Placeholder

Author1, Author2, Author3
Institution, City, Country
$\{$*author1, author2, author3*$\}$*@organisation.com*

***Abstract*—Placeholder**

***Keywords*-simulation; performance; energy; genetic algorithms; optimisation**

I. Introduction

Computer simulations enable one to model a real-world system through the use of software. These simulations mimic the behaviour of the physical system and enable stakeholders to perform hypothesis testing. Hypothesis testing enables users to perform less risk-averse strategies in the virtual system, which may otherwise have unintended consequences and lead to detrimental impacts in the real system.

Recently, the concept of a 'digital twin' has emerged. These digital twins are designed to be a digital replica of a specific system, as opposed to a generalised system. For instance, the electricity market in the UK, as opposed to any liberalised electricity market in the UK. This provides greater confidence that changes made to the digital twin will happen in the physical system. These digital twins enable tests to be performed much quicker than a traditional, physical world test, at frequently lower costs, and without the risk of unforeseen impacts.

However, it remains true that not all digital twins are equal. Whilst digital twins should not be seen as tools to predict the future, and can not be expected to [1], a degree of confidence can be attributed to a model depending on how well a model mimics real life over time. In this paper, we show how it is possible to infer the optimal parameters to improve the validation of a digital twin. We use the ElecSim agent-based electricity market model as an exemplar to this problem [2]

II. Motivation

Parameter spaces for simulations rapidly become large as the number of parameters and valid values for each parameter increases. This is perhaps why in their work McGough *et al.* [**?**] only ever vary two input parameters at a time and even then only consider a maximum of eleven different values for each of these parameters – leading to 121 different simulations which were be executed.

Continuous value parameters are the hardest to deal with when performing parameter space optimisations. Selecting size of discretisation for the parameter is vitally important – too small will lead to excessively large numbers of simulations, whilst too corse will mean one is more likely to miss the optimal value. Integer values are similar in complexity save from the fact that their is a minimum level of discretisation – the unit value.

Let us assume here that we wish to perform a parameter sweep over just six continuous values for a simulation which takes just five minutes per run. If we discretise each of the continuous parameters to one hundred values then we would require 10^{12} simulation runs, which is just over 9.5 million years of execution time. If we were to restrict the discretisation to just ten values per parameter this would reduce our parameter space to one million simulation runs and 9.5 years of simulation. Either case is far in excess of what can be performed – and would be a significant energy drain in its own right. Thus the use of a machine learning optimisation approach is highly desirable here.

We use here Figure **??** from [**?**] to illustrate the motivation for identifying a Pareto frontier. The variation in colours represents the different learning rates of the RL approach whilst the spread of each colour represents variations in how much importance is placed on the computers selected. It can be seen from this figure that there is no global optimal – minimising energy and average overhead. This figure demonstrates that a Pareto frontier is present, though, due to the small number of sample points this is most likely not the actual Pareto frontier.

III. Related Work

Multi-objective optimisation problems are commonplace, with applications as diverse as electoral zone design [3] to generation expansion planning [4]. Here we review various applications that have utilized multi-objective optimization.

Multi-objective optimization has been used in many different fields in the literature, and many multi-objective problems have been solved with the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) algorithm [5]. There are, however, examples of other algorithms being used such as the Multi-Objective Genetic Algorithm [6].

Ponsich *et al.* apply the NSGA-II algorithm to electoral zone design [3]. The criteria in which the geographical units must be aggregated are population equality, compactness, and contiguity. They found that the NSGA-II obtains promising results when compared with the simulated annealing algorithm, producing better-distributed solutions over a wider-spread front.

Kannan *et al.* also used the NSGA-II algorithm for the generation expansion planning problem [4]. This problem seeks to identify which generating units should be commissioned and when they should become available over the long-term planning horizon. They optimised for two trade-off solutions: to minimize cost, and minimize sum of normalized constraint violations; and to minimize investment cost and minimize outage cost. They were able to find a Pareto-front with very high computational efficiency.a

Wei *et al.* used NSGA-II to optimize energy consumption and indoor environment thermal performance [7]. They used simulation data which contained information on energy consumption and indoor thermal comfort. The study used a back propagation network optimised by a genetic algorithm to characterize building behaviour that was used for predicting the energy consumption and indoor thermal comfort status of residential buildings. This model was used as a fitness function to the NSGA-II algorithm.

