Res. Lett. Inf. Math. Sci., 2005, Vol.**7**, pp89-115
Available online at http://iims.massey.ac.nz/research/letters/

Sparse Cross-Products of Metadata in Scientific Simulation Management

H. A. James and K. A. Hawick
Computer Science
Institute of Information and Mathematical Sciences
Massey University, Albany
North Shore 102-904, Auckland, NEW ZEALAND
Email: {h.a.james,k.a.hawick}@massey.ac.nz

Managing scientific data is by no means a trivial task even in a single site environment with a small number of researchers involved. We discuss some issues concerned with posing well-specified experiments in terms of parameters or instrument settings and the metadata framework that arises from doing so. We are particularly interested in parallel computer simulation experiments, where very large quantities of warehouse-able data are involved. We consider SQL databases and other framework technologies for manipulating experimental data. Our framework manages the the outputs from parallel runs that arise from large cross-products of parameter combinations. Considerable useful experiment planning and analysis can be done with the sparse metadata without fully expanding the parameter cross-products. Extra value can be obtained from simulation output that can subsequently be data-mined. We have particular interests in running large scale Monte-Carlo physics model simulations. Finding ourselves overwhelmed by the problems of managing data and compute ¿resources, we have built a prototype tool using Java and MySQL that addresses these issues. We use this example to discuss type-space management and other fundamental ideas for implementing a laboratory information management system.

Keywords: parameter cross-products; metadata; data mining; data management.

1 Introduction

A common *modus operandi* for computational scientists running numerical simulations is shown in figure 1. A numerical model for the phenomena under study is constructed. The model is initialised and is spun-up into a realistic or at least representative state, whereupon measurements can be taken. Depending upon the model involved, measurements are made from static configurations, which may be stored separately, or measurements are made as part of the evolutionary process of taking the model configuration from one state to another. These configurations can usefully be warehoused for later mining.

Some important examples include numerical models for weather and climate study (2), where a set of model variables such as atmospheric temperature, pressure and wind velocity are time-evolved from one configuration to the next, to predict how real weather systems will develop. Climate study is similar except that the time scales simulated are much longer and the model granularity generally coarser. Other models in computational physics and engineering studies fall into this *general pattern of operation*. Some examples we consider in this paper (section 3) are Monte Carlo lattice models (3); stochastic network models (4); and artificial life growth models (5).

In running models that have even a few separate parameters it is necessary to manage the range and combinations of parameters. Sometimes the (computational) cost of running a model is small and it is feasible to throw away the configuration outputs and just preserve the few measurements that are made "during the run". It is sometimes however either too expensive to be able to justify

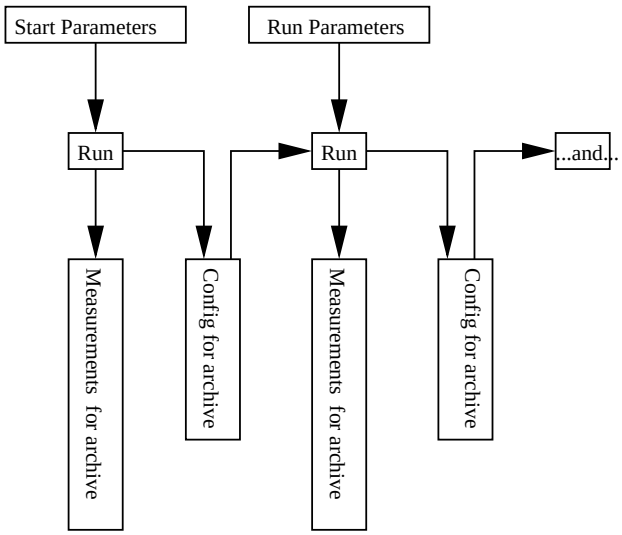


Figure 1: Flow of archive-able data from simulation runs in a common pattern for numerical experiments.

re-running models with the same parameters or in some cases it is important for legal or other operational reasons to keep all model output in an archive. Ideally the working computational scientist would like to afford the storage capacity to preserve the output from all past runs for possible future further analysis or for bootstrapping new model runs. There are tradeoff costs for this storage that must be weighed against the computational cost of regenerating data from runs. However, more than the physical cost of storage media there are "total cost of archiving" issues that need to be considered more deeply.

We explore one important complexity contribution to the total cost of archiving. Managing data that is simulated from codes that are continually evolving in a way which is forward-compatible is non trivial. The output formats are likely to differ slightly as the codes evolve. One step towards this is to consider the cross-products of all possible parameters values that could be used and to explore the implications of labelling experimental run outputs by these parameter values. Another step is to use a robust textual tagging of output values that offers some some defence against changing output formats through the use of generalised data extraction utilities and scripts.

Consider a simulation with just two parameters, as shown in figure 2. We can imagine the computational scientist steering this simulation does not systematically explore all of the possible parameter space, but rather poses some preliminary experiments that explore some combinations of parameter values that span some area of parameter space he expects have some "interesting" properties. Parameter one is represented by rows, and parameter two by columns. The scientist has carried out runs over the parts of parameter space shown and subsequently wishes to keep track of the "runs" obtained. Run data may be reused later and to have "archive value" it must be easily retrievable.

