Ant colony optimization algorithms with multiple simulated colonies offer potential advantages for solving the traveling salesman problem and, by extension, other optimization problems

M. Evan Wildenhain[1] and Ian Sacco[1]

[1] Scripps Ranch High School, San Diego, California.

Summary

The traveling salesman problem (TSP) is a classic problem in optimization, frequently used for measuring the performance of optimization algorithms. The goal in solving the TSP is to determine the lowest-cost circuit through a set of cities on a graph. Ant colony optimization (ACO) algorithms, inspired by nature, use simulated ants that modify their environment through laying and removing pheromone, represented by weights on the edges of the graph connecting each city. In this study, a novel algorithm is developed, Multi-Colony System (MCS), which uses multiple colonies of simulated ants in combination to produce superior solutions to the TSP. In comparison with Ant Colony System (ACS), a standard well-performing ACO algorithm, MCS has displayed improved performance, producing tours up to 19.4% shorter than those of ACS in the same amount of time. The performance of MCS in this study presents potential advantages in applications beyond the TSP, including the ability of multiple colonies to both develop a greater number of solutions simultaneously and to more efficiently avoid local maxima in the search space.

Received: July 4, 2014; **Accepted:** March 4, 2015; **Published:** May 22, 2015

Introduction

The traveling salesman problem (TSP) is defined as: given a complete graph with n vertices, find the lowest-cost path that starts from one vertex, visits all the other vertices exactly once, and returns to the original vertex (1). Though traveling logistics is an obvious application for the TSP, it can also be applied to genome mapping, aiming telescopes, improving industrial efficiency (e.g. drilling circuit boards), and even clustering data points (3). Because of the broad applications of the TSP, even modest improvements in the solutions can lead to dramatic savings in time or money (3). Furthermore, thanks to the existence of a large body of sample instances, the TSP is a useful tool to measure the relative performance of various general algorithms applicable to other optimization problems (2).

Ant colony optimization (ACO) is a class of optimization algorithms inspired by nature. Simulated ants move from vertex to vertex on a graph. They develop solutions and coordinate their simulated actions through *stigmergy*, or indirect communication by modifying their environment, just as real ants will communicate optimal paths from nest to food source by laying pheromone trails (4). A general application of ACO to the TSP is as follows: "ant" agents are arbitrarily distributed among the vertices of a complete graph. In every iteration of the simulation, each ant constructs its tour in a probabilistic, step-by-step fashion; an ant is more likely to choose to move along an edge from one vertex to another if that edge has a lower cost weight (a shorter length) or has a higher pheromone weight. After the ants have constructed their tours, the pheromone weights of the edges comprising the whole colony's best-so-far tour (i.e. the tour of the best-performing ant "to date") are increased. Pheromone is added in inverse proportion to the total cost of the tour; shorter tours receive more pheromone. The amount of pheromone on each edge decays by a constant percentage after each iteration. Over the course of several iterations, the edges that tend to be part of shorter tours receive more pheromone, and ants are gradually encouraged to explore tours that use "better" edges. This process leads to steady improvement in the quality of solutions generated to the TSP instance until an optimal or near-optimal tour is reached—depending, of course, on the amount of time allotted to the simulation to run (5).

While ACO algorithms are intuitively applied to the TSP, making the TSP an effective testing ground for innovative algorithms of that type, ACO algorithms are among the best-performing algorithms for other optimization problems. These include the sequential ordering problem, similar to the TSP except that it finds a minimum-cost path rather than circuit; the vehicle routing problem, in which *n* customers must be served from one central depot, with the aim of satisfying the customers' demand for merchandise while minimizing travel time between customers; and the quadratic assignment problem, in which one must assign a set of facilities to a set of locations with given distances between each location and flows between the facilities, with the goal of minimizing the sum of the products between flows and distances (2).