Guha *et al.* used multi-objective optimization to design a ship hull [8]. As their objective functions were not smooth, they found that evolutionary techniques to attain optimum hull forms the most practical. They tested a number of different algorithms, and found that Sequential Quadratic Programming, Pattern Search and Interior-Point were very sensitive to the initial guess and prone to getting stuck in a local minima. The genetic algorithm and particle swarm optimisation, however, proved to be more robust and able to determine the global minima in most trials.

IV. Optimization methods

Classical optimization methods, such as non-linear programming, find single solutions per simulation run. However, many real-world problems naturally have multiple objectives to optimise. Traditionally, classical optimization methods are used by artificially converting them into a single-objective problem.

However, this does not take into account the various trade-offs between equally optimal solutions, known as Pareto-optimal solutions. It, therefore, becomes important to find multiple Pareto-optimal solutions. A Pareto frontier is made up of many Pareto-optimal solutions. The results of which can be displayed graphically, enabling a user to choose between various solutions and trade-offs.

Classical methods require multiple applications of an optimization algorithm, with various scalings between rewards to achieve a single reward. The population approach of genetic algorithms, however, enable the Pareto frontier to be found in a single simulation run. An example of a multi-objective genetic algorithm is NSGA-II, which is used here.

A. Genetic Algorithms

Genetic algorithms are a class of evolutionary algorithms first proposed by John Holland in 1975 [9]. We detail the workings of genetic algorithms in this section.

Algorithm 1 Genetic algorithm [10]

```
1: $t = 0$
2: initialize $P(t)$
3: evaluate structures in $P(t)$
4: **while** termination condition not satisfied **do**
5:     $t = t + 1$
6:     select reproduction $C(t)$ from $P(t-1)$
7:     recombine and mutate structures in $C(t)$
         forming $C'(t)$
8:     evaluate structures in $C'(t)$
9:     select each individual for $P(t)$ from $C'(t)$
         or $P(t-1)$
10: **end while**
```

An initial population of individual structures $P(0)$ is generated and each individual of the population is evaluated for fitness. A subset of individuals, $C(t) \subset P(0)$, are chosen for mating, selected proportionally to their fitness. The better individuals are given a higher chance to reproduce and create the offspring group $C'(t)$. $C'(t)$ have characteristics dependent on the genetic operators, crossover and mutation. The genetic operators are an implementation decision [10].

Once the new population has been created via the mutation and crossover genetic operators, the new population $P(t)$ is created by merging individuals from $C'(t)$ and $P(t-1)$. See Algorithm 1 for detailed pseudocode.

B. NSGA-II

NSGA-II is used in this work due to it being shown to be efficient for multi-objective optimization on a number of benchmark problems. It has been shown to find a better spread of solutions than the Pareto Archived Evolution Strategy (PAES) [11] and Strength Pareto EA (SPEA) [12] when approximating the true Pareto-optimal front [5].

The majority of multi-objective optimization algorithms use the concept of *domination* during population selection [13]. A non-dominated genetic algorithm seeks to achieve the Pareto-optimal solution, so no single optimization solution should dominate another.

We define a process to determine which solutions to keep:

1) Non-dominated sorting: We assume that there are M objective functions to minimise, and that $\mathbf{x^{(1)}} = x_j^{(1)}$ and $\mathbf{x^{(2)}}$ are two solutions. In minimisation, $x_j^{(1)} < x_j^{(2)}$ denotes that solution $\mathbf{x^{(1)}}$ is better than solution $\mathbf{x^{(2)}}$ on objective j. A solution $\mathbf{x}^{(1)}$ is said to dominate the solution $\mathbf{x}^{(2)}$ if the following conditions are true:

1) The solution $\mathbf{x}^{(1)}$ is no worse than $\mathbf{x}^{(2)}$ in every objective. I.e. $x_j^{(1)} \geq x_j^{(2)} \;\; \forall j \in \{1, 2, \ldots, M\}$.
2) The solution $\mathbf{x}^{(1)}$ is better than $\mathbf{x}^{(2)}$ in at least one objective. I.e. $\exists j \in \{1, 2, \ldots, M\} \;\; s.t. \;\; x_j^{(1)} > x_j^{(2)}$.

