Designing Pheromone Update Strategies with Strongly Typed Genetic Programming

Jorge Tavares[1] and Francisco B. Pereira[1,2]

CISUC, Department of Informatics Engineering, University of Coimbra
Polo II - Pinhal de Marrocos, 3030 Coimbra, Portugal[1]
ISEC, Quinta da Nora, 3030 Coimbra, Portugal[2]
jorge.tavares@ieee.org, xico@dei.uc.pt

Abstract. Ant Colony algorithms are population-based methods widely used in combinatorial optimization problems. We propose a strongly typed genetic programming approach to automatically evolve the communication mechanism that allows ants to cooperatively solve a given problem. Results obtained with several TSP instances show that the evolved pheromone update strategies are effective, exhibit a good generalization capability and are competitive with human designed variants.

1 Introduction

The application of a global optimization algorithm to a specific problem usually requires some customization of the method. Tailoring ranges from simple parameter specification to more delicate problem-specific modifications, such as the design of operators or search strategy tuning. This is a crucial step for the success of the optimization, as one aims to maintain the ability of the algorithm to perform a robust exploration of the search space, while granting it some specific information that helps to efficiently discover good quality solutions for that particular problem. Usually, the design and adaptation of the algorithms is carried out manually, even though there are some recent proposals that discuss the possibility to automate the design of optimization methods [8, 4, 9].

Ant Colony Optimization (ACO) is a population-based method loosely inspired by pheromone-based strategies of ant foraging. It was originally proposed by Dorigo in 1992 and, since then, it has been successfully applied to difficult optimization problems [5]. Actually, there are several variants of ACO algorithms with differences, e.g., in the way pheromone levels are updated throughout the optimization process. Adjusting the key components of an ACO algorithm may allow its application to new situations and/or enhance its effectiveness on problems that it usually addresses [1]. However, performing the right modifications is far from trivial and requires a deep understanding of both the algorithm's behavior and the properties of the problem to solve.

In this paper we propose a framework to automatically discover effective pheromone update strategies for an ACO algorithm. In our approach, a Genetic Programming algorithm (GP) [10] evolves a population of strategies, seeking for

candidates that can be used by an ACO method applied to the optimization of the Traveling Salesperson Problem (TSP). An accurate evaluation of individuals generated by the GP algorithm is a crucial step in the evolution and it must reflect how well they help ACO algorithms to discover TSP short tours. In concrete, the evaluation of each individual is done by inserting it in an ACO algorithm and verifying how well the encoded strategy behaves in the optimization of a simple TSP instance.

In a recent work, we reported a preliminary application of standard GP to evolve pheromone update methods [13]. Results obtained with the TSP showed that the evolved strategies performed well when compared to three simple ACO architectures: Ant System, Elitist Ant System and the Rank-based Ant System. Here we adopt a strongly typed GP variant (STGP) [7] and consider several combinations of functions and terminals sets used to build solutions. A comprehensive analysis of the GP algorithm behavior is accomplished by investigating how the evolved strategies perform when compared to the competitive Max-Min Ant System (MMAS) [12, 5]. Different instances of the TSP will be used to assess the effectiveness of the proposed approach and we focus our analysis on three key issues: i) Study how the composition of the function and terminal sets impact the performance of the GP algorithm; ii) Compare the effectiveness of strategies that were evolved by the GP against the MMAS human developed strategies; iii) Verify the generalization ability of the evolved strategies. Results presented in this paper show that, even though the GP algorithm relies on a single TSP instance, evolved strategies maintain its effectiveness when applied to other instances.

The paper is structured as follows: in section 2 we present a general description of ACO algorithms. Section 3 comprises a detailed presentation of the system used to evolve pheromone update strategies. Section 4 contains the experimentation and analysis. Finally, in section 5 we summarize the conclusions and highlight directions for future work.

