On Taxonomy of Evolutionary Computation Problems

Dan Ashlock
Mathematics Department
Iowa State University,
Ames, Iowa 50011
danwell@iastate.edu

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

ABSTRACT

Taxonomy is the practice of classifying members of a group based on their measurable characteristics. In evolutionary computation the problem of telling when two problems are similar is both challenging and important. An accurate classification technique would yield large benefits by permitting a researcher to rationally choose algorithm and parameter setting based on past experience. A good classification technique would also permit the selection of diverse test suites that would give a useful sense of the proper domain of application of a new technique. This study uses a standard taxonomic technique, hierarchical clustering, on a set of taxonomic characters derived from a comparative study using graph based evolutionary algorithms. The results is a cladogram that classifies the problems used in a reasonable fashion. Based on this we then argue that the technique given here can be used to provide an objective, automatic, extensible classification tool for any collection of evolutionary problems and discuss possible methods for improving the technique.

I. INTRODUCTION

A taxonomy is a hierarchical classification of a set. Linnaeus established the first definite hierarchy used to classify living organisms. Each organism was assigned a kingdom, phylum, class, order, family, genus, and species. This hierarchy gave a tree structure to *taxonomy*, the enterprise of classifying living creatures. Modern taxonomy has nineteen levels of classification, extending Linnaeus' original seven. A *cladogram* is a tree diagram showing the evolutionary relationship among various taxonomic groups. The reader should see [11] for details on modern taxonomic procedures for living organisms.

Constructing trees to find and visualize relationships among members of a group is a process with more general application. Hierarchical clustering can produced a tree-structured classification of any set of data given only a similarity measure on members of the set and some sort of averaging procedure for members of the set. In this study we will extract a set of measurements or *taxonomic characters* from a collection of problems and use hierarchical clustering to produce a cladogram that classifies the problems as more and less similar. Hierarchical clustering starts with the members of a set, thought of as singleton subsets. It then joins the closest pair and replaces them with their union or average.

The choice of taxonomic characters used for clustering is critical. Imagine, for example, that you are attempting to derive a family tree for a group of twenty insect species. All have six legs and so "number of legs" is a useless character, at least within the group, for classifying the insects. The size of the insects varies a great deal with the weather, which determines the amount of food they can find. Thus size is a bad character for classification. The ratio of thorax length to abdomen length turns out both to vary within the group and remain the same in different years. The color of the larva also varies within the group and does not depend on the weather. These latter two characters may be useful. One is a real number the other is a class variable taking on the values red, white, orange, or yellow. If we are performing a numerical taxonomic analysis we will need to assign numbers in a somewhat arbitrary fashion to the colors.

The preceding brief discussion gives only a taste of the difficulty of choosing good taxonomic characters. Readers familiar with automatic classification, decision trees, and related branches of machine learning will recognize that those choosing decision variables face similar issues. Any taxonomic character or decision variable must be relevant to the decision being made, vary across the set of objects being classified, and be cleanly computable for all members of the set of objects being classified.

This study presents a source of taxonomic characters that are computable for any evolutionary computation problem that has a detectable solution or end point. These characters are numerical. We also will defend these characters as objective, in the sense that they do not favor any particular choice of representation or parameter setting. In outline these characters are computed in the following fashion. This study uses a type of evolutionary algorithm, *graph based evolutionary algorithms*. This type of evolutionary algorithm takes a geographical structure encoded as a combinatorial graph and use it to restrict mating choice. Past work on GBEAs appear in [7], [4], [1], [5]. The time-to-solution for a problem varies in a complex manner with the choice of graphical connection topology. This

complexity is itself the genesis of our taxonomic characters. The taxonomic characters used to describe a problem are the normalized mean solution times for the problem on each of a variety of graphs.

