Pre-scheduled Colony Size Variation in Dynamic Environments

Michalis Mavrovouniotis[1(✉)], Anastasia Ioannou[2], and Shengxiang Yang[3]

[1] School of Science and Technology, Nottingham Trent University, Clifton Lane, Nottingham NG11 8NS, UK
michalis.mavrovouniotis@ntu.ac.uk
[2] Department of Informatics, University of Leicester, University Road, Leicester LE1 7RH, UK
ai63@le.ac.uk
[3] Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, UK
syang@dmu.ac.uk

Abstract. The performance of the $\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) in dynamic optimization problems (DOPs) is sensitive to the colony size. In particular, a large colony size may waste computational resources whereas a small colony size may restrict the searching capabilities of the algorithm. There is a trade off in the behaviour of the algorithm between the early and later stages of the optimization process. A smaller colony size leads to better performance on shorter runs whereas a larger colony size leads to better performance on longer runs. In this paper, pre-scheduling of varying the colony size of $\mathcal{MM}$AS is investigated in dynamic environments.

1 Introduction

Ant colony optimization (ACO) is a metaheuristic inspired by the foraging behaviour of real ant colonies [2,3]. ACO algorithms have been successfully applied to many $\mathcal{NP}$-hard combinatorial problems such as the travelling salesman problem (TSP) [4] and vehicle routing problem (VRP) [7]. In this paper, we focus on a particular ACO variation, i.e., $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [22], which is one of the best performing variations.

The construction of solutions from ants is biased by existing pheromone trails and heuristic information. Pheromone trails are updated according to the search experience and towards solution with good quality. This is similar to a learning reinforcement scheme. The behaviour and performance of $\mathcal{MM}$AS algorithm depends strongly on the number of ants used [5,24]. When a given computational budget is available, e.g., the maximum number of function evaluations, a smaller number of ants will produce more algorithmic iterations whereas a larger number of ants less. Hence, the colony size affects the duration of the learning reinforcement [18].

G. Squillero and K. Sim (Eds.): EvoApplications 2017, Part II, LNCS 10200, pp. 128–139, 2017.
DOI: 10.1007/978-3-319-55792-2_9

In [24], it was investigated that when fewer ants are used, the algorithm may converge quickly at early stages of the optimization process, but get stuck in the stagnation behaviour later on. In contrast, when more ants are used, the algorithm converges slower but to better solutions at later stages of the optimization process. Considering this $\mathcal{MM}$AS's behaviour, it can be observed that the optimal number of ants depends on the stage of the optimization process. Therefore, $\mathcal{MM}$AS can benefit from varying the number of ants. For example, a pre-schedule of varying the colony size may improve the performance of $\mathcal{MM}$AS in dynamic environments. The key idea is to start a few ants when a change occurs and gradually increase the number of ants. In this way, $\mathcal{MM}$AS will benefit from both merits of a small (fast convergence) and large (improve solution quality) colony size at different stages of the optimization process.

Several dynamic test cases of the dynamic TSP (DTSP) are generated for our study. The rest of the paper is organized as follows. Sections 2 and 3 describe the DOPs generated and the ACO algorithm used for this study, respectively. Section 4 discusses the importance of the colony size parameter. Section 5 presents the experimental study with discussions. Finally, Sect. 6 concludes this paper.

2 Dynamic Environment

2.1 Dynamic Travelling Salesman Problem (DTSP)

The DTSP is modelled by a fully connected weighted graph $G = (N, A)$, where $N = \{v_1, \ldots, v_n\}$ is a set of n nodes (e.g., cities) and $A = \{(v_i, v_j) \mid v_i, v_j \in N, i \neq j\}$ is a set of arcs (i.e., links), where n represents the size of a problem instance. Each arc $(v_i, v_j) \in A$ is associated with a non-negative value $d_{ij} \in \mathbb{R}^+$, which represents the distance between cities v_i and v_j. The objective of the problem is to find the shortest Hamiltonian cycle that starts from one node and visits each of the other cities once before returning to the starting city.