2 ACO Algorithms

The first ACO algorithm, Ant System (AS), was conceived to find the shortest path for the well-known TSP, but soon it was applied to several different types of combinatorial optimization problems [5]. To apply an ACO algorithm to a given problem, one must first define the solution components. A connected graph is then created by associating each component with a vertex and by creating edges to link vertices. Ants build solutions by starting at a random vertex and iteratively selecting edges to add new components. From a specific vertex, ants make a probabilistic choice of the new edge to cross. The probability of choosing an edge depends on the heuristic information and pheromone level of that specific path. Higher pheromone levels signal components that tend to appear in the best solutions already found by the colony. After completing a solution, ants provide feedback by depositing pheromone in the edges they just crossed. The amount of pheromone is proportional to the quality of the solution. To avoid stagnation,

pheromone trail levels are periodically decreased by a certain factor. Following these simple rules until a termination criterion is met, a solution to the problem will emerge from the interaction and cooperation made by the ants.

MMAS is an ACO variant proposed by Stützle and Hoos [12]. It focuses on the exploitation of recent search history since only the best ant is allowed to update the pheromone trail according to the following rule:

$$\tau_{ij}(t+1) = \rho \times \tau_{ij}(t) + \frac{1}{f(s^{best})} \tag{1}$$

where τ_{ij} is the pheromone level on edge joining solution components i and j, ρ is the evaporation rate and $f(s^{best})$ is the cost of the solution of the best ant. The selected ant might be the one that found the best solution in the current iteration or the one that found the best solution since the beginning of the run. Additionally, MMAS has a mechanism to limit the range of possible pheromone trail levels and it may perform a restart when no improvement is seen in a given number of iterations.

3 Evolving Pheromone Trail Update Methods

The framework used to evolve pheromone update strategies contains two components: a STGP engine and an AS algorithm. The main task of GP is to evolve individuals that encode effective trail update strategies, i.e., it aims to evolve a rule that replaces equation 1. It starts with a population of random strategies and iteratively seeks for enhanced solutions. The job of the AS algorithm is to assign fitness to each solution generated by GP: whenever an evolved pheromone update strategy needs to be evaluated, GP executes the AS algorithm to solve a given TSP instance (using the encoded strategy as the update policy). The result of the optimization is assigned as the fitness value of that individual.

3.1 Strongly Typed Genetic Programming Engine

The GP engine adopts a standard architecture: individuals are encoded as trees and ramped half-and-half initialization is used for creating the initial population. The algorithm follows a steady-state model, tournament selection chooses parents and standard genetic operators for manipulating trees are used to generate descendants. STGP is a variant of GP that enforces data type constraints in the evolved programs [7]. In STGP, each terminal has an assigned type and every function has a return type and a specified type for each of its arguments. Restrictions enforced by STGP provide an advantage over standard GP when dealing with situations that consider multiples data types. This is what happens with the problem addressed in this paper, as the GP algorithm must deal with, e.g., real values or sets of ants. In a previous work [13] we assumed the closure principle [10], but the analysis of results showed that forcing all functions to handle any possible data type hindered the performance of the optimization framework. Therefore, in this paper, we adopt the strong typing principle and hence assure

that GP only generates trees satisfying type constraints (see [7] for details about the initialization procedure and the application of genetic operators).

A key decision in the development of the framework is the definition of the function and terminal sets used by GP, as they will determine which components can be used in the design of pheromone update strategies. Our aim is to show that the proposed approach is able to evolve such strategies. Therefore, to validate our ideas we keep the definition of the function and terminal sets as simple as possible. We consider three different function and terminals sets. The first set allows the replication of standard AS and EAS methods and is composed by:

- *(prog2 p1 p2)* and *(prog3 p1 p2 p3)*: Sequential execution of two or three functions/terminals. The last result is returned; all types are generic, i.e., any type is accepted.
- *(evaporate rate)*: Standard evaporation formula with a given *rate*. The rate is of type real and the return type is generic.
- *(deposit ants amount)*: *ants* deposit a given *amount* of pheromone (type integer). The parameter *ants* can be an array of ants or a single one (type ant). The return type is generic.
- *(all-ants)*, *(best-ant)*, *(rho)*: Return respectively an array with all ants (type ant), the best ant found so far or a fixed evaporation rate (type real).
- *(integer)* and *(real)*: Ephemeral integer and real constants.

