Evolutionary Nonlinear Projection

Daniel Ashlock and Andrew McEachern

***Abstract*—This paper examines evolutionary nonlinear projection (NLP), a form of multidimensional scaling (MDS) performed with an evolutionary algorithm. MDS is a family of techniques for producing a low dimensional data set whose points have a one-to-one correspondence with the points of a higher dimensional data set with the added property that distances or dissimilarities in the higher dimensional space are preserved as much as possible in the lower dimensional space. The goal is typically visualization but may also be clustering or other forms of analysis. In this paper, we review current methods of NLP and go on to characterize NLP as an evolutionary computation problem, gaining insight into MDS as an optimization problem. Two different mutation operators, one introduced in this paper, are compared and parameter studies are performed on mutation rate and population size. The new mutation operator is found to be superior. NLP is found to be a problem where small population sizes exhibit superior performance. It is demonstrated experimentally that NLP is a multimodal optimization problem. Two broad classes of projection problems are identified, one of which yields consistent high-quality results and the other of which has many optima, all of low quality. A number of applications of the technique are presented, including projections of feature vectors for polyominos, of vectors that are members of an error correcting code, of behavioral assessments of a collection of agents, and of features derived from DNA sequences.**

***Index Terms*—Evolutionary computation, multidimensional scaling (MDS), variation operators, visualization.**

I. Introduction

NONLINEAR projection (NLP) is a form of multidimensional scaling (MDS) performed with evolutionary computation. A recent survey of techniques for MDS is given in [14], which offers many fast heuristics and variations on MDS. Visualization of high dimensional data, such as biomedical and image processing data, is important and requires sophisticated computational techniques. Dimensionality reduction is often core to these techniques, and is necessary to transform the high dimensional data into data in a more manageable number of dimensions. NLP is a flexible, generic method for performing this task.

The goal of NLP is to provide a relatively faithful projection of points from a higher dimensional space into a 2-D space that distorts the interpoint distances as little as possible. In [5], a simple evolutionary technique for visualizing data, an early version of NLP, was presented. The earlier form of NLP used evolutionary computation to minimize the squared error between the distance matrix of the original data and the Euclidean distance matrix of the projection of the points in 2-D. The representation is a simple list of point coordinates and the problem is treated as a standard real-valued evolutionary optimization.

One problem with minimizing the squared error between original and projected distances is that the optimizer must get the scale of the problem right. Estimating the scale of the data is possible but is an added and, as it turns out, unnecessary complication. In this paper, we instead maximize the Pearson correlation coefficient, given in (1), of the original distances with distances of points that are within the unit square. Pearson correlation of distance matrices is invariant under translation, rotation, reflection across a line, and scaling of either of the data sets whose correlation coefficient is being given and so permits the evolutionary algorithm to solve the problem of relative distance without worrying about other quantities. The Pearson correlation coefficient is given by

$$\text{cor} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})\,(y_i - \bar{y})}{(n-1)s_x s_y} \tag{1}$$

where for $z \in \{x, y\}$, $\bar{z}$ denotes the sample mean and s_z denotes the sample standard deviation. We note that maximizing the Person correlation is computationally equivalent to interval MDS minimizing the stress loss function for MDS, though the two functions taken as stress measures produce substantially different visualizations. The stress loss function and other potential fitness functions may be found in [14], which is an excellent introduction to MDS. We chose to use the Pearson correlation, as it is much more widely known to members of the evolutionary computation community.

In this paper, the set of points denoted by $\{x_i : i \in I\}$, for some index set I, are points in a high dimensional space. The projected points in 2-D Euclidean space are denoted by $\{y_i : i \in I\}$. The Pearson correlation coefficient determines how faithful the distances in the projection are to the actual distances between the points in the higher dimensional space. In this paper, evolution is used to discover projections that maximize the Pearson correlation coefficient.

A. MDS

MDS is a family of methods that analyzes proximity data on pairs of objects [14]. Classical MDS is one of the simplest and oldest MDS methods available, as it is based on a simple eigen decomposition. In [13], it was shown that it has a close relation with principal components analysis, a technique to be described later. Proximity information can consist of the similarity or dissimilarity between those pairs. If the

Manuscript received April 9, 2014; revised October 7, 2014; accepted November 24, 2014. Date of publication January 23, 2015; date of current version November 25, 2015. This work was supported by the National Science and Engineering Research Council of Canada.

The authors are with the Department of Mathematics and Statistics, University of Guelph, Guelph, ON N1G 2W1, Canada (e-mail: dashlock@uoguelph.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2015.2395091

data is given as similarities, some monotone decreasing transformation will convert them into dissimilarities. Dissimilarities between objects i and j can be assumed to have a value $d_{i,j}$. The goal of MDS is to map a set of objects $i = 1, \ldots, n$ to points $x_1, \ldots, x_n \in \Re^k$ such that the dissimilarities $d_{i,j}$ are well approximated by the distances $\|x_i - x_j\|$, denoted as $\delta_{i,j}$ in the embedding space. High dimensional data often contains many data points and often some error or bias with regard to the fitted distances and the original dissimilarities. Thus, a measure of fit by the minimization of a cost function called "Stress," given by Kruskal, has become the leading MDS method. In the simplest case, Stress is a residual sum of squares, where

$$\text{Stress} = \left(\sum_{i<j} \left(d_{i,j} - \delta_{i,j} \right)^2 \right)^{1/2}$$

and is minimized by straightforward gradient descent. Note that, this is very similar to the original fitness function used in the earlier version of NLP.

B. Survey of Related Methods

Reviewed here are a number of techniques for performing MDS. Readers interested only in evolutionary computation should skip to Section II. This section is included so that readers that wish to perform MDS will have a variety of options.

1) Principle Component Analysis (PCA): PCA, mathematically defined as an orthogonal linear transform, transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components [18]. Each principal component accounts for as much of the variability in the data as possible, with subsequent components accounting for as much of the remaining variability as they can. For a data matrix X^T with zero empirical mean, the PCA transformation is

$$Y^T = X^T W$$
$$Y^T = V \Sigma^T$$

where Σ is an m by n matrix with nonnegative reals on the diagonal and $W \Sigma V^T$ is the singular value decomposition of X.

Given a set of points in Euclidean space, the first principal component is the eigenvector with the largest eigenvalue and it corresponds to a line that passes through the mean and minimizes sum of squared error with those points. After all correlation with the first principal component has been subtracted out from the points, the second principal component follows suit. Eigenvalues indicate the portion of the variance that is correlated with the eigenvectors. What PCA does is essentially rotate the set of points around their mean to align them with the first few principal components. This moves as much variance as possible into the first few dimensions. The values of the remaining dimensions tend to be highly correlated and can be dropped with a minimal loss of information.

2) Isomap: Isomap is an approach to solving dimensionality reduction problems that uses local metric information to learn the underlying global geometry of a data set [17]. By using geodesic distances in place of Euclidean ones, it makes the projection of a certain class of nonlinear manifolds possible. Geodesic distances, in the presence of a metric, are defined to be (locally) the shortest path between points in the space [22]. This class gathers all compact, smooth submanifolds of $\Re^n$ that can be isometrically mapped to a convex domain of $\Re^m$. Isomap uses the following procedure.

1) Select randomly l points.
2) *Connect Neighboring Points*: Connect each point either with the k closest other ones, or with those lying closer than a certain threshold.
3) *Compute the Matrix D of All Pairwise Geodesic Distances*: Run Dijkstra's algorithm [9] for each point and store the pairwise distances between all points in a symmetric matrix M.
4) Center M by computing the mean of the rows, the mean of the columns, and the mean of all of the entries. Then, subtract the mean of the rows from each row, subtract the mean of the columns from each column and the grand mean to all entries; this makes M equivalent to $P^T P$.
5) Calculate the eigenvalues and eigenvectors of the centered M and sort eigenvectors according to the descending order of their associated eigenvalues.
6) Choose p such that the residual variance associated with the $l - p$ last eigenvectors is sufficiently small; suppress those eigenvectors.

The p kept eigenvectors give the coordinates of the mapped points in the p-dimensional projection space.

