Adding Local Edge Mobility to Graph Evolution

Daniel Ashlock and Meghan Timmins

Abstract—This study extends an earlier generative representation for the evolution of graphs to include a local reconfiguration operator and a null operator. The local reconfiguration operator is shown to be more effective in evolving graphs with a particular geometric character (eccentricity sequence). The null operator permits evolution to select the number of active commands used, avoiding a problem with needing to tune a “gene length” parameter. The representation is parametrized by the probability of each command appearing in initial populations and during mutation. A parameter study leads to a number of rules of thumb for using the new representation and it is found that the number of failures to find a solution, in 3000 attempts, varies from 17 in 3000 as the parameters are changed. The representation is tested on 100 instances of the eccentricity sequence matching problem. Use of the null operator has the beneficial side effect of reducing observed variation in problem difficulty.

I. INTRODUCTION

This study adds two commands to a generative representation for graph evolution first defined in [2]. The first of these new command permits strictly local changes to the connectivity of the graph - something lacking the the earlier representation. The second command does nothing, permitting evolution to select the number of active commands used to build a graph. The new representation is tested on the eccentricity sequence matching problem introduced in [6], because this test function should benefit from permitting local adjustment of the connectivity of evolving graphs.

A graph or network is a structure with a set of nodes or vertices together with a designation of which pairs of vertices are adjacent. The adjacency relationship is reflexive and stored as a set of ordered pairs of vertices. The individual adjacencies are called edges or links. In biological applications these terms network, node, and link are more commonly used. In a mathematical and computer science context the terms graph, vertex, and edge are more common. This study will adopt the mathematical terminology. A graph $G$ will also be written $G(V, E)$ where $V$ is the set of vertices and $E$ is the set of edges. When there is an edge from vertex $v$ to vertex $u$ we will denote that edge as $(u, v)$.

The remainder of this study is organized as follows. Section II specifies the augmented representation for evolving graphs and defines the eccentricity matching test problem. Section III reviews applications of evolutionary computation to graph and networks. Section IV specifies the experimental design, Section V gives and discusses results, and Section VI gives conclusions and outlines possible next steps.

II. THE THADS-N REPRESENTATION

The T(oggle), H(op), A(dd), D(elete), S(wap), N(ull) representation is a linear representation consisting of a sequence of commands for modifying a starting graph. Given a starting graph $G(V, E)$ and vertices $u, v, w, x \in V(G)$ these commands act as follows.

- The **Toggle** command on vertices $u, v$ adds the edge $(u, v)$ to $E$ if it is not currently an edge, otherwise it removes it. It toggles the status of an arbitrary edge.
- The **Hop** operation is applied when $(u, v), (v, w)$ are both edges and $(u, w)$ is not. The operation removes the edge $(u, v)$ and adds the edge $(u, w)$. The operation locally “hops” an edge. This permits local modification of the connectivity of the graph.
- The **Add** command on vertices $u, v$ adds the edge $(u, v)$ to $E$ if it is not already there. If the edge exists already the command does nothing.
- The **Delete** command on vertices $u, v$ removes the edge $(u, v)$ to $E$ if it is present, otherwise it does nothing.
- The **Swap** command on vertices $u, v, w, x$ only works if the only two edges on the four specified vertices are $(u, v)$ and $(w, x)$. In that case, the command removes the two extant edges and adds the edges $(u, x)$ and $(w, v)$. This permits modification of the connectivity of the graph that may or may not be local.
- The **Null** command which does nothing. Use of this command permits commands to be added and removed from the list of active commands by mutation.

The null command was added after initial experiments showed that the length of the gene (the number of active commands issued) is a critical parameter. Including the null command permits evolution to select the number of active commands issued. Examples of applications of the active commands are shown in Figure 1.

The THADS-N representation encodes a family of representations. An instance of the THADS-N representation is specified by giving six probabilities $p_H, p_A, p_D, p_S,$ and $p_N$. These probabilities are indexed by the first letters of the commands and represent the probability of a command being chosen both during generation of initial population and during mutation. An instance of the THADS-N representation will be reported by specifying $(p_H, p_A, p_D, p_S, p_N)$, a vector of probabilities, or by explicitly giving those probabilities.

This notion of parametrizing a family of graph-evolution representations was introduced in [2]. In that study it was...
found that these parameters have a significant impact on performance - and that the best performing values are different on different problems. The representations used in many of the author’s previous publications are special cases of the THADS-N representation.

