Evolving Random Graph Generators: A Case for Increased Algorithmic Primitive Granularity

Aaron S. Pope*†, Daniel R. Tauritz* and Alexander D. Kent†
* *Department of Computer Science,*
Missouri University of Science and Technology,
Rolla, Missouri 65409
† *Los Alamos National Laboratory*
Los Alamos, New Mexico 87545
Email: aaron.pope@mst.edu, dtauritz@acm.org and alex@lanl.gov

***Abstract*—Random graph generation techniques provide an invaluable tool for studying graph related concepts. Unfortunately, traditional random graph models tend to produce artificial representations of real-world phenomenon. Manually developing customized random graph models for every application would require an unreasonable amount of time and effort. In this work, a platform is developed to automate the production of random graph generators that are tailored to specific applications. Elements of existing random graph generation techniques are used to create a set of graph-based primitive operations. A hyper-heuristic approach is employed that uses genetic programming to automatically construct random graph generators from this set of operations. This work improves upon similar research by increasing the level of algorithmic sophistication possible with evolved solutions, allowing more accurate modeling of subtle graph characteristics. The versatility of this approach is tested against existing methods and experimental results demonstrate the potential to outperform conventional and state of the art techniques for specific applications.**

1. Introduction

Graphs are a powerful tool for modeling a wide variety of concepts. Social, computer, transportation or communication networks are common examples. Others include infrastructure applications such as power or water distribution systems. The transmission patterns of contagious diseases are also commonly modeled using graphs. Because these concepts translate so well to graphs, many application specific algorithms are designed to work directly with the graph representations. For example, computer networks use graph theory to avoid problematic cycles in traffic routing [1].

When new graph algorithms are developed, they typically need to be tested on a variety of graphs to demonstrate versatility and scalability. For some applications, information is readily available to create graphs which model real-world data, such as actual computer networks. In other application domains, this data is in limited supply. For example, deploying wireless sensors to build a graph model can be infeasibly expensive. In these situations, researchers have commonly turned to random graph generation to test their graph algorithms.

However, not all random graph generation techniques are suitable for all applications. Certain types of random graphs are better at naturally representing specific concepts. For instance, wireless sensor network deployment is typically modeled with random geometric graphs, because this random graph model captures the physical proximity requirement needed for short-range communication. When generating random graphs for testing the performance of a new graph algorithm, it is important to select an appropriate random graph model. Whether or not a partitioning algorithm can find high-quality partitions of random geometric graphs is of little importance if the algorithm is intended to be applied to enterprise computer networks, for example.

The selection of a random graph model for a specific application is typically done by comparing a few characteristics of the graph, such as degree distribution or edge density. These coarse selection methods have the potential to miss some of the more subtle characteristics of the concepts to be modeled. For instance, a preferential attachment random graph model produces the power-law degree distribution observed in social networks, but the process might not capture the common presence of certain community structures. When less obvious graph characteristics are not considered, the random graphs produced are likely to be artificial representations of the actual concept. For this reason, random graph generators that are specifically tailored to certain applications can improve the accuracy and appropriateness of these graphs as conceptual models.

Manually developing an application specific random graph model is a complicated process, even when the model only needs to capture a single characteristic, such as an arbitrary degree distribution [2]. Hyper-heuristics employing genetic programming (GP) have been used in the past to automate the process of developing novel algorithms that are customized to an application [3]. This work leverages the power of GP to create new random graph generation algorithms that are capable of capturing the more subtle graph characteristics often missed by traditional techniques.

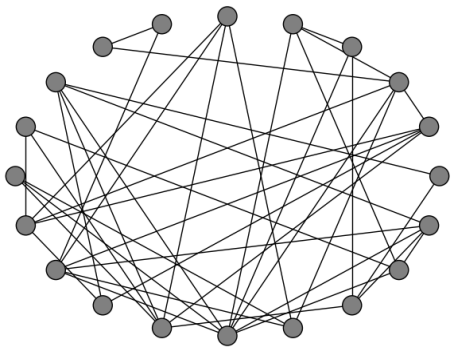


Figure 1. Erdös-Rényi random graph for $n = 20$ with $p = 0.2$.

