Automatic Design of Ant Algorithms with Grammatical Evolution

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. We propose a Grammatical Evolution approach to the automatic design of Ant Colony Optimization algorithms. The grammar adopted by this framework has the ability to guide the learning of novel architectures, by rearranging components regularly found on human designed variants. Results obtained with several TSP instances show that the evolved algorithmic strategies are effective, exhibit a good generalization capability and are competitive with human designed variants.

1 Introduction

Ant Colony Optimization (ACO) identifies a class of robust population-based optimization algorithms, whose search behavior is loosely inspired by pheromone-based strategies of ant foraging [1]. The first ACO algorithm, Ant System (AS), was proposed by Dorigo in 1992. Since then, many different variants have been presented with differences, e.g., in the way artificial ants select components to build solutions or reinforce promising strategies. Researchers and practitioners aiming to apply ACO algorithms to a specific optimization situation are confronted with several non-trivial decisions. Selecting and tailoring the most suitable variant for the problem to be addressed and choosing the best parameter setting helps to enhance search effectiveness. However, making the right decisions is difficult and requires a deep understanding of both the algorithm's behavior and the properties of the problem to solve.

In recent years, several automatic ACO design techniques were proposed to overcome this limitation. Reports in literature range from approaches to autonomous/adaptive parameter settings to the configuration of specific algorithmic components [2]. Two examples of automatic design are the evolution of pheromone update strategies by Genetic Programming (GP) [3, 4] and the synthesis of an effective ACO for multi-objective optimization [5].

In this paper we present a complete framework to evolve the architecture of a full-fledged ACO algorithm. Grammatical Evolution (GE) [6] is adopted as the design methodology, as it allows for a simple representation and modification of flexible algorithmic strategies. The grammar used by the GE algorithm defines both the potential components that can be adopted when designing an ACO algorithm and also the general structure of the optimization method. We describe a set of experiments that deal with the discovery of ACO architectures for the optimization of the Traveling Salesperson Problem (TSP). The analysis of results helps to understand if the design algorithm converges to ACO architectures regularly applied to the TSP, or, on the contrary, evolves novel combinations of basic components that enhance the effectiveness of the optimization. Additionally, we address the generalization ability of the learned architectures.

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

2 Ant Colony Optimization

AS was the first ACO algorithm and it was conceived to find the shortest path for the well-known TSP, although it was soon applied to different types of combinatorial optimization problems [1]. To apply an ACO algorithm to a given problem, one must first define the solution components. A connected graph is then assembled by associating each component with a vertex and by creating edges to link vertices. Ants build solutions by starting at a random vertex and by stochastically selecting edges to add new components. 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 deposited 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.

Algorithm 1 Ant Colony Optimization

```
init_parameters
init_pheromone
while not_termination do
   generate_solutions
   pheromone_update
   daemon_actions
end while
return  best_solution
```

Algorithm 1 presents the pseudo-code of a general ACO algorithm. Daemon actions include a set of optional actions, e.g., the application of local search or pheromone matrix restart.

3 The Evolutionary Framework

The framework used to evolve the architecture of an ACO algorithm contains two components: a GE engine and a compiler/evaluator of ACO architectures. GE is a GP branch that allows for an efficient evolution of complete algorithmic structures. It relies on a grammar composed by a set of production rules in a Backus-Naur form, defining both the components that can appear and the overall organization of the algorithm to be evolved. The GE iterative search process is somehow decoupled from the actual programs being evolved, since individuals are codified as integer vectors. This allows for the application of a straightforward evolutionary algorithm to perform optimization. Whenever a solution needs to be evaluated, a complementary mapping process uses the grammar to decode an integer solution into a syntactically correct program (see [6] for details).

The quality of a GE individual is directly related to its ability to find good solutions for the TSP. In fitness assignment, the first step is to apply an ACO compiler to assemble a decoded GE individual into a running ACO. Then, the evolved architecture is executed and tries to solve a given TSP instance. The result of the optimization is assigned as the fitness value of that GE individual.