It has been our own experience that scientists often use some *ad hoc* approach to keeping track of model output data often involving file or directory names, perhaps with some `README` files to describe what experiments have been done. This is analogous to the online "lab notebook" easy to use for very small sets of runs but rapidly becomes hard to manage for even just a three parameter experiment. A common situation for our running stochastic simulations is that we rapidly accumulate new parameters as we develop the model. Two main ones accrue from the model itself, a further one is the sample number if we are averaging over many different stochastic simulation sequences, and a fourth is the random number generator seed if we wish to keep track of separate repeatable stochastic sequences. It is common to rely on metadata (10) in the form

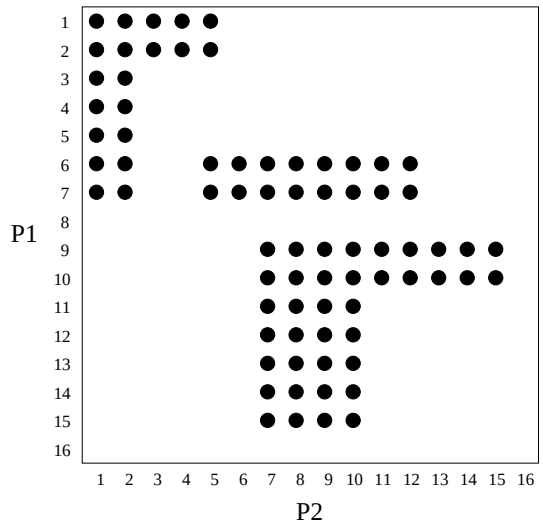


Figure 2: Sparse cross-product of a two-parameter system. A simple two-parameter system where various experiments have been carried out yielding non-null entries in the data base or matrix of possible values. The extreme values of the parameters and the discretisation scheme have set the bounds and size of the matrix table. The outer product need not and likely will not be fully populated.

of long file names or sometimes in terms of headers stored in the configuration files themselves. These can be opaque to the browsing scientist planning a follow-on experiment however.

In this paper we explore how relational database technology can be augmented with some simple conventions and easily produced tools to help manage more complex batches of runs and archives of data.

2 A Virtual Spreadsheet

Consider the parameters available in a multi-dimensional virtual spreadsheet. We have a hyper-space of parameter values, that is sparse as not all combinations of parameters are necessarily deemed worth running, and of course the actual values used over a range will be limited. We want the scientist to be able to pose the questions: **1)** What have I run already?; and **2)** What can I now run to pose a new research question, making best use of my existing data sets? We want the "housekeeping" operations for keeping track of the data to be as automated as possible, while still being compatible with the simulation programs already in use. We also want new programs to be able to easily access previous data.

One classic approach to these problems is to write simulation programs that "log" all sorts of extra information and measurements as well as the parameters actually used. The log files can then be scanned for relevant information and often new questions can be posed using old data that was generated before the question had been thought of. We have often adopted conventions for logging data to support this. For instance a textual tag is invented to describe the numerical measurement in question and the output value is prefixed with this tag in the log file. Standard `Unix` tools such as `grep`, `cut` and `paste` and other text manipulation programs or languages such as `perl` or `python` can readily be combined into scripts to extract relevant values from an archive of log files, in suitable form for plotting for example. Another common technique is to encode some parameter values in the filenames or in the names of the directories. This can aid casual browsing and experiment planning up to a point, but rapidly becomes cumbersome when many values are involved. How can the scientist view and visualise the sparse hyper-space of parameters, and hence

assess existing data availability and plan experiments?

Imagine a virtual spreadsheet that supports looking at any two axes from the hyper-dimensional data set. We would like the virtual spreadsheet tool to allow specification of these axes and to cope with what is a sparse set of data that may not be stored online or locally. The posed queries or high level commands can in principle be organised by the virtual spreadsheet tool into the necessary data retrieval requests to be scheduled and the resulting "plot" assembled. ideally the tool would have enough information to at least estimate how long satisfying the request will take, if it is not in "interactive time".

3 Application Examples

A recent real experiment, of interest to us, involved the simulation of the Ising model under Small-World conditions, as described in (3). The model uses Markov-Chain Monte Carlo (MCMC) (6) sampling of a data space. There are four major parameters that we wish to track, as well as at least two extra parameters that can be significant. These parameters are shown in table 1. Some of the parameters combine to give other properties, such as: the total number of points in the system is calculated by the number of lattice points per dimension raised to the power of the dimensionality. This number can then be multiplied by the probability of small world effects on the lattice to give the number of lattice points that need to be modified as per the effects. Recording the random number seed ensures that any experiment we perform is reproducible, an important consideration in any scientific investigation.

Parameter Description	**Primary?**	**Min**	**Max**	**Min Stride**
dimensionality of study (d)	yes	1	5	1
number of lattice points per dimension (L)	yes	1	1024	1
temperature of system (T)	yes	4.000	5.000	0.001
prob. of small world effects on lattice (p)	yes	0	1	0.000000001
number of update steps in data file (*steps*)	no	1	9999999999	1
random number generator seed (s)	no	0	9999999999	1
update method (u)	no	0	1	1

Table 1: An enumeration of the relevant parameters in our Small-World Ising model simulations. The abbreviation used to refer to each parameter is given in parenthesis after the parameter's description. Some parameters are marked as primary, signifying they are crucial quantities to keep track of; in practice we use these parameters to define the data table's primary key. For other parameters, such as *seed*, min, max and stride has no real meaning – as the random number generator will use a random seed for each run.

