Doctorate of Philosophy

Schrödinger's Catwalk

Brian Flynn

University of Bristol

December, 2020

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LISTINGS

ACRONYMS

BF	Bayes factor. 24–26, 32–34, 47, 50, 54, 56, 61–63
CLE	classical likelihood estimation. 13
EDH	experiment design heuristic. 18–20, 22, 25, 34, 44, 54, 55
ES	exploration strategy. 26–34, 39, 42, 44, 48, 50, 61, 64, 73, 76
ET	exploration tree. 27, 28, 30, 31, 33, 34, 44, 46
FH	Fermi-Hubbard. 57
GA	genetic algorithm. 33
HPD	high particle density. 17
IQLE	interactive quantum likelihood estimation. 13, 14
LTL	log total likelihood. 15
ML	machine learning. 6, 25, 26
MS	model search. 26–28, 30, 34, 42, 44
MVEE	minimum volume enclosing ellipsoid. 17
NV	nitrogen-vacancy. 9
NVC	nitrogen-vacancy centre. 14
PGH	particle guess heuristic. 18–20, 44
QHL	quantum Hamiltonian learning. v, 8–14, 16, 18, 20, 21, 24–26, 30, 31, 33, 34, 39, 42, 47, 54, 55, 73
QL	quadratic loss. 16
QLE	quantum likelihood estimation. 13
QMLA	Quantum Model Learning Agent. v, viii, 8, 13, 23, 24, 26–30, 33, 34, 39, 42–48, 50, 51, 59–64, 73

SMC sequential monte carlo. 11–13, 15, 18, 19, 22, 30

TLTL total log total likelihood. 16, 24–26, 33

GLOSSARY

Jordan Wigner transformation (JWT)	Jordan Wigner transformation . 59, 60, 63
Loschmidt echo (LE)	Quantum chaotic effect described. . 13, 14
hyperparameter	Variable within an algorithm that determines how the algorithm itself proceeds.. 11
instance	a single implementation of the QMLA algorithm. 47, 73
model	The mathematical description of some quantum system. 23
probe	Input probe state, $\|\psi\rangle$, which the target system is initialised to, before unitary evolution. plural. 13, 14, 18, 20, 21, 53
results directory	Directory to which the data and analysis for a given run of QMLA are stored. . 48
run	collection of QMLA instances. ii, viii, 47, 48, 63, 64, 73
spawn	Process by which new models are generated by combining previously considered models.. 28
success rate	. 47, 48
term	Individual constituent of a model, e.g. a single operator within a sum of operators, which in total describe a Hamiltonian. . 23
volume	Volume of a parameter distribution's credible region.. 16, 17, 54, 55, 62
win rate	. 47, 48

Part I

THEORETICAL STUDY

1

GENETIC ALGORITHMS

The QMLA framework lends itself easily to the family of optimsation techniques called *evolutionary algorithms*, where individuals, sampled from a population of candidates, are considered, in generations, as solutions to the given problem, and iterative generations aim to efficiently search the available population, by mimicing biological evolutionary mechanisms [**?**]. In particular, we develop a exploration strategy (ES) which incorporates an genetic algorithm (GA) in the generation of models; GAs are a subset of evolutionary algorithms where candidate solutions are expressed as strings of numbers representing some configuration of the system of interest [**?**]. Here we will first introduce the concept of an GA, before describing the adaptations which allow us to build a genetic exploration strategy (GES).

1.1 GENETIC ALGORITHM DEFINITION

GAs work by assuming a given problem can be optimised, if not solved, by a single candidate among a fixed, closed space of candidates, called the population, $\mathcal{P}$. A number of candidates are sampled at random from $\mathcal{P}$ into a single *generation*, and evaluated through some objective function (OF), which assesses the fitness of the candidates at solving the problem of interest. Candidates from the generation are then mixed together to produce the next generation's candidates: this *crossover* process aims to combine only relatively strong candidates, such that the average candidates' fitness improve at each successive generation, mimicing the biological mechanism whereby the genetic makeup of offspring is an even mixture of both parents. The selection of strong candidates as parents for future generations is therefore imperative; in general parents are chosen according to their fitness as determined by the OF. Buidling on this biological motivation, much of the power of GAs comes from the concept of *mutation*: while offspring retain most of the genetic expressions of their parents, some elements are mutated at random. Mutation is crucial in avoiding local optima of the OF landscape by maintaining diversity in the examined subspace of the population.