3.1 Grammar Definition

The grammar creates individuals containing an initialization step followed by an optimization cycle. The first stage consists in the initialization of the pheromone matrix and the selection of parameters. The main loop comprises the construction of the solutions by the ants, trail evaporation, trail reinforcement and daemon actions. Each component contains several alternatives to implement its specific task. Additionally, most blocks are optional. Different values for the parameters required by an ACO algorithm are also defined in the grammar, which allows the GE engine to automatically adapt the most suitable setting for a given architecture. The research described in this paper focus on the relevance assessment of existing ACO components and on the discovery of novel interactions that might help to enhance optimization performance. As such, daemon actions do not consider the possibility of applying local search, as the effect of this operation would dilute the influence of the remaining components.

The grammar defined is constrained in the sense that production rules enforce a global structure that is similar to existing ACO architectures. Components, e.g., solution construction, reinforcement or evaporation, can only appear in a pre-specified order and certain blocks cannot be used more than once. This is a deliberate design option. With this framework, the GE system is able to generate all main ACO algorithms: Ant System (AS), Elitist Ant System (EAS), Rank-Based Ant System (RAS), Ant Colony System (ACS) and Max-Min Ant System (MMAS). Additionally, the search space contains many other combinations of blocks that define alternative ACO algorithms. The experiments described in the next section will provide insight into the search behavior of GE in this task. We will analyze if search converges to manually designed ACO architectures regularly applied to the TSP or, on the contrary, if it discovers novel combinations of blocks leading to the automatic design of optimization algorithms with enhanced effectiveness. The grammar for the GE engine is:

⟨*aco*⟩ ::= (aco ⟨*parameters-init*⟩ ⟨*optimization-cycle*⟩)
⟨*parameters-init*⟩ ::= (init ⟨*pheromone-matrix-init*⟩ ⟨*choice-info-matrix-init*⟩)
⟨*pheromone-matrix-init*⟩ ::= (init-pheromone-matrix ⟨*trail-amount*⟩)
⟨*trail-amount*⟩ ::= ⟨*initial-trail*⟩ | (uniform-trail ⟨*trail-min*⟩ ⟨*trail-max*⟩)
⟨*initial-trail*⟩ ::= ⟨*trail-min*⟩ | ⟨*trail-max*⟩ | (tas) | (teas ⟨*rate*⟩) | (tras ⟨*rate*⟩
    ⟨*weight*⟩) | (tacs) | (tmmas ⟨*rate*⟩)
⟨*choice-info-matrix-init*⟩ ::= (init-choice-info-matrix ⟨*alpha*⟩ ⟨*beta*⟩)
⟨*optimization-cycle*⟩ ::= (repeat-until ⟨*loop-ants*⟩ ⟨*update-trails*⟩ ⟨*daemon-actions*⟩)
⟨*loop-ants*⟩ ::= (foreach-ant make-solution-with ⟨*decision-policy*⟩
    (if ⟨*bool*⟩ (local-update-trails ⟨*decay*⟩)))
⟨*decision-policy*⟩ ::= (roulette-selection) | (q-selection ⟨*q-value*⟩) | (random-selection)
⟨*update-trails*⟩ ::= (progn ⟨*evaporate*⟩ ⟨*reinforce*⟩)
⟨*evaporate*⟩ ::= (do-evaporation
    (if ⟨*bool*⟩ (full-evaporate ⟨*rate*⟩))
    (if ⟨*bool*⟩ (partial-evaporate ⟨*rate*⟩ ⟨*ants-subset*⟩)))
⟨*reinforce*⟩ ::= (do-reinforce
    (if ⟨*bool*⟩ (full-reinforce))
    (if ⟨*bool*⟩ (partial-reinforce ⟨*ants-subset*⟩))
    (if ⟨*bool*⟩ (rank-reinforce ⟨*many-ants*⟩))
    (if ⟨*bool*⟩ (elitist-reinforce ⟨*weight*⟩))
⟨*daemon-actions*⟩ ::= (do-daemon-actions
    (if ⟨*bool*⟩ (update-pheromone-limits ⟨*update-min*⟩ ⟨*update-max*⟩))
    (if ⟨*bool*⟩ (restart-check)))
⟨*ants-subset*⟩ ::= ⟨*single-ant*⟩ | ⟨*many-ants*⟩
⟨*single-ant*⟩ ::= (all-time-best) | (current-best) | (random-ant) | (all-or-current-
    best ⟨*probability*⟩)
⟨*many-ants*⟩ ::= (all-ants) | (rank-ant ⟨*rank*⟩)
⟨*update-min*⟩ ::= ⟨*trail-min*⟩ | (mmas-update-min)
⟨*update-max*⟩ ::= ⟨*trail-max*⟩ | (mmas-update-max ⟨*rate*⟩)
⟨*weight*⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | (n-ants) | (max-rank ⟨*rank*⟩)
⟨*trail-min*⟩ ::= 0.000001
⟨*trail-max*⟩ ::= 1.0
⟨*alpha*⟩ ::= 1 | 2 | 3
⟨*beta*⟩ ::= 1 | 2 | 3
⟨*q-value*⟩ ::= 0.7 | 0.75 | 0.8| 0.85 | 0.9 | 0.95 | 0.98 | 0.99
⟨*decay*⟩ ::= 0.01 | 0.025 | 0.05 | 0.075 | 0.1
⟨*rate*⟩ ::= 0.01 | 0.1 | 0.25 | 0.5 | 0.75 | 0.9
⟨*probability*⟩ ::= 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9
⟨*rank*⟩ ::= 5 | 10 | 25 | 50 |75
⟨*bool*⟩ ::= t | nil

