Evolving an Edge Selection Formula for Ant Colony Optimization

Andrew Runka
Dept. of Computer Science
Brock University
St. Catharines, Ontario, Canada
ar03gg@brocku.ca

ABSTRACT

This project utilizes the evolutionary process found in Genetic Programming to evolve an improved decision formula for the Ant System algorithm. Two such improved formulae are discovered, one which uses the typical roulette wheel selection found in all well-known Ant Colony Optimization algorithms, and one which uses a greedy-style selection mechanism. The evolution of each formula is trained using the Ant System algorithm to solve a small Travelling Salesman Problem (TSP) and tested using larger unseen TSP instances.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program modification, program synthesis*

General Terms

Algorithms Experimentation

Keywords

Genetic Programming, Ant Colony Optimization, Edge Selection

1. INTRODUCTION

The aim of this project is to utilize the natural evolutionary mechanism of Genetic Programming (GP) to create an improved formula for Ant Colony Optimization (ACO) algorithms to use in their construction of solutions. ACO is a population based meta-heuristic, first developed by Marco Dorigo in 1992[1]. It has been successfully applied to many combinatorial optimization problems[2] including the Travelling Salesman Problem[3] examined here. By using a combination of *a priori* and *a posteriori* knowledge, each ant in the population constructs a solution step by step with the use of a decision formula. The traditional decision formula used in ACO was designed on preconceived notions which may or may not be ideal. The use of GP as an invention/innovation machine offers the potential to discover novel and perhaps unexpected improvements or replacements to the preexisting formula. It is hoped that a new formula created through GP evolution will lead to an improvement in the overall performance of ant algorithms.

From the symbolic regression toy problem to complex image recognition, the concept of evolving formulae is central to the GP paradigm. However, the domain of this project appears to be somewhat unique as compared to the existing GP literature. Similar work has been done on the evolution of parameter sets[4][5], neural networks[6][7], and crossover operators[8], but none have specifically considered evolving or improving the essential formulae of an ACO algorithm.

The remainder of this paper is structured as follows, Section 2 provides the background on the problems and algorithms examined. Section 3 describes the procedures used in this study and section 4 describes and discusses the results of this study.

[1]Later published as [1]

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

2. BACKGROUND

2.1 Travelling Salesman Problem

The Ant Colony Optimization formula evolved will be tested using the Travelling Salesman Problem (TSP). The TSP is a combinatorial optimization problem which typical ACO algorithms perform well on. The simple problem formulation used here is as follows. Given a graph G=(V,E), where V is the set of n vertices and E is the set of fully-connected bi-directional weighted edges between $v_i, v_j \in$V, find a hamiltonian cycle which minimizes $\sum_{i=0}^{n} w_{i,j}$, where $w_{i,j}$ is the weight on the edge between v_i and v_j.

2.2 Ant Colony Optimization

Ant Colony Optimization (ACO) is a meta-heuristic modelled on the natural optimization behaviour of real ants. In reality, a population of ants cooperate by use of pheromone trails to find optimal paths between a nest and a food source. The concept of pheromone was borrowed for ACO to act as a means of balancing between exploration and exploitation in a combinatorial optimization search space. The ACO meta-heuristic can be broken down into three main phases: generate solutions, update pheromone, and daemon actions.

Several instantiations of the ACO meta-heuristic exist, this project is concerned with only the simplest such instantiation, known as Ant System (AS). Other instantiations, such as Max-Min Ant System[9] and Ant Colony System[3] share many similarities with AS. The following explanation

is given in terms of AS, however much of what is described is true of many ACO algorithms.

The first phase of the ACO meta-heuristic encompasses the construction of solutions to the given problem. Solutions are in the form of paths through the problem graph. At each step during construction each ant adds one vertex to its path. In typical ACO algorithms, the ant will move from vertex i to vertex j with a probability calculated as follows:

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\Sigma(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \quad (1)$$

where $\tau_{i,j}$ is the amount of pheromone on edge$_{i,j}$ and $\eta_{i,j}$ is the desirability of edge$_{i,j}$ which is the *a priori* information known about the problem (e.g. $\frac{1}{dist_{i,j}}$ for the TSP). The exponents α and β are parameters which affect the amount of influence of τ and η on the final probability respectively. The methods used to evolve a replacement for this formula are described in section 3.

