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 [19]. 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 [20]. 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 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 candidate, $\{s_j\}$

   (a) e.g. by normalising the fitnesses, or by removing some poorly-performing candidates and then normalising.

4. Generate the next generation of candidates
   (a) Reset $\mu = \{\}$
   (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 candidates 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'

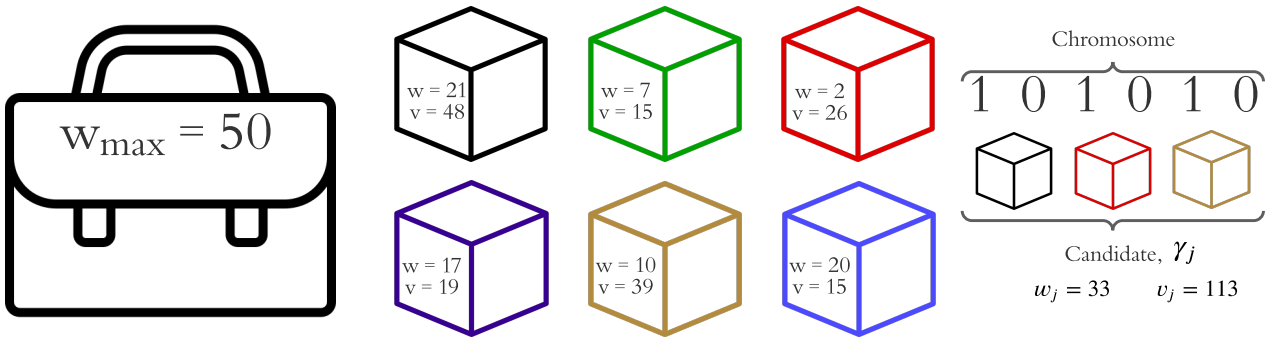

Figure 1.1: Depiction of the knapsack problem. **Left**, A knapsack which can hold any number of objects but is constrained by the total weight it can support, $w_{max} = 50$. **Centre**, A set of objects are available, each with associated weight, w, and value v. The objective is to find the subset 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 $\mu + 1$ [21]. 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.

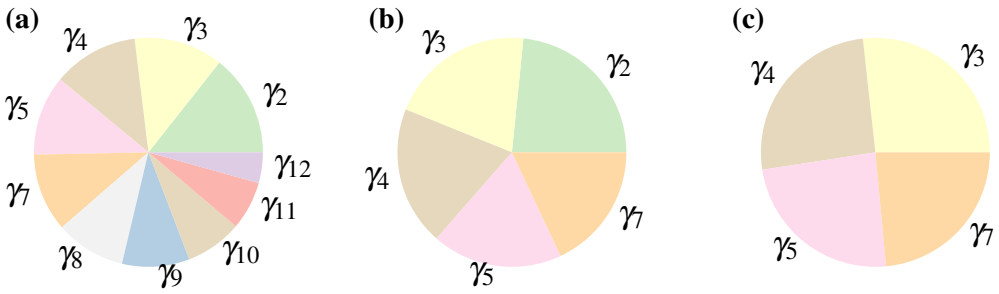

Figure 1.2: Roulette wheels showing selection probability s_i for corresponding candidates γ_i. Colours here only distinguish candidates, they do not encode any information. **b**, The set of potential parents is truncated to include only the strongest five candidates. **a**, All valid candidates are 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* [21]. 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

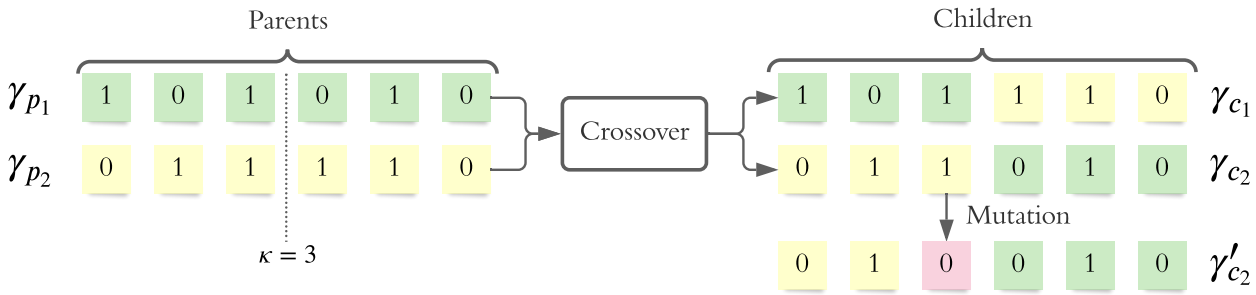

Figure 1.3: Crossover and mutation of chromosomes. Two parents, $\gamma_{p_1}, \gamma_{p_2}$, are nominated from the process in Fig. 1.2. They are then crossed-over via a one-point crossover with crossing point $\kappa = 3$, resulting in children candidates $\gamma_{c_1}, \gamma_{c_2}$. One child chromosome 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 [22], 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.

Common to all proposed OFs, however, is that candidates should first be trained before evaluation, so that their assessment is based on their actual power in explaining the target system, rather than some initial paramterisation which may not capture their potential. This is a tenet of QMLA: for each candidate $\hat{H}_j(\vec{\alpha}_j)$, we use a subroutine to optimise $\vec{\alpha}_j$, again for this study we rely on quantum Hamiltonian learning (QHL).

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 1.2.2. 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.

The structure of the overall QMLA algorithm, recall **??**, is unchanged. In a genetic exploration strategy (GES):

- models are still grouped in branches, here called generations;
- models are still trained, again through QHL;
- branches are evaluated according the the OF to be described in Section 1.3;
- new models are spawned through the genetic algorithm by selecting pairs of parents for crossover, with the resultant offspring models probabilistically mutated.