Pseudocode for a generic GA is given in Algorithm 1, but we can also informally define the procedure as follows. Given access to the population, $\mathcal{P}$,

1. Sample N_m candidates from the population at random
   (a) call this group of candidates the first generation, μ.
2. Evaluate each candidate $\gamma_j \in \mu$.
   (a) each γ_j is assigned a fitness, g_j
   (b) the fitness is computed through an objective function acting on the candidate, $g(\gamma_j)$.
3. Map the fitnesses of each candidate, $\{g_j\}$, to selection probabilities for each model, $\{s_j\}$

   (a) e.g. by normalising the fitnesses, or by removing some poorly-performing candidates and then normalising.
4. Generate the next generation of candidates, μ'
   (a) $\mu = \{\}$
   (b) Select pairs of parents, p_1, p_2, from μ
      i. Each candidate's probability of being chosen is given by their s_j
   (c) Cross over p_1, p_2 to produce children candidates, c_1, c_2.
      i. mutate c_1, c_2 according to some random probabilistic process
      ii. keep c_i only if it is not already in μ', to ensure N_m unique models are tested at each generation.
   (d) until $|\mu'| = N_m$, iterate to step (b).
5. Until the N_g^{th} generation is reached, iterate to step 2..
6. The strongest candidate on the final generation is deemed the solution to the posed problem.

Candidates are manifested as *chromosomes*, i.e. strings of fixed length, whose entries, called *genes*, each represent some element of the system. In general, genes can have continuous values, although usually, and for all purposes in this thesis, genes are binary, capturing simply whether or not the gene's corresponding feature is present in the chromosome.

1.1.1 *Example: knapsack problem*

One commonly referenced combinatorial optimisation problem is the *knacksack problem*: given a set of objects, where each object has a defined mass and also a defined value, determine the set of objects to pack in a knapsack which can support a limited weight, such that the value of the packed objects is maximised. Say there are n objects, we can write the vector containing the values of those objects as $\vec{v}$, and the vector of their weights as $\vec{w}$. We can then represent configurations of object sets as candidate vectors $\vec{\gamma}_j$, whose genes are binary, and simply indicate whether or not the associated object is included in the set. For example, with $n = 6$,

$$\gamma_j = 100001 \Longrightarrow \vec{\gamma}_j = (1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1), \tag{1.1}$$

indicates a set of objects consisting only of those indexed first and last, with none of the intermediate objects included.

The fitness of any candidate is then given by the total value of that configuration of objects, $v_j = \vec{v} \cdot \vec{\gamma}_j$, but candidates are only admitted[1] if the weight of the corresponding set of objects is less than the capacity of the knapsack, i.e. $\vec{w}_j \cdot \vec{\gamma}_j \leq w_{max}$.

1 Note there are alternative strategies to dealing with candidates who violate the weight condition, such as to impose a penalty within the OF, but for our purposes let us assume we simply disregard violators.

Algorithm 1: Genetic algorithm

Input: $\mathcal{P}$ // Population of candidate models
Input: $g()$ // objective funtion
Input: map_g_to_s() // function to map fitness to selection probability
Input: select_parents() // function to select parents among generation
Input: crossover() // function to cross over two parents to produce offspring
Input: N_g // number of generations
Input: N_m // number of candidates per generation

Output: γ' // strongest candidate

$\mu \leftarrow \text{sample}\,(\mathcal{P}, N_m)$
for $i \in 1, ..., N_g$ **do**
  for $\gamma_j \in \mu$ **do**
    $g_j \leftarrow g(\gamma_j)$ // assess fitness of candidate
  end
  $\{s_j\} \leftarrow$ map_g_to_s($\{g_j\}$) // map fitnesses to normalised selection probability
  $\mu_c = \arg\max_{s_j}\{\gamma_j\}$ // record champion of this generation

  $\mu \leftarrow \{\}$ // empty set for next generation
  while $|\mu| < N_m$ **do**
    $p_1, p_2 \leftarrow$ select_parents($\{s_j\}$) // choose parents based on candidates' s_j
    $c_1, c_2 \leftarrow$ crossover(p_1, p_2) // generate offspring candidates based on parents
    for $c \in \{c_1, c_2\}$ **do**
      if $c \notin \mu$ **then**
        $\mu \leftarrow \mu \cup \{c\}$ // keep if child is new
      end
    end
  end
end
$\gamma' \leftarrow \arg\max_{s_j}\{\gamma_j \in \mu\}$ // strongest candidate on final generation

return γ'