The distance matrix of the DTSP is subject to changes, which is defined as follows: $\mathbf{D}(t) = \{d_{ij}(t)\}_{n \times n}$, where t is the period of a dynamic change. A particular TSP solution $s = [s_1, \ldots, s_n]$ in the search space is specified by a permutation of the nodes (cities) and it is evaluated as follows:

$$f(s,t) = d_{s_n s_1}(t) + \sum_{i=1}^{n-1} d_{s_i s_{i+1}}(t). \tag{1}$$

2.2 DTSP Benchmark Generators

The concept of DTSPs was initially introduced by Psaraftis [20]. Since then, several variations of DTSPs were introduced, where the set of nodes N [1,9,10,12,13,25] and/or the cost from the set of arcs A [6,17,19,21,25] cause the weight matrix $\mathbf{W}(t)$ to change during the optimization process. However, there is still no any unified benchmark problem for DTSPs, which makes the

comparison with algorithms from the literature a very challenging task. One popular benchmark is the DTSP where cities are exchanged: half of the cities from the problem instance are removed to create a spare pool [9,10,14], and the cities from the spare pool are then used to replace cities from the problem instance. Another popular benchmark is the DTSP where the weights of arcs change probabilistically [17,25] (the complete benchmark generator description is given in Sect. 2.3 since it is the benchmark generator we consider in the experiments). In [6,21], only the weights of arcs that belong to the best tour increase or decrease accordingly.

Younes *et al.* [26] introduced a benchmark generator for the DTSP with different modes: (1) topology changes as in [10], (2) weight changes in [6], and (3) swap cities. Based on the last mode (i.e., swap cities) of the aforementioned benchmark generator, a general dynamic benchmark generator for permutation-encoded problems (DBGP) was proposed that can generate test cases with known optima [15]. DBGP can convert any stationary TSP instance into a DTSP with specific properties (i.e., frequency and magnitude of changes). Although with DBGP one can observe how close to the optimum an algorithm converges, it sacrifices real-world models for the sake of benchmarking.

2.3 Generating Dynamic Test Cases

Considering the problem formulation above, a dynamic test case of a TSP can be generated by modifying the value of the arc between nodes v_i and v_j as follows:

$$w_{ij}(T+1) = \begin{cases} w_{ij}(0) + \mathcal{N}(\mu, \sigma), & \text{if } (i,j) \in A_S(T); \\ w_{ij}(T), & \text{otherwise;} \end{cases} \tag{2}$$

where $T = \lceil t/f \rceil$ is the environmental period index, f is the frequency of change, t is the evaluation count of the algorithm, $\mathcal{N}(\mu, \sigma)$ is a random number generated from a normal distribution with $\mu = 0$ and $\sigma = 0.2 \times w_{ij}(0)$, $w_{ij}(0)$ is the weight between nodes v_i and v_j for the initial instance and $A_S(T) \subset A$ contains exactly $\lceil m(n(n-1)) \rceil$ arcs in which their weights will be subject to changes (either increase or decrease) [25].

Since many real-world problems can be formulated as DTSPs and methods for solving static TSPs can be applied to solve them [8]; the dynamic changes generated in this paper can be generalized and may represent different factors depending on the application. For example, in logistics, the weight changes may represent traffic on the road system or in telecommunications the weight changes may represent delays on the network.

3 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System

3.1 Construct Solutions

One of the state-of-the-art ACO variations is the $\mathcal{MM}$AS [22]. A colony of ω ants read pheromones in order to construct their solutions and write pheromones

to store their solutions. Each ant k uses a probabilistic rule to choose the next node to visit. The decision rule of the kth ant to move from node v_i to node v_j is defined as follows:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} [\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^{\alpha} [\eta_{il}]^{\beta}}, \text{if } j \in \mathcal{N}_i^k, \tag{3}$$

where τ_{ij} and η_{ij} are the existing pheromone trail and heuristic information available a priori between nodes v_i and v_j, respectively. The heuristic information is defined as $\eta_{ij} = 1/d_{ij}(t)$, where $d_{ij}(t)$ is defined as in Eq. (1). $\mathcal{N}_i^k$ is the neighbourhood of unvisited nodes incident to node i available for ant k to select. α and β are the two parameters which determine the relative influence of τ_{ij} and η_{ij}, respectively.

