Automated Design of Tailored Link Prediction Heuristics for Applications in Enterprise Network Security

Aaron Scott Pope
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri
Los Alamos National Laboratory
Los Alamos, New Mexico
aaron.pope@mst.edu

Daniel R. Tauritz
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri
dtauritz@acm.org

Melissa Turcotte
Los Alamos National Laboratory
Los Alamos, New Mexico
mturcotte@lanl.gov

ABSTRACT

The link prediction problem, which involves determining the likelihood of a relationship between objects, has numerous applications in the areas of recommendation systems, social networking, anomaly detection, and others. A variety of link prediction techniques have been developed to improve predictive performance for different application domains. Selection of the appropriate link prediction heuristic is critical which demonstrates the need for tailored solutions. This work explores the use of hyper-heuristics to automate the selection and generation of customized link prediction algorithms. A genetic programming approach is used to evolve novel solutions from functionality present in existing techniques that exploit characteristics of a specific application to improve performance. Applications of this approach are tested using data from a real-world enterprise computer network to differentiate normal activity from randomly generated anomalous events. Results are presented that demonstrate the potential for the automated design of custom link prediction heuristics that improve upon the predictive capabilities of conventional methods.

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**; *Anomaly detection*; *Neural networks*; • **Mathematics of computing** → *Graph algorithms*;

ACM Reference Format:
Aaron Scott Pope, Daniel R. Tauritz, and Melissa Turcotte. 2019. Automated Design of Tailored Link Prediction Heuristics for Applications in Enterprise Network Security. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3319619.3326861

1 INTRODUCTION

The link prediction problem involves predicting the existence of a relationship between entities. This problem occurs in a number of research domains and applications. Social networks have used link prediction to suggest new contacts [22]. Media streaming platforms leverage the technique to recommend material based on a customer's viewing history [16]. Link prediction techniques have also seen use within cybersecurity to differentiate anomalous behavior from normal activity [19]. Three network security applications involving the identification of abnormal activity are used to demonstrate proof-of-concept in this work.

Graphs provide a natural method of representing data about relationships between entities. In the social networking example, vertices in the graph represent network users and edges indicate connections between users. Utilizing a graph representation makes it possible to leverage graph theoretical approaches to network analysis. For example, social networks often use graph-based algorithms to detect communities within the network [5].

A common graph-based approach for predicting the existence of edges in a graph aims to position vertices in a space of latent (or hidden) features. Distances within this space are used to predict the likelihood of connections between the vertices. Information about the entities the vertices represent can be used to define these features. For example, a movie streaming application might ask a customer to choose their favorite movies or rank various genres to characterize the customer's interests.

However, many applications are restricted to the information contained in the topology of an existing network when making predictions for new links. A variety of techniques have been developed to predict new links based solely on existing edges (and the weights on those edges, if available) in a graph. Many of these techniques work by factorizing the graph's adjacency matrix. This approach produces a set of vectors that can be used as features of the vertices within the graph. Different methods of factorizing matrices have been used including singular-value decomposition [12], eigenvector decomposition [9], and Poisson matrix factorization [8].

The various factorization methods differ in terms of complexity, efficiency, and predictive capabilities for a number of applications. The ongoing research developing new methods demonstrates that the optimal link prediction technique likely depends on the specific application. Heuristic selection approaches can be used to identify the best algorithm for an application from a pool of candidate solutions, but this is limited by the quality and variety of available algorithms. New heuristics can be developed for additional applications, but this process can be difficult or expensive.

This work investigates the use of a generative hyper-heuristic search to automate the design of novel link prediction heuristics that are tailored to improve predictive performance on specific

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326861

problem applications. Functionality is extracted from existing techniques to create a set of primitive graph-based operations. Genetic programming (GP) is used to combine these operations to generate new heuristic solutions. The evolutionary search guides this process to optimize predictive performance for a specific problem. Results show that increases in prediction accuracy are possible for multiple real-world applications using this approach.

2 BACKGROUND

This section covers background information on graph representation and the link prediction problem, including some commonly used link prediction techniques. Also included is some background on hyper-heuristics, which are leveraged in this work.

2.1 Graphs and Adjacency Matrices

A graph, $G(V, E)$, is made up of a set of vertices V and a set of edges between these vertices $E \subseteq V \times V$. If vertices $v_i, v_j \in V$ are connected, an edge (v_i, v_j) exists in E. The edge information can also be expressed as an adjacency matrix with dimensions $|V| \times |V|$. For each vertex $v_i \in V$, the i^{th} row and column of A described the outgoing and incoming edges of v_i. For vertices v_i and v_j, the value at $A[i][j]$ will be zero if no edge connects the vertices, or non-zero otherwise. For an undirected graph, A is symmetric as all incoming edges are also outgoing. If the graph does not contain self-loops, the diagonal entries of A are all zero. For graphs with weighted edges (as seen in this work), a non-zero value at $A[i][j]$ indicates the weight of the edge (v_i, v_j).