Our simulation was originally developed by choosing an initial T and p, and then refining the values as the simulation produced results. When we had identified a promising (or "interesting") area of the $T \times p \times L$ parameter space we performed a production run using the local supercomputer cluster. *Each* simulation took approximately 28 hours to complete 11 million update steps. By the end of the study we had produced approximately **0.5TB** of data. The problem was organising it and searching through it in an efficient manner. Our first approach was to write a series of `Unix` shell-scripts. We used long-named data files, including most of the relevant parameters as either part of the file name or at least as a comment-style metadata within the file. The scripts were a good *ad-hoc* solution - they enabled us to do basic searching and sorting on the data, and allowed us to execute our custom-written analysis programs on the data files. However they were not good at helping us identify "holes" in the data. It is particularly difficult to develop statistical analysis scripts that can cope with gaps in the data space.

Parallel supercomputers and clusters are often shared resources. Our is no exception, being

shared between computer scientists, computational chemists and computational biologists; it is not under our control. We observed that sometimes due to queue failures or lack of scratch space our simulations would either not start, crash upon startup, crash part-way through the simulation, or simply freeze - and not make any progress. The latter condition was later identified as a transient hard-drive problem in a small number of the compute nodes. To compound matters, we were using the resulting configurations from previous runs as a starting point for subsequent runs in the same parameter space. When a "gap" was identified this meant that before we could start the next iteration of that configuration the gap would have to be filled. Because of the number of jobs that were being created as the $T \times p \times L$ product, it was very difficult to spot a single job that had failed. A series of scripts written to help with this task proved to be large and quite unwieldy. An integrated approach allows us to harvest data from the distributed nodes directly into the database - or even to keep track of data that remains distributed amongst nodes' local disks. We believe these sorts of operational problems are quite common amongst computational scientists.

Our data analysis programs also require exploration of different parts of the parameter space we measured. For example, some require the first series of data collected for each of the different values of the parameters' cross-product, while others require all the series for a given value of the parameters' cross-product in a single sequence. It has been quite difficult maintaining the scripts necessary to extract all the required data in a portable manner and to cope with the exception handling routines for dealing with missing data.

We also run our own programs which simulate *Ad-hoc* network structures (4) and Artificial Life (ALife) (5) predator-prey models for studying species evolution. The *Ad-hoc* network simulation involved the variation of four distinct parameters: the number of radio transmitter sites in the simulation, the radius of perception of each transmitter, the type and degree to which the network was perturbed by Small-World effects, and the random number seed for configuration. A parameterised study was performed using a wide-range of parameter values; not all possible parameter values between the minimum and maximum were used. Searching the database of values means that it is not necessary to compute the complete parameter cross-product. We had the aim in the experiment of *steering* the experiment while it was in progress, through the modification of parameters, to investigate interesting phenomena.

The ALife simulation uses six independent parameters to represent predator and prey birth rates, their longevity, evolutionary periods and a random number seed for configuration information. This model was particularly interesting as we had parallelised it. The simulation was implemented as a parallel program using an optimal number of **42 processors**. Configurations had to be consistently stored to obtain meaningful statistics on the experimental runs. When analysing the results of this experiment we had to ensure that individual processors' output logs were safely archived to prevent data being lost, particularly should subsequent manual intervention be required for exception handling.

In summary, these simulations require a mix of integer and floating-point parameters, and have potentially large parameter spaces to explore. Gaps in the run-sequences are difficult to cope with for statistical analysis purposes without a good management framework.

4 Framework Architecture

Our prototype scientific data management framework is based around the use of a MySQL database (7) and a Java driver program using the Java Database Connection (JDBC) package. We have defined the table structure in a way that we hope will lend itself to being able to represent many different types of simulation systems. Each simulation system will have its own experimental parameters. The parameters that have been defined for our Ising experiment are shown in table 1. The database not only records the *meta-data* about the parameters such as their description and minimum and maximum values, but also the values of the parameters that have actually been used in real experiments.

Our control scripts have been modified to test for successful experiment completion. In the

case of our Ising model a simple example of this is to ensure the number of lines on the output log file is 10 million, and that there exists a final configuration file. After the experiment is deemed to be successful the file is moved to a known (standard) directory and the relevant data on the experiment run is inserted into the database. A typical output log from our Ising experiment is 50MB, which needs to be subsequently analysed. Instead of copying the actual data into the database (perhaps as a Binary Large Object - BLOB) in the prototype we insert the absolute path name to the file in the database.

We recognised very early that our scientific data archive will likely consist of experiments with a **sparse** cross-product of parameters as shown in figure 2. Using a series of `SELECT` statements allows us to determine whether all the required data is in the database. It also allows us to select only sequences of data that we are interested in for our data analysis programs. While the ability for a tool to compute the necessary cross-product of a parameter set and initiate experiments is not new (e.g. Nimrod (1)), as we conceive of new data ranges and analysis techniques that we would like to investigate, we are able to play "what if" games and interrogate the database as to what data is already in the database, and what will need to be generated. In the case that more data may need to be generated, the database queries are able to output the precise values that are required to generate execution scripts and then schedule the jobs.

In future versions of the framework we hope to extend our graphical user interface to easily allow novice users to interrogate the system, using some of the ideas mentioned in section 2. The underlying tools used by the first version of our prototype uses a programmer interface and a library of interface routines that can be linked with our simulation programs written in C++ or Java.