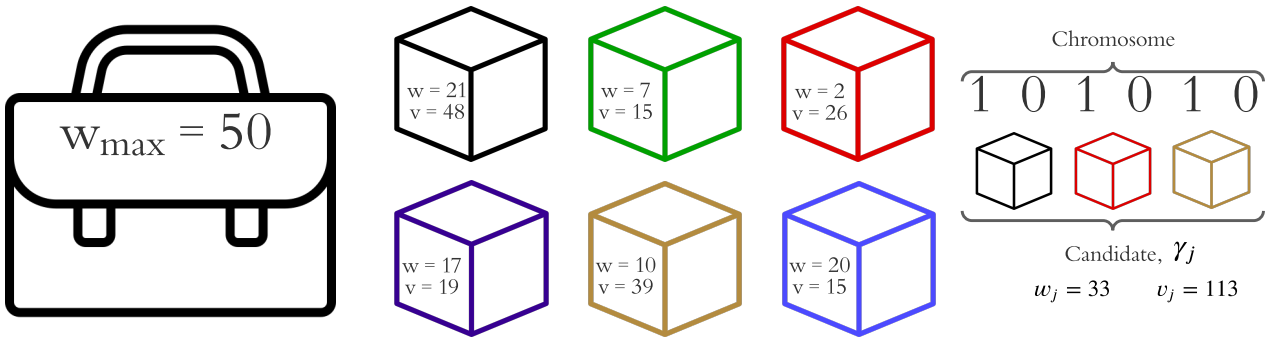

Figure 1.1: Depiction of the knapsack problem. **Left**, A knapsack which can hold any number of objects but is constrained by the total weight it can support, $w_{max} = 50$. **Centre**, A set of objects are available, each with associated weight, w, and value v. The objective is to find the subset of objects which maximise the total value, while not exceeding the capacity of the knapsack. **Right**, An example chromosome, i.e. candidate γ_j, where the bits of the chromosome indicate whether the corresponding object is included, allowing for calculation of the total weight and value of the candidate, w_j, v_j.

For example where each individual object has value < 50 and weight < 25 and $w_{max} = 50$, recalling $\gamma_j = 100001$, say,

$$\vec{v} = (48 \quad 15 \quad 26 \quad 19 \quad 39 \quad 15) \implies v_j = \vec{\gamma}_j \cdot \vec{v} = 48 + 15 = 63; \tag{1.2a}$$

$$\vec{w} = (21 \quad 7 \quad 2 \quad 17 \quad 10 \quad 20) \implies w_j = \vec{\gamma}_j \cdot \vec{w} = 21 + 20 = 41. \tag{1.2b}$$

We can hence assess the fitness of γ_j as 63 and deem it a valid candidate since it does not exceed the weight threshold. We can likewise compute the total weight and value of a series of randomly generated candidates, and deem them valid or not.

Name	Candidate	Value	Weight	Valid
γ_1	110011	117	58	No
γ_2	101010	113	33	Yes
γ_3	011110	99	36	Yes
γ_4	011011	95	39	Yes
γ_5	111000	89	30	Yes
γ_6	010111	88	54	No
γ_7	100010	87	31	Yes
γ_8	110001	78	48	Yes
γ_9	011101	75	46	Yes
γ_{10}	110000	63	28	Yes
γ_{11}	000011	54	30	Yes
γ_{12}	000101	34	37	Yes

Table 1.1: Candidate solutions to the knapsack problem.

The strongest (valid) candidates from Table 1.1 are 101010, 011110. By spawning from these candidates through a one-point crossover at the midpoint, we get $\gamma_{c_1} = 101110, \gamma_{c_2} = 011010$, from which we can see $v_{c_1} = 132, w_{c_1} = 50$, i.e. by combining two strong candidates we produce the strongest-yet-seen valid candidate.

By repeating this procedure, it is expected to uncover candidates which optimise v_j while maintining $w_j \leq w_{max}$, or at least to produce near-optimal solutions, using far less time/resources than brute-force evaluation of all candidates, which is usually sufficient. For instance, with $n = 100$ objects to consider, there are $2^{100} \approx 10^{30}$ candidates to consider; the most powerful supercomputers in the world currently claim on the order of Exa-FLOPs, i.e. 10^{18} operations per second, of which say ~ 1000 operations are required to test each candidate, meaning 10^{15} candidates can be checked per second in a generous example. This would still require 10^{12} seconds to solve absolutely, so it is reasonable in cases like this to accept *approximately optimal* solutions[2].

1.1.2 *Selection mechanism*

A key subroutine of every GA is the mechanism through which it nominates candidates from generation μ as parents to offsping candidates in $\mu + 1$ [**?**]. All mechanisms have in common that they act on a set of candidates from the previous generation, where each candidate, γ_j, has

2 Simply put: in machine learning, *good enough* is good enough. We will adopt this philosophy for the remainder of this thesis and life.

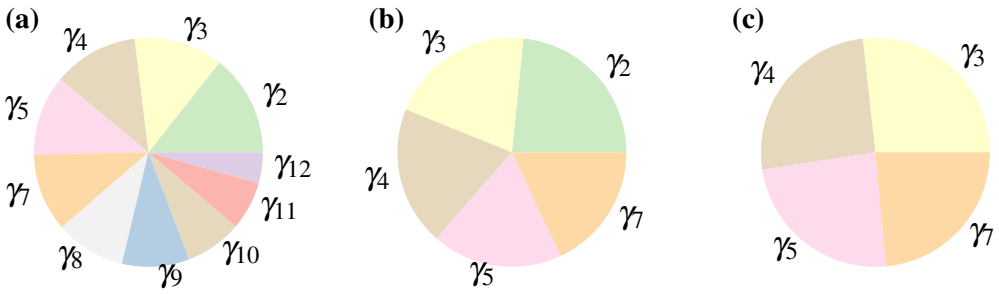

Figure 1.2: Roulette wheels showing selection probability s_i for corresponding candidates γ_i. Colours here only distinguish candidates, they do not encode any information. **b**, The set of potential parents is truncated to include only the strongest five candidates. **a**, All valid candidates are assigned selection probability based on their value in Table 1.1. **c**, After one parent (γ_2) has been chosen, it is removed from the roulette wheel and the remaining candidates' probabilities are renormalised for the selection of the second parent.

