrained. For instance, the number of people moving home (migrating) is constrained by the capacity and availability of the services for this activity.

In the long term, it is hoped that explicit linkages between the traffic and demographic models can be made and they can interact becoming more of a single model. This is another reason for choosing a daily time step for demographic modelling in GENESIS as various linkages between the models can be envisaged. For example, the journey that many take to hospital for a birth, the timing of the birth can be given by the demographic model simulation, and the detailed scheduling and journey for the individuals involved can be determined in the traffic model simulation. A slightly more complex feedback, going the other way, from the traffic model simulation to the demographic simulation model, is that if Agents experience an inefficient commute to work, they may have an increased likelihood of changing their working practices (work times - shifts), their work or their home location. One further reason for a daily time step is computational. There are a similar number of time steps running a model that works with time steps of a day and runs for years as there for one that works with time steps of seconds and runs for days.

Development of the demographic model focussed initially on the processes of death and birth, and this is described in the remainder of this section:

There is a common class of People Agents in the model representing people for which there are two main extended Classes. Male for representing males and Female for representing females. At each time step all People have a chance or probability of dying dependent on their age and gender. The age and gender specific mortality probabilities were fixed in the first set of models. In a simulation, the next number is a pseudo-random sequence is obtained and used to determine if each Person dies at each time step . If they die, they are no longer part of the simulation and in the first set of models their data is never needed again and they are simply set to null and their memory resources are re-used.

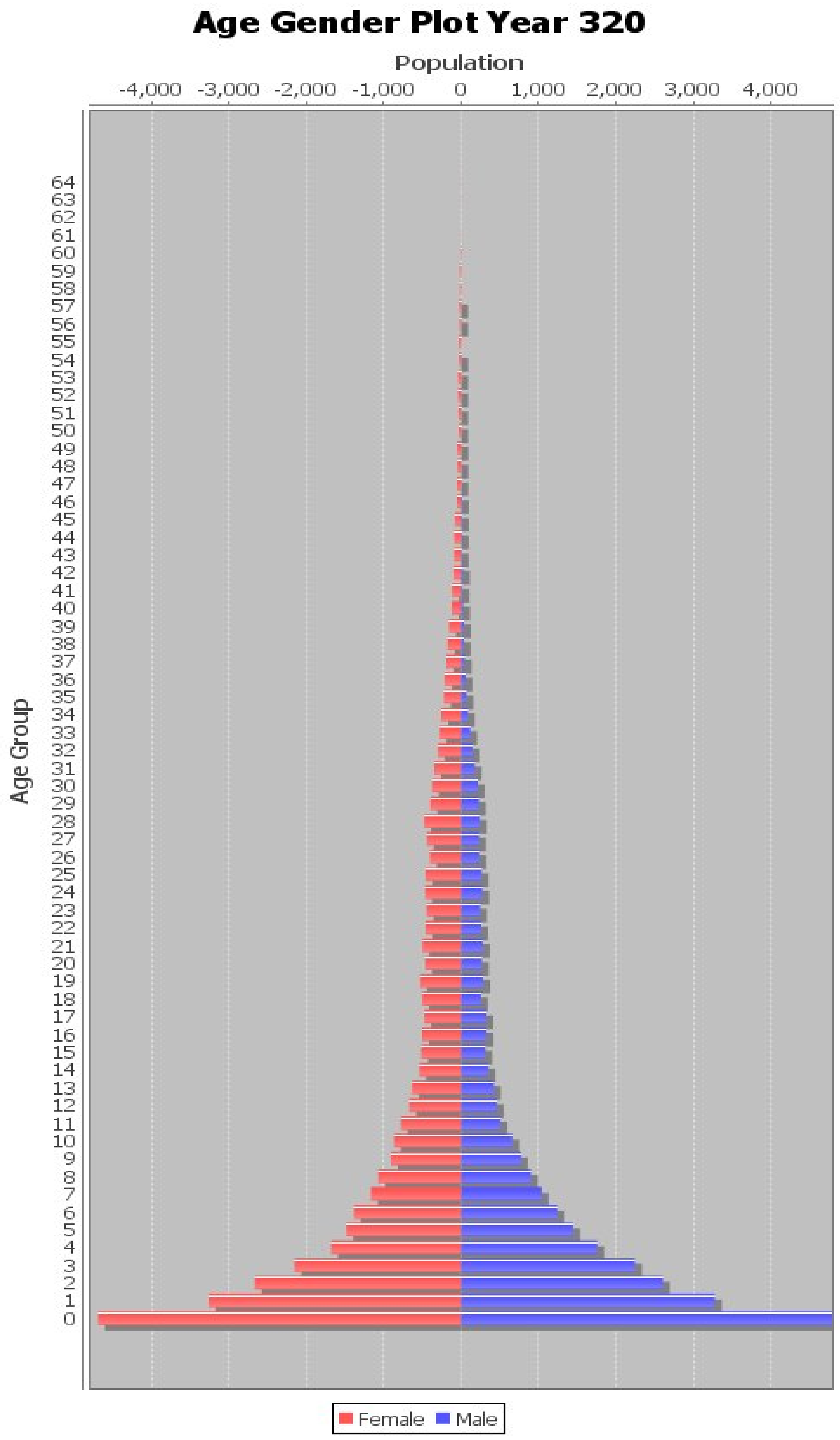
For Female agents additional processing is done each time step. Firstly pregnancy (conception) is determined. For each Female that is not yet pregnant, their fertility probability is obtained. Like mortality probability, fertility probability is age specific. Of course, in reality it is more complex than that! It will depend on other characteristics of the female (such as, day in a menstruation cycle, number of existing children, whether using a birth control deliberately reducing fertility, whether they are in good health) and whether there is a potential father around. But the basic model ignores this complexity and a Female Agent is determined as pregnant again using a comparison of their fertility probability with the next number in a pseudo-random sequence.