We detail the corresponding $\mathrm{generate\_models}$ subroutine in Algorithm 2. We can restate the informal description of GAs, now in the context of QMLA, as

1. Sample N_m models from $\mathcal{P}$ at random
   (a) this is the first generation, μ.
2. Evaluate each model $\hat{H}_j \in \mu$.
   (a) train $\hat{H}_j$ through QHL
   (b) apply the objective function to assign the model's fitness g_j

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

3. Map the fitnesses of each model, $\{g_j\}$, to selection probabilities for each model, $\{s_j\}$
   (a) e.g. by normalising the fitnesses, or by removing some poorly-performing models and then normalising.
4. Generate the next generation of models
   (a) Reset $\mu = \{\}$
   (b) Select pairs of parents, $\hat{H}_{p_1}, \hat{H}_{p_2}$, from μ
      i. Each model's probability of being chosen is given by their s_j
   (c) Cross over $\hat{H}_{p_1}, \hat{H}_{p_2}$ to produce children models, $\hat{H}_{c_1}, \hat{H}_{c_2}$.
      i. mutate $\hat{H}_{c_1}, \hat{H}_{c_2}$ according to some random probabilistic process
      ii. keep $\hat{H}_{c_i}$ 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 model on the final generation is deemed the approximation to the system, $\hat{H}'$.

1.2.1 *Models as chromosomes*

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.

1.2.2 *F_1-score*

We need a metric against which to evaluate models, and indeed the entire QMLA procedure. We can gauge the performance of QMLA's model search by the quality of candidate models produced at each generation, so we introduce a mertic to act as proxy for model quality: the F_1-score. In short, $f \in (0, 1)$ indicates the degree to which $\hat{H}_i$ captures the physics of the target system: $f = 0$ indicates that $\hat{H}_i$ shares no terms with $\hat{H}_0$, while $f = 1$ is found uniquely for $\hat{H}_i = \hat{H}_0$. We will define the concept formally next. Note that here we are able to compute f for candidate models because the target $\hat{H}_0$ is simulated, i.e. we know the true terms $\mathcal{T}_0$; this would not be available for a real system with unknown $\hat{H}_0$, but is useful for the analysis of the algorithm itself.

We emphasise that the goal of this work is to identify the *model* which best describes quantum systems, and not to improve on parameter-learning when given access to particular models, since those already exist to a high standard [2, 23]. Therefore we can consider QMLA as a

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 $0 \to 1$ 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.

classification algorithm, with the goal of classifying whether individual terms $\hat{t}$ from a set of available terms $\mathcal{T} = \{\hat{t}\}$ are helpful in describing data which is generated by $\hat{H}_0$, which has $\mathcal{T}_0$. Candidate models $\hat{H}_i$ then have $\mathcal{T}_i$. We can assess $\hat{H}_i$ using standard metrics used regularly in the machine learning (ML) literature, which simply count the number of terms identified correctly and incorrectly,

- true positives (TP): number of terms in $\mathcal{T}_0$ which are in $\mathcal{T}_i$;
- true negatives (TN): number of terms not in $\mathcal{T}_0$ which are also not in $\mathcal{T}_i$;
- false positives (FP): number of terms in $\mathcal{T}_i$ which are not in $\mathcal{T}_0$;
- false negatives (FP): number of terms in $\mathcal{T}_0$ which are not in $\mathcal{T}_i$.

These concepts allow us to define

- *precision*: how precisely does $\hat{H}_i$ capture $\hat{H}_0$, i.e. if a term is included in $\mathcal{T}_i$ how likely it is to actually be in $\mathcal{T}_0$, Eqn 1.3a;
- *sensitivity*: how sensitive is $\hat{H}_i$ to $\hat{H}_0$, i.e. if a term is in $\mathcal{T}_0$, how likely $\mathcal{T}_i$ is to include it, Eqn. 1.3b.

$$\text{precision} = \frac{TP}{TP + FP} \tag{1.3a}$$

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{1.3b}$$

Informally, precision prioritises that predicted terms are correct, while sensitivity prioritises that true terms are identified. In practice, it is important to balance these considerations. F_β-score is a measure which balances these, with F_1-score in particular giving them equal importance.

$$F_1 = \frac{2 \times (precision) \times (sensitivity)}{(precision + sensitivity)} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}. \tag{1.4}$$

We give an example of these quantities in Fig. 1.4, where $TP = 3, TN = 4, FP = 1, FN = 2$, giving $precision = 3/4$ and $sensitivity = 3/5$, with a final $f = 0.67$, i.e. the average of the indicators of model quality which we care about.

We adopt F_1-score as an indication of model quality because we are concerned both with precision and sensitivity of output models. We can use F_1-score to measure the success of the algorithm, by recording f for all models in all generations, allowing us to see whether or not the approximation of the system is improving on average.

Of course in realistic cases we can not assume knoweldge of $\mathcal{T}_o$ and therefore cannot compute F_1-score, but it is a useful tool in the development of the GES itself, or in cases where $\hat{H}_0$ is known, such as when the target system is simualated, e.g. in the case of device calibration. Our search for an effective OF can then be guided by seeking the method which correlates most strongly with F_1-score in test-cases.

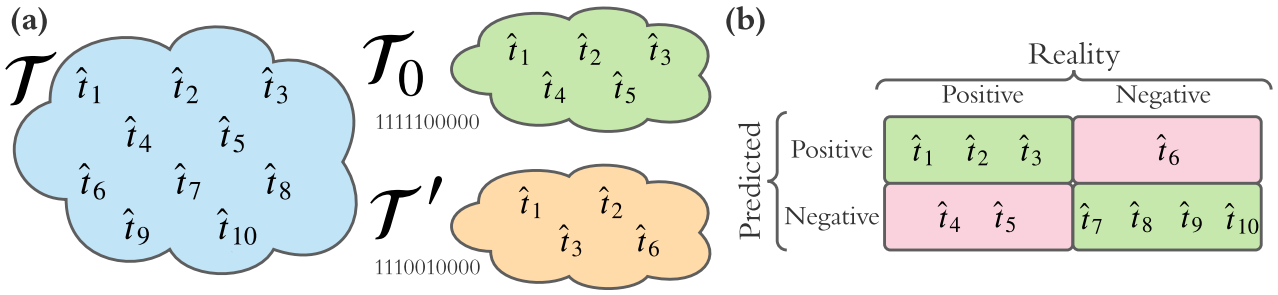

Figure 1.4: Concepts used for classification. **a**, the set of available terms $\mathcal{T}$ containing individual terms $\hat{t}_1$ to $\hat{t}_{10}$. The true model $\hat{H}_0$ is constructed from the set $\mathcal{T}_0$. Suppose a candidate $\hat{H}'$ has the set $\mathcal{T}'$. **b**, the confusion matrix for $\hat{H}'$. Correctly classified terms are true positives and true negatives (green), and incorrectly classified terms are false positives and true negatives (red).