In the remaining sections of the paper we will review graph theory, explain graph based evolutionary algorithms in some detail, list the problems used in this pilot taxonomic study, give the cladogram found using the time-to-solution characters, and discuss how to improve and extend the taxonomy. While this study presents a set of objective character that enable automatic classification we do not claim these are the "right" or only characters.. We invite others to add characters and study the impact on the resulting classification.

II. Graphs Used

We assume some familiarity with graph theory [13]. 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 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, 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 longest that a shortest path between any two of the vertices can be. The diameter is, in some sense, the shortest path across the graph. In this paper a graph used to constrain mating in a population will be called the *population structure*. The general strategy is to use the graph to specify the geography on which a population lives, permitting mating only between neighbors, and finding graphs that preserve diversity without hindering progress due to heterogeneous crossover.

This paper utilizes a nonstandard operation on graphs that generates a valuable substructure, *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. By *simplexification* of a graph, we mean simultaneous simplexification of all the graphs' vertices.

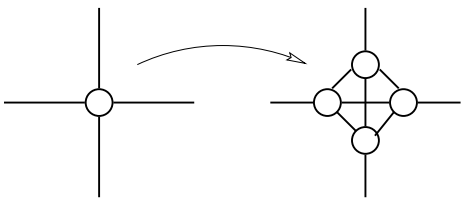

Fig. 1. Simplexification of a vertex with four neighbors

A. List of graphs

In this section we give the list of combinatorial graphs used in this study, as well as defining graphs required to describe the graphs used.

Definition 1: The *complete graph* on n vertices, denoted K_n, has n vertices and all possible edges. An example of a complete graph is shown in Figure 2.

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. The $3-pre-Z$ graph shown in Figure 2 is the complete bipartite graph $K_{4,4}$.

Definition 3: The *n-cycle*, denoted C_n, has vertex set $\mathbb{Z}_n$. Edges are 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. A 4-hypercube is shown in Figure 2.

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. These graphs are $n \times m$ grids that wrap (as tori) at the edges. A 12×6-torus is shown in Figure 2.

Definition 6: The *generalized Petersen graph* with parameters n, k, denoted $P_{n,k}$ has vertex set $0, 1, \ldots, 2n-1$. The two sets of vertices are both considered to be copies of $\mathbb{Z}_n$. The first n vertices are connected in a standard n-cycle. The second n vertices are connected in a cycle-like fashion, but the connections jump in steps of size $k(mod\ n)$. The graph also has edges joining corresponding members of the two copies of $\mathbb{Z}_n$. The graph $P_{32,5}$ is shown in Figure 2.

Definition 7: A *tree* is a connected graph with no cycles. Degree zero or one vertices are termed *leaves* of the tree. A *balanced regular 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. 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. Two of the steps leading to the graph Z are shown in Figure 2.

In addition, four classes of *random* graphs are examined. A random graph is specified by a randomized algorithm which corresponds to a type of random graph (a probability distribution on some set of graphs). We use three instances of each type of random graph used.

Definition 9: An *edge move* is performed as follows. Two edges $\{a,b\}$ and $\{c,d\}$ are found that have the property that none of $\{a,c\}$, $\{a,d\}$, $\{b,c\}$, or $\{c,d\}$ are themselves edges. The edges $\{a,b\}$ and $\{c,d\}$ are deleted from the graph, and the edges $\{a,c\}$ and $\{b,d\}$ are added. Notice that edge moves preserve the regularity of a graph if it is regular.

