Towards the Development of Self-Ant Systems

Jorge Tavares
CISUC, Department of Informatics Engineering
University of Coimbra
3030 Coimbra, Portugal
jorge.tavares@ieee.org

Francisco B. Pereira
CISUC, Department of Informatics Engineering
University of Coimbra
ISEC, Polytechnic Institute of Coimbra
3030 Coimbra, Portugal
xico@dei.uc.pt

ABSTRACT

We propose a computational framework for the self-generation of components used by an Ant Colony Optimization algorithm. The approach relies on Strongly Typed Genetic Programming to automatically seek for effective update pheromone strategies. Best evolved strategies are then inserted in an Ant Colony Algorithm used to find good quality solutions for the Quadratic Assignment Problem. Results reveal that evolved update rules are competitive with human designed variants and can be effectively reused on different instances of the same problem. Moreover, we investigate the possibility of evolving general strategies that can be used across different optimization problems.

Categories and Subject Descriptors

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

General Terms

Algorithms, Design

Keywords

Self-Ant systems, pheromone update methods, self-generation

1. INTRODUCTION

Ant colony optimization (ACO) algorithms are a powerful metaheuristic for global optimization. They were originally proposed by Marco Dorigo and, as its name suggests, explore the search space in a way loosely inspired by pheromone-based strategies of ant foraging [1]. In simple terms, a set of agents (i.e., ants) iteratively build solutions for a given problem. Construction is guided by static problem-specific information and by dynamic feedback from promising solutions already discovered. Feedback is modeled as pheromone information that ants deposit on components used to create a solution. The amount of pheromone on each component is modified during the run and is proportional to the quality of the solutions in which it appears. This procedure implements a mechanism for indirect communication and allows ants to cooperatively solve a problem. Actually, there are several variants of ACO algorithms with differences, e.g., in the way pheromone levels are updated throughout the optimization. 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 [2]. 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 and analyze a computational model that implements the automatic programming of ACO components. Research described here is a step towards the development of a Self-Ant System that is able to generate on-the-fly effective ACO frameworks to specific problems. This will remove the need to carry out cumbersome manual adaptations of algorithms and contribute to the appearance of more robust ACO architectures. In concrete, we describe an automated process to generate pheromone update strategies. This process can be modeled as a search problem, where one aims to discover a plan of action for a key step of a global optimization algorithm. Genetic programming (GP) [3] is adopted as the search method and it will seek for promising update strategies that can be used by an ACO algorithm when solving an optimization problem. There are two key issues to address when developing this type of computational prototype. First, one must select the composition of the function and terminal sets granted to the GP algorithm. This choice defines the primitives that can be used, which, in turn, help to induce a bias on the kind of update strategies that can be generated. Also, the existence of specific basic components may allow a proper tuning of some control parameters (e.g., the evaporation rate). The evaluation of individuals generated by the GP algorithm is another crucial step, as it must reflect how well the encoded strategy helps to enhance the effectiveness of ACO algorithms. In our approach, the fitness assignment is accomplished by inserting the update rule in an ACO method and verifying how it behaves in a specific optimization situation.

The framework studied in this paper was proposed in a recent work [4]. There, a strongly-typed GP variant (STGP) was applied to evolve pheromone update rules that helped ACO algorithms to discover good solutions for the Traveling Salesman Problem (TSP). Results obtained showed that the evolved strategies outperformed Max-Min Ant System (MMAS), which is one of the most competitive ACO algo-

GECCO'11, July 12–16, 2011, Dublin, Ireland.

rithms [5, 1]. Here we will apply the computational prototype to the Quadratic Assignment Problem (QAP), another optimization problem for which ACO algorithms are considered state-of-the-art methods [1]. Our research study has two main goals: first, we aim to verify if STGP is also able to evolve effective update strategies for the QAP, thereby gaining confidence that the proposed framework can be applied to different problems. However, our foremost goal is to investigate the reusability of generated strategies. When dealing with a single problem, we will provide evidence that an update rule evolved for a single QAP instance maintains its effectiveness when applied to other instances. Finally, we study inter-problem generalization, where strategies generated for a specific situation are applied in the optimization of a different problem. Analysis of the results helps to gain insight on how evolved strategies can be reused on a different optimization situation.

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, such as the QAP or routing situations [1]. 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 static heuristic information and the 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 premature convergence, 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 [5]. 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:

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

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 TRAILS UPDATE STRATEGIES

The framework we use to evolve pheromone update strategies was proposed in [4]. It contains two components: a STGP engine and an AS algorithm. The main task of GP is to generate 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 QAP 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 GP 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 [6]. 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 [7, 4]. This is what happens with the problem addressed in this paper, as the GP algorithm must deal with, e.g., numeric values or sets of ants. We adopt the strong typing principle and hence assure that GP only generates trees satisfying type constraints. For details about the initialization procedure and the application of genetic operators, consult [6].

The definition of the function and terminal sets used by STGP is a key decision, since they determine which components can be used in the design of strategies. We keep the definition of the sets as simple as possible and consider two different, although similar, possibilities. The first proposal, identified as set 1, contains the following components:

- *(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.
- *(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.
- *(rank number)*: This function sorts the ants by decreasing quality and returns an array (type ant) of size *number* (type integer) with the best ants.
- *(best-ant)*, *(rho)*: Return respectively the best ant found so far (type ant) or a fixed evaporation rate (type real).