3) Locally Linear Embedding (LLE): LLE uses geometric intuition as the basis for its algorithm. Provided there is sufficient data (such that the manifold is well-sampled), it is expected that each data point and its neighbors lie on, or close to, a locally linear part of a manifold. With the choice of k neighbors, each point in a data set is reconstructed from linear coefficients associated with those neighbors. Reconstruction error is measured by

$$E(W) = \sum_i \left| \vec{X}_i - \sum_j W_{i,j} \vec{X}_j \right|^2$$

which adds up the squared distances between all the data points and their reconstructions. $W_{i,j}$, are the weights which summarize the contribution of the jth data point to the ith reconstruction. $E(W)$ is minimized by constrained linear fits. LLE then constructs a neighborhood preserving mapping by mapping each $\vec{X}_i$ to a lower dimensional $\vec{Y}_i$ representing global internal coordinates on the manifold. This is done by minimizing the embedding cost function

$$\phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{i,j} \vec{Y}_j \right|^2.$$

This cost function is based on locally linear reconstruction errors, but the weights are fixed from the previous equation and the data points $\vec{Y}_i$ are optimized.

In summary, the LLE algorithm works are as follows.

1) Compute the neighbors of each data point, $\vec{X}_i$.

2) Compute the weights, $W_{i,j}$, that best reconstruct each $\vec{X}_i$ from its neighbors, minimizing the cost by constrained linear fits.
3) Compute the vectors $\vec{Y}_i$ best reconstructed by $W_{i,j}$, minimizing the cost by its bottom nonzero eigenvectors.

4) Laplacian Eigenmaps (LEM): LEM uses an approach that constructs a graph incorporating neighborhood information of the data set. By using the notion of the Laplacian of the graph, it then computes a low dimensional representation of the data set that optimally preserves local neighborhood information. The algorithm of LEM is as follows.

1) Construct an adjacency graph. Place an edge between two points if they are "close," based on some measure, such as Euclidean distance.
2) Choose weights for the edges, according to the distances between the points they join.
3) Compute eigenvalues and eigenvectors for the generalized eigenvector problem

$$Lf = \lambda Df$$

where D is the diagonal weight matrix, its entries are column sums of W, $D_{ii} = \sum_j W_{ij}$. $L = D - W$ is the Laplacian, a symmetric, positive semidefinite matrix. Let $f_0, \ldots, f_{k-1}$ be the solutions of the generalized eigenvector problem, ordered according to their eigenvalues

$$\begin{aligned} Lf_0 &= \lambda_0 Df_0 \\ Lf_1 &= \lambda_1 Df_1 \\ &\cdots \\ Lf_{k-1} &= \lambda_{k-1} Df_{k-1} \\ 0 = \lambda_0 &\leq \lambda 1 \leq \cdots \leq \lambda_{k-1}. \end{aligned}$$

The f_0 eigenvector corresponding to the 0 eigenvalue is left out and the next m eigenvectors are used for embedding in m dimensional Euclidean space.

5) Curvilinear Distance Analysis (CDA): CDA comes from curvilinear component analysis (CCA), intended to be a nonlinear alternative to PCA [16]. As well, CDA may be seen as a neural version of Sammon's NLM, created as a nonlinear version of metric MDS using gradient descent instead of eigenvalue decomposition. CDA uses a vector quantization to select a subset from the data. Then, CDA optimizes a criterion that explicitly measures the preservation of the pairwise distances

$$E_{\text{CDA}} = \sum_{\substack{1 \leq i \leq l \\ i < j}} \left(\delta_{i,j} - d_{i,j}\right)^2 F\left(d_{i,j}, \lambda\right)$$

where $\delta_{i,j}$ is the distance between points in the data space and $d_{i,j}$ is the distance between points in the projection space. $F(d_{i,j}, \lambda)$ is the weighted contribution of a pair of points in the projection space. Usually, F is given as the heaviside step function.

The remainder of this paper is structured as follows. Section II specifies the evolutionary algorithm used for NLP, including the two mutation operators that are compared. It also specifies the data used in experiments, lists the experiments performed, and describes the analysis techniques used. Section III investigates NLP as an optimization problem using idealized mathematical data sets, comparing the two mutation operators. Section IV gives applications of NLP to the visualization and understanding of data. Section V discusses the results and draws the conclusion and Section VI outlines possible next steps for the research.

6) Evolutionary Techniques and Applications: There have been a number of publications that perform MDS with evolutionary algorithms. Groenen *et al*. [15] compared the genetic algorithm to other approaches for discovering global minima with stress in data sets that arise in the analysis of data drawn from mobile communication. Amenta and Klingner [1] used an evolutionary form of MDS to perform a visualization of evolutionary trees. Publications using evolutionary computation to perform MDS are typically focused on presenting a particular set of data effectively, rather than parameter setting, and considerations of representation. Pohlheim [21] used MDS for the analysis of the results of evolutionary computation and cites several other papers that use MDS in this fashion.

II. Evolutionary Algorithm

The evolutionary algorithm used to perform NLP employs a population of tentative projections stored as lists of points (x, y). The points are initially generated to lie within the unit square with corners $(0, 0)$ and $(1, 1)$ but may move out of it under the influence of mutation. The selection technique is single tournament selection with tournament size 7. The variation operators used are two point crossover of the lists of points (points are treated as atomic objects that cannot be split by crossover) and two different mutation operators that are experimentally compared to one another.

The first mutation operator selects from one to M loci in the coordinate list, with the number of loci selected uniformly at random. The operator then adds to each selected loci either a uniformly distributed random variable in the range $[-0.1, 0.1]$ or a Gaussian random variable with a standard deviation of 0.1. The two distributions are equally likely to be selected. The uniform mutation operator is intended to permit exploitation while the Gaussian permits exploration.

The second mutation operator generates two vectors with a number of coordinates equal to the number of points being projected. Both vectors are created by filling in coordinate values in the range $[-1, 1]$. A fraction W of the coordinates are forced to be zero in each vector. After choosing the number of coordinates to be zero, the vectors are then normalized to unit vector size and then scaled by a factor of 0.1. The first vector is added to the x coordinates of the points in the projection and the second vector is added to the y coordinates of the points in the projection. This mutation operator makes small changes in a large number of points, with W controlling how many points are moved. In essence, W is an inverse mutation rate control with $W = 1$ indicating no mutation and $W = 0$ moving each coordinate of every point.

This second mutation operator is designed to function efficiently as a stochastic hillclimber and simulates uniform mutation. Unlike uniform mutation, the operator does not use one random number per loci to determine if that loci will be

modified, rather it selects the number of loci not to modify and then picks them. This reduces the average number of random numbers required to perform the mutation.

A. Experimental Data

Several data sets are used to test evolutionary NLP. They are as follows.

1) A set of 100 points sampled uniformly at random from a circle of radius 1 in a plane through the origin generated by the vectors (1,0,1,0,1,0) and (0,1,0,1,0,1) in $\mathbb{R}^6$. This data set is included to illustrate the irrelevance of the input dimension. The algorithm rapidly detects that the data set is in a 2-D subset of six space, as witnessed by the fitness of $\rho = 1.0$.
2) The vertices of a 3-, 4-, and 5-cube, realized as the set of points in $\mathbb{R}^3$ or $\mathbb{R}^4$ in which all possible points with coordinates equal to 0 or 1 are chosen. These data sets were chosen because they vary equally in all their dimensions and have nontrivial structure in every subset of their set of dimensions.
3) A version of the 5-cube compressed along the coordinate axes so that the first dimension has length 1, the second length 0.95, the third length 0.90, the fourth length 0.85, and the fifth length 0.80. This data set is called the compressed 5-cube and is intended to give a small, difficult data set lacking the intrinsic symmetry of the other cubes.
4) A set of 100 points sampled from a torus with major radius six and minor radius one rotated isometrically into $\mathbb{R}^6$. The torus is chosen because it has 3-D structure that cannot be displayed in a 2-D projection.
5) A set of 300 points, 100 each from three tori with major radius six and minor radius one rotated isometrically into $\mathbb{R}^6$ and displaced in different directions. This data set is a simple generalization of the torus data set, chosen to demonstrate the ability of NLP to preserve discrete collections of points.
6) A set of 300 points, 100 each from three tori with major radius six and minor radius one rotated isometrically into $\mathbb{R}^9$ so that each set of points is in an orthogonal subspace. This data set is a second generalization of the torus data set that shows how NLP uses distance in 2-D to display orthogonality.