We recognise that many application scientists are uncomfortable with having to remember complicated database access routines. To alleviate the need for users to remember the routines we introduce a new Library layer to shield the user. This library layer is shown in figure 3. The library actually serves two purposes. The first is to shield the users from the database. The second is actually to protect the database from the users.

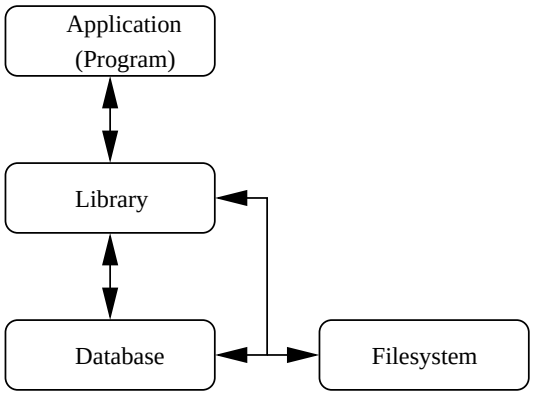


Figure 3: Layered software stack diagram showing indirect access to the filesystem via the database.

A significant problem could arise if the files that store the raw experimental data are moved, deleted, or renamed. We have considered a number of options, including creating a file system space that is only writable by programs using the database as an access control mechanism, but as yet this issue is unresolved. We can also write our own "safe" versions of the `cp` and `mv` programs that update the database as any data files are moved. These new programs would work in conjunction with the library layer to ensure the database's consistency in the face of different file-system operations.

In an idea similar to that reported in (8) we are attempting to "fool the application program" into believing that separate data files actually exist where they are actually pseudo-files accessed

by the underlying database and manipulated by our database access routines. The application and user writes to "files" in the normal way, but the calls are intercepted by library functions that access the database rather than the normal file-system.

5 Database Implementation

In our current prototype we have identified a number of issues that we have not yet been able to adequately resolve. These include: the difficulty in assigning data ranges; ensuring links to data files remain valid; and computing the cross-product of a variable number of parameters. These points are discussed below.

A naive implementation of parameter spaces, as used above, is useful when a simple range of parameter values is required. It works simply because most parameters to our modelling experiments are continuous variables that we can give a reasonable delta (or smallest change amount). A problem arises when we wish to represent a data range which is not sampled at even intervals. An example of this is when the scientist wishes to have their values evenly spaced on a $log - log$ graph: the anti-log values are not evenly spaced! The only real solution to this is either to specify a formula for the calculation of values in the required range, or alternatively to enumerate the entire set of valid values. Our current solution to the problem of un-even parameter values has been to define a `ListParameter` class that simply maintains a list of valid values; for variables able to take every value in the range of $[min, max]$ we use a `RangeParameter` class.

At present we represent all numerical parameters as a fixed-length (Java) string and perform increment/decrement operations using the fixed length strings. This is done for two reasons: firstly when our data analysis programs iterate over the raw data files they have a consistent representation of the data value, and secondly we wish to specify a value's precision unambiguously. We don't want to have any rounding effects (due to the floating point representation) creeping into the system. We will not end up with a situation that 2.00000000001 is stored instead of 2.0 due to rounding errors and machine epsilons.

We often wish to inspect the database to find out what data is present and what is missing. This involves a cross-product of the parameters used to record the data. A large problem is then efficiently and algorithmically computing cross-products of variable numbers of parameters. The number of elements in a cross-product increases exponentially with each new parameter added to the vector. The traditional method for enumerating the cross-products is via nested loops. In the situation that the number of parameters is not known in advance, the enumeration is difficult to achieve.

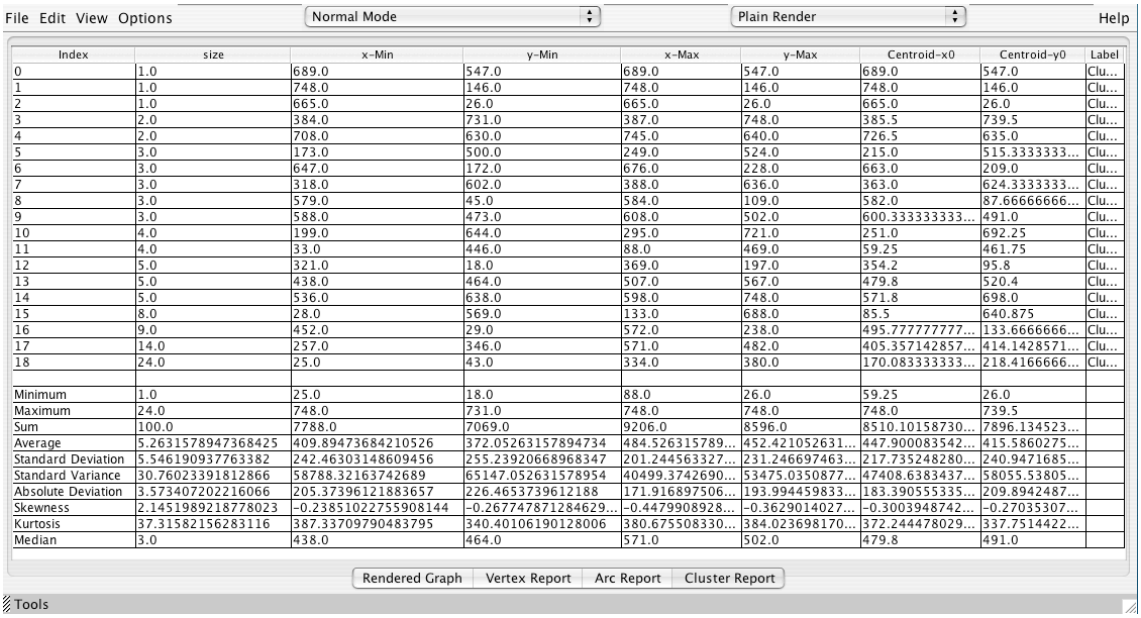
We have defined a `ParameterArray` class that can be used to group together different ListParameters and RangeParameters. The array can be iterated over, producing each element of the cross-product in the parameter range, for an arbitrary number of parameters. The major benefit of this implementation is that it is not necessary to evaluate every member of the cross-product if not required.