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. Grammar-based GP and GE have been used before to evolve complete algorithms, such as evolving data mining algorithms [7] and local search heuristics for the bin-packing [8].

In the area of Swarm Intelligence, there are some reports describing the self-adaptation of parameter settings (see, e.g., [9, 10]). Poli et al. [11] use GP to evolve the equation that controls particle movement in Particle Swarm Optimization (PSO). Diosan and Oltean also worked on the evolution of PSO structures [12]. On the topic of ACO evolution, Runka [13] applies GP to evolve the probabilistic rule used by an ACO variant to select the solution components in the construction phase. Tavares et al. applied GP and Strongly Typed GP to evolve pheromone update strategies [3, 4, 14]. Finally, López-Ibáñez and Stützle synthesize existing multi-objective ACO approaches into a flexible configuration framework, which is then used to automatically design novel algorithmic strategies [5].

4 Experiments and Analysis

In this section, our purpose is to gain insight into the ability of GE to automatically design effective ACO algorithms. Selected TSP instances from the TSPLIB[1] are used to validate our approach. For all experiments we performed 30 independent runs. Results are expressed as a normalized distance to the optimum.

4.1 Learning the Architectures

In the first set of experiments, we address the ability of GE to learn ACO architectures using the previously described grammar. We adopt the strategy proposed by [3, 4] to evaluate the individuals generated by the GE engine. In concrete, the ACO algorithm encoded in a solution is used to optimize a predetermined TSP instance (1 single run comprising of 100 iterations). The fitness value of that individual is given by the best solution found. This minimal evaluation methodology was adopted mainly due to efficiency reasons, since assigning fitness to an evolved architecture is a computational intensive task.

For all tests, the GE settings are: Population size: 64; Number of generations: 25; Individual size: 128 (with wrap); One-Point crossover with rate: 0.7; Standard Integer-Flip mutation with rate: 0.05; Tournament selection with tourney size: 3; Elitist strategy. Three distinct TSP training instances were selected to learn ACO architectures: eil76, pr76 and gr96 (the value represents the number of cities). Evolved individuals encode nearly all parameters required to run the ACO algorithm in the fitness assignment step. The exceptions are the size of the colony and the λ parameter for the branching factor. We set the number of ants to 10% of the number of cities (rounded to the nearest integer) and λ to 0.05.