B. Experiments Performed

The first experiment performed is a parameter setting study comparing the two mutation operators on the data set consisting of three tori in six dimensions. This data set was selected as being most typical, among the idealized mathematical data sets, of those that might occur in a real application. It does not have collections of points neatly orthogonalized into different dimensions and has several hundred points, a feature that makes the hypercubes unsuitable for testing. In this experiment, three collections of 100 evolutionary runs were performed with the first mutation operator setting the maximum number of mutations M to 1, 3, and 5. Three collections of runs with the second mutation operator were performed with the zero coordinate control parameter W set to 0.7, 0.8, and 0.9. The population size for this experiment was ten, the best performing number located in the population size studies.

Using the best of the six mutation operators and rates tested in the first experiment, the second experiment compared population sizes of 10, 32, 100, 320, and 1000 on the 6-D three-torus data set. Comparisons of small and large populations have been performed, informally, before for NLP algorithms and found smaller population sizes work best. This experiment increases the number of population sizes tested and sets the result forth more formally. In the first two experiments, the algorithm is run for 20 000 mating events, collecting population fitness statistics every 200 mating events.

It is worth mentioning that the population size that works best for a given evolutionary algorithm varies substantially with the two largest categories being "bigger is better" and "small below a threshold works well." This issue is explored in some detail in [6] for genetic programming problems. The result for genetic programming has led to the testing, in our group, of populations sizes any time we are characterizing the evolutionary behavior of a problem. Because a population size of 10 worked best, we also performed a comparison of very small population sizes: 4, 6, 8, and 10. These experiments used size four tournament selection rather than seven because tournament size may not exceed population size. Comparing sizes smaller than four would require fundamentally changing the algorithm which would invalidate comparison.

A third experiment was performed using the best parameter settings found in the parameter setting experiments. This set of experiments was run with the compressed 5-cube, performing 100 independent replicates of the evolutionary algorithm. In addition to running the evolutionary algorithm a standard MDS package called SMACOF [10] implemented in the R-statistical software was applied to the data.

In both the first and second experiment, we use total maximum fitness as the reporting statistic. For a run of the evolutionary algorithm the total maximum fitness is the area under a plot of the maximum fitness in each generation, after the curve has been normalized so that the maximum possible fitness is one. Evolutionary time is also scaled to run from an initial time of zero and a final time of one. This forces the maximum value of the statistic—representing discovery of a global optima in the initial population—to 1.0 and the minimum value to zero. Since our fitness function, Pearson correlation, has a maximum of one the fitness normalization is trivial in this case. Total maximum fitness is higher when the optima located is better but also is higher the sooner fitness reaches a high value. This statistic is good for parameter setting because it simultaneously addresses issues of speed and quality. If the algorithm is run for a very long time, this statistic converges to simple maximum fitness. The use of time of last innovation, defined subsequently, can be used to check and see if the algorithm is being run for too many generations.

For experiments, the total maximum fitness computed for each of 100 individual runs will be plotted to permit comparison. This reporting statistic combines measurement of final quality with the speed with which that final quality was reached—algorithms gain area under their maximum fitness

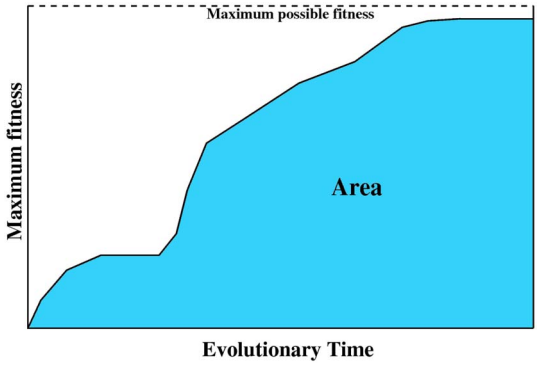

Fig. 1. Abstract plot of maximum fitness in the population over evolutionary time displays the area used to compute the maximum total fitness statistic.

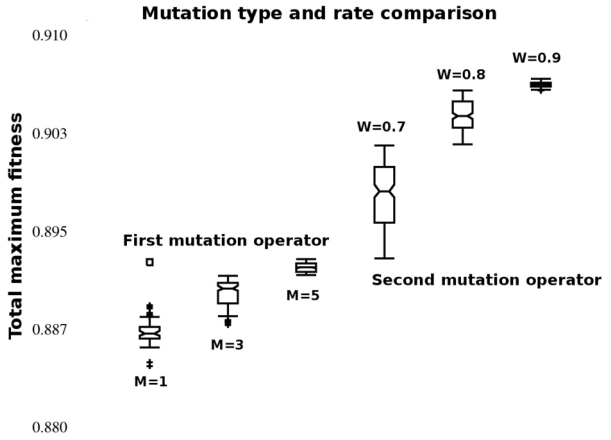

Fig. 2. Mutation parameter study showing inflected box plots of total maximum fitness for all six mutation types and rates tested.

curve both by achieving a high final value and by getting to high fitness values earlier. Fig. 1 shows the area that is normalized to compute maximum total fitness.

The third experiment applied NLP, as tuned by the parameter study, to the hypercubes, and three toroidal data sets to compare their difficulty. This experiment is intended to test the hypothesis that the hypercubes, which have equal amounts of variation in all dimensions, would be more difficult problems. Here, the algorithm was run for 400 000 mating events, substantially past the point where fitness approached its final value, with fitness statistics gathered every 4000 mating events.

The remaining experiments are applications to different interesting data sets. These are a simple feature set for small polyominos [12], the code words of the (7,3) Hamming code [20], an agent case embeddings (ACEs) [4] for the AI test problem Tartarus, and data produced from synthetic aperiodic DNA with a spectrum string kernel [19], applied to DNA rather than protein residues. The data sets and problems from which they arise are described in detail in Section IV to place the problem descriptions and analysis close to one another.

III. Characteristics of NLP as Evolutionary Optimization Problem

Comparing results from different runs of the NLP algorithm can be difficult because the fitness function is indifferent to position and rotation of the projection. To reduce this difficulty, the following normalization is applied to the projections to give them a standard form.

1) All projected points are translated so that the projection of the first point is at (0,0).
2) The set of projected points rotates about the origin so that the projection of the second point is on the x-axis of the plane.
3) If the projection of the third point has negative y-coordinate the projected points are reflected about the x-axis to give it a positive y coordinate.

A. Parameter Setting Results

The results of the first and second experiments are shown in Figs. 2 and 3 with the former reporting the results for mutation type and rate and the latter reporting the impact of population size.

Fig. 2 provides evidence that the second mutation operator is superior to the first one over all parameter settings. For the first mutation operator, the highest rate of mutation tested yields the best results. Analyzing the second mutation operator demonstrates the lowest rate yields the best results. Remember that W is the fraction of positions that are not changed at all by the mutation operator and so $W = 0.9$ is the setting that yields the smallest number of mutations.

The experiment on population size, shown in Fig. 3 yielded similarly conclusive results, matching informal results reported in past studies. Smaller population sizes achieve superior fitness, with significant degradation in performance as the population size is increased. The small population size studies with $n = 4, 6, 8,$ and 10 show that the effect continues as we continue to decrease the population size. The same number of tournament selections were performed in each evolutionary run so the number of fitness evaluations is held constant in the comparison. This suggests that burning away diversity is helpful in increasing fitness; this also implies mutation can perform most of the search. This in turn suggests that the new (second) mutation operator will increase NLPs ability to search for high fitness projections—it is introducing new information at a higher rate than the original (first) mutation operator.

Another fact apparent in Fig. 3 is that the variability of outcomes increases sharply with population size. Not only did smaller populations produce significantly better results than large ones, they did so with greater consistency. The plots for population size overlap only at their very ends, suggesting performance for a smaller population size is uniformly better, with the exception of the comparison of population sizes 10 and 32; even these two sets of runs exhibit significantly different performance. For the small populations significant improvement was obtained between 8 and 6 and 6 and 4. This strongly suggests that a simple stochastic hillclimber of an evolution strategy [8] would outperform the evolutionary algorithm at finding high fitness solutions.

