2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

An Updated Taxonomy of Evolutionary Computation Problems using Graph-based Evolutionary Algorithms

Daniel A. Ashlock
Mathematics and Statistics
University of Guelph
Guelph, ON Canada N1G 2R4
dashlock@uoguelph.ca

Kenneth M. Bryden
Mechanical Engineering
Iowa State University
Ames, Iowa 50011
kmbryden@iastate.edu

Steven Corns
Mechanical Engineering
Iowa State University
Ames, Iowa 50011
scorns@iastate.edu

Justin Schonfeld
Bioinformatics
Iowa State University
Ames, Iowa
50010, USA
shonfju@iastate.edu

Abstract

Graph based evolutionary algorithms use combinatorial graphs to impose a topology or "geographic structure" on an evolving population. It has been demonstrated that, for a fixed problem, time to solution varies substantially with the choice of graph. This variation is not simple with very different graphs yielding faster solution times for different problems. Normalized time to solution for many graphs thus forms an objective character that can be used for classifying the *type* of a problem, separate from its hardness measured with average time to solution. This study uses fifteen combinatorial graphs to classify 40 evolutionary computation problems. The resulting classification is done using neighbor joining, and the results are also displayed using non-linear projection. The different methods of grouping evolutionary computation problems into similar types exhibit substantial agreement. Numerical optimization problems form a close grouping while some other groups of problems scatter across the taxonomy. This paper updates an earlier taxonomy of 23 problems and introduces new classification techniques.

I. Introduction

Graph based evolutionary algorithms (GBEAs) [8] use a combinatorial graph [18] to place a geographic structure on an evolving population. Members of the population can only be replaced by variations of the neighbors. Highly connected graphs yield behavior quite similar to a standard evolutionary algorithm, while sparse graphs restrict the rate at which information flows within the graph. Simple unimodal problems have the shortest time to solution (measured in number of fitness evaluations) on highly connected graphs, while harder problems exhibit better performance on sparse graphs. This general trend is, however, not uniform, and the details of the connectivity of the graph have an impact on performance [8].

If a problem is run on several different graphs, then the pattern of faster and slower times to solution on different graphs forms a characterization of the *type* of problem. The representation (encoding, choice of variation operators, details of fitness evaluation) all have an impact on the type of the problem, a notion that will be left intentionally vague, save that the details of each evolutionary computation problem are given in detail. The overall difficulty of a problem is defined, for this study, to be its average time to solution over all graphs. Our goal is to normalize this hardness away and examine the difficulty-free character of the problem. By normalizing the times-to-solution for different graphs to lie in the range [0,1], the overall difficulty of the problem can be removed leaving behind its difficulty-free character.

This study uses this difficulty-free problem characterization to taxonomize 40 evolutionary computation problems using a set of fifteen combinatorial graphs. This extends and, for the self-avoiding-walk problems corrects, a taxonomy of 23 problems given in [8]. The list of graphs is a winnowed subset of those used in [8]; it uses single representatives from groups of graphs whose members gave very similar time-to-solution results. These graphs are available in machine-readable form from the first author.

The remainder of this study is organized as follows. Section II gives the definition of the combinatorial graphs used in this study. Section III gives a brief definition of GBEAs. Section IV briefly describes the evolutionary computation problems used. Section V gives the analysis techniques used. Section VI gives the results. Section VII states conclusions and invites other researchers to participate in the ongoing taxonomy in the Bryden and Ashlock labs.

II. The Combinatorial Graphs Used

We assume some familiarity with graph theory [18]. A *combinatorial graph* or *graph* G is a collection $V(G)$ of vertices and $E(G)$ of edges where $E(G)$ is a set of unordered pairs from $V(G)$. Two distinct vertices of the graph are *neighbors* if they are members of the same edge. The number of edges containing a vertex is the *degree* of that vertex. If all vertices in a graph have the same degree, then the graph is said to be *regular*. If the common degree of a regular graph is k, then the graph is said to be k-regular. A graph is *connected* if one can go from any vertex to any other vertex by traversing a sequence of vertices and edges. The *diameter* of a graph is the

0-7803-9487-9/06/$20.00/©2006 IEEE

largest number of edges in a shortest path between any two of the vertices. The diameter is, in some sense, the shortest path across the graph.

This paper utilizes a nonstandard operation on graphs called *simplexification*. Simplexification at a vertex v replaces v with a cluster of vertices, one for each neighbor of v so that all the new vertices are neighbors of one another and each is a neighbor of exactly one of v's former neighbors. Simplexification of a vertex with four neighbors is shown in Figure 1. The effect of simplexification is to create small groups of vertices that are closely coupled to one another but less closely coupled to the rest of the graph. This creates an analog of a biological refuge in the graphical connection topology. By *simplexification* of a graph, we mean simultaneous simplexification of all the graph's vertices.