The objective of this experiment is to develop a

Algorithm:	ACS	MCS (asymmetric)	MCS (symmetric)
Trial 1	662	659	645
Trial 2	659	651	648
Trial 3	651	664	648
Trial 4	658	662	652
Trial 5	645	652	641
Trial 6	657	658	651
Trial 7	650	654	659
Trial 8	650	659	645
Trial 9	651	652	642
Trial 10	647	656	651
Mean	653	656.7	648.2
Best	645	651	641
Std. Dev.	5.617433182	4.448470399	5.349974039
Avg. Err.	0.038155803	0.044038156	0.030524642

Table 1. Data for the eil101 TSP instance (Optimal: 629, n = 101)

Algorithm:	ACS	MCS (asymmetric)	MCS (symmetric)
Trial 1	16095	16371	15969
Trial 2	16434	16085	15925
Trial 3	15963	16060	16127
Trial 4	16074	16079	15979
Trial 5	16114	16078	15985
Trial 6	16061	15985	16190
Trial 7	16063	16147	16147
Trial 8	16136	16149	15916
Trial 9	16165	16170	15942
Trial 10	16124	16231	15935
Mean	16122.9	16135.5	16011.5
Best	15963	15985	15916
Std. Dev.	122.4585644	107.2300227	102.4220571
Avg. Err.	0.021730038	0.022528517	0.014670469

Table 2. Data for the d198 TSP instance (Optimal: 15780, n = 198)

novel form of ACO that utilizes multiple unique colonies in parallel to construct superior solutions to the TSP; this algorithm is the Multi-Colony System (MCS). A ubiquitous challenge with ACO algorithms is striking the proper balance between exploitation of already good solutions and exploration of a wider set of solutions. Too much of the former can lead to premature convergence of the pheromone matrix towards favoring a suboptimal solution, preventing improvement, while too much of the latter can slow the optimization of solutions already found. This study seeks to produce an ACO algorithm that runs multiple colonies based on ACS, an existing "standard" ACO algorithm (2), with unique parameters, allowing each colony to have a different degree of focus on exploitation and exploration. The individual colonies can then share their best solutions with each other in order to cooperate in searching for solutions more efficiently. This delegation of tasks to separate colonies has the potential to result in superior solutions in a shorter timeframe.

Three algorithms were tested in this study: ACS, a standard ACO algorithm, which served as a reference for comparing MCS variants; Asymmetric MCS, which was composed of two ACS colonies – one with parameters focused on exploitation of known good edges and the other colony focused on broader exploration of less-used edges; and Symmetric MCS, which has both colonies focused on exploitation of known good edges. It is important to note that ACS with "recommended" parameters is tuned, relative to the parameters used in this experiment, toward exploitation of known good edges (5). A previous study that has made a similar attempt at a parallelized ACO algorithm for the TSP failed to significantly improve upon standard algorithms by using a multiple-colony setup (6); however, that study used a base algorithm, Elitist Ant System, that is not as well-suited to parallelization as ACS. The distinction is that ACS only reinforces the edges of the best-so-far tour in each iteration (5), making sharing the best tours across colonies more effective for coordinating the colonies' search for solutions. Thus, our hypothesis was that MCS (Asymmetric and Symmetric) would construct tours shorter than tours constructed by ACS on easy (101-city), moderate (198-city), and difficult (442-city, 1002-city) TSP instances within the same amount of time.

Results

This experiment compared the performance of two variations of MCS, a novel ACO algorithm proposed in this study, to that of ACS, an established ACO algorithm for generating TSP solutions. The two variations of MCS tested, Asymmetric MCS and Symmetric MCS, differed only in the $q0$ parameter; $q0$ represents the probability of an ant automatically choosing the single most probable edge according to the transition rule instead of choosing its next edge probabilistically. A higher $q0$ makes an ant more likely to pick the "best" available edge, leading to more exploitation of known good edges rather than exploration of a more diverse range of possible paths. Asymmetric MCS was run with two colonies, one with $q0 = 0.8$ and the other with $q0 = 0.2$, and Symmetric MCS was run with two colonies, each with $q0 = 0.8$. The setting of the q0 parameter was, in fact, the only difference between the MCS variants. Four TSP instances with known optimal solutions were used for testing: eil101, a 101-city problem (**Table 1**); d198, a 198-city problem (**Table 2**); pcb442, a 442-city problem (**Table 3**); and pr1002, a 1002-city problem (**Table 4**). Each algorithm was run 10 times against each of the problems for 5040 seconds per trial.