Next, all pregnant Female agents are iterated over to determine if any miscarry. In this basic model the miscarriage probability is constant.

The final evaluation for each time step is birth. In this basic model, the due date for a birth is determined at conception and it happens exactly 266 days after with no variation. At each time step, each pregnant Female is examined and if they are due then a birth occurs. This is something that is easily optimised as checking every tick should only be necessary for those close to being due.

So, with death and conception and miscarriage and birth, there is a basic model, arguably the most basic demographic model. It allows for the generation of Age Gender Plots showing counts of Male and Female Agents. These are interesting graphical outputs which demographers commonly study and I automated a procedure for developing these using the J-FreeChart library[18]. Figure 1 shows a basic Age by Gender Plot output image. Female populations are depicted in red on the left. Male populations in blue on the right. Each coloured bar represents the number of people of a certain age. The ages bands are shown from oldest at the top, to youngest at the bottom.

Figure 1: A Basic Age by Gender Plot Output From A GENESIS Demographic Simulation Model
.



Much can be done to make the basic model more realistic, and more fitting in terms of local circumstances. For instance, to model population in contemporary mainland China where a one child policy is in place[19], Female fertility will need to be modified so that if a Female already has one living child, they become much less likely to have another. Furthermore, the probability may depend upon the gender of the existing child. For the basic model, all the probabilities used can be modified to create different shapes and types of Age Gender Plot. I modified the default probabilities so that for a specific random seed of Java random number generator and from a simple seeded population of 100 15 to 20 years old People Agents, a gradually increasing populations was obtained. This could then be allowed to run for many hundreds indeed thousands of years. I could scale up the simulation reasonably easily by starting with a bigger seed population to test and develop memory handling code. This allowed for the examination of scaling properties of the model leading to the development of algorithmic optimisations.