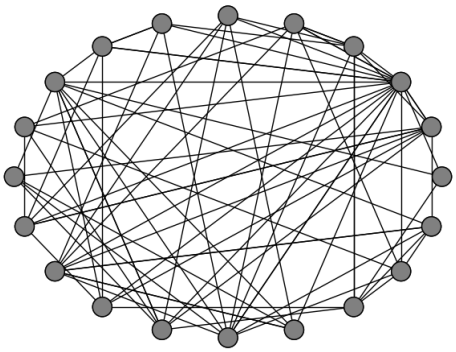


Figure 2. Barabási-Albert random graph with $n = 20$ and $c = 2$.

2. Background

Random graphs and their applications have been studied extensively in previous research. Traditional random graphs are usually described in terms of the mathematical model used to generate them; two of the most common random graph models are Erdös-Rényi and Barabási-Albert.

2.1. Erdös-Rényi Random Graph Model

The Erdös-Rényi random graph model, usually referred to as $G(n, p)$, is one of the most basic models, but also one of the most studied [4], [5]. Each of the possible $\binom{n}{2}$ edges is included in the graph with probability p. See Algorithm 1 for an implementation of the Erdös-Rényi random graph model, and Figure 1 for an example graph it generated.

Algorithm 1 Erdös-Rényi random graph

```
1: procedure G(n, p)
2:     G ← newGraph()
3:     for i ∈ 1 ... n do
4:         u ← newVertex()
5:         for v ∈ G.vertices do
6:             if random() < p then
7:                 G.addEdge(u, v)
8:         G.addVertex(u)
9:     return G
```

This simple random graph model has proven useful for demonstrating a variety of graph theoretic properties [6]. Unfortunately, it has also been shown to poorly represent real-world systems such as computer networks due to the vertex degree values produced, which follow a Poisson distribution [7], [8].

2.2. Barabási-Albert Random Graph Model

The Barabási-Albert model [9] improves upon the unrealistic degree distribution of the Erdös-Rényi model. Instead of using a constant probability for including each edge, each new vertex is connected to c existing vertices that are chosen with probability proportional to their degree. As a result, high-degree vertices are connected to more often. This phenomenon is referred to as *cumulative advantage* or *preferential attachment* and produces a power-law degree distribution that is common in graphs which model real-world networks [10], [11]. The Barabási-Albert model is implemented in Algorithm 2 and Figure 2 shows a random graph created using the Barabási-Albert model.

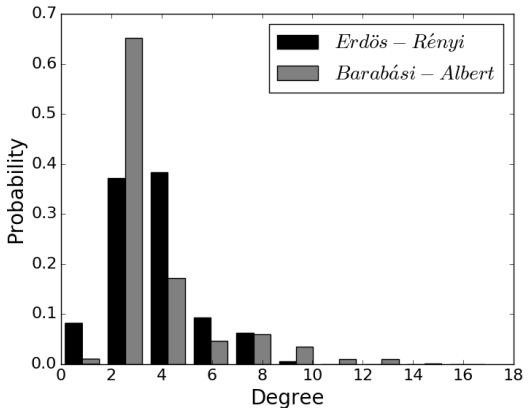


Figure 3. Comparison of degree distributions for Erdös-Rényi ($n = 20$, $p = 0.2$) and Barabási-Albert ($n = 20$, $c = 2$) random graph models.

Algorithm 2 Barabási-Albert random graph

```
1: procedure G(n, c)
2:     G ← newGraph()
3:     for i ∈ 1 ... n do
4:         u ← newVertex()
5:         for j ∈ 1 ... c do
6:             v ← randomVertexByDegree(G)
7:             G.addEdge(u, v)
8:         G.addVertex(u)
9:     return G
```

Figure 3 compares the degree distribution of Erdös-Rényi and Barabási-Albert random graphs. Although graphs produced by the Barabási-Albert model are similar to real-world networks in terms of their degree distribution, the process still resembles Erdös-Rényi in that it places no

limitations on which pairs of vertices can be connected. Many real-world networks have restrictions on connections that cannot be captured by the Erdös-Rényi and Barabási-Albert models. For instance, social networks are often modeled using graphs where vertices represent people and edges correspond to a relationship between two people. Although it is possible for any two people to form a relationship (especially in the case of online social networks), it is more likely for relationships to form between individuals if they are physically nearby, or share mutual contacts. Unfortunately, traditional random graph models lack the complexity needed to reflect such considerations.