been evaluated and has fitness value, g_j. Among the viable schemes for selecting individual parents from the set of candidates, μ are

- Rank selection: candidates are selected with probabilty proportional to their ranking relative to the fitness of contemporary candidates in the same generation.
- Tournament selection: a subset of k candidates are chosen at random from μ, of which the candidate with the highest fitness is taken as the parent.
- Stochastic universal sampling: candidates are sampled proportional to their fitness, but the sampling algorithm is biased to ensure high-fitness candidates are chosen at least once within the generation.

We will only detail the mechanism used later within QMLA, the common fitness proportional selection, known as *roulette selection* [**?**]. This is a straightforward strategy where we directly map candidates' fitness, g_i to a selection probability, s_i, simply by normalising $\{g_i\}$, allowing us to visualise a roulette wheel of uneven wedges, eachh of which correspond to a candidate. Then we need only conceptually spin the roulette wheel to select the first parent, γ_{p_1}. We then remove γ_{p_1} from the set of potential parents, renormalise the remaining $\{s_i\}$, and spin the wheel again to choose the second parent, γ_{p_2}.

Practically, we repeat the process outlined until the next generation is filled, usually we have $|\mu| = N_m$, and desire that every generation should contain the same N_m candidates, so we repeat the roulette selection $N_m/2$ times per generation, since every pair of parents yield two offspring. It is important that meaningful differences in fitness are reflected by the selection probability, which is difficult to ensure for large N_m, e.g. with ten models, the strongest candidate is only a marginally more probable parent than the worst – this effect is amplified for larger N_m. We therefore wish to reduce the set of potential parents to ensure high quality offspring: we truncate

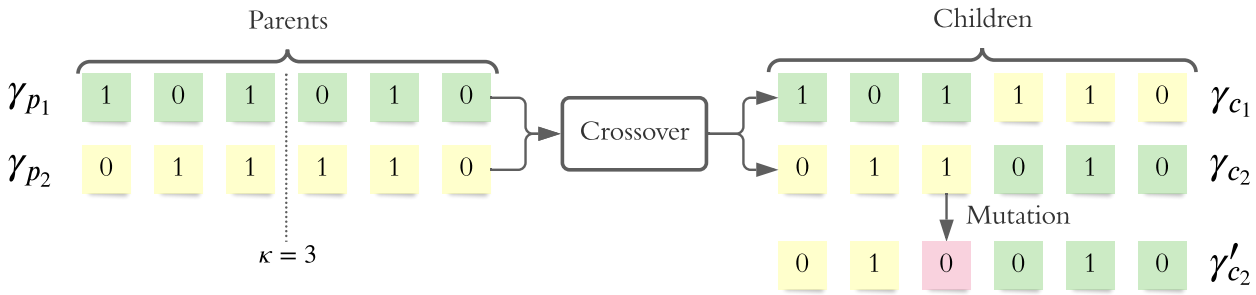

Figure 1.3: Crossover and mutation of chromosomes. Two parents, $\gamma_{p_1}, \gamma_{p_2}$, are nominated from the process in Fig. 1.2. They are then crossed-over via a one-point crossover with crossing point $\kappa = 3$, resulting in children candidates $\gamma_{c_1}, \gamma_{c_2}$. One child chromosome is mutated to yield a new candidate, γ'_{c_2}. The candidates added to the next generation are then $\{\gamma_{c_1}, \gamma'_{c_2}\}$.

μ and retain only the highest-fitness $\frac{N_m}{2}$ models as selectable parents. The roulette selection is shown in Fig. 1.2.

1.1.3 *Reproduction*

When a pair of parents have been nominated by the selection mechanism above, it remains to use those parents to *reproduce*, i.e. to produce offspring which should inherit and improve upon the properties of their parents. Here we use a *one point crossover*, whereby the two parent chromosomes are mixed together to form two offspring, about a single point, κ: for candidates of n genes, the first κ genes of γ_{p_1} are conjoined with the latter $n - \kappa$ genes of γ_{p_2}. Often κ is restricted to the midpoint of the chromosomes, although in general we need not impose this: we will instead consider $\kappa \in \left(\frac{n}{4}, \frac{3n}{4}\right)$, e.g. with $n = 12$, $\kappa \in (3, 9)$. The one-point crossover is shown for $n = 6$ with $\kappa = 3$ in Fig. 1.3, recalling the chromosome structure from Section 1.1.1.

