Using Genetic Algorithms to optimize ACS-TSP

Marcin L. Pilat and Tony White

School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada
{mpilat,arpwhite}@scs.carleton.ca

Abstract. We propose the addition of Genetic Algorithms to Ant Colony System (ACS) applied to improve performance. Two modifications are proposed and tested. The first algorithm is a hybrid between ACS-TSP and a Genetic Algorithm that encodes experimental variables in ants. The algorithm does not yield improved results but offers concepts that can be used to improve the ACO algorithm. The second algorithm uses a Genetic Algorithm to evolve experimental variable values used in ACS-TSP. We have found that the performance of ACS-TSP can be improved by using the suggested values.

1 Introduction

Models created based on natural systems have been successfully used to solve NP-hard combinatorial optimization problems. The Ant Colony Optimization (ACO) meta-heuristic [2] is a generic framework for ant-based optimization algorithms. ACO algorithms, such as Ant System (AS) [5] and Ant Colony System (ACS) [4], were successfully used to solve instances of the Travelling Salesman Problem (TSP) [7], and other combinatorial optimization problems [3].

The ACS-TSP algorithm [4] produced better results on many TSPs compared to some genetic algorithms (GA), simulated annealing (SA), and evolutionary programming (EP) [1]. The goal of our research was to improve the performance of ACS-TSP by augmenting it with ideas from Genetic Algorithms (GAs) [6].

We look at two modifications to the ACS-TSP algorithm. The first algorithm, called ACSGA-TSP, uses a population of genetic ants modified by a GA. We present a comparison between the performance of ACSGA-TSP and ACS-TSP and provide some ideas that can be used to improve the ACSGA-TSP algorithm. The second algorithm uses a Meta GA to evolve the optimal values of experimental parameters used in ACS-TSP. We discuss improvements that can be made to the ACS-TSP algorithm based on our findings.

2 ACS-TSP Algorithm

The ACS-TSP algorithm is a modified AS-TSP algorithm where the update of the pheromone trail happens locally while an ant is building its trail and globally by the best performing ant. The tour nodes are chosen based on a more

complicated transition function that is either deterministic (to promote use of graph knowledge) or probabilistic (to promote exploration).

The performance of ACS-TSP was experimentally found to be better than that of AS-TSP [4]. We have implemented the ACS-TSP algorithm as defined by [1] and were able to verify the results of [4].

3 ACSGA-TSP Algorithm

We propose the ACSGA-TSP algorithm as a Genetic Algorithm modification to ACS-TSP. The algorithm uses a GA to evolve a population of genetically modified ants to improve the performance of the ACO algorithm.

In the original ACS-TSP algorithm, the ants used constant global parameter values. We have augmented each ant with its own value of ACS-TSP parameters. Each GA ant was encoded by a 21 bit string composed of three parameters (β, ρ, and q_0) encoded using 7 bits each. The allowed value ranges were: integer value 0-127 for β and double value 0-1 for ρ and q_0.

Ant chromosomes were initially randomized at ant creation. The crossover operator used a simple single point crossover scheme on whole chromosomes of two parents to create two children. A bit-wise mutation operator was able to mutate each bit of a chromosome based on a given probability.

In each iteration of the algorithm, four GA ants were selected from the ant population. This selection was done using a tournament selection algorithm of size 4. Each of the four selected ants were then asked to build their TSP tours.

Each GA ant stored its values of the three encoded parameters. The β and q_0 parameters were used by the ants to choose the next city to visit. Local update of the pheromone trail was done by each ant using the value of its ρ parameter.

Once the tours were completed, the algorithm checked to see whether a new tour has been found, as in ACS-TSP. The global update of the pheromone trail was done by the ant that produced the best overall tour using the encoded value of the ρ parameter.

Fitness was calculated as the length of the generated tour. After the pheromone update, the best two selected ants were crossed over to produce two children. Mutation was then performed on the offspring and the worst two selected ants were replaced by the children. This kept the population size constant and provided pressure for the population to improve its performance.

Starting location of each ant was randomized at the start of each iteration. The behavior of the ants is influenced by the pheromone trail left during a run of the algorithm, thus, the performance of later ants can be biased. This could be solved by restarting the algorithm with the same population.

We have used a population of 20 ants in our experiments. ACSGA-TSP was run for a number of iterations such that the total number of ant tours built was equivalent to that of the ACS-TSP algorithm we have used. Probability of crossover was set to 0.9 and probability of mutation to 0.01.

Results of the experiments are given in Table 1. The algorithms shared similar results. The standard deviation of the numerical solutions found by ACSGA-TSP

was on average smaller than that of ACS-TSP. This was especially noticeable for larger problems.