4. Scaling up

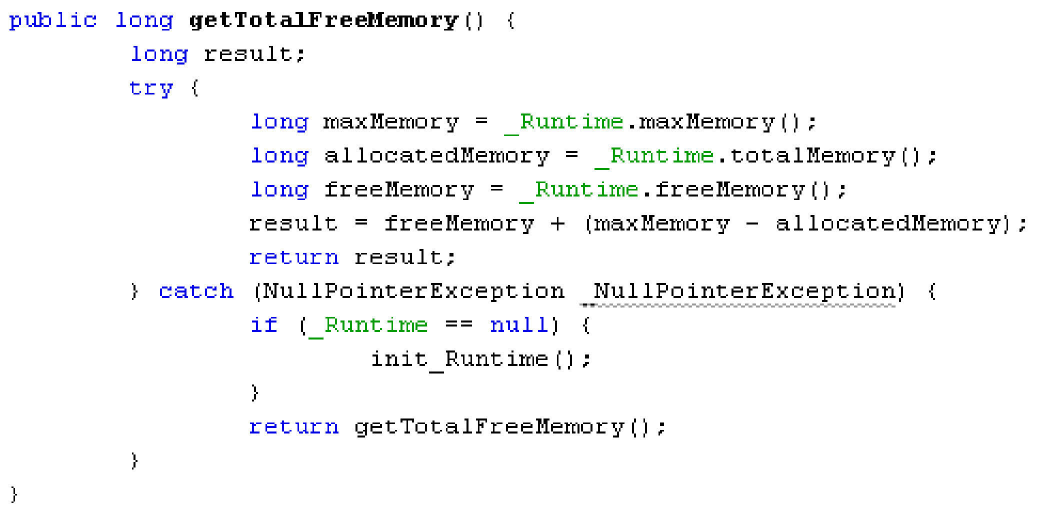
There are various ways to scale up to run larger simulations. An obvious solution is to use more computational resource. For example, use a machine with a bigger available memory (at this time this required a larger than 32 bit memory addressing system) that can be used by the Java Virtual Machine. Although perhaps an easy solution, bigger machines cost money to buy and they are generally harder to access. Part of my strategy was to ensure the simulation models could run on machines demanding less than 2 Gigabytes of fast access memory, but using larger amounts of slower access persistent disk memory as required. Another part of my strategy which is yet to be implemented involves multi-threading and parallelising the models. This latter part would help with scaling by increasing the speed at which computation is performed. This could be key to further scaling as swapping data to and from disk and fast access memory is computationally expensive.

The difficulty with memory management is knowing which Objects are needed and in what order. Assuming we know that the total volume of data for a simulation is more than can be stored in the fast access memory and yet more data is to be created. It is useful to have a method that returns us the amount of available fast access memory and to have methods which can serialise Objects in fast access memory and store them on disk. Metadata objects of fixed size can be used to determine the size of any Object to be loaded from disk to fast access memory. With enough available memory to do this a program can evaluate whether some other data need to be swapped from fast access memory to disk before the Object can be loaded. An arguably more simple way provided by Java is to simply try to load the Object that is required and if the program runs out of available memory, an OutOfMemoryError is thrown. With a try{} catch{} statement this error can be caught and handled. Handling OutOfMemoryErrors is at the core of what I have implemented. Another part of the implementation is to call a method which tries to keep a reasonable sized memory buffer available by swapping data to disk in advance to reduce the frequency of OutOfMemoryErrors handling.

Making a program robust and more capable by enabling it to handle OutOfMemoryErrors is a good exercise and has rewards. It is good exercise as it requires all the used methods to be revisited and the code generally benefits from this and additionally more testing code tends to be developed at the same time. The rewards are that new results are produced for larger problem sizes. At least, that is the case if the models have sufficient time to run. I have practised exercise before with a library called Grids[20] which I originally developed supported by a number of European Commission funded projects and which is a core library which I use for the developing the traffic model. Indeed I first presented about this way of memory handling at the FOSS4G 2006 conference in Lousanne Switzerland[21].

In the rest of this section, the key Java source code developed for handling OutOfMemoryErrors is presented. The colours and layout are not important, they just help to read and understand the code.

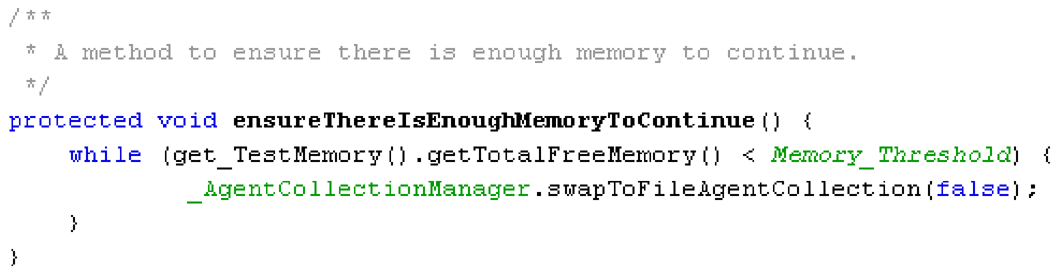
The method called getTotalFreeMemory() is shown below:



```
public long getTotalFreeMemory() {
        long result;
        try {
                long maxMemory = _Runtime.maxMemory();
                long allocatedMemory = _Runtime.totalMemory();
                long freeMemory = _Runtime.freeMemory();
                result = freeMemory + (maxMemory - allocatedMemory);
                return result;
        } catch (NullPointerException _NullPointerException) {
                if (_Runtime == null) {
                        init_Runtime();
                }
                return getTotalFreeMemory();
        }
}
```