3.2 Pheromone Update

The pheromone trails in $\mathcal{MM}$AS are updated by applying evaporation as follows:

$$\tau_{ij} \leftarrow (1 - \rho)\, \tau_{ij}, \forall (v_i, v_j), \tag{4}$$

where ρ is the evaporation rate which satisfies $0 < \rho \leq 1$, and τ_{ij} is the existing pheromone value. After evaporation, the best ant deposits pheromone as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \forall (v_i, v_j) \in T^{best}, \tag{5}$$

where $\Delta\tau_{ij}^{best} = 1/C^{best}$ is the amount of pheromone that the best ant deposits and C^{best} defines the solution quality of tour T^{best}. The best ant that is allowed to deposit pheromone may be either the best-so-far, in which case $C^{best} = C^{bs}$, or the iteration-best, in which case $C^{best} = C^{ib}$, where C^{bs} and C^{ib} are the solution quality of the best-so-far and the iteration best ant, respectively. The best-so-far ant is a special ant that may not necessarily belong in the current colony of ants as the iteration best ant. Both update rules are used in an alternate way in the implementation [23].

The lower and upper limits τ_{min} and τ_{max} of the pheromone trail values are imposed. The τ_{max} value is bounded by $1/(\rho C^{bs})$, where C^{bs} is initially the solution quality of an estimated optimal tour and later on is updated whenever a new best-so-far ant solution quality is found. The τ_{min} value is set to $\tau_{min} = \tau_{max}/2n$.

Since only the best ant is allowed to deposit pheromone, the colony may quickly converge towards the best solution found in the first iteration. Therefore, the pheromone trails are occasionally reinitialized to the τ_{max} value to increase exploration. For example, whenever the stagnation behaviour occurs or when no improved solution is found for a given number of iterations, the pheromone trails are reinitialized.

3.3 Adapting to Dynamic Changes

$\mathcal{MM}$AS is able to use knowledge from previous environments via pheromone trails and can be applied directly to DOPs without any modifications [1,16]. For example, when the changing environments are similar, the pheromone trails of the previous environment may provide knowledge to speed up the optimization process to the new environment. However, the algorithm needs to be flexible enough to accept the knowledge transferred from the pheromone trails, or eliminate the pheromone trails, in order to adapt well to the new environment. In particular, pheromone evaporation enables the algorithm to forget bad decisions made in previous iterations. When a dynamic change occurs, evaporation eliminates the pheromone trails of the previous environment from areas that are generated on the old optimum and helps ants to explore for the new optimum.

In case the changing environments are different, then pheromone reinitialization may be a better choice rather than transferring the knowledge from previous pheromone trails [1,9,10,16]. For instance, when a change occurs, the pheromone trails are initialized with an equal amount.

4 Varying the Colony Size

4.1 Effect of the Colony Size in Dynamic Environments

A previous empirical study showed that the colony size of the $\mathcal{MM}$AS algorithm, one of the best performing ACO algorithms, is sensitive to the properties of DOPs [18]. In particular, if for a given DOP only a certain computation budget, e.g., the maximum number of function evaluations, is available, then the colony size, i.e., the number of ants, is a very critical parameter. Since each ant in a colony corresponds to a single function evaluation, an unnecessarily large colony size may waste computations whereas an extremely small colony size may restrict the searching capabilities of ACO.

Furthermore, the colony size has a direct relation with the reinforcement learning period of ACO because it determines its duration: less ants corresponds to larger duration whereas more ants corresponds to smaller duration. Also the colony size determines how broad the search is at each iteration (e.g., more ants means broader search). Hence, the number of ants needs to be tuned accordingly in order not to waste computation resources and degrade the solution quality.