As formula (1) gives a *probability* of selecting a given edge, a method of utilizing this information is required to actually determine the next step in an ant's solution. Typically this takes the form of roulette selection. In roulette selection, all options are given a probability of being selected, then a random number in the range (0,1) is drawn. The probabilities of selecting each option are added to a sum one after another. If adding a given probability to the sum increases it beyond the random number then the corresponding option is selected. In ACO this means that the selected edge is added to the ant's solution. Additional methods for utilizing the probability information are examined in section 4.

It should be noted that the original decision formula remains the same in all well-known instantiations of the ACO meta-heuristic. Extensions beyond the simple AS algorithm are primarily concerned with variations to uses of the pheromone matrix (described below). The Ant Colony System introduced a so called "pseudo-random proportional rule", but this altered the way the probability information from (1) was used, not how it was generated.

Once each ant in the population has constructed a solution, the second phase, known as the pheromone update, takes place. The pheromone update can be separated into two steps: evaporation and deposit. In the evaporation step, the amount of pheromone removed is calculated as follows:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} \quad (2)$$

where $\tau_{i,j}$ is the amount of pheromone on edge$_{i,j}$, and ρ is a parameter that controls the rate of evaporation. This formula is applied globally to all edges in the graph. The second step, deposit, is calculated for each ant over the path it took through the graph, and typically takes the form:

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{1}{C_k}, & if\ ant\ travels\ edge_{i,j} \\ 0, & otherwise \end{cases} \quad (3)$$

where C_k is the cost of the kth ant's solution.

By iteratively applying formulae 1, 2, and 3, the amount of pheromone will accumulate on edges which appear to be parts of good solutions to the problem thus attracting future ants towards those edges.

The third phase of the ACO metaheuristic, daemon actions, is an optional phase. This is used to perform any post-processing steps such as local search to the solutions generated in the previous phase.

2.3 Genetic Programming

Genetic Programming (GP) was first popularized in 1992[10] and has since been successful in achieving human-competitive results in many fields including electronic design, game playing, searching, sorting and more[11][12]. "GP is a systematic, domain-independent method for getting computers to solve problems automatically"[12]. In GP, solutions are traditionally represented as program trees (or s-expressions), where each internal node in the tree is an operator and its subtrees are its operands. The leaf nodes of the tree are terminals in the expression. Together the set of all terminals and operators in the GP language are known as the primitive set of the given GP system. Figure 1 illustrates the chromosome representation of a simple formula.

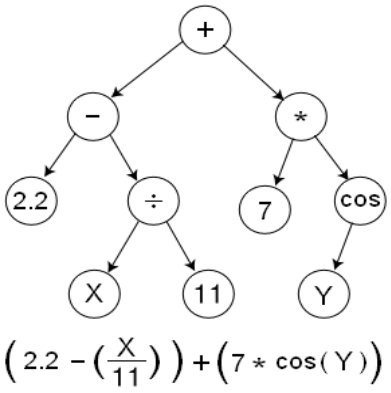

Figure 1: GP chromosome representation (genotype and phenotype)

The task of genetic programming is to evolve a population of programs to solve a given problem. This is done using the concepts of natural evolution found in the Evolutionary Algorithm (EA) framework. In EAs and in GP an initially random population of solutions to a problem is evolved by iteratively applying crossover, mutation, and selection operators.

The original GP crossover operator (also the one used in this project) is known as the subtree crossover. As depicted in figure 2, offspring are created by exchanging the subtree of one parent for the subtree of another. The original GP mutation, known as subtree mutation, works in the same fashion, one parent is selected for mutation, and subtree crossover is performed with a randomly generated second parent. A third operator, known as reproduction, is sometimes used in place of crossover or mutation. In reproduction an individual is simply copied over from one generation to the next.

With a new population of programs instantiated by the crossover, mutation, and reproduction operators, the individual programs are evaluated on their performance at solving some problem based on a problem-dependant fitness function. The resulting fitness scores are the basis for the subsequent selection operation.