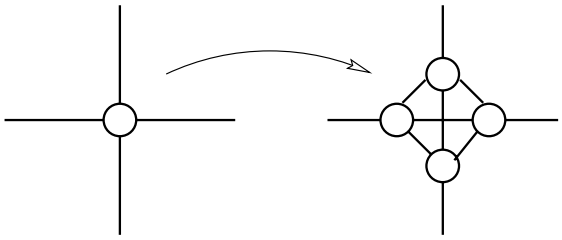

Fig. 1. Simplexification of a vertex with four neighbors.

A. List of Graphs

This sections provides some necessary mathematical definitions and describes the combinatorial graphs used in this study.

Definition 1: The *complete graph* on n vertices, denoted K_n, has n vertices and all possible edges.

Definition 2: The *complete bipartite graph* with n and m vertices, denoted $K_{n,m}$, has vertices divided into disjoint sets of n and m vertices and all possible edges that have one end in each of the two disjoint sets.

Definition 3: The *n-cycle*, denoted C_n, has vertex set $\mathbb{Z}_n$. Edges join pairs of vertices that differ by 1 $(mod\ n)$ so that the vertices form a ring with each vertex having two neighbors.

Definition 4: The *n-hypercube*, denoted H_n, has the set of all n-character binary strings as its set of vertices. Edges consist of pairs of strings that differ in exactly one position.

Definition 5: The *$n \times m$-torus*, denoted $T_{n,m}$, has vertex set $\mathbb{Z}_n \times \mathbb{Z}_m$. Edges are pairs of vertices that differ either by 1 $(mod\ n)$ in their first coordinate or by 1 $(mod\ m)$ in their second coordinate but not both. The tori are thus four-neighbor rather than eight-neighbor tori. These graphs are $n \times m$ grids that wrap (as tori) at the edges.

Definition 6: The *generalized Petersen graph* with parameters n and k, with k relatively prime to n, is denoted $P_{n,k}$ and has vertex set $0, 1, \ldots, 2n-1$. The vertices $0, \ldots, n-1$ are connected in a standard n-cycle. The vertices $n, \ldots, 2n-1$ are also connected in an n-cycle but with the ith vertex connected to the $(i+k)$th (mod n) vertex. Finally, pairs of vertices $i, n+i$ are connected.

Definition 7: A *tree* is a connected graph with no cycles. Degree zero or one vertices are termed *leaves* of the tree. A *regular balanced tree* of degree k is a tree constructed in the following manner. Begin with a single vertex. Attach k neighbors to that vertex and place these neighbors in a queue. Processing the queue in order, add $k-1$ neighbors to the vertex most recently removed from the queue and add these neighbors to the end of the queue. Continue in this fashion until the tree has the desired number of vertices. The resulting graph is a tree in which all non-leaves have degree k and which has, constructively, the smallest possible diameter among trees with all non-leaves having degree k. We denote these graphs $RBT(n,k)$ where n is the number of vertices. Notice that not all n are possible for a given k.

Definition 8: The graph Z is created by starting with $K_{4,4}$ and then simplexifying the entire graph three times.

In addition *random* graphs are used in this study. A random graph is specified by the algorithm used to create it.

Definition 9: Generate *random toroidal graphs* as follows. A set of 512 points are randomly placed onto the unit torus (the unit square wrapped at the edges, not the torus graph) and edges are created between those at distance 0.07 or less from one another. This distance was chosen to give an average degree of about 6. After generation, the graph is checked to see if it is connected. Graphs that are not connected are rejected. These graphs are denoted $R(r,1)$, where $r = 0.07$ is the radius for edge creation.

Table I lists the graphs used in this study together with some of their graph invariants. The complete graph K_{512} is included as a baseline; the fact that it is completely connected causes a GBEA using it to perform like a standard evolutionary algorithm.

Graph	Index Name	Diameter	Mean Degree
C_{512}	C512	256	2
H_9	H9	9	9
K_{512}	Complete	1	511
$P_{256,1}$	P256_1	129	3
$P_{256,17}$	P256_17	18	3
$P_{256,3}$	P256_3	46	3
$P_{256,7}$	P257_7	22	3
$R(0.07,1)$	RTor07_1	19	7.445
$T_{16,32}$	T16_32	24	4
$T_{4,128}$	T4_128	66	4
$T_{8,64}$	T8_64	36	4
Z	Z	19	4
$RBT(512,3)$	RT1n512d3	16	1.996
$RBT(512,4)$	RT1n512d4	11	1.996
$RBT(510,5)$	RT1n510d5	9	1.996

TABLE I
GRAPHS USED AND THEIR INDEX NAMES. INDEX NAMES ARE USED TO INDEX THE GRAPHS IN FIGURES.

III. GRAPH BASED EVOLUTIONARY ALGORITHMS

Graph-based evolutionary algorithms, as they are used in this study, are defined here. Clearly, many other implementations are possible. Choose a graph G with vertex set $V(G)$

and edge set $E(G)$ to use as a population structure. Place one individual on each vertex of G. A steady-state evolutionary algorithm [17], [19] is used in which evolution proceeds one mating event at a time. A mating event is performed as follows. Pick a vertex $v \in V(G)$ uniformly at random. A neighbor of v is then picked for mating using fitness proportional selection. The variation operators, crossover and mutation, are used to produce a single new individual that replaces the individual on vertex v if its fitness is no worse than that of the individual it is replacing. The variation operators used in mating events are specified for each problem individually.