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

Table 1. Overview of GE evolution results (for 30 runs with 25 generations).

Instances	eil76	pr76	gr96
Best Hits	20	4	12
Mean Best	0.0014 (±0.003)	0.0079 (±0.007)	0.0043 (±0.008)
Mean Ants	0.0725 (±0.062)	0.1171 (±0.073)	0.0941 (±0.074)
Mean Branching	8.58	11.21	11.24
Mean Generations	13.1	21.2	14.0

Table 1 contains the optimization results of GE using the three training instances. The row *Best Hits* displays the number of runs where evolved architectures were able to discover the optimal solution of the training instance. Row *Mean Best* contains the mean of the best solutions found, while *Mean Ants* gives the mean of all solutions discovered in the last generation. These two lines also display the standard deviation (in brackets). The last two rows contain the mean branching factor at the end of the optimization and the average number of generations that GE needs to reach the best solution. The low values appearing in lines *Mean Best* and *Mean Ants* confirm that evolution consistently found architectures that were able to find the optimum or near-optimum solutions. Moreover, there are no noteworthy variations in the outcomes obtained with different training instances. The branching factor indicates the degree of convergence of an ACO pheromone matrix. For the TSP, a value of 2.0 indicates full convergence. In Table 1 the mean branching is low for all training instances, although not close to full convergence. However, a closer inspection of the evolved architectures reveals that those which are able to generate better quality solutions for the TSP training instances, have a branching factor close to 2.0. As expected, higher branching values are usually present in architectures that did not perform as well.

Results from line *Best Hits* show that the instance selected for training impacts the likelihood of evolving ACO architectures that can discover the optimal solution in the evaluation step. With eil76, 20 runs were able to learn algorithms that discover the training instance optimum, whereas in pr76 only 4 runs were successful. The reason for this effect is probably related to the spatial distribution of the cities. Although all training instances have a similar dimension, the distribution pattern is not the same. Eil76 has a uniform random distribution, pr76 has a clustered distribution and gr96 is randomly distributed but not in a uniform way. It is easier for GE to evolve architectures that perform better in a uniform-random distribution than in a clustered one. The mean of generations it takes for GE to reach the best individual also corroborates this fact.

The outcomes presented in Table 1 suggest that GE is able to learn ACO architectures that effectively solve the training instances. However, the most important topic in this research is to analyze the algorithmic structure of the evolved architectures. Two questions arise: did GE discover solutions which are replicas of standard ACO algorithms, e.g., EAS or ACS? If not, did the system converge to a particular solution or to a set of different solutions? The answer

Table 2. Frequency of appearance of the grammar components in best solutions evolved for each instance and average among all training instances. Evaporation and Reinforcement are not exclusive.

Components		eil76	pr76	gr96	Avg
Tau Init	Uniform Distribution	0.60	0.42	0.25	0.42
	Min	0.05	0.08	0.25	0.13
	Max	0.05	0.08	0.00	0.04
	Ant	0.05	0.00	0.25	0.10
	EAS	0.05	0.00	0.00	0.02
	RAS	0.00	0.17	0.00	0.06
	ACS	0.10	0.17	0.00	0.09
	MMAS	0.10	0.08	0.25	0.14
Selection	Roulette Selection	0.40	0.25	0.25	0.30
	Q Selection	0.60	0.75	0.75	0.70
	with Local Trails	0.30	0.17	0.00	0.16
Evaporation	Full Evaporation	0.65	0.50	1.00	0.72
	Partial Evaporation	0.50	0.33	0.25	0.36
Reinforcement	Full	0.40	0.08	0.00	0.16
	Partial	0.45	0.42	0.25	0.37
	Rank	0.50	0.17	0.00	0.22
	Elitist	0.90	0.83	1.00	0.91
Pheromone Limits		0.45	0.92	1.00	0.79
Restart		0.50	0.50	0.75	0.58

to the first question is obtained by inspecting the 90 best solutions generated by the GE engine (30 best solutions for each training instance). In this set there is not an exact copy of the standard algorithms, and only 5 individuals could be considered as similar to manually designed ACO approaches (3 for the ACS, 1 for RAS and 1 for EAS). There is then evidence that GE tends to converge to areas in the search space of ACO algorithms containing innovative strategies.