3. Related Work

Instead of producing a single random graph with the desired properties, this research aims to provide random graph generation heuristics. One possible solution would be to employ heuristic selection techniques. Machine learning has been used to automate the process of selecting the best heuristic for a problem from a set of available heuristics with high accuracy [12]. Unfortunately, this approach is limited by the quality and variety of the set of predefined heuristics; an optimal solution to a given problem cannot be selected if it is not already present in the heuristic set. Techniques that are capable of generating entirely new heuristics help avoid this limitation.

A heuristic that searches to find or create new heuristics is known as a *hyper-heuristic* [3], [13]. Unlike *metaheuristics* [14], which search within the space of possible problem solutions, hyper-heuristics search within the space of possible problem heuristics. This means that instead of searching for a direct solution to a specific problem, hyper-heuristics can select, create, or adapt a heuristic that efficiently finds a solution to the specified problem.

Hyper-heuristics most commonly employ *genetic programming* (GP) to search a problem-specific space of algorithmic primitives. GP is a field of evolutionary computation, which uses a biologically inspired process to evolve a population of solutions to a given problem. In GP, the solutions being evolved take the form of programs or heuristics. One common method of representing these program solutions is through the use of parse trees [15].

Previous work has demonstrated the potential for GP to evolve custom random graph generation algorithms. Bailey et al. evolved new random graph generation algorithms that mimic the output of traditional random graph models [16]. Harrison implemented a similar approach and studied the use of various graph similarity metrics during solution evaluation [17]. Both of these works assume a common structure for the random graph generator solutions, which can be seen in Algorithm 3. Three components of this process are controlled by the evolved parse tree solutions. Graph initialization (line 3) determines if the graph is initially empty, or contains some basic topology, such as a ring. Inside the main loop body, the graph is "grown" by adding new vertices one at a time. The edge addition step (line 6) determines which vertices, if any, a new vertex is connected to as it is added. During finalization (line 8), existing edges can be removed or rewired at random.

Algorithm 3 Basic structure of a random graph generator

```
1: procedure RANDOMGRAPH(n)
2:     G ← newGraph()
3:     initializeGraph(G)
4:     for i ∈ 1 ... n do      // "grow" loop
5:         u ← newVertex()
6:         addEdges(u)
7:         G.addVertex(u)
8:     finalizeGraph(G)
9:     return G
```

This common structure is obviously inspired by traditional random graph generation techniques, such as those discussed in Section 2. This representation lends itself well to reproducing traditional models, as well as new models that are similar in structure. However, this restriction on the structure limits the search space of possible random graph generating algorithms. For that reason, evolved solutions suffer some of the same drawbacks as the traditional models when attempting to accurately simulate certain types of networks.

This work aims to improve on previous research by relaxing the restrictions on the basic algorithm structure. A more expansive set of operations are made available to the GP when constructing graph generation algorithms. The new operation set breaks down some of the constructs used in previous work into lower level functionality. For example, the basic "grow" loop is replaced by more general *for* and *while* loop operators, as well as basic *if* and *if/else* conditionals. A larger set of primitive operations will increase the search space of possible algorithm solutions. Previous work has demonstrated that using a larger set of primitive operations can increase the evolution time required to reach convergence, but also improve the overall final solution quality [18].

4. Methodology

In order to accommodate a wide variety of possible applications, some of which might involve multiple competing measures of quality, a multi-objective optimization approach is employed. The nondominated sorting genetic algorithm II (NSGA-II) [19], which promotes population diversity without a significant increase in complexity, is used to evolve a population of random graph generating algorithms.

Representation: Solution algorithms are represented using strongly-typed GP parse trees [20]. While it has been demonstrated that the choice of representation can impact the overall performance of the GP [21], the choice of representation was made to isolate the effect of changing the primitive operation set when comparing against previous work.

Initialization: An initial population of parse tree solutions is randomly constructed from the available input and

TABLE 1. NSGA-II AND GP PARAMETER VALUES

Parameter	Value
Population size and offspring per generation	400
Iterations per evaluation	10
Minimum initial parse tree height	3
Maximum initial parse tree height	5
Recombination probability	65%
Mutation probability	35%

operation nodes. A configurable maximum height parameter is used to limit the size of the initial parse tree solutions. Ramped half-and-half solution generation is used, which produces full parse trees of maximum height for half the population and variable height trees (up to the maximum) for the remainder.