In this method: _Runtime is a Java Runtime class instance that is initialised by the init_Runtime() method. It should be a field available to the class holding this method. The method is in a class called TestMemory in another code base I developed. To remove dependency on this other code base, or have common dependency to a utility library this class is probably best moved. Anyway, the getTotalFreeMemory() method essentially returns the number of bytes that are not yet allocated.

The method ensureThereIsEnoughMemoryToContinue() is shown below:



```
/**
 * A method to ensure there is enough memory to continue.
 */
protected void ensureThereIsEnoughMemoryToContinue() {
    while (get_TestMemory().getTotalFreeMemory() < Memory_Threshold) {
            _AgentCollectionManager.swapToFileAgentCollection(false);
    }
}
```

In this method: get_TestMemory() returns an instance of the TestMemory class (which I know as a field available to this class due to the _ notation) and the getTotalFreeMemory() method is called. The output of this is compared (using less than) to a static threshold value Memory_Threshold. While it is less than, the method swapToFileAgentCollection(boolean) is called from _AgentCollectionManager. The field _AgentCollectionManager is an instance of the AgentCollectionManager class which is available to the class containing this method. In my GENESIS code[22] version 0.1, the method is in a class called Environment. There is just one instance of Environment in any Simulation run and it is accessible to all classes. It contains all of the methods that are called to try to ensure there is enough memory to continue (of which there are many more complex types, which call different types of swapping methods of the AgentCollectionManager class).

The method swapToFileAgentCollection(boolean) is shown below:

```
/**
 * Swaps to file any AgentCollection.
 * @param handleOutOfMemoryError If true then OutOfMemoryErrors are caught,
 * swap operations are initiated, then the method is re-called. If false
 * then OutOfMemoryErrors are caught and thrown.
 */
public void swapToFileAgentCollection(
        boolean handleOutOfMemoryError) {
    try {
        swapToFileAgentCollection();
        _Environment.ensureThereIsEnoughMemoryToContinue(
                handleOutOfMemoryError);
    } catch (OutOfMemoryError a_OutOfMemoryError) {
        if (handleOutOfMemoryError) {
            clear_MemoryReserve();
            try {
                swapToFileDataAny();
                init_MemoryReserve(false);
                _Environment.ensureThereIsEnoughMemoryToContinue(false);
            } catch (OutOfMemoryError b_OutOfMemoryError) {
                b_OutOfMemoryError.printStackTrace();
                System.exit(
                        OutOfMemoryErrorHandler.OutOfMemoryErrorExitStatus);
            }
        } else {
            throw a_OutOfMemoryError;
        }
    }
}
```

In this method: a boolean true or false variable handleOutOfMemoryError is passed in; _Environment is field which is the Environment instance (accessible to all classes). The method contents is wrapped like all public methods in a try{} catch{} statement. In the try{} part, the method swapToFileAgentCollection() is called and following this, there is a call to an the ensureThereIsEnoughMemoryToContinue(boolean) method in _Environment. If this try{} part executes without throwing an OutOfMemoryError then the program continues, otherwise the program catches the OutOfMemoryError. If this occurs, handleOutOfMemoryError is tested as to whether it is true or false. If it is true then the program proceeds by calling the clear_MemoryReserve() method otherwise it throws the OutOfMemoryError it caught. The method clear_MemoryReserve() essentially releases a memory reserve reserved previously. This essentially is to provide enough working space to swap some data do disk. This method continues with a further try{} catch{} statement. In the try{} part it calls swapToFileDataAny() which is a method that would swap any data that _Environment is aware of if no AgentCollection were available to be swapped, else it would just swap an available AgentCollection. Next, the method init_MemoryReserve(boolean) is called, which initialises a memory reserve as was cleared by clear_MemoryReserve(). Finally it calls ensureThereIsEnoughMemoryToContinue(boolean) which is a slightly more convoluted form of ensureThereIsEnoughMemoryToContinue() as shown above.