In order to determine statistical significance, a one-tailed t-test with a 95% confidence level was used. The null hypothesis was that the performances of the MCS variants were equal to the performance of ACS; the alternate hypothesis was that the MCS variants would improve upon ACS (produce shorter tours).

Tables 1-4 display the trial results. The first row indicates the algorithm to which the row's data belongs. Rows 2–11 display the shortest tour achieved by the algorithm during their respective trials. Row 12 displays the mean of rows 2–11. Row 13 lists the shortest tour achieved out of all of the trials. Row 14 is the standard deviation of the trials, and row 15 is the average error:

$$\text{Average Error} = (\text{Mean} - \text{Optimal})/\text{Optimal} \qquad [1]$$

Algorithm:	ACS	MCS (asymmetric)	MCS (symmetric)
Trial 1	68226	58467	55852
Trial 2	70744	56427	53949
Trial 3	63101	59383	55123
Trial 4	68415	57129	54051
Trial 5	67960	57400	53825
Trial 6	69998	55205	56015
Trial 7	68353	55684	53742
Trial 8	63001	58687	55308
Trial 9	69035	55420	55215
Trial 10	71715	56621	54506
Mean	68054.8	57042.3	54758.6
Best	63001	55205	53742
Std. Dev.	2900.261896	1443.67379	852.5800321
Avg. Err.	0.340241837	0.123366419	0.078392217

Table 3. Data for the pcb442 TSP instance (Optimal: 50778, n = 442)

Algorithm:	ACS	MCS (asymmetric)	MCS (symmetric)
Trial 1	419202	385251	367218
Trial 2	426787	378195	361933
Trial 3	408941	396589	363680
Trial 4	426450	399555	367813
Trial 5	418461	405461	356929
Trial 6	426551	401049	365936
Trial 7	422585	418455	365870
Trial 8	418576	425374	357756
Trial 9	409363	402497	363535
Trial 10	440408	402366	358968
Mean	421732.4	401479.2	362963.8
Best	408941	378195	356929
Std. Dev.	9214.833527	13784.55832	3948.292171
Avg. Err.	0.628027563	0.549843463	0.401161188

Table 4. Data for the pr1002 TSP instance (Optimal: 259045, n = 1002)

With eil101, Asymmetric MCS (one colony with $q0$ = 0.8, the other with $q0$ = 0.2) generated tours that were, on average, 0.56% longer than tours generated by ACS (**Table 1**). In d198, Asymmetric MCS generated tours 0.08% longer on average than those of ACS. The data indicates that Asymmetric MCS does not significantly outperform ACS at smaller TSP instances ($p > 0.05$) (**Table 2**). However, there was a significant difference in the results for pcb442 and pr1002 (**Tables 3 and 4**, respectively). For pcb442, Asymmetric MCS generated tours that were on average 16.18% shorter than tours generated by ACS, as well as tours that were 4.80% shorter for pr1002 compared to those of ACS. Both of these differences were significant ($p < 0.05$). These results indicate that ACS and MCS are similar in performance for the two smaller TSP instances, but Asymmetric MCS is superior for larger TSP instances.

In contrast, Symmetric MCS (with two colonies, each with $q0$ = 0.8) performed slightly better than ACS and Asymmetric MCS at eil101 and d198 (**Tables 1 and 2**, respectively), and significantly outperformed them at pcb442 and pr1002 (**Tables 3 and 4**, respectively). Symmetric MCS generated tours 0.74% shorter than ACS's tours for eil101, 0.69% shorter for d198, 19.54% shorter for pcb442, and 13.94% shorter for pr1002. All of these differences were significant ($p < 0.05$).

The hypothesis was that MCS could construct tours shorter than tours constructed by ACS on easy (101-city), moderate (198-city), and difficult (442-city, 1002-city) TSP instances within the same amount of time. The data indicates that Asymmetric MCS's improvement over ACS was significant for only the two larger TSP instances, while Symmetric MCS provided significant improvement over ACS for all four instances ($p < 0.05$).