A common selection operator is tournament selection. In tournament selection, a number of individuals are picked at random from the population. From these individuals the best is then selected. Two such selected individuals are used for crossover, and one such selected individual is used for

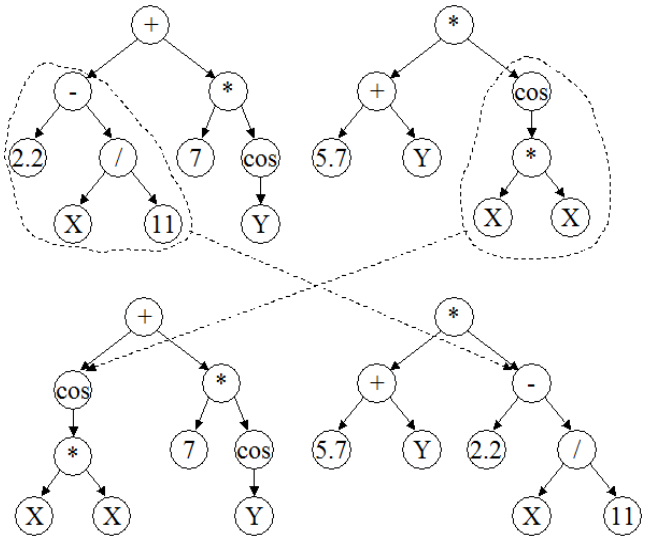

Figure 2: Example subtree crossover. Subtrees of the parents (top) are swapped to create offspring (bottom)

mutation or reproduction. The iterative application of this process is intended to refine the quality of solutions in the population from a random collection to a converged population of near-optimal solutions.

Variations to each of these operators exist in the literature, but are not used in the project. For information and references on extensions to the simple GP system refer to [12].

3. EXPERIMENTAL SETUP

3.1 Problem Formulation

The objective of this project can be described as applying one problem-solving method to the improvement of another problem-solving method. That is, GP is applied to the improvement of ACO. To accomplish this, each individual in the GP population represents a potential replacement for the ACO decision formula (formula (1) in section 2.2). Then, during the evaluation of each individual, an entire run of the AS algorithm on a simple TSP instance is performed using the individual's version of the decision formula. The performance of the AS algorithm with the given decision formula is used as the individual's fitness score. Evolution of the GP system proceeds from there as usual with selection, crossover, and mutation.

The TSP was selected as the problem for the AS algorithm not only because it is a very quick problem to solve, but also because many more complicated problems solved by the ACO algorithm are first translated into a TSP problem (e.g. Vehicle Routing Problem).

Due in part to the removal of the array operations from the primitive set (see section 3.4), the information contained in each GP individual is not the complete decision formula equivalent to formula (1). Instead each GP individual contains the equivalent to the numerator of formula (1). The values calculated by the GP individual's expression tree are summed over all J and each is subsequently converted to a value between 0 and 1. This effectively applies a denominator equivalent to that of (1). Also, the original decision formula uses $\eta_{i,j} = 1/dist_{i,j}$ whereas in the GP execution here $\eta_{i,j} = dist_{i,j}$ is used. Thus, given these two facts a GP individual whose formula equals

$$(\tau_{i,j}^2) \div (\eta_{i,j}^3) \tag{4}$$

would be equivalent to formula (1) with $\alpha = 2$ and $\beta = 3$. One final concession was made to ease the GP system's ability to find solutions, the absolute value of the result returned by each GP individual's formula is used.

3.2 Fitness Function

The fitness function used in the GP system, as mentioned before, is based on the results of a full Ant System execution on a simple TSP instance. The TSP instance used throughout training was the berlin52.tsp[2]. This instance (as well as all other symmetric TSP instances found in the TSPLIB) have been provably solved to optimality by a branch-and-cut method, with the optimal distance scores reported. Using this information, the fitness function for the GP run can be set such that an optimal result from the AS run will result in a 0 fitness score. Thus the raw fitness score of a GP individual is calculated as follows:

$$RawFitness = ASResult - Optimal \tag{5}$$

Where ASResult is the total distance of the best solution found during the AS execution. A hit occurs when ASResult is equal to Optimal.

Due to time and computing resource constraints each individual in the population is evaluated on only a single AS execution. At first glance this may seem to be too limited of a sample from which to draw a fitness value. However, there are three concepts to bear in mind here. First, during a single AS execution, the evolved formula is applied several thousand times as needed during the repeated construction of solutions to the TSP. Thus the formula contained in each individual is utilized many times in conjunction to produce a single fitness score. Secondly, it has been shown that noise in the fitness function leads to a lowered selection pressure[13]. This is accounted for by a (slight) increase in tournament size (see next section). Third, in order to counteract the possibility of a good individual resulting from a fluke, the best results from the GP system are tested on two unseen TSP instances (rat195.tsp and bier127.tsp), as well as more extensive testing on the seen TSP instance (berlin52.tsp). The final quality of an evolved formula is thus judged based on the testing phase as opposed to the training phase.