In this paper, we study the impact of the colony size on the performance of the $\mathcal{MM}$AS algorithm for DOPs. This kind of problems in a nutshell are a series of stationary optimization problems that all need to be optimized. Therefore, it is straightforward that more challenges exist and the colony size will have impact on the performance of the algorithm. This is because it determines the number of iterations and the broadness of the search as in stationary optimization problems. For example, for a given DOP a predefined computation budged is available between each environmental change that is typically synchronized with the algorithm, i.e., every f evaluations a change occurs [15]. Therefore, an algorithm with a larger colony size means that it will perform a broader search (i.e.,

more evaluations per iteration) but it will have limited reinforcement learning (i.e., less number of iteration) for each environmental change.

4.2 Pre-scheduling the Colony Size

The colony size of $\mathcal{MM}$AS was investigated on the stationary TSP [24]. In particular, the number of ants used shows a trade-off between the early and later optimization process of the algorithm regarding the solution quality. At early stages of the optimization process fewer ants result to better performance, whereas at later stages more ants result to better performance. With fewer ants the algorithm seems to initially progress faster but leads to the stagnation behaviour at later stages. More ants give better results only on later stages of the optimization process. This behaviour of $\mathcal{MM}$AS can be observed in Fig. 1. Similar behaviour was observed for other problem instances (`kroA150.tsp` and `kroA200.tsp`).

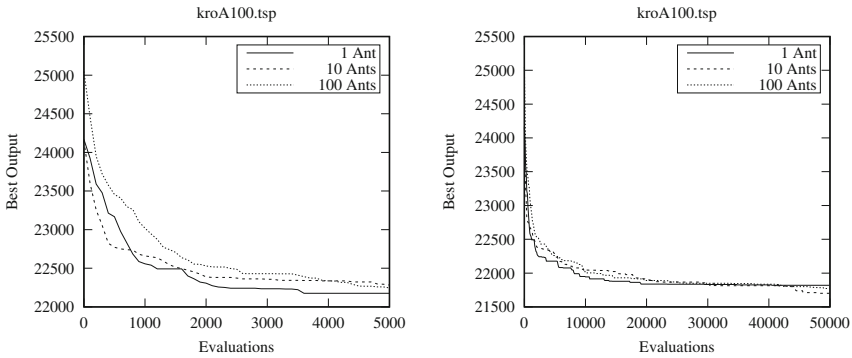

Fig. 1. Performance of $\mathcal{MM}$AS with different fixed number of ants for a short run of 5000 evaluations (left) and a long run of 50,000 evaluations (right), respectively.

Clearly, at different stages of the optimization process the optimal colony size of $\mathcal{MM}$AS varies. Therefore, adjusting the colony size during the optimization process seems a better choice rather than keeping a fixed colony size. In fact, pre-scheduling the colony size in stationary environments has proved that it can combine the merits of few ants on shorter runs and the merits of more ants on longer runs [24]. However, in this paper, we are concerned with pre-schedules for dynamic environments. Considering the observations in Fig. 1, a potential good pre-schedule in a dynamic environment could be starting with a small colony size when a change occurs to quickly converge and then gradually increase the colony size to further improve the solution quality.

In particular, four pre-schedules are investigated which are defined as follows:

1. Pre-schedule 1: every 15 iterations add a single ant
2. Pre-schedule 2: every 10 iterations add a single ant
3. Pre-schedule 3: every 5 iterations add a single ant
4. Pre-schedule 4: every 2 iterations add a single ant

All schedules start with an initial colony size of 1 ant and increase by 1 ant at a time. When a change occurs the colony size is reset back to 1 and starts to grow until the next dynamic change occurs. An arbitrary number of different pre-schedules can exist but in this paper we consider these four to determine under which frequency of increasing the number of ants $\mathcal{MM}$AS performs better. The size of the colony increases faster from pre-schedule 1 to pre-schedule 2