Once each of the solutions, for example $\mathbf{x}^1$ and $\mathbf{x}^2$, are calculated for the M objective functions, the solutions are sorted according to their level of non-domination. An example of layering of levels is shown in Figure **??**. In this example, f_1 and f_2 are the objective functions to be minimized. The Pareto frontier is the first front which contains solutions that are not dominated by any other solution. The solutions in layer 1 are dominated only by those in the Pareto front, and are non-dominated by layer 2 and layer 3.

The solutions are then ranked according to the level of their layer. For example, the solutions contained in the Pareto front are given a fitness rank – referred to as i_{rank} – of 1, the solutions in layer 1 are given a fitness rank of 2.

2) Density Estimation: An average density estimation is computed for each solution by calculating the average distance between the two points either side of the solution in question. This quantity is known as $i_{distance}$, and is an estimate of the size of the largest cuboid which contains only i and no other points.

3) Crowded comparison operator: We use the crowded operator ($\prec_n$) to ensure that the final front is an evenly spread out Pareto-optimal front. Each solution has two attributes; Non-domination rank (i_{rank}) and Local crowding distance ($i_{distance}$). We can then define a partial order:
$i \prec_n j$ if ($i_{rank} < j_{rank}$) or (($i_{rank} = j_{rank}$) and ($i_{distance} > j_{distance}$)) [5].

This concludes that a point with a lower rank is preferred, and if two points have the same rank the point which is located in a less dense area is preferred.

4) Main loop: As in the standard genetic algorithm a random population $P(0)$ is created. The population is then sorted according to non-domination. Binary tournament selection, recombination and mutation operators are used to create a child population $C'(0)$ of size N.

After the first population the procedure changes. This procedure is shown in Algorithm 2. Initially, a combined population is formed $R(t) = P(t) \cup C'(t)$ of size $2N$. $R(t)$ is then sorted according to non-domination. A new population is now formed (P_{t+1}), adding solutions from each front level until the size of P_{t+1} exceeds N. The solutions of the last accepted level are then sorted according to $\prec_n$, and a total of N solutions are chosen, rejecting those from the last layer that have a smaller crowding distance [5].

Algorithm 2 NSGA-II main loop [5]

```
1:  $R_t = P_t \cup C'_t$ combine parent and child population
2:  $\mathcal{F} =$ fast-non-dominated-sort ($R_t$)
        where $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$
3:  $P_{t+1} = \emptyset$
4:  while do $|P_{t+1} < N|$
5:      Calculate the crowding distance of ($\mathcal{F}_i$))
6:      $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$
7:  end while
8:  Sort($P_{t+1}, \prec_n$) sort in descending order using $\prec_n$
9:  $P_{t+1} = P_{t+1}[0 : N]$ select the first $N$ elements of $P_{t+1}$
10: $Q_{t+1} =$ make-new-population($P_{t+1}$) using
        selection, crossover and mutation to create
        the new population $Q_{t+1}$
11: $t = t + 1$
```

The entire process is shown in Figure **??**, and is repeated until the termination condition is met.

V. Simulation environment

A. HTC-Sim

The simulation environment is HTC-Sim, a trace-driven simulation framework for energy consumption in High Throughput Computing systems [**?**]. The simulation handles two types of users – interactive users who can sit down in-front of a computer and use it along with high-throughput users who submit multiple jobs through a batch submission system which are run on computers when idle. The computers in the system are considered at three logical levels – the whole system, a cluster of computers (a number of computers in a distinct location) and individual computers.

This work is primarily concerned with two metrics constituting our objectives, namely average overhead and total energy consumed, described below.