Table 1. Comparison of results obtained from ACSGA-TSP and ACS-TSP algorithms. Results given are averages of best tour lengths over 10 runs (eil51, ft70) or 5 runs (kroA100) and standard deviation values for the results. Iteration values were adjusted to yield the same total number of ant tours in each algorithm.

Problem	ACS AV	STD	iter.	ACSGA AV	STD	iter.
eil51	428.7	2.45	2K	432.4	4.18	10K
kroA100	21712	316	3K	21948	92	15K
kroA100	21614	310	5K	21544	250	25K
ft70	41144	452	4K	40868	379	20K

From our results, the ACSGA-TSP algorithm does not guarantee finding the optimal solution for a problem. The ant population converges to experimental variable values that produced the best results during the algorithm run. At that point, it is difficult for the ants to improve the algorithm since the individuals can be trapped at local minima. Pheromone trails due to good solutions can be erased by worse performing ants. Thus, in its current form, ACSGA-TSP would not outperform the ACS-TSP algorithm.

The rate at which good solutions were found was observed to be quicker using ACSGA-TSP, as seen in Table 2. Quick convergence to good solutions is thus a desirable characteristic of the ACSGA-TSP algorithm. For large problems, where optimal solutions are intractable or not desired, this algorithm provides good solutions faster than the ACS-TSP algorithm, as shown in Table 2. This characteristic is mainly due to the variety in the population of ants which facilitates early exploration of the search space.

Bad performing ants can disrupt the search for an optimal solution, but can also be used to improve it. In the ACS-TSP algorithm, each ant uses the same variable values, thus a wrong choice of trails can lead other ants into computing bad tours. Ants with abnormal variable values producing bad solutions can erase the trails of other, better performing ants by saturating the graph with their own pheromone information. We conjecture that if implemented properly, this can help to improve the performance by helping out good solutions stuck at local minima of the search space.

4 Meta ACS-TSP Algorithm

The proposed Meta ACS-TSP algorithm is a meta-level Genetic Algorithm running on top of ACS-TSP. The GA is used to evolve the optimal parameter values used in the ACS-TSP algorithm. We have taken the task of verifying the optimality of the specific values used by Dorigo and Gambardella [4].

Table 2. Comparison of results obtained from ACSGA-TSP and ACS-TSP algorithms on the large TSP instance rat783. Results given are averages of best tour lengths over 3 runs, standard deviation values for the results, and the number of iterations. In (a), the algorithms were run such that the same number of tours were explored as in ACS-TSP runs. In (b), the algorithms were run for the same amount of time as in ACS-TSP runs.

Algorithm	Average	STD	iterations
ACS-TSP	12181	135	100
ACSGA-TSP (a)	10057	201	500
ACSGA-TSP (b)	10323	80	186 (3min)

The Meta ACS-TSP algorithm is a wrapper around the ACS-TSP algorithm. It uses a population of encoded values of the ACS-TSP parameters. Each individual in the population can be thought of as a separate instance of the ACS-TSP algorithm with unique parameter values.

Individuals were encoded as bit-strings of length 12 with experimental variables β, ρ, and q_0 encoded using 4 bits each. The length of 4 bits for each variable was chosen in order to decrease the search space. Value of the β parameter was an integer in range 0-15. Values of the ρ and q_0 variables were doubles in range 0-1. In our initial experimentation, we were interested in value range of approximately 10 increment units. We conjecture that a smaller digitization of the value would not improve results of the algorithm.

Pseudocode of the Meta ACS-TSP algorithm is given in Fig. 1. The selection process was done by a tournament selection algorithm with tournament of size 4.

```
1: for each generation do
2:    choose 4 individuals randomly from the population
3:    for each of 4 chosen individuals do
4:       run ACS-TSP given β, ρ, and q_o value encoded in each individual and record
         the result as the fitness of the individual
5:    end for
6:    choose 2 individuals with best fitness from chosen 4
7:    produce 2 children by crossover or copy from 2 chosen best individuals
8:    mutate the 2 children
9:    replace 2 worst individuals from chosen 4 in the population with the 2 children
10: end for
```

Fig. 1. Pseudocode of the Meta ACS-TSP algorithm.

We have used a single point crossover operator that treated each encoded variable as an atomic unit. Thus, our three variable chromosome contained four crossover points. Using this specialized crossover operator the values of the variables were inherited by the children from one of their parents. The operator did not modify the parent values thus creating a greater probability of passing useful

and well performing genetic material to the next generation. In our experiments, we have used a crossover probability of 0.9.

Our mutation operator modified only one of the encoded variables by incrementing or decrementing the variable value by 1. This resulted in small changes between generations while still allowing for slow climbing toward local optima. We have used a mutation probability of 0.2 with our mutation operator.