Definition 10: We generate *random regular* graphs by the following algorithm. Start with a regular graph and then

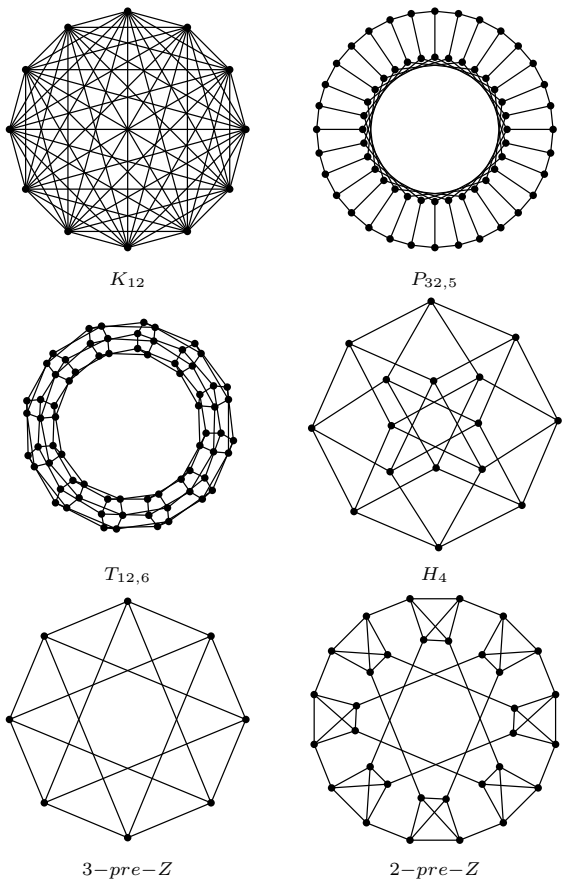

Fig. 2. Examples of complete, Petersen, Torus, and hypercube graphs, and some of the steps leading to the Z graph.

Graph	Reg.	Diam.	Mean Degree
C_{512}	**2**	**256**	**2**
H_9	**9**	**9**	**9**
K_{512}	**511**	**1**	**511**
$P_{256,1}$	**3**	**129**	**3**
$P_{256,17}$	**3**	**18**	**3**
$P_{256,3}$	**3**	**46**	**3**
$P_{256,7}$	**3**	**22**	**3**
$R(512,3,1)$	**3**	**11**	**3**
$R(512,3,2)$	**3**	**10**	**3**
$R(512,3,3)$	**3**	**10**	**3**
$R(512,4,1)$	**4**	**8**	**4**
$R(512,4,2)$	**4**	**7**	**4**
$R(512,4,3)$	**4**	**7**	**4**
$R(512,9,1)$	**9**	**4**	**9**
$R(512,9,2)$	**9**	**4**	**9**
$R(512,9,3)$	**9**	**4**	**9**
$R(0.07,1)$	**n/a**	**19**	**7.445**
$R(0.07,2)$	**n/a**	**20**	**7.805**
$R(0.07,3)$	**n/a**	**16**	**7.520**
$T_{16,32}$	**4**	**24**	**4**
$T_{4,128}$	**4**	**66**	**4**
$T_{8,64}$	**4**	**36**	**4**
Z	**4**	**19**	**4**
$RBT(512,3)$	**3,1**	**16**	**1.996**
$RBT(512,4)$	**4,1**	**11**	**1.996**
$RBT(510,5)$	**5,1**	**9**	**1.996**

TABLE I
GRAPHS USED TOGETHER WITH SOME OF THEIR PARAMETERS.

repeatedly perform 3,000 edge moves on vertices selected uniformly at random from those that are valid for edge moves. For 3-regular random graphs use $P_{256,1}$ as the starting point. For 4-regular random graphs use $T_{16,32}$ as the starting point. For 9-regular random graphs use H_9 as the starting point. These graphs are denoted $R(n,k,i)$ where n is the number of vertices, k is the regular degree, and $i = 1,2,3$ is the instance of the graph in this study.

Definition 11: We generate random toroidal graphs as follows. A set of 512 points are placed onto the unit torus (unit square wrapped at the edges), 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 was checked to see if it was connected. Graphs that were not connected were rejected so that the graphs used were chosen at random from the connected random toroidal graphs. These graphs are denoted $R(r,i)$, where $r = 0.07$ is the radius for edge creation, and $i = 1,2,3$ is the instance of the graph in this study.