Average task overhead is defined as the time difference between the task entering (q_t) and departing (f_t) the system, and the actual task execution time (d_t) for a set of tasks T:

$$\frac{1}{|T|} \sum_{t}^{t \in T} (f_t - q_t - d_t).$$

Energy consumption is the total energy consumed for the HTC workload. Fine-grained energy consumption results are recorded at the per- computer, cluster and system levels. Providing a breakdown of energy consumption for each state, e.g. sleep, idle, active (HTC and/or interactive user). The total energy consumption is then calculated as follows:

$$\sum_{c=0}^{n} \sum_{p=0}^{m} t_{c,p} E_{c,p} \tag{1}$$

where n is the number of computers, m is the number of power states, $t_{c,p}$ is the time spent by computer c in state p and $E_{c,p}$ is the power consumption rate of computer c in state p. For non-HTC states $E_{c,p} = 0$.

B. Reinforcement Learning Scheduler

Reinforcement Learning [**?**] is a machine learning technique used to learn how to react to an environment. In RL there is an environment which is observed by an agent. Often the agent will build a state space representing the environment. For each state in the state space is a corresponding action vector representing every action which can be taken in that state. Initially each action in the action vector has the same probability of being selected. When the agent observes the environment being in a specific state it chooses an action from the action vector based on either an explorative or exploitative policy. If an explorative policy is chosen then the action is selected at random from the action vector whilst if an exploitative policy is in force then the action which has seen the 'best' historical reward is selected. The choice of policy is made by randomly selecting explorative with probability ϵ and exploitative otherwise. Once the action is completed and it is known if the action was good or bad then the action within the action vector is updated – increasing the value if the choice was good and decreasing it otherwise.

The RL scheduler by McGough *et al.* [**?**] is based upon this RL approach where the state of the environment is a combination of whether computers with the HTC system are free for use and the hour of the day. The granularity of state space can be varied in size from representing each computer individually through only being aware of how many computers are free in a given cluster to only knowing how many free computers there are across the whole HTC setup. Likewise the hour of the day could be for any day (24 states) or for each hour within a week (168 states). The actions can be which computer / cluster or just the whole system to place the task on or that the task is unlikely to complete and should be held in a queue.

C. State space parameters

We present here the state space parameters that the GA will search over in order to identify the optimal policies. Further details can be found in [**?**]:

Bring ϵ out of the list (as common to many) and then for each explain how they could benefit energy / overhead.

- **Week**: is the state space for the RL in terms of a day or a week $\{$*boolean*$\}$.
- **Entity level**: is the state space for the RL in terms of individual computers, clusters of computers or just the whole system *(computer, cluster, whole)*.
- ϵ **policy**: What is the ϵ-greedy policy changed on? $\{$*days, previous, ratio, hit*$\}$. Note that this has an impact on the meaning of days, change, threshold and previousPolicy.
- **ranges**: The date range on which to change ϵ. $\{$*[0,999999], ...*$\}$[1],[2].
- **reward boundaries**: reward values over which the ϵ value will be changed. $\{$*(0,1], ...*$\}$[3],[4]
- ϵ: The set of ϵ values. $\{$*(0,1], ...*$\}$[3],[4]
- σ: The amount of influence the computer energy efficiency has on RL reward *[0,1]*.
- **days**: Change ϵ based on the number of days of RL which have been performed *[0, 365], 0 = don't use*.
- δ: Increase ϵ by δ if current day into RL $\leq$ days *[0,1]*.
- **history**: The number of previous tasks to consider when computing the action *[-1,999 999], -1 = all tasks*.
- **gaussian**: Do we apply a gaussian decay over the task history when computing the action? *boolean*.
- **prior**: If all prior actions gave a negative reward then increase ϵ by 0.1 *boolean*.
- **threshold**: If the ratio of best reward to average reward is less than threshold, use the previous value of ϵ *[0,1]*.
- **sanity**: Defer running a task during the same hour that the computer is due to be rebooted *boolean*.

If we assume here, conservatively, that each continuous parameter is discretised into one hundred values then this creates a search space of some $2.81x10^{37}$. Again, assuming that the simulation takes five minutes to run for each parameter combination this is $2.67x10^{32}$ years.