The answer to the second question can be partially obtained in Table 2, which contains the rate of appearance of the grammar components in the best solutions evolved (parameter values are not included). The rows contain all components that appear in the best solutions and are grouped by section (e.g., selection, evaporation). Some of the components are not exclusive (e.g., reinforcement), while others are (e.g., roulette vs. Q selection). The values are normalized to the interval between 0.0 and 1.0. There is a column with results obtained with each training instance and a fourth column *Avg* containing a weighted average of the frequencies by the number of *Best Hits*. An inspection of the results reveals that evolution did not converge to a unique solution. There are some variations between instances, but the general trend is to have a few alternative components appearing in the best solutions evolved by the GE in different runs. In any case, some components are frequently used to create promising strategies (e.g., elitist reinforcement and Q selection), while others are rarely considered (e.g., elitist pheromone initialization). This outcome shows that GE is able to identify a subset of components that increase the likelihood of creating effective strategies.

Table 3. Optimization results of selected best strategies, with 10000 iterations for 30 runs. Rows display the best solution found (B) and the mean best fitness (M) for every training instance. Bold values indicate the overall M best value.

		ei7612	ei7618	ei7621	pr7609	pr7626	pr7627	gr9603	gr9610	gr9622
att48	B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0024	**0.0003**	0.0072	0.0028	0.0069	0.0034	0.0031	0.0021	0.0020
eil51	B	0.0000	0.0000	0.0023	0.0000	0.0000	0.0023	0.0000	0.0000	0.0000
	M	0.0067	0.0023	0.0174	0.0046	0.0091	0.0186	**0.0020**	0.0045	0.0061
berlin52	B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0043	**0.0000**	0.0104	0.0055	0.0207	0.0108	0.0057	0.0053	0.0095
kroA100	B	0.0011	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0065	**0.0000**	0.0039	0.0037	0.0075	0.0044	0.0022	0.0068	0.0048
lin105	B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0043	**0.0001**	0.0014	0.0042	0.0081	0.0060	0.0016	0.0043	0.0027
gr137	B	0.0000	0.0017	0.0083	0.0000	0.0000	0.0000	0.0016	0.0000	0.0000
	M	0.0139	0.0071	0.0203	**0.0034**	0.0047	0.0053	0.0071	0.0053	0.0049
u159	B	0.0000	0.0000	0.0124	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0034	**0.0001**	0.0262	0.0039	0.0116	0.0066	0.0012	0.0021	0.0038
d198	B	0.0059	0.0067	0.0094	0.0035	0.0032	0.0001	0.0350	0.0035	0.0053
	M	0.0211	0.0103	0.0164	0.0125	0.0111	**0.0069**	0.0432	0.0082	0.0124
pr226	B	0.0000	0.0034	0.0098	0.0000	0.0000	0.0000	0.0001	0.0001	0.0000
	M	0.0042	0.0040	0.0291	**0.0022**	0.0040	0.0049	0.0053	0.0029	0.0030
lin318	B	0.0113	0.0139	0.0833	0.0086	0.0084	0.0052	0.1417	0.0079	0.0051
	M	0.0271	0.0188	0.1034	0.0199	0.0188	0.0159	0.1589	0.0186	**0.0148**