Algorithm 2: ES subroutine: generate_models via genetic algorithm

Input: ν // information about models considered to date

Input: $g(\hat{H}_i)$ // objective function

Output: $\mathbb{H}$ // set of models

$N_m = |\nu|$ // number of models
for $\hat{H}_i \in \nu$ **do**
  $g_i \leftarrow g(\hat{H}_i)$ // model fitness via objective function
end
$r \leftarrow \text{rank}(\{g_i\})$ // rank models by their fitness
$\mathbb{H}_t \leftarrow \text{truncate}(r, \frac{N_m}{2})$ // truncate models by rank: only keep $\frac{N_m}{2}$
$s \leftarrow \text{normalise}(\{g_i\})\ \forall \hat{H}_i \in \mathbb{H}_t$ // normalise remaining models' fitness
$\mathbb{H} = \{\}$ // new batch of chromosomes/models
while $|\mathbb{H}| < N_m$ **do**
  $p_1, p_2 = \text{roulette}(s)$ // use s to select two parents via roulette selection
  $c_1, c_2 = \text{crossover}(p_1, p_2)$ // produce offspring models
  $c_1, c_2 = \text{mutate}(c_1, c_2)$ // probabilistically mutate
  $\mathbb{H} \leftarrow \mathbb{H} \cup \{c_1, c_2\}$ // add new models to batch
end
return $\mathbb{H}$

1.2.3 *Hyperparameter search*

Firstly we will validate our reasoning that F_1-score is a sensible figure of merit, by directly invoking it as the objective function. That is, we first implement a GA, using the mapping between models and chromosomes outlined above, where we fix the numbers of sites $d = 4$, and assume full connectivity between the sites, with $x-$, $y-$ and $z-$ couplings available, such that there are $N_t = 3 \times \binom{4}{2} = 18$ terms in $\mathcal{T}$, so that the total population is of size 2^{18} chromosomes. We can then sweep over the yperparameters to find a suitable configuration: in Fig. 1.5 we show how the choice of parameters affect the success rate of preciesly identifying the target chromosome, which is chosen at random for each instance, and we run 20 instance of each configuration. The studied hyperparameters[4] are

1. number of generations;
2. number of models per generation;
3. mutation rate, r_m;
4. number of generation a candidate must reign as the strongest observed, before the search terminates, the *cutoff*.

Naturally, we expect that running for more generations with more models per generation will result in a more effective search in the model space, having examined $N_g N_m$ models. We must also consider, however, that – in realistic cases of QMLA – the total computation time scales badly with these, since training and comparing models are such expensive subroutines. Our goal is therefore to identify the set of hyperparameters which best searches the model space with minmial N_g, N_m. We see that, unsurprisingly, the GA performs poorly when run with few resources, but broadly the performances are similiar provided it is run with sufficient resources. We can bound the parameters $r_m \geq 0.1, cutoff \geq 5, N_m \geq 16, N_g \geq 16$ to ensure a reasonable search through the model space, without having to consider a prohibitive number of models. We must bear in mind, however, that this parameter sweep refers only to the trivial case where the F_1-score is used as the OF, so we do not expect such high success rates in realistic cases.

1.3 OBJECTIVE FUNCTIONS

We have alluded to the central probelm in building a GA into QMLA: how to evaluate trained candidate models in the absence of a natural OF. Here we will propose and analyse a number of potential OFs, some of which will underlie later studies in this thesis. Readers may prefer to skip to Section 1.3.8, where we conclude this study by choosing a single OF for consideration in this chapter.

We will show how each OF computes g_i for candidate models $\hat{H}_i$, and summarise the outcomes in Table 1.4. For each $\hat{H}_i$, we may refer to

4 These and further hyperparameters can be swept using code given in the QMLA codebase, in the directory scripts/genetic_alg_param_sweep.