2.2 Link Prediction

The link prediction problem involves predicting the presence of a relationship between two entities. In graph terms, the goal is to determine the likelihood of an edge existing between any two vertices. One simple approach to this problem is to consider the relative popularity, or tendency to connect, of the two potential edge endpoints. Conventionally, this is calculated using the formula $1-exp(-2p_ip_j)$ where p_i and p_j are the popularity values of vertices v_i and v_j, respectively [2]. For simple undirected graphs, a vertex's degree can be used as an approximation for its popularity value. In this work, the Node Popularity (NP) score uses the natural log of the weighted degree of the vertex as the popularity metric for that vertex. The logarithmic transformation can improve accuracy when very active links have weights that are orders of magnitude greater than those of low activity links. This bursty activity behavior is common in many real-world applications including those targeted in this work.

2.3 Adjacency Matrix Decomposition

The adjacency matrix representing a graph can be mathematically decomposed into two or more new matrices. For a normal decomposition, the decomposed matrices can be used to reconstruct the original adjacency matrix, often by repeated matrix multiplication. For example, singular value decomposition (SVD) decomposes a matrix $A \in \mathbb{R}^{n \times n}$ into the orthogonal matrices $U, V \in \mathbb{R}^{n \times n}$ and the diagonal matrix $D \in \mathbb{R}^{n \times n}$, where $A = UDV$. The n rows of U and columns of V can be used as feature vectors to characterize the vertices in the graph.

Making predictions with the matrices produced by normal decomposition is difficult because the matrices only capture the exact information about existing connections in the graph. Predictions based on these inputs will perfectly reproduce the original graph's connections. Fortunately, methods exist that do a better job of predicting unseen edges by finding approximations for the decomposed matrices. For instance, the rank r truncated singular value decomposition instead produces the matrices $U_r \in \mathbb{R}^{n \times r}$, $V_r \in \mathbb{R}^{r \times n}$, and the diagonal matrix $D_r \in \mathbb{R}^{r \times r}$. For values of r strictly less than n, this is an approximate decomposition and $A \approx A_r = U_r D_r V_r$. The closer the value of r is to n, the more accurate the approximation A_r is to A.

While this approximate decomposition might seem counterintuitive, it has a couple of significant benefits. First, for large values of n and small values of r, storing U_r, V_r, and D_r requires less space than the original A. Second, the approximate nature of A_r makes it possible to inform predictions about edges that were not originally in the graph. Multiple techniques leverage this approximation approach for the link prediction task. Two examples that are used in this work are truncated singular value decomposition (TSVD) and truncated eigenvector decomposition (TED).

Both TSVD and TED produce matrices whose rows or columns can be used to generate length r feature vectors that characterize each of the graph's vertices. Conventionally, these are not used directly; the decomposed matrices produced are multiplied to obtain the approximate adjacency matrix A_r. The relative values of A_r are used to inform predictions. A high value at $A_r[i][j]$ suggests the edge (v_i, v_j) is likely. However, this work also considers using the feature vectors more directly by providing them as input to a neural network classifier.

2.4 Neural Network Classification

Neural networks have been leveraged to classify inputs as normal or anomalous in a variety of applications [7, 14, 23]. In this work, neural networks are trained to take an edge as input and produce a score indicating how likely that edge is to occur in the network. To provide the neural network with useful information, the feature vectors produced by adjacency matrix decomposition for each of the edge's endpoints are provided as input. The neural networks utilized in this work are relatively simple fully connected feed-forward networks with a variable number of levels. The architecture of these networks is optimized by a hyper-heuristic search approach.

2.5 Hyper-Heuristics

The goal of a hyper-heuristics approach is not to produce the best solution to a specific problem instance, but instead to find a heuristic that produces high quality solutions to a specific problem class [1]. In this work, genetic programming (GP) is used to evolve programs that perform the link prediction task. By guiding the evolution with input from a subclass of the link prediction problem, the GP finds tailored heuristics that exploit characteristics of that problem class to improve predictive performance. This work uses the common Koza-style parse tree representation for evolved heuristics [11]. The primitive operation set available to the GP was extracted from existing approaches to the link prediction problem. To allow for a

variety of functionalities, a strongly-typed parse tree variant [15] is used that enforces compatibility between operations.

3 RELATED WORK

The link prediction problem has seen a lot of recent research activity. Both Wang et. al. [21] and Liben-Nowell et. al. [12] covered a variety of link prediction methods and applications in the field of social networking. Lü et. al. [13] summarized a number of approaches for other complex network types, such as those seen in biology and e-commerce. Many link prediction methods specifically leverage matrix decomposition. Dunlavy et. al. [3] discusses multiple decomposition-based techniques for temporal link prediction. Poisson matrix factorization [8] has seen a number of successful applications for link prediction, including in the field of cybersecurity [19]. The approach presented in this work leverages some of these methods as primitive operations available to the heuristic search. Unlike some of the application-specific link prediction methods discussed, the hyper-heuristic framework developed in this work is not limited to the applications presented here and can be easily applied to new problem domains.

The heuristic search utilized in this work is capable of combining multiple link prediction techniques to improve predictive performance. A number of ensemble learning methods take a similar approach. Gomes et. al. [6] describes a wide variety of ensemble learning techniques. The hyper-heuristic in this work is not limited to a preset method of combining multiple link prediction algorithms. The evolutionary search has the capability to optimize novel and often unintuitive ways to combine these algorithms to improve performance.