Table 3 summarizes our results. For most problems, the average values of the experimental variables were similar to the best values. The differences between the averages of the average results and averages of the best results were very small, thus we used the average results in our analysis. Statistical analysis was done on most of the runs and the deviation from the average values in a population was reasonable.

Table 3. Results of Meta ACS-TSP algorithm runs on TSP/ATSP instances. Average values for each problem are averages over 8 runs (eil51), 4 runs (eil76), and 3 runs (kroA100, p43, ry48p, ft70). Best values are best fitness (tour) results over all the runs of a problem. Overall average values are calculated from the table data. The simulations were run for 1K-3K iterations depending on the problem and setup.

Problem	AV β	AV ρ	AV q_0	AV Fit.	Best β	Best ρ	Best q_0	Best Fit.	Optimal
eil51	6.25	0.127	0.495	428.5	6	0.13	0.4	426	426
eil76	6.75	0.128	0.5	542.5	6	0.13	0.4	538	538
kroA100	4.67	0.145	0.503	21513	5	0.13	0.53	21308	21282
p43	2	0.363	0.927	5628	2	0.2	0.93	5620	5620
ry48p	6.67	0.173	0.3	14584	6	0.07	0.47	14495	14422
ft70	8.33	0.34	0.767	40569	8	0.73	0.87	39804	38673
Average	**5.78**	**0.187**	**0.669**	N/A	**5.5**	**0.232**	**0.6**	N/A	N/A

The most unique problem was the ATSP instance p43. It had the lowest β value, and the highest ρ and q_0 values. The variability in results between different problems would lead us to believe that the optimal values of the experimental variables are unique to the problem.

We conjecture that there is no magic value that will make ACS-TSP yield the optimal solution for every TSP/ATSP instance. From our experimentation using the Meta ACS-TSP algorithm, we propose that the suggested values given in Table 4 can be used instead of those of Dorigo and Gambardella [4] in order to yield better solutions with the ACS-TSP algorithm.

5 Conclusion

We have tried to improve the performance of the ACS-TSP algorithm by proposing the ACSGA-TSP algorithm which introduced a genetic approach to ACS-TSP. Our results have shown that the algorithm does not perform as well as ACS-TSP with respect to finding the optimal solution. Quick convergence and

Table 4. Comparison of average values of variables found by Meta ACS-TSP and the values used by Dorigo and Gambardella in ACS-TSP [4]. Meta ACS-TSP results are from Table 3. Values we feel should yield best solutions are listed as suggested values.

	β	ρ	q_0
Meta ACS-TSP Average	5.78	0.187	0.669
Meta ACS-TSP Best	5.5	0.232	0.6
ACS-TSP	2	0.1	0.9
Suggested Values	6	0.2	0.7

low variability of ACSGA-TSP can be an advantage for time constrained problems.

The ACSGA-TSP algorithm can be improved by a more complex GA model. A hybrid system of ACS-TSP and ACSGA-TSP can be used to take advantage of the early convergence of ACSGA-TSP and optimal results of ACS-TSP. This hybrid system would initially execute ACSGA-TSP and eventually switch to ACS-TSP by turning off the GA. It would be interesting to apply GA models to other ACO algorithms and other combinatorial optimization problems.

We used the Meta ACS-TSP algorithm to evolve experimental variable values used in ACS-TSP. The algorithm offered results that enabled us to suggest variable values alternate to those used by Dorigo and Gambardella [4].

We have only run Meta ACS-TSP on six small TSP/ATSP instances. Running the algorithm on more and larger TSP/ATSP instances would allow for better statistical analysis of the results and improved suggested variable values. The values we suggested should also be tested to see if they statistically improve the performance of ACS-TSP over the original values.

References

1. Bonabeau E., Dorigo M., Theraulaz G. Swarm Intelligence: From Natural to Arftificial Systems. New York: Oxford University Press, 1999.
2. Dorigo M., Di Caro G.: The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization. McGraw-Hill, 1999.
3. Dorigo M., Di Caro G., Gambardella L.M.: Ant Algorithms for Discrete Optimization. Artificial Life **5** (1999) 137-172
4. Dorigo M., Gambardella L.M.: Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. IEEE Trans. Evol. Comp. **1** (1997) 53-66
5. Dorigo M., Maniezzo V., Colorni A.: The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Trans. Syst. Man Cybern. B **26** (1996) 29–41
6. Holland J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, 1975.
7. Stützle T., Dorigo M.: ACO Algorithms for the Traveling Salesman Problem. In K. Miettinen, M. Makela, P. Neittaanmaki, J. Periaux, editors, Evolutionary Algorithms in Engineering and Computer Science. Wiley, 1999.