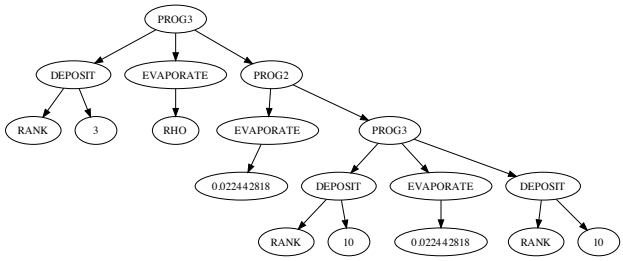

Figure 1: An example of a strategy evolved by our STGP system. It performs three evaporation actions interleaved with different deposit actions.

- *(integer)* and *(real)*: Ephemeral constants.

The second proposal (set 2) is obtained from set 1 just by making a simple modification. The function *rank* is transformed into a terminal that always returns the 10% best ants. The second set simplifies the task of obtaining a subset of promising ants, as this information is readily available in a terminal. The first set must compose an appropriate subset of blocks to obtain this knowledge, but it has the ability to deal with subsets of different sizes. Differences in the performance of both sets, in what concerns the ability to generate successful strategies, will help us to gain insight into the key building blocks that must be granted to the GP. The components available in both sets allow the development of update strategies similar to those existing in current ACO variants, such as the AS, Elitist AS or MMAS. The analysis of experimental results will help to understand if the GP evolution converges to existing designs and parameterizations or, on the contrary, generates new and improved strategies for pheromone update.

In Figure 1 we can observe an example of an evolved strategy (using set 2) for the TSP from [4]. The encoded plan is different from existing human designs, but it is easy to understand. In the beginning, the 10% best ants deposit pheromone to reinforce their paths and afterwards a standard amount of pheromone is removed from the matrix. Then, the last actions of the strategy induce a very greedy behavior by promoting two large deposits of pheromone in promising trails, interleaved with light evaporations.

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 [8, 9] and multimeme strategies [10] deal with the development of the best combination of methods for a given optimization problem. The concept of hyper-heuristics identifies a framework composed by a search methodology that seeks for a combination of specific heuristics that can be successfully applied to a given problem [9]. The suitability of adopting GP as the meta search method inside a hyper-heuristic framework is discussed in [11], together with a comprehensive review of existing approaches. As for multimeme strategies, they consist of memetic algorithms that learn on-the-fly which local search component should be used. Diosan and Oltean recently proposed another interesting approach for the automatic adaptation of an optimization algorithm [12]. In their work, an evolutionary framework is used to generate a full-featured Evolutionary Algorithm (EA), which is then applied to numerical function optimization.

As for the Swarm Intelligence area, there are some reports describing the self-adaptation of parameter settings (see, e.g., [2, 13]). Additionally, a couple of approaches resemble the framework proposed in this paper. Poli et. al [14] 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 [15]. Finally, Runka [16] 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

In this section we focus our attention on the generalization ability of the computational framework. There is evidence from previous works that competitive update strategies can be evolved for the TSP (see [7, 4]). Here, we present a set of results obtained with the Quadratic Assignment Problem (QAP), which will enable us to verify if the approach can be applied to a different optimization problem. Moreover, we investigate how strategies evolved for a specific scenario generalize to a different optimization situation.

4.1 Problem Description

Many real-world optimization problems, e.g., scheduling and layout, can be formulated as QAPs. It is a NP-Hard problem and it is considered one of the hardest optimization problems since instances with size $n \geq 25$ cannot be exactly solved [17]. ACO algorithms are considered state-of-the-art optimization methods for this problem [1, 17].

The QAP can be described as the problem of assigning a set of facilities to a set of locations with given distances between the locations and given flows between the facilities. The goal is to assign facilities to locations, such that the sum of the products between flows and distances is minimized. Formally, given n facilities and n locations, two nxn matrices $A = (a_{ij})$ and $B = (b_{rs})$, where a_{ij} is the flow between facilities i and j and b_{rs} is the distance between locations r and s, one aims to minimize the following expression :

$$min_{\phi \in \Phi(n)} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\phi_i \phi_j} \quad (2)$$

where $\Phi(n)$ is the set of all assignments permutations of the integer set $\{1, \ldots, n\}$, and ϕ_i gives the location of unit i in the current solution.

4.2 Evolution of the Update Strategies

In the first set of experiments we aim to confirm the ability of STGP to discover update strategies for the QAP. A single instance is used to assign fitness to solutions generated by the GP algorithm. We present results from experiments performed with the two sets presented in section 3.1 and using QAP training instances of different size, to study how these choices affect performance. Six training instances from

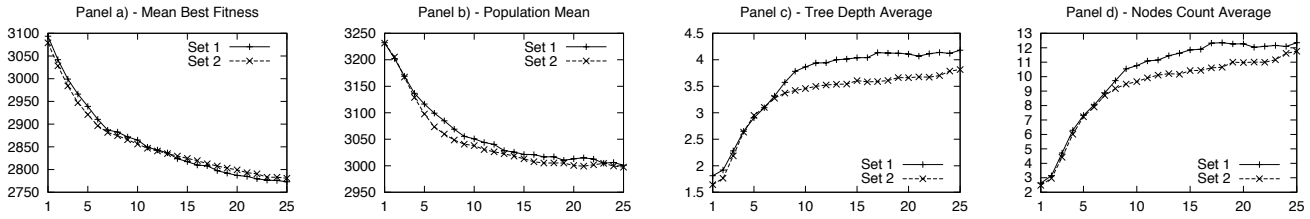

Figure 2: Evolution plots, on the nug20 instance, with the 2 sets with generations on the x axis and fitness values on the y axis. Panels: a) Mean Best Fitness; b) Population Fitness; c) Tree Depth; d) Nodes count.