This work makes use of simple neural networks to classify links as likely or unlikely. Neural networks have a long history of being applied to such tasks [23]. More recently, a lot of activity has been seen in the field of neuroevolution which leverages evolutionary optimization to improve neural network performance [4]. More specifically, genetic programming has been applied to optimize neural network architectures [18]. This work uses a similar approach to optimize neural networks for the link prediction task, but is also capable of combining these networks with alternative link prediction techniques for further performance gains.

4 METHODOLOGY

Genetic Programming (GP) is used to evolve a population of link prediction heuristics that are targeted at a specific application.

4.1 Representation

Strongly typed parse trees [15] are used to represent evolved solutions. The initial pool of solutions is randomly generated from the available primitive operation set (described in Section 4.4) using a ramped half-and-half approach. An example parse tree representation of a basic link prediction heuristic can be seen in Figure 1.

4.2 Evaluation

During evaluation, an evolved solution is used to score a set of input edges for one or more test cases. For each test case, the scoring is compared to the true labels for the edges using a receiver operating characteristic (ROC) curve. A ROC curve compares the true positive rate (TPR) with the false positive rate (FPR) at different classification thresholds. The area under the ROC curve (AUC) is a value in the range $[0, 1]$ and is maximized when the scores for positive edge samples are consistently higher than the scores for negative edge samples. The fitness of an evolved solution is the average AUC values across each of the evaluation test cases. This fitness value is maximized when the link prediction heuristic clearly differentiates likely edges from unlikely ones.

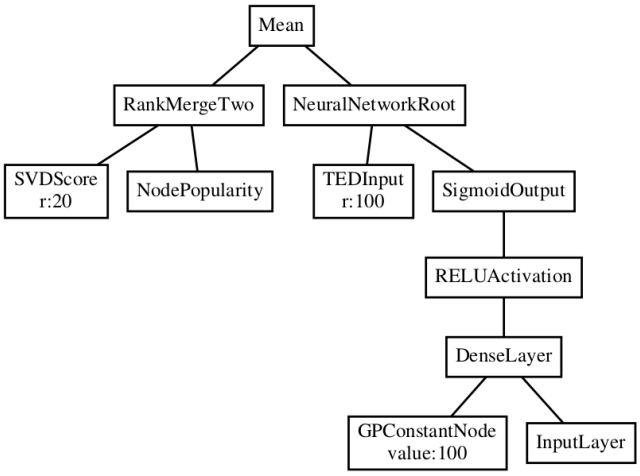


Figure 1: Example link prediction parse tree.

If a heuristic contains a neural network subtree, a configurable portion of the edge list for each test case is used to train the neural network. The remainder of the edge list is used as a validation set to evaluate the fitness of the entire solution. This is done to prevent the heuristic's fitness from being inflated by over-fitting on the training data. Neural networks are optimized using the Adam optimizer [10] with a binary cross-entropy loss function.

4.3 Evolution

Parents are chosen using tournament selection. According to a configurable probability, either subtree crossover between two parents or subtree mutation from a single parent is used to generate new offspring solutions. Only crossover or mutation is used for a single offspring, not both, due to the dramatic effect of subtree crossover on a solution's genotype. Offspring are added to the existing population, then truncation based on fitness is used for survival selection. If a configurable number of generations (convergence threshold) pass without seeing improvement in the population's best fitness, execution is terminated early.

4.4 Primitive Operations

As this work employs a strongly typed GP approach, each instance of an operation has an associated type to enforce compatibility. The available primitive types are as follows:

Integer: returns a whole number
Float: returns a floating point number
Weight: returns a floating point number bound to the range $[0, 1]$

ScoreArray: an array of floating point values calculated for each edge in the input list
Numeric: pseudo-type that refers to operations that can handle Integer, Float, Weight, or ScoreArray types (e.g., Add)
NNOutputLayer: neural network classification output layer
NNInputLayer: input layer
NNHiddenLayer: hidden layer
NNDropoutLayer: dropout layer
NNActivationLayer: activation function layer
NNRegularizer: kernel regularizer for neural network layer
NNInputVector: feature vector formatted for input to a neural network

See Table 1 for a description of the primitive operation set.

4.5 Parameters

See Table 2 for a list of the configurable parameter values used in this work. These values were manually selected to keep the GP execution time reasonable across multiple applications. More extensive tuning targeting a specific application would likely improve the quality of the final solutions. However, the results presented in this work demonstrate that even without application-specific tuning the GP is able to produce heuristics that outperform tuned versions of the conventional methods. Future work will investigate the impact of additional tuning of the GP parameters for applications where prediction accuracy is critical enough to warrant the increased cost of execution time.

5 EXPERIMENT

To demonstrate the potential of this approach to improve predictive performance for real-world applications, this work is applied to multiple cybersecurity prediction tasks. These applications utilize data collected from the enterprise computer network at Los Alamos National Laboratory (LANL) [20]. The data set contains traffic information in the form of NetFlow entries as well as networked host logs that track authentication and process execution events. Evolved heuristics are evaluated by how well they differentiate legitimate network activity from randomly generated anomalous events.

5.1 Predicting Process Execution

Process execution events are collected from the LANL data to create two types of graphs. The first contains vertices for user accounts and process names, along with edges that indicate a process was executed by (or on the behalf of) a user. The second replaces the users with computers connected to processes that have been executed on those computers. Edge weights indicate the number of times a user-process or computer-process pair was seen in the data. It is worth noting that these process execution graphs are bipartite; the vertex set can be divided into the set of users (or computers) and the set of processes and edges can only connect a vertex in one set to a vertex in the opposite set. Only links that connect a user (or computer) and a process are considered for prediction.