Evaluation: Solutions are evaluated in terms of multiple objectives. The size of the parse tree of a solution is used as a minimization objective to prevent the trees from growing to impractical sizes during evolution. Other objectives used depend on the application, but typically evaluate some metric of the graphs produced by the solution. For example, one objective could be how closely the generated graphs match the degree distribution of the graphs the generator is meant to reproduce. Any objective that evaluates the graphs produced by a solution are calculated by generating multiple graphs and taking the average objective value.

Parent Selection: Standard NSGA-II parent selection is used, which consists of binary tournaments that favor solutions in less dominated Pareto fronts. Ties are broken using NSGA-II's solution distance metric to encourage population diversity.

Recombination: Due to the destructive nature of parse tree variation operators, offspring are generated using either recombination or mutation, not both. If a pair of parent solutions are selected for recombination, two offspring are produced using random subtree crossover.

Mutation: If recombination is not selected, an offspring is created by cloning a single parent, then performing random subtree replacement.

Survival Selection: NSGA-II's elitist survival selection is used. This approach selects solutions from the least dominated Pareto fronts. Solution diversity is encouraged by using the distance to other solutions in the objective space to break ties for partial Pareto fronts.

Parameters: The parameters for NSGA-II and GP initialization can be seen in Table 1. These values were automatically tuned using a random restart hill climbing search.

Primitive Operation Set Solution individuals are constructed from the set of terminal and operation nodes shown in Table 2. Except where individually noted, all operation nodes have at least one child operation node, allowing for variable length sequences of operations. The available values for *prob_from_integer*, *integer_constant* and *prob_constant* were chosen to be able to recreate or expand upon the functionality of previous work [16].

5. Experiment

The flexibility of the implementation is tested by evolving random graph generators for two example applications. The first application tests the ability of the GP to evolve algorithms which mimic traditional random graph generation techniques. Another application targets a random graph process that generates identifiable communities of well connected subgraphs.

5.1. Traditional Random Graph Models

Random graph generator solutions are evolved to recreate the behavior of two traditional techniques: Erdös-Rényi ($n = 100$, $p = 0.05$) and Barabási-Albert ($n = 100$, $c = 2$). The model parameter values were selected to produce small, sparse graphs, since a large number of these graphs will need to be generated throughout the course of evolution. Solutions are evaluated by how similar the graphs they produce are to graphs generated by the target method. In [17], Harrison demonstrated that when evaluating graph similarity for purposes of guiding evolution, there are diminishing returns in terms of solution quality as the number of different metrics used is increased. Comparing the set of degree, betweenness [22] and PageRank [23] centrality distributions was found to strike a balance between evaluation complexity and solution quality. For this reason, these three metrics will be used as competing objectives. For each distribution, the objective value is set to the test statistic returned by a Kolmogorov-Smirnov (KS) test comparing the distributions produced by both methods. This method has been used to demonstrate similarity in distributions before [17], and produces a natural minimization objective as the more similar the distributions, the lower the test statistic will be.

For comparison, a GP developed in previous work is also used to evolve generators targeting this model. See [16] for the implementation details of that approach. Both algorithms are run until convergence is detected by ten consecutive generations with no change to the non-dominated Pareto front, as described in [24]. In order to select a representative solution for comparison, the objective values of the final populations are normalized and summed for each solution. Since all objectives are minimization, the solution with the lowest objective value sum is selected. The final solution chosen from each method is used to generate 100 random graphs, and the objective values of these graphs are compared using Wilcoxon rank-sum tests at a 95% significance level.

5.2. Random Community Graphs

Algorithm 4 describes a process of creating a random graph with k communities. Vertices within the same community are connected with probability p_1. Vertices from different communities are connected with probability p_2. If $p_1 \gg p_2$, edges will be more likely within communities, making them tightly connected. Figure 4 shows an example of a graph generated with the random community model