By allowing κ other than the midpoint, we drastically increase the number of combinations of parents available for reproduction. Finally, then, parent selection is done by constructing a database of pairs of potentital parents with all available crossover points, with selection probability given by the product of their individual fitnesses. This is conceptually equivalent to selection via roulette wheel as above. Recalling the fitnesses (values) of Table 1.1, for example:

Parent 1	Parent 2	κ	s_{ij}
γ_2	γ_3	2	$11,187\ (= 113 \times 99)$
γ_2	γ_3	3	$11,187$
γ_2	γ_3	4	$11,187$
γ_2	γ_4	2	$10,735\ (= 113 \times 95)$
γ_2	γ_4	3	$10,735$
γ_2	γ_4	4	$10,735$
		$\vdots$	
γ_5	γ_7	2	$7,743\ (= 89 \times 87)$
γ_5	γ_7	3	$7,743$
γ_5	γ_7	4	$7,743$

Table 1.2: Example of parent selection database. Pairs of parents are selected together, with the (unnormalised) selection probability, s_{ij}, given by the product of the individual candidates' fitnesses. Pairs of parents are repeated in the database for differing κ, and all κ are equally likely.

The GA maintains diversity in the subspace of $\mathcal{P}$ it studies, by *mutating* some of the newly proposed offspring candidates. Again, there are a multitude of approaches for this step [**?**], but for brevity we only describe those used in this thesis. For each proposed child candidate, γ_c, we probabilistically mutate each gene with some mutation rate r_m: if a mutation occurs, the child is replaced by γ_c'. That is, γ_c' is added to the next generation, and γ_c is discarded. r_m is a *hyperparameter* of the GA: the performance of the algorithm can be optimised by finding the best r_m for a given problem.

1.1.4 *Candidate evaluation*

Within every generation of the GA, each candidate must be evaluated, so that the relative strength of candidates can be exploited in constructing candidates for the next generation. In the example of the knapsack problem used above, candidates were evaluated by the value of their contennts, but also by whether they would fit in the knapsack. Idenitfiyng the appropriate method by which to evaluate candidates is arguably the most important aspect of designing a GA: while the choice of hyperperameters (N_g, N_m, r_m) dictate the efficacy of the search, the lack of an effective metric by which to distinguish candidates would render the procedure pointless. Considerations are hence usually built into the objective function (OF).

Unlike previous aspects of generic GAs, in the context of QMLA, here we must deviate from default mechanisms. Recalling the overarching goal of QMLA, to characterise some black box quantum system, Q, we do not have access to a natural OF. We wish to optimise the modelling of $\hat{H}'$, but assume we do not know the target $\hat{H}_0$, so we can not simply invoke some loss function, for example. Instead, we must devise schemes which exploit the knowledge we *do* have about

each candidate $\hat{H}_j$, which is the primary challenge in building an ES based on a GA. We propose and discuss a number of options in Section 1.3.

1.2 ADAPTATION TO QMLA FRAMEWORK

Ultimately, the conceived role of a GA within QMLA is to generate the sets of models to place on successive branches of the exploration trees (ETs) in **??**. The apparatus for this is to implement an exploration strategy (ES) whose model generation subroutine calls an external GA. Recall from **??**, that we capture the space of available terms as $\mathcal{T}$, i.e. we list – in advance – the feasible terms which may be included in models[3], with $N_t = |\mathcal{T}|$ the number of terms considered. QMLA is then an optimisation algorithm, attempting to find the set $\mathcal{T}'$ which *best* represents the true terms $\mathcal{T}_0$. Note, this does not require idenitifcation of the precise true model to be successful, as insight can be gained from approximate models which capture the physics of the target system. We introduce metrics for success in Section 1.2.1. We recognise the limitations this structure imposes: we can only identify terms which were conceived in advance; this may restrict QMLA's applicability to entirely unknown systems, where such a primitive set can not even be compiled.

We first need a mapping from models to chromosomes; this is straightforward given the description of chromosomes as binary strings, exemplified in Section 1.1.1. We assign a gene to every term in $\mathcal{T}$, so that candidate models are succinctly represented by bit strings of length N_t. We give an example of the mapping between models and chromosomes in Table 1.3.

1.2.1 F_1-*score*

We need a metric against which to evaluate models, and indeed the entire QMLA procedure.

1.3 OBJECTIVE FUNCTIONS

1.4 APPLICATION

3 Recall that models impose structure on sets of terms: $\hat{H}_j = \vec{\alpha}_j \cdot \vec{T}_j = \sum_{k \in \mathcal{T}_j} \alpha_k \hat{t}_k$.

