Solving the Rubik's Cube with Learned Guidance Functions

Colin G. Johnson
School of Computing, University of Kent
Canterbury, Kent, United Kingdom; C.G.Johnson@kent.ac.uk

***Abstract*—This paper introduces *move sequence problems*—problems where a system can exist in a number of states, including a goal state, with moves between those states. This paper introduces *Learned Guidance Functions* (LGFs) as a machine learning method to tackle these. An LGF is a function learned by supervised machine learning that predicts how far a particular state is from the goal state. These methods are applied to the challenging problem of unscrambling a Rubik's Cube.**

***Index Terms*—Machine learning; error measures; loss functions; fitness landscapes; human-like computing**

I. Introduction

Consider a system that can exist in a large number of states, and, given a particular starting state, the task is to find a sequence of moves—transitions from state-to-state—that terminate in some goal state. Call these *move sequence problems* (MSPs). What makes these difficult for traditional machine learning approaches is that for each instance of the problem, the sequence of moves required will vary. It is not possible to simply learn a sequence of moves.

Examples of such problems include:

- Calculating the folded configuration of a protein. The states are configurations of the proteins, moves are changes between configurations, starting states are unfolded proteins (amino acid sequences), and goal states are folded proteins.
- Cleaning a distorted/noisy audio file: a recording made in a noisy environment, or one that has been deliberately distorted such as an audio CAPTCHA. The states are audio files, moves are the application of a filter or transformation to the audio, starting states are distorted audio files, and goal states are clean audio files.
- Similarly, cleaning a distorted/noisy image or video file.
- Mathematical problems such as taking a knot and working out the sequence of moves to unknot it. The states are knot diagrams, moves local changes to the knot, starting states knotted diagrams, and the goal state an unknotted diagram.
- Solving a puzzle where the player is given a random starting state and has to apply a sequence of moves to get to some configuration. The states are configurations of the puzzle, moves legal moves in the puzzle, stating states the initial state of the puzzle, and the goal state the solution to the puzzle.

In all of these cases we have a set of states, moves between the states, and we can compare two states in the dataset (but, we do not have an oracle that can compare two arbitrary states). Also, in these examples we have access to plentiful data.

This paper introduces *learned guidance functions* (LGFs). These do not learn the sequence directly, but learn the concept of how close a particular state is to being a goal. The LGF learns to map from states to labels (nonnegative integers), where 0 represents a goal state, 1 a state close to a goal state, 2 a state a little further away etc.; the idea is to learn to generalise this from a sample of labelled states. In this paper, which applies this approach to the Rubik's Cube, this discrete representation is appropriate, as these numbers represent the number of turns. In other problems, a continuous representation might be more appropriate.

Experimental evidence starting with the work of Rosch [1] has shown that human concept representations are graded, that we learn categories not as sharp boundaries but as continua from core examples of the concept to peripheral ones. As Gärdenfors [2] has emphasised, these "conceptual spaces" around a concept are not typically based around a single feature, but are multidimensional feature spaces. In solving complex problems, people move through this conceptual space, guided by the idea of closeness to the core concept; this work is inspired by this cognitive representation.

These LGFs are learned from *graded training sets* (GTSs). A GTS consists of a set of example states, each labelled with how close that state is to the goal state. This is not a simple error measure, but is sampled from trajectories from start state to goal state. The following are possible ways in which the graded labels can be obtained:

- Synthetically, by making sequences of reverse moves from known goal states.
- From known move sequences from experimental or observational studies, or move sequences provided by a domain expert.
- By the judgement of a domain expert on how close each of a set of examples are to the goal state.
- By applying problem-specific heuristics, or metaheuristics, to some instances the problem, particularly where these heuristics take longer in time than is needed in the final application, or where they can be applied to simple examples but not more complex ones.

The LGF is the model discovered by applying a classification algorithm to the GTS, with the aim of building a predictive model associating labels to states. This is called this a *guidance* function because it guides a search in the state-space

of the MSP. Most simply, this can be used in a simple hill-climbing approach: at each hillclimbing step, make a choice of move that results in a state that the LGF predicts to be closer to the goal state than the current state.

This paper shows how LGFs can be applied to the MSP of unscrambling the Rubik's Cube puzzle. Section II discusses some related techniques. Section III formalises the notion of MSP, GTS, and LGF in general, and then Section IV constructs Rubik's cube unscrambling as an MSP. Sections V and VI give experimental results, then analysis based on fitness landscapes is given in Section VII. Finally, Section VIII summarises the work, gives broader context, and suggests future work.

II. Related Work

The idea of LGFs has some similarity to the technique of *target analysis* (TA)—a technique for setting the parameters of a metaheuristic algorithm [3]. This takes a number of high quality solutions that have been generated by application of exhaustive or manual methods on small scale examples, or else examples where a large amount more effort is expended to solve them than would be practical in the final application. These are then used to tune the parameters of the meta-heuristic. This tuned metaheuristic is then applied to realistic examples. There are some similarities between TA and LGF approaches, in that both work from a set of solved examples as their initial material. In TA it is the parameters of the metaheuristic that is tuned. By contrast, the LGF approach tries to learn the concept of degrees of distance from the target state.

LGFs also share some ideas with *reinforcement learning* (RL) [4]: in particular, the idea of applying learning to create a set of local decision-making rules that produce a global behaviour. However, the information used for training is different. In RL the system is trained based on a reward function, which measures the value of actions taken. In LGF-based learning, the driver of learning is the LGF learned from the GTS.