TABLE 2. PRIMITIVE OPERATION SET

Operation Name	Description
root	Initializes empty graph, executes child operations, returns final graph
for_index_range	Executes subtree a number of times equal to an integer input value
for_node/edge_loop	Executes subtree once for each node (vertex) or edge in input list
do_while_loop	Executes subtree repeatedly until input conditional is false
if(_then)	Branching based on an input conditional
noop	"no-op", terminates sequence of operations
create_ring/clique/star	Add edges incident to an input list of nodes to create a ring, clique, or star topology
connect_to_nodes_with_prob	As create_star, but add edges according to an input probability
add_edges_with_prob	As create_clique, but add edges according to an input probability
remove/rewire_edges(_with_prob)	Removes or rewires input edges from the graph (optionally according to an input probability)
add_pairwise_edges(_with_prob)	Add pairwise edges connecting two input node lists (optionally according to an input probability)
create_triangles(_with_prob)	As add_pairwise_edges, but for triplets of nodes taken from three input lists
add_stub	Add node to a queue of nodes awaiting edges
connect_stub	Pops node from queue of nodes awaiting edges, connects to another input node
get_all_nodes/edges	List of all nodes or edges
get_incident_nodes/edges	Nodes incident to an input list of edges, or edges incident to an input list of nodes
get_internal_edges	Returns the list of edges whose endpoints are both within an input list of nodes
list_intersection/union	Intersection or union of two lists
list_filter_with_prob	Randomly filters a list according to an input probability
list_portion	First $\lfloor l * p \rfloor$ elements of the input list of length l for probability p
list_shuffle	Returns randomly re-ordered input list
sort_nodes/edges_by_map	Sorts a list of nodes or edges using a $(node : value)$ or $(edge : value)$ mapping
node_degree/betweenness/closeness_map	Returns a mapping of node centrality values for an input node list
edge_degree/betweenness/closeness_map	Returns a mapping of centrality values for the incident nodes of an input edge list
average/max_degree	Current average or maximum degree
node/edge_count	Current number of nodes or edges present
true/false_constant	Constant boolean terminal
true_with_prob	True or false according to a probability
bool_and/or	Logical conjunction or disjunction of inputs
less_than	True if the first input numeric is less than the second input, false otherwise
math_add/subtract/multiply/divide/modulus	Standard math operations (note: division by zero instead divides by 10^{-10})
prob_add/subtract/multiply/divide/modulus	Same as previous, but clamps output to $[0, 1]$
prob_from_integer	Returns probability from $[0.01, 0.02, 0.025, 0.05, 0.1, 0.2, 0.3, \ldots, 1.0]$ selected using input integer
prob_from_float	Floating point input clamped to $[0, 1]$
integer_constant	Constant chosen randomly from $\{0, 1, 2, \ldots 9\}$
prob_constant	Constant chosen randomly from $\{0.001, 0.01, 0.02, 0.025, 0.05, 0.1, 0.2, 0.3, \ldots, 1.0\}$

using a force based layout. Both random graph GP implementations are run targeting this graph model to determine if evolution can reproduce the underlying community structure.

Algorithm 4 Random community graph generator

```
procedure COMMUNITYGRAPH(n, k, p1, p2)
    G ← newGraph()
    for i ∈ 1...n do
        u ← newVertex()
        G.addVertex(u)
    for i ∈ 1...n do
        u ← getVertex(i)
        for j ∈ i+1...n do
            if (i mod k) == (j mod k) then
                if random() < p1 then
                    G.addEdge(i, j)
            else
                if random() < p2 then
                    G.addEdge(i, j)
    return G
```

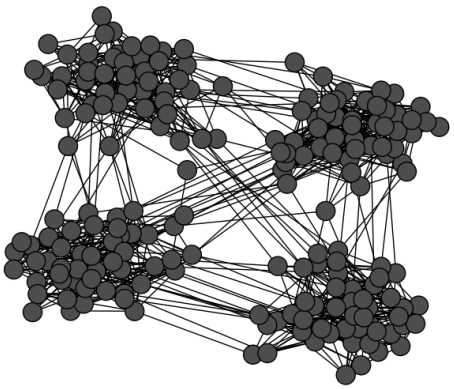


Figure 4. Graph generated using Algorithm 4 with $n = 200$, $k = 4$, $p_1 = 0.2$, and $p_2 = 0.005$.

6. Results

This section shows some representative experimental results and associated statistical tests. The next section discusses their implications.