IV. The Evolutionary Computation Problems

The evolutionary computation problems used in this study are listed here in no particular order. This set of problems includes *all* the problems for which the protocol described in Section V for assessing a problem type with GBEAs is available.

A. The De Jong functions

The De Jong functions $F_1 - F_5$ are described in detail in [10]. F_1 is a three-dimensional bowl. F_2 is a fourth-degree bivariate polynomial surface featuring a broad suboptimal peak. F_3 is a sum of integer parts of five independent variables creating a function that is flat where it is not discontinuous, a kind of six-dimensional ziggurat. F_4 is a fourth-order paraboloid in 30 dimensions with distinct diameters in different numbers of dimensions made more complex by adding Gaussian noise. F_5 is the so-called "foxhole" function with many narrow local optima placed on a grid. These functions are traditional test problems in function optimization but do not serve as a complete test suite. See [20] for incisive comments. All real function optimization problems in this study use two-point crossover and uniform real mutation of size 0.1.

B. The Plus-one-recall-store problem

The plus-one-recall-store (PORS) problem is described in detail in [6]. It is a type of maximum problem within the domain of genetic programming [7] with a small operation set and a calculator-style memory. The goal of the test problem, called the PORS efficient node use problem, is to find parse trees that evaluate to the largest integer result possible given a fixed maximum number of nodes. The language has two operations: integer addition and a store operation that places its argument in an external memory location. The language also has two terminals: the integer 1 and recall from an external memory. The difficulty of the PORS efficient node use problem varies strongly according to the congruence class $(mod\ 3)$ of the number of nodes permitted. Experiments on $n = 15$, $n = 16$, and $n = 17$ nodes, representing all three equivalence classes, were run. The hardest case is $n = 15$; the easiest is $n = 16$.

Fitness for a given parse tree was the size of the number it produced when evaluated. In this set of experiments, the initial population was composed of randomly generated trees with exactly n nodes. A successful individual was defined to be a tree that produced the largest possible number (these numbers are computed in [3]). Crossover was performed by the usual subtree exchange [11]. If this produced a tree with more than n nodes, then a subtree of the root node iteratively replaced the root node until the tree had fewer than n nodes. This operation is called *chopping*. Mutation was performed on each new tree produced by replacing a subtree picked uniformly at random with a new random subtree of the same size.

C. Differential equation solution

Solving differential equations is a standard genetic programming problem. Modifying the usual technique, the algorithm in this study when computing fitness extracts the derivatives symbolically. Code for solving differential equations with symbolic derivatives used in the fitness function is available by contacting the first author.

We solve the differential equation

$$y'' - 5y' + 6y = 0, \tag{1}$$

a simple homogeneous equation with a two-dimensional solution space,

$$y = Ae^{2x} + Be^{3x}, \tag{2}$$

for any constants A, B.

The parse tree language used is given in [6]. Trees were initialized to have six total operations and terminals. Fitness for a parse tree coding a function $f(x)$ was computed as the sum of the error function $E(x) = (f''(x) - 5f'(x) + 6f(x))^2$ over 100 equally-spaced sample points in the range $-2 \leq x \leq 2$. This is the squared deviation from agreement with the differential equation. This function is to be minimized, and the algorithm continues until the fitness function summed over all 100 sample points drops below 0.001. A filter was included to prevent trivial solutions, e.g., $f(x) = 0$, and trivial solutions were assigned a fitness of 1×10^{12} when they were detected. Crossover and chopping were performed as in the PORS experiments; trees were chopped if they had more than 22 total operations and terminals. In addition to subtree mutation of the sort used in the PORS experiments, a *constant mutation* was applied to each new parse tree. Constant mutation has no effect on parse trees that do not contain ephemeral real constants. For a tree that does contain such constants, one of the constants is selected uniformly at random, and a number uniformly distributed in the region $[-0.1, 0.1]$ is added to it. Each new tree produced resulted from a subtree crossover and was subjected to both a subtree mutation and a constant mutation.

D. DNA Barcodes

DNA barcodes [2] are error correcting codes [12] over the DNA alphabet $\{C, G, A, T\}$ which are able to correct errors relative to the *edit metric* [9]. They are used as embedded markers in genetic constructs to permit retention of source information when sequencing pooled genetic libraries. An example of their successful use to retrieve sequence source information appears in [14].

Unlike binary error correcting codes over the Hamming metric, edit metric codes lack a beautiful algebraic theory. Those used were located with a *greedy closure evolutionary algorithm* [2]. This type of evolutionary algorithm uses a representation consisting of a partial structure. The fitness of an individual partial structure is the quality (in this case size) of its completion by a greedy algorithm. When searching for DNA barcodes, the partial structure is a choice of three random DNA codewords, and the greedy algorithm is Conway's lexicode algorithm [2]. Fitness is simply the size of the code located by Conway's algorithm. The DNA barcode search problem exhibits a high degree of epistasis, and work thus far suggests it has an exceedingly rugged fitness landscape. The variation operators used are uniform crossover on the set of words in the partial structures, apportioning duplicate words one to each child and mutation that changes one letter of one word uniformly at random.