When using any command-based representation that specifies how to build the object, the length of the list of commands must be chosen. This length is an additional algorithmic parameter. In this study this parameter is called the representation length. Many applications of evolutionary computation use fixed-length representations where each loci specifies a part of the solution. THADS-N and its precursors act to edit an initial configuration; as we will see, having the correct number of active commands is important, which explains the value of the null command. Use of null permits evolution to select the number of active commands, below the gene length used.

A. The Test Problem

The test problem used in this study is to match the eccentricity sequence of a graph. At this point, the graph theory needed to define this test problem is given. A good introduction to graph theory is [29].

Definition 1: Each edge in a graph has two vertices at its ends. We say these vertices are adjacent and call them neighbors.

Definition 2: A path is a sequence of pairs of vertices joined by edges.

Definition 3: A graph is connected if, for each pair \( u, v \in V(G) \) of distinct vertices in \( G \) there is a path beginning with \( u \) and ending with \( v \).

Definition 4: For a graph \( G \) and vertices \( u, v \in V(G) \) the path distance from \( u \) to \( v \) is either the length of the shortest path from \( u \) to \( v \) or infinite if no such path exists.

We note that the path distance places a metric space structure on a graph [11]. In this study we will work with connected graphs, forcing connectivity by awarding a very bad objective function score to graphs that are not connected.

Definition 5: The eccentricity of a vertex is the largest distance any other vertex is away from it in the path metric.

Definition 6: The eccentricity sequence of a graph is a list of the eccentricities of its vertices sorted into descending order. Sequences are written in exponential notation, e.g. \( 4^{20}3^{12} \) denotes a graph with twenty vertices of eccentricity 4 and twelve with eccentricity 3.

We now have sufficient machinery to specify the test problem and fitness function. Given an eccentricity sequence, the test problem is to find a graph that has that eccentricity sequence. Section IV specifies how the 100 eccentricity sequences used in this study were obtained. The fitness function in this study rewards a graph for having an eccentricity sequence close to a target eccentricity sequence. When the fitness drops to zero, a solution to the problem has been located.

Definition 7: For a target eccentricity sequence \( E \) the eccentricity sequence match fitness (ESM-fitness) of a graph \( G \) with a number of vertices equal to the length of \( E \) is the RMS error of the eccentricity sequence of \( G \) with \( E \) when both are sorted into descending order.

This fitness function has an infinite number of different instances with difficulty ranging from trivial to impossible and with problem sizes selectable via the number of entries in the target sequence. A fitness of zero represents a perfect match. For most possible eccentricity sequences, many graphs can have a given eccentricity sequence. Exceptions are sequences like \( 1^n \) which has a unique graph with the sequence - the graph on \( n \) vertices with all possible edges.

III. PREVIOUS WORK ON EVOLUTION OF GRAPHS

Graphs have been featured in the evolutionary computation literature almost since its inception. The target has typically been to locate a property of a particular graph or network rather than to search the space of networks. Evolving the connection topology of an artificial neural net (ANN) [30], [25] is possibly the most common appearance of evolved networks, but the graph-theoretic properties of these networks are not examined closely by most researchers. These graphs are assessed by examining their trainability as neural nets, something that leverages the large body of literature on training neural nets [18].

Proper coloring a graph is the problem of assigning colors to vertices so that the ends of each edge are distinct colors. When the vertices are meetings and the edges represent common membership in the groups that are meeting, finding a proper coloring with the fewest possible colors is equivalent to finding the meeting schedule that uses the least slots. In [16], [14] evolutionary algorithms are used to locate proper colorings of combinatorial graphs. The precursor to the THADS-N representation [2] was used to locate graphs that are difficult to properly color.

Additional research has focused on using graph properties - such as the interplay between orderings and colorings of vertices [13] or the relationship between a graph’s chromatic number and its acyclic orientations [10]. Others like [15]...
have found problem-independent constraint-handling mechanisms that can be used to improve evolutionary algorithm performance. These mechanisms are used to find optimal colorings or evolve “programs to color all graphs belonging to a class whose members all have the same number of nodes and other common attributes” [10]. Finally, some researchers have drawn further inspiration from nature and utilized ant-like behavior to generate solutions to the graph coloring problem [12]. While the representation and methods of implementation may vary, the understanding that evolutionary algorithms are able to find graph coloring solutions just as well, if not better, than standard methods remains constant.

Another application of evolutionary computation to graph theory is the location of minimal cutsets. These are minimal collections of links that if deleted will disconnect a graph. Work on this problem appears in [19], [1]. The problem of clustering a graph into sets of tightly coupled subgraphs or communities is the subject of [20], [17]. This task is important in the analysis of large data where data objects have some natural connection that induces edges on the data.