The second and third sets differ from the first one in two ways. We remove an explicit way for all the ants to make a deposit, i.e., *(all-ants)* does not exist. In the second set, we add the function *(rank number)* that receives an integer and returns an array of ants. This function sorts the ants by decreasing quality and returns the best *number* of them. In the third set, this function is transformed into a terminal that always returns the 10% best ants. A comparison between the results obtained by the three different sets will help us to gain insight into the key features that help to evolve successful update strategies.

3.2 Related Work

There are several efforts for granting bio-inspired approaches the ability to self adapt their strategies. On-the-fly adaptation may occur just on the parameter settings or be extended to the algorithmic components. One pioneer example of self-adaptation is the well-known 1/5 success rule used to control the mutation strength for the (1+1)-ES. Hyper-heuristics [2] and multimeme strategies [6] deal with the development of the best combination of methods for a given problem. The concept of hyper-heuristics identifies a framework composed by an EA that evolves a combination of specific heuristics for effectively solving a problem. Multimeme strategies are memetic algorithms that learn on-the-fly which local search component should be used. In what concerns the adaptation of the optimization algorithm, Diosan and Oltean proposed an evolutionary framework that aims to evolve a full-featured Evolutionary Algorithm (EA) [4].

As for the Swarm Intelligence area, there are some reports describing the self-adaptation of parameter settings (see, e.g., [1, 14]). Additionally, a couple of

Table 1. Results from GP evolution for 30 runs with 25 generations.

	Hits	Best	MBF	Deviation	Average Depth	Average Nodes
STGP 1	0	427	430.03	4.90	4.10	13.77
STGP 2	23	426	439.17	26.77	4.07	9.64
STGP 3	30	426	426.00	0.0	3.77	10.06
GP 1	1	426	430.63	4.14	n/a	n/a
GP 2	6	426	446.33	43.38	n/a	n/a

approaches resemble the framework proposed in this paper. Poli et. al [9] use GP to evolve the equation that controls particle movement in Particle Swarm Optimization (PSO). Diosan and Oltean also did some work with PSO structures [3]. Finally, Runka [11] applies GP to evolve the probabilistic rule used by an ACO variant to select the solution components in the construction phase.

4 Experiments and Analysis

Selected instances from the TSPLIB[1] are used in the experiments. For all tests, the GP settings are: Population size: 100; Maximum tree depth: 5; Crossover rate: 0.9; Mutation rate: 0.05; Tourney size: 3. For the AS algorithm used to evaluate GP individuals we adopt the standard parameters found in the literature [5]: number of ants is the number of cities, $\alpha = 1$, $\beta = 2$, $\rho = 0.5$. The number of runs for all experiments is 30.

4.1 Evolution of the Update Strategies

In the first set of experiments we aim to establish the ability of GP to evolve update strategies and to identify performance differences between the three function/terminal sets defined in the previous section. The GP algorithm relies on a single TSP instance to evolve the strategies (eil51 – a small instance with 51 cities). The main reason for this choice is that the evaluation of each evolved strategy requires the execution of an AS algorithm. Two parameters, the number of runs and the number of iterations per run, define the optimization effort of the AS. On the one hand, if we grant the AS a small optimization period to evaluate a GP individual, then it might not be enough to correctly estimate the quality of an update strategy. On the other hand, a longer evaluation period can increase computational costs to unbearable levels. For the eil51 instance, AS finds near-optimal solutions within approximately 100 iterations. Extensive testing allows us to conclude that a single AS run is enough to evaluate a GP individual. The fitness value sent back to the GP engine is given by the best solution found (see [13] for a discussion of other evaluation configurations).

Table 1 contains the results of the STGP evolution using the three sets (STGP 1, STGP 2 and STGP 3). For completeness, we also present the evolution outcomes of standard GP, i.e., the non strongly typed GP described in