4.2 Validation of the Evolved Architectures

Results from the previous section show that the structure of the training instance influences the structure of the evolved architectures. It is therefore essential to validate the learned strategies in an optimization scenario. This will confirm, not only their effectiveness, but also their ability to scale and generalize. Specifically, we aim to: i) verify how evolved architectures behave in different TSP instances; ii) access how they perform in larger instances; iii) measure the absolute optimization performance of evolved architectures, by running them as normal ACO algorithms. To focus our analysis, we randomly selected 3 of the best evolved strategies with each training instance. Strategies are identified with an ID: the first two letters and the first two digits identify the training instance; the last two digits indicate the run (e.g., architecture ei7618 was trained with instance eil76 on run 18). Ten TSP instances were selected to access the optimization performance of the evolved architectures: att48, eil51, berlin52, kroA100, lin105, gr137, u159, d198, pr226 and lin318. The number of iterations is increased to 10000 to allow a correct optimization period for larger TSP instances.

Table 3 contains the optimization results of the selected strategies on the 10 instances (one strategy in each column). For every test instance, the table displays two rows: one with the best solutions found (B) and another with the mean best fitness (M). Bold values highlight the best M value among all evolved strategies. In general, results show that evolved strategies perform well across

Table 4. Statistical differences between architectures by applying the Wilcoxon rank sum test ($\alpha = 0.01$). A capital B highlights the strategy with the best average fitness, while a lower b indicates an architecture with an equivalent performance.

	ei7612	ei7618	ei7621	pr7609	pr7626	pr7627	gr9603	gr9610	gr9622
att48		B							
eil51		b					B		
berlin52	b	B		b					
kroA100		B							
lin105		B							
gr137				B	b	b		b	b
u159		B						b	
d198						B		b	
pr226	b			B	b			b	b
lin318						b			B

the instances, and thus, are able to generalize. In most cases, the automatically designed algorithms found the optimal solution. The exceptions are instances d198 and lin318, where the shortest tour was never found. Still, the normalized distances to the optimum are small (e.g., pr7627 has distances 0.0001 and 0.0052 in d198 and lin318, respectively). The mean best fitness values confirm that evolved strategies are robust and scale well. Instances range between 48 and 318 cities, but the M value rarely exceeds 0.02 and it does not grow excessively with the size of the instances. A closer inspection of the results reveals that, despite the overall good behavior, the performance of the evolved designs is not identical. If we recall Table 2, this variation was likely to happen. There are differences in the composition of strategies evolved with a specific instance and there are differences between strategies evolved with distinct training examples. Then, it is expectable that there are performance variations when a subset of promising strategies is selected to an optimization task.

To confirm that performance variations are not the result of random events, we applied a statistical test to the data of Table 3. In concrete, for each test instance we compared the mean best fitness of the best evolved architecture with the results obtained by each of the other strategies. Table 4 contains the results of applying the Wilcoxon rank sum test with continuity correction and confidence level $\alpha = 0.01$. The capital B highlights the best architecture for a given testing instance (e.g., gr9603 for eil51). A lower b indicates that no statistically significant difference was found between the performance of two strategies (the one in the column where the b appears and the best strategy for the instance in that line); e.g., ei7618 is equivalent to gr9603 in eil51. Finally, the white spaces indicate that significant differences were found. A combined analysis of tables 3 and 4 confirms that ei7618 has the overall best performance. It obtains the lowest mean best fitness in 5 out of 10 instances and is equivalent to the best strategy in a sixth case. However, results also show that the performance of ei7618 deteriorates as the instances grow in size, suggesting that it might not scale well. On the other extreme, ei7621 exhibits a poor behavior showing that the training instance is not the only key issue to learn effective optimization architectures. Finally, the behavior of strategy gr9610 is worth noting. It never achieves the absolute best mean performance, but, in 4 instances, it obtains results statistically equivalent to the best. In the upper-half of listing 1, we present the Lisp code for the strategy ei7618 (due to space constraints, we cannot include the code from other evolved strategies).