TABLE 3. ER OBJECTIVE VALUE COMPARISON

	Low-GP			High-GP	
Metric	**Mean**	σ	**Comparison**	**Mean**	σ
Degree	0.101	0.048	=	0.108	0.047
Betweenness	0.104	0.031	=	0.105	0.033
PageRank	0.110	0.032	=	0.112	0.029

TABLE 4. BA OBJECTIVE VALUE COMPARISON

	Low-GP			High-GP	
Metric	**Mean**	σ	**Comparison**	**Mean**	σ
Degree	0.058	0.025	=	0.060	0.021
Betweenness	0.127	0.049	=	0.127	0.043
PageRank	0.112	0.037	<	0.130	0.044

6.1. Reproducing Erdös-Rényi

Figures 5a and 5b show the parse trees of the best solutions produced by the low-level and high-level GP approaches, respectively. Figure 6 shows the centrality distribution comparisons for graphs produced using the Erdös-Rényi random graph model (Actual), the high-level GP from previous research (High-GP), and the low-level GP implemented in this work (Low-GP). In each case, both methods are able to closely mimic the required distribution.

The results of the statistical comparison are shown in Table 3, with '<', '=', and '>' indicating better, equivalent, and worse performance, respectively. For all three objectives, the performance difference between the solutions produced by each GP method is statistically insignificant.

6.2. Reproducing Barabási-Albert

For brevity, the parse trees and distribution comparisons produced for the Barabási-Albert random graph model application are omitted, but the statistical comparison of the objective values achieved for the two GP implementations is shown in Table 4. These results indicate that the low-level GP statistically outperforms the high-level GP in terms of PageRank distribution.

6.3. Reproducing Random Community

Table 5 indicates that the solution produced by the low-level GP statistically outperforms the solution produced by the high-level GP in terms of all three objective values. The reason for this discrepancy in performance is obvious when examining sample graphs produced by each solution. Figure 7a shows a graph produced by the high-level GP solution, while Figure 7b shows one created by the low-level GP solution. The low-level implementation clearly does a better job of capturing the community structures present in the original model.

7. Discussion

Not only are both GP implementations able to almost perfectly reproduce the Erdös-Rényi (ER) graph model, but they both converge quickly on a good set of evolved solutions. This does not come as much of a surprise, however, because the ER model is the simplest of the three applications considered. Although this certainly provides a proof-of-concept, the particular result is not likely to be of much use considering how few real-world applications can be accurately represented using the ER model.

TABLE 5. RANDOM COMMUNITY OBJECTIVE VALUE COMPARISON

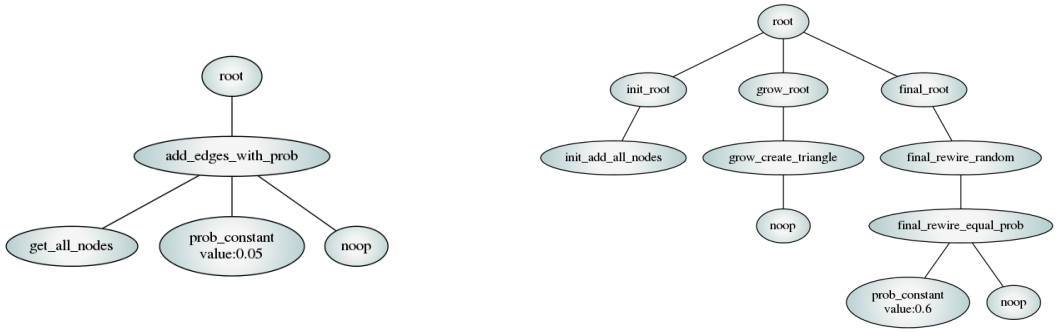
	Low-GP			High-GP	
Metric	**Mean**	σ	**Comparison**	**Mean**	σ
Degree	0.436	0.075	<	0.458	0.055
Betweenness	0.209	0.105	<	0.320	0.126
PageRank	0.127	0.029	<	0.150	0.036

Both GP approaches are able to recreate the behavior of the Barabási-Albert model reasonably well; however, the low-level GP solution manages to achieve a more accurate PageRank distribution. Although in about 90% of the experimental runs, the low and high-level GPs need about the same number of fitness evaluations to converge on good solutions, it is worth noting that in the remaining 10% of the cases, the low-level GP requires almost twice as many evaluations to converge. This is evidence of the drawback of increasing the search space with a lower-level implementation. While the low-level representation allows for a wider range of algorithm possibilities, it also increases the difficulty of finding any specific algorithm. However, the required a priori time of the hyper-heuristic is not typically of critical importance, since this time investment is amortized over repeated uses of the evolved solutions.