3.3 Parameters

A number of parameters are involved in this total system, including those for the GP system and those for the Ant System. These are listed in tables 1 and 2 below.

3.4 GP Language

The primitive set used in the final executions of this project is essentially the arithmetic operators, with ephemeral random constants and problem specific data as the terminals. The original conception of the problem used a somewhat broader primitive set, however this was reduced to save on

[2]Retrievable from TSPLIB: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Table 1: GP parameters settings

Parameter	Value
Max Tree Depth	15
Generations	50
Population Size	200
Elitism	1 individual
Initialization	Ramped Half-and-half [2,6]
Selection	Tournament (size 4)
Crossover Type	Subtree Crossover
Crossover Percent	80%
Mutation Type	Subtree Mutation
Mutation Percent	20%
Reproduction	0%
Terminal Node Selection	10%

Table 2: AS parameters settings

Parameter	Value
Duration	100
Evaporation Rate (ρ)	0.1
Number of ants	n (problem size)
α (for default formula)	2
β (for default formula)	1
Tournament Size (when needed)	7

execution time. The primitive set used in the experimental results and discussion section is shown in table 3.

Table 3: GP Primitive Set

Primitive	Returns	Children	Description
+	Double	2 double	Addition
-	Double	2 double	Subtraction
*	Double	2 double	Multiplication
%	Double	2 double	Division
^	Double	2 double	Exponentiation
ElementOf	Double	1 array, 2 int	Dereference 2D array
Tau	Array	n/a	2D pheromone array
Eta	Array	n/a	2D distance array
I	Int	n/a	Position of ant
J	Int	n/a	Destination of ant
smallERC	Int	n/a	Ephemeral random constant [-10,10)

All of the operators applied in this project use a sort of safety net to detect illegal values. If a given primitive results in an infinite value (occurs in java beyond the "MAX_VALUE" of a given type), or a NaN result (occurs in java upon an invalid calculation such as $(-1.4)^{3.2}$), then that primitive returns a 0.

Strong typing was applied to limit the usage of certain parameters to only those places where their use makes sense. The use of strong typing is not a necessity in the final primitive set. The 'ElementOf' operator and the 'I' and 'J' terminals could all be removed having Tau and Eta represent Tau[I][J] and Eta[I][J] instead. In this way the entire primitive set would use a single type. The reason this was not done was partly historic and partly with consideration for future utilization. As previously mentioned, the original primitive set had a greater number of primitives. These removed primitives are listed in table 4. It can be seen here that the use of arrays made strong typing a necessity originally. The future consideration is that given a greater time-span and/or more computing resources, the study of these and other additional primitives could offer further insight to this problem and potentially better results.

Table 4: Removed primitives

Primitive	Returns	Children	Description
SumArray	Double	1 Array	Sums the elements of the given 1D array
AddArrays	Array	2 Arrays	Adds the elements of each array
MulArrays	Array	2 Arrays	Multiplies the elements of each array
ExpArray	Array	1 Array, 1 Double	Raise the element of each array to the double
SizeOf	Int	1 Array	Returns the size of the array (first dimension if 2D)
IndexERC	Int	n/a	Ephemeral random constant [0,n)

3.5 Variations of Experiments

Initial test executions of this problem revealed that usage of the array operations in the original primitive set were far too costly to complete the execution of an entire ant system for each individual. Even upon removing these costly primitives, the executions of each GP system remains a time-consuming affair, requiring up to 50 hours of execution time on a 2.4GHz Intel Core2 Quad CPU with 2GB of RAM. As such, each GP system was run only once for the final results (after debugging and tuning). A single execution is practical in this case, as it is not an averaged result that is required, but a single specific output.

Three versions of the final GP system were executed, the difference between each was within the Ant System itself. Section 2.2 mentions ACO performs a roulette selection to utilize the probability information gained from the decision formula. This project experiments with variations to this rule. The three types of selection methods for decision information usage are:

a) **Roulette Selection** - as described in 2.2,

b) **Greedy Selection** - the largest probability is always selected,

c) **Tournament Selection** - 7 probabilities are chosen at random and the best is selected.