Table 1: Results from GP evolution for 30 runs with 25 generations in six QAP instances.

Instances	Set	Best	Dist	MBF	Dev	Depth	Nodes
nug12	1	**578.00**	0.0	**593.17**	17.58	4.13	12.77
(578)	2	**578.00**	0.0	595.97	13.50	4.00	11.67
nug14	1	1028.00	1.4	**1075.53**	28.55	4.33	12.47
(1014)	2	**1024.00**	1.0	1100.07	41.12	3.37	9.43
nug15	1	**1152.00**	0.2	**1204.40**	42.10	4.20	12.80
(1150)	2	1166.00	1.4	1210.03	44.20	4.10	11.43
nug17	1	1840.00	6.3	**1910.00**	66.63	4.33	12.30
(1732)	2	**1839.33**	6.2	1912.76	60.82	3.80	10.90
nug18	1	**2024.00**	4.9	**2107.67**	49.27	4.83	15.63
(1930)	2	2054.00	6.4	2138.33	69.10	4.00	11.33
nug20	1	**2696.00**	4.9	**2773.10**	52.64	4.47	13.50
(2570)	2	**2696.00**	4.9	2780.91	92.55	4.07	12.67

the Nugent dataset of QAPLIB[1] were selected, with sizes $n = 12, 14, 15, 17, 18, 20$. For all tests, the GP settings are: Number of runs: 30; Number of generations: 25; Population size: 100; Initial tree depth: 2; 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 [1]: number of ants is the problem size, $\alpha = 1$, $\beta = 2$, $\rho = 0.5$. Our goal in this paper is to focus on the influence of pheromone update methods and not on the whole AS architecture. Therefore, we do not address the issue of using Local Search.

The main reason for choosing moderate size instances is that the evaluation of each strategy generated by GP requires the execution of an AS algorithm which can be computationally expensive. Two parameters define the optimization effort of the AS: the number of runs and the number of iterations per run. If we grant the AS a small optimization period to evaluate an STGP individual, then it might not be enough to correctly estimate the quality of an update strategy. However, a longer evaluation period can increase computational costs to insupportable levels. Preliminary experiments allowed us to conclude that a single AS run with 1000 iterations is enough to evaluate a GP individual. The fitness value sent back to the GP engine is given by the 15% best solutions found by the ants in the AS (consult [7] for a discussion of other evaluation configurations).

Table 1 contains the results of the STGP evolution. The first two columns identify, respectively, the instance and the component set used in the tests. The value in brackets below each instance name is the corresponding optimal solution. Column *Best* displays the best solution found by STGP, whereas column *MBF* is the Mean Best Fitness, i.e., the mean of the best solutions found in the 30 runs. Columns *Dist* and *Dev* contain the distance of the best solution to the optimum and the standard deviation of the MBF. The last two columns (*Depth* and *Nodes*) show the mean depth and the mean number of nodes of the best solutions.

An overview of the results reveals that, for $n = 12, 14, 15$, the system discovers individuals that help the AS algorithm to find solutions with fitness values close to the optimum. Then, as the size of the instances increases, the performance of STGP decreases. For $n > 15$, the distance to the optimal solution is usually above 5% and the MFB standard deviation is also higher. This is a result of the hardness of the training instances. Although they are still moderate size problems, the increase in difficulty is not linear. Also, these results are probably amplified by the sharp experimental conditions (low number of generations performed by the STGP and small maximum tree depth).

Despite the decrease in absolute performance, the plots in Figure 2 clearly show the usefulness of the STGP exploration. The panels in this figure summarize the evolutionary behavior of the framework for the nug20 instance: panels a) and b) display, for the two sets, the evolution of the MBF and of the mean fitness of the population. A brief overview confirms that the STGP engine is gradually discovering enhanced update strategies for the QAP. For completeness, panels c) and d) display the evolution of the tree depth and of the average number of nodes.

Results obtained are not fully conclusive in what concerns the efficacy of the two component sets used in the experiments. There is a slight trend for set 1 to obtain better MBF values, but differences are small. Further experiments are needed to confirm if allowing the GP to learn the ideal proportion of ants for the rank function indeed leads to better strategies. On any case, this trend contrasts with our previous work with the TSP [4], where a fixed proportion of high quality ants simplified the generation of effective strategies. These results support the idea that the evolution of problem specific pheromone update rules might allow the discovery of more effective strategies.

4.3 Validation and Comparison with the Max-Min Ant System

To confirm the effectiveness of the best evolved solutions we describe a set of additional experiments. Specifically, we aim to: i) verify if update strategies can be effectively reused in QAP instances different from the one selected for

[1]http://www.opt.math.tu-graz.ac.at/qaplib/

Table 2: Comparison of the best STGP Trees with MMAS, with 10000 iterations for 30 runs.