Discussion

This study demonstrated that both Asymmetric and Symmetric MCS can outperform ACS on larger TSP instances. For the TSP instances pcb442 and pr1002 (442 cities and 1002 cities, respectively), both Asymmetric MCS and Symmetric MCS produced tours significantly shorter than those produced by ACS. However, in the smaller TSP instances eil101 and d198, the performance of ACS, Asymmetric MCS, and Symmetric MCS were all comparable. It is possible that MCS appears to be a superior algorithm for the larger TSP instances because of the tremendous difference in the size of the solution space between d198 and pcb442. The number of possible solutions to a symmetric TSP instance is the number of cities factorial divided by two (a symmetric TSP instance's edges have the same travel cost in both directions). The difference between 442! / 2 and 198! / 2 is gigantic—a factor of approximately 5.5 x 10^{608}. When operating in a significantly larger search space, the ability of MCS to pursue potentially twice as many solutions as ACS and maintain two distinct but cooperative pheromone maps is more advantageous.

Symmetric MCS, with two colonies focused on exploitation, outperformed Asymmetric MCS, with one colony focused on exploitation and the other on exploration, in all four TSP instances. One potential explanation is that the primary advantage of MCS is the ability of multiple colonies to explore multiple solutions semi-independently, and having the focus on exploitation afforded by $q0$ = 0.8 is simply a better-performing parameter for the transition rule (7). The transition rule, a probabilistic formula through which a simulated "ant" selects the next city to visit in its tour, is explained further in the Materials and Methods section. Exploitation allows for more rapid improvement of solutions, and avoidance of local maxima can be handled by the presence of multiple colonies.

Symmetric MCS appears to be a promising alternative to ACS. Delivering significantly improved performance with the 442-city instance and the 1002-city instance, Symmetric MCS has the potential to be a superior algorithm for solving even more challenging TSP instances. While Symmetric MCS has yet to be compared to other ACO algorithms such as Best-Worst Ant System, with further testing and development, Symmetric MCS can become a leading ACO algorithm for solving the TSP.

Future research similar to the experiments carried out here could be beneficial to furthering the understanding

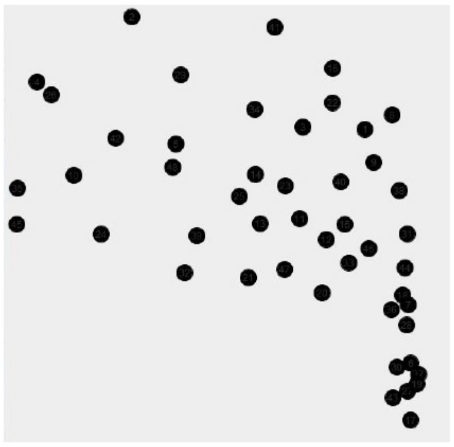

Figure 1. A 48-city TSP instance. At the beginning, every path from each city to all the other cities has the same initial amount of pheromone, set based on a nearest-neighbor heuristic.

of these algorithms. Asymmetric MCS and Symmetric MCS were run with only two colonies, and the only parameter that differed between the colonies was *q*0 (only in Asymmetric MCS); other parameters and more colonies could be used to better tune the algorithm. Symmetric MCS outperformed Asymmetric MCS in a two-colony configuration, but with more colonies, it is possible that having different parameters or even algorithms for each colony could improve performance. "Local search" is a term used to describe algorithms that slightly modify existing solutions by searching for better similar solutions—solutions "local" to the existing one in the problem graph. To improve upon new global-best tours, ACS and both MCS variants used 2-opt local search, an algorithm that reorders a tour in order to eliminate points where the tour crosses over itself. 2-opt local search could be applied differently: it was used only in this study to improve new global-best tours, but it could be used as part of the construction of every tour. This would increase the resulting computation costs. A better local search algorithm could be applied, such as 2.5-opt or 3-opt—or conversely, a less computationally expensive algorithm such as greedy 2-opt. Modifications to the control group of ACS could be made to result in better comparisons; multiple ACS colonies run independently parallel during a trial would be more comparable to MCS, although the consistency of results in this study suggests that this would not significantly change the outcome. For pcb442 and pr1002, the best tour lengths achieved by ACS were longer than the mean tour lengths for both Asymmetric MCS and Symmetric MCS. In fact, for pcb442 and pr1002, every Symmetric MCS trial outperformed the best ACS trial, indicating that MCS still presents a significant improvement.