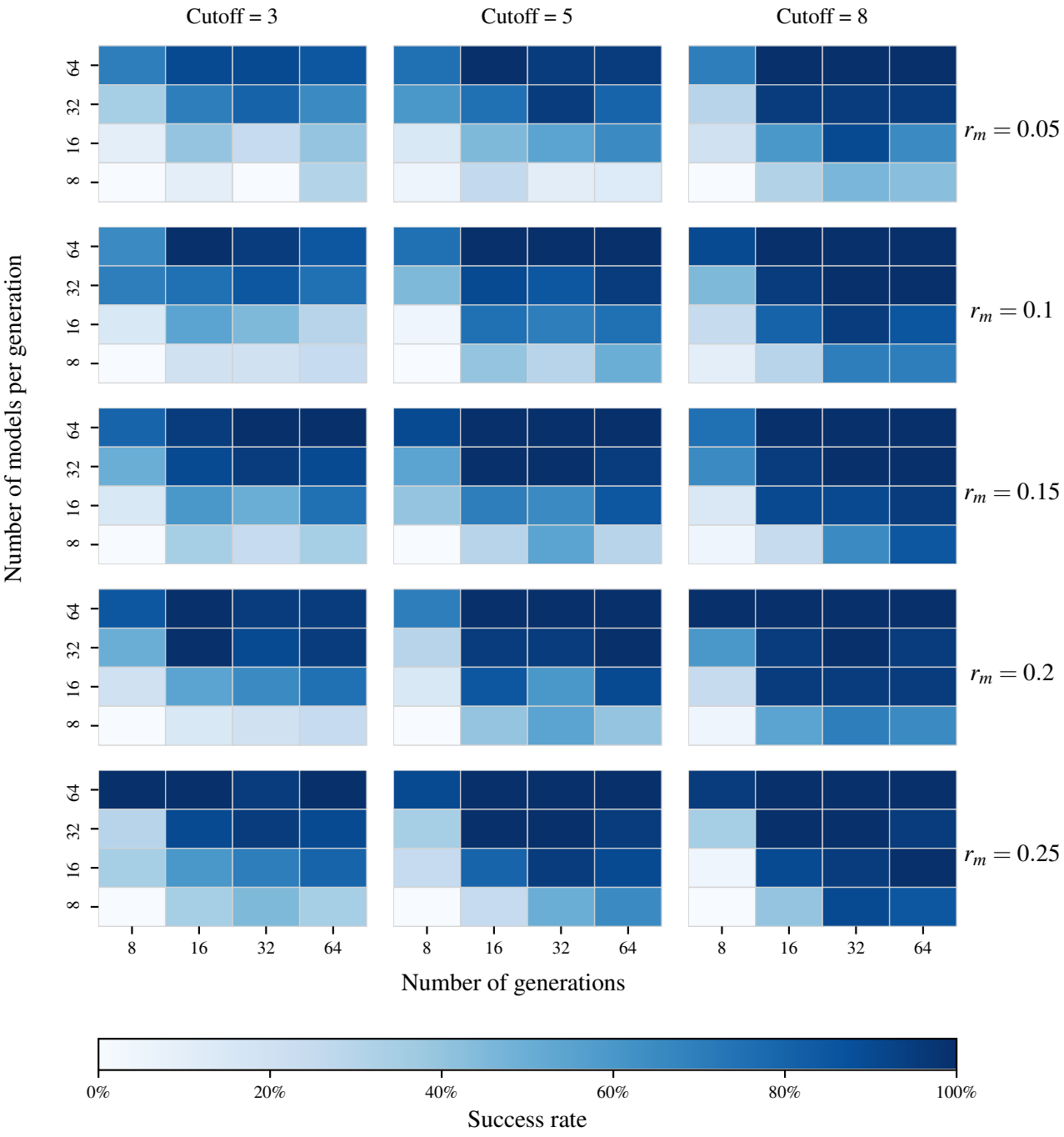

Figure 1.5: Genetic algorithm parameter sweep. Each subplot shows the success rates for varying numbers of generations, $N_G \in \{8, 16, 32, 64\}$, and numbers of models per generation, $N_m \in \{8, 16, 32, 64\}$. A subplot is generated for ranges of the mutation rate, r_m and the number of generations for which the elite model is unchanged after which the GA is cut off.

- $\mathcal{L}_i$, total log total likelihood (TLTL), introduced in **??**
- k_i: the model's cardinality, i.e. number of terms in its parameterisation;
- $\mathcal{E}_i$: the bespoke set of experiments composed by the experiment design heuristic (EDH) solely for trainng $\hat{H}_i$;
- $n = |\mathcal{E}_i|$: the number of samples used in trainng $\hat{H}_i$.

In Table 1.4, we consider six models as examples of the outcome using each OF; the models, randomly generated of varying quality with respect to the target $\hat{H}_0$, are

$$\begin{aligned}
\hat{H}_0 &= \sigma^z_{(1,2)}\sigma^z_{(1,3)}\sigma^z_{(2,3)}\sigma^z_{(2,5)}\sigma^z_{(3,5)};\\
\hat{H}_a &= \sigma^z_{(1,5)}\sigma^z_{(3,4)}\sigma^z_{(4,5)};\\
\hat{H}_b &= \sigma^z_{(1,4)}\sigma^z_{(1,5)}\sigma^z_{(2,5)}\sigma^z_{(3,4)};\\
\hat{H}_c &= \sigma^z_{(1,2)}\sigma^z_{(1,5)}\sigma^z_{(2,4)}\sigma^z_{(2,5)}\sigma^z_{(4,5)};\\
\hat{H}_d &= \sigma^z_{(1,3)}\sigma^z_{(1,4)}\sigma^z_{(1,5)}\sigma^z_{(2,4)}\sigma^z_{(2,5)}\sigma^z_{(3,4)}\sigma^z_{(3,5)};\\
\hat{H}_e &= \sigma^z_{(1,2)}\sigma^z_{(1,3)}\sigma^z_{(1,5)}\sigma^z_{(2,3)}\sigma^z_{(2,5)}\sigma^z_{(4,5)};\\
\hat{H}_f &= \sigma^z_{(1,2)}\sigma^z_{(1,3)}\sigma^z_{(2,3)}\sigma^z_{(2,4)}\sigma^z_{(2,5)}\sigma^z_{(3,4)}\sigma^z_{(3,5)}.
\end{aligned} \tag{1.5}$$

1.3.1 *Inverse Log-likelihood*

$\mathcal{L}_i$ can be thought of as a measure of the success of a given model at explaining data from any set of experiments, $\mathcal{E}$. This can be immediately interpreted as an OF, provided each candidate model computes a meaningful total log total likelihood (TLTL), requiring that they are all based on the same set of experiments, $\mathcal{E}_v$, which are designed explicitly for the purpose of model evaluation.

TLTL are negative and the strongest model has lowest $|\mathcal{L}_i|$ (or highest $\mathcal{L}_i$ overall), so the corresponding OF for candidate $\hat{H}_i$ is

$$g^L_i = \frac{-1}{\mathcal{L}_i}. \tag{1.6}$$

In our tests, Eqn. 1.6 is found to be too generous to poor models, assigning them non-negligible probability. Its primary flaw, however, is its reliance on $\mathcal{E}_v$: in order that the TLTL is significant, it must be based on meaningful experiments, the design of which can not be gauranteed in advance, or at least risks introducing strong bias.

1.3.2 *Akaike Information Criterion*

A common metric in model selection is Akaike information criterion (AIC) [24]. Incorporating TLTL, AIC objectively quantifies how well a given model accounts for data from the target