Update	nug12		nug14		nug15		nug17		nug18		nug20	
Strategy	Best	MBF	Best	MBF	Best	MBF	Best	MBF	Best	MBF	Best	MBF
MMAS	590	611.2	1076	1095.60	1222	1265.80	1884	1922.33	2118	2154.13	2846	2904.13
Tree 12124	**578**	597.00	1028	1052.80	1166	1197.33	1782	1838.13	**1968**	**2049.20**	2680	**2751.60**
(imp %)	**2.03**	2.32	4.46	3.91	4.58	5.41	5.41	4.38	**7.08**	**4.87**	5.83	**5.25**
Tree 12214	586	599.20	1036	1074.00	1190	1213.13	1836	1865.67	2066	2092.33	2738	2806.20
(imp %)	0.68	1.96	3.72	1.97	2.62	4.16	2.55	2.95	2.46	2.87	3.79	3.37
Tree 14123	**578**	**595.20**	1028	1063.80	1186	1220.87	1836	1873.00	2020	2098.07	2732	2822.67
(imp %)	**2.03**	**2.62**	4.46	2.90	2.95	3.55	2.55	2.57	4.63	2.60	4.01	2.81
Tree 14223	586	609.13	1064	1090.87	1186	1221.00	1794	1858.00	2034	2087.53	2754	2815.40
(imp %)	0.68	0.34	1.12	0.43	2.95	3.54	4.78	3.35	3.97	3.09	3.23	3.06
Tree 15111	582	606.53	1056	1091.13	1202	1248.07	1854	1912.87	2102	2134.13	2748	2871.87
(imp %)	1.36	0.76	1.86	0.41	1.64	1.40	1.59	0.49	0.76	0.93	3.44	1.11
Tree 15221	**578**	595.80	**1018**	1069.20	**1150**	**1191.67**	**1752**	1830.87	2012	2059.20	**2656**	2755.40
(imp %)	**2.03**	2.52	**5.39**	2.41	**5.89**	**5.86**	**7.01**	4.76	5.00	4.41	**6.68**	5.12
Tree 17108	**578**	595.47	1028	1061.13	1164	1202.67	1814	1863.93	2046	2089.73	2748	2813.80
(imp %)	**2.03**	2.60	4.65	4.56	4.58	5.51	6.37	4.72	5.85	4.21	6.04	4.94
Tree 17225	**578**	597.60	1032	1071.27	1170	1220.33	1834	1878.40	2032	2096.73	2734	2815.80
(imp %)	**2.03**	2.23	4.09	2.22	4.26	3.59	2.65	2.29	4.06	2.66	3.94	3.04
Tree 18124	**578**	595.33	1026	**1045.67**	1166	1196.00	1764	1831.53	1994	2063.47	2674	2760.60
(imp %)	**2.03**	2.60	4.65	**4.56**	4.58	5.51	6.37	4.72	5.85	4.21	6.04	4.94
Tree 18211	**578**	597.93	1022	1061.67	1184	1222.13	1806	1857.60	1996	2070.67	2714	2786.47
(imp %)	**2.03**	2.17	5.02	3.10	3.11	3.45	4.14	3.37	5.76	3.87	4.64	4.05
Tree 20128	586	596.40	1020	1046.00	1160	1195.07	1768	**1825.60**	1986	2055.47	2692	2751.73
(imp %)	0.68	2.42	5.20	4.53	5.07	5.59	6.16	**5.03**	6.23	4.58	5.41	5.25
Tree 20209	**578**	598.47	1040	1078.47	1220	1247.33	1832	1885.33	2028	2115.60	2736	2836.80
(imp %)	**2.03**	2.08	3.35	1.56	0.16	1.46	2.76	1.92	4.25	1.79	3.87	2.32
QAP Trees Avg	580.33	598.67	1033.17	1067.17	1178.67	1214.63	1806.00	1860.08	2023.67	2084.34	2717.17	2799.03
(imp %)	1.64	2.05	4.00	2.71	3.53	4.09	4.36	3.38	4.66	3.34	4.74	3.77

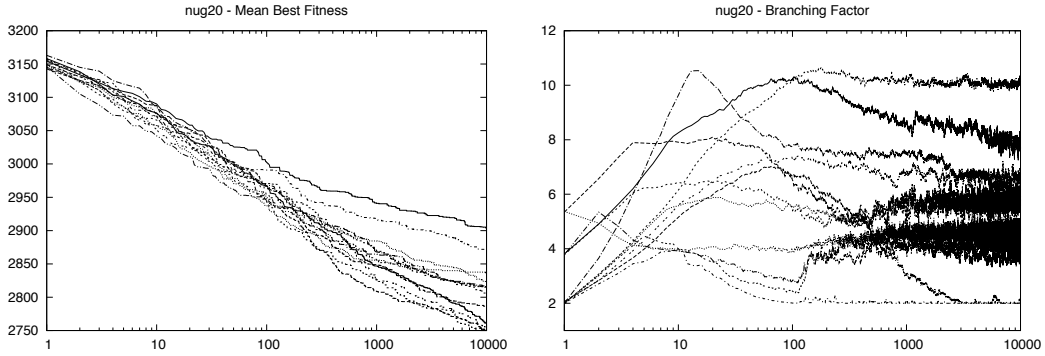

Figure 3: Evolution of MBF and branching factor for the STGP trees and MMAS for 10000 iterations (x axis) on the nug20 instance. Plots are representative for all instances.

training; ii) assess the absolute optimization performance of evolved strategies, by comparing them with MMAS.