A third related topic is the use of *evaluation functions* in game-playing AI—that is, functions that take a board state and evaluate the strength of that position. Both this and LGFs use the idea of attributing value to a particular system state and then making decisions about moves based on local optimisation of state. Evaluation functions can be built built as a combination of heuristics devised by expert players. For example, the IAGO Othello system [5] is driven by a weighted combination of four human-created features, with weights varying to emphasise the different importance of these factors at different times in the game; in the BILL system [5], similar features are used but the weights are learned from example games. This is the most similar to the approach taken in this paper. Other methods use an approach of learning features in the form of patterns of pieces on the board [6]. Another approach is to create a set of generic heuristics for a broad set of games [7].

One specific form of heuristic that has been applied to the Rubik's Cube is the *pattern database* [8]. These consist of databases of sets of partial states of the cube (for example, all corner pieces), and the set of moves required to solve that partial state, with the remaining faces considered to be a "don't care" state. The results from multiple pattern databases can then be combined to give an estimate of the number of moves needed to solve a particular combination, and thus provide a heuristic evaluation function to guide a search process.

Another problem that has similarities to the MSP is *super resolution* [9] enhancing a low-resolution image. Clearly, there is not enough information in the image alone, so these methods make use of additional information in the form of models of what is likely to be found in the image. For example, Dahl et al. [10] does this for faces, using one part of the to find the most likely baseline match from a large database, and another part to modify that baseline. This can be framed as an MSP problem—start with a large number of high-resolution faces, create a GTS from the intermediate resolution levels, and then learn an LGF from that GTS; this approach has not yet been tried.

Metric learning [11] is another related topic. This consists learning distances (usually similarity) between items in a space, from a set of examples where the distance has been given. LGFs can be seen as a kind of discrete distance learning, where discrete distances are being learned between points in the GTS and the set of goal states of the problem.

There are bespoke methods for specifically solving the Rubik's Cube [12]. The experiments in this paper are designed to show that a complex problem such as the Cube can be solved by a generic machine learning algorithm, together with a set of rules for the specific problem, not an approach that has been hand-tuned to the specific problem (the importance of AI methods being able to learn without explicit human knowledge has been emphasised in the recent paper by Silver et al. [13]).

In summary: a number of other methods are based on the idea of learning by generalising from solved examples of the problem at hand. However, the idea of explicitly learning the distance of instances from the target has not previously been applied. The closest to this is the use of machine learning to learn board evaluation functions.

III. Learned Guidance Functions

Define a *move sequence problem* (MSP) as a 3-tuple (S, M, G) consisting of a set S called the states, a set M of partial functions from S to S, called moves, and a subset $G \subset S$ of states called *goal states*.

Define a *graded training set* (GTS) as a set of states from a given MSP, each assigned a label from $\mathbb{Z}^{\geq 0}$, where all goal states in the GTS are mapped to 0, and no non-goal states are mapped to 0. Informally, the labels represent how far away from the goal state the state is.

A *learned guidance function* (LGF) is a function $L : S \to \mathbb{Z}^{\geq 0}$. This is a model obtained by applying a classification algorithm to the graded training set.

Once an LGF has been learned for a specific MSP, a process can be applied to solve a specific problem consisting of a pair of an MSP and a starting state $p \in S \setminus G$. Solving the problem

consists of finding a sequence of moves $m_0, m_1, \ldots, m_n$ such that $m_n(m_{n-1}(\ldots m_1(m_0(p))\ldots)) \in G$. This paper will focus on the use of simple hillclimbing processes to successively choose the moves. Such a simple approach is used is that the complexities of searching the set of states (e.g. local minima) are handled in this approach by the construction of the LGFs rather than by the search algorithm.

IV. LGFs Applied to the Rubik's Cube

The Rubik's Cube [12] is a combinatorial puzzle consisting of a cube broken into sub-cubes, with sides of different colours; the canonical cube has 3×3 squares on each face. The cube can be twisted in a number of ways, changing the colours on each side. In particular, each face can be turned clockwise or anticlockwise. Therefore, there are 12 basic moves, consisting of turning a face clockwise or anticlockwise by 90°; a well-known notation [14] notates the clockwise moves as members of the set $\{F, B, R, L, U, D\}$ where each letter corresponds to a face; a prime symbol ($'$) appended to the letter denotes an anticlockwise turn. These are called the *quarter turns*, and the *quarter turn metric* is the number of quarter turns carried out in a given sequence of moves.

Other notations exist to denote turning the middle layer, a 180° turn, and changing the position of the cube in space relative to the observer. However, the first two can be expressed as composites of the above basic moves, and the last is irrelevant to an algorithmic solution that can "see" all sides of the cube simultaneously.

A state of the cube is a specification of all of the coloured faces. The goal of a problem instance is to apply a sequence of moves to transform the cube from a given starting state to the single goal state, i.e. the state where the cube has the same colours on every side.