III. The graph based evolutionary algorithm

This section defines a graph based evolutionary algorithm (GBEA) as it is used in this study. Assume that a graph G with vertex set $V(G)$ and edge set $E(G)$ has been chosen as a population structure. One individual is placed on each vertex of G. A steady state evolutionary algorithm [12], [15] is used. Each mating event is performed with the local mating rule of the GBEA. Picking who breeds and deciding if the new individual replaces the individual on v are together called the *local mating rule* of the GBEA. In this research the local mating rule will pick a first vertex at random and then pick neighbors in direct proportion to their fitness. A single new individual is generated and replaces the individual on the vertex selected at random if it is more fit. We call this local mating rule *local elite roulette mating*. One problem, onemax, has the child replace the parent absolutely rather than if it is better. This modification yields a harder problem (the elite onemax is too easy to yield useful data).

IV. The problems used

A suite of 21 test problems is used. In generating the data used for taxonomic analysis 5,000 simulations were performed for twenty of the test problems for each of the graphs listed in Table I. 10,000 simulations were performed for one of the test problems, solving a differential equation, to obtain better resolution on the relative ranking of graphs by mean time to solution. The number of mating events required to find a correct solution to the problem were saved. For each graph and problem the mean number of mating events to solution were normalized to yield the taxonomic characters for the problems.

Normalization consisted of subtracting the minimum average time from each average time and then dividing through by the maximum among the resulting reduced times. The taxonomic characters for each problem are thus numbers in the set [0,1] for each graph.

A. String Problems

The one-max problem uses a string of 20 bits for a chromosome. The fitness of a string is its weight (number of 1's in the string). Two point crossover and single bit-flip mutation were used.

The self-avoiding walk (SAW) problem uses a string over the alphabet $\{up, down, left, right\}$ as its chromosome. The letters correspond to moves on a grid. The SAW problem on grids of size $3x3, 3x4, 4x4, 4x5, 5x5, 5x6$, and $6x6$ is used in this study. 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. When the walk is completed fitness is set equal to the number of squares visited at least once. The problem is called the *self-avoiding* walk problem because optimal solutions may not revisit squares. Examples of SAW chromosomes and their fitness evaluations are given in Figure 3.

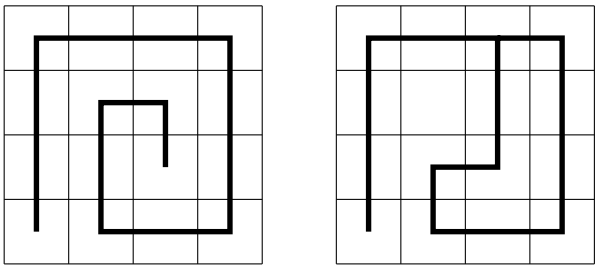

Fig. 3. Examples of an optimal and a sub-optimal walk for the $4x4$ instance of the SAW problem.

Every instance of the SAW problem has a known best fitness. This makes the collection and interpretation of statistics on algorithm behavior easier than on problems such as NK-landscapes. The expressed walk for a given SAW chromosome yields a simple and intuitive visualization that can be used to help in the analysis of the behavior of evolutionary algorithms. Two point crossover and single character mutation are used.