B. Does NLP Act Like Unimodal Evolutionary Optimization Problem?

Standard MDS is known to have the potential for multiple local optima and tools like SMACOF have an option for

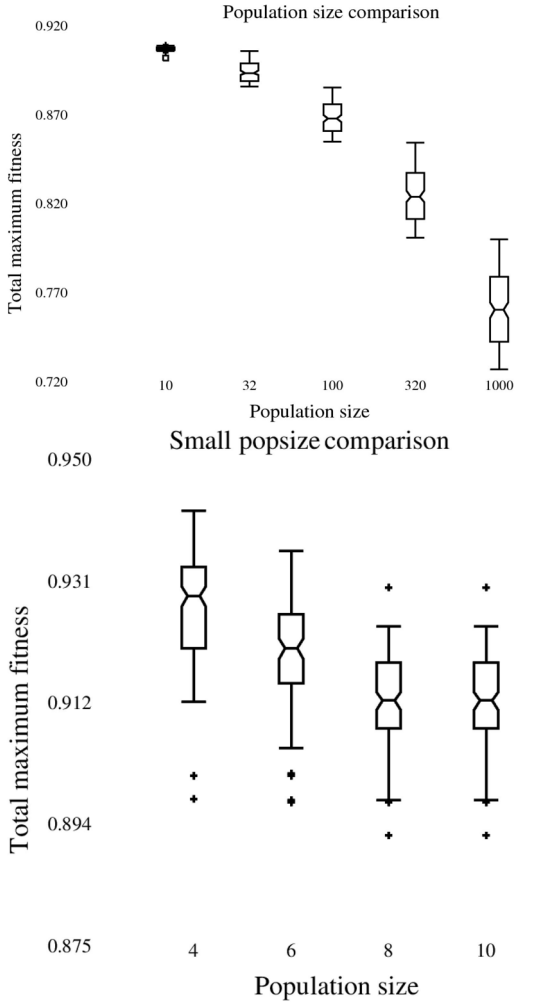

Fig. 3. Population size parameter study showing inflected box plots of total maximum fitness for the widely spread population sizes (top) and the small population sizes (bottom).

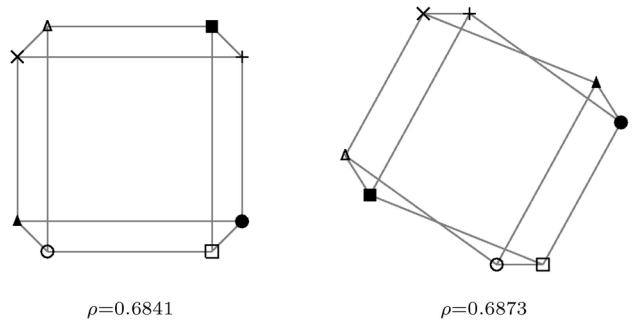

Fig. 4. Two forms of the vertices of the cube when projected from $\mathbb{R}^3$ to $\mathbb{R}^2$. The value ρ is the Pearson correlation of the 2-D and 3-D distance matrices, used as the fitness measure by the evolutionary algorithm.

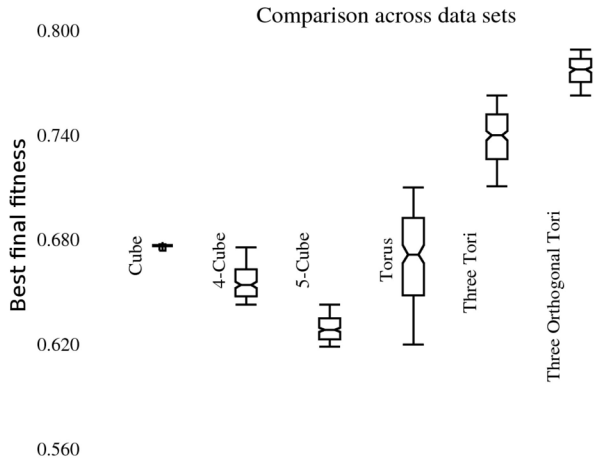

Fig. 5. Shown are distributions, in the form of inflected box pots, of the best fitness of final populations over 100 replicates of the tuned algorithm on the six mathematically structured data sets using a population of ten.

multiple restarts to permit sampling of these optima. When used in applications, the evolutionary version of the algorithm finds the same best result quite often across multiple runs. In many cases this best optimum was located in a majority of the runs. This suggests that the evolutionary algorithm was efficiently locating a global optima. This led to the hypothesis that the algorithm was, in most cases, performing a simple unimodal optimization after locating the basin of attraction of a global optima. Generally, if there are many basins of attraction, then a large population is required to locate good ones. Small populations excel when a single basin of attraction is present or dominant. The population size study thus supports the notion that the algorithm, on the triple tori data set, is behaving in a manner with spending most of its effort climbing a single hill. One goal of this paper was to examine this situation in a simple, low-dimensional case.

Using the vertices of a cube in $\mathcal{R}^3$ as a test data set, we are able to demonstrate that the algorithm, even in this very simple case, finds an inferior optima 10% of the time. In 100 replicates, the cube exhibited two distinct optima, examples of which are shown in Fig. 4. These optima are distinguished both by their fitness values and their renderings of the cube that result.

The distribution of best fitness in the final generation of evolution for the mathematical data sets, other than the circle, are shown in Fig. 5. In the experiments using points drawn from the circle a correlation coefficient of 1.0 was obtained in 99 of 100 replicates with a correlation of 0.9997 obtained in the other replicate; a rendering of one of the projections appears in Fig. 6. Fig. 5 demonstrates that the 3-, 4-, and 5-cubes are the most difficult data sets, with difficulty rising with dimension. The three toroidal sets also have an intuitive difficulty order. The set with only one torus is easy, with the algorithm managing to efficiently fit distance orthogonal to the circle at the major diameter into a 2-D annulus. The data set with three tori in random orientations in $\mathbb{R}^6$ had relatively large distances between the tori, making the problem relatively easy. The three orthogonal tori in $\mathbb{R}^9$ were closer but the algorithm managed to map the use of orthogonal dimensions into distance in the plane fairly well with a best fitness of $\rho = 0.947$.

Examples of renderings of projections of the circle, the 4- and 5-cubes and data sets based on tori are shown in Fig. 6. Notice that the cubes manage their projection by almost ignoring or crushing the 3rd and 4th, or 3rd, 4th, and 5th dimensions. The tori are somewhat distorted, but their essential characters are preserved in the renderings.

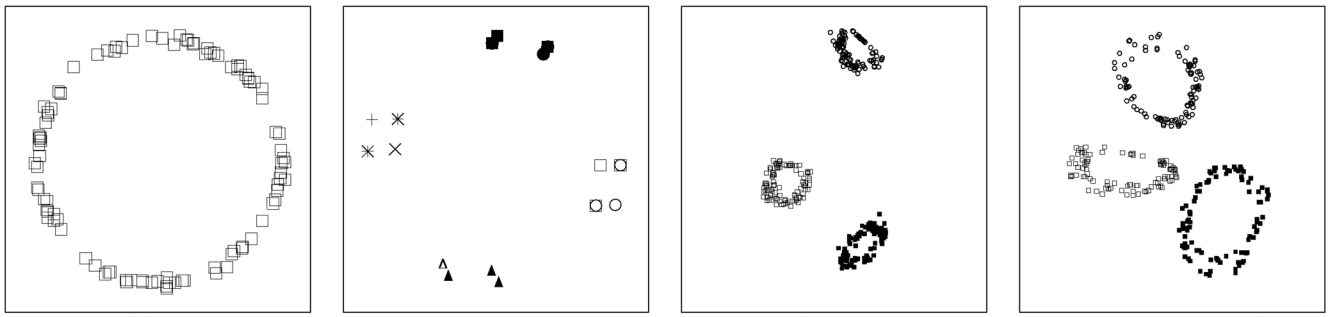

Fig. 6. Shown are renderings of the projections of some of the mathematically structured data sets. The starting dimension of the data set is given after the name of each data set. Glyph choice is made to separate categories of points when there are natural categories; glyphs are tied to square faces of the hypercubes. The number ρ is the fitness value for the projection shown.