Methods

MCS shares much of its basic function with ACS. All ACO algorithms are based on stigmergy, a process by which ant agents leave pheromone on a graph in order to communicate and influence other ants' construction of solutions. In the case of the TSP, ants communicate and gradually improve their solutions by modifying the amounts of pheromone on a graph of edges between the cities of a TSP instance. At the beginning of the run, each edge has a low initial amount of pheromone. At the start of every iteration, the ants of each colony are placed at random starting cities (**Figure 1**).

The ants construct their tours in the same way as ants do in ACS. At each step of an ant's tour, a random number q between 0 and 1 is compared with a parameter q0, where 0 < q0 < 1:

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \{[\tau(r,u)] * [\eta(r,u)]^{\beta}\}, & \text{if } q \le q0 \text{ (exploitation)} \\ S, \text{otherwise (biased exploration)} \end{cases} \quad [2]$$

where s is the next city the ant will visit, r is the current city the ant is on, J_k is the set of cities not yet visited by the ant on its tour, τ is the amount of pheromone on an edge, η is visibility, or (the length of the edge)$^{-1}$, and β is a parameter determining the relative importance of pheromone and visibility. This rule determines whether the ant will build its tour according to exploitation or exploration. If *q* ≤ *q*0, the "best" edge (i.e. the one with the highest probability of being selected) is taken. Otherwise, S is found via the probabilistic transition rule of AS:

$$p_k(r, s) = \begin{cases} \frac{[\tau(r,s)]*[\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)]*[\eta(r,u)]^{\beta}}, & \text{if } s \in J_k(r) \\ 0, \ otherwise \end{cases} \quad [3]$$

where p_k(r, s) is the edge between cities r and s. Informally, shorter edges with more pheromone are more likely to be traveled than longer edges with less pheromone. A higher q0 thus leads to increased selection of the "best" edges, while a lower q0 makes the selection of other edges more likely. In MCS, the colonies can have differing q0 values, so a high-q0 colony has a higher probability of exploitation of the "best" edges, while

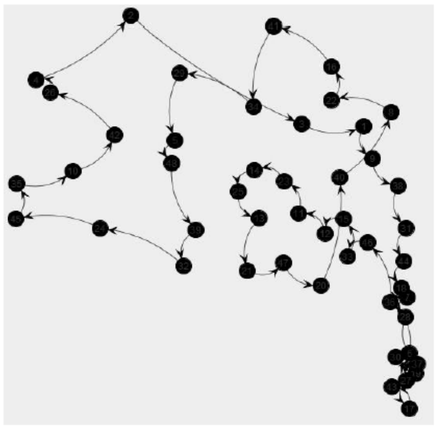

Figure 2. After all of a colony's ants have completed their tours, this is found to be the colony's best tour of the iteration—or rather, the best tour after it is improved via a 2-opt local search algorithm. This tour will be reinforced according to the global pheromone update rule.

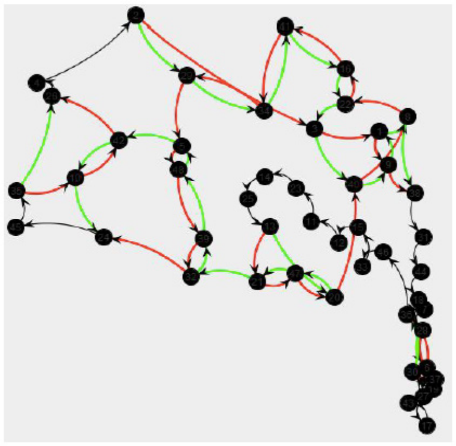

Figure 3. The two colonies of the run have reported their best-so-far tours. Paths shared are in black, paths unique to the colony from Figure 2 are in red, and paths unique to the other colony are in green. Only the best tour between the two colonies (in this case, the tour with edges highlighted in green) will have its paths reinforced, again according to the global pheromone update rule.

low-q0 colonies' ants are more likely to construct tours that differ from the best-so-far tour.