The motivation for using alternative selection mechanisms was that initially it was feared that the use of roulette selection within the AS (which acts as the fitness function for the GP) would be too random of a factor for the GP to evolve properly. Thus the greedy rule was first implemented as a potential replacement because it has the property of zero-variance. That is, given the same inputs, the greedy rule will always produce the same results. Following this the notion of using a tournament selection seemed like a natural third choice as it is a popular selection mechanism. A tournament size of 7 was chosen in order to maintain high selection pressure, thus making the tournament selection similar to a

relaxed-greedy selection. Since the use of alternative selection schemes within an ACO algorithm is a unique idea in and of itself, the default AS decision formula will be tested on these alternative selection schemes as well.

Following the execution of each of these experiments, the best formula discovered is outputted in text format. This text-formatted formula is then manually converted into a java-executable formula and entered into a stand-alone version of the AS code. This is done for rapid testing of the formula over 100 executions on each of the testing data sets. The results of this testing are compared to those of the original decision rule using the same stand-alone AS code. One potential issue with this method of testing is rounding errors. The text-formatted formula uses a 4-decimal place precision, which is significantly less than the full capability of the *Double* type and thus some information and possibly quality, is lost. However, this is a necessary loss as otherwise testing of individuals would take nearly as long as training, making this system infeasible in the given timeline.

4. RESULTS AND DISCUSSION

The training results for the roulette, greedy, and tournament are presented in table 5. Recall that the fitness is the difference between the actual result and the optimal result (in this case 7542). For example a fitness of 100 would mean a tour on the berlin52.tsp instance of size 7642. The evolved formulas for each are simplified and presented as formulas (6), (7), and (8) respectively.

Table 5: Training Results

Selection	Best Training Fitness
Roulette	2.365902
Greedy	6.99271
Tournament	2329.0845

Roulette Selection:

$$\frac{\left(\frac{(((\eta_{j,i}^{-11.5299})\div 14.2606)\div -0.4783)}{(\eta_{j,i}-(1.5321\div(\tau_{j,i}\div((\eta_{j,i}^{-7.6439})+0.5959))))}\right)}{-0.4783} \quad (6)$$

Greedy Selection:

$$\frac{-2.2824+7.3090^{-24.3948\tau_{i,j}}}{(-7.3090-\frac{\eta_{j,i}*-0.3779}{-0.7022}+3.3090)+-0.3779\eta_{j,i}} \quad (7)$$

Tournament Selection:

$$\frac{3.0640-\tau_{i,j}}{\eta_{i,j}} \quad (8)$$

A comparison of the above formulas to formula (1) (or perhaps more correctly formula (4)) illustrates that the one which appears the most similar to the original formula (the tournament selection result), actually achieves the worst fitness in training (approximately 1.3x greater than the optimal). The remaining two evolved formulas, however, appear quite distinct from the default formula and yet achieve nearly optimal results in training. The convergence graphs for each training session are presented in figures 3, 4, and 5.