VI. Results Alex + Matt + Steve

We first evaluate if the NSGA-II approach will lead to a Pareto frontier for total energy consumed and average overhead. Figure **??** illustrates progressive iterations of the NSGA-II algorithm with successive iterations in lighter colours. The initial (dark) colours are scattered widely whilst the final iteration (light) indicates a sharp edge closest to the two axis. It is interesting to note that although there is no global optimal for both energy and overhead the Pareto front is 'sharp' in the bottom left corner indicating that there is a good compromise for both objectives. There is also a separate region of points with lower energy consumption but substantially higher overheads. This appears to be cases where tasks are only allowed to run on a small number of energy efficient resources – at the expense of significantly reducing throughput. For all parameter sets along the Pareto front *sanity* was true, demonstrating that not running tasks during the hour when a computer will be rebooted was the best policy. We therefore consider *sanity* no further.

Table I presents the parameter sets and objective values for minimum overhead, minimum energy and optimal combination of speed and consumption. The optimal combination of speed and consumption was attained by first scaling average overhead and total power consumed between 0 and 100. Next, we summed the scaled overhead and total power consumed and chose the combination with the minimum value. This enabled us to choose the minimum combination of both objectives with equal weighting. The scaling, however, could be changed to suit individual preferences of power consumption or average overhead. As the following parameters were identical for all cases, we present them here rather than in the table: ϵ-policy was previous, gaussian was false, $ranges_1$ was 0, ratio was 0.9375 and R_1 was -0.5842. The energy consumption here is far better than those presented in the paper by McGough *et al.* [**?**] we achieve approximately the same energy consumption as they did for their best energy case for our best overhead case and we have reduced the energy consumption for the best energy case by 19MWh – though at the expense of significantly increasing overhead. In comparison our compromise case outperforms the original work on both energy and overhead. It should be noted that this was achieved solely through the tuning of

[1]Although this can be an arbitrarily long list we limit this to three values for this work.

[2]Note $value_i \leq value_2$

Table I
PARAMETERS FOR OPTIMAL OBJECTIVES

		State space												
type	change	Space	week	days	ϵ_1	ϵ_2	history	prior	$Ranges_2$	R_2	R_3	threshold	Energy	Overhead
Overhead	0.9025	Cl	F	218	0.8323	0.3897	488962	F	0	0	0.7290	0.4168	66.64	10.94
Energy	0.9025	Wh	T	218	0.5496	0.5496	805676	T	633334	-0.4638	-0.4638	0	41.56	3864
Balance	0	Cl	F	0	0.8323	0.3897	847384	F	0	-0.4638	-0.4638	0.4168	49.26	22.93

the simulation parameters with the use of NSGA-II, as both sets of results run the same underlying code.

To better understand the parameters which effect the Pareto Front, we fit a Lasso regression [14] to distinct clusters of the Pareto Front. Lasso regression is a linear regression technique which produces coefficients of exactly 0 to give interpretable models like subset selection. The parameters were all scaled between 1 and 100 allowing direct comparison between parameter coefficients.

Initially, we clustered the data using the unsupervised learning technique DBSCAN [15]. This technique was chosen due to its effectiveness at clustering data points which are close together. This yields better results, for our dataset, than a method such as k-means clustering which partitions the data into Voronoi cells. The results, shown in Figure **??**, show a large negative coefficient for ratio, epsilon 3 and reward boundaries when predicting average overhead.

The coefficients for total power show that at low average overheads, and high total power consumed, ratio, history, epsilon 1 and a false value for prior have a positive effect on increasing power consumed or reducing average overhead. That is, an increase in history leads to an increase in total power consumed.

Figure **??** demonstrates the impact of adding δ to ϵ if the RL trainer has been running for less than the prescribed number of days. Here the GA has learnt to use large values of δ when energy consumption is more important, suggesting a more explorative approach favours energy efficiency – perhaps due to the fact that over the simulation period the state varies significantly. By contrast, the number of days shows no clear pattern. Though as the δ value is often very small this may have little if any impact.

The granularity for the RL state space with respect to computers is presented in Figure **??**.a). Here in almost all cases the cluster level is as the best choice. This is most likely a consequence of the fact that it is a compromise between fine-grained computer level and course-grained whole system level. Interestingly for minimum energy whole system becomes more optimal. Perhaps a con sequence of most tasks being held not to be executed thus the cluster not being relevant anymore.