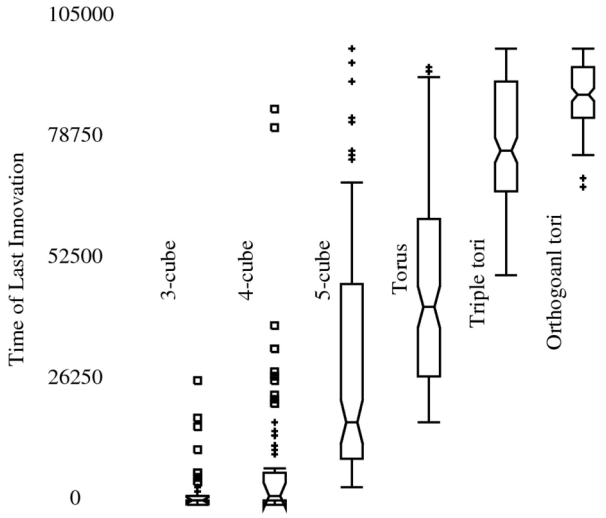

Fig. 7. Inflected box plots of the hypercube and torus data sets showing the index of the sample in which fitness arrived within 0.001 of its final value. This quantity is called the time of last innovation, given in generations on the vertical axis.

C. Time of Last Innovation

One method of deciding if an algorithm has been run long enough is to see when it is arriving at its final fitness value. Since the correlation coefficient depends continuously on the position of each point, tiny increases in fitness can continue indefinitely. For this reason, we track the index of the first fitness sampling in which a replicate came within 0.001 of its final value. Fig. 7 shows this time of last innovation for the data sets derived from hypercubes and tori.

If we juxtapose the fitness data in Fig. 6 against the time of last innovation data in Fig. 7 then the data sets fall into two distinct groups, the hypercubes, and the tori. The time of last innovation increases as the difficulty of each NLP problem increases for both groups of data. There is a noticeable difference between the mean time of last innovation between the groups of data sets, as the hypercube data sets tend to have a much lower mean time of last innovation compared to the tori data sets. However, the variability of time of last innovation increases with the dimension of the hypercube, and thus the difficulty of the problem, increases. For the tori data sets, as the difficulty increases the variability of the time of last innovation decreases. The following facts will help in analysis of the results: the hypercube data sets have the same amount of variation in each dimension, and the variability of the tori are concentrated in particular directions spanned by the major radius of the tori. A plausible explanation for the behavior demonstrated in Fig. 7 is as follows.

Problems that cannot yield accurate projections, such as the hypercubes, have a large number of inaccurate projections, depending on which dimensions are ignored or crushed. Any innovations that occur at later evolutionary times represent the algorithm choosing between the possible inaccurate projections, rather than refining a given projection. Most of the projections of the hypercubes are going to have poor fitness, so once a decent projection is found evolution will spend a majority of the time discovering and then discarding lower quality projections, rather than finding projections that cause an increase in fitness. Problems that have accurate projections spend evolution refining those projections. A later time of innovation, like with the tori problem, represents improvements in the accuracy of the projection, rather than choice of type of inaccuracy. That is why the mean times of last innovation for the tori problems are substantially higher than the hypercube problems on average. Rather than choosing between several bad options, the algorithm can search for a few high quality ones, and once one is found, can refine that projection for the rest of the time given by the algorithm. Summed up, the better the best possible projection is, all other factors being equal, the longer innovation will continue. This explanation requires additional testing.

D. Comparison With SMACOF

Fig. 8 shows the projections found by two of the evolutionary runs and one found by the SMACOF package. The SMACOF result displays one square face of the cube with complete fidelity and no perspective on the other faces. The evolutionary computation-generated visualizations contain perspective on more than one square face. By looking at the glyphs, which were assigned in groups of four to the 8 square faces of the 5-hypercube, it is possible to

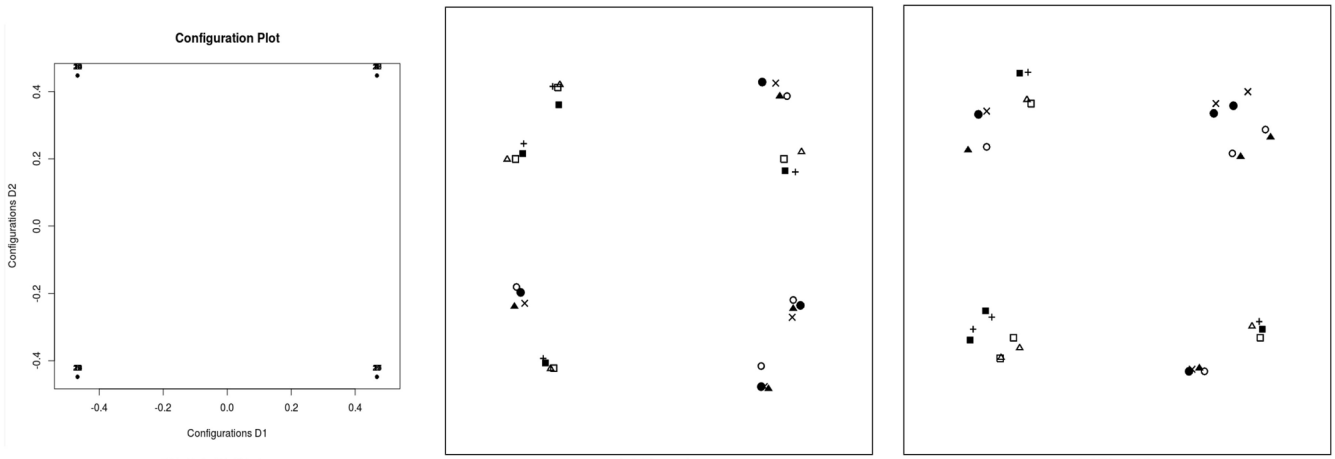

Fig. 8. Shown are NLPs of the compressed 5-hypercube data set found with the SMACOF package (left) and evolutionary algorithm (middle and right). The fitness of each projection is given below the picture. It is important to note that the SMACOF package was not optimizing for the fitness function used—this value demonstrates not an inferiority of SMACOF, rather it shows SMACOF locates different types of optima.

see that the two projections shown are centered on different faces, in different sets of dimensions, of the 5-cube. These projections are showing distinct views that contain distinct subsets of the information contained in the data. The large difference in fitness follows from the use, by SMACOF, of a different objective function. The evolutionary algorithm located, in 100 trials, 6 distinct optima representing centerings on six of the eight possible square faces of the hypercube.

IV. Applications

In this section, we demonstrate some of the ways that NLP can be applied.

A. Visual Classification of Polyominos

A polyomino is a set of squares, connected by sharing a face, that form a connected set. The following set of features were extracted from the set P_5 of 18 polyominos with five or fewer squares.

1) Number of squares.
2) Longest dimensions.
3) Shortest dimension.
4) Number of corners.
5) Number of concave corners.
6) Perimeter.
7) Longest line of squares.
8) Number of dihedral symmetries admitted by the polyomino.
9) Polyomino has a 180 degree rotational symmetry (0 = no, 1 = yes).

This feature set was chosen because no two of the polyominos in P_5 have equal feature vectors. The vector's coordinates were then normalized to lie in the range [0,1] and treated as points in $\mathbb{R}^9$. The resulting points were then subjected to NLP. The result is shown in Fig. 9 with the polyominos themselves used as glyphs for display. Notice that position agrees well with the sorting principles implied by several of the features.

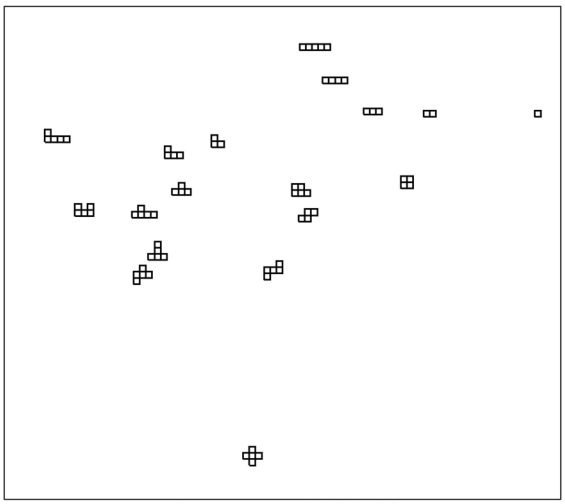

Fig. 9. Projection of the feature vectors of the members of the set of polyominos with five or fewer squares.

B. Projection of the (7,3)-Hamming Code

The (7,3)-Hamming code [20] is a selection of 16 binary vectors of length seven that have the following properties: 1) the smallest Hamming distance between any two members is 3; 2) every binary vector of length seven is at Hamming distance one from a member of the code; and 3) the code has a transitive automorphism group—performing a bitwise XOR of any word with all other words takes each member of the code to another member. This third condition means that the symmetry group of the code can take any member to any other member, like the vertices of a cube.