5.2 Predicting Network Traffic

A graph is constructed to represent the communication between devices on the LANL network. An edge indicates that the two devices it connects communicated at least once. Edges are weighted by the number of distinct communication sessions occurring between devices in the data set. Unlike the process execution application, these network traffic graphs are not bipartite. Any pairing of two networked devices is considered a valid link during prediction.

5.3 Training and Evaluation

The first four weeks (28 days) of data are used to generate the initial historical graph for each application. See Table 3 for a summary of the data set used. Adjacency matrices are created for each of these graphs using the transformation $A[i][j] = ln(1+weight(v_i, v_j))$; this log transformation corrects for the bursty nature of this data set to improve predictive performance. TSVD and TED feature vectors are pre-computed for these adjacency matrices at various truncation levels. These feature vectors are cached and made available to the GP operations to prevent redundant computations and reduce evaluation time whenever possible.

Each of the following 14 days is used to create an evaluation link prediction test case. Links seen on these days are compared to the historical graphs. To be included in the test case, both endpoints must be present in the historical graph, but the link itself must not be present. This requires that the evolved heuristics be able to predict new (previously unseen) links, but does not expect the solutions to be able to predict links to vertices they have no historical information about.

The collections of new links are labeled as positive samples for each test case. To provide negative samples, an equal number of links missing from both the historical graph and the test case are randomly selected. The positive and negative samples are concatenated for each test case and the order of the samples is randomized.

For each application, a population of heuristics is evolved with the goal of differentiating the positive and negative samples from each test case. Evolved candidate solutions are executed for each test case to produce a score for each sample. These scores are compared to the true labels to produce an AUC score for each test case and the solution's fitness is its average AUC.

To establish a baseline for comparison of solution quality, each of the basic link prediction methods (TSVD, TED, node popularity) are programmatically tuned for each application and are evaluated in a similar manner. A simple densely connected feed-forward neural network, shown in Figure 2, is also trained for each application and used for comparison. To examine the benefit of heuristic specialization, the best evolved solution from each application is also applied to each of the other applications and compared to the heuristic specifically evolved for that application.

6 RESULTS AND DISCUSSION

See Figure 3 for a visualization of evolved solution fitness versus execution time of the hyper-heuristic search for example runs from each application. Because solutions leverage existing link prediction techniques as primitive operations, the search finds mediocre solutions almost immediately. However, the steady growth in fitness values shows the search optimizing performance by combining these primitive operations in ways that exploit characteristics of the application.

Table 1: Primitive Operations

Primitive	Type	Inputs	Description
Add	Numeric	x, y:Numeric	$x + y$
Subtract	Numeric	x, y:Numeric	$x - y$
Multiply	Numeric	x, y:Numeric	$x * y$
SafeDivide	Numeric	x, y:Numeric	0 if $y = 0$, else x/y
Mean	ScoreArray	x, y[, z]:ScoreArray	array of mean values at each index
Absolute	ScoreArray	x:ScoreArray	absolute value of each element
Rescale	ScoreArray	x:ScoreArray	scales values in x to the range $[0, 1]$
MeanNormalize	ScoreArray	x:ScoreArray	scales values in x to the range $[-0.5, 0.5]$
RankConvert	ScoreArray	x:ScoreArray	converts scores to a ranking
ScaledMean	ScoreArray	w_x, w_y[, w_z]:Weight, x, y[, z]:ScoreArray	finds weighted mean
RankMerge	ScoreArray	x, y[, z]:ScoreArray	converts scores to ranks then finds mean rank
ScaledRankMerge	ScoreArray	w_x, w_y[, w_z]:Weight, x, y[, z]:ScoreArray	converts to ranks then finds weighted mean
SVDScore	ScoreArray	*None*	scores input edge list using TSVD
TEDScore	ScoreArray	*None*	scores input edge list using TED
NodePopularity	ScoreArray	*None*	scores input edge list using node popularity
NeuralNetworkRoot	ScoreArray	x:NNInputVector, y:NNOutputLayer	scores edge list using a neural network
SVDInput	NNInputVector	*None*	TSVD feature vectors
TEDInput	NNInputVector	*None*	TED feature vectors
NodePopularityInput	NNInputVector	*None*	node popularity scores
ConcatenateInputs	NNInputVector	x, y:NNInputVector	concatenates two NNInputVectors
SigmoidOutput	NNOutputLayer	[x:NNRegularizer,] y:NNActivationLayer	size one sigmoidal output layer for classification with optional regularizer
SigmoidActivation	NNActivationLayer	x:NNHiddenLayer/NNDropoutLayer	sigmoidal activation layer
RELUActivation	NNActivationLayer	x:NNHiddenLayer/NNDropoutLayer	rectified linear unit activation layer
DropoutLayer	NNDropoutLayer	x:Float, y:NNHiddenLayer	dropout layer with drop rate x
DenseLayer	NNHiddenLayer	x:Integer, [y:NNRegularizer,] z:NNActivationLayer/NNInputLayer	typical densely connected layer of size x with optional regularizer
L1Regularizer	NNRegularizer	x:Float	L1 kernel regularizer with weight x
L2Regularizer	NNRegularizer	x:Float	L2 kernel regularizer with weight x
InputLayer	NNInputLayer	*None*	input layer of a neural network
GPConstantNode	Numeric	*None*	randomly initialized number