The algorithm in this study searches for six-letter DNA words that are at a mutual distance of at least three. These are the parameters used for the wet lab testing of the technique in [14]. Barcodes of this size and distance can correct one sequencing (edit) error.

E. The Griewangk function

The Griewangk function is a sum of quadratic bowls, one per dimension, with cosine terms added to them, subsequently translated to yield a positive function. It has a plethora of local optima and is a natural member of a test suite. As the dimension of the Griewangk function increases, it approaches a unimodal bowl [20]. For this reason, we include this function in five cases of relatively low dimension, $N = 3, 4, 5, 6, 7$.

F. The one-max problem

The one-max problem uses a string of bits for a chromosome. In this study, we used 20-bit strings. The fitness of a string is its weight (the number of 1's in it). For the one-max study, we used local roulette mating. The crossover operator was two-point crossover. The mutation operator flipped one bit selected uniformly at random. The choice not to use elite replacement on the one-max problem reflected the essentially trivial character of the problem. The search problem is harder without elite replacement and so more likely to yield information about the relative merit of graphs.

G. Ordered Gene Problems

Ordered gene problems are those using a permutation or ordered list as their representation. Two ordered gene problems were studied: sorting a list into order (the Order problem) and maximizing the period of a permutation. The *period* of a permutation is the smallest number of times it must be composed with itself to obtain the identity permutation. Both these problems are discussed in [6].

Two variation operators were used. The crossover operator functions as follows. A crossover point is chosen. The entries of the list before the point are preserved. Those after the crossover point are retained but in the order they appear in the other permutation. The mutation operator exchanges two entries of the permutation chosen uniformly at random. Both these variation operators were applied, once each, for each mating event. The list ordering problem was run for lists of length 8, 9, and 10 while the period maximization problem was run for lists of length 30, 32, 34, and 36. For both problems the sizes were selected so that the smallest size of a problem run was the first at which performance became significantly different on different graphs.

H. Parity Problems

Odd-parity is a boolean genetic programming [7] problem. The goal is to compute the truth value of the proposition "an odd number of the input variables are true." Two forms of genetic programming are used for parity in this study: simple parse trees and function stacks. Fitness for both representations is the number of cases of the parity problem computed correctly.

Simple parse trees use no ADFs, and the variation operators are as for PORS. The operations used are logical and, or, nand, and nor; the terminals are the constants true and false and the input variables. Only the three-input version of the problem was done with simple parse trees – without ADFs the problem becomes exceedingly difficult.

A *function stack* is a representation derived from *Cartesian Genetic Programming* [13], [21]. The parse tree structure used in genetic programming is replaced with a directed acyclic graph. The vertices of this graph are stored in a linear chromosome. Each node specifies a binary Boolean operation, an initial output value for that operation, and two arguments for the operation. The available Boolean operations are: and, or, nand, and nor. The available arguments are: Boolean constants true and false, the input variables, and the output of any Boolean operation with a larger index in the chromosome than the current one. Permitting references to the current output of nodes with larger indexes gives function stacks a *feed forward* topology. The binary variation operator used on function stacks is two-point crossover of the linear chromosome. The single-point mutation operator chooses a random operation three-eighths of the time, a random argument half the time, and an initial value for a node's memory one-eighth of the time. If an operation is selected, then it is replaced with another operation selected uniformly at random. If an argument is selected, then it is replaced with a valid argument selected according to the scheme used in initialization. If an initial memory value is selected, it is inverted. The 3-, 4-, and 5-odd parity problems were run with function stacks.

I. The Self-Avoiding Walk problem

The self-avoiding walk (SAW) problem uses a string as its chromosome. The string is over the alphabet $\{U, D, L, R\}$ with the letters corresponding to up, down, left, and right moves on a grid, respectively. Grids of sizes $3 \times 3, 3 \times 4, 4 \times 4, 4 \times 5, 5 \times 5, 5 \times 6$, and 6×6 are used. The length of a SAW chromosome is equal to the number of cells in the grid minus one. Fitness is evaluated by starting in the lower left

corner of the grid and then making the moves specified by the chromosome. The sequence of moves made is referred to as the *walk*. If a move is made that would cause the walk to leave the grid, then that move is ignored. The walk can also revisit cells of the grid. Fitness is equal to the number of squares visited when the walk is completed. The problem is called the *self-avoiding* walk problem because optimal solutions do not revisit squares; they are self-avoiding walks.