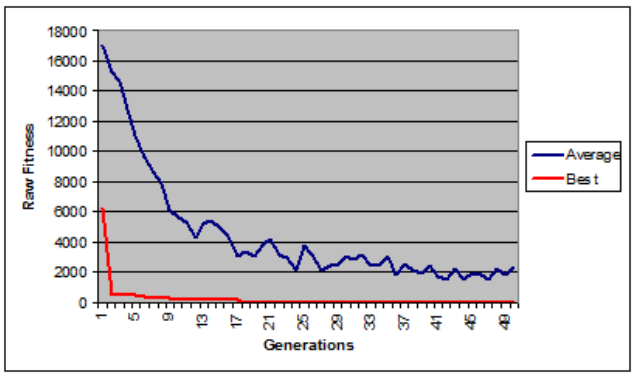

Figure 3: GP system convergence with AS using typical roulette selection

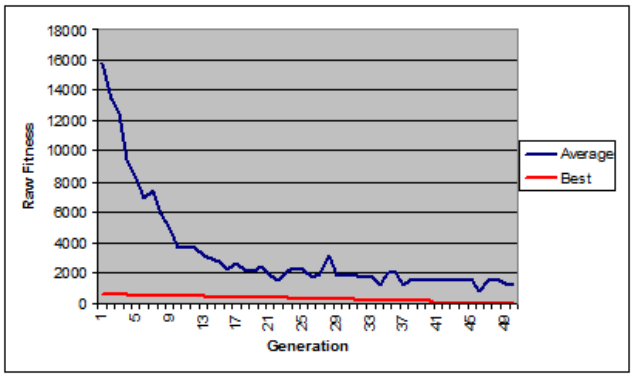

Figure 4: GP system convergence with AS using greedy selection

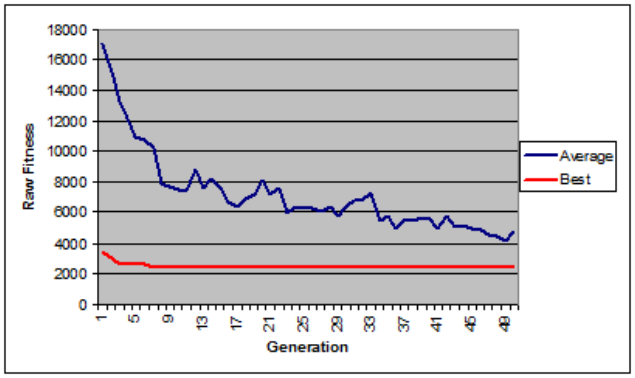

Figure 5: GP system convergence with AS using tournament selection

The testing results for the berlin52.tsp, beir127.tsp, and rat195.tsp instances are presented in tables 6, 7, and 8 respectively. Average fitness is the average result of 100 testing executions and best fitness is the best result of those 100 executions. Selection refers to the type of selection mechanism used by the AS algorithm. The results preceded by the term 'Evolved' refer to those executions that utilize the formulae evolved by the GP system, while the results preceded by the term 'Default' refer to testing executions which use the original AS decision formula. Finally, the smallest fitness per table (best and average), are highlighted in bold.

Table 6: Testing results for the berlin52.tsp instance (Optimal: 7542)

	Selection	Best Fitness	Avg. Fitness
Evolved	Roulette	7663.59	**7880.08**
	Greedy	8093.35	8093.35
	Tournament	10575.28	11004.64
Default	Roulette	**7549.29**	7884.52
	Greedy	8182.19	8182.19
	Tournament	10472.68	11216.07

Table 7: Testing results for the bier127.tsp instance (Optimal: 118282)

	Selection	Best Fitness	Avg. Fitness
Evolved	Roulette	**124266.47**	**127710.96**
	Greedy	128035.00	128035.00
	Tournament	239203.57	255756.58
Default	Roulette	125840.87	130336.13
	Greedy	127849.42	127849.42
	Tournament	238776.42	257512.69

Table 8: Testing results for the rat195.tsp instance (Optimal: 2323)

	Selection	Best Fitness	Avg. Fitness
Evolved	Roulette	2464.47	2495.69
	Greedy	2489.81	**2489.81**
	Tournament	7110.37	7450.82
Default	Roulette	**2436.33**	2532.93
	Greedy	2550.94	2550.94
	Tournament	7091.15	7595.48

It can be readily seen from these tables that the evolved formula for the roulette selection mechanism is at least comparable to the original (default) decision formula. In all three instances, the evolved roulette formula achieves a better average fitness than the original formula, and on one instance (bier127) actually improves on the best fitness found. The evolved greedy formula does not always outperform the original AS system on either its typical setup (using roulette), or on the newly attempted greedy setup, but it does remain on a comparable footing. The tournament selection mechanism appears to require more work for both the evolved and default formulas.

A more formal comparison of the above algorithms utilizing a two tailed t-test produces the results found in table 9. Here both the greedy and roulette evolved formulas are compared to the default formula using roulette selection to see if any improvements are gained over the original AS. Given the standard 95% confidence interval the p-values which signify a positive change that is not likely to be due to randomness are highlighted in bold in this table. This occurs for both the evolved roulette and greedy formulas on the bier127 and rat195 instances. However, the difference on the berlin52 instance using the evolved roulette formula is too similar to the original to dismiss the possibility of randomness. The change using the evolved greedy formula is a decrease in fitness on this instance.