Listing 1. Evolved architecture ei7618 and the hand-adjusted GE-best.

```
(ACO-ei7618
   (INIT (INIT-PHEROMONE-MATRIX (UNIFORM-TRAIL 1.e-5 1.0))
         (INIT-CHOICE-INFO-MATRIX 1 3))
   (REPEAT-UNTIL-TERMINATION
      (FOREACH-ANT MAKE-SOLUTION-WITH (ROULETTE-SELECTION))
      (UPDATE-TRAILS
         (DO-EVAPORATION
            (FULL-EVAPORATE 0.25)
         (DO-REINFORCE
            (RANK-REINFORCE (RANK-ANT 10))
            (ELITIST-REINFORCE 1))))
      (DAEMON-ACTIONS
         (RESTART-CHECK))))

(ACO-GE-best
   (INIT (INIT-PHEROMONE-MATRIX (UNIFORM-TRAIL 1.e-5 1.0))
         (INIT-CHOICE-INFO-MATRIX 1 3))
   (REPEAT-UNTIL-TERMINATION
      (FOREACH-ANT MAKE-SOLUTION-WITH (ROULETTE-SELECTION))
      (UPDATE-TRAILS
         (DO-EVAPORATION
            (FULL-EVAPORATE 0.45))
         (DO-REINFORCE
             (PARTIAL-REINFORCE (ALL-TIME-BEST))
             (RANK-REINFORCE (RANK-ANT 10))
             (ELITIST-REINFORCE 1)))
      (DAEMON-ACTIONS
         (UPDATE-PHEROMONE-LIMITS (MMAS-UPDATE-MIN)
                                  (MMAS-UPDATE-MAX 0.01))
         (RESTART-CHECK))))
```

4.3 Comparison with Standard ACO Algorithms

To assess the absolute optimization performance of the learned architectures, we compare the results obtained by the two best evolved strategies (ei7618 and gr9610) with those achieved by the most common ACO algorithms: AS, EAS, ACS and MMAS. We include in this study two hand-adjusted architectures, based on the best strategies obtained in the learning process: GE-avg is the strategy obtained by selecting the most frequent components identified in column *Avg* from Table 2, whereas GE-best is a hybrid of the two best evolved architectures previously mentioned (see the bottom-half of listing 1 for the Lisp code of GE-best). The standard algorithms and the hand-adjusted strategies are applied to the selected 10 test instances in same conditions described in the previous section. The parameters settings for the standard approaches follow the recommendations from [1]. The exceptions are: colony size (we keep the 10% rule); $q0$ is set to 0.7; α and β are set to 1 and 2.

The Wilcoxon test was applied to perform a statistical comparison of the mean best fitness achieved by the different methods. Results are presented in

Table 5. Statistical differences between selected evolved architectures, hand-adjusted architectures and standard algorithms, by applying the Wilcoxon rank sum test ($\alpha = 0.01$). A capital B highlights the strategy with the best average fitness, while a lower b indicates an architecture with an equivalent performance.

	ei7618	gr9610	GE-avg	GE-best	AS	EAS	RAS	ACS	MMAS
att48	B								
eil51	b			B					
berlin52	B			B			B		b
kroA100	B								
lin105	b			B			b		
gr137				B			b		b
u159	b			B			b		
d198		B							
pr226		b		B			b		
lin318	b	B	b				b		

Table 5, where the symbols B and b should be interpreted as before. Overall, learned architectures and hand-adjusted strategies inspired by the evolved ones are competitive with standard variants. Nearly all best results appear in the first 4 columns, which correspond to fully evolved or fine-tuned evolved strategies. There is just a single B in the RAS column, identifying a tie with ei7618 and GE-best. Individually, the behavior of ei7618 is remarkable, since it achieves the best or an equivalent to the best performance in 7 instances. GE-best is also effective, as it achieves the absolute best performance in 6 instances. The results obtained by gr9610 are worth noting, as it achieves the best or an equivalent to the best performance on the 3 largest test instances. These outcomes suggest that gr9610 has a good scaling ability. Finally, when compared to the other evolved strategies, the results of GE-avg are poor. This confirms that designing architectures which are a linear combination of components found in successful strategies is not a guarantee of success.