6 Tools

In this section we describe the virtual spreadsheet tool we have prototyped using the Java Swing (9) graphical user interface (GUI) library components and the JDBC package wrapping around a MySQL database. Our tool was built to meet our pragmatic need to manage large numbers of simulations that have been run on various cluster computer resources over a six month time period.

The JDBC technology for interfacing to relational databases is well established and need not be described here. In summary, various Java classes and methods wrap around the database server in a client-server software model. The Java Swing *JTable* has provided the basis for our virtual spreadsheet GUI and deserves some comment. It provides standard GUI widget behaviours for a tableaux of edit-able cells that has higher run-time performance than would a simple array

of separate text-field widgets. Its basis is an interface specifying method signatures for accessing, editing and counting the cells in the tableaux. We map a two dimensional tableaux to a cut through our hyper-brick of parameters. The power of being able to construct partial cross-products is that we need *not* expand them in full, and that we can manipulate entire swathes of meta-data visually.



Index	size	x-Min	y-Min	x-Max	y-Max	Centroid-x0	Centroid-y0	Label
0	1.0	689.0	547.0	689.0	547.0	689.0	547.0	Clu...
1	1.0	748.0	146.0	748.0	146.0	748.0	146.0	Clu...
2	1.0	665.0	26.0	665.0	26.0	665.0	26.0	Clu...
3	2.0	384.0	731.0	387.0	748.0	385.5	739.5	Clu...
4	2.0	708.0	630.0	745.0	640.0	726.5	635.0	Clu...
5	3.0	173.0	500.0	249.0	524.0	215.0	515.3333333...	Clu...
6	3.0	647.0	172.0	676.0	228.0	663.0	209.0	Clu...
7	3.0	318.0	602.0	388.0	636.0	363.0	624.3333333...	Clu...
8	3.0	579.0	45.0	584.0	109.0	582.0	87.66666666...	Clu...
9	3.0	588.0	473.0	608.0	502.0	600.333333333...	491.0	Clu...
10	4.0	199.0	644.0	295.0	721.0	251.0	692.25	Clu...
11	4.0	33.0	446.0	88.0	469.0	59.25	461.75	Clu...
12	5.0	321.0	18.0	369.0	197.0	354.2	95.8	Clu...
13	5.0	438.0	464.0	507.0	567.0	479.8	520.4	Clu...
14	5.0	536.0	638.0	598.0	748.0	571.8	698.0	Clu...
15	8.0	28.0	569.0	133.0	688.0	85.5	640.875	Clu...
16	9.0	452.0	29.0	572.0	238.0	495.777777777...	133.6666666...	Clu...
17	14.0	257.0	346.0	571.0	482.0	405.357142857...	414.1428571...	Clu...
18	24.0	25.0	43.0	334.0	380.0	170.083333333...	218.4166666...	Clu...
Minimum	1.0	25.0	18.0	88.0	26.0	59.25	26.0	
Maximum	24.0	748.0	731.0	748.0	748.0	748.0	739.5	
Sum	100.0	7788.0	7069.0	9206.0	8596.0	8510.10158730...	7896.134523...	
Average	5.2631578947368425	409.89473684210526	372.05263157894734	484.526315789...	452.421052631...	447.900083542...	415.5860275...	
Standard Deviation	5.546190937763382	242.46303148609456	255.23920668968347	201.244563327...	231.246697463...	217.735248280...	240.9471685...	
Standard Variance	30.76023391812866	58788.32163742689	65147.052631578954	40499.3742690...	53475.0350877...	47408.6383437...	58055.53805...	
Absolute Deviation	3.573407202216066	205.37396121883657	226.4653739612188	171.916897506...	193.994459833...	183.390555335...	209.8942487...	
Skewness	2.1451989218778023	-0.23851022755908144	-0.267747871284629...	-0.4479908928...	-0.3629014027...	-0.3003948742...	-0.27035307...	
Kurtosis	37.31582156283116	387.33709790483795	340.40106190128006	380.675508330...	384.023698170...	372.244478029...	337.7514422...	
Median	3.0	438.0	464.0	571.0	502.0	479.8	491.0	

Figure 4: A view of a JTable showing edit-able cells and a statistical report generated from them.

Figure 4 shows some of the capabilities of our system. A view is being generated of some derived data from a simulation. It is suitably sorted and displayed as a sheet report, with some simple statistical measurements also shown. Generally for the sort of experiments we report here (such as the Ising simulations) statistics are *not* generated directly within the tool but are accumulated as output from separate analysis programs. It is not uncommon for specially optimised analysis programs to be written for studies of this sort, and the management tool needs to be flexible enough to accommodate these.