In the experiments in this paper a GTS is created synthetically by repeatedly starting from the goal state and applying a number of moves randomly, adding the state of the cube, labelled by the number of moves made, to the GTS after each move (Figure 1. This takes two input parameters: n_s is the number of times this process is carried out, and n_ℓ is the number of labels (i.e. the label set is $[0, \ldots, n_\ell - 1]$) (Algorithm 1)

Algorithm 1 Construct a GTS for the Rubik's cube

```
1: procedure GTSRubik(n_s, n_ℓ)
2:     M = {F, B, R, L, U, D, F', B', R', L', U', D'}
3:     T = ∅
4:     for s ∈ [0, ..., n_s − 1] do
5:         Let C be a new cube in the solved state
6:         for ℓ ∈ [0, ..., n_ℓ − 1] do
7:             T = T ∪ {(C, ℓ)}
8:             m = random selection from M
9:             C = m(C)
10:        end for
11:    end for
12:    return T
13: end procedure
```

One criticism of this procedure is that for the smaller values of ℓ, the number of states is small; indeed for $\ell = 0$, there is only one state (call it g), so the training set contains n_s instances of $(g, 0)$. However, it is well known that imbalanced class distributions can cause problems for classification algorithms [15], and so the duplicates are retained.

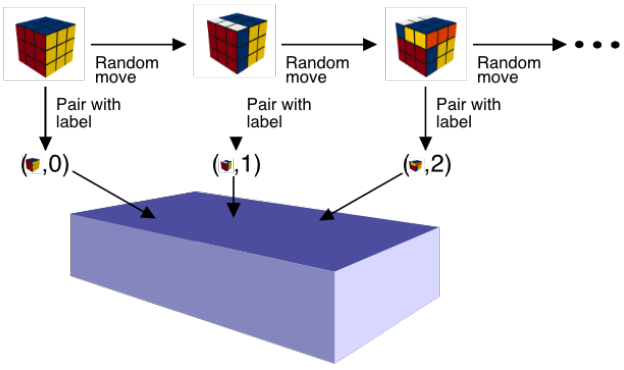

Fig. 1. Synthetic construction of the GTS for the Rubik's cube. This process is repeated n_s times.

A second criticism is that some move sequences can result in a state that could have been reached via a shorter sequence. As a simple example, the state reached by $F(B(U'(U(g))))$ is the same state reached by $F(B(g))$, but the former would be labelled with 4 by Algorithm 1. For a specific move sequence problem there might be a way to identify these cases, however there is no easy general method for identifying these. Ignoring these is in the spirit of investigating whether the LGF approach can learn naïvely from a basic problem description, without adding in additional domain knowledge. This issue is revisited in Section V-B.

V. Experiment 1: Can an LGF Accurately Learn from a GTS?

The first set of experiments tests a number of classification algorithms on the task of learning to label GTSs created by Algorithm 1.

A. Experiment 1a: Using scikit-learn Classifiers

The classification algorithms used are drawn from the well-known Python *scikit-learn* package [16]. Experiments were performed using version 0.18.2 of scikit-learn, using Python 3.6.2. The classification algorithms used in these experiments are detailed in Table I.

These classifiers were run on GTSs built with all combinations of the following parameter sets: $n_s \in \{100, 1000, 10000, 100000\}$ (number of labels per class) and $n_\ell \in [1, \ldots, 14]$ (largest class). If a classifier took more than 24 hours to run for all values of n_ℓ for a particular value of n_s, then these were stopped and left out of the tables. These were run using 10-fold cross-validation and the mean predictive accuracy is given in Table II. Also given is the time in seconds required to run the cross-validation experiments for each $(classifier, n_s)$ pair; this gives a broad sense of the comparative time required to train various classifiers. All

TABLE I
scikit-learn CLASSIFIERS USED IN THE EXPERIMENTS.

Method	**sklearn Name and Parameters**
Random Forests (RF)	`RandomForestClassifier(n_estimators=10, max_depth=20,min_samples_split=2)`
K-Nearest Neighbours (k-NN)	`KNeighborsClassifier(3)`
Support Vector Machine: linear kernel (Linear SVM)	`SVC(kernel="linear", C=0.025)`
Support Vector Machine: RBF kernel (RBF SVM)	`SVC(gamma=2, C=1)`
Decision Tree (DecTree)	`DecisionTreeClassifier(max_depth=5)`
AdaBoost	`AdaBoostClassifier()`
Gaussian Naïve Bayes (Gaussian NB)	`GaussianNB()`
Linear Discriminant Analysis (LDA)	`LinearDiscriminantAnalysis()`
Quadratic Discriminant Analysis (QDA)	`QuadraticDiscriminantAnalysis()`

experiments in the paper were run on a mid-2015 Retina iMac with a 3.3 GHz Intel Core i5 processor and 16GB memory.

In conclusion, these experiments give some *prima facie* evidence that enough accuracy is available to apply these these LGFs to the Rubik's Cube problem.

Experiment 1b: Using TPOT to Build a Bespoke Classifier

In this experiment, rather than choosing between specific classifiers from *scikit-learn*, a meta-learning approach is used. approach. Specifically, the TPOT [17], [18] system is used, which constructs *pipelines* of classifiers drawn from scikit-learn using an optimisation method, and optimises the parameters of the components of that pipeline. The idea is to produce a specific bespoke classification pipeline for a specific dataset.

In this case, TPOT is run using GTSs with $n_\ell \in [1, 14]$ and of $n_s \in \{100, 1000, 10000\}$, as in the experiments with single classifiers above. The meta-parameters are `generations`=5 and `population_size`=50.

The results are given in the TPOT sub-table of Table II. Due to TPOT being a meta-learning algorithm, and therefore needing to run with many different base classifiers, it was not practical to run this on very large training sets.

Final Classifier Choice

Based on the cross-validation results in Table II, the calculation of the LGFs for the remainder of the paper is based on the Random Forests classifier, trained using $n_s = 100,000$ except for the smaller cases of n_ℓ, which used: $n_\ell = 1, n_s = 100$; when $n_\ell = 2, n_s = 1000$; $n_\ell = 3, n_s = 10000$ (this was because, looking at Table II, these gave minimal improvement).