As SAW problems are a new type of test problem they should be checked against the list of criteria for good test suite problems given in [14]. Criterion 1: SAW problems are resistant to hill climbing. Testing with a single character mutation hill climber showed that the ratio of local to global optima located explodes combinatorially as the problem size increases. Criterion 2: the SAW problem is constructively non-linear, nonseperable, and non-symmetric. If we permute the order of moves made the fitness of a given chromosome varies widely and rapidly. Since sequences of moves are good only from a particular starting position the SAW problem is quite non-separable. Loci near the beginning of the chromosome have fitness independent of later loci, but the fitness of later loci deeply depends on the values of earlier loci. Fitness is thus not even close to additive and the problem is non-linear. Criterion 3: scalability. The SAW problems contain an infinite number of cases NxM that can be scaled from trivial to to arbitrarily hard. Criterion 4: scalable evaluation cost. This is the sole criterion that the SAW problem, to some degree, fails. The evaluation cost of a SAW problem is small when its size is such that there is any hope of solving it. A 6x6 SAW problem requires that 36 moves be checked on a 6x6 grid requiring a microscopic amount of computer power. A 100x100 SAW problem would still not require much time for a fitness evaluation. At size 100x100, however, the difficulty of the search problem will have grown to where there is no hope of finding the answer without supplying some form of additional information to the evolutionary algorithm. Thus we cannot come up with slow-to-evaluate cases of the SAW problem. Criterion 5: canonical representation. The SAW problem uses a string over a four letter alphabet. This is a canonical representation. The SAW problems thus can satisfy four of the five criteria needed for members of a good test suite and so, if paired with a problem that has scalable cost, yield an acceptable test suit.

B. Real variable problems

We use DeJong's classic five functions as well as the Griewangk function in 3-7 dimensions. The DeJong's functions $F_1 - F_5$ are described in detail in [8]. The function F_1 is a three dimensional bowl. De Jong's function F_2 is a fourth degree bivariate polynomial surface featuring a broad sub-optimal peak. Function 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. Function F_4 is a fourth-order paraboloid in thirty dimensions, with distinct diameters in different numbers of dimensions, made more complex by adding Gaussian noise added to it. Function 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 the review article by Whitley [14] for incisive comments.

The Griewangk function is a sum of quadratic bowls with cosine terms added to them, translated to be a positive function. The function has one such bowl summed per dimension. It provides a test case with a plethora of local optima and is a natural member of a test suite. As the dimension of the Griewangk increases the function becomes smoother and hence less challenging [14]. For this reason we include this function in five cases of relatively low dimension, $N = 3, 4, \ldots 7$.

C. Genetic programming problems

We used three genetic programming problems. Two were version of the plus-one-recall-store (PORS) problem, described in detail in [2]. The third was a differential equation solution problem described subsequently. PORS is a type of maximum problem within the domain of genetic programming [10], [9], [3] 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, when executed, generate the largest integer result possible given a fixed maximum number of parse tree 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. The cases for $n = 15$ and $n = 16$ nodes, respectively the hardest and easiest of the three classes, were used in this study. An example of a solution located for $n = 16$ is given in Figure 4. Fitness for a given parse tree was the size of the number it produced when evaluated. The initial population was composed of randomly generated trees with exactly n nodes. A successful individual was defined to be a tree that produces the largest possible number (these numbers are computed in [2]). Crossover was performed by the usual subtree exchange [10]. 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 less than n nodes. This operation is called *chopping*. Mutation was performed by replacing a subtree picked uniformly at random with a new random subtree of the same size for each new tree produced.

(+ (Sto (+ (+ (Sto (+ (Sto (+ (+ 1 1) 1)) Rcl)) Rcl) Rcl)) Rcl)

Fig. 4. An optimal PORS tree located by a GBEA with graph C_{512} for $n = 16$ nodes shown in LISP-like notation.

Solving differential equations is a standard genetic programming problem [10]. In addition to solving a differential equation within a GBEA, we also modify the usual GP technique by extracting the derivatives needed to compute fitness symbolically. This method is described more completely in [6].

We solve the differential equation

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

a simple homogeneous equation with a two dimensional solution space,

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

for any constants A, B.

The parse tree language used has operations and terminals given in Table II. 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 over 100 equally spaced sample points in the range $-2 \leq x \leq 2$ of the error function $E(x) = (f''(x) - 5f'(x) + 6f(x))^2$. This is essentially the squared deviation from agreement with the differential equation. This function is to be minimized and the algorithm continues until 1,000,000 mating events have taken place (this did not happen in practice) or until the fitness function summed over all 100 sample points drops below 0.001.