To focus our analysis, 12 trees were selected: the best tree for each set and training instance combination. Trees are identified with a numerical ID: the first two digits refer to the QAP training instance and the third identifies the component set used. The final digits indicate the run in which the tree was discovered. 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 training process did not use these mechanisms, this is an additional robustness test for the evolved strategies. The settings used before are maintained, with just two exceptions: ρ is set to 0.02 (the recommended value for MMAS) and the number of iterations is increased to 10000 to allow for a fair optimization period.

Table 2 contains the results (Best solution found and MBF) obtained by the 12 selected update strategies on all the QAP instances. The first line contains the results obtained by the standard MMAS variant. Then, every pair of rows displays the outcomes of the selected trees. For a better understanding of the results, each update strategy contains a row (imp %) identifying the percentage of improvement over MMAS. A positive value indicates that the evolved strategy performs better and a negative value otherwise. For completeness, the last two rows of the table exhibit average values of all the evolved strategies.

In general, results show that evolved strategies perform well across the instances and, thus, are able to generalize. This is an important conclusion, as it confirms that strategies trained on a specific situation can be reused on different instances of the same problem. Another significant observation is that every single evolved strategy, in every instance, outperforms the human developed strategy, MMAS, in terms of best solutions found and MBF. On average, the improvements obtained by the evolved strategies range from 2.05%

Table 3: STGP Trees evolved for the TSP applied in the QAP instances, with 10000 iterations for 30 runs.

Update Strategy	nug12		nug14		nug15		nug17		nug18		nug20	
	Best	MBF	Best	MBF	Best	MBF	Best	MBF	Best	MBF	Best	MBF
MMAS	590	611.2	1076	1095.60	1222	1265.80	1884	1922.33	2118	2154.13	2846	2904.13
Tree 222	586	601.67	1024	1079.73	1182	1232.93	1838	1885.93	2044	2114.27	2788	2851.07
(imp %)	0.68	1.56	4.83	1.45	3.27	2.60	2.44	1.89	3.49	1.85	2.04	1.83
Tree 224	**578**	603.67	1056	1074.00	1182	1229.80	**1816**	1881.60	2054	**2100.87**	2782	2824.00
(imp %)	**2.03**	1.23	1.86	1.97	3.27	2.84	**3.61**	2.12	3.02	**2.47**	2.25	2.76
Tree 226	598	611.13	1048	1083.53	1222	1247.80	1852	1894.87	2068	2115.80	2786	2852.87
(imp %)	-1.36	0.01	2.60	1.10	0.00	1.42	1.70	1.43	2.36	1.78	2.11	1.77
Tree 230	586	603.73	1056	1079.33	1210	1238.80	1830	**1879.93**	2072	2107.67	2804	2839.07
(imp %)	0.68	1.22	1.86	1.48	0.98	2.13	2.87	**2.21**	2.17	2.16	1.48	2.24
Tree 31	**578**	598.40	1034	1067.80	1182	1238.87	1842	1892.73	2058	2121.00	2794	2850.20
(imp %)	**2.03**	2.09	3.90	2.54	3.27	2.13	2.23	1.54	2.83	1.54	1.83	1.86
Tree 321	**578**	600.60	1046	1079.93	1198	1254.20	1864	1906.47	2090	2131.80	2746	2864.60
(imp %)	**2.03**	1.73	2.79	1.43	1.96	0.92	1.06	0.83	1.32	1.04	3.51	1.36
Tree 326	586	607.53	1024	**1057.40**	1176	**1213.47**	1836	1880.60	2056	2109.20	2748	2839.47
(imp %)	0.68	0.60	4.83	**3.49**	3.76	**4.13**	2.55	2.17	2.93	2.09	3.44	2.23
Tree 327	**578**	609.20	1044	1071.20	**1166**	1223.93	1844	1881.00	2080	2114.93	2764	**2823.60**
(imp %)	**2.03**	0.33	2.97	2.23	**4.58**	3.31	2.12	2.15	1.79	1.82	2.88	**2.77**
Tree 329	582	**597.07**	1022	1058.60	1194	1241.93	1830	1889.80	2066	2125.27	2786	2856.80
(imp %)	1.36	**2.31**	5.02	3.38	2.29	1.89	2.87	1.69	2.46	1.34	2.11	1.63
Tree 330	**578**	598.73	1034	1060.13	1214	1253.13	1838	1904.33	**2006**	2124.07	**2744**	2849.00
(imp %)	**2.03**	2.04	3.90	3.24	0.65	1.00	2.44	0.94	**5.29**	1.40	**3.58**	1.90
TSP Trees Avg	582.8	603.17	1038.8	1071.17	1192.6	1237.49	1839	1889.73	2059.4	2116.49	2774.2	2845.07
(imp %)	1.22	1.31	3.46	2.23	2.41	2.24	2.39	1.70	2.77	1.75	2.52	2.03
QAP Trees Avg	580.33	598.67	1033.17	1067.17	1178.67	1214.63	1806.00	1860.08	2023.67	2084.34	2717.17	2799.03
(imp %)	1.64	2.05	4.00	2.71	3.53	4.09	4.36	3.38	4.66	3.34	4.74	3.77

to 4.09% for MBF, and from 1.64% to 4.74% in terms of best solutions found. These values increase when we look at the best results obtained by the best trees (bold values). For example, in instances nug17 and nug18, the improvement rate for the best solution is around 7%. Overall, trees evolved with set 1 tend to obtain better results than trees generated with set 2, thereby confirming our previous finding. Anyway, differences in performance are minimal and further research is mandatory to clarify this issue.