While there is a lack of literature on the direct evolution of graphs using evolutionary algorithms to satisfy a fixed fitness function, other than work on evolving epidemic graphs summarized subsequently, much exists on the co-evolution of graph or network structures and game-playing agents [21], [22], [23], [24], [27], [28]. In most of these cases, agents begin playing an evolutionary version of a simple game - prisoners dilemma, snowdrift, stag-hunt, etc - on a predetermined network structure such that the network determines the other agents with whom one can interact. As play progresses, the network structure changes to reflect decisions made by the agents. In many cases, to many defections results in the breaking of a connection while multiple decisions to cooperate can result in additional connections being made. These co-evolutionary models have many obvious real-world applications, as most social networks are not static but rather change dependent on the actions of those within them. Additionally, this method could be used to evolve new graphs, but the new graphs will likely not be dramatically different from the original seed graph.

A good deal of past work on graph evolution has applied graph evolution to the creation of contact networks that exhibit a desired epidemiological behavior. In [3] the authors evolve networks with long lasting or locally severe epidemics using combinatorial representations that strongly limit the space of graphs searched. In [4] a more general representation that conserves the contact numbers of vertices is introduced and the impact of changing the average contact number is investigated on the same fitness function as [3]. This representation requires a starting graph that implicitly specifies the contact numbers. This study also introduces a new type of evolutionary algorithm, the restarting recentering algorithm, that updates its starting point and restarts periodically. A novel analysis tool called diffusion characters are used for analysis of evolved networks in [5]. In [7], [8] a new representation that can vary contact numbers, the toggle-delete representation is introduced. In these papers graphs are evolved to fit epidemic profiles that specify the number of individual that contract a disease at each time-step of a simulation. This permits the fitting of plausible contact graphs to epidemic data.

Algorithm 1: Eccentricity Sequence Generator

Start with a 32-cycle.
Randomly toggle five edges
Repeat
For(i=1..20)
	Perform five edge swaps
	Toggle a random edge
	Extract the eccentricity sequence
	Save the sequence if it is not already found
End For
Until(100 distinct sequences located)

IV. Experimental Design

The EMS-fitness function can be used on an infinite number of different eccentricity sequences. In order to obtain a thorough test of the evolutionary algorithm used in this study, a large collection of problem instances are used. The 100 eccentricity sequences for the 32-vertex graphs used were generated using Algorithm 1.

The vertices of a cycle have very large eccentricity values and a cycle has very few edges. In any connected graph, having a small number of edges will lead to higher eccentricity values. Almost all toggle results applied to a graph with few edges will add edges, reducing eccentricities. Swaps and hops will, with very high probability, reduce eccentricity values or disconnect a graph; this latter outcome will kill the graph as disconnected graphs are awarded the worst possible fitness. The choice of 20 iterations per step in Algorithm 1 was made based on preliminary experiments. These showed that the eccentricities had reached an eccentricity sequence of the form 4a5b6c for the 100th sequence located. The collection of sequences used in this study is the same as those used in [6]. These eccentricity sequences are available in electronic form from the first author on request.

A. The Evolutionary Algorithm

The evolutionary algorithm used is steady state [26] using size seven single tournament selection. This model of evolution chooses seven population members, with the two most fit undergoing reproduction to replace the two least fit. An instance of tournament selection is called a mating event. The chromosomes are strings of THADS-N commands applied to an initially empty graph - one with no edges. Reproduction copies the parents and then performs two point crossover of the string of commands on the copies. Mutation chooses a number of loci to modify uniformly at random in the range 1-3. Loci in the chromosome are also selected uniformly at random and those loci are replaced with new commands.

The algorithm operates on a population of 100 chromosomes for 500,000 mating events, reporting summary
statistics every 5000 mating events. For each of the 100 different eccentricity sequences, 30 independent instances of the evolutionary algorithm were run. The number of runs that failed to exactly match the given eccentricity sequence were also recorded. The number of failures out of 3000 runs is a primary reporting tool.

The parameters specified above, other than representation length and the various operation probabilities $p_a$, are based on the earlier studies that this one builds on, adopting parameters that work well for the ESM fitness function. This study initially examines representation length and then focuses on setting the probability-of-used parameters for the different commands.

B. Experiments Performed

An initial set of experiments to assess good representation length was performed for the $(0.25,0.25,0.5,0,0,0)$ instance of the THADS-N representation; the last zero is vacuous because the null operation had not yet been added. The representation lengths 40, 48, 56, 64, and 96 were tested. The number of failures, in 3000 runs, are given in Table I. These preliminary results showed that the representation length was a problem - one that was solved by the addition of the null command. A gene length of 56 was fixed for the rest of the experiments. Since the null operator can only reduce the effective representation length, the length larger than the best fixed length is a logical choice.