Treating the vectors in the code as points in $\mathbb{R}^7$ with coordinates equal to 0 or 1, a NLP of the code was performed. All 100 replicates returned a version of the picture in Fig. 10 with fitness ranging from 0.498129 to 0.498323. The variation is almost certainly due to minor jittering of the points.

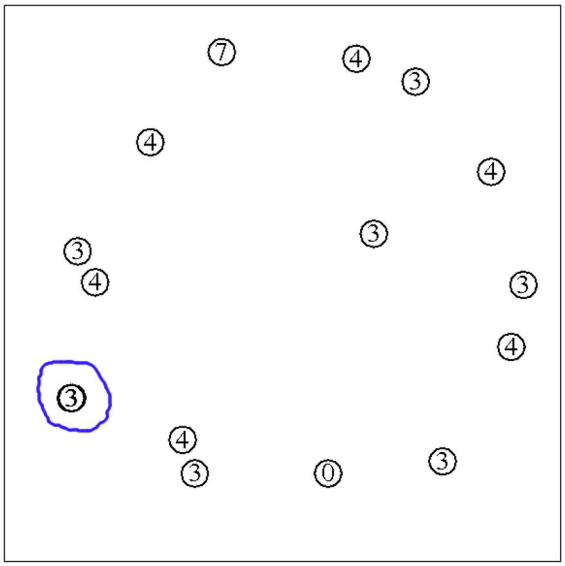

Fig. 10. NLP of the (7,3)-Hamming code. Points are marked with the weight of the vector. The apparently bold "3" in the lower left, which is circled, is a superposition of two vectors of weight 3.

The weight of a vector is the number of ones in the vector. The (7,3)-Hamming code contains one word each of weights 0 and 7 and seven words each of weight 3 and 4. The drawing exhibits a number of symmetries that follow the 7-D structure of the code. The words of weight 7 and 0 are at maximum distance; the ring shape made by 14 of the points corresponds to a chain of close pairs of code words that go around the "waist" of the 7-cube, from which the words are drawn, and there is an obvious reflective symmetry.

The fitness values for this projection, less than $1/2$, is quite bad. As we have noted, the (7,3)-Hamming code is a subset of H_7, the 7-D hypercube. Additionally, the subset spans all seven dimensions. Given our experiments with H_5 the low fitness values are what would be expected.

C. Tartarus

Tartarus [2] is a simple AI test problem. An agent called the bulldozer attempts to push six boxes on a 6×6 grid so that as many box sides are against the wall as possible. The bulldozer is given 80 moves in which it may turn left or right or attempt to advance. It can only advance into an empty space or push one box ahead of it, other attempts to advance leave it in place. A valid starting board for the Tartarus problem must have no 2×2 blocks of boxes (this is an impossible configuration) and must start with no box sides against the wall. There are over 300 000 possible valid starting configurations, one of which is shown in Fig. 11.

Using the experimental design [3], we run four experiments each of which is comprised of 30 independent runs of an evolutionary algorithm. Two of the experiments use random initial populations of 60 agents with a linear genetic programming representation called an If Skip Action (ISAc) list [2]. In the first experiment ISAc lists with 30 instructions are used, in the second ISAc lists with 60 are used. The second pair

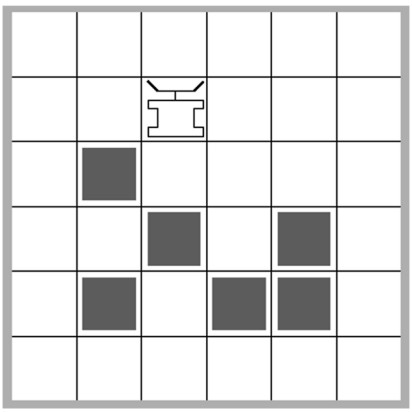

Fig. 11. Example starting configuration for Tartarus.

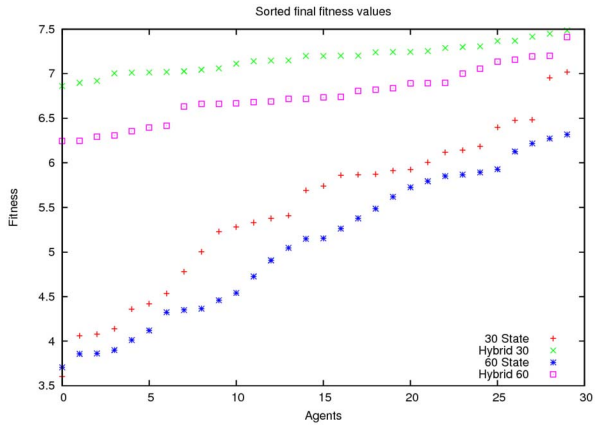

Fig. 12. Fitness values of 30 final agents for all four Tartarus experiments shown in sorted order from left to right.

of experiments are identical to the first except that the best results of the previous experiments (using the name number of instructions) are placed into the initial populations. This technique is called hybridization and is explored in [3]. This gives us four sets of 30 agents, each the best agent produced in a run of an evolutionary algorithm. The algorithm is run for 200 generations using 400 randomly selected initial Tartarus boards, different in each generation, to evaluate fitness. The fitness of the agents for each experiment is compared in Fig. 12.

In order to turn agents represented in the form of a linear genetic program into points that can be visualized via NLP, we use ACEs [4]. A set of 100 starting boards were selected uniformly at random and each agent is mapped to a point in $\mathbb{R}^{100}$ by scoring it on each of the boards. The resulting set of 120 (four groups of 30) agents is thus transformed into 120 points in 100-D space. The resulting NLP is shown in Fig. 13.

Examining Fig. 13 in comparison with Fig. 12, we see fitness increases toward the bottom of the projection. We also see that agents with 30 instructions reach higher fitness values than those with 60 and all hybrid agents are more fit than the nonhybrid agents. The nonhybrid agents form large, interpenetrating clouds, but the hybrid agents form two fairly tight clouds. The tightness of the clouds is easy to explain—the hybrid agents are less diverse because they are descended from a small pool of agents. A curious feature of the projection is

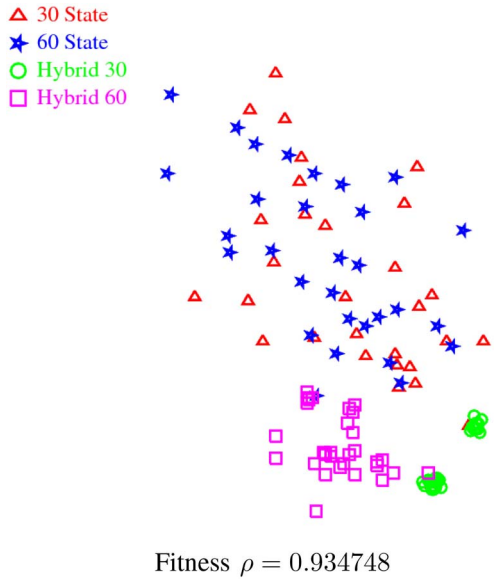

Fig. 13. NLP of an ACE using 100 randomly selected Tartarus boards for four categories of Tartarus agents.

TABLE I
SHOWN ARE 95% CONFIDENCE INTERVALS ON THE EXCESS HANDEDNESS OF EACH OF THE FOUR TARTARUS EXPERIMENTS FROM GENERATIONS 100 TO 200

Length 30	4.875+/-0.394
Length 60	3.520+/-0.113
Hybrid 30	2.436+/-0.005
Hybrid 60	1.951+/-0.012

the two clouds formed by the 30 instruction hybrid agents. These clouds turn out to have a simple explanation—they are right and left handed robots.

To quantify this, the excess handedness of a robot is defined to be the ratio of the more common of the left or right turns over the rarer of the two types of turns. A ratio of one indicates an agent with no preference for left or right turns. A preference for one type of turn typically appears early in evolution; it is an immediate, simple source of higher fitness. A fixed tendency to prefer one type of turn is associated with some types of local optima [2].