Our Ising experiment uses a cross-product of parameter values given by:

$$p \times T \times d \times L \times s \times u \times steps \tag{1}$$

where each parameter is described in table 1. We envisage that an "experiment" will generally represent a body of work such as we describe. Namely, one or more simulation programs that may have separate versions; a set of adjustable input parameters and a resulting collection of output files that will typically occupy considerable disk space. The main aim of our tool is to support the design and operation of numerical experiments like this, so that better use can be made of compute resources and of prior generated data.

The classical *modus operandi* that we and many colleagues use is, having created a simulation program, to write various shell scripts that generate job runs and to use these to generate output files. Output is often organised rather simply either with parameters embedded in the filenames or sometimes in the names of sub directories. Recognising this, our tool is designed to import metadata in the form of path information - file and directory names. Regular expression utilities such as are provided by the Java *String* class are useful for parsing prior filename metadata. It is perfectly satisfactory for the tool to manage the files in their existing naming scheme, providing this file and path information can be left fixed and stored as indirect addresses in the database.

Generally the post-run analysis of simulations such as we describe is much too computationally intensive and the data sets too large to load it all up into a conventional spreadsheet program. A tool like we describe is needed to manage the subsequent analysis processing runs that will let loose a highly optimised statistical analysis sub program across the bulk data files.

In the case of our Ising runs described above, each output data set consists of 2 or 3 measured values at each of upwards of 11 million steps. Part of the experiment is to use different statistical analysis techniques to pass over the data constructing statistics and correlation functions, and other values derived from them. At present our data set consists of over 9 posed experiments, each with its own parameter cross-products and typically more than 150 output files per experiment, each file or set of files storing 11 million data triplets. This totals approximately 0.5 terabytes of data at present.

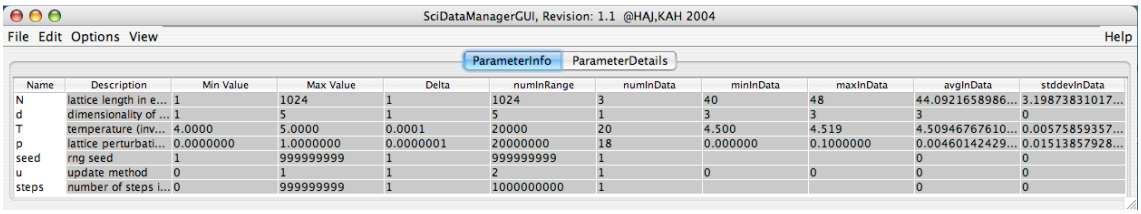
Having collected this data and used each experiment to attack a particular research question, it is proving valuable to subsequently mine the data base for other trends and correlations. New research questions have arisen from close examination of the outcomes of the simpler experiments. The tool helps highlight gaps in the existing parameter space coverage of the problem and helps plan subsequent runs. Working on a simple proportional depreciation value model for our supercomputer cluster, the existing data cost approximately NZ$50k to generate. It is therefore very worthwhile to mine it for maximal research value and also to make optimal use of further compute resource committed to the project.

Figures 5 and 6 show screen-dumps of our experiment planning tool. Figure 5 shows the summary of statistical information computed from the database's parameter records. It is supremely useful for us to be able to define parameters, such as T (temperature of the model) with large ranges – shown as 4.000 to 5.000 – but only perform experiments on subsets of those ranges. In this illustration, the actual range of parameters used to store data in this database is 4.500 to 4.519 inclusive. This also means that when we create a cross-product of this parameter with other parameters in the model, we have the choice of being able to use either: (i) the complete range of the parameter; (ii) the range of actual values in the database for this parameter, or (iii) another range, which may overlap with the actual values in the database.

The statistics shown in figure 5 are generated from the metadata and do not represent an analysis of the measurements from the raw data files to which the metadata is only an indirect guide. In experiments like this one there are various phase transitions involved so it is non-trivial *a priori* to estimate sensible parameter values for T and p. Indeed finding the phase transition values in these two parameters is one of the experiments goals. The planning tool is therefore valuable to guide progress and having carried out a preliminary experiment to scan coarsely in T, p space, a finer grained scan can be carried out subsequently. Management of the *prior* data means that is is not wasted and that new measurements can be progressively interleaved with old ones.