Model		Chromosome					
	$\vec{T}$	$\hat{\sigma}^x_{(1,2)}$	$\hat{\sigma}^z_{(1,2)}$	$\hat{\sigma}^y_{(2,3)}$	$\hat{\sigma}^x_{(2,3)}$	$\hat{\sigma}^y_{(2,3)}$	$\hat{\sigma}^x_{(2,3)}$
γ_{p_1}	$(\hat{\sigma}^x_{(1,2)} \quad \hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)})$	1	0	1	0	1	0
γ_{p_2}	$(\hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)} \quad \hat{\sigma}^z_{(2,3)})$	0	0	1	0	1	1
γ_{c_1}	$(\hat{\sigma}^x_{(1,2)} \quad \hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)} \quad \hat{\sigma}^z_{(2,3)})$	1	0	1	0	1	1
γ_{c_2}	$(\hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)})$	0	0	1	0	1	0
γ'_{c_2}	$(\hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^x_{(2,3)} \quad \hat{\sigma}^y_{(2,3)})$	0	0	1	1	1	0

Table 1.3: Mapping between QMLA's models and chromosomes used by a genetic algorithm. Example shown for a three-qubit system with six possible terms, $\hat{\sigma}^w_{i,j} = \hat{\sigma}^w_i \hat{\sigma}^w_j$. Model terms are mapped to binary genes: if the gene registers 0 then the corresponding term is not present in the model, and if it registers 1 the term is included. The top two chromosomes are *parents*, $\gamma_{p_1} = 101010$ (blue) and $\gamma_{p_2} = 001011$ (green): they are mixed to spawn new models. We use a one–point cross over about the midpoint: the first half of γ_{p_1} is mixed with the second half of γ_{p_2} to produce two new children chromosomes, $\gamma_{c_1}, \gamma_{c_2}$. Mutation occurs probabilistically: each gene has a 25% chance of being mutated, e.g. a single gene (red) flipping from $0 \rightarrow 1$ to mutate γ_{c_2} to γ'_{c_2}. The next generation of the genetic algorithm will then include $\gamma_{c_1}, \gamma'_{c_2}$ (assuming γ_{c_1} does not mutate). To generate N_m models for each generation, $N_m/2$ parent couples are sampled from the previous generation and crossed over.

APPENDIX

A

FIGURE REPRODUCTION

Most of the figures presented in the main text are generated directly by the QMLA framework. Here we list the implementation details of each figure so they may be reproduced by ensuring the configuration in Table A.1 are set in the launch script. The default behaviour of QMLA is to generate a results folder uniquely identified by the date and time the run was launched, e.g. results can be found at the *results directory* qmla/Launch/Jan_01/12_34. Given the large number of plots available, ranging from high-level run perspective down to the training of individual models, we introduce a plot_level $\in \{1, ..., 6\}$ for each run of QMLA: higher plot_level informs QMLA to generate more plots.

Within the results directory, the outcome of the run's instances are stored, with analysis plots broadly grouped as

- evaluation: plots of probes and times used as the evaluation dataset.
- single_instance_plots: outcomes of an individual QMLA instance, grouped by the instance ID. Includes results of training of individual models (in model_training), as well as sub-directories for anlaysis at the branch level (in branches) and comparisons.
- combined_datasets: pandas dataframes containing most of the data used during analysis of the run. Note that data on the individual model/instance level may be discarded so some minor analyses can not be performed offline.
- exploration_strategy_plots plots specifically required by the ES at the run level.
- champion_models: analysis of the models deemed champions by at least one instance in the run, e.g. average parameter estimation for a model which wins multiple instances.
- performance: evaluation of the QMLA run, e.g. the win rate of each model and the number of times each term is found in champion models.
- meta analysis of the algorithm' implementation, e.g. timing of jobs on each process in a cluster; generally users need not be concerned with these.

In order to produce the results presented in this thesis, the configurations listed in Table A.1 were input to the launch script. The launch scripts in the QMLA codebase consist of many configuration settings for running QMLA; only the lines in snippet in Listing A.1 need to be set according to altered to retrieve the corresponding figures. Note that the runtime of QMLA grows quite quickly with N_E, N_P (except for the AnalyticalLikelihood ES), especially for the entire QMLA algorithm; running quantum Hamiltonian learning (QHL) is feasible on a personal computer in < 30 minutes for $N_e = 1000; N_p = 3000$.

```
#!/bin/bash
```

```
###############
# QMLA run configuration
###############
num_instances=1
run_qhl=1 # perform QHL on known (true) model
run_qhl_mulit_model=0 # perform QHL for defined list of models.
exp=200 # number of experiments
prt=1000 # number of particles

###############
# QMLA settings
###############
plot_level=6
debug_mode=0

###############
# Choose an exploration strategy
###############

exploration_strategy='AnalyticalLikelihood'
```

Listing A.1: "QMLA Launch scipt"