The other aspect of state space – day or week – is presented in Figure **??**.b). Here all but the most extremely long overhead cases are optimal with the day case. This would suggest that, although there is a difference based on the day of the week, this can only be exploited in the case where energy reduction is key. The ϵ policy is compared in Figure **??**.a (left). In almost all cases the previous policy is optimal apart from a small number of cases. This indicates that basing ϵ on the average reward of the previous day is the best policy. The ratio of best reward to average reward makes up most of the remaining points indicating that for both of these cases an adaptive policy which can move between explorative and exploitative as time moves on is the best approach – a consequence of the state of the system changing as time progresses. Only one 'static' policy is seen as optimal - where the value of ϵ changes by the number of days the RL has been running for.

The reward history and whether we apply a gaussian decay over this is presented in Figures **??**.b (middle) and **??**.c (right). The size of the history seems to be bimodal with the extreme and larger overhead objectives having a value around 840,000 whilst most of the low overhead values are in the region of 500,000. Suggesting that forgetting history more quickly favours lower overheads – but at the expense of higher energy consumption. By contrast the choice of when to use a gaussian decay is less obvious suggesting that other factors are at play.

VII. CONCLUSIONSALEX + MATT + STEVE

In this paper we have demonstrated the potential of genetic algorithms, specifically NSGA-II, to provide efficient design space exploration of the operating policies of digital twin simulations. We apply the approach to parameterise the operating policies of a target system, a digital twin simulation of a high-throughput computing infrastructure. We evaluate the performance of the system with respect to energy consumption and performance.

References

[1] W. Jager, "Simulating consumer behaviour: a perspective," *Environmental Policy and Modelling in Evolutionary Economics*, pp. 1–28, 2006.

[2] A. Kell, M. Forshaw, and A. S. Mcgough, "ElecSim : Monte-Carlo Open-Source Agent-Based Model to Inform Policy for Long-Term Electricity Planning," pp. 556–565.

[3] A. Ponsich, E. A. R. García, R. A. M. Gutiérrez, S. G. de-los Cobos Silva, M. A. G. Andrade, and P. L. Velázquez, "Solving electoral zone design problems with NSGA-II," pp. 159–160, 2017.

[4] S. Kannan, S. Baskar, J. D. McCalley, and P. Murugan, "Application of NSGA-II algorithm to generation expansion planning," *IEEE Transactions on Power Systems*, vol. 24, no. 1, pp. 454–461, 2009.

[5] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II," *CEUR Workshop Proceedings*, vol. 1133, pp. 850–857, 2000.

[6] T. Murata and H. Ishibuchi, "MOGA: Multi-objective genetic algorithms," no. November, pp. 289–294, 1995.

[7] W. Yu, B. Li, H. Jia, M. Zhang, and D. Wang, "Application of multi-objective genetic algorithm to optimize energy efficiency and thermal comfort in building design," *Energy and Buildings*, vol. 88, pp. 135–143, 2015.

[8] A. Guha and J. Falzaranoa, "Application of multi objective genetic algorithm in ship hull optimization," *Ocean Systems Engineering*, vol. 5, no. 2, pp. 91–107, 2015.

[9] J. H. Holland, *Search methodologies: Introductory tutorials in optimization and decision support techniques, second edition*, 1975.

[10] T. Back, D. B. Fogel, and Z. Michalewicz, "Evolutionary Computation 1 Basic Algorithms and Operators," *Comprehensive Chemometrics*, pp. ix – x, 2009.

[11] J. Knowles and D. Corne, "The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimisation," *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, vol. 1, pp. 98–105, 1999.

[12] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms A comparative case study," pp. 292–301, 2006.

[13] E. K. Burke and K. Graham, *Search methodologies: Introductory tutorials in optimization and decision support techniques, second edition*, 2014.

[14] R. Tibshirani, "Regression Shrinkage and Selection Via the Lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[15] X. and others Ester, Martin and Kriegel, Hans-Peter and Sander, Jörg and Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." *Kdd*, vol. 96, pp. 226–231, 1996.