The random community graph model application, on the other hand, highlights the strengths of the lower-level implementation. The richer primitive operation set is better able to capture the underlying community structure of the model, which is very obvious when comparing graphs produced by the resulting solutions. This is a promising result for applications that require a more accurate model than what can be achieved by simply comparing one or two basic graph metrics, such as degree distribution. The fixed structure of the high-level GP approach limits the information it can consider when deciding how to place edges. It is easy to imagine any number of graph applications where more information is needed when placing edges. For instance, graphs that model power grids need to account for geographic proximity when connecting two devices due to the nature of the physical properties of the object that the edge represents.

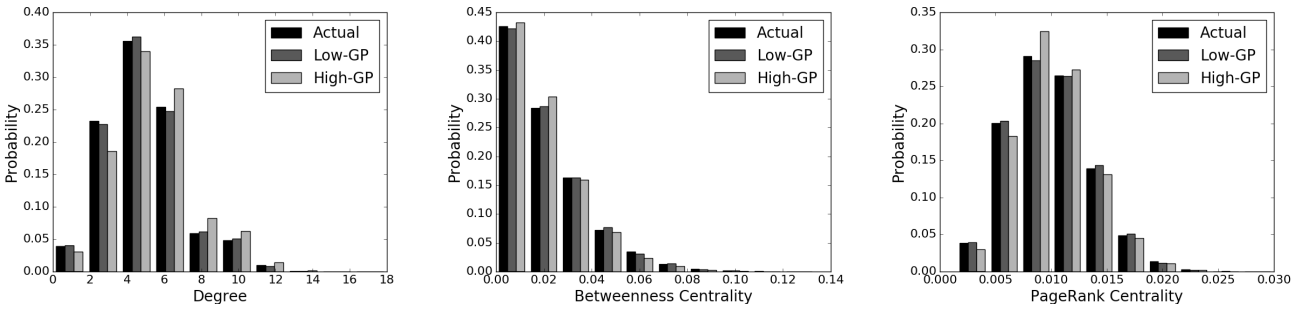
8. Conclusion

Random graph models provide an invaluable resource in many research domains. Conventionally, a traditional random graph model is selected to produce graphs which represent some application specific concept. The selection process is usually based on a small set of graph similarity measures, and this process can even be automated using machine learning techniques. Unfortunately, a selected model



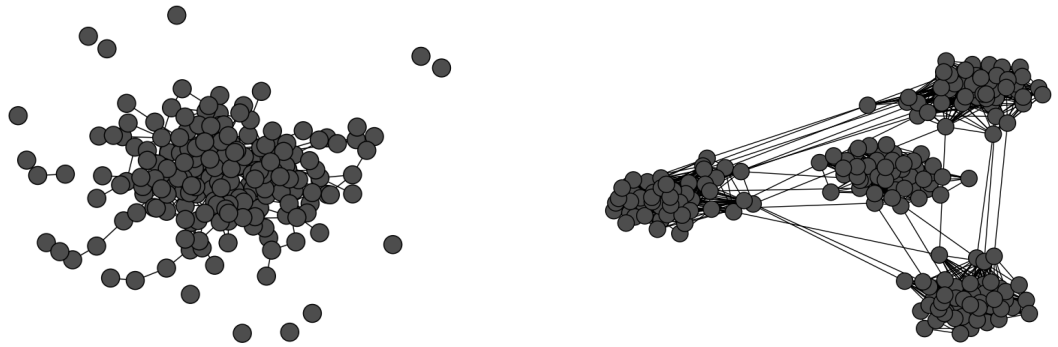
(a) Low-level GP (b) High-level GP

Figure 5. Random graph generators produced by both GP approaches when targeted to reproduce the Erdös-Rényi random graph model.



(a) Degree (b) Betweenness (c) PageRank

Figure 6. Comparison of centrality distributions for Erdös-Rényi random graph model as well as two evolved graph generators.



(a) High-level GP (b) Low-level GP

Figure 7. Graphs generated by both GP solutions trained on random community graphs.

might only be an accurate representation with respect to a few graph characteristics, leading to artificial graphs. The model selection process also relies on the set of available models; if an accurate model is not present in the selection set, this approach cannot generate a new, high-quality model tailored to the particular application.

The goal of this research was to address this limitation by automating the development of accurate random graph models for new applications. The platform implemented in this work features a richer set of lower-level primitive operations than those that have been used in the past, allowing for more expressive algorithm representation. The increased flexibility makes it is possible to evolve more sophisticated algorithms that can truly capture a wider range of graph characteristics. Experimental results illustrate that the less restricted representation is capable of capturing more subtle details of a random graph model than normally possible with conventional methods. However, this improvement in modeling accuracy can come at the cost of additional evolution time. This trade-off might not be acceptable for some applications, but the approach still has potential when the accuracy of the model is of utmost importance.

