The ACO Encoding

Alberto Moraglio, Fernando E. B. Otero, and Colin G. Johnson

School of Computing and Centre for Reasoning,
University of Kent, Canterbury, UK
{A.Moraglio, F.E.B.Otero, C.G.Johnson}@kent.ac.uk

Abstract. Ant Colony Optimization (ACO) differs substantially from other meta-heuristics such as Evolutionary Algorithms (EA). Two of its distinctive features are: (i) it is constructive rather than based on iterative improvements, and (ii) it employs problem knowledge in the construction process via the heuristic function, which is essential for its success. In this paper, we introduce the *ACO encoding*, which is a *self-contained algorithmic component* that can be readily used to make available these two particular features of ACO to *any* search algorithm for continuous spaces based on iterative improvements to solve combinatorial optimization problems.

1 Introduction

Hybrid meta-heuristics that combine skilfully elements of two or more meta-heuristics are often superior to the original meta-heuristics when considered in isolation [1] as it is the case, for example, of Memetic Algorithms [2], which combine Evolutionary Algorithms [3] with Local Search, and more recent incarnations of Ant Colony Optimization [4] which also embed Local Search as a standard sub-component. The success of hybrid meta-heuristics is rooted in the combination of the complementary strengths of their compounding meta-heuristics [1].

Leaving aside the compelling ant foraging behavior metaphor that inspired the original design of ACO, from an algorithmic point of view, Ant Colony Optimization differs greatly from most meta-heuristics. Perhaps the two most characterizing algorithmic features of ACO are: (i) it is constructive rather than based on iterative improvements; and (ii) beside the feedback obtained by the evaluation of complete solutions, common to all other meta-heuristics, the heuristic function provides extra problem knowledge fed directly in the construction process.

One weakness of ACO is that the pheromone update rules are not guaranteed to find always the optimal pheromone levels that lead to the construction of the best solution. This is because the update rules make small incremental changes to the pheromone levels—which can be seen as a form of hill-climbing in the pheromone space—that may lead the pheromone levels to converge to a point which is only locally optimal. It would be interesting, therefore, to replace the search done in the pheromone space by the update rules with more global types

of search algorithms, such as Evolutionary Algorithms, which may be less prone to get stuck in local optima.

Beside the standard hybridization of ACO with Local Search, there are a number of works on hybrid ACO algorithms, see for example [5–7]. In this paper, rather than introducing one more hybrid ACO algorithm, we intend to present a *self-contained algorithmic component* obtained by distilling a distinctive feature of ACO behind its success—namely the use of heuristic information in the solution construction—in a way that it can be used as an off-the-shelf component in any search algorithm for continuous spaces to solve combinatorial problems. The new algorithmic component will be termed the *ACO Encoding*. This component was derived derandomizing the ACO construction mechanism and interpreting it as a decoding procedure for solutions of combinatorial problems represented as real vectors formed by pheromone levels, as illustrated in the next section. This component has advantages from two different perspectives: (i) it can be seen as endowing (a de-randomized variant of) ACO with a more global search in the pheromone space than that using the standard pheromone update rule, and, at the same time, (ii) it feeds extra problem knowledge via the heuristic function to the base search algorithm (e.g., an evolutionary algorithm) using the ACO encoding, which normally does not have such knowledge at its disposal.

2 The ACO Encoding

Search algorithms are normally characterized by:

1. the search space S defining the set of candidate solutions to the problem at hand P (i.e., the set of feasible solutions).
2. the underlying representation R of the candidate solutions (e.g., real vectors, permutations, binary strings) on which the search operators O are acting on.
3. the search strategy A that the search algorithm employs to search the given search space (e.g., local search, population-based search, annealing search).

Whereas perhaps ACO is not normally looked at this way, it is insightful to ask what its search space, its solution representation and its search strategy are.