Terminals	
x	the independent variable.
r	ephemeral real constants.
Unary operations	
$\sim$	Negation
sin	trigonometric sine function
cos	trigonometric cosine function
atan	trigonometric arctangent function
exp	exponential function
log	natural log function
sqr	square function
Xr	scaling by an ephemeral real constant
Binary operations	
$+$	binary addition
$-$	binary subtraction
$*$	binary multiplication
$/$	binary division

TABLE II
OPERATIONS AND TERMINALS FOR THE DIFFERENTIAL EQUATION PARSE TREE LANGUAGE. THE SYMBOL R DENOTES A REAL NUMBER.

Crossover and chopping were performed as in the PORS experiments, except that trees were chopped if they had in excess of 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, either as a terminal or as a part of the unary scaling operation, one of the constants is selected with a uniform distribution and then a number uniformly distributed in the region $[-0.1, 0.1]$ is added to the constant. Ephemeral constants are initialized in the range $[-1, 1]$ but may be taken outside of this range by constant mutation. For the differential equation solution experiments we used local elite roulette mating. Each new tree produced results from a subtree crossover and is subjected to both a subtree mutation and a constant mutation.

V. THE TAXONOMIC TECHNIQUE

With the taxonomic characters available from computing normalized time to solution for 21 problems on 26 graphs the cladogram describing the relationship among the problems is computed with a nearest neighbor joining algorithm. Initially the taxonomic characters (vectors of 26 numbers) for the problem are made available to the algorithm as objects. The Euclidean distances between all pairs of objects is computed. The two closest objects were then joined. The objects joined were removed from consideration and were replaced by their average. This process is repeated until a single object remains. When average taxonomic characters were computed they were