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

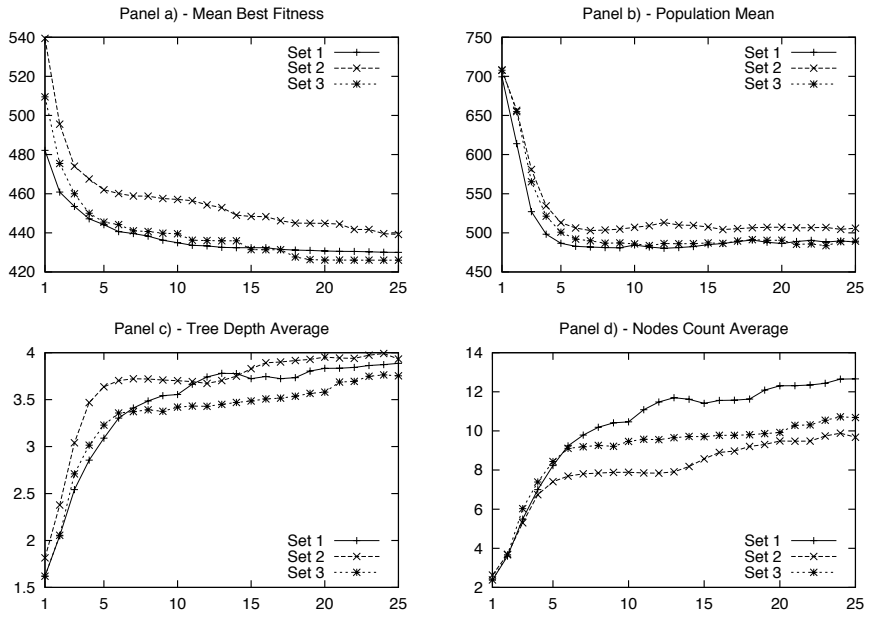

Fig. 1. STGP evolution plots with the 3 sets. Panels: a) Mean Best Fitness; b) Population Fitness; c) Tree Depth; d) Number of Nodes. Generations are on the x axis and fitness values are on the y axis in panel a) and b). Optimum solution has fitness 426.

[13]: the function/terminal sets of GP1 and GP2 are equivalent to STGP1 and STGP2, respectively. The column *Hits* displays the number of runs where evolved strategies were able to discover the optimal solution for eil51. A brief overview of these results reveals that there are noteworthy differences between the STGP sets. Whilst set 1 was unable to find a tree that helped AS to discover the optimum, the remaining sets performed well. Set 2 evolved trees that found the optimum in 23 runs and set 3 was successful on all the 30 runs.

As expected, the components used by STGP are relevant to the quality of the evolved strategies. The terminal/function sets available to STGP1 do not contain high level components granting immediate access to a set of good quality ants, thus complicating the evolutionary optimization. Also, poor results are probably amplified by the sharp experimental conditions (low number of generations performed by the GP and small maximum tree depth). On the contrary, ranking components used by STGP2 and STGP3 allow a simplified evolution of greedy update strategies. Results presented in table 1 confirm that greedy elements are important to enhance the effectiveness of ACO algorithms. The difference in performance between STGP2 and STGP3, particularly visible in the Mean Best Fitness (MBF) values, reveals that fixing the proportion of high quality ants further simplifies the discovery of effective strategies. Since GP does

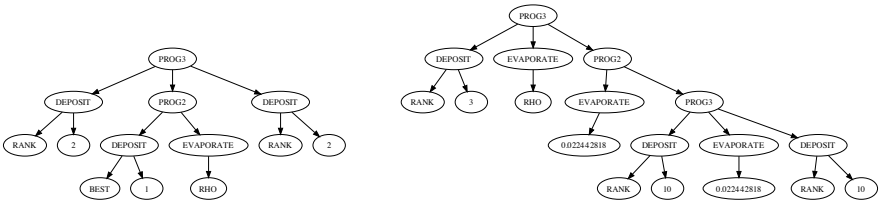

Fig. 2. Tree examples from set 3: on the left tree 326 and on the right tree 327.

not have to learn an appropriate proportion of ants used to update pheromone levels. It is nevertheless important to notice that there is an obvious drawback in providing a fixed proportion for the rank function, as this might compromise the robustness of the evolved strategies. This effect is not visible in the results displayed in table 1, but it might be perceived when harder TSP instances are used. If we compare the results of STGP to the original GP version, one can see that the evolutionary pattern of STGP1 and GP1 is identical. On the contrary, STGP2 clearly outperforms GP2, both when we consider the number of hits or MBF. These results give evidence of the advantage provided by the addition of specific type constraints, particularly when combined with components that promote greedy update strategies.