Figure	Exploration Strategy	Algorithm	N_E	N_P	Data
??	AnalyticalLikelihood	QHL	500	2000	Nov_16/14_28
??	DemoIsing	QHL	500	5000	Nov_18/13_56
??	DemoIsing	QHL	1000	5000	Nov_18/13_56
??	DemoIsing	QHL	1000	5000	Nov_18/13_56
??	IsingLatticeSet	QMLA	1000	4000	Nov_19/12_04
3***??**	IsingLatticeSet	QMLA	1000	4000	Sep_30/22_40
	HeisenbergLatticeSet	QMLA	1000	4000	Oct_22/20_45
	FermiHubbardLatticeSet	QMLA	1000	4000	Oct_02/00_09

Table A.1: Implementation details for figures used in the main text.

B

EXAMPLE EXPLORATION STRATEGY RUN

A complete example of how to run the ;sqmla framework, including how to implement a custom ES, and generate/interpret analysis, is given.

BIBLIOGRAPHY

[1] Antonio A. Gentile, Brian Flynn, Sebastian Knauer, Nathan Wiebe, Stefano Paesani, Christopher E. Granade, John G. Rarity, Raffaele Santagati, and Anthony Laing. Learning models of quantum systems from experiments, 2020.

[2] Christopher E Granade, C Ferrie, N Wiebe, and D G Cory. Robust online Hamiltonian learning. *New Journal of Physics*, 14(10):103013, October 2012.

[3] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and David Cory. Quantum hamiltonian learning using imperfect quantum resources. *Physical Review A*, 89(4):042314, 2014.

[4] N Wiebe, C Granade, C Ferrie, and D G Cory. Hamiltonian Learning and Certification Using Quantum Resources. *Physical Review Letters*, 112(19):190501–5, May 2014.

[5] Jianwei Wang, Stefano Paesani, Raffaele Santagati, Sebastian Knauer, Antonio A Gentile, Nathan Wiebe, Maurangelo Petruzzella, Jeremy L O'Brien, John G Rarity, Anthony Laing, et al. Experimental quantum hamiltonian learning. *Nature Physics*, 13(6):551–555, 2017.

[6] Jane Liu and Mike West. Combined parameter and state estimation in simulation-based filtering. In *Sequential Monte Carlo methods in practice*, pages 197–223. Springer, 2001.

[7] Rodolfo A Jalabert and Horacio M Pastawski. Environment-independent decoherence rate in classically chaotic systems. *Physical review letters*, 86(12):2490, 2001.

[8] Nathan Wiebe, Christopher Granade, and David G Cory. Quantum bootstrapping via compressed quantum hamiltonian learning. *New Journal of Physics*, 17(2):022005, 2015.

[9] Arseni Goussev, Rodolfo A Jalabert, Horacio M Pastawski, and Diego Wisniacki. Loschmidt echo. *arXiv preprint arXiv:1206.6348*, 2012.

[10] Bas Hensen, Hannes Bernien, Anaïs E Dréau, Andreas Reiserer, Norbert Kalb, Machiel S Blok, Just Ruitenberg, Raymond FL Vermeulen, Raymond N Schouten, Carlos Abellán, et al. Loophole-free bell inequality violation using electron spins separated by 1.3 kilometres. *Nature*, 526(7575):682–686, 2015.

[11] Alexandr Sergeevich, Anushya Chandran, Joshua Combes, Stephen D Bartlett, and Howard M Wiseman. Characterization of a qubit hamiltonian using adaptive measurements in a fixed basis. *Physical Review A*, 84(5):052315, 2011.

[12] Christopher Ferrie, Christopher E Granade, and David G Cory. How to best sample a periodic probability distribution, or on the accuracy of hamiltonian finding strategies. *Quantum Information Processing*, 12(1):611–623, 2013.

[13] Christopher E Granade. Characterization, verification and control for large quantum systems. page 92, 2015.

[14] Christopher Granade, Christopher Ferrie, Steven Casagrande, Ian Hincks, Michal Kononenko, Thomas Alexander, and Yuval Sanders. QInfer: Library for statistical inference in quantum information, 2016.

[15] Christopher Ferrie. High posterior density ellipsoids of quantum states. *New Journal of Physics*, 16(2):023006, 2014.

[16] Michael J Todd and E Alper Yıldırım. On khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discrete Applied Mathematics*, 155(13):1731–1744, 2007.

[17] Ian Hincks, Thomas Alexander, Michal Kononenko, Benjamin Soloway, and David G Cory. Hamiltonian learning with online bayesian experiment design in practice. *arXiv preprint arXiv:1806.02427*, 2018.

[18] Lukas J Fiderer, Jonas Schuff, and Daniel Braun. Neural-network heuristics for adaptive bayesian quantum estimation. *arXiv preprint arXiv:2003.02183*, 2020.