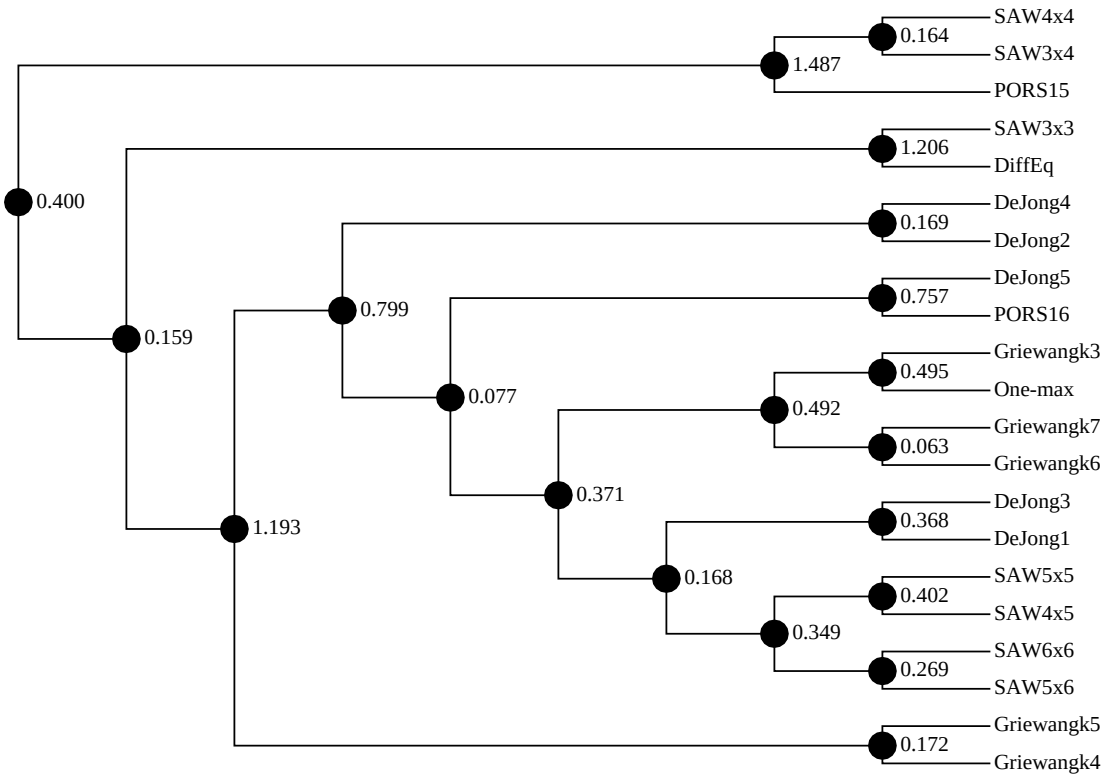

Fig. 5. The problem taxonomy derived from time-to-solution data on the graphs listed in Table II

.

weighted by the number of original characters contributing to each.

The cladogram resulting from nearest neighbor joining is shown in Figure 5. The numbers next to the join nodes represent the minimal Euclidean distances that caused the join. We will call these distances *joining numbers*. Note that these distances are not monotone with left-ward distance in the tree. This is because distances are re-computed with new taxonomic characters resulting from averaging as the tree is constructed.

VI. Discussion

Figure 5 shows a reasonable taxonomy for the problems used, given what we know about these problems. In [15] it is demonstrated that the Griewangk function smooths and becomes less difficult as the dimension increases. In three dimensions the Griewangk function is easy because of small size. The algorithm groups these three relatively easy instances of the Griewangk function together with a 20 bit one max, which was made slightly more difficult by not using elite replacement. In this discussion "hardness" is measured by mean number of mating events to effect a solution.

The two harder instances of the Griewangk function are paired together and then joined to the rest of the tree late and with a large joining number. This shows the separation from the other problems is substantial.

The SAW problem's essentially trivial 3x3 case is paired with the also near-trivial differential equation problem. These two are the only two problems that are solved in the initial random population with positive probability. They are also, according to the cladogram, correctly identified as behavioral outliers. Their joining distance is quite large indicating that joining these was one of the last things the neighbor joining algorithm did. The somewhat deceptive SAW4x4 and SAW3x3 problems are grouped with the also deceptive and far harder PORS $n = 15$ problem. The joining number of the two SAW problems with the PORS problem indicates another late joining. This deceptive grouping ended up as an outlier clade in the entire tree.

The four hardest versions of the SAW problem, 4x5-6x6, take more average time that SAW 3x4 and 4x4. The do not, however, exhibit the fitness evolution characteristic of deceptive functions. This is possibly because they possess a combinatorially exploding collection of optimal solutions. Mean fitness in these runs climbs over time and discovery of a correct solution takes place as an inevitable result of mutation. The cladogram groups these four together.

The unimodal DeJong functions F_1 and F_3 were grouped together and then, somewhat startlingly, grouped with the four highest dimensional SAW problems. While these SAW problem cases are highly polymodal the modes are mutationally

close together and so a hard unimodal search character is not implausible.

In general nodes with large joining numbers indicate that dissimilar objects or groups of objects are being joined. Likewise when large sub-trees are involved in a join then the object causing the join is highly composite. The top (leftmost) two or three nodes in the tree are joins with either large joining numbers or of very large sub-trees resulting from a lot of averaging. Ignoring these nodes would probably be sensible. Likewise the SAW3x3/DiffEq node and the node connected directly to PORS15 could be ignored. The forest resulting from the deletion of these nodes yields a reasonable set of groupings for the problems.

VII. Conclusions

It is the firm belief of the authors that the techniques presented here are a single example that can and should be extended. Our taxonomic characters, times until location of first acceptable solution, is only one of a plethora of possible characters. Likewise, GBEAs are only one types of evolutionary algorithm. Varying rates of mutation and crossover, the population size, and the problem representation all generate different "species" for the type of taxonomic analysis performed. The authors feel that the GBEA time-to-solution based characters are good ones in that they are both diverse and objective.