After all the ants in a colony have completed their tours for the iteration, the local pheromone update rule is applied to an edge once for each time it was visited by an ant that iteration:

$$\tau(r,s)\leftarrow(1-\rho)*\tau(r,s)+\rho*\tau_0 \qquad [4]$$

where ρ is the pheromone decay parameter and τ_0 is the initial amount of pheromone, set to be the same for all edges at the beginning of the run. τ_0 is set as the number of cities, multiplied by 1 / (Cnn), where Cnn is the length of a tour constructed by a nearest-neighbor heuristic (**Figure 2**).

After the application of the local pheromone update rule, the lowest-cost, or shortest, tour of the iteration is compared to the shortest tour the colony has recorded since the beginning of the run. If the former is shorter than the latter, the iteration-best tour is run through a 2-opt local search algorithm, and the resulting optimized tour replaces the previous colony-best tour. The colony-best tour is then allocated pheromone according to the following global pheromone update rule:

$$\tau(r,s)\leftarrow(1-\alpha)*\tau(r,s)+\alpha*\ 1/L_{gb} \qquad [5]$$

where α is a tuning parameter and L_{gb} is the length of the colony-best tour. After each colony has completed this process, it reports its colony-best tour. If a colony's best tour is shorter than the global-best tour across all colonies previously recorded, it replaces the global-best tour (**Figure 3**).

In each iteration, the edges of global-best tour are allocated pheromone in all colonies according to the same global pheromone update rule. This step allows the colonies to share their best tours and thus work

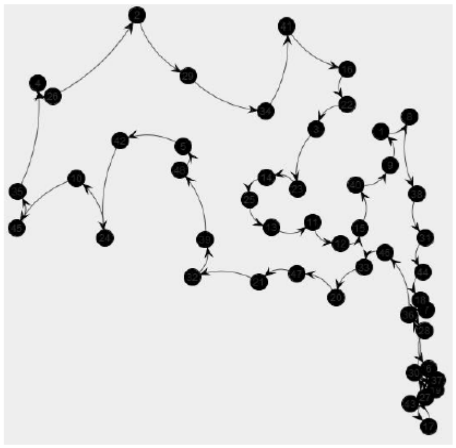

Figure 4. As the iterations progress, the pheromone map will favor increasingly shorter tours. Eventually, given enough time and iterations, an ant from one of the colonies will reach an optimal or near-optimal tour for the TSP instance, as seen here.

to construct solutions in parallel. In the colony that produced the global-best tour, the edges of the global-best tour are reinforced twice, while they are reinforced once in the other colony. The above process is repeated every iteration until the program either reaches a predetermined number of iterations or a time limit (**Figure 4**).

The algorithms tested in this experiment were all implemented entirely by the student researcher in Java. The trials were run on an Intel Core i5-3450 CPU at 3.10 GHz with 8 GB DDR3 RAM, within the Java Eclipse IDE. The TSP instances tested (eil101, d198, pcb442, and pr1002) were obtained from TSPLIB, an online library of sample TSP instances (8).

References

1. Lin, Shen. "Computer solutions of the traveling salesman problem." Bell System Technical Journal 44.10 (1965): 2245-2269.
2. Dorigo, Marco, and Thomas Stutzle. Ant Colony Optimization. Cambridge, MA: Massachusetts Institute of Technology, 2004. Print.
3. Cook, William. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton: Princeton University Press, 2012. Print.
4. Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni. "Ant system: optimization by a colony of cooperating agents." Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 26.1 (1996): 29-41.
5. Dorigo, Marco, and Luca Maria Gambardella. "Ant colony system: A cooperative learning approach to the traveling salesman problem." Evolutionary Computation, IEEE Transactions on 1.1 (1997): 53-66.
6. Alonso, Sergio, Oscar Cordón, Iñaki Fernández de Viana, and Francisco Herrera. "Integrating evolutionary computation components in ant colony optimization." Recent Developments in

Biologically Inspired Computing, L. Nunes de Castro, FJ Von Zuben (Eds.), Idea Group Publishing (2004): 48-180.

7. García, Oscar Cordón, I. Fernández de Viana, and Francisco Herrera Triguero. “Analysis of the best-worst Ant System and its variants on the TSP.” Mathware & soft computing 9.3 (2002): 177-192.
8. “TSPLIB.” *TSPLIB*. Heidelberg University, 6 Aug. 2008. Web. 30 Nov. 2013. <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>.