Table I shows the excess handedness from the second half of each evolutionary run for all four experiments. The non-hybrid runs exhibit typical early-evolution dependence on a preference for one type of turn, but variations in fitness prevent the formation of a clear left or right handed group in the projection. The hybrid experiments, which have agents that have been evolving longer, both have significantly less preference for one sort of turn over the other. The hybrid agents with 30 instructions are significantly more "handed" than the 60 instruction agents. Inspection of turn statistics saved during evolution show that the 30 instruction agents not only have a strong preference for one sort of turn over another but there are 19 "left-handed" agents and 11 "right-handed" agents.

D. DNA Data

DNA is the primary long-term information storage molecule in living organisms [11]. Separating different classes of DNA, e.g., protein coding, regulatory, or junk, is a standard classification task [7]. In this paper, synthetic DNA data was produced using self-driving finite state transducers. An example of one of these appears in Fig. 14. The machine initially outputs the symbols on the sourceless arrow pointing to the first state. The output is buffered and used to drive the machine. The fact that some of the labels emit two characters ensures that the output grows faster than the input (at least if those transitions are used). This in turn typically forces the output to be an aperiodic string.

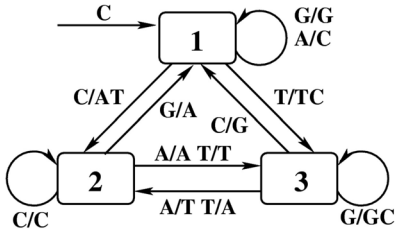

Fig. 14. Example of a self-driving finite state transducer.

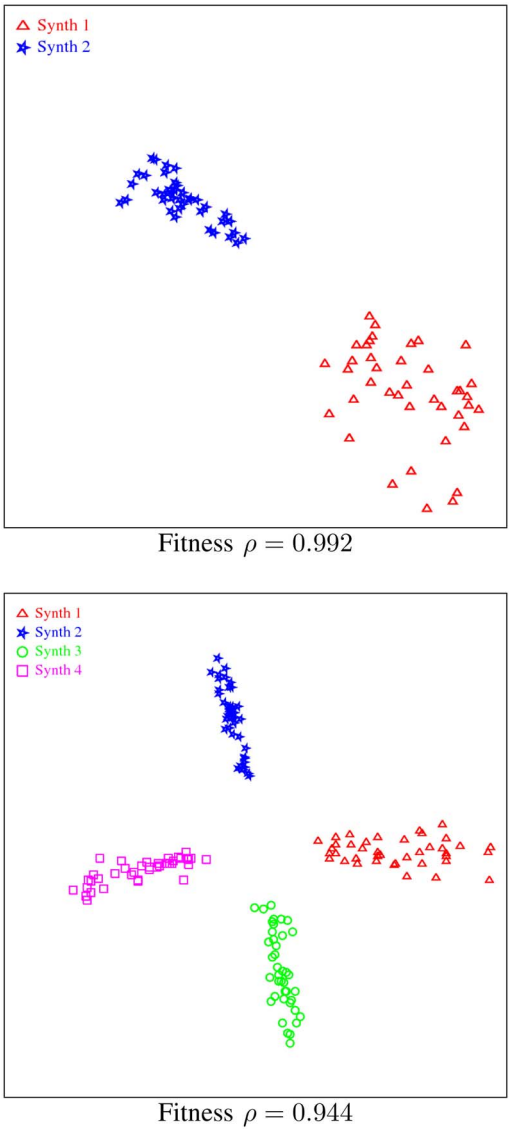

Fig. 15. NLPs of the spectrum-3 kernel for two and four synthetic data sets.

The four sets of data used were generated by four 12-state automata of this type selected as being in the top 1% of entropy values among a set of 100 000 randomly generated automata. The entropy mentioned is the Shannon entropy of the DNA 6-mers that are substrings of the machine's output measured over the first 1 000 000 characters of output. This ensures the DNA, while containing a deterministic pattern, is not low-complexity. The output string for an automata is sampled for nonoverlapping DNA strings of length 150–250 with length chosen uniformly at random. A set of 40 strings are selected from each automata's output to create a data set.

The points to be projected were extracted from the DNA strings by counting the number of times each of the 64 possible 3-character substrings occurred. This is the spectrum-3 kernel for DNA [19]. The first two and all four synthetic DNA data sets were projected yielding the pictures shown in Fig. 15.

The data in Fig. 15 are separated clearly into correct groups, so the spectrum string kernel had no problem separating the different types of DNA and the NLP algorithm was able to display those 64-D differences accurately in two dimensions. There is another feature of the data that is visible in both displays, though it is more visible in the four-class data. The projected points for a particular type of DNA form a line or banner-like shape. This feature is caused by the length of the DNA strings. The shorter strings induce points, via the spectrum strong kernel, that are closer to the origin of 64-D space while the longer strings induce points that are farther away.

V. Conclusion

This paper introduces and evaluates a form of evolutionary NLP. There are many ways to perform this kind of projection, each of which provides a different, inaccurate picture of the initial distance or dissimilarity data. As optimizers, it is tempting to say that once you have chosen a fitness function, a stress or strain measure in traditional MDS, then you simply want the best optima. As the experiments with the compressed 5-cube in Section III-D show, this is incomplete, and possibly incorrect. A strain measure may have many global and local optima. Unless the data are directly representable in the target dimension of the projection, every one of these optima destroys information about the data in a different way. When using PCA, it is not uncommon to plot the first versus the second, the second versus the third, the third versus the fourth, and so on, until the components become unimportant fractions of the variation. In a similar fashion, distinct optima of a strain function contain different information and may be worth looking at. A researcher with a set of data that is difficult to understand or interpret should use several methods of projection and attempt to gain insight by juxtaposing them all.

This paper does present quantitative results in the form of testing of a new mutation operator designed particularly for high dimensional problems. The new mutation operator is useful for modifying many genetic loci a small amount, with the overall total change bounded by a size parameter. We demonstrate that the mutation operator yields higher fitness relative to the mutation operators previously used for NLP. This paper also summarizes experience gained while using the older version of the NLP as a research tool.

A. Types of Problems and Projections

This paper presented here also grants us new insight into the nature of NLP as an evolutionary optimization problem. The hypercube results are projections that choose 2-D from those available, yielding very poor fitness scores, and exhibiting a relatively large number of optima, roughly half the size of the symmetry group of the hypercube, $n! \cdot 2^{n-1}$ [23]. In contrast the results for the data sets built from Tori have optima with high fitness values and the algorithm locates multiple versions of a single optima. The juxtaposition of these results suggest that poor fitness values are at least partially diagnostic of data sets with multiple optima of similar quality. The evidence for this is that the same optima, up to rotation and translation, show up over and over for the data sets with good fitness values. These data sets are those where either the variation being projected is simply in two dimensions (suggesting principle components would be a superior technique) or data sets that admit something like a "Mercator projection" in which different parts of space are flattened and then placed in the plane in a fashion that substantially distorts the actual arrangement of points while preserving most of the information about their distance relationships. The data sets with three tori exhibit this "Mercator like" effect.

An example of this type of distortion occurs in the circle data set as well. While six dimensional, it has all its variation in one plane and is an example of a data set where PCA would clearly be better than NLP. Notice that the projection of the circle in Fig. 6 does not place the points on a circle—since the points are drawn randomly from a circle, small displacements off the circle yield a better match to the distance data than correctly placing the points on an abstract circle.

The six dimensional data set with three tori is an example of a problem where the projection preserves distance relationships very well, Pearson correlation ($\rho = 0.994$), but with no hope that the actual variation is taking place in two dimensions. Each of the three tori has most of its variation in two dimensions, but these pairs of dimensions are different for each of the tori. What the algorithm has done is map the local plane of variation for each tori into the plane and then both spaced out and individually distorted them to get the best overall fit. This is the type of projection used in map making with the tori representing land and empty space representing the sea. Distance relationships concerning land are more important and are what the projection seeks to preserve.

At this point, we return to the hypercube data sets and the (7,3)-Hamming code, which is a subset of a hypercube. A hypercube has an equal amount of distance variation in all dimensions and so a map-projection like solution with high fitness does not exist. Rather, the algorithm must choose which distance data to mostly ignore. The projections of hypercubes in Fig. 6 show the hypercubes as squares with groups of points crushed into each corner of the square. This

also explains the time-of-last-innovation results in Fig. 7. As the dimension of the hypercube increases, the algorithm must make more decisions about what information to ignore. This lengthens convergence time and hence time of last innovation.