Table 2: Parameters

Parameter	Value
Population size	30
Offspring size	20
Generation limit	100
Convergence threshold	10
Crossover chance	0.7
Mutation chance	0.3
Parent selection tournament size	5
Initial minimum tree depth	1
Initial maximum tree depth	5
Mutation minimum tree depth	1
Mutation maximum tree depth	3
Neural network validation split	0.6
Neural network training epochs	20

Table 3: Data Set Summary

User-Process	
Unique users	25,761
Unique processes	27,944
Historical user-process links	2,106,120
Total user-process test links	216,352
Computer-Process	
Unique computers	13,465
Unique processes	27,944
Historical computer-process links	1,976,705
Total computer-process test links	190,857
Network Traffic	
Unique devices	60,185
Historical communication links	1,136,854
Total communication test links	250,815

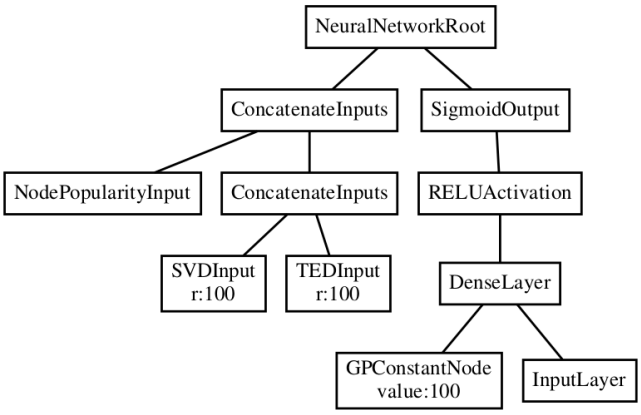


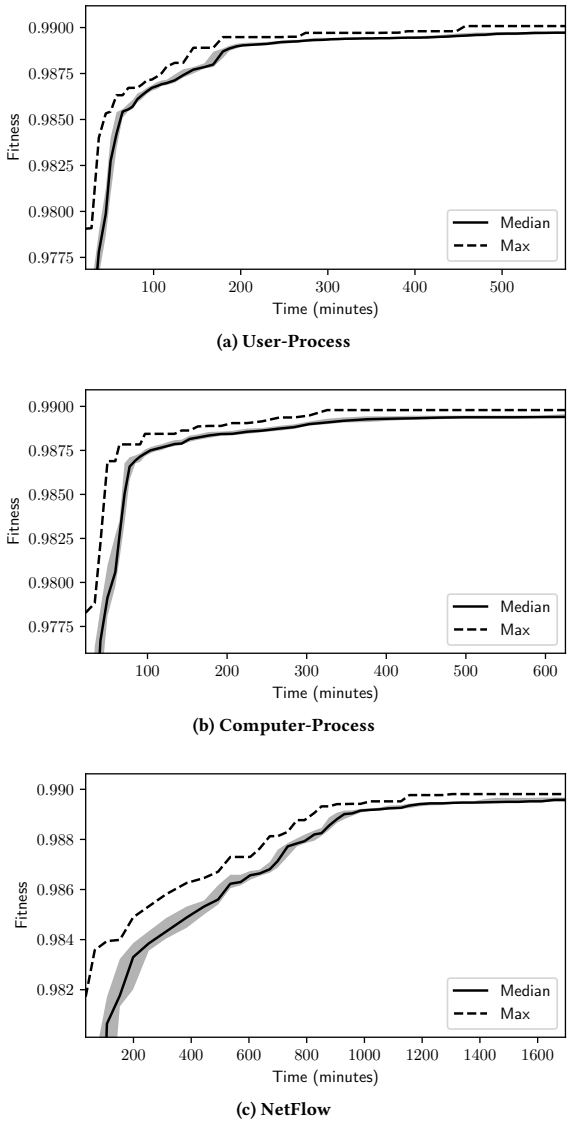
Figure 2: Parse tree representation of the basic neural network used for baseline comparison.

Although the search converges at around the same fitness level of 0.99 for each application, the shape of these curves reveal some significant differences in the difficultly of the problems. The search targeting the Computer-Process application shows the steepest initial increase in population fitness. This might suggest that the Computer-Process prediction task is easier than the other two applications. It is reasonable to think that information about the computer on which a process is being executed is more useful than which user is running the process. For instance, many processes are platform dependent and will never be seen running on a platform they weren't designed for (e.g., a service related to a Windows desktop application running on an email server). Alternatively, a network user might work on multiple computers using different platforms.

The difference in problem difficulty is even more apparent in the fitness values for the NetFlow application. Not only is the increase in fitness more gradual, but the search takes significantly longer to converge. There are a couple of differences in the NetFlow application that could explain the increased difficulty. The NetFlow application has fewer historical links to learn from, more test links to predict, and more unique vertices than both of the other applications. Additionally, the NetFlow graphs are not bipartite, meaning that the number of possible links is much higher ($60,185^2 \approx 362$ million) than for both the User-Process ($\approx$ 72m) and Computer-Process ($\approx$ 37m) applications.