Method		$\hat{H}_a$	$\hat{H}_b$	$\hat{H}_c$	$\hat{H}_d$	$\hat{H}_e$	$\hat{H}_f$
	F_1	0.0	0.2	0.4	0.5	0.7	0.8
	k	3	4	5	7	6	7
	$\overline{l_e}$	0.86 ± 0.29	0.84 ± 0.29	0.77 ± 0.27	0.78 ± 0.29	0.79 ± 0.26	0.79 ± 0.26
	$\mathcal{L}_i$	-143	-152	-131	-150	-125	-124
Inverse log-likelihood	g_i^L	0.00698	0.00659	0.00766	0.00669	0.00803	0.00804
	%	23	0	25	0	26	26
Akaike Info Criterion	AIC	293	311	271	313	261	263
	AICc	293	312	272	314	262	264
	w_i^A	1.81e-07	1.4e-11	0.00724	4.15e-12	1	0.334
	g_i^A	1.17e-05	1.03e-05	1.35e-05	1.01e-05	1.46e-05	1.43e-05
	%	22	0	25	0	27	26
Bayesian Info Criterion	BIC	301	322	284	331	277	281
	w_i^B	5.49e-66	1.26e-70	1.97e-62	1.11e-72	8.43e-61	8.95e-62
	g_i^B	1.11e-05	9.65e-06	1.24e-05	9.11e-06	1.31e-05	1.27e-05
	%	23	0	25	0	27	26
Bayes factor points	g_i^p	0	2	3	2	3	5
	%	0	13	20	13	20	33
Ranking points	Ranking	6	5	2	4	3	1
	g_i^R	0	0	0.3	0.1	0.2	0.4
	%	0	0	30	10	20	40
Elo rating	Rating	909	944	1042	1007	1011	1084
	g_i^E	0	35	133	98	102	175
	%	0	0	26	19	20	34
Residuals	mean$\{\tilde{r_p^e}\}$	0.132	0.146	0.114	0.138	0.0858	0.0715
	g_i^r	0.753	0.729	0.785	0.743	0.836	0.862
	%	23	0	24	0	26	27

Table 1.4: Examples of how each objective function, g as described in Section 1.3.1 to Section 1.3.7, assign selection probability (denoted %) to the same set of candidate models, $\{\hat{H}_i\}$, when attempting to learn data from $\hat{H}_0$, listed in Eq. (1.5). For each model we first summarise its average likelihood l_e (Eqn. **??**), total log-likelihood $\mathcal{L}_i$ (Eqn. **??**), as well as F_1-score and number of terms k. We use $n = 250$ samples, i.e. $\mathcal{L}_i$ is a sum of n likelihoods. The set of models is truncated so that only the strongest four are assigned selection probability.

system, and punishes models which use extraneous parameters, by incurring a penalty on k_i. AIC is given by

$$AIC_i = 2k_i - 2\mathcal{L}_i. \tag{1.7}$$

In practice we use a slightly modified form of Eqn. 1.7 which corrects for the number of samples $n = |\mathcal{E}_i|$, called the Akaike information criterion corrected (AICC),

$$AICC_i = AIC_i + 2k_i \frac{k_i + 1}{n - k_i - 1}. \tag{1.8}$$

Model selection from a set of candidates occurs simply by selecting the model with $AICC_{\text{min}}$. A suggestion to retrieve selection probability, by using Eqn. 1.8 as a measure of *relative likelihood*, is to compute *Akaike weights* (as defined in in Chapter 2 of [24]),

$$w_i^A = exp\left(\frac{AICC_{\text{min}} - AICC_i}{2}\right), \tag{1.9}$$

where $AICC_{\text{min}}$ is the lowest $AICC$ observed among the models under consideration e.g. all models in a given generation.

Clearly, Akaike weights impose quite strong penalties on models which do not explain the data well, but also punish models with extra parameters, i.e. overfitting models, effectively searching for the strongest and simplest model simultaneously. The level of punishment for poorly performing models is likely too drastic: very few models will be in a range sufficiently close to $AICC_{\text{min}}$ to receive a meaningful Akaike weight, suppressing diversity in the model population. Indeed, we can see from Table 1.4 that this results in most models being assigned negligible weight, which is not useful for parent selection. Instead we compute a straightforward quantity as the AIC–inspired fitness, Eqn. 1.10,

$$g_i^A = \left(\frac{1}{AICc_i}\right)^2, \tag{1.10}$$

where we square the inverse AIC to amplify the difference in quality between models, such that stronger models are generously rewarded.

1.3.3 *Bayesian Information Criterion*

Related to the idea of AIC, Eqn. 1.7, is that of Bayesian information criterion (BIC),

$$BIC_i = k_i \, ln(n_i) - 2\mathcal{L}_i, \tag{1.11}$$

where k_i, n_i and $\mathcal{L}_i$ are as defined on Page 15. Analagously to Akaike weights, *Bayes weights* as proposed in §7.7 of [25], are given by

$$w_i^B = exp\left(-\frac{BIC_i}{2}\right). \tag{1.12}$$

BIC is harsher than AIC in its punishment of the number of parameters in each model, therefore requiring strong statistical justification for the addition of any parameters. Again, this may be overly cumbersome for our use case: with such a relatively small number of parameters, the punishment is disproportionate; moreover since we are trying to uncover physical interactions, we do not necessarily want to suppress models merely for their cardinality, since this might result in favouring simple models which do not capture the physics. As with Akaike weights, then, we opt instead for a simpler objective function,

$$g_i^B = \left(\frac{1}{BIC_i}\right)^2. \tag{1.13}$$

1.3.4 *Bayes factor points*

A cornerstone of model selection within QMLA is the calculation of Bayes factor (BF) (see **??**). We can compute the pairwise Bayes factor (BF) between two candidate models, B_{ij}, according to Eqn. **??**. B_{ij} can be based on some evaluation dataset, $\mathcal{E}_v$, but can also be calculated from $\mathcal{E}_i \cup \mathcal{E}_j$: this is a strong advantage since the resulting insight (Eqn. **??**) is based on experiments which were bespoke to both $\hat{H}_i, \hat{H}_j$. As such we can be confident that this insight accurately points us to the stronger of two candidate models.

We can utilise this facility by simply computing the BF between all pairs of models in a set of N_m candidates $\{\hat{H}_i\}$, i.e. compute $\binom{N_m}{2}$ BFs. Note that this is computationally expensive: in order to train $\hat{H}_i$ on $\mathcal{E}_j$ requires a further $\left|\mathcal{E}_j\right|$ experiments, each requiring N_P particles[5], where each particle corresponds to a unitary evolution and therefore the caluclation of a matrix exponential. The combinatorial scaling of the model space is then quite a heavy disadvantage. However, in the case where all pairwise BF are performed, we can assign a point to $\hat{H}_i$ for every comparison which favours it.

$$g_i^p = \sum_{j \in \mu} b_{ij}, \quad b_{ij} = \begin{cases} 1, & B_{ij} > 1 \\ 0, & \text{otherwise} \end{cases} \tag{1.14}$$

This is a straightforward mechanism, but is overly blunt because it does not account for the strength of the evidence in favour of each model. For example, a dominant model will receive only a slightly higher selection probability than the second strongest, even if the difference between them was $B_{ij} = 10^{100}$. Further, the unfavourable scaling make this an expensive method.