The method swapToFileAgentCollection() is shown below:

```
/**
 * Swaps to file any AgentCollection. If there are none to be swapped this
 * calls on _Environment to swap to file other data.
 */
protected void swapToFileAgentCollection() {
    Iterator a_Iterator = this._AgentCollection_HashMap.keySet().iterator();
    Long a_AgentCollection_ID = null;
    AgentCollection a_AgentCollection;
    if (a_Iterator.hasNext() == false){
        System.out.println(
                "Warning: No AgentCollections to swap in " +
                this.getClass().getName() +
                ".swapToFileAgentCollection()");
        _Environment.swapToFileData();
    }
    while (a_Iterator.hasNext()) {
        a_AgentCollection_ID = (Long) a_Iterator.next();
        Object value = this._AgentCollection_HashMap.get(
                a_AgentCollection_ID);
        if (value != null) {
            a_AgentCollection = (AgentCollection) value;
            //a_AgentCollection.swapToFileAgents();
            a_AgentCollection.write();
            a_AgentCollection = null;
            break;
        }
    }
    this._AgentCollection_HashMap.remove(a_AgentCollection_ID);
}
```

In this method: an Iterator a_Iterator for iterating over all the AgentCollections stored in the AgentCollection HashMap collection _AgentCollection_HashMap is obtained. The _AgentCollection_HashMap is a field available to the AgentCollectionManager class (the class containing this method). The typed HashMap _AgentCollection_HashMap is a collection containing all the AgentCollections for the simulation stored with keys being the numerical identifiers of these AgentCollections. Now, all the Agents in a simulation are stored in AgentCollections, these are HashSet collections of Agents. The number of Agents in a typical AgentCollection is a configuration parameter for the model. Essentially AgentCollections are one of the main data Objects swapped to and from fast access memory and disk . The swapping is not done on the individual Agent level as this is too inefficient in terms of input and output (IO). The IO overhead caused by opening and closing files for writing is significantly reduced by having Agents in serialisable collections. To return to the swapToFileAgentCollection() method, if there are no AgentCollections in fast access memory that can be swapped to disk, then the _Environment.swapToFileData() method is called, in which other non Agent related data is swapped. However, if there are AgentCollections in fast access memory that can be swapped to disk then the first AgentCollection returned by a_Iterator (a_AgentCollection) is written to a file via the write() method and set to null. When set to null, the resource used by a_AgentCollection are available to be reused via garbage collection.

Now, let me explain the file structure for Agent files and AgentCollections. The file structure is such that given a numerical identifier for an Agent or an AgentCollection, the location of the file storing the serialised Object can be calculated. In other words the location of the files are implicit given some configuration parameters. These include: the maximum number of Agents the simulation is expected to cope with (_MaximumNumberOfAgents); the maximum number of Agents to be stored in an AgentCollection (_MaximumNumberOfAgentsPerAgentCollection); and, the maximum number of Objects to store per file or directory (_MaximumNumberOfObjectsPerDirectory). With most operating systems there is a maximum number of files it is sensible to store in a directory. Also, there is a maximum length of filename including the directory path it can cope with. Using MaximumNumberOfObjectsPerDirectory set to 100 it is possibly to store a vast number of files in a directory structure with a depth of say 6 directories, more files indeed than there are people believed to be alive on the planet Earth at the present time(ref http://en.wikipedia.org/wiki/World_population):

1
100 = 100 * 1
10000 = 100 * 100
1000000 = 100 * 10000
100000000 = 100 * 1000000
10000000000 = 100 * 100000000
1000000000000 = 100 * 10000000000

So with _MaximumNumberOfAgents set to 1000000000000. There are that many directories 6 directories below a start directory. Files for an Agent with ID 0 are stored in the following directory: Agents/0000000000-0000000099/00000000-00000099/000000-000099/0000-0099/00-99/0/

Similarly AgentCollections can be stored in this way. The general AgentCollections contain sequential Agent records, so given an Agent numerical identifier the AgentCollection that should contain it can be identified and if this does not reside in fast access memory collection Environment._AgentCollection_HashMap, then an attempt can be made to load it from file. The method AgentCollectionManager.getAgent(Long, boolean) can then be safely used to get an Agent from its numerical ID.