The pilot study presented here is an opportunistic one, built on research initially performed to study the impact of GBEAs on a fairly broad set of problems. In spite of this the process resulting is a defensible taxonomy of the problems used. This taxonomy did not make any clearly inappropriate joinings and was consonant with much of what is known about the functions used.

The taxonomy has some immediate and obvious uses. No more than three of the seven cases of the SAW problem used are needed. The DeJong functions F_1, F_2, and F_5 yield as much information as all five together. Jumping the dimension of the Griewangk function by twos seems a potentially profitable winnowing of the problems used here.

The two cases of the PORS problem used, provable possessed of different fitness landscapes, behave very differently under the probe of time to solution. The two very easy problems (SAW 3x3, differential equation solution), while none too similar, are more similar to one another than any of the other problems.

As a technique for exploring problem similarity as well as certifying and shrinking sets of test problem the technique presented here has significant potential. Adding more combinatorial graphs will increase the sharpness of taxonomic resolution. Adding more problems requires only that the evolutionary runs be performed. The technique is easily extensible to more problems and can be made more powerful by simply using more graphs.

VIII. Improving the Taxonomy

The taxonomic methods presented here is one of several that could have been used with the available data. Averaging neighbors, for example, can be replaced by taking the minimum distance between and two members of the groups being compared. In the future comparison across different neighbor joining methods and other taxonomic methods will be a priority.

The taxonomic characters chosen are numerical and, in the sense that no human judgment is required to compute them, objective. The choice of graphs used is somewhat arbitrary. It could be improved by excluding graphs that are too similar to one another and by adding additional types of graphs. In addition there are many other possible sources of characters. In addition to the mean time to solution the standard deviation was also computed. This is another potential rich source of characters, not exploited in this study for lack of space.

IX. Acknowledgments

The authors would like to thank the Iowa State Virtual Reality Applications Center for its support of this research. We would also like to thank the members of the Iowa State Complex Adaptive Systems Program for many helpful discussions.

References

[1] Dan Ashlock, Kenneth Bryden, Peter Johnson, and Douglas McCorkle. Improving data segregation with a graph based evolutionary algorithm. In *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 417–422, 2002.

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

[3] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming : An Introduction.* Morgan Kaufmann, San Francisco, 1998.

[4] Kenneth M. Bryden, Daniel A. Ashlock, Douglas S. McCorkle, and Gregory L. Urban. Optimization of heat transfer utilizing graph based evolutionary algorithms. *International Journal of Heat and Fluid Flow*, 24:267277, 2003.

[5] Kenneth M. Bryden, Daniel A. Ashlock, Douglas S. McCorkle, and Gregory L. Urban. Optimization of heat transfer utilizing graph based evolutionary algorithms. *International Journal of Heat and Fluid Flow*, 24(2):267–277, 2003.

[6] Kenneth M. Bryden, Steve Kirstukas, and Daniel Ashlock. A hybrid genetic programming approach for analytical solution of differential equations. to appear: International Journal of Smart Engineering System Design, 2003.

[7] Ashlock D., Walker J., and Smucker M. Graph based genetic algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1362–1368, San Francisco, 1999. Morgan Kaufmann.

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

[9] Kenneth Kinnear. *Advances in Genetic Programming*. The MIT Press, Cambridge, MA, 1994.

[10] John R. Koza. *Genetic Programming*. The MIT Press, Cambridge, MA, 1992.

[11] Ernst Mayr and Peter D. Ashlock. *Principles of Systematic Zoology*. McGraw-Hill, New York, 1991.

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

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

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

[15] Darrel Whitley. The genitor algorithm and selection pressure: why rank based allocation of reproductive trials is best. In *Proceedings of the 3rd ICGA*, pages 116–121. Morgan Kaufmann, 1989.