1.3.5 *Ranking*

Related to Section 1.3.4, we can rank models in a generation based on their number of BF points. BF points are assigned as in Eqn. 1.14, but instead of corresponding directly to fitness, we assign models a rank R, i.e. the model with highest g_i^p gets $R = 1$, and the model with n^{th} highest g_i^p

5 Caveat the reduction in overhead outlined in **??**.

gets $R = n$. Note here we truncate μ, meaning we remove the worse-performing models and retain only N'_m models, before calculating R. This is because computing R using all N_m models results in less distinct selection probabilities.

$$g_i^R = \frac{N'_m - R_i + 1}{\sum_{n=1}^{N'_m} n}, \tag{1.15}$$

where R_i is the rank of $\hat{H}_i$ and N'_m is the number of models retained after truncation.

1.3.6 *Residuals*

Recall at each experiment, N_P particles are compared against a single experimental datum, d. For consistency with QInfer [14] – on which QMLA's code base extends – we call the expectation value for the system $\Pr_Q(0)$, and that of each particle $\Pr_p(0)$, recall **??**. Typically, $\Pr_Q(0) = \left|\langle\psi| e^{-i\hat{H}_0 t} |\psi\rangle\right|^2$, but this can be changed to match given experimental schemes, e.g. the Hahn-echo sequence applied in [5]. By definition, the datum d is the binary outcome of the measurement on the system under experimental conditions e. That is, d encodes the answer to the question: after time t under Hamiltonian evolution, did Q project onto the basis we have labelled 0 (usually the same as the input probe state $|\psi\rangle$)? However, in practice we often have access also to the likelihood , i.e. rather than a binary value, a number representing the probability that Q will project on to 0 for a given experiment e, $\Pr_Q(0|e)$. Likewise, we can simulate this quantity for each particle, $\Pr_p(0|e)$. This allows us to calculate the *residual* between the system and individual particles' likelihoods, r_p^e, as well as the mean residual across all particles in a single experiment r^e:

$$\begin{aligned} r_p^e &= \left|Pr_Q(0|e) - Pr_p(0|e)\right| \\ r^e &= \underset{p}{\text{mean}}\{r_p^e\} \end{aligned} \tag{1.16}$$

Residuals capture how closely the particle distribution reproduced the dynamics from Q: $r_p^e = 0$ indicates perfect prediction, while $r_p^e = 1$ is completely incorrect. We can therefore maximise the quantity $1 - r$ to find the best model, using the OF

$$g_i^r = \left|1 - \underset{e\in\mathcal{E}}{\text{mean}}\{r^e\}\right|^2. \tag{1.17}$$

This OF can be thought of in frequentist terms as similar to the residual sum of squares, although instead of summing the residual squares, we average to ensure $0 \leq r \leq 1$. g_i^r encapsulates how well a candidate model can reproduce dynamics from the target system, as a proxy for whether that candidate describes the system. This is not always a safe figure of merit: in most cases, we do not expect parameter learning to perfectly optimise $\vec{\alpha}_i$. Reproduced

dynamics alone can not capture the likelihood that $\hat{H}_i = \hat{H}_0$. However, this OF provides a useful test for QMLA's GA: by simulating the case where parameters *are* learned perfectly, such that we know that g_i^r truly represents the ability of $\hat{H}_i$ to simulate $\hat{H}_0$, then this OF gaurantees to promote the strongest models, especially given that $\hat{H}_i = \hat{H}_0 \implies r_p^e = 0 \; \forall \; \{e, p\}$. In realistic cases, however, the non-zero residuals – even for strong $\hat{H}_i$ – may arise from imperfectly learned parameters, rendering the usefulness of this OF uncertain. Finally, it does not account for the cardinality, k_i, of the candidate models, which could result in favouring severely overfitting models in order to gain marginal improvement in residuals, which all machine learning protocols aim to avoid in general.

1.3.7 *Bayes factor enhanced Elo-ratings*

A popular tool for rating individual competitors in sports and games is the *Elo rating* scheme, e.g. used to rate chess players and soccer teams [**?**, 26], also finding application in the study of animal hierarchies [27]. Elo ratings allow for evaluating relative quality of individuals based on incomplete pairwise competitions, e.g. despite two football teams having never played against each other before, it is possible to quantify the difference in quality between those teams, and therefore to predict a result in advance [28]. We recognise a parallel with these types of competitions by noting that in our case, we similarly have a pool of individuals (models), which we can place in direct competition, and quantify the comparitive outcome through BF.

Elo ratings are transitive: given some interconnectivity in a generation, we need not compare *every* pair of models in order to make meaningful claims about which are strongest; it is sufficient to perform a subset of comparisons, ensuring each individual is tested robustly. We can take advantage of this transitivity to reduce the combinatorial overhead usually associated with computing bespoke BF between all models (i.e. using their own training data $\mathcal{E}_i$ instead of a generic $\mathcal{E}_v$). In practice, we map models within a generation to nodes on a graph, which is then sparsely connected. In composing the list of edges for this graph, we primarily prioritise each node having a similar number of edges, and secondarily the distance between any two nodes. For example, with 14 nodes we overlay edges such that each node is connected with 5,6 or 7 other nodes, and all nodes at least share a competitor in common.