To compare the various link prediction techniques, the validation links from each of the test cases are combined to create a single evaluation case. Each method is used to score this evaluation case and the AUC metrics for each method can be seen in Table 4. In each application, the heuristic targeting that application resulted in the highest AUC. The improvement in predictive performance demonstrates the potential for specialization of link prediction heuristics for applications where maximizing accuracy is critical. This is often the case when using link prediction for detecting anomalous activity where investigating false positives is prohibitively expensive.

It is worth noting that the evolved heuristics also outperform the baseline approaches even when applied to applications they were not trained to target. This is likely the result of the evolved solutions simply leveraging more information than any of the individual basic approaches. The receiver operating characteristic curves produced when comparing the various link prediction methods are shown in Figure 4.



(a) User-Process

(b) Computer-Process

(c) NetFlow

Figure 3: Population fitness values versus execution time from example evolutionary runs for each application. The shaded region indicates the interquartile range.

Although space constraints do not allow for the inclusion of each solution's parse tree representation, an example evolved heuristic for the NetFlow application can be seen in Figure 5. This tree has been programmatically simplified after evolution via constant folding and redundant subtree pruning to make interpretation easier without affecting predictive performance. The example tree exhibits an unintuitive characteristic that is fairly common in evolved

Table 4: Link Prediction Accuracy

Method	Application		
	UP	CP	NF
NP	0.76963	0.74226	0.52967
TSVD	0.94186	0.90334	0.92936
TED	0.97478	0.97697	0.92390
NN	0.98725	0.98661	**0.98836**
GP-UP	**0.99066**	**0.98718**	0.98051
GP-CP	**0.98897**	**0.98996**	**0.99090**
GP-NF	**0.98867**	**0.98874**	**0.99241**

AUCs produced by the method indicated on the left for the User-Process (UP), Computer-Process (CP), and NetFlow (NF) applications. Methods include node popularity (NP), truncated singular value decomposition (TSVD), truncated eigenvector decomposition (TED), neural networks (NN), and evolved heuristics (GP-*). The shaded value in each row indicates the best predictive performance for each application. Bold values are not significantly worse (alpha = 0.05) than the best.

solutions. Namely, not only does it combine multiple basic link prediction techniques, but it also leverages different configurations of the same algorithm (in this example, TED decomposition at multiple ranks).

Manual inspection of typical evolved solutions reveals some other interesting behavior. Final solutions that do not leverage a neural network in some way are extremely rare and only seen in runs that prematurely converged early at lower population fitness values. However, solutions that rely solely on neural network classifiers are also rare. This provides some evidence that the combination of multiple link prediction techniques is critical and has the capability to produce solutions that are more than a sum of their parts.

7 FUTURE WORK

The heuristic search parameters used in this work were chosen to provide a proof-of-concept while limiting evolution time. In future work, automated parameter tuning will be used to further increase performance. The approach might also benefit from conversion to a multi-objective search with training or execution time as a minimization objective. Evolved solutions tend to incorporate redundant subtrees that likely have little or no effect on predictive performance. Minimizing execution time would encourage more efficient solutions.

During evaluation, a number of negative samples is generated to match the number of positive samples. This is typical for methods using the area under the ROC curve for comparison, but not necessarily representative of real-world data where anomalous events might be far more rare than legitimate activity. The performance impact of unbalanced sample sizes is currently unknown and will be investigated in future work.

The results presented in this work demonstrate promising results despite having a relatively small set of simple link prediction primitive operations. There are additional link prediction methods whose functionality could be incorporated to further improve performance. One example is Poisson Matrix Factorization [8], a state-of-the-art link prediction technique.

The evolution of neural network subtrees leveraged in this work is also fairly rudimentary. The set of available neural network primitives could be expanded to allow discovery of more flexible classifiers. More sophisticated neuroevolution techniques, such as those used in NEAT [17], might further improve the quality of the neural network components produced in this work.