Let us consider a basic ACO without local search for a specific problem, e.g., TSP (i.e., P=TSP). The problem being addressed is clearly a combinatorial problem. A standard solution representation for the TSP are permutations (i.e., R=permutations), which naturally encode the order of the cities to be visited by the traveling salesperson (i.e., S=permutations (*genotypes*) encoding cities tours (*phenotypes*)). As ACO constructs TSP solutions encoded using permutations, it may be argued that the construction procedure employed by ACO corresponds to the "ACO search operator" in the space of permutations (i.e., O=construction procedure). However, this way of aligning ACO to the algorithmic framework outlined above is partially unsatisfactory because there are no elements in ACO that *directly* correspond to search operators acting on such a representation. As a consequence, the search strategy A of ACO cannot be explicitly characterized on the permutation space.

Let us now consider an alternative way of looking at ACO, attempting at equating the search space of ACO with the set of all pheromone values. In this case, TSP solutions are constructed using the information stored in the pheromone values (i.e., S=pheromones vectors (genotypes) encoding cities tours (phenotypes)). Importantly, the pheromone updates rule acting on pheromone levels can be interpreted as search operators acting on such a representation (i.e., R=pheromones vectors and O=update rules). The solution construction can be then interpreted as a *growth function* that maps pheromone levels (genotypes) to permutations (intermediate phenotypes), which are in turn mapped to cities tours (phenotypes). Interestingly, in ACO the growth function maps a continuous space to a combinatorial space, hence ACO, seen in this way, searches a combinatorial space indirectly by searching a continuous space. Can the pheromone space, therefore, be considered as the search space searched by ACO? The answer is negative as an essential property of a solution representation is missing when we consider the pheromone levels as representation. This is the ability of the genotype to identify uniquely a phenotype. This is essential, as in lack of it, it would not be possible to determine the fitness of the genotype unambiguously, and as a consequence, the optimal genotype would not be well-defined.

If we are willing to allow for a change of the ACO solution construction procedure, we could indeed identify the pheromone space with the search space of ACO. There are two aspects of the ACO solutions construction procedure that are problematic when it is interpreted as a growth function: (i) normally, the ACO construction procedure returns more than one solution (one for each ant on the graph); and (ii) it is a probabilistic procedure which does not always construct the same solutions from the same pheromone levels. The first issue can be simply solved by using always a single ant. The second issue can be solved by de-randomizing the construction procedure. This can be done by placing the ant on a fixed initial node (i.e., the nest) and by de-randomizing its decisions about which node to go next by always choosing the alternative with highest probability, i.e. *by treating transition probabilities as priorities* and returning deterministically only the most probable solution. The priorities used in the de-randomized construction are obtained with the traditional ACO transition probability formula which combines heuristic information and pheromone levels. Since the heuristic information does not change in the course of the search, these priorities are uniquely determined by the pheromone levels on the construction graph, and consequently, given the same pheromone levels, the TSP tour built by the ant is unique. So, this construction procedure is a well-defined growth function.

The above derandomized variation of ACO is a degenerate form of ACO, which, as it is, does not perform any useful search. This is because it produces deterministically a single solution from the current pheromone levels, and when the pheromone update rule is applied, the pheromone values corresponding to that solution are reinforced, hence, fixating the search to that single solution. However, the interesting aspect of this derandomized variation of ACO is that it forms a conceptual bridge between the traditional ACO and other types of meta-

heuristics as this algorithm can now be characterized in terms of search space, solution representation and search operators. Seen in this algorithmic framework, the ACO representation of solutions (pheromone levels) together with *the ACO encoding* (deterministic solution construction procedure) can be naturally decoupled from the specific ACO search (pheromone update rule). *This, interestingly, allows us to use the ACO encoding as a self-contained algorithmic component in combination with any search algorithm for continuous optimization.* So, for example, we could use it in combination with a standard Evolutionary Algorithm based on a real vector representation. Each real vector (individual in the population) corresponds to a combination of pheromone levels. The fitness of that individual would be the fitness of the solution constructed by the ant using those pheromone levels. Then, the evolutionary algorithm would proceed as it normally does by selecting above average individuals, recombining and mutating them with standard search operators, to produce the next population of individuals.