B. Analysis of Misclassifications

This section contains analysis of the misclassifications that are made by the classifier. A confusion matrix is presented in Table III for the Random Forests, with 9 labels, trained on 50,000 examples per label. These values were chosen as mid-range figures, applying a decent amount of learning effort to a problem of decent size.

There is an interesting pattern in Table III. For many of the rows, the values alternate in a high-low pattern. For example, for the row where the true value is 4, the highest predicted value is 4 (30,050 correct predictions); but, *none* of these have been confused for 3, whereas a large number (12,853) have been confused for label 2. One possible explanation for this is that many of the examples labelled 4 in the training set have a pair of moves that cancel each other, as discussed above.

To investigate this a little further, a new GTS set was constructed using similar the same basic procedure as Algorithm 1, but restricting the choice of of m so that it cannot be the inverse of the immediate previous state. The confusion matrix for the Random Forest classifier with with 9 labels, trained on 50,000 examples constructed using this revised GTS is shown in Table IV. It is notable that confusion with smaller labels is considerably reduced.

VI. Experiment 2: Can the LGF Solve the Cube?

This LGF can now be used to tackle the MSP of unscrambling the Rubik's Cube. The idea is to use the LGF as the objective function for a hillclimbing-style search process, with random moves if the guidance function provides no positive direction. For comparison, the same process with a simple error metric in place of the LGF is used; call this the *mismatch* error. This consists of counting the faces in the current state of the cube that are of a different colour from the solved state of the cube, with the optimum being that they are all matched.

Algorithm 2 is the unscrambling procedure. $Cube$ is the set of possible states of the cube. $E : Cube \to \mathbb{Z}^{\geq 0, < n_s}$ is the error function (either LGF or mismatch), n_s the number of labels in the model, n_p the number of random moves made to pose the problem, n_n the maximum number of random moves made when a downward hillclimbing move cannot be made, and G is the goal state. $M = \{F, B, R, L, U, D, F', B', R', L', U', D'\}$ is a set of functions from $Cube \to Cube$, representing the quarter-turn moves.

Algorithm 2 was run using all combinations $n_s, n_p \in [1, 14]$, with $n_n = 20$ (20 was determined by informal experimentation; in practice, the neutral moves were rarely beneficial; the range [1,14] was determined by practicality in terms of time taken to run experiments, and the lack of many positive results when the problems became more complex).

Table V presents the results. The unscrambling algorithm driven by the LGF is generally much more likely to solve the problem than the error-based approach. Also, using a model of size greater than $n_p + 1$ rarely has any beneficial effect.

The number of moves needed is surprisingly small; in particular, the mean number is often *smaller* than the problem size. This suggests that often the problem generated by carrying out n_p moves is actually solvable in a smaller number

TABLE II

10-FOLD CROSS VALIDATION PREDICTIVE ACCURACY RESULTS. ROWS REPRESENT NUMBER OF MOVES (n_ℓ) AND TIME IN SECONDS, COLUMNS REPRESENT NUMBER OF SAMPLES PER LABEL (n_s). THE FINAL SUB-TABLE SHOWS THE RESULTS FROM THE TPOT META-LEARNER.

Random Forests	**100**	**1000**	**10000**	**100000**
1	1.000	1.000	1.000	1.000
2	0.970	0.973	0.971	0.972
3	0.827	0.925	0.933	0.935
4	0.646	0.804	0.888	0.894
5	0.588	0.679	0.788	0.849
6	0.487	0.599	0.691	0.761
7	0.466	0.539	0.620	0.682
8	0.389	0.482	0.561	0.621
9	0.367	0.442	0.512	0.566
10	0.322	0.401	0.469	0.521
11	0.322	0.370	0.435	0.484
12	0.297	0.349	0.406	0.450
13	0.291	0.335	0.379	0.421
14	0.258	0.307	0.357	0.395
Time(sec)	*3.4*	*20.9*	*221.0*	*54471.4*

k-NN	**100**	**1000**	**10000**
1	1.000	1.000	1.000
2	0.800	0.968	0.972
3	0.615	0.854	0.930
4	0.576	0.695	0.813
5	0.442	0.597	0.687
6	0.411	0.508	0.604
7	0.355	0.458	0.536
8	0.323	0.416	0.488
9	0.299	0.379	0.442
10	0.262	0.337	0.403
11	0.242	0.320	0.378
12	0.232	0.302	0.350
13	0.218	0.278	0.322
14	0.217	0.255	0.304
Time(sec)	*1.3*	*73.9*	*10071.4*

AdaBoost	**100**	**1000**	**10000**
1	1.000	1.000	1.000
2	0.540	0.509	0.584
3	0.487	0.493	0.484
4	0.330	0.382	0.363
5	0.370	0.373	0.348
6	0.303	0.297	0.315
7	0.267	0.276	0.286
8	0.268	0.255	0.251
9	0.228	0.218	0.243
10	0.174	0.214	0.214
11	0.202	0.184	0.205
12	0.175	0.172	0.177
13	0.161	0.177	0.155
14	0.157	0.154	0.139
Time(sec)	*14.1*	*71.0*	*899.2*