<table>
<thead>
<tr>
<th>Rep. Length</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>157</td>
</tr>
<tr>
<td>48</td>
<td>18</td>
</tr>
<tr>
<td>56</td>
<td>73</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
</tr>
<tr>
<td>96</td>
<td>321</td>
</tr>
</tbody>
</table>

By fixing the gene length at 56 with the null operator, it became possible to focus on comparing the performance of different instances of the THADS-N representation with differing probabilities of using each command. Table II lists the experiments, their operation probabilities, and the number of times the algorithm failed among the 3,000 instances of the evolutionary algorithm run for each experiment. The experiments listed in Table II do the following.

1. uses an equal probability of all commands.
2. uses and equal probability of all commands except swap.
3. uses equal probabilities of all commands except delete.
4. uses equal probabilities of all commands except delete and swap.
5. uses equal probabilities of toggle, add, and delete.
6.-(12) use a significantly larger probability of toggle, hop, or add with lower probabilities for each of the other commands, including null. Hereafter swap and delete are not used.

- (13)-(15) use a slightly larger probability of toggle, hop, or add with lower probabilities for each of the other commands, including null.
- (16)-(18) use a slightly larger probability of T, H, or A with lower probabilities for each of the other commands
- (19)-(21) use a significantly larger probability of T and H, T and A, or A and H with lower probabilities for each of the other commands
- (22)-(24) use a slightly larger probability of T and H, T and A, or A and H with lower probabilities for each of the other commands
- (25)-(27) these experiments vary the probability of null as opposed to toggle.

C. Analysis Tools

In addition to counting the number of runs that exhibit failure, shown in Table II, two other reporting tools are used for analysis. As we will see, the number of failures was far from evenly distributed among the different eccentricity sequences tested. A scatter-plot of time to solution versus fraction of failures, plotted using the index number of the eccentricity sequence, is used to examine which sequences were difficult. Box plots of the log of time to solution are also used. Histograms of the number of failures for each problem instance appear along the bottom of the box plots.
V. Results and Discussion

Summarized in Table I, the preliminary experiments leading to the implementation of the null operator demonstrate there is great sensitivity of the system to the representation length. Of the experiments performed with the null operation, all of which used representation length 56, a total of thirteen out of twenty-nine had lower failure rates than the length 56 experiment without null. This suggests that the null operator is quite helpful, given the exploratory nature of the other experiments.

Another factor in the initial five length-setting experiments is that the failures were far from evenly distributed among the problem instances. Almost all the failures occurred in the problems with the smallest numbers in their eccentricity sequence. This issue will be revisited in Section V-B. We now turn to the parameters study on command probabilities.

Experiment 1 confirms that, if edges are to be moved, the hop operator is a better choice than the swap operator on the eccentricity match problem. Based on the failure data, the best parameters setting for the THADS-N representation for the 100 examples of EMS-fitness was Experiment 3: (0.25,0.25,0.25,0,0,0.25), though Experiments 10, 19, 21, 24, and 27 yielded very similar results. These instances of the THADS-N managed to find a graph with an exactly matching eccentricity sequence for all targets.

Runs 8 and 11, and to a lesser degree 2, show that the number of toggle and add operations must be large enough to generate enough edges, a critical factor in choosing probabilities for the THADS-N representation. While evolution can move evolving populations to have a higher number of edges via selection, some choices of probabilities make this too hard.

The failure rates in Experiments 6, 7, and 8 show that too much hop is bad. While hop can reconfigure edges in a manner that changes eccentricity sequences gently, if there are a large number of hops there may no be enough adds and toggles. Large numbers of hops are also more likely to undo or modify what others have done, which is less efficient that simply evolving the hop that is needed. The outcome of experiment three suggests that the number of null commands needs to be large enough to permit effective selection of the number of active commands. The failure rates in 6 and 7 are more than double that of 3. This supports they hypothesis that adding the null command removes the difficulty of finding the correct representation length. Similarly the relatively high failure rates in Experiments 17 and 18 are coupled with a very low probability for the null action.

A. The Effect of Null on Problem Difficulty

Figure 2 shows log-time to solution for all problem cases in Experiments 3, 4, and 14. The value of $p_N$ for these three experiments is 0.25, 0.0, and 0.1. First contrasting Experiments 3 and 4, both of which have sufficient ability to generate edges, it appears that the availability of the null action goes a very long way toward evening out problem difficulty. This is consistent with the hypothesis that each problem has an individual preferred representation length for active commands. When the null command is not available, more complex combinations of commands - like toggling the same edge twice - are required to avoid unneeded commands.