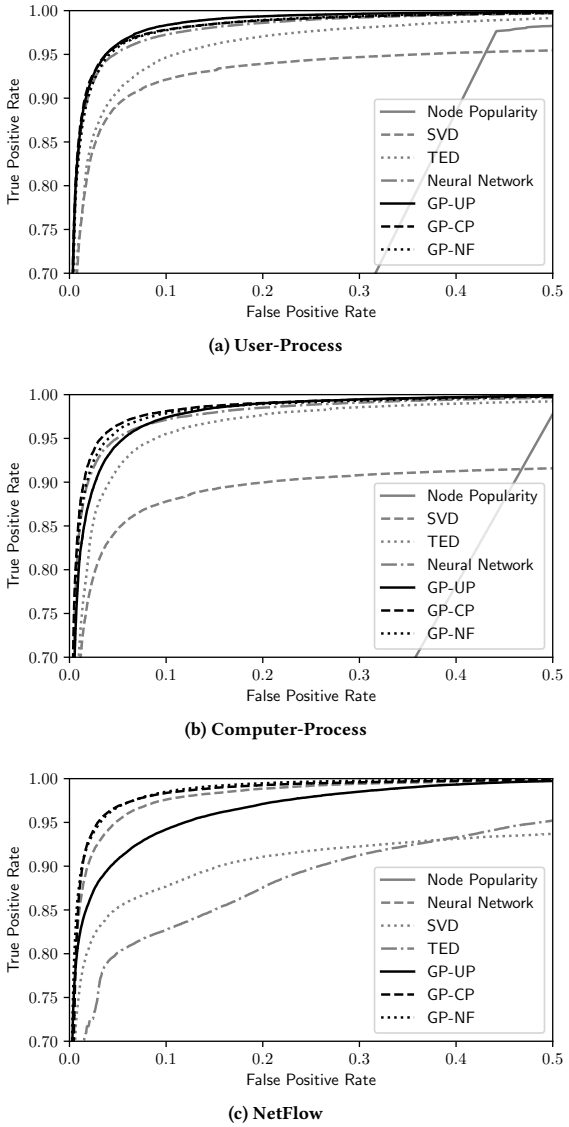


Figure 4: Comparison of receiver operating characteristic (ROC) curves for each application. The area of these plots is focused on the upper left corner to make it easier to see the differences between the curves. Curves closer to the upper left corner indicate better predictive performance. Evolved heuristics are indicated by black lines and basic methods are shown in gray.

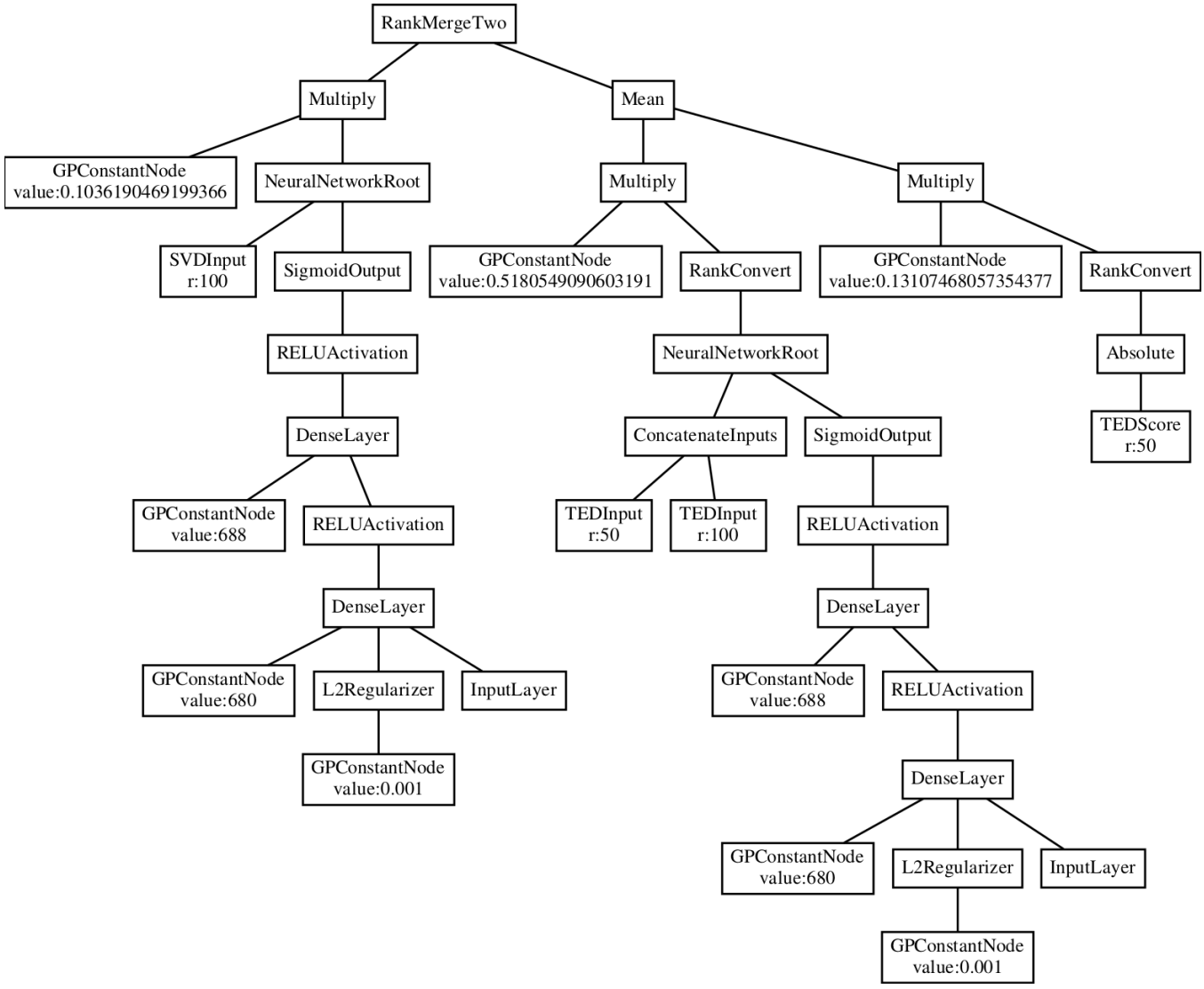


Figure 5: Example final heuristic evolved targeting the NetFlow application.

8 CONCLUSION

Link prediction is a commonly occurring problem with applications in recommendation systems, social networking, anomaly detection, and others. Research in the development of new link prediction heuristics has produced a variety of techniques with various accuracy and efficiency trade-offs. Performance in link prediction applications requires leveraging the appropriate method. While the selection of the best link prediction heuristic can be automated, this relies on the quality and variety of available heuristics.