Gaussian NB	**100**	**1000**	**10000**	**100000**
1	1.000	1.000	1.000	1.000
2	0.840	0.944	0.954	0.954
3	0.645	0.797	0.808	0.811
4	0.574	0.641	0.675	0.673
5	0.507	0.548	0.589	0.584
6	0.463	0.516	0.521	0.518
7	0.411	0.466	0.470	0.465
8	0.323	0.406	0.423	0.423
9	0.317	0.376	0.388	0.387
10	0.295	0.346	0.358	0.357
11	0.275	0.337	0.332	0.331
12	0.266	0.297	0.309	0.308
13	0.252	0.270	0.287	0.288
14	0.237	0.265	0.270	0.271
Time(sec)	*0.8*	*5.7*	*60.3*	*621.8*

LDA	**100**	**1000**	**10000**	**100000**
1	0.770	0.805	0.754	0.750
2	0.950	0.971	0.972	0.972
3	0.797	0.864	0.882	0.885
4	0.616	0.709	0.715	0.725
5	0.552	0.591	0.596	0.604
6	0.441	0.516	0.505	0.511
7	0.401	0.448	0.438	0.442
8	0.369	0.376	0.386	0.393
9	0.331	0.339	0.346	0.353
10	0.274	0.313	0.314	0.313
11	0.283	0.304	0.291	0.278
12	0.243	0.271	0.257	0.259
13	0.221	0.249	0.247	0.244
14	0.209	0.216	0.224	0.226
Time(sec)	*1.1*	*9.3*	*111.3*	*1237.4*

QDA	**100**	**1000**	**10000**	**100000**
1	0.500	0.500	0.500	0.500
2	0.333	0.333	0.333	0.333
3	0.250	0.250	0.250	0.250
4	0.200	0.200	0.200	0.200
5	0.167	0.167	0.167	0.167
6	0.143	0.143	0.143	0.143
7	0.125	0.125	0.125	0.125
8	0.111	0.111	0.111	0.111
9	0.100	0.100	0.100	0.100
10	0.091	0.091	0.091	0.091
11	0.083	0.083	0.083	0.083
12	0.077	0.077	0.077	0.077
13	0.071	0.071	0.071	0.071
14	0.067	0.067	0.067	0.067
Time(sec)	*1.5*	*7.5*	*83.9*	*1050.8*

Linear SVM	**100**	**1000**
1	1.000	1.000
2	0.950	0.958
3	0.812	0.881
4	0.590	0.743
5	0.547	0.649
6	0.497	0.566
7	0.446	0.501
8	0.400	0.462
9	0.365	0.418
10	0.335	0.389
11	0.307	0.355
12	0.270	0.332
13	0.274	0.311
14	0.259	0.288
Time(sec)	*8.3*	*719.1*

RBF SVM	**100**	**1000**
1	1.000	1.000
2	0.977	0.975
3	0.805	0.935
4	0.664	0.801
5	0.570	0.694
6	0.491	0.608
7	0.438	0.536
8	0.404	0.484
9	0.351	0.437
10	0.305	0.389
11	0.299	0.362
12	0.270	0.337
13	0.256	0.316
14	0.259	0.290
Time(sec)	*12.6*	*1122.8*

DecTree	**100**	**1000**	**10000**	**100000**
1	1.000	1.000	1.000	1.000
2	0.907	0.940	0.936	0.937
3	0.758	0.790	0.802	0.803
4	0.604	0.662	0.670	0.667
5	0.508	0.565	0.583	0.582
6	0.479	0.495	0.505	0.509
7	0.437	0.445	0.455	0.458
8	0.378	0.399	0.413	0.414
9	0.330	0.367	0.380	0.378
10	0.313	0.335	0.342	0.345
11	0.298	0.322	0.317	0.319
12	0.272	0.292	0.298	0.295
13	0.255	0.272	0.278	0.279
14	0.245	0.260	0.257	0.258
Time(sec)	*1.0*	*8.3*	*96.0*	*1506.6*

TPOT	**100**	**1000**
1	1.000	1.000
2	0.960	0.973
3	0.820	0.923
4	0.680	0.823
5	0.533	0.718
6	0.537	0.598
7	0.430	0.545
8	0.391	0.490
9	0.316	0.445
10	0.280	0.411
11	0.350	0.357
12	0.243	0.343
13	0.254	0.334
14	0.259	0.292
Time(sec)	*5994.6*	*46184.1*

TABLE III
CONFUSION MATRIX USING THE RANDOM FORESTS CLASSIFIER WITH $n_\ell = 8$

		Predicted Label								
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
True Value	**0**	50000	0	0	0	0	0	0	0	0
	1	0	50000	0	0	0	0	0	0	0
	2	4214	0	45786	0	0	0	0	0	0
	3	0	9179	0	40771	0	50	0	0	0
	4	743	0	12853	0	30050	79	5297	75	903
	5	0	2155	9	15393	620	14595	3307	10371	3550
	6	193	0	3607	11	14638	3523	9282	8251	10495
	7	0	541	7	5106	476	10524	6112	14180	13054
	8	41	0	1046	12	5708	3699	8749	12710	18035

TABLE IV
CONFUSION MATRIX USING THE RANDOM FORESTS CLASSIFIER WITH $n_\ell = 8$, CONSTRUCTED FROM A GTS WITH IMMEDIATE INVERSE MOVES DISALLOWED.