5 Conclusions

We proposed a GE framework to accomplish the automatic evolution of ACO algorithms. The grammar adopted is mostly formed by components appearing in human-designed variants, although the interpretation of individuals allows for the emergence of alternative algorithmic strategies. To the best of our knowledge, this is the first work that evolves a full-fledged ACO algorithm.

Results reveal that evolution is able to discover original architectures, different from existing ACO algorithms. Best evolved strategies exhibit a good generalization capability and are competitive with human-designed variants. The system uses a constrained grammar, which undermines the possibility of evolving strategies that strongly deviate from standard ACO structures. Nevertheless, GE discovers effective novel solutions and does not converge to the standard algorithms. This suggests that the human-designed variants regularly adopted in the optimization of the TSP might be local optima in the search space of ACO algorithms. The outcomes from the last section reveal that the hybrid architecture based on the two best evolved strategies is particularly effective. This shows that GE can also act as a supplier of promising strategies that can be later combined and/or fine-tuned.

The study presented here raises important research questions. There are several design options concerning the training phase that impact the likelihood of discovering effective and scalable strategies. In the near future we will investigate the influence played by the training instances properties (e.g., size, structure) on the discovery of good ACO algorithms. This will help us to gain insight on how to select the most favorable training environment. Also, we will address the issue of selecting the most promising strategies from the pool of best solutions. Another important research topic is the development of less constrained grammars, which might allow the evolution of ACO architectures with a higher degree of innovation. Finally, testing the approach with other problems and doing cross-problem validation will be significant steps in the effort of development evolutionary-based Ant Systems.

References

1. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
2. Eiben, A., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. Evolutionary Computation, IEEE Transactions on **3** (1999) 124 –141
3. Tavares, J., Pereira, F.B.: Evolving strategies for updating pheromone trails: A case study with the tsp. In: PPSN XI Proceedings. Volume 6239 of Lecture Notes in Computer Science., Springer (2010) 523–532
4. Tavares, J., Pereira, F.B.: Designing pheromone update strategies with strongly typed genetic programming. In: EuroGP 2011 Proceedings. Lecture Notes in Computer Science, Springer (2011)
5. López-Ibáñez, M., Stützle, T.: Automatic configuration of multi-objective ACO algorithms. In: ANTS 2010 Proc. Volume 6234 of LNCS., Springer (2010) 95–106
6. O'Neill, M., Ryan, C.: Grammatical Evolution. Springer-Verlag (2003)
7. Pappa, G.L., Freitas, A.A.: Automatically Evolving Data Mining Algorithms. Volume XIII of Natural Computing Series. Springer (2010)
8. Burke, E.K., Hyde, M.R., Kendall, G.: Grammatical evolution of local search heuristics. IEEE Transactions on Evolutionary Computation (2011)
9. Botee, H., Bonabeau, E.: Evolving ant colony optimization. Advances in Complex Systems **1** (1998) 149–159
10. White, T., Pagurek, B., Oppacher, F.: ASGA: Improving the ant system by integration with genetic algorithms. In: Proceedings of the 3rd Genetic Programming Conference, Morgan Kaufmann (1998) 610–617
11. Poli, R., Langdon, W.B., Holland, O.: Extending particle swarm optimisation via genetic programming. In: EuroGP 2005 Proceedings. (2005) 291–300
12. Diosan, L., Oltean, M.: Evolving the structure of the particle swarm optimization algorithms. In: EvoCOP 2006 Proceedings. (2006) 25–36
13. Runka, A.: Evolving an edge selection formula for ant colony optimization. In: GECCO 2009 Proceedings. (2009) 1075–1082
14. Tavares, J., Pereira, F.B.: Towards the development of self-ant systems. In: GECCO 2011 Proceedings, ACM (2011)