Table 9: P-values from 2-tailed t-test between the Evolved and Default results

		Roulette	Greedy
Default	Berlin52	0.834606	3.7373E-19
	Bier127	**3.5871E-22**	**8.13137E-20**
	Rat195	**2.01331E-14**	**4.90738E-18**

5. CONCLUSION

Overall the notion of evolving an improvement to a relatively well established and successful algorithm is an exciting one. The formulae evolved here have exhibited significant promise in this endeavour. The best two evolved formulas (roulette and greedy) outperformed the original Ant System decision formula on average over two unseen instances that were both larger than double the size of the original instance.

The formulas evolved here, although not aesthetically pleasing, do potentially offer an improvement to the original Ant System. Further testing on a wider variety of problems must be done before these formulas can be sincerely declared an effective replacement to the original formula. However, the results presented here do suggest that this area of research can lead to beneficial improvements.

There are many directions for potential future work from here. Firstly, the reintroduction of array operations may offer the GP system more freedom of expression to improve on the decision formula. In that same direction it is hoped that by removing the explicit conversion to numbers between 0 and 1 which occurs here, that the GP system can fully utilize the array operations. Additional language primitives such as the logic set may provide further expressive power. The pitfall with the above directions, however, is that the increase in freedom may make the search space far too broad to find any good areas to exploit. Alternatively, the application of this process to more complex problems, such as the Vehicle Routing Problem could create more effective or perhaps problem specific improvements to the Ant System algorithm.

6. REFERENCES

[1] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, pp. 29–41, 1996.

[2] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization - artificial ants as a computational

intelligence technique," *IEEE Comput. Intell. Mag*, vol. 1, pp. 28–39, 2006.

[3] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53–66, 1997. [Online]. Available: http://dx.doi.org/10.1109/4235.585892

[4] J. C. F. Pujol and R. Poli, "Optimization via parameter mapping with genetic programming," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. LNCS, vol. 3242. Birmingham, UK: Springer-Verlag, 18-22 Sep. 2004, pp. 382–390.

[5] A. H. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 205–218.

[6] A. I. Esparcia-Alcazar and K. C. Sharman, "Evolving recurrent neural network architectures by genetic programming," Faculty of Engineering, Glasgow G12 8QQ, Scotland, Technical Report CSC-96009, 1996.

[7] J. C. F. Pujol and R. Poli, "Evolving the architecture and weights of neural networks using a weight mapping approach," University of Birmingham, School of Computer Science, UK, Technical Report CSRP-99-05, 10 Feb. 1999. [Online]. Available: ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1999/CSRP-99-05.ps.gz

[8] L. Dioşan and M. Oltean, "Evolving crossover operators for function optimization," in *Proceedings of the 9th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, vol. 3905. Budapest, Hungary: Springer, 10 - 12 Apr. 2006, pp. 97–108.

[9] T. Stutzle and H. Hoos, "Improvements on the ant-system: Introducing the max-min ant system," *Journal of Future Generation Computer Systems*, vol. 16, pp. 889–914, 2000.

[10] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.

[11] "Genetic programming homepage," July 2008. [Online]. Available: http://www.genetic-programming.com

[12] M. O'Neill, R. Poli, W. B. Langdon, and N. F. McPhee, "A field guide to genetic programming," *Genetic Programming and Evolvable Machines*, march 2008. [Online]. Available: http://dx.doi.org/10.1007/s10710-008-9073-y

[13] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems*, vol. 9, pp. 193–212, 1995.

APPENDIX

A. JAVA-EXECUTABLE FORMULAS

The following formulas are the java-executable versions of formulas (6), (7), and (8). Here, pheromone replaces τ and dist replaces η.

Roulette:
((Math.pow(dist[j][i], -11.5299) / 14.2606) / -0.4783) / (dist[j][i] - (1.5321 / (pheromone[j][i] / (Math.pow(dist[j][i], -7.6439) + 0.5959)))) / -0.4783

Greedy:
(-2.2824 + Math.pow(7.3090, (-14.1231 * (1.7273 * pheromone[i][j])))) / (((-7.3090 - ((dist[j][i] * -0.3779) / -0.7022)) + 3.3090) + (dist[j][i] * -0.3779))

Tournament:
(3.0640 - pheromone[i][j]) / (dist[i][j])