B. Runtime

An important question about a new algorithm, or an old one that is being characterized, is its running time. The new mutation operator reduced the number of mating events required to obtain useful projections (this is a qualitative judgment) from 400 000 to 20 000 mating events on the three torus data set. With the new mutation operator, the time to obtain one projection on a commodity box is a few seconds for one hundred points and a few minutes for thousands. Since quantifying the exact performance of the algorithm is difficult, we list the speed advantage of the new mutation operator as a subjective accomplishment. The run time of the NLP algorithm has been improved to values perhaps best characterized as "practical" and "not annoying." Since the fundamental step in the algorithm is computing the correlation coefficient of two matrices whose dimension is the number of points being projected, the time to perform one fitness evaluation is $O(n^2)$ for n points.

C. Partnering With PCA

Analysis of the juxtaposition of the hypercube and toroidal data sets suggests that PCA can be profitably partnered with NLP. First, if almost all of the variation is in the first two principle components, then performing NLP would be gratuitous. If the variation is in a relatively small number of principle components then PCA is likely to work as well as NLP, if not better. If the variation is spread over a large number of components, then a high fitness for NLP suggests that the variation in different dimensions is in collections of points that are well separated from one another, as in the data sets with three tori. If the variation is spread over a large number of components and the NLP only achieves low fitness, it is likely that the data is analogous to the data created with hypercubes. In these data sets, the variation appears in many directions and occurs in points that are relatively localized. Combining the concepts of PCA, namely how much each variation is attributed to each dimension, with NLP would be very informative as to what sort of fitness to expect from NLP, which can help to inform run-time and other parameter settings.

VI. Next Steps

The obvious next step is a paper in which more difficult data sets are examined with the NLP tool and other tools. The goal is not so much to compare the tools as to understand how, and on what type of data, they work well together. Another possible direction is to look at how time of last innovation and fitness can be juxtaposed to potentially classify a problem into two categories, one where map-like projections are possible and one where they are not. The boundaries of these categories are not crisp, but this might nevertheless be a fruitful line of investigation.

A. Better Initialization

Many of the available MDS techniques are faster than NLP. This suggests that they could profitably be used as initializers, either directly by adopting their output as sets of points or indirectly. PCA, for example, can be used to find which dimensions contain most of the variation. Generating pairs of unit vectors restricted to the subspaces of high variation and then taking projections based on those initial population members could yield higher quality initial populations.

B. Possible Technical Innovations

After the first publications in which NLP was used as a projection tool, the authors were approached by colleagues with $10^5 - 10^7$ point data sets. The algorithm definitely cannot work with this sort of data in its current form. The following techniques permit some progress.

1) Project a random sample of the data of tractable size.
2) Cluster the data and project the cluster centers.
3) Build minimal spanning distance trees of the data, with each forbidden to use edges used by the others, and then only compute and compare distances on the edges of those trees.

The first two techniques, as far as the authors can tell, are folklore. We find these techniques in common use without a definitive reference. The third is idea in introduced here. It would reduce the algorithm time for a fitness evaluation to $O(\text{kn})$ with the number k of spanning trees used, which would create a speed-quality tradeoff parameter. The spanning tree system is an early priority for future research.

We note that starting with some of the points and adding more as evolution proceeds is an idea that our group has tested. The results are much faster but also yield projections with awful quality. The initial points and order of added points, was selected uniformly at random, so the idea is not hopeless—other orders of point selection may demonstrate the validity of the idea.

Finally, we use Pearson correlation of the distance matrices as a fitness function because it is translation and rotation invariant to the positions of the projected points. For the type of data sets encountered by the authors, this is ideal. As we saw in the literature review, many other strain measures are available. Aside from speed considerations related to the difficulty of computing the strain measure, the evolutionary algorithm we use can transparently be converted to use other strain measures.

References

[1] N. Amenta and J. Klingner, "Case study: Visualizing sets of evolutionary trees," in *Proc. IEEE Symp. Inf. Vis. (INFOVIS)*, 2002, pp. 71–74.
[2] D. Ashlock, *Optimization and Modeling With Evolutionary Computation*. New York, NY, USA: Springer-Verlag, 2006.
[3] D. Ashlock and K. M. Bryden, "Breeding schedules improve grid robot performance," *Intell. Eng. Syst. Through Artif. Neural Netw.*, vol. 16, pp. 143–148, 2007.
[4] D. Ashlock and C. Lee, "Agent-case embeddings for the analysis of evolved systems," *IEEE Trans. Evol. Comput.*, vol. 17, no. 2, pp. 227–240, Apr. 2013.
[5] D. Ashlock and J. Schonfeld, "Nonlinear projection for the display of high dimensional distance data," *IEEE Congr. Evol. Comput.*, vol. 3, pp. 2776–2783, Sep. 2006.

[6] W. Ashlock, "Using very small population sizes in genetic programming," in *Proc. Congr. Evol. Comput.*, Vancouver, BC, Canada, 2006, pp. 319–326.
[7] W. Ashlock and S. Datta, "Evolved features for DNA sequence classification and their fitness landscapes," *IEEE Trans. Evol. Comput.*, vol. 2, no. 17, pp. 185–197, Apr. 2013.
[8] H. G. Beyer, *The Theory of Evolution Strategies*. New York, NY, USA: Springer, 2001.
[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. New York, NY, USA: MIT Press, 2001.
[10] J. de Leeuw and P. Mair, "Multidimensional scaling using majorization: SMACOF in R," *J. Stat. Softw.*, vol. 31, no. 3, pp. 1–30, 2009.
[11] E. S. Goldstein, J. E. Krebs, and S. T. Kilpatrick, *Genes XI*. Woods Hole, MA, USA: Jones and Bartlett Learning, 2013.
[12] S. W. Golomb, "Tiling with sets of polyominoes," *J. Comb. Theory*, vol. 9, no. 1, pp. 280–296, 1966.
[13] J. C. Gower, "Some distance properties of latent root and vector methods used in multivariate data analysis," *Biometrika*, vol. 53, nos. 3–4, pp. 315–328, 1966.
[14] P. J. F. Groenen and I. Borg, *Modern Multidimensional Scaling: Theory and Applications*. Secaucus, NJ, USA: Springer-Verlag, 2005.
[15] P. J. F. Groenen, R. Mathar, and J. Trejos, "Global optimization methods for multidimensional scaling applied to mobile communication," in *Data Analysis*, W. Gaul, O. Opitz, and M. Schader, Eds. Heidelberg, Germany: Springer, 2000, pp. 459–469.
[16] M. Verleysen, J. A. Lee, and A. Lendasse, "Nonlinear projection with curvilinear distances: Isomap versus curvilinear distance analysis," *Neurocomputing*, vol. 57, pp. 49–76, Mar. 2004.
[17] J. C. Langford, J. B. Tenenbaum, and V. de Silva, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000.
[18] S. K. Kachigan, *Multivariate Statistical Analysis: A Conceptual Introduction*. New York, NY, USA: Radius Press, 1991.
[19] C. Leslie, E. Eskin, and W. S. Noble, "The spectrum kernel: A string kernel for SVM protein classification," *Pac. Symp. Biocomput.*, vol. 7, pp. 566–575, 2002.
[20] R. McEliece, *Theory of Information and Coding*. Boston, MA, USA: Cambridge Univ. Press, 2002.
[21] H. Pohlheim, "Visualization of evolutionary algorithms—Set of standard techniques and multidimensional visualization," in *Proc. Genet. Evol. Comput. Conf.*, San Francisco, CA, USA, 1999, pp. 533–540.
[22] M. Schiffer R. Adler, and M. Bazin, *Introduction to General Relativity*, 2nd ed. New York, NY, USA: McGraw-Hill, 1975.
[23] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*. Cambridge, U.K., Cambridge Univ. Press, 2001.

Daniel Ashlock received the doctorate degree in mathematics from the California Institute of Technology, Pasadena, CA, USA.

He is a Professor of Mathematics, specializing in bioinformatics, with the Department of Mathematics and Statistics, University of Guelph, Guelph, ON, Canada. His current research interests include bioinformatics, computational intelligence, games, and the theory of evolutionary computation. He has authored over 200 refereed scientific publications.

Andrew McEachern received the undergraduate and doctoral degrees in mathematics from the University of Guelph, Guelph, ON, Canada.

His current research interests include evolutionary computation, mathematics applied to biological problems, and mathematics education at all levels.