J. Steiner System Problems

Steiner systems [1] are used in the statistical design of experiments. Suppose we have a set V of n objects. A *Steiner k-tuple system* on V is a set of k-subsets of V with the property that every pair of elements from V appears in one and only one of the k subsets. For the set $\{A, B, C, D, E, F, G\}$ an example of a Steiner triple system would be the set of 3-tuples: $\{\{A, B, D\},\ \{B, C, E\},\ \{C, D, F\},\ \{D, E, G\},\ \{A, E, F\},\ \{B, F, G\},\ \{A, C, G\}\}$. Notice that every pair of letters is present and each appears in exactly one triple. The example given is a Steiner *triple* system on seven points.

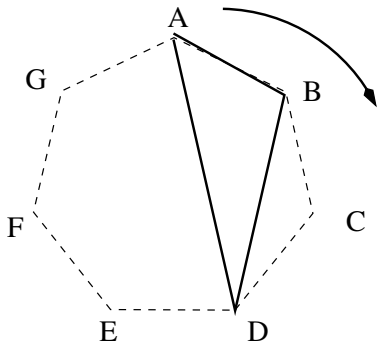

Fig. 2. A difference set for a Steiner triple system on 7 points.

Searching directly for the k-tuples that make up a Steiner system is inefficient. Figure 2 illustrates how if we place seven points on a heptagon then any two points are separated by a distance of 1, 2, or 3 (if we are allowed to go in either direction). The triangle shown in Figure 2 has a side of length 1, a side of length 2, and a side of length 3. If we rotate it in all seven possible ways, then the vertices of the seven resulting triangles form a Steiner triple system on seven points. The triangle is called a *difference set*.

Formally a *difference set* for a Steiner k-tuple system on n points is a collection of

$$\frac{n-1}{k(k-1)} \tag{3}$$

k-tuples placed on a regular n-gon, as in Figure 2, with the property that all of the distances $1 \leq d \leq (n-1)/2$ occur as the difference of two members of exactly one of the tuples. Notice that these distances are shortest distances in either direction about the polygon.

The representation we will use to search for Steiner systems is based on difference sets. It was first used in [5]. Formula 3 tells us that the representation has $\frac{n-1}{k(k-1)}$ k-tuples, or a total of $\frac{n-1}{k-1}$ integer parameters. These parameters are vertices of the polygon, organized into k-tuples. We store this in a linear chromosome, an array of $\frac{n-1}{k-1}$ integers in the range $0 \leq i \leq n-1$. The k-tuples are adjacent blocks of k integers in the chromosome. The representation uses two-point crossover and a mutation operator that replaces an integer with another picked uniformly at random. Repeated integers in a k-tuple are allowed but are penalized in a natural fashion by the fitness function.

The fitness of a collection of k-tuples is the number of distinct distances in the range $1 \leq d \leq n-1$ that occur between members of the same tuple. Repeated integers within a k-tuple generate both repeated differences (bad) and zero differences which are not scored. The instances of the Steiner problem $S(k, n)$ used in this study are: S(3,49), S(3,55), S(4,37), S(4,49), S(4,61), and S(5,41). Triple systems are denoted STSn in figures, quadruples by SQSn, and quintuples by SQun.

V. Experimental Design

For each problem and graph, 5000 independent evolutionary runs are performed to find a relatively accurate average time to solution. These average times-to-solution for each problem are normalized by first subtracting the smallest time to solution for any graph and then dividing by the difference between the largest and smallest times to solution. These normalized numbers, ranging from 0-1, represent the *relative* time to solution of a problem on each graph and hence have the intrinsic problem hardness removed. These data are used to build a neighbor-joining taxonomy and they are displayed with a non-linear projection algorithm. Each of these techniques is described below.

A. Neighbor-Joining Taxonomy

For each of the 40 problems P, a real vector $m(P)$ with 15 entries corresponding to the normalized mean solution time for each of the 15 graphs was created. For each pair of problems P and Q, the Euclidean distance $d(P, Q)$ between the vectors $m(P)$ and $m(Q)$ was then computed using the formula

$$d(P, Q) = \sqrt{\sum_{i=1}^{15} [m(P, G_i) - m(Q, G_i)]^2}.$$

$d(P, Q)$ was interpreted as the distance between the problems P and Q. An "UPGMA" tree was used to describe the taxonomic relationships among the problems.

UPGMA is a clustering method commonly used to transform distance data into a tree. It received attention in [15], and a good recent description may be found in [16]. It is especially reliable if the distances have a uniform meaning. Normalization of the numbers $m(P, G)$ makes the widely different rates of convergence comparable so that the inferred distances are appropriate for analysis by UPGMA.

UPGMA is an acronym for "Unweighted Pair Group Method with Arithmetic mean." Given a collection of taxa and distance d_{ij} between taxa i and j, the method first links

the two taxa x and y that are least distant. The taxa x and y are merged into a new unit z. For all taxa i other than x and y, a new distance d_{iz} is computed as the average of d_{ix} and d_{iy}, and it is noted that the new taxon z represents the average of two original taxa. Henceforth, x and y are ignored, and the procedure is repeated to find the next pair of taxa that are least distant. When two taxa u and v are combined into a new taxon w, the new distance d_{iw} is the average of d_{iu} and d_{iv} weighted according to the number of original taxa in u and v respectively; w contains all the original taxa in both u and v. The procedure ends when the last two taxa are merged.