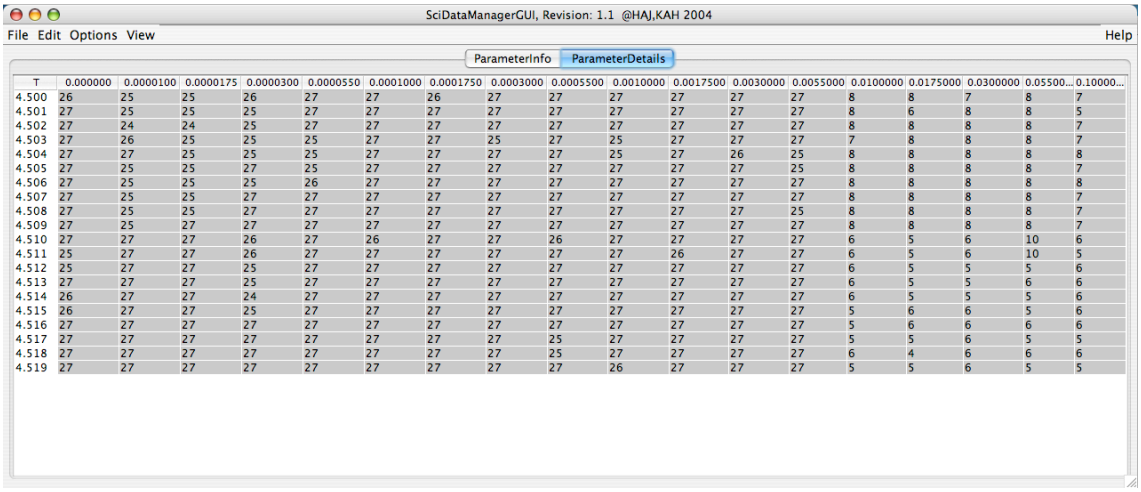
Figure 6 shows the output of the program when the user chooses two parameters to display as horizontal and vertical axes. The major parameters of the experiment under consideration are $d = \{3\} \times L = \{40, 44, 48\} \times T = \{4.500 to 4.519\} \times p = \{0.0 to 0.1 in \log steps\}$ The user has selected to view $T \times P$ for this experiment. Each cell in the virtual spreadsheet represents a hyper block of the remaining parameters' values. The tool can be adjusted to show various summary information for each cell. In the screen-dump it shows a count of the number of records in the database corresponding to the particular value of $T \times p$. Other options include: a colour highlight for missing data; an estimate of the resource time used so far/required to fill in "holes"; the number of elements in the remaining cross-product component.

Figure 6 shows some cells with a much smaller count value than the majority. These represent holes in the data. This view is effectively an automatically generated version of the phenomenon shown in figure 2. The available data is sparse over this parameter cross-product range - either deliberately or by accident - which can occur if supercomputer job runs fail. Our management tool therefor e helps us extract value from what may be an imperfect incomplete set of runs, without the difficulties of hand editing analysis job scripts.



Name	Description	Min Value	Max Value	Delta	numInRange	numInData	minInData	maxInData	avgInData	stddevInData
N	lattice length in e...	1	1024	1	1024	3	40	48	44.0921658986...	3.19873831017...
d	dimensionality of ...	1	5	1	5	1	3	3	3	0
T	temperature (inv...	4.0000	5.0000	0.0001	20000	20	4.500	4.519	4.50946767610...	0.00575859357...
p	lattice perturbati...	0.0000000	1.0000000	0.0000001	20000000	18	0.000000	0.1000000	0.00460142429...	0.01513857928...
seed	rng seed	1	999999999	1	999999999	1			0	0
u	update method	0	1	1	2	1	0	0	0	0
steps	number of steps i...	0	999999999	1	1000000000	1			0	0

Figure 5: A view from our tool showing summary statistical information of the parameter ranges in our experiments. Note the statistics are derived only from metadata parameters, not from the millions of measured data values the metadata indirectly addresses.



T	0.000000	0.0000100	0.0000175	0.0000300	0.0000550	0.0001000	0.0001750	0.0003000	0.0005500	0.0010000	0.0017500	0.0030000	0.0055000	0.0100000	0.0175000	0.0300000	0.05500...	0.10000...
4.500	26	25	25	26	27	27	26	27	27	27	27	27	27	8	8	7	8	7
4.501	27	25	25	25	27	27	27	27	27	27	27	27	27	8	6	8	8	5
4.502	27	24	24	25	27	27	27	27	27	27	27	27	27	8	8	8	8	7
4.503	27	26	25	25	25	27	27	25	27	25	27	27	27	7	8	8	8	7
4.504	27	27	25	25	25	27	27	27	27	25	27	26	25	8	8	8	8	8
4.505	27	25	25	27	25	27	27	27	27	27	27	27	25	8	8	8	8	7
4.506	27	25	25	25	26	27	27	27	27	27	27	27	27	8	8	8	8	8
4.507	27	25	25	27	27	27	27	27	27	27	27	27	27	8	8	8	8	7
4.508	27	25	25	27	27	27	27	27	27	27	27	27	25	8	8	8	8	7
4.509	27	25	27	27	27	27	27	27	27	27	27	27	27	8	8	8	8	7
4.510	27	27	27	26	27	26	27	27	26	27	27	27	27	6	5	6	10	6
4.511	25	27	27	26	27	27	27	27	27	27	26	27	27	6	5	6	10	5
4.512	25	27	27	25	27	27	27	27	27	27	27	27	27	6	5	5	5	6
4.513	27	27	27	25	27	27	27	27	27	27	27	27	27	6	5	5	6	6
4.514	26	27	27	24	27	27	27	27	27	27	27	27	27	6	5	5	5	6
4.515	26	27	27	25	27	27	27	27	27	27	27	27	27	5	6	6	5	6
4.516	27	27	27	27	27	27	27	27	27	27	27	27	27	5	6	6	6	6
4.517	27	27	27	27	27	27	27	27	25	27	27	27	27	5	5	6	5	5
4.518	27	27	27	27	27	27	27	27	25	27	27	27	27	6	4	6	6	6
4.519	27	27	27	27	27	27	27	27	27	26	27	27	27	5	5	6	5	5

Figure 6: A view from our tool showing a partial cross-product of parameters for the Ising model experiments. Rows are for parameter T and columns for parameter p. The value in each cell is the number of distinct records in the database corresponding to the particular values of T and p. Note that some cell values are lower than others, meaning that there are fewer records pertaining to those parameters. This could be caused by fewer data points being investigated in that parameter space or failed simulation runs.