		Predicted Label								
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
True Value	**0**	50000	0	0	0	0	0	0	0	0
	1	0	50000	0	0	0	0	0	0	0
	2	0	0	50000	0	0	0	0	0	0
	3	0	1235	0	48765	0	0	0	0	0
	4	113	0	2468	17	46225	223	776	27	151
	5	0	281	6	3932	1837	29794	5580	6016	2554
	6	14	1	494	64	6335	7855	14758	10186	10293
	7	0	40	2	793	710	9182	10443	13608	15222
	8	2	0	84	26	1488	4442	10391	14308	19259

of moves. This is due to cancellations between pairs (or more) of successive moves. To investigate this, the experiments were re-run using variants on the data-generation method from Algorithm 1 and Algorithm 2, the unscrambling algorithm, where the algorithm disallowed the second of a pair of moves where the second member of the pair was the inverse of the first. Results are presented in Table VI. Note that many states generated from a larger number of moves can still be solved in a simpler number of moves, though the effect is weaker. Creating problems to test these methods is future work.

Algorithm 2 Unscramble the Rubik's cube

procedure GTSRUBIK(E, n_s, n_p, n_n)
  $M = \{F, B, R, L, U, D, F', B', R', L', U', D'\}$
  Let C be a new cube
  for $i \in [0, n_p)$ **do**
    $m =$ random selection from M
    $C = m(C)$
  end for
  $t = 0$
  for $i \in [0, n_n)$ **do**
    $M' = \text{shuffle}(M)$
    for $m \in M$ **do**
      $D = C$
      $D = m(D)$
      if $E(D) < E(C)$ **then**
        C = D
        $t = t + 1$
        Break
      end if
    end for
    if $C == G$ **then**
      return $(True, t)$
    end if
    $m =$ random selection from M
    $C = m(C)$
  end for
  return $(False, t)$
end procedure

VII. LANDSCAPE ANALYSIS

A conjecture for the superior performance of the LGF over the naive error metric is that it provides a smoother path from points in the search space to the global minimum. If the LGF were able to make perfect predictions, there would be no local minima: the hillclimber would start at a state that was m moves away from the target, iterate through possible local moves until one $m - 1$ moves away was found, move to that state, and iterate to the goal state.

A landscape analysis algorithm (Algorithm 3) was implemented. This starts by generating a solved cube C and scrambling it using n_p quarter turns; this is added to the empty list L. All possible moves are tried on this cube, and any that have an error E (depending on the experiment, LGF error or mismatch error) less than C are added to the list L. This process then iterates: each cube from L is popped from the list, and all possible moves tried. In the end, either a solved cube is found (return *True*), or L becomes empty because all possibilities (including backtracks) have been explored, meaning that there is no route from the scrambled state to the solved state by following that error metric.

Results are in Table VII. Each entry in the table is the result of running Algorithm 3 100 times, reporting the number of times the algorithm returned a *True* result, i.e. the error measure used included a route from the scrambled state to the solved state. This was run from $n_p = 2$ to $n_p = 13$ (not 14, because the LGF experiments used model size $n_p + 1$ and 14 was the largest model size). These experiments did not use the variant where inverses are prevented. It is notable that many more runs of the LGF version had a monotonically decreasing route to the goal state compared to the mismatch.

Algorithm 3 Landscape Analysis. Parameters/variables have the same meaning as in Algorithm 2. Model size for the LGF is $n_p + 1$.

procedure GTSRUBIK(E, n_p)
  $M = \{F, B, R, L, U, D, F', B', R', L', U', D'\}$
  Let C be a new cube
  for $i \in [0, n_p)$ **do**
    $m =$ random selection from M
    $C = m(C)$
  end for
  Let L be a new list
  $L = [C]$
  while $L \neq \emptyset$ **do**
    for $m \in M$ **do**
      $C = \text{pop}(L)$
      $D = m(C)$
      if $E(D) < E(C)$ **then**
        $L = L \cup \{D\}$
      end if
      if $D == G$ **then**
        return $True$
      end if
    end for
  end while
  return $False$
end procedure

TABLE V

UNSCRAMBLING EXPERIMENTS. *Size of Problem* INDICATES THE NUMBER OF MOVES USED TO CONSTRUCT THE PROBLEM. *Size of Model* INDICATES THE MAXIMUM LABEL SIZE USED TO TRAIN THE LGF; SUCCESS RATES USING THE MISMATCH ERROR METRIC ARE GIVEN IN THE ROW *Error*. ENTRIES IN (A) ARE PERCENTAGE OF SUCCESSFUL TRIALS; IN (B), MEAN NUMBER OF MOVES NEEDED TO REACH THE GOAL STATE.

a) Percentage of successful unscrambling trials:

Size of Model	Size of Problem 1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	98	-	-	-	-	-	-	-	-	-	-	-	-	-
2	100	96	-	-	-	-	-	-	-	-	-	-	-	-
3	100	100	97	-	-	-	-	-	-	-	-	-	-	-
4	100	100	100	99	-	-	-	-	-	-	-	-	-	-
5	100	100	100	100	90	-	-	-	-	-	-	-	-	-
6	100	100	100	98	83	55	-	-	-	-	-	-	-	-
7	100	100	100	94	76	55	39	-	-	-	-	-	-	-
8	100	100	100	94	77	58	31	27	-	-	-	-	-	-
9	100	100	100	95	80	56	34	23	23	-	-	-	-	-
10	100	100	100	92	76	52	36	26	22	11	-	-	-	-
11	100	100	100	97	82	57	36	27	9	12	5	-	-	-
12	100	100	100	96	75	53	35	29	16	5	7	2	-	-
13	100	100	100	98	75	62	45	20	17	11	9	7	0	-
14	100	100	100	98	76	55	36	21	16	10	7	3	3	1
Error	100	62	33	24	10	4	3	2	0	0	0	1	0	0

b) Mean number of moves needed to reach goal state for successful trials:

Size of Model	Size of Problem 1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1.592	-	-	-	-	-	-	-	-	-	-	-	-	-
2	1.000	1.958	-	-	-	-	-	-	-	-	-	-	-	-
3	1.000	1.840	2.670	-	-	-	-	-	-	-	-	-	-	-
4	1.000	1.860	2.680	3.697	-	-	-	-	-	-	-	-	-	-
5	1.000	1.740	2.680	3.500	4.467	-	-	-	-	-	-	-	-	-
6	1.000	1.860	2.660	3.592	4.229	4.291	-	-	-	-	-	-	-	-
7	1.000	1.860	2.500	3.638	4.026	4.545	4.846	-	-	-	-	-	-	-
8	1.000	1.880	2.420	3.553	4.091	4.793	4.161	4.667	-	-	-	-	-	-
9	1.000	1.800	2.720	3.474	3.825	4.643	4.647	4.957	5.087	-	-	-	-	-
10	1.000	1.840	2.500	3.413	4.289	4.154	4.333	4.692	5.545	5.455	-	-	-	-
11	1.000	1.780	2.760	3.320	3.878	4.596	4.667	4.741	6.111	4.500	5.400	-	-	-
12	1.000	1.800	2.600	3.625	3.987	4.679	4.371	5.034	4.875	6.800	5.571	5.000	-	-
13	1.000	1.780	2.560	3.286	3.853	4.161	4.511	4.800	5.471	4.909	4.556	4.857	-	-
14	1.000	1.780	2.680	3.694	4.079	4.509	4.778	5.238	4.750	6.000	5.286	4.667	5.000	6.000
Error	1.000	2.065	2.152	2.333	2.800	4.500	3.667	2.000	-	-	-	4.000	-	-

VIII. CONCLUSIONS AND FUTURE WORK

This paper has presented the idea of LGFs, and demonstrated on a complex problem how they can learn. This demonstrates a richer use of training material than traditional machine learning approaches.

One advantage to this learning method is that learning happens prior to the application of the model to a specific problem. Other approaches to this use a goal-directed search process with mechanisms to avoid local optima. For LGFs, the means of avoiding the local optima is part of the process of learning the LGFs, rather than solving a specific problem. This is in the tradition of ideas such as geometric semantic genetic programming [19], which re-configure the fitness landscape so that e.g. hill-climbing can be used.

This work also fits into the ideas explored by Krawiec and Swan [20], [21] of using complexity measures from machine learned models as a driver for search. Also similar is Monte Carlo tree search [22], which uses random forward explorations from a state as a way of estimating its value.

This work fits into a broad direction in machine learning by increasing the size of the *set of verbs*: classification asks "what is this?"; metric learning "how similar are these two things?"; clustering "how do these things divide up?", etc. LGFs add the new question "how close are we to the goal state?".

There are a number of algorithm improvements. One is using backtracking (Section VII) in the search algorithm. Experimenting with other search processes to avoid local minima caused by inaccurate assignment of states to grades is another. Another would be learning qualitative differences between states: assigning a label from $\{closer, same, further\}$ to pairs of states. This could draw on methods from the literature on *learning to rank* [23]. Also, connections between the learning methods used in this paper and metric and similarity learning [11]. Finally, experiments with class-imbalance correction approaches and feature-selection algorithms.

Code for the experiments:
`https://www.cs.kent.ac.uk/people/staff/cgj/software/IEEE_SSCI_2018/cube.py`

REFERENCES

[1] E. Rosch, "Cognitive representations of semantic categories," *Journal of Experimental Psychology*, vol. 104, no. 3, pp. 192–233, 1975.

[2] P. Gärdenfors, *The Geometry of Meaning: Semantics Based on Conceptual Spaces*. MIT Press, 2014.

[3] F. Glover and H. Greenberg, "New approaches for heuristic search: A bilateral linkage with artificial intelligence," *European Journal of Operational Research*, vol. 39, no. 2, pp. 119–130, 1989.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[5] M. Buro, "The evolution of strong othello programs," in *Entertainment Computing: Technologies and Application*, R. Nakatsu and J. Hoshino, Eds. Springer US, 2003, pp. 81–88.

TABLE VI

UNSCRAMBLING EXPERIMENTS WHERE THE PROBLEMS WERE CONSTRUCTED AVOIDING TWO SUCCESSIVE MOVES THAT CANCEL EACH OTHER OUT. *Size of Problem* INDICATES THE NUMBER OF MOVES USED TO CONSTRUCT THE PROBLEM. *Size of Model* INDICATES THE MAXIMUM LABEL SIZE USED TO TRAIN THE LGF; SUCCESS RATES USING THE MISMATCH ERROR METRIC ARE GIVEN IN THE ROW *Error*. ENTRIES IN (A) ARE PERCENTAGE OF SUCCESSFUL TRIALS; IN (B), MEAN NUMBER OF MOVES NEEDED TO REACH THE GOAL STATE.

a) Percentage of successful unscrambling trials:

Size of Model	Size of Problem 1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	96	-	-	-	-	-	-	-	-	-	-	-	-	-
2	100	100	-	-	-	-	-	-	-	-	-	-	-	-
3	100	100	96	-	-	-	-	-	-	-	-	-	-	-
4	100	100	100	99	-	-	-	-	-	-	-	-	-	-
5	100	100	100	99	95	-	-	-	-	-	-	-	-	-
6	100	100	100	99	92	54	-	-	-	-	-	-	-	-
7	100	100	100	98	85	47	14	-	-	-	-	-	-	-
8	100	100	100	97	82	38	16	9	-	-	-	-	-	-
9	100	100	100	98	82	39	23	11	4	-	-	-	-	-
10	100	100	100	97	80	31	18	9	3	6	-	-	-	-
11	100	100	100	99	74	36	20	11	7	3	1	-	-	-
12	100	100	100	98	81	37	19	8	1	2	0	0	-	-
13	100	100	100	100	81	39	19	9	2	1	0	0	1	-
14	100	100	100	96	73	34	13	4	5	1	0	0	0	0
Error	100	54	23	14	3	0	0	0	0	0	0	0	0	0

b) Mean number of moves needed to reach goal state for successful trials:

Size of Model	Size of Problem 1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1.146	-	-	-	-	-	-	-	-	-	-	-	-	-
2	1.000	2.200	-	-	-	-	-	-	-	-	-	-	-	-
3	1.000	2.000	2.875	-	-	-	-	-	-	-	-	-	-	-
4	1.000	2.000	2.980	3.960	-	-	-	-	-	-	-	-	-	-
5	1.000	2.000	2.940	3.960	5.126	-	-	-	-	-	-	-	-	-
6	1.000	2.000	2.980	3.899	4.761	5.667	-	-	-	-	-	-	-	-
7	1.000	2.000	2.940	3.898	4.929	5.660	6.143	-	-	-	-	-	-	-
8	1.000	2.000	3.000	3.835	4.780	5.526	5.750	6.667	-	-	-	-	-	-
9	1.000	2.000	2.960	3.898	4.902	5.641	6.043	6.182	5.000	-	-	-	-	-
10	1.000	2.000	2.980	3.918	4.900	5.935	6.111	5.778	5.667	6.000	-	-	-	-
11	1.000	2.000	3.000	3.919	4.730	5.444	6.400	6.000	5.571	6.000	5.000	-	-	-
12	1.000	2.000	2.980	3.898	4.704	5.622	6.263	5.750	5.000	7.000	-	-	-	-
13	1.000	2.000	2.940	3.960	4.827	5.897	6.684	6.889	9.000	8.000	-	-	7.000	-
14	1.000	2.000	2.940	3.979	4.945	5.412	6.385	5.500	7.400	6.000	-	-	-	-
Error	1.000	2.000	4.652	3.286	5.667	-	-	-	-	-	-	-	-	-

TABLE VII

LANDSCAPE ANALYSIS EXPERIMENTS. ENTRIES INDICATE THE NUMBER OF TIMES OUT OF 100 RUNS THAT ALGORITHM 3 RETURNED A *True* RESULT, I.E. THE ERROR MEASURE USED INCLUDED AT LEAST ONE ROUTE FROM THE SCRAMBLED STATE TO THE SOLVED STATE.

	Size of Problem 2	3	4	5	6	7	8	9	10	11	12	13
LGF	94	100	98	96	82	59	46	33	29	13	15	6
Error	82	78	69	43	39	28	22	12	7	3	2	5

[6] ——, "Statistical feature combination for the evaluation of game positions," *Journal of Artificial Intelligence Research*, vol. 3, pp. 373–382, 1995.

[7] J. Clune, "Heuristic evaluation functions for general game playing," in *Proceedings of the Twenty-second AAAI Conference on Artificial Intelligence*, R. C. Holte and A. E. Howe, Eds., 2007, pp. 1134–1139.

[8] R. E. Korf, "Finding optimal solutions to rubik's cube using pattern databases," in *Proceedings of the Fourteenth Nation Conference on Artificial Intelligence*. AAAI, 1997, pp. 700–705.

[9] K. Nasrollahi and T. B. Moeslund, "Super-resolution: a comprehensive survey," *Machine Vision and Applications*, vol. 25, no. 6, pp. 1423–1468, 2014.

[10] R. Dahl, M. Norouzi, and J. Shlens, "Pixel Recursive Super Resolution," *ArXiv e-prints*, Feb. 2017. [Online]. Available: https://arxiv.org/abs/1702.00783

[11] B. Kulis, "Metric learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2013.

[12] J. Slocum *et al.*, *The Cube: The Ultimate Guide to the World's Best-Selling Puzzle*. Black Dog and Leventhal, 2011.

[13] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[14] D. Singmaster, *Notes on Rubik's Magic Cube*. Hillside, NJ: Enslow Publishing, 1981.

[15] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sept 2009.

[16] "scikit-learn: Machine learning in Python." [Online]. Available: http://scikit-learn.org/

[17] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. New York, NY, USA: ACM, 2016, pp. 485–492.

[18] R. S. Olson *et al.*, "Automating biomedical data science through tree-based pipeline optimization," in *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Part I*, G. Squillero and P. Burelli, Eds. Springer, 2016, pp. 123–137.

[19] A. Moraglio, K. Krawiec, and C. G. Johnson, "Geometric semantic genetic programming," in *Parallel Problem Solving from Nature - PPSN XII: 12th International Conference, Part I*, C. A. Coello Coello *et al.*, Eds. Springer, 2012, pp. 21–31.

[20] K. Krawiec and J. Swan, "Pattern-guided genetic programming," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2013, pp. 949–956.

[21] K. Krawiec, *Behavioural Program Synthesis with Genetic Programming*. Springer, 2016.

[22] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Transactions On Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[23] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *Journal of Artificial Intelligence Research*, vol. 10, pp. 243–270, 1999.