When the taxonomy is displayed, as in Figure 3, vertical differences are simply equally spaced while horizontal distances are proportional to the distance between problems or the averages of problems that form the interior nodes. Problems separated by a small horizontal distance (such as Griewangk5, 6, and 7) should be regarded as very similar. Wider separations should be regarded as significant.

B. Nonlinear Projection

Nonlinear projection [4] is used to create a two-dimensional picture of an n-dimensional data set with the property that the Euclidean two-space distances are as similar as possible to the Euclidean n-space distances. It thus has applications similar to principal components analysis. It is more general because all projections of the data points into $\mathbb{R}^2$, not just the linear ones, are searched. The nonlinear projection used to display the dispersion of the 40 problems via their normalized mean-time-to solution is identical to that used in [4] and details may be found there.

VI. Results

Figure 3 gives the neighbor joining taxonomy for the problems. Figure 4 gives the result of non-linear projection of the problems into the plane. The two methods of displaying the relative distance between problems substantially agree. Inspection suggests that enlarging the taxonomy from the 23 problems given in [8] to the 40 treated here did not cause much rearrangement of the relative positions in the taxonomy *except* for the SAW problems. The runs performed for the SAW problems using the complete graph in [8] were normalized incorrectly, and the taxonomy given here corrects the mistake.

The numerical optimization problems, $F_1 - F_5$ (the De Jong functions) and the Griewangk functions are part of a subtree of the taxonomy with only the one-max and 5x5 SAW; they also cluster well in the non-linear projection. Parity problems done with function stacks scatter across the taxonomy, but examination of the individual cases suggests this is reasonable. The order of the graphs from best to worst changes substantially from the 3- to the 5-parity problem with function stacks.

The six Steiner problems, shown in [1] to have excellent scaling behavior for GBEAs, are six of the members of an eight-member subtree; this is what one would expect since "scaling well" and "problem's type does not change across different cases" are very similar notions.

The two kinds of ordered gene problems are grouped in different subtrees. This is unsurprising as the order problem is unimodal and the period problem is highly polymodal. The PORS problems, which are know to have very different fitness landscapes for different problem sizes $(mod\ 3)$, are each in a different major branch of the tree. The DNA barcode problem, which uses a very strange greedy representation, is grouped with the Steiner problems.

VII. Conclusions and an Invitation

The taxonomic technique and non-linear projection technique based on data gathered with GBEAs demonstrated here give a method of classifying evolutionary computation problems. While non-linear projection and neighbor-joining taxonomy yield similar groupings they give different views into the classification space of the set of evolutionary computation problems.

Much work remains to be done. The set of problems needs to be expanded. The expansion of the taxonomy to 40 problems, while requiring a great deal of computer run-time, did not break the taxonomic methods outlined in [8]. Another direction that should be pursued is checking how much the character of a problem changes when its representation is varied. Suppose we solve a numerical optimization problem, for example, with many different sets of variation operators. The techniques in this study could quantify the relative impact of the changes in a new way. Finally the relationship between the GBEA characterization of problem type and other characterizations must be studied.

This paper concludes with an invitation. The Ashlock and Bryden labs are interested in collaborating on the extension of the problem taxonomy given here. Other researchers are invited to submit data to the effort. The first author should be contacted for a copy of the experimental protocol and all data will be made available to any interested party through the world wide web. Substantially increasing the number of problems in the taxonomy will increase its value and permit rigorous testing of the hypothesis that taxonomically similar problems yield to similar evolutionary computation techniques.

References

[1] D. Ashlock, K. M. Bryden, and S. Corns. Graph based evolutionary algorithms enhance the location of steiner systems. In *Proceedings of the 2005 Congress on Evolutionary Computation*, volume 2, pages 1861–1866. IEEE Press, 2005.

[2] D. Ashlock, L. Guo, and F. Qiu. Greedy closure genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1296–1301, Piscataway, NJ, 2002. IEEE Press.

[3] D. Ashlock and J. I. Lathrop. A fully characterized test suite for genetic programming. In *Evolutionary Programming VII*, pages 537–546, New York, 1998. Springer-Verlag.

[4] D. Ashlock and J. Schonfeld. Nonlinear projection for the display of high dimensional distance data. In *Proceedings of the 2005 Congress on Evolutionary Computation*, volume 3, pages 2776–2783. IEEE Press, 2005.

[5] Daniel Ashlock. Finding designs with genetic algorithms. In W. D. Wallis, editor, *Computational and Constructive Design Theory*, pages 49–65. Kluwer Academic Publishers, The Netherlands, 1996.

[6] Daniel Ashlock. *Evolutionary Computation for Opimization and Modeling*. Springer, New York, 2006.