7 Types and Associated Issues

Our tool as described in section 6 was developed specifically to address our Ising experiments. There are some interesting issues concerned with generalising it for other experiments with out having to recode it entirely. These are concerned with data typing and introspection issues.

We envisage the general case whereupon an "experiment" is designed with some number N_p of parameters. Each of these parameters $P_i, i = 1, 2, ...N_p$ may have specific type information. As we discuss in section 5 it is convenient to use a fixed length string as the storage container for manipulating both integer and floating point data in our database and inside the management tool. For most of the sort of numerical simulation work we envisage the two simple data types "double" and "int" are sufficient, but even the fact that we must distinguish between these two poses a problem.

The JTable widget from the Java Swing library utilises a generalised *Object* model for cells, and it is a matter for the application developer to pack and unpack Objects into the actual data types used by the program. We are considering a flat list of well known (simple) data types that can be specified when the user designs an experiment. Objects which are ferried around as fixed length strings are then effectively introspect-ed and treated as their appropriate simple type. This

model is sufficient for this sort of tool where we do not concern our selves with compound types such as arrays, lists or collections of the simpler types.

Our design strategy is that the tool itself copes with compound types through its parameter cross-products' data structures. It is not trivial to see how to tackle what would otherwise become a combinatorial explosion of possible (compound) data types which would need to be supported.

Posing constraints on the parameters presents similar type grounded issues. For example some of our Ising parameters are unconstrained doubles, some are constrained to be positive only. Some integers such as our "dimension" parameter are constrained to small positive values. For the sorts of experiment we envisage a short flat list of applicable constraints is manageable. These can be hard coded and enabled appropriately. We are considering how a generalised "supertype" object could also contain constraint information. At present our tool copes with this issue by using what are effectively enumerated types in the form of explicit lists of allowable values for each parameter. This is feasible in the context of a particular experiment such as the Ising model where although we would ideally like to be able to sample a large range of double precision parameter values, in practice we are limited by computational and storage feasibility to relatively short lists (around 100 members at most).

8 Summary and Conclusions

We have identified a common operational pattern for scientific experiments - and which is particularly common for numerical simulation experiments. We have described some of the problems facing a computational scientist managing "runs", their measurements, and the resulting configuration files. We have described how *ad-hoc* solutions can be augmented using commonly available public domain software tools, and how a "Computational Laboratory Information System" can be based around a database. Our prototype and the ideas arising from it can be usefully applied to situations where large amounts of data are generated and must be curated. We believe the tools we describe here can be used to construct a system that is capable of coping with quite large repositories, but which is also open enough that distributed components can be readily added to cope with collaborative grid environments.

We have described the sparse data structure that arises from partial cross-products of parameters into a simulation, when the scientist does not want to explore the full parameter space. We have shown that this need not be an obstacle to a simulation management system, and that gaps in the data can be handled. We are presently extending our prototype to include some simple data-mining utilities that will be compatible with the data management system.

Some general issues have arisen from this work - specifically those concerning practical approaches to type space management and sub-type/enumerated type constraint management. We believe our approach and the technological solution we describe may be of use to other researchers trying to manage complex numerical simulations.

9 Acknowledgements

We thank Massey University and the Allan Wilson Centre for use of "Helix" supercomputer cluster time for the Ising simulation work reported in this paper.

References

[1] Abramson D., Sosic R., Giddy J. and Hall B., Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations, in *The 4th IEEE Symposium on High Performance Distributed Computing*, Virginia, August 1995.

[2] Hawick, K.A., Coddington, P.D. and James, H.A., Distributed Frameworks and Parallel Algorithms for Processing Large-Scale Geographic Data, in *Parallel Computing* 10, 1297 (2003).

[3] Hawick, K.A. and James, H.A., Ising Model Scaling Behaviour on Small-World Networks, Technical Note CSTN-006, Available from http://www.massey.ac.nz/~kahawick/cstn, March 2004.

[4] Hawick, K.A. and James, H.A., Small-World Effects in Wireless Sensor Networks, Technical Note CSTN-001, Available from http://www.massey.ac.nz/~kahawick/cstn, March 2004.

[5] James, H.A., Scogings, C.J. and Hawick, K.A., A Framework and Simulation Engine for Studying Artificial Life, in *Res. Lett. in the Information and Mathematical Sciences*, Vol 6, May 2004, ISSN 1175-2777.

[6] Metropolis, N., Rosenbluth A.W., Rosenbluth M.N., Teller, A.H. and Teller E., Equation of state calculations by fast computing machines, in *J. Chem. Phys.*, 21(6), pp 1087–1092, June 1953.

[7] MySQL, MySQL Database homepage, Available from http://www.mysql.com Last visited July 2004.

[8] Patten, C.J., Vaughan, F.A., Hawick, K.A. and Brown, A.L., DWorFS: File System Support for Legacy Applications in DISCWorld, in *Proc. of the 5th IDEA Workshop*, Fremantle, February 1998.

[9] Sun Microsystems. Inc, "Java Foundation Classes (JFC/Swing) Web page", Available from http://java.sun.com/products/jfc/index.jsp last visited 26 August, 2004.

[10] World Wide Web Consortium (W3C), "Metadata at W3C". Available from http://-www.w3.org/Metadata/ Last visited July 2004.