In attempting to get an Agent care is needed not to iteratively swap to file the AgentCollection containing it only to attempt to swap it back into fast access memory again (its size does not change). So there are methods as described above which have as argument an AgentCollection. These will swap to file an AgentCollection except that which is known to be wanted. This prevents the program getting stuck in an infinite loop of swapping an Object. There can be an increasing level of sophistication in the memory handling so that a collection of AgentCollections instead of simply an AgentCollection not to be swapped. This can be useful in calculations where data about multiple Agents is required and these are not all referenced from a common AgentCollection.

Whether it is faster to calculate if memory is going to run out and prevent it, or deal with it if it happens depends on the context. Code that can deal with OutOfMemoryErrors if they are thrown is inherently more robust, but without detailing what should be swapped out this can be so inefficient that results cannot be attained in a reasonable time frame. So, much work needs to be done to consider what to swap out. Also, depending on the context, it might be appropriate to ensure there is a sufficiently large amount of unallocated and available memory before continuing. Much optimisation has been implemented in version 0.1 of my GENESIS Java code, but much more optimisation is probably necessary to get spectacular city simulation results. Rather than detail any more intricate methods for memory handling that have been implemented or present any more code in this paper I will draw to a conclusion in Section 6 after a brief consideration of further development in Section 5 considers. Those that want to study my code it is openly available on-line[22].

5. Further Development

The GENESIS social simulation code I have started to develop can be developed in lots of ways. Traffic and demographic simulation models can be developed for lots of different parts of the world seeded with data about local population and the environment.

The next step for enhancing traffic models is to be able to constrain traffic to carrying capacities of infrastructure and allow for queueing of traffic. To do this I am going to work on an example area and identify the network links with the heaviest traffic load and gradually constrain traffic there forming the first ever queue. Once this is done I want to produce a set of results for various traffic constraints and identify and compare locations where it is congested in the simulation and what is known about traffic in reality. Next, for commute journeys I want to allow agents to try alternative routes and maybe also allow for dynamic re-routing around queues. I want to incorporate more journeys including journeys to school and to retail and places of entertainment. For this in the UK I hope to make use of the linked data being made available via the data.gov.uk initiative[23].

The next step for enhancing demographic model is to explicitly model Male parentage and develop a process for partnering parents. This I plan to do at the same time as making this model spatial. People will develop collections of other Agents that they have been close to and these will more likely be those that are partnered. Once this is done the processes of migration and change in health status will be worked on.

Linking the traffic and demographic models is still some way off!

In terms of computation, one of the key challenges is to parallelise the models to exploit distributed resources. This is non-trivial and there are various options that need to be explored for partitioning the models. Similarly, with the multi-core nature of many compute resources, threading the processing should be considered not least in terms of memory management. The use of Virtual File System (VFS) technology needs to be considered. Have a common look up to data about a simulation from distributed resources running parts of a simulation will be extremely useful, but optimisation issues abound in configuring a VFS.

Maybe I also need to step back and look at other technology for helping with memory handling including databases and tools like Terracotta[24] and maybe also using a different language entirely such as C++[25], or Erlang[26].

Along side the GENESIS project, I am working on the NeISS[27] project which is funded by the UK Joint Information Systems Committee[28]. NeISS is developing an e-Infrastructure to support social simulation and so hopefully the traffic and demographic models I developed in GENESIS will be part of this and hopefully so will a community of users. The e-Infrastructure should make the models more easy to use by allowing for computational resource to be tapped and the results of running the models easier to archive and reproduce. The idea is to make it easy for others to modify and run the simulation models and produce outputs they are interested in.

Perhaps key to the future development of these models is to build up a community of users to test and get involved in development and application of social simulation models.

6. Summary and Concluding Remarks

Two different simulation models being developed in the GENESIS project have been outlined, a traffic model that operates on the scale of a days with a second time step, and a demographic model that operates over the time scale of a number of years with a daily time step. The long term goal is to link these models and develop a suit of general geographical social simulation models and use them to study social and environmental interactions. In the short term the hope is to develop the models sufficiently to be able to study: the relationships between migration, commuting and transport; service planning; retail location and property prices.