Examine Experiment 14, which has 80% of its probability allocated to toggle and add - edge generators. Here the low probability of hop appears to have increased the variability of problem hardness substantially. In addition, the relatively low probability of null may have made the problem of selecting the number of active commands more difficult.

Experiment 3 required more time to solve the easier problems and was faster to solve the harder ones. This demonstrates that the parameters it used are the most reliable for time-to-solution.

B. Problem Difficulty and Edge Density

Figure 3 shows scatterplots for the six runs with the smallest number of failures, where the axes are fraction of failure and mean time to solution of successful runs. The plotted numbers are the index numbers of the experiments.

- As already noted, failure is not evenly distributed across the problems. Problems 91, 92, and 96 appear as failures in all six experiments. All experiments with index numbers below 69 have zero failures.
- Log-time to solution is well correlated with problem index in these scatterplots. All the problem cases with indices in the 80s have lower failure rates than those with indices in the 90s, for example.
- It is worth noting that there are several experiments with high index numbers that have relatively few failures. This means that while hard problem instances are more likely when the eccentricity sequence has many small values, this quality is not reliably predictive of hardness.

Experiments 17 and 18 have higher number of failures. Their scatterplots are shown in Figure 4. The pattern of failure is very different. Experiment 17, which de-emphasized hop, has larger numbers of failures on the same problem that the six experiments with the lowest failure rate had trouble with. Problem 18, which emphasized hop at the expense of providing edges with toggle, has a small numbers of failures for any particular problem, but these failures occur on a large number of different problems. This supports the notion that, if enough edges are provided, that hop is good at rearranging those edges to match a given eccentricity sequence.

VI. Conclusions and Next Steps

This study demonstrated two important results:

1) The new hop operator is far superior for providing edge mobility than the older swap operator in contexts, like eccentricity sequence matching, where the long-range rearrangement of edges is problematic.

2) The original sensitivity of the TADS, THADS and their precursor representations to the length of the string of commands used can be greatly lessened by introducing the null operator.
Fig. 2.Shown are the distribution of log-time to solution for all 100 instances of the eccentricity match problem in Experiments 3, 4, and 14 (bottom). The differences in problem difficulty are far smoother when the null action is available.
An additional result, which verges on the painfully obvious, is that (i) either initial edges must be provided in the starting graph that is being edited or (ii) the probability of commands that generate edges must be set in a manner consistent with having enough edges to solve the problem under consideration.

This study continues the process of developing the idea of parametrized families of representations started in [2]. THADS-N can be specialized to reproduce the representations used in all the previous studies published in the author’s group. This makes it a quite versatile tool and opens the door to adaptive representation. If the probabilities that control operation use were encoded in the evolving organisms, then they could evolve to adapt the representation to the problem in question. If the restarting-recentering technique was used [3], then the character of the region being searched changes with each restart - this would make adaptive representation even more valuable.

A. Shaping in THADS-N

In [9] the idea of shaping representations was introduced.

Definition 8: In evolutionary computation, a shape is a restriction imposed on the way a representation may be instantiated to create population members.

While trying to match eccentricity sequences, some of the experiments achieved very high failure rates, hitting 100% in two experiments (8 and 11) and hitting 98.8% when the swap operator was used (Experiment 1). Evolution can overcome the default probabilities given selection and enough time; in these experiments there was not enough time. Consider this issue more deeply. If hop commands are issued at vertices that are not yet part of a two-edge chain, those hop commands are ignored. Toggles that occur before adds are, in effect, adds; there are no edges to turn off.

A natural way to shape an instance of the THADS-N representation is to require that toggle and add commands come before hop and swap commands which, in turn, come before delete commands. This would mean that all available edges are present before hops and swaps are executed. It also astronomically reduces the size of the search space evolution is dealing with.

B. Return to Epidemiology

In the earlier studies fitting a graph to an epidemic behavior, very large numbers of mating events were required to obtain a good fit. The number of links desired was put into the starting graph and the swap operator alone was used to reshape the graph. Given that the eccentricity sequence of the graph is strongly related to its epidemic behavior - long
chains in the graph topology yield longer lasting epidemics because chains of infection are longer, it seems quite possible that the hop operator, in place of the swap operator, would improve the epidemic graph fitting behavior. Checking this is an early priority for future research.

REFERENCES