EPSRC Centre for Doctoral Training
Quantum Engineering

University of BRISTOL

Doctorate of Philosophy

Schrödinger's Catwalk

Brian Flynn

University of Bristol

November, 2020

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LISTINGS

ACRONYMS

ES	exploration strategy. 2, 4, 7
GA	genetic algorithm. 2
GES	genetic exploration strategy. 2
QHL	quantum Hamiltonian learning. 4
QMLA	Quantum Model Learning Agent. vii, 2, 4

GLOSSARY

objective funtion (OF)	Function which acts on a candidate in a genetic algorithm (GA) to assess the fitness of the candidate at solving the problem of interest. . 2
instance	a single implementation of the Quantum Model Learning Agent (QMLA) algorithm. 4
run	collection of QMLA instances. 4

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 [1]. In particular, we develop a exploration strategy (ES) which incorporates an 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 [2]. Here we will first introduce the concept of a 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 funtion (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.

1.2 ADAPTATION TO QMLA FRAMEWORK

1.3 OBJECTIVE FUNCTIONS

1.4 APPLICATION

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: $\gamma'$                         // strongest candidate

$\mu \leftarrow \text{sample}(\mathcal{P}, N_m)$
for $i \in 1, ..., N_g$ do
    for $\gamma_j \in \mathbb{S}$ 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 $\gamma'$
```

Algorithm 2: ES subroutine: generate_models (example: genetic algorithm)

```
Input: $\nu$                                        // 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
$r \leftarrow$ rank($\{g_i\}$)                      // rank models by their fitness
$\mathbb{H}_t \leftarrow$ truncate($r, \frac{N_m}{2}$)   // truncate models by rank: only keep $\frac{N_m}{2}$
$s \leftarrow$ 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
    $p_1, p_2$ = roulette($s$)                      // use $s$ to select two parents via roulette selection
    $c_1, c_2$ = crossover($p_1, p_2$)              // produce offspring models
    $c_1, c_2$ = mutate($c_1, c_2$)                 // probabilistically mutate
    $\mathbb{H} \leftarrow \mathbb{H} \cup \{c_1, c_2\}$   // add new models to batch
end
return $\mathbb{H}$
```

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
1. evaluation: plots of probes and times used as the evaluation dataset.
2. 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.
3. 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.
4. exploration_strategy_plots plots specifically required by the ES at the run level.
5. 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.
6. 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.
7. 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] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[2] 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.