[19] Sheng-Tao Wang, Dong-Ling Deng, and Lu-Ming Duan. Hamiltonian tomography for quantum many-body systems with arbitrary couplings. *New Journal of Physics*, 17(9):093017, 2015.

[20] Stefan Krastanov, Sisi Zhou, Steven T Flammia, and Liang Jiang. Stochastic estimation of dynamical variables. *Quantum Science and Technology*, 4(3):035003, 2019.

[21] Emmanuel Flurin, Leigh S Martin, Shay Hacohen-Gourgy, and Irfan Siddiqi. Using a recurrent neural network to reconstruct quantum dynamics of a superconducting qubit from physical observations. *Physical Review X*, 10(1):011006, 2020.

[22] Murphy Yuezhen Niu, Vadim Smelyanskyi, Paul Klimov, Sergio Boixo, Rami Barends, Julian Kelly, Yu Chen, Kunal Arya, Brian Burkett, Dave Bacon, et al. Learning non-markovian quantum noise from moir\'{e}-enhanced swap spectroscopy with deep evolutionary algorithm. *arXiv preprint arXiv:1912.04368*, 2019.

[23] Eliska Greplova, Christian Kraglund Andersen, and Klaus Mølmer. Quantum parameter estimation with a neural network. *arXiv preprint arXiv:1711.05238*, 2017.

[24] Andrey Y Lokhov, Marc Vuffray, Sidhant Misra, and Michael Chertkov. Optimal structure and parameter learning of ising models. *Science advances*, 4(3):e1700791, 2018.

[25] Giovanni Acampora, Vittorio Cataudella, Pratibha R Hegde, Procolo Lucignano, Gianluca Passarelli, and Autilia Vitiello. An evolutionary strategy for finding effective quantum 2-body hamiltonians of p-body interacting systems. *Quantum Machine Intelligence*, 1(3):113–122, 2019.

[26] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.

[27] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

[28] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[29] Ruhul A Sarker and Tapabrata Ray. Agent based evolutionary approach: An introduction. In *Agent-Based Evolutionary Search*, pages 1–11. Springer, 2010.

[30] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[31] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[32] Roger W Hockney and Chris R Jesshope. *Parallel Computers 2: architecture, programming and algorithms*. CRC Press, 2019.

[33] Redis, Nov 2020. [Online; accessed 7. Nov. 2020].

[34] RQ: Simple job queues for Python, Nov 2020. [Online; accessed 7. Nov. 2020].

[35] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925.

[36] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.

[37] Stephen G Brush. History of the lenz-ising model. *Reviews of modern physics*, 39(4):883, 1967.

[38] Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[39] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[40] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

[41] Giuseppe E Santoro and Erio Tosatti. Optimization using quantum mechanics: quantum annealing through adiabatic evolution. *Journal of Physics A: Mathematical and General*, 39(36):R393, 2006.

[42] Victor Bapst, Laura Foini, Florent Krzakala, Guilhem Semerjian, and Francesco Zamponi. The quantum adiabatic algorithm applied to random optimization problems: The quantum spin glass perspective. *Physics Reports*, 523(3):127–205, 2013.

[43] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[44] Walter Greiner, Ludwig Neise, and Horst Stöcker. *Thermodynamics and statistical mechanics*. Springer Science & Business Media, 2012.

[45] John Hubbard. Electron correlations in narrow energy bands. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 276(1365):238–257, 1963.

[46] Richard T Scalettar. An introduction to the hubbard hamiltonian. *Quantum Materials: Experiments and Theory*, 6, 2016.

[47] Editorial. The hubbard model at half a century. *Nature Physics*, 2013.

[48] Pascual Jordan and Eugene Paul Wigner. über das paulische äquivalenzverbot. In *The Collected Works of Eugene Paul Wigner*, pages 109–129. Springer, 1993.

[49] Mark Steudtner and Stephanie Wehner. Fermion-to-qubit mappings with varying resource requirements for quantum simulation. *New Journal of Physics*, 20(6):063010, 2018.

[50] Jarrod McClean, Nicholas Rubin, Kevin Sung, Ian David Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, Eric Schuyler Fried, Craig Gidney, Brendan Gimby, et al. Openfermion: the electronic structure package for quantum computers. *Quantum Science and Technology*, 2020.

[51] Jonas B Rigo and Andrew K Mitchell. Machine learning effective models for quantum systems. *Physical Review B*, 101(24):241105, 2020.

[52] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33, 2020.

[53] Eyal Bairey, Itai Arad, and Netanel H Lindner. Learning a local hamiltonian from local measurements. *Physical review letters*, 122(2):020504, 2019.

[54] Chris J Pickard and RJ Needs. Ab initio random structure searching. *Journal of Physics: Condensed Matter*, 23(5):053201, 2011.

[55] Eli Chertkov and Bryan K Clark. Computational inverse method for constructing spaces of quantum models from wave functions. *Physical Review X*, 8(3):031029, 2018.