The Elo rating scheme is as follows: upon creation, $\hat{H}_i$ is assigned a rating R_i; every comparison with a competitor $\hat{H}_j$ results in B_{ij}; R_i is updated according to the known strength of its competitor, R_j, as well as the result B_{ij}. The Elo update ensures that winning models are rewarded for defeating another model, but that the extent of that reward reflects the quality of its opponent. As such, this is a fairer mechanism than BF points, which award a point for every victory irrespective of the opposition: if $\hat{H}_j$ is already known to be a strong or poor model, then

ΔR_i proportionally changes the credence we assign to $\hat{H}_i$. It achieves this by first computing the *expected* result of a given comparison with respect to each model, based on the current ratings,

$$E_i = \frac{1}{1 + 10^{\frac{R_j - R_i}{400}}}; \tag{1.18a}$$

$$E_i + E_j = 1, \tag{1.18b}$$

Then, we find the binary *score* from the perspective of each model:

$$\begin{cases} B_{ij} > 1 & \Rightarrow \; S_i = 1; \; S_j = 0 \\ B_{ij} < 1 & \Rightarrow \; S_i = 0; \; S_j = 1 \end{cases} \tag{1.19}$$

which is used to determine the change to each model's rating:

$$\Delta R_i = \eta \times (S_i - E_i) \, . \tag{1.20}$$

An important detail is the choice of η, i.e. the *weight* of the change to the models' ratings. In standard Elo schemes this is a fixed constant, but here – taking inspiration from football ratings where η is the number of goals by which one team won – we weight the change by the strength of our belief in the outcome: $\eta \propto \left| B_{ij} \right|$. That is, similarly to the interpretation of Eqn. **??**, we use the evidence in favour of the winning model to transfer points from the loser to the winner, albeit we temper this by instead using $\eta = log_{10}(B_{ij})$, since BF can give very large numbers. In total, then, following the comparison between models $\hat{H}_i, \hat{H}_j$, we can perform the Elo rating update

$$R_i' = R_i + \log_{10}(B_{ij}) \left(S_i - \frac{1}{1 + 10^{\frac{R_j - R_i}{400}}} \right) . \tag{1.21}$$

This procedure is easiest to understand by following the example in Table 1.5.

Finally, it remains to select the starting rating R_i^0 to assign models upon creation. Although this choice is arbitrary, it can have a strong effect on the progression of the algorithm. Here we impose details specific to the QMLA GA: at each generation we admit the top two models automatically for consideration in the next generation, such that strongest models can stay alive in the population and ultimately win. These are called *elite* models, $\hat{H}_e^1, \hat{H}_e^2$. This poses the strong possibility for a form of generational wealth: if elite models have already existed for several generations, their Elo ratings will be higher than all alternatives by defintion. Therefore by maintaining a constant R_i^0, i.e. a model born at generation 12 gets the same R_i^0 as $\hat{H}_e^1$ – which was born several generations prior and has been winning BF comparisons ever since – we bias the GA to continue to favour the elite models. Instead, we would prefer that newly born models can overtake the Elo rating of elite models. We achieve this through an imprecise mechanism:

6 Note to achieve $B_{ij} = 10^{100} = e^{\mathcal{L}_i - \mathcal{L}_j} \implies \mathcal{L}_i - \mathcal{L}_j = ln(10^{100}) \approx 7$.

Model		R_i	E_i	S_i	B_{ij}	$\log_{10}(B_{ij})$	ΔR_i	R'_i
$\hat{H}_a > \hat{H}_b$	$\hat{H}_a$	1000	0.76	1	1e+100	100	0.24	1024.0
	$\hat{H}_b$	800	0.24	0	1e-100	100	-0.24	776.0
$\hat{H}_b > \hat{H}_a$	$\hat{H}_a$	1000	0.76	0	1e-100	100	-0.76	924.0
	$\hat{H}_b$	800	0.24	1	1e+100	100	0.76	876.0

Table 1.5: Example of Elo rating updates. We have two models, where $\hat{H}_a$ is initially believed to be a stronger candidate than $\hat{H}_b$, i.e. has a higher starting Elo rating. We show the effect of strong evidence[6] in favour of each model following BF comparison, $B_{ij} \sim 10^{100}$. In the case where $\hat{H}_a$ defeats $\hat{H}_b$, because this was so strongly expected given their initial ratings, the reward for $\hat{H}_a$ (and cost to $\hat{H}_b$) is relatively small, compared with the case where – contrary to prediction – $\hat{H}_b$ defeats $\hat{H}_a$.

newly born models are given the Elo rating of the second-most-elite model, $\hat{H}_e^2$. This performs three key functions:

i. new models are immediately *within range* of the elite models; if they perform well enough, they have a realistic and fair chance of overtaking them;

ii. the strongest model retains some of its advantage gained over previous generations – in order that the ratings are meaningful, there must be some advantage accrued over series of victories;

iii. $\hat{H}_e^2$ is allowed continue to compete, but has no advantage: in order to retain its status as elite, it must perform well *again* in this generation, so it can not simply rely on results from previous generations – against inferior opposition – to remain active in the gene pool.

Given the arbitrary scaling of the Elo rating scheme, and in order to derive a meaningful selection probability, we ought to ground the raw Elo rating somehow at each generation μ. We do this by subtracting the lowest rating among the entertained models, R^{μ}_{min}. This serves to ensure the range of remaining R_i is defined only by the difference between models as assessed within μ: a very strong model might have much higher R_i than its contemporaries, but that difference was earned exclusively by comparison within μ, so it deserves higher fitness. We perform this step before truncation, so that the reamining models post-truncation all have non-zero fitness. Finally, then, we name this OF the *Bayes-factor enhanced Elo rating*, whereby the fitness of each model is attained directly from its rating after undergoing Elo updates in the current generation minus the minimum rating of any model in the same generation μ,

$$g_i^E = R_i^{\mu} - R^{\mu}_{\text{min}}. \tag{1.22}$$

The advantage of this OF is that it gives a meaningful value on the absolute quality of every model, allowing us to determine the strongest, and importantly to find the relative strength

between models. Further, it exploits bespoke BFs, i.e. based on the considered models' individually designed $\mathcal{E}_i$, removing the impetus to design $\mathcal{E}_v$ which can evaluate models definitively. One disadvantage is that it does not explicitly punish models based on their cardinality, however this feature is partially embedded by adopting BF for the comparisons, which are known to protect against overfitting.

1.3.8 *Choice of objective function*

Having proposed a series of possible objective functions, we are now in a position to analyse which of those are most appropriate for QMLA. Recall from Section 1.2.2 the figure of merit we use for models, F_1-score, which we will use to distinguish between the outputs of each OF.

First we can remark on the examples listed in Table 1.4. The OFs which rely on the TLTL, i.e. g^L, g^A, g^B, g^r, are effectively tricked by the log likelihood, which appears reasonably convincing for poor models, e.g. $\hat{H}_a, \hat{H}_c$. This underlines the risk in buidling $\mathcal{E}_v$, which can be biased towards weak models, for example resulting in high selection probability for $\hat{H}_a$ which has $f = 0$, while $\hat{H}_d$, with $f = 0.4$ is discarded. On the other hand, OFs grounded by the BF (g^p, g^R, g^E) invariably promote models of higher F_1-score, justifying the role of statistical evidence used for those calculations.