This work demonstrates the potential of using a generative hyper-heuristic search to automate the development of novel link prediction heuristics that are customized to specific applications. Results targeting three cybersecurity prediction tasks using real-world enterprise network data show that the evolutionary process is capable of improving predictive performance over baseline techniques by exploiting characteristics of the problem subclasses. This improved performance comes at the cost of additional a priori computation time, but the resulting solutions provide higher accuracy in applications where investigating false positives can be costly and time consuming.

ACKNOWLEDGMENTS

This work was supported by Los Alamos National Laboratory via the Cyber Security Sciences Institute under subcontract 259565 and the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project numbers 20180607ECR and 20170683ER.

REFERENCES

[1] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* 64, 12 (2013), 1695–1724.

[2] FranÃğois Caron and Emily B. Fox. 2017. Sparse graphs using exchangeable random measures. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 79, 5 (2017), 1295–1366. https://doi.org/10.1111/rssb.12233 arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12233

[3] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. 2011. Temporal Link Prediction Using Matrix and Tensor Factorizations. *ACM Trans. Knowl. Discov. Data* 5, 2, Article 10 (Feb. 2011), 27 pages. https://doi.org/10.1145/1921632.1921636

[4] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 1 (01 Mar 2008), 47–62. https://doi.org/10.1007/s12065-007-0002-4

[5] M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99, 12 (2002), 7821–7826. https://doi.org/10.1073/pnas.122653799 arXiv:https://www.pnas.org/content/99/12/7821.full.pdf

[6] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A Survey on Ensemble Learning for Data Stream Classification. *ACM Comput. Surv.* 50, 2, Article 23 (March 2017), 36 pages. https://doi.org/10.1145/3054925

[7] F. Gonzalez, D. Dasgupta, and R. Kozma. 2002. Combining Negative Selection and Classification Techniques for Anomaly Detection. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 1. 705–710 vol.1. https://doi.org/10.1109/CEC.2002.1007012

[8] Prem K Gopalan, Laurent Charlin, and David Blei. 2014. Content-based recommendations with Poisson factorization. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3176–3184. http://papers.nips.cc/paper/5360-content-based-recommendations-with-poisson-factorization.pdf

[9] Tsuyoshi IDÉ and Hisashi KASHIMA. 2004. Eigenspace-based Anomaly Detection in Computer Systems. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*. ACM, New York, NY, USA, 440–449. https://doi.org/10.1145/1014052.1014102

[10] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

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

[12] David Liben-Nowell and Jon Kleinberg. 2007. The Link-Prediction Problem for Social Networks. *Journal of the American Society for Information Science and Technology* 58, 7 (2007), 1019–1031. https://doi.org/10.1002/asi.20591 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.20591

[13] L. Lü and T. Zhou. 2011. Link Prediction in Complex Networks: A Survey. *Physica A: Statistical Mechanics and its Applications* 390, 6 (2011), 1150 – 1170.

[14] M. Meneganti, F. S. Saviello, and R. Tagliaferri. 1998. Fuzzy Neural Networks for Classification and Detection of Anomalies. *IEEE Transactions on Neural Networks* 9, 5 (Sep. 1998), 848–861. https://doi.org/10.1109/72.712157

[15] David J. Montana. 1995. Strongly Typed Genetic Programming. *Evolutionary Computation* 3, 2 (June 1995), 199–230. https://doi.org/10.1162/evco.1995.3.2.199

[16] Guy Shani and Asela Gunawardana. 2011. *Evaluating Recommendation Systems*. Springer US, Boston, MA, 257–297. https://doi.org/10.1007/978-0-387-85820-3_8

[17] K. O. Stanley and R. Miikkulainen. 2002. Efficient Evolution of Neural Network Topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 2. 1757–1762 vol.2. https://doi.org/10.1109/CEC.2002.1004508

[18] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, New York, NY, USA, 497–504. https://doi.org/10.1145/3071178.3071229

[19] M. Turcotte, J. Moore, N. Heard, and A. McPhall. 2016. Poisson Factorization for Peer-Based Anomaly Detection. In *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*. 208–210. https://doi.org/10.1109/ISI.2016.7745472

[20] Melissa J. M. Turcotte, Alexander D. Kent, and Curtis Hash. 2018. *Unified Host and Network Data Set*. World Scientific, Chapter Chapter 1, 1–22. https://doi.org/10.1142/9781786345646_001 arXiv:https://www.worldscientific.com/doi/pdf/10.1142/9781786345646_001

[21] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. 2015. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences* 58, 1 (01 Jan 2015), 1–38. https://doi.org/10.1007/s11432-014-5237-y

[22] Z. Wang, J. Liao, Q. Cao, H. Qi, and Z. Wang. 2015. Friendbook: A Semantic-Based Friend Recommendation System for Social Networks. *IEEE Transactions on Mobile Computing* 14, 3 (March 2015), 538–551. https://doi.org/10.1109/TMC.2014.2322373

[23] G. P. Zhang. 2000. Neural Networks for Classification: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 30, 4 (Nov 2000), 451–462. https://doi.org/10.1109/5326.897072