Figure 3 displays the optimization behavior of the strategies studied in this section. Results were obtained with nug20, but the same trend is visible for other instances. Panel a) shows the evolution of the MBF for the 12 trees and for standard MMAS (the black straight line). In general, the plot lines from the evolved strategies exhibit a steady decrease in the MBF until the end of the run. As for MMAS, it is already the worst strategy in iteration 10 and it shows signs of stagnation around iteration 1000.

The evolution of the branching factor (with $\lambda = 0.05$), a measure to determine the convergence of the pheromone matrix, is shown in panel b). Some strategies follow the pattern of standard MMAS, exhibiting initial low diversity, followed by a peak and subsequent stabilization. On the contrary, a few other trees show different behaviors, either maintaining a high diversity level throughout the run or never being able to raise the branching factor. The impact of restarts is also evident in some of the strategies, while for others is not so visible.

4.4 Cross-problem Validation

Results presented in the previous section and in [4] confirm that the proposed framework is able to evolve update strategies for specific problems, which are competitive with state-of-the-art ACO variants. It is, however, important to investigate if strategies evolved for a given scenario can be reused in a different problem. In concrete, we will verify how update rules evolved for the TSP behave with the QAP and how strategies generated for the QAP perform on the TSP.

This cross-problem validation raises interesting research questions. The most relevant is to assess the ability of STGP to generate strategies that, despite being evolved in a specific environment, generalize well to other situations. We already provided evidence of reusability for other instances of the same problem, but tests described in this section deal with a more general framework. The outcomes of the experiments will also help to gain insight into the structural similarity of strategies evolved for different problems. The STGP engine used the same basic components to build update rules for the TSP and QAP. Cross testing will unravel the eventual existence of problem-specific arrangements that maximize adaptability to a given situation.

We start to answer these questions by selecting the best 10 trees evolved for the TSP [4] and apply them to the QAP. We maintain the same experimental conditions adopted in the previous sections to allow for a fair comparison. Table 3 contains the results obtained by these 10 trees in the 6 QAP instances. For easiness of comparison, we repeat the outcomes obtained by MMAS and the average results of the best trees specifically evolved for the QAP (last line). Two noteworthy conclusions can be gathered from the information contained in table 3. Strategies evolved for the TSP are competitive with MMAS when solving QAP instances. With just a few exceptions, results are better than those achieved by this ACO variant: the average improvement rate for the best solutions ranges from 1.22% to 3.46% with a mean of 2.46%, whereas for the MBF, the improvement rate averages 1.88%, within an interval from 1.31% to 2.24%.

The second important conclusion is that strategies specifically generated for the QAP obtain better results than those achieved by trees evolved in a TSP environment. A brief perusal of the last rows containing the average results of both sets of trees reveals that evolved QAP strategies achieve higher improvements over MMAS in every single instance

Table 4: STGP Trees evolved with the QAP applied in the TSP instances, with 10000 iterations for 30 runs.