There are significant computational challenges in social simulation work. Running a model for the contemporary UK involves modelling and recording changes in the locations and other aspects of the state of Agents representing approximately 60 million real people. The entire history of an Agent in terms of the interactions it has with other Agents and the Environment could be massive and there is a challenge to resources the storage of these data.

The crux of this paper is the technical aspects of the modelling work, not the details of the models, but in the model development these have gone hand in hand and the paper hopefully provides a useful history of this development. In terms of the technical aspects, a generic data storage solution is outlined and its use in memory handling is described. The generic data storage solution is for storing data objects in files using a simple branching directory structure which allows for huge numbers of data objects to be stored in file locations that are implicit from a number used as the identifier for the data objects. Details of some memory handling in my GENESIS Java code are provided. For this Agents are collected into AgentCollections and these are swapped to and from fast access memory and disk. When fast access memory is short or runs out, AgentCollections in fast access memory are swapped to disk. When Agent data is required and the AgentCollections which contain them are not in fast access memory they are swapped in from disk. The details hopefully illustrate how this is done in what I claim to be a robust way.

In addition, this paper outlines further development of the GENESIS social simulation models described. Key to this is thought to be the establishment of a community of users to test and get involved in development and application of specific social simulation models. As a major step towards this, it is hoped that the NeISS project will provide a supporting e-Infrastructure to make the models more easy to use.

7. Acknowledgements

GENESIS is funded by the ESRC (RES-149-25-1078).

Thanks to: The international e-Research community; developers of open source software and Java; the University of Leeds[29] and especially the School of Geography[30] and Centre for Computational Geography[31] for help and support over the years.

Special thanks to: Paul Townend[32] for providing a correct getTotalFreeMemory() method which is now core to the memory handling efforts at the crux of this paper; and, Alex Voss[3] for taking an interest in social simulation and collaborating with me as a geographer.

8. References

[1] http://www.geog.leeds.ac.uk/people/a.turner/projects/GENESIS
[2] http://java.sun.com/
[3] http://www.cs.st-andrews.ac.uk/~avoss/
[4] http://repast.sourceforge.net/
[5] http://portal.ncess.ac.uk/access/wiki/site/%7Ea.g.d.turner%40leeds.ac.uk/isgc%202009.html
[6] http://www.nesc.ac.uk/nesc/staff/dfergusson.html

[7]
http://portal.ncess.ac.uk/access/wiki/site/socsim/social%20simulation%20tutorial%20at%205th%20international%20e-social%20science%20conference%202009.html
[8] http://www.merc.ac.uk/?q=Rob
[9] https://e-research.cs.st-andrews.ac.uk/site/bin/view/SocialSimulation/ISGC2010
[10] http://cider.census.ac.uk/
[11] http://www.openstreetmap.org/
[12] http://www.ordnancesurvey.co.uk/
[13] http://www.edina.ac.uk
[14] http://sourceforge.net/apps/mediawiki/travelingsales/
[15] http://sourceforge.net/projects/travelingsales/
[16] http://www.geog.leeds.ac.uk/people/b.wu
[16] http://www.geog.leeds.ac.uk/people/m.birkin
[17] Wu, B. M., Birkin, M. H. and Rees, P. H. (2008) A spatial microsimulation model with student agents, Journal of Computers, Environment and Urban Systems (32) pp. 440-453 DOI information: 10.1016/j.compenvurbsys. 2008.09.013
[18] http://www.jfree.org/jfreechart/
[19] http://en.wikipedia.org/wiki/One-child_policy
[20] http://www.geog.leeds.ac.uk/people/a.turner/src/andyt/java/grids/
[21] http://www.foss4g2006.org/contributionDisplay.py?contribId=21&sessionId=50&confId=1
[22] http://www.geog.leeds.ac.uk/people/a.turner/src/andyt/java/projects/GENESIS/
[23] https://www.data.gov.uk/
[24] http://www.terracotta.org/
[25] http://en.wikipedia.org/wiki/C%2B%2B
[26] http://www.erlang.org/
[27] http://www.geog.leeds.ac.uk/people/a.turner/projects/e-ISS
[28] http://www.jisc.ac.uk
[29] http://www.leeds.ac.uk
[30] http://www.geog.leeds.ac.uk
[31] http://www.ccg.leeds.ac.uk/
[32] https://www.comp.leeds.ac.uk/people/staff/scspmt.html