The conceptual link between ACO and other meta-heuristics via the ACO Encoding is interesting, as the same real values assume two quite distinct meanings when they are understood in the two different contexts: as pheromone values in an ACO construction graph, and as solution representation in the genotype of an individual. Let us consider more closely this difference for the specific case of TSP. For this problem, in the ACO practice, the pheromone is normally stored on edges rather than on nodes of the construction graph, because its function is understood being that of "modifier" of the heuristic information, which is associated with the edges of the construction graph (since it is based on the distance between the cities). On the other hand, if we look at pheromone values as a solution representation for an EA, as a rule of thumb, we want to have the most compact representation that can represent any TSP solution, because larger spaces take normally more time to be searched. This would require to put pheromone on the nodes of the construction graph, for a total of n pheromone locations, rather than pheromone on edges which would amount to a total of n^2 pheromone locations, hence giving rise to a much larger search space to search. Note that putting pheromone on nodes is enough to represent any solution of the TSP, as when the heuristic information is set to zero, the pheromone levels on the nodes, in fact, specify the order of the cities to include in the salesperson tour by their ranks. This is known in the literature as the random-keys encoding for permutations [8]. Therefore, there seems to be a very interesting inter-play between heuristic information injected in the ACO Encoding and its redundancy.

The idea of injecting heuristic information in the decoding procedure from genotype to phenotype is not new in the ACO Encoding, as it has been used before, for example, in encoding procedures for scheduling problems using the genotypes to guide the choice of scheduling policies [9]. However, the ACO Encoding makes it possible to use heuristic information in a much more standardized way and for a much larger class of problems, borrowing from the large and rapidly increasing library of previously solved problems using ACO.

3 Experiments and Discussion

Let us consider the ACO Encoding included as a component in a evolutionary algorithm for real-vectors. The ACO Encoding raises a number of interesting questions. In the previous section, we decoupled the ACO Encoding (solution representation and construction) from the ACO search in pheromone space (pheromone update rule). The first natural question is: given the same ACO Encoding of the problem at hand, is the more global search done by an EA better than the search done by the ACO pheromone update rule? A second interesting aspect of the ACO encoding is that it allows the inclusion of heuristic information in the EA. So, a second question is: since the use of this information is essential in ACO to obtain good performance, is the heuristic information of any help to the EA? A third question relates with the dual interpretation of the real-vector as pheromone in a construction graph and as genotype to represent a solution, as discussed in the previous section. So, we ask which of the two following options is better: (i) locating pheromone on the edges of the construction graph, which from an ACO perspective it is meaningful for the TSP, or alternatively (ii) locating pheromone on the nodes of the construction graph, which makes sense from an EA perspective as it gives rise to a smaller search space to search.

In the following, we present three sets of experiments using 10 TSP instances from the TSPLIB to give preliminary answers to the above questions. As our aim is to compare directly algorithmic components rather than reaching state-of-the-art performances, we use the ACO Encoding with a very simple evolutionary algorithm for real-vectors (a generational scheme with discrete uniform crossover, creep mutation and tournament selection). Furthermore, we do not include local search neither in ACO nor in the EA and, for fairness, we give the same number of fitness evaluations (number of solutions constructed and evaluated) to each of the algorithms in the comparison.

We have selected two well-known ACO algorithms, namely ant colony system (ACS) [10] and $\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) [11, 12]. For the ACO algorithms, the following default parameter settings from the literature are used: $\beta = 2$, $\alpha = 1$, $m = n$ (where m is the number of ants and n is the number of cities) and $\rho = 0.98$ (where ρ is the evaporation rate). In the case of our EA using the proposed ACO Encoding, we have used three variations: EA-ACO, EA-ACO-b0 (EA-ACO without heuristic information) and EA-ACO-n (EA-ACO with pheromone on the nodes). We have performed a systematic (but coarse) parameter tuning in order to determine a suitable combination of values, leaving $\alpha = 1$ and $\beta = 2$ parameters fixed—the only exception is that in EA-ACO-b0, β is set to 0 since it does not use heuristic information. For the remaining parameters, we have tested the following values: *population_size* = $\{n \cdot 10, \boldsymbol{n \cdot 10}\}$, *crossover_rate* = $\{0.5, \mathbf{1.0}\}$, *mutation_rate* = $\{1/n, \boldsymbol{2/n}\}$ and *tournament_size* = $\{\mathbf{2}, 4\}$. The values in bold represent the best combination of values out of the 16 possible combinations and the ones used in our experiments. Although the parameter tuning did not show significant differences between the combinations of values, the EA seems to be more sensitive to the *tournament_size* parameter,