Update Strategy	eil51		kroA100		d198		lin318		pcb442	
	Best	MBF	Best	MBF	Best	MBF	Best	MBF	Best	MBF
MMAS	426	427.23	21431	21553.97	16141	16276.23	45243	45912.77	58211	60009.20
Tree 12124	426	429.27	21442	22189.83	16403	17509.07	**45142**	48867.87	63007	67206.63
(imp %)	0.00	-0.48	-0.05	-2.95	-1.62	-7.57	**0.22**	-6.44	-8.24	-11.99
Tree 12214	426	427.73	**21427**	22106.23	16896	17136.30	47031	48094.63	61048	62191.30
(imp %)	0.00	-0.12	**0.02**	-2.56	-4.68	-5.28	-3.95	-4.75	-4.87	-3.64
Tree 14123	426	**426.83**	21518	21882.20	16914	17566.63	47897	52714.93	60641	62698.80
(imp %)	0.00	**0.09**	-0.41	-1.52	-4.79	-7.93	-5.87	-14.82	-5.86	-5.18
Tree 14223	426	428.13	21491	21838.17	16824	17132.57	47191	48618.37	61621	63117.83
(imp %)	0.00	-0.21	-0.28	-1.32	-4.23	-5.26	-4.31	-5.89	-5.86	-5.18
Tree 15111	430	450.77	22439	23645.17	17032	17710.63	49019	51739.43	**58023**	61973.27
(imp %)	-0.94	-5.51	-4.70	-9.70	-5.52	-8.81	-8.35	-12.69	**0.32**	-3.27
Tree 15221	426	**427.17**	21479	21827.17	16765	17064.97	**43578**	48126.67	60950	62874.13
(imp %)	0.00	**0.02**	-0.22	-1.27	-3.87	-4.85	**3.68**	-4.82	-4.71	-4.77
Tree 17108	426	427.73	21585	21997.83	16955	17709.50	48697	51714.73	59013	61385.43
(imp %)	0.00	-0.12	-0.72	-2.06	-5.04	-8.81	-7.63	-12.64	-1.38	-2.29
Tree 17225	426	**426.77**	21460	21890.47	16338	16993.60	45619	47279.63	**56208**	63159.17
(imp %)	0.00	**0.11**	-0.14	-1.56	-1.22	-4.41	-0.83	-2.98	**3.44**	-5.25
Tree 18124	427	430.00	21483	22112.33	16714	17528.17	**44909**	51534.40	65137	67115.70
(imp %)	-0.23	-0.65	-0.24	-2.59	-3.55	-7.69	**0.74**	-12.24	-11.90	-11.84
Tree 18211	426	**426.90**	**21429**	21706.37	16972	17219.90	**44779**	48871.10	60974	62540.77
(imp %)	0.00	**0.08**	**0.01**	-0.71	-5.15	-5.80	**1.03**	-6.44	-4.75	-4.22
Tree 20128	426	429.53	**21332**	21879.20	16837	17235.90	46974	49304.60	**55507**	**59191.97**
(imp %)	0.00	-0.54	**0.46**	-1.51	-4.31	-5.90	-3.83	-7.39	**4.65**	**1.36**
Tree 20209	426	**426.80**	21566	21895.40	16815	17133.07	**44332**	47079.63	61436	63579.80
(imp %)	0.00	**0.10**	-0.63	-1.58	-4.18	-5.26	**2.01**	-2.54	-5.54	-5.95
QAP Trees Avg	426.42	429.80	21554.25	22080.86	16788.75	17328.36	46264.00	49495.50	60378.75	63121.15
(imp %)	-0.10	-0.60	-0.58	-2.44	-4.01	-6.46	-2.26	-7.80	-3.72	-5.19
TSP Trees Avg	426.00	428.48	21282.00	21368.97	15985.60	16110.34	42516.10	43000.90	54104.60	55079.25
(imp %)	0.00	-0.29	0.70	0.86	0.96	1.02	6.03	6.34	7.05	8.22

(both for the best solutions found and the MBF). A detailed analysis of the best results in tables 2 and 3 leads to the same conclusion. This result confirms that the framework evolves strategies that are specific to the training optimization situation. There are, nevertheless, signs that the evolved solutions can be reasonably generalized to other problems.

In the reverse experiment, the 12 best trees evolved for the QAP are applied to the TSP. The five instances from the TSPLIB[2], used in [4], are selected to assess the optimization performance of the evolved methods: eil51, kroA100, d198, lin318 and pcb442. Results are presented in Table 4. Once again, we include the outcomes from MMAS and the average results of the best trees specifically evolved for the TSP (taken from the above mentioned reference). A quick overview of the table immediately reveals that QAP trees are not successful in the TSP, as average results are always worse than those obtained by MMAS (see line QAP Trees Avg). This outcome contrasts with the performance of trees specifically generated for the TSP. As it can be confirmed in the last row of the table, evolved strategies clearly outperform MMAS (for more details consult [4]). In short, trees evolved for the TSP perform reasonably well in the QAP, while strategies trained with QAP obtain poor results with the TSP. It is important to notice that the study is limited just to two problems and it is, therefore, impossible to infer definite judgements. Anyway, it is safe to conclude that the training scenario creates a strong bias on the structure and applicability of the evolved strategies. An inspection of the best trees evolved for the TSP and the QAP reveals that they have a different composition. Effective strategies for the TSP usually contain a sequence of *rank* functions, each one combined with a specific deposit action performed by a subset of high quality ants. On the contrary, strategies for the QAP tend to rely on the best ant, i.e., many deposits are made just by the best ant. If we calculate the ratio of the appearance of rank functions over best ant terminals in evolved strategies, the distinction is clear: this value is 4.33 on trees evolved for the TSP, whereas it drops to 1.07 for the QAP trees. The evolutionary process discovered strategies specific for a given problem, providing no assurance that they can be effectively applied to distinct optimization scenarios. Given the training environment proposed, this is the desired outcome. Our future research will address what are the minimal conditions that might allow the development of strategies that can be applied to different problems.

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

4.5 Scalability

Until this point, the evolved strategies were only tested in small and moderate QAP instances ($n \leq 20$). The scalability of the rules discovered by STGP is an important issue and we will present a final set of experiments where they are applied to larger (and harder) QAP instances. This goal is accomplished by selecting six instances from the Skorin-Kapov dataset (also available at the QAPLIB), with $n = 100$. Due to space reasons, in table 5 we present the outcomes obtained only with 4 trees, but the same trend is visible for other strategies: two of the trees were trained with the TSP (Ids 224 and 329) and the other two were evolved with QAP (Ids 15221 and 18124). Clearly, evolved strategies achieve competitive results, as they outperform MMAS in all 6 instances selected. Anyway, it is important to notice that the improvement rate is small. The mean improvement for the best solution is 0.63% and for the MBF is 0.74%. A closer look to the results reveals that TSP trees generalize well to large QAP instances and confirm that strategies specifically trained for QAP tend to obtain better results.