Retuning to the task of determining our favoured OF, we choose some random target $\hat{H}_0$, and run a single generation using each OF, judging them by the quality of models they produce. We train the same batch of $N_m = 28$ random models in each case, and allow each OF to compute the selection probabilities for those models, and therefore direct the design of the hypothetical next generation of models. We plot the distribution of F_1-score that each OF produces in Fig. 1.6, also accounting for the time taken in each case, i.e. we report the time to train and evaluate the single generation on a 16–core node.

Overall, then, we can see that a strong balance of outcome with resource considerations are achieved by the Bayes factor enhanced Elo rating strategy, Section 1.3.7 so we use that for the case study presented in this chapter. We strongly emphasise, however, that the performance of each objective function can vary under alternative conditions, and therefore similar analysis may be warranted for future applications. For instance, if t_{max} is known to be small, in smaller model spaces, using g^r results in higher success rates. We retain Bayes factor enhanced Elo ratings (BFEER), however, for generality and novelty, but it is important to recognise that the results listed not reflect an upper limit of QMLA's performance, but rather reflect the constraints of the system under study; each Q will bring its own unique considerations which can result in significantly stronger or weaker performance. In particular, we will later use the OF based on residuals, Section 1.3.6, to study a much larger model space under assumptions of perfect parameter learning, **??**.

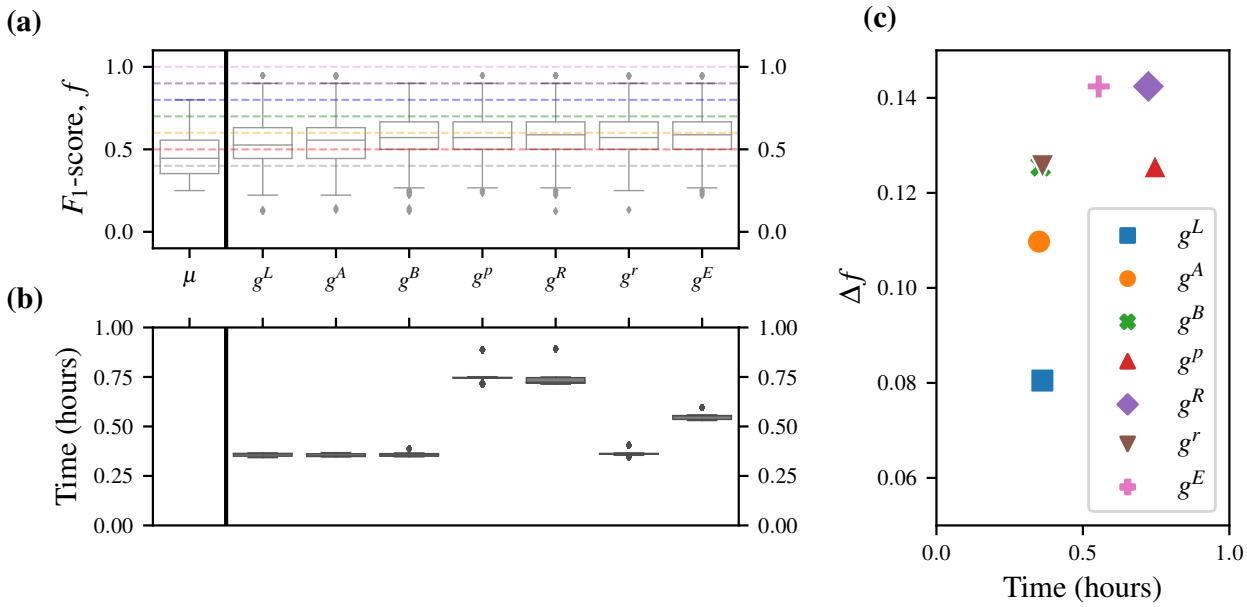

Figure 1.6: Comparison between proposed OFs. Each OF trains the same initial generation of $N_m = 28$ models with resources $N_E = 500, N_P = 2000$, and then design a new set of N_m models through the same roulette strategy, such that the only difference between OF's output is how they assign selection probability. We run each OF 25 times for the same target system, a 4–qubit Heisenberg–XYZ model. **(a)** shows the box–plot of new models' F_1-score, f, where the median and inter–quartile ranges are indicated by the boxes, as well as those of the initial generation μ centered on $f_\mu = 0.45$. We mark $f = \{0.4, 0.5, ..., 1.0\}$ for ease of interpretation. **(b)** shows box–plots of the time taken to compute the single generation in each case. In **(c)** we report the difference between the median f among the newly proposed models from f_μ, Δf, plotted against the time to achieve the result.

1.4 APPLICATION

Having introduced all the necessary concepts of GAs, mapped them to the QMLA framework and chosen a suitable OF, we can finally use the GES for model search.

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

iv. evaluation: plots of probes and times used as the evaluation dataset.

v. 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.

vi. 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.

vii. exploration_strategy_plots plots specifically required by the ES at the run level.

viii. 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.

ix. 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.

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

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

[3] 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.

[4] 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.

[5] 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.

[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] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[20] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[21] Sean Luke. 1 essentials of metaheuristicsl. 1.

[22] Lothar M Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1-2):1–61, 2001.

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

[24] Burnham KP Anderson DR. Model selection and multimodel inference: a practical information-theoretic approach, 2002.

[25] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[26] FIFA.com. Who we are - news - 2026 fifa world cup™: Fifa council designates bids for final voting by the fifa congress, Jun 2018.

[27] Christof Neumann, Julie Duboscq, Constance Dubuc, Andri Ginting, Ade Maulana Irwan, Muhammad Agil, Anja Widdig, and Antje Engelhardt. Assessing dominance hierarchies: validation and advantages of progressive evaluation with elo-rating. *Animal Behaviour*, 82(4):911–921, 2011.

[28] Lars Magnus Hvattum and Halvard Arntzen. Using elo ratings for match result prediction in association football. *International Journal of forecasting*, 26(3):460–470, 2010.