9. Future Work

There are several obvious possibilities for continuation of this work that might further improve the quality of the random graph generators produced. Some amount of improvement might be possible with a GP variant that uses an alternate solution representation, such as a stack-based GP [21].

The potential of this implementation could be further demonstrated by applying it to real-world complex network applications. Possibilities include modeling large scale computer networks, or disease transmission patterns. The more flexible representation also allows for the possibility of using this approach to evolve algorithms that alter existing graphs. With very little modification, this could be used to create random dynamic graphs in addition to the static applications already studied.

In this work, as well as previous research, the quality of random graph generators is measured solely by how much they resemble graphs taken from other sources. It is worth noting that this method of evaluation is not strictly required. Alternative methods of evaluation are possible as long as the detection of desirable and undesirable graph characteristics can be implemented as a fitness or objective function. This would allow for the creation of suitable random graph generators with absolutely no knowledge of the process used to create them.

This work demonstrates the potential of evolving graph related algorithms. Other types of graph algorithms, such as graph partitioning heuristics, could also benefit from the same approach. Since random graph models are often used to test the performance of graph algorithms, it is possible that the results might be further improved by co-evolving such graph algorithms along with the random graph generators used to test them. For example, a competitive co-evolution strategy could be used to evolve new graph partitioning algorithms while encouraging random graph generators that produce graphs that are difficult to partition. This could be used to develop a collection of highly specialized graph partitioning algorithms instead of using general purpose algorithms off the shelf.

References

[1] R. Perlman, "An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN," in *ACM SIGCOMM Computer Communication Review*, vol. 15, no. 4. ACM, 1985, pp. 44–53.

[2] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Phys. Rev. E*, vol. 64, p. 026118, Jul 2001. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.64.026118

[3] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[4] P. Erdös and A. Rényi, "On Random Graphs, I," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[5] ——, "On the Evolution of Random Graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[6] ——, "On the Strength of Connectedness of a Random Graph," *Acta Mathematica Hungarica*, vol. 12, no. 1-2, pp. 261–267, 1961.

[7] D. J. Watts and S. H. Strogatz, "Collective Dynamics of 'Small-world' Networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[8] M. Newman, *Networks: An Introduction*. New York, NY, USA: Oxford Univ. Press, 2010.

[9] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[10] D. J. Price, "Networks of Scientific Papers," *Science*, vol. 149, no. 3683, pp. 510–515, 1965.

[11] S. Wasserman, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[12] P. D. Hough and P. J. Williams, "Modern Machine Learning for Automatic Optimization Algorithm Selection," in *Proceedings of the INFORMS Artificial Intelligence and Data Mining Workshop*, 2006, pp. 1–6.

[13] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, *Handbook of Metaheuristics*. Boston, MA: Springer US, 2003, ch. Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pp. 457–474.

[14] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing: an international journal*, vol. 8, no. 2, pp. 239–287, 2009.

[15] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[16] A. Bailey, M. Ventresca, and B. Ombuki-Berman, "Genetic Programming for the Automatic Inference of Graph Models for Complex Networks," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 405–419, 2014.

[17] K. R. Harrison, "Network Similarity Measures and Automatic Construction of Graph Models using Genetic Programming," 2014.

[18] M. A. Martin and D. R. Tauritz, "Hyper-Heuristics: A Study On Increasing Primitive-Space," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO Companion '15, 2015, pp. 1051–1058.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[20] D. J. Montana, "Strongly Typed Genetic Programming," *Evol. Comput.*, vol. 3, no. 2, pp. 199–230, Jun. 1995.

[21] S. Harris, T. Bueter, and D. R. Tauritz, "A Comparison of Genetic Programming Variants for Hyper-Heuristics," in *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 2015, pp. 1043–1050.

[22] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977. [Online]. Available: http://www.jstor.org/stable/3033543

[23] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120.

[24] T. Goel and N. Stander, "A Study on the Convergence of Multiobjective Evolutionary Algorithms," in *Preprint submitted to the 13th AIAA/ISSMO conference on Multidisciplinary Analysis Optimization*, 2010, pp. 1–18.