Table 1. Computational results for symmetric (upper section) and asymmetric (lower section) instances from the TSPLIB. The number following the instance name corresponds to the number of cities. The column '*opt*' indicates the known optimal tour length for each instance. A value in the remaining columns represents the average tour length of the best tour found by the correspondent algorithm over 25 runs (*average*±*standard deviation*); the value closer to the optimal tour length is shown in bold.

instance	*opt*	ACS	$\mathcal{MM}$AS	EA-ACO	EA-ACO-b0	EA-ACO-n
berlin52	7542	7868.9±192.7	**7562.8±57.4**	7849.8±68.7	9585.1±460.2	7886.0±31.6
eil51	426	468.5±11.8	**428.7±1.6**	430.6±4.1	504.8±13.3	439.9±2.9
kroA100	21282	28497.8±562.6	**21492.8±145.2**	21852.8±148.2	69800.5±2424.2	22306.5±131.0
st70	675	806.6±15.2	**681.5±4.5**	715.1±8.1	1206.9±48.6	712.9±11.6
br17	39	**39.0±0.0**	**39.0±0.0**	41.7±6.4	39.0±0.2	39.7±3.4
ft70	38673	45072.3±320.0	39586.2±331.2	39732.4±200.3	44474.2±689.3	**39370.8±154.4**
p43	5620	5630.9±1.3	**5629.3±0.9**	5652.6±6.8	5673.5±15.5	5638.5±6.4
ry48p	14422	16147.9±335.9	14598.3±70.6	**14534.6±46.6**	17252.4±631.5	14721.4±145.2

where the combinations using the smaller value 2 perform better than the ones using the greater value 4. The stopping criteria for all algorithms was set to $14 \cdot 10^4$ fitness evaluations. Therefore, even though the algorithms use a different number of ants (ACO algorithms) and population size (EA algorithms), the comparison is based on the total number of candidate solutions evaluated, which will be the same for all algorithms.

Table 1 presents the computational results. For each of the algorithms, we report the average tour length of the best tour found in 25 independent runs of the algorithms. The value closer to the optimal tour length is shown in bold in Table 1. The results show that generally $\mathcal{MM}$AS achieves the best performance. The only exception are the 'ft70' and 'ry48p' asymmetric instances, where the EA-ACO and EA-ACO-n achieve best performance. It is interesting to note that EA-ACO and EA-ACO-n outperform ACS, except on 'br17', where ACS found the optimal solution, and on 'p43'. However, EA-ACO-b0 performs poorly compared to the other algorithms, except for the 'br17' instance, where it outperforms the other two EA variations. In terms of computational time, both ACS and $\mathcal{MM}$AS require on average 27 seconds to complete the $14 \cdot 10^4$ fitness evaluations, EA-ACO-n is a factor of 2.6 slower than ACS and $\mathcal{MM}$AS, while EA-ACO and EA-ACO-n are a factor of 110 slower than ACS and $\mathcal{MM}$AS.

Let us now consider the questions that were posed at the beginning of the section in the light of the experimental results obtained. Given the same encoding, is the EA search (EA-ACO) better than the ACO search (ACS and $\mathcal{MM}$AS) by means of the update rule? From the experiments, it seems that the EA search reaches better solutions than the simple ACS, but it does not as well as the more sophisticated $\mathcal{MM}$AS. Note that one of the major differences between ACS and $\mathcal{MM}$AS is the ability of the latter to change adaptively the learning rate to prevent getting stuck in local optima in the phenotype space. This seems to be

consistent with our initial hypothesis that the EA search is more global than the simple search done by ACS. However, the adaptive search of $\mathcal{MM}$AS seems to be very effective. The EA used in this work is a very simple one. It would be, therefore, interesting in future work to consider EA with an adaptive type of mutation and compare it with $\mathcal{MM}$AS. In terms of CPU time, the ACO systems are much more efficient than the EA-ACO. The critical component seems to be the crossover operator that takes quadratic time with respect with the number of cities. However, this problem may be overcome by using a parameter set for the EA that requires to apply crossover only a limited number of times.