Table 5: Results with the best trees and MMAS in the 100 size instances, with 10000 iterations for 30 runs.

Update	sko100a		sko100b		sko100c		sko100d		sko100e		sko100f	
Strategy	Best	MBF	Best	MBF	Best	MBF	Best	MBF	Best	MBF	Best	MBF
MMAS	171290	171976.60	173266	173911.27	167306	168232.27	168106	169377.60	168992	169765.86	167426	168398.86
Tree 224	169892	170872.14	172128	**171280.00**	166184	167133.86	167750	168256.60	168244	168716.47	166464	167290.33
(imp %)	0.82	0.64	0.66	**1.51**	0.67	0.65	0.21	0.66	0.44	0.62	0.57	0.66
Tree 329	169998	171192.27	172634	172046.00	166756	167364.00	168030	168553.33	168200	168922.94	167116	167649.47
(imp %)	0.75	0.46	0.36	1.07	0.33	0.52	0.05	0.49	0.47	0.50	0.19	0.45
Tree 15221	170068	170733.80	**171280**	172537.86	166198	**166794.06**	167144	**168001.27**	167616	**168328.06**	166324	**166936.40**
(imp %)	0.71	0.72	**1.15**	0.79	0.66	**0.85**	0.57	**0.81**	0.81	**0.85**	0.66	**0.87**
Tree 18124	**169638**	**170649.40**	172046	172628.86	**165660**	166903.14	**166646**	168069.80	**167510**	168477.86	**166256**	167060.94
(imp %)	**0.96**	**0.77**	0.70	0.74	**0.98**	0.79	**0.87**	0.77	**0.88**	0.76	**0.70**	0.79
Avg	169899.00	170861.90	172022.00	172123.18	166199.50	167048.77	167392.50	168220.25	167892.50	168611.33	166540.00	167234.29
(imp %)	0.81	0.65	0.72	1.03	0.66	0.70	0.42	0.68	0.65	0.68	0.53	0.69

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 obtained with the QAP confirm that evolved strategies are effective, as they are competitive with state-of-the-art ACO variants. The STGP algorithm relies on a single QAP instance of moderate size to generate promising update rules. In any case, experiments described in this paper clearly show that strategies are robust and scalable, maintaining effectiveness when reused in different instances. Moreover, and despite the adoption of high level components that appear in existing ACO variants, evolution did not fully converge to standard methods and, instead, found novel designs in structure and parameters.

An inspection of evolved trees reveals that its composition is dependent on the training problem. The STGP framework is successful in identifying important features of the problem being addressed and generates tailored strategies for that particular situation. This specificity prevents inter-problem generalization and further research is needed to understand the requirements that might allow the development of such general strategies.

The study presented here raises several other questions that we intend to address in the near future. Evolved strategies tend to exhibit a greedy behavior, which might compromise robustness and scalability. We intend to investigate if changes in the training process (e.g., relying on several instances) can contribute to alleviate this limitation. Also, we aim to perform a detailed analysis that allows the identification of a minimal component set required to evolve successful update strategies for different optimization problems. These are all important aspects and this paper is a step forward in the effort of developing Self-Ant Systems.

6. REFERENCES

[1] Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)

[2] Botee, H.M., Bonabeau, E.: Evolving ant colony optimization. Adv. Complex Systems **1** (1998) 149–159

[3] 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).

[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 (2010)

[5] Stutzle, T., Hoos, H.: Max-min ant system and local search for the traveling salesman problem. In: ICEC Proceedings, IEEE Press (1997) 309–314

[6] Montana, D.J.: Strongly typed genetic programming. Evolutionary Computation Journal **3** (1995) 199–230

[7] 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

[8] Ross, P.: Hyper-heuristics. In Burke, E., Kendall, G., eds.: Search Methodologies. Springer (2005) 529–556

[9] Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Technical Report NOTTCS TR SUB 0906241418-2747, University of Nottingham (2010)

[10] Krasnogor, N., Blackburnem, B., Hirst, J., Burke, E.: Multimeme algorithms for protein structure prediction. In: PPSN VII Proceedings. Volume 2439 of LNCS., Springer (2002) 769–778

[11] Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: Computational Intelligence: Collaboration, Fusion and Emergence. Springer (2009) 177–201

[12] Diosan, L., Oltean, M.: Evolutionary design of evolutionary algorithms. Genetic Programming and Evolvable Machines **10** (2009) 263–306

[13] White, T., Pagurek, B., Oppacher, F.: ASGA: Improving the ant system by integration with genetic algorithms. In: Proceedings of the 3rd Genetic Programming Conference, MK (1998) 610–617

[14] Poli, R., Langdon, W.B., Holland, O.: Extending particle swarm optimisation via genetic programming. In: EuroGP 2005 Proceedings. (2005) 291–300

[15] Diosan, L., Oltean, M.: Evolving the structure of the particle swarm optimization algorithms. In: EvoCOP 2006 Proceedings. (2006) 25–36

[16] Runka, A.: Evolving an edge selection formula for ant colony optimization. In: GECCO 2009 Proceedings. (2009) 1075–1082

[17] Stutzle, T.: Max-min ant system for the quadratic assignment problem. Technical Report AIDA-97-4, FB Informatik, TU Darmstadt (1997)