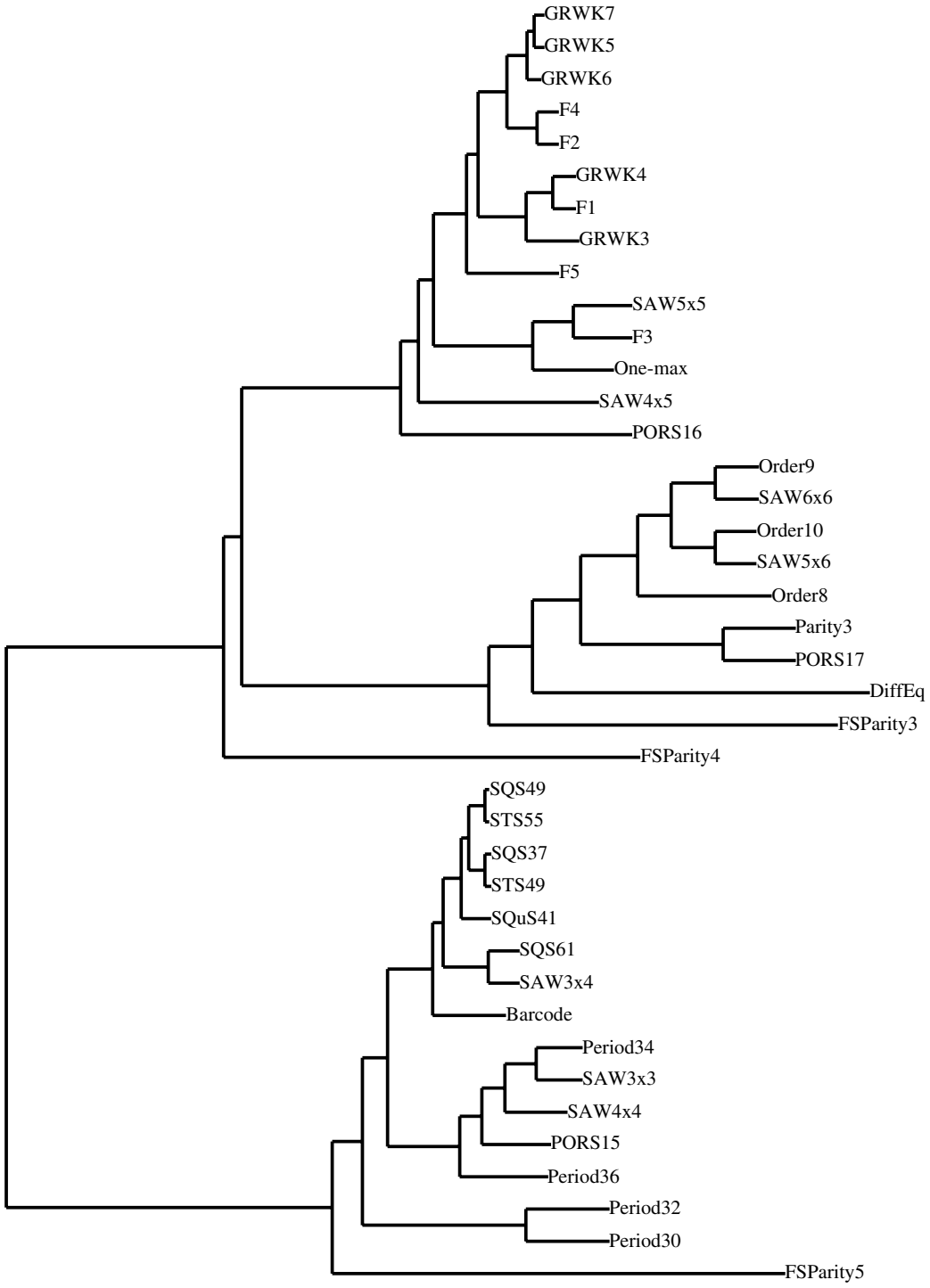

Fig. 3. The neighbor-joining taxonomy for the evolutionary computation problems used in this study. Vertical spacing is even while horizontal distance reflects the abstract distance between problems.

[7] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming : An Introduction*. Morgan Kaufmann, San Francisco, 1998.

[8] K. M. Bryden, D. A. Ashlock, S. Corns, and S. J. Willson. Graph based evolutionary algorithms. to appear in IEEE Transactions on Evolutionary Computation, 2006.