A second question was about the impact of the heuristic information injected in the EA using the ACO Encoding. From the experiments, it is evident that the heuristic information in EA-ACO clearly allows it to outperform the same algorithm when this information is not available (EA-ACO-b0). This is a very interesting result as it shows that the ACO Encoding is indeed able to inject heuristic information in a EA in a way that can be usefully employed to reach better performance. More generally, the ACO Encoding may open up at the possibility to inject heuristic information in any search algorithm operating on a continuous representation and systematically boost its performance.

A third question we put forward was about whether it is better to locate the pheromone on edges (as in EA-ACO) or on the nodes (as in EA-ACO-n). In terms of efficiency, which is, CPU time needed, EA-ACO-n is much more efficient than the EA-ACO, as the time required by the crossover operator is now linear with the number of cities in a solution. In terms of performance, the pheromone on edges seem to be more advantageous than the pheromone on nodes, but not of much. This is interesting as it shows something very counter-intuitive from an EA point-of-view: a heavily redundant representation corresponding to a much larger than necessary search space to search seems to lead to a better performance than a more compact one. It is also interesting to note that EA-ACO-n, which incorporates heuristic information, is both superior to and much more efficient than EA-ACO-b0, which has pheromone on edges and it does not incorporate heuristic information. Therefore, EA-ACO-n seems to be an interesting trade-off in terms of performance, efficiency and use of heuristic information.

Finally, we can consider the experimental results above as representing a test of whether the effectiveness of ACO is due to the *encoding/representation* or to the *search* algorithm. The results tentatively suggest that the *encoding/representation* is at the core of the power of ACO, as replacing one search algorithm by another does not compromise the effectiveness of the search. Naturally, more experimentation across problem domains is needed to provide stronger support for this argument. This constitutes an important piece of future work.

4 Conclusions

This paper has introduced the ACO Encoding, an algorithmic component obtained by decoupling the encoding used in ACO from the search component, so that the encoding can be used with other search techniques. We have demon-

strated some basic experiments on TSP instances, consisting of combining this encoding with a very simple EA search, showing a similar rate of success to a good ACO variant, and presented an analysis of these results. In future work, we will use the ACO encoding with state-of-the-art algorithms for continuous optimization, such as differential evolution and self-adaptive evolution strategies, in the attempt to reach top performance on hard combinatorial optimization problems.

References

1. C. Blum, M.J. Blesa Aguilera, A.R., Sampels, M., eds.: Hybrid Metaheuristics: An Emerging Approach to Optimization. Springer (2008)
2. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program (1989)
3. Bäck, T., Fogel, D.B., Michalewicz, T., eds.: Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing (2000)
4. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
5. Lee, Z.J., Su, S.F., Chuang, C.C., Liu, K.H.: Genetic algorithm with ant colony optimization (ga-aco) for multiple sequence alignment. Applied Soft Computing **8(1)** (2008) 55–78
6. Xiong, W., Wang, C.: A hybrid improved ant colony optimization and random forests feature selection method for microarray data. In: Networked Computing and Advanced Information Management, International Conference on. (2009)
7. Wong, Kuan Yew; See, P.C.: A hybrid ant colony optimization algorithm for solving facility layout problems formulated as quadratic assignment problems. Engineering Computations: Int J for Computer-Aided Engineering **27(1)** (2010) 117–128
8. Snyder, L.V., Daskin, M.S.: A random-key genetic algorithm for the generalized traveling salesman problem. European Journal of Operational Research **171(1)** (2006) 38–53
9. Runwei Cheng, M.G., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithmsi. representation. Computers and Industrial Engineering **30(4)** (1996) 983–997
10. Dorigo, M., Gambardella, L.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation **1**(1) (1997) 53–66
11. Stützle, T., Hoos, H.: Improvements on the Ant System: Introducing $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. In: Proc. Int. Conf. Artificial Neural Networks and Genetic Algorithms. (1997)
12. Stützle, T., Hoos, H.: $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. Future Generation Computer Systems **16**(8) (2000) 889–914