The plots in Figure 1 summarize the evolutionary behavior of the GP engine. Panels a) and b) display, for the three sets, the evolution of the MBF and of the mean fitness of the population. The gradual discovery of increasingly effective update strategies is evident, confirming the usefulness of the GP exploration. The evolution of the tree depth is comparable in the three sets (panel c)), but there are some noteworthy differences in the average number of nodes (panel d)). As expected, strategies evolved by STGP1 use more nodes than those of other GP configurations, because only the best ant or all the ants are allowed to deposit pheromone. AS algorithms are greedy and, to explore this property, trees created by STGP1 require more nodes to reinforce the best solution trails. This situation does not happen in the other sets because the rank-based components allow a direct access to subsets of high quality ants.

4.2 Validation and Comparison with Max-Min Ant System

To confirm the effectiveness of the best evolved solutions we will describe a set of additional experiments. Specifically, we aim to: i) verify how update strategies behave in larger TSP instances; ii) assess the absolute optimization performance of evolved strategies, by comparing them with MMAS. To focus our analysis, 10 representative trees were selected from the set of the 53 evolved solutions that were able to discover the optimal solution for eil51. Trees are identified with a numerical ID: solutions obtained by STGP2 start with digit 2, whereas

Table 2. Overview of the best trees, with 10000 iterations for 30 runs.

Instance	Tree ID	Best	Dist	MBF	Deviation	Dist	Branching
	222	426	0.00	429.20	1.92	0.75	2.05
	224	426	0.00	426.17	0.37	0.04	3.15
	226	426	0.00	428.70	2.24	0.63	2.03
	230	426	0.00	429.00	2.16	0.70	2.09
eil51	31	426	0.00	426.97	1.11	0.23	2.34
	321	426	0.00	427.37	1.30	0.32	2.37
	326	426	0.00	430.90	3.24	1.15	2.07
	327	426	0.00	432.13	4.29	1.44	2.01
	329	426	0.00	426.50	0.56	0.12	2.36
	330	426	0.00	427.83	2.00	0.43	2.15
	Mean	426	0.00	428.48	1.92	0.58	2.26
	222	21282	0.00	21470.97	265.82	0.89	2.11
	224	21282	0.00	21315.30	28.68	0.16	3.06
	226	21282	0.00	21444.00	196.77	0.76	2.08
	230	21282	0.00	21308.20	57.59	0.12	2.19
kroA100	31	21282	0.00	21289.17	17.39	0.03	2.50
	321	21282	0.00	21293.20	21.90	0.05	2.50
	326	21282	0.00	21372.47	112.86	0.43	2.13
	327	21282	0.00	21582.87	317.85	1.41	2.03
	329	21282	0.00	21300.70	16.28	0.09	2.45
	330	21282	0.00	21312.87	55.29	0.15	2.30
	Mean	21282	0.00	21368.97	109.04	0.41	2.33
	222	16118	2.14	16340.97	114.86	3.55	3.02
	224	16075	1.87	16180.97	62.24	2.54	3.61
	226	15942	1.03	16061.13	70.23	1.78	2.61
	230	15901	0.77	16011.23	61.23	1.47	2.57
d198	31	16032	1.60	16197.67	72.98	2.65	3.27
	321	16009	1.45	16122.00	51.13	2.17	3.00
	326	15955	1.11	16031.80	63.22	1.60	2.45
	327	15849	0.44	15924.90	53.49	0.92	2.19
	329	16031	1.59	16199.87	74.02	2.66	2.82
	330	15944	1.04	16032.90	57.77	1.60	2.80
	Mean	15985.6	1.30	16110.34	68.12	2.09	2.83
	222	42468	1.04	42859.90	206.69	1.98	2.28
	224	43052	2.43	43388.27	211.70	3.23	3.28
	226	42223	0.46	42771.57	194.80	1.77	2.25
	230	42455	1.01	42792.23	199.48	1.82	2.46
lin318	31	42660	1.50	42919.60	159.26	2.12	2.73
	321	42507	1.14	43090.90	227.63	2.53	2.76
	326	42372	0.82	42817.07	214.48	1.88	2.35
	327	42203	0.41	43081.30	333.34	2.50	2.16
	329	42990	2.29	43514.30	271.44	3.53	2.68
	330	42231	0.48	42773.90	163.68	1.77	2.60
	Mean	42516.1	1.16	43000.90	218.25	2.31	2.56
	222	55452	9.20	56733.43	627.16	11.73	3.48
	224	56263	10.80	57386.23	498.18	13.01	3.71
	226	56199	10.68	56995.47	461.99	12.24	3.33
	230	51917	2.24	52755.30	382.93	3.89	2.85
pcb442	31	56108	10.50	57321.90	581.36	12.89	3.51
	321	54662	7.65	55754.27	645.12	9.80	3.51
	326	51804	2.02	52408.70	415.42	3.21	2.52
	327	51408	1.24	51870.83	274.58	2.15	2.22
	329	52815	4.01	53777.63	365.61	5.91	3.12
	330	54418	7.17	55788.73	724.40	9.87	3.43
	Mean	54104.6	6.55	55079.25	497.67	8.47	3.17

Table 3. Comparison with MMAS, with 10000 iterations for 30 runs.

Instance	AS	Best	Dist	MBF	Deviation	Dist	Branching
eil51	Tree 224	426	0.00	426.17	0.37	0.04	3.15
kroA100	Tree 31	21282	0.00	21289.17	17.39	0.03	2.50
d198	Tree 327	15849	0.44	15924.90	53.49	0.92	2.19
lin318	Tree 226	42223	0.46	42771.57	194.80	1.77	2.25
pcb442	Tree 327	51408	1.24	51870.83	274.58	2.15	2.22
eil51	MMAS	426	0.00	427.23	0.67	0.29	8.33
kroA100	MMAS	21431	0.70	21553.97	63.82	1.28	7.61
d198	MMAS	16141	2.29	16276.23	61.27	3.14	5.54
lin318	MMAS	45243	7.65	45912.77	335.12	9.24	6.11
pcb442	MMAS	58211	14.64	60009.20	714.16	18.18	4.40

solutions discovered by STGP3 begin with digit 3. In figure 2 we present two examples of trees that were evolved by STGP3. To allow for a fair comparison, all experiments presented in this section were performed in a MMAS environment. Therefore, evolved strategies were run in conjunction with a restart mechanism and a lower-upper bound update of the pheromone limits. Since the evolutionary process did not use these mechanisms, this is an additional robustness test for the evolved strategies. Five TSP instances were selected to assess the performance of the evolved methods: eil51, kroA100, d198, lin318 and pcb442. The size of each instance is given by the number in the name (e.g., lin318 has 318 cities). The AS maintains the same parameter settings used before, with two exceptions: ρ is set to 0.02 (the recommended value for MMAS) and the number of iterations is increased to 10000 to allow a correct optimization period for larger instances.

Table 2 contains the results for the 10 selected trees on the five TSP instances. This table displays the identification of the strategy (column *Tree ID*), the best solutions found, the MBF and standard deviation, with the respective percentage distance to the optimum for all these values (columns *Dist*). The last column is the average of the evolution of the branching factor (with $\lambda = 0.05$), a measure to determine the convergence of the pheromone matrix. For completeness, we add a row containing the average values for each instance. In general, results show that evolved strategies perform well across the instances, and thus, are able to generalize. For the two smaller instances, all strategies are able to find the optimum and the MBF is close to the optimal solution. For instances d198 and lin318, the best tour was not found but the solutions obtained by different trees are very close (between 0.41% and 2.43%). MBF values follow the same pattern, with distances still close to the optimum. As for the larger instance, the performance of the evolved strategies is not so effective. Nevertheless, some of the trees are able to generate good quality solutions. The most remarkable example is tree 327 with distances of 1.24% (best solution) and 2.15% (MBF). Two factors help to explain the performance differences visible in the larger instance. First, a visual inspection reveals that the most effective trees are the ones that include more evaporation instructions. This, coupled with the addition of the restart mechanism, prevents the pheromone matrix from saturating in certain positions and allows evolved strategies to avoid local minima. The difference in

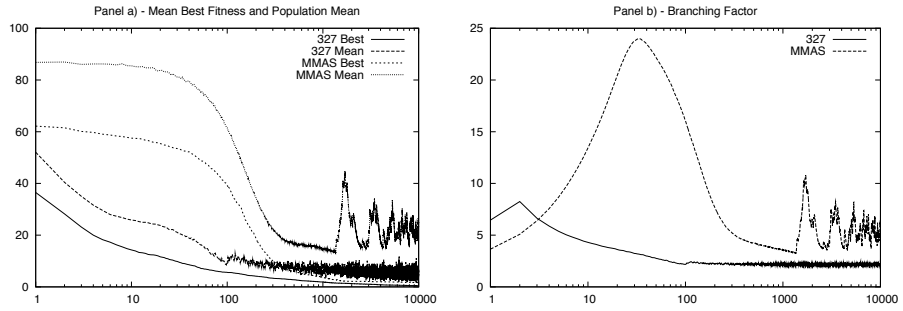

Fig. 3. Plots showing the evolution of the best solution, mean solutions and the branching factor for 10000 iterations (x axis) on the d198 instance.

the composition of the function/terminal sets is another relevant issue. In the largest TSP instance, update strategies that were evolved by STGP3 tend to obtain better results. When compared to STGP2, the optimization task of STGP3 is simpler as it does not need to discover the proportion of high quality ants that update the pheromone matrix. For efficiency reasons, we selected a small TSP instance to evolve update strategies. Possibly this instance is too simple and it does not create enough evolutionary pressure to push STGP2 to discover the correct proportion of ants involved in pheromone level update. This inefficiency is not visible in easy instances, but as the number of cities increases, the performance of strategies evolved by STGP2 gradually decreases. The impact of the training instances on the quality and robustness of the evolved update strategies is a topic that we will address in future research.

A fundamental issue in the analysis is the comparison of the evolved solutions against human developed strategies. The top lines of table 3 contain the best results obtained by the evolved strategies on the selected TSP instances. The bottom half of the table shows the outcomes of our MMAS implementation without local search using standard parameter settings (see [5] for a complete description). Results clearly show that the evolved strategies outperform MMAS in every single instance and in every performance criteria (best solution found and MBF). With the exception of eil51, even the averages of all evolved trees from table 2 are better than MMAS results. The only weak point of the evolved solutions is the branching factor. MMAS consistently maintains a higher diversity level, suggesting that the evolved strategies might suffer from excessive greediness. In Figure 3 we can see the comparison between an evolved strategy and MMAS in instance d198. This is representative of all trees on all instances. Panel a) displays the evolution of the MBF and of the population mean (measured in % distance to the optimum). The standard behavior of MMAS is clearly seen: a slow improvement in the first 100 iterations, followed by a quick optimization until 1000 iterations and finally a slow improvement until the end. The evolved strategy performs a much quicker optimization attaining good quality solutions

Table 4. Comparison with MMAS+LS [12], with n x 100 iterations for 30 runs.

	kroA100		d198		lin318		pcb442	
Variant	MBF	Dist	MBF	Dist	MBF	Dist	MBF	Dist
Best Evolved	21289.17	0.03	15907.40	0.81	42586.73	1.33	51363.9	1.15
MMAS+LS	21481.00	0.94	16056.00	1.75	42934.00	2.15	52357.00	3.11
10+all+LS	21502.00	1.03	16197.00	2.64	43667.00	3.90	53993.00	6.33
10+best+LS	21427.00	0.68	15856.00	0.48	42426.00	0.94	51794.00	2.00

earlier. The restart mechanism is activated around iteration 100 with the evolved strategy, whereas for MMAS it only appears after 1000 iterations. The impact of restarts is higher on MMAS, as it seems to reintroduce a larger diversity in the population. This is confirmed by the evolution of the branching factor, shown in panel b). The convergence of the pheromone matrix is stronger with the evolved strategy, thereby confirming its greedy behavior.

We also address the issue of using Local Search (LS). It is well-known that AS algorithms in general, and MMAS in particular, become more effective with the addition of a LS step. Our goal in this paper is to focus on the influence of pheromone update methods and not on the whole AS architecture. In spite of that, in table 4 we provide a simple performance comparison between the evolved strategies and MMAS with LS. The outcomes of the MMAS with LS were obtained in [12]. In that paper, the number of iterations performed is set to $n \times steps$, where n is the number of cites in the instance and *steps* is a constant. Table 4 presents results when *steps* is set to 100. The first row (*Best Evolved*) contains the best results obtained by the evolved strategies. The other rows display results from several MMAS+LS variants (see [12] for details). A brief perusal of the results reveals that the evolved strategies without LS are competitive with MMAS architectures with LS. They discover better solutions in instance kroA100 and, in the remaining instances, obtain tours close to those discovered by the most effective MMAS+LS variant. The indications given by this preliminary comparison are encouraging. Still, it is important to refer that the LS used is the 2-opt operator and more effective operators exist, e.g., the Lin-Kernighan heuristic.

5 Conclusions

We proposed a STGP framework to accomplish the automatic evolution of update pheromone strategies, a key component in the design of ACO algorithms. Results reveal that update strategies are effective for solving the TSP instance for which they were evolved, while they also exhibit a good generalization capability. Moreover, they are competitive in performance with MMAS and preliminary results also indicate that the evolved strategies without LS can be competitive with the standard MMAS using a basic LS operator. It must be pointed out that the system uses high-level function and terminal sets, resembling existing ACO methods. This choice undermines the possibility of evolving strategies that

strongly deviate from standard ones. Moreover, the evolution is not too difficult as the key components are provided and GP just needs to find the proper arrangement. Nevertheless, the results show that evolution did not converge to the standard methods and found different designs in structure and in parameters.

The study presented here raises important questions that we intend to pursue in the near future. The evolved strategies show a greedy behavior, even more than standard AS architectures. We will study alternatives that help to reduce greediness, as we believe this will help to evolve more robust and scalable strategies. Also, the selection of the instance used to evaluate GP solutions plays an important role and we intend to investigate the impact of using instances of different size and/or distinct properties. These are all important aspects and this paper is a step forward in the effort of developing Self Ant Systems.

References

1. Botee, H.M., Bonabeau, E.: Evolving ant colony optimization. Advanced Complex Systems 1, 149–159 (1998)
2. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Tech. Rep. NOTTCS-TR-SUB-0906241418-2747, University of Nottingham (2010)
3. Diosan, L., Oltean, M.: Evolving the structure of the particle swarm optimization algorithms. In: EvoCOP 2006 Proceedings. pp. 25–36 (2006)
4. Diosan, L., Oltean, M.: Evolutionary design of evolutionary algorithms. Genetic Programming and Evolvable Machines 10(3), 263–306 (2009)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
6. Krasnogor, N., Blackburnem, B., Hirst, J., Burke, E.: Multimeme algorithms for protein structure prediction. In: PPSN VII Proceedings. Lecture Notes in Computer Science, vol. 2439, pp. 769–778. Springer (2002)
7. Montana, D.J.: Strongly typed genetic programming. Evolutionary Computation Journal 3(2), 199–230 (1995)
8. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. Evolutionary Computation Journal 13, 387–410 (September 2005)
9. Poli, R., Langdon, W.B., Holland, O.: Extending particle swarm optimisation via genetic programming. In: EuroGP 2005 Proceedings. pp. 291–300 (2005)
10. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk` (2008), (With contributions by J. R. Koza)
11. Runka, A.: Evolving an edge selection formula for ant colony optimization. In: GECCO 2009 Proceedings. pp. 1075–1082 (2009)
12. Stutzle, T., Hoos, H.: Max-min ant system and local search for the traveling salesman problem. In: ICEC Proceedings. pp. 309–314. IEEE Press (1997)
13. Tavares, J., Pereira, F.B.: Evolving strategies for updating pheromone trails: A case study with the tsp. In: PPSN XI Proceedings. Lecture Notes in Computer Science, vol. 6239, pp. 523–532. Springer (2010)
14. White, T., Pagurek, B., Oppacher, F.: ASGA: Improving the ant system by integration with genetic algorithms. In: Proceedings of the 3rd Genetic Programming Conference. pp. 610–617. Morgan Kaufmann (1998)