[9] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press,

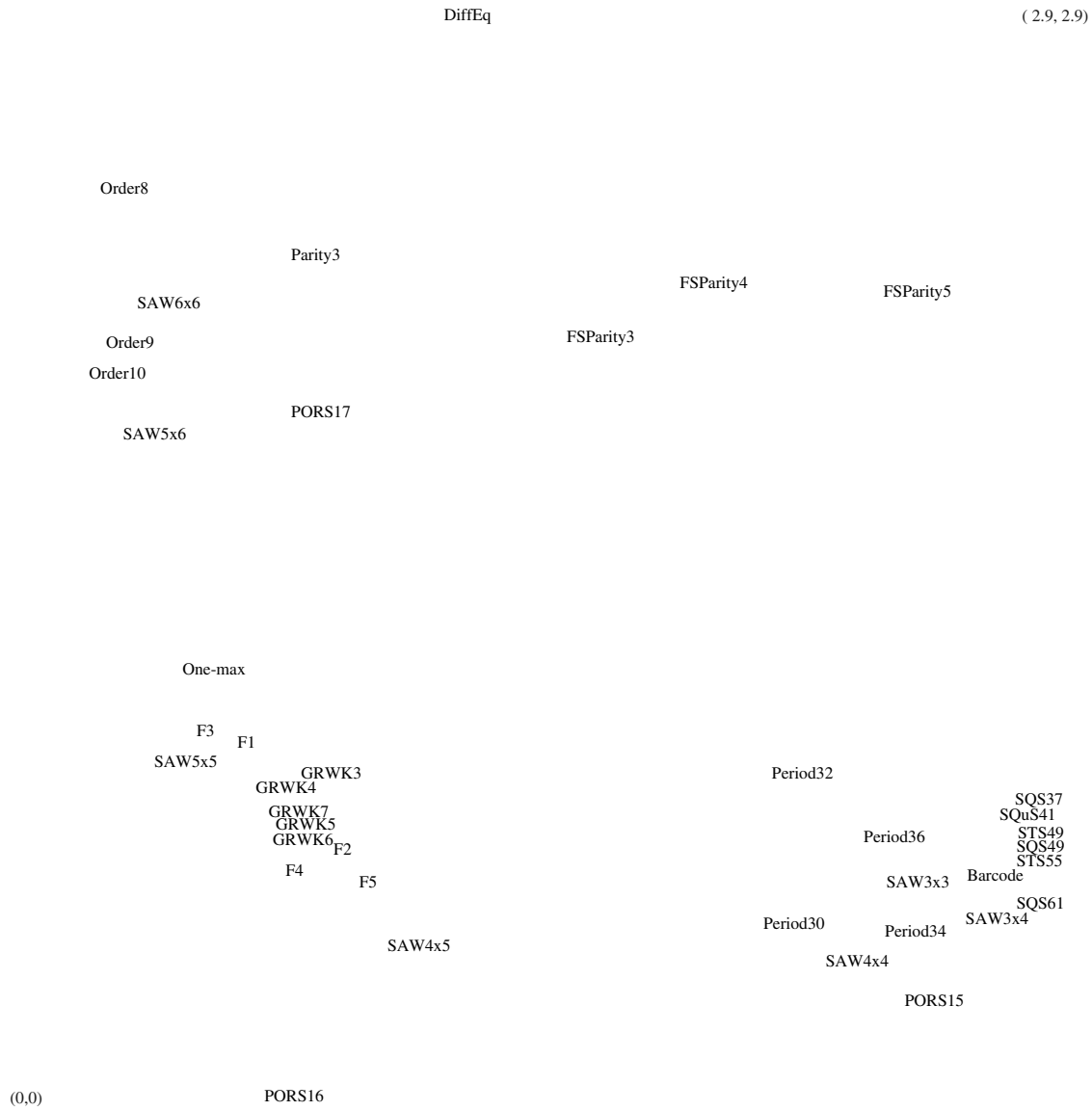

Fig. 4. A non-linear projection in two dimensions of the problem distances in 15 dimensions. The actual position of a problem on the diagram is the beginning of the string representing its name. The points at the corners give the scale of the projection.

Cambridge, Mass, 1997.

[10] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[11] J. R. Koza. *Genetic Programming*. The MIT Press, Cambridge, MA, 1992.

[12] R. McEliece. *The Theory of Information and Coding*. Addison-Wesley, Reading, Mass, 1977.

[13] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In *Proceedings of the European Conference on Genetic Programming*, pages 121–132, London, UK, 2000. Springer-Verlag.

[14] F. Qiu, L. Guo, T.J. Wen, D.A. Ashlock, and P.S. Schnable. Dna sequence-based bar-codes for tracking the origins of ests from a maize cdna library constructed using multiple mrna sources. *Plant Physiology*, 133:475–481, 2003.

[15] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy; the Principles and Practice of Numerical Classification.* W.H. Freeman, San Francisco, 1973.

[16] D.L. Swofford, G.J Olsen, P.J. Waddell, and D.M.Hillis. Phylogenetic inference. In D. Hillis, C. Moritz, and B. Mable, editors, *Molecular Systematics, second edition*, pages 407–514. Sinauer, Sunderland, MA., 1996.

[17] G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. In *Foundations of Genetic Algorithms*, pages 94–101. Morgan Kaufmann, 1991.

[18] D. B. West. *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ 07458, 1996.

[19] D. Whitley. The genitor algorithm and selection pressure: why rank based allocation of reproductive trials is best. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[20] D. Whitley, K. Mathias, and Rana J. Dzubera. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85:245–276, 1996.

[21] Tina Yu and Julian F. Miller. Finding needles in haystacks is not hard with neutrality. In *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*, pages 13–25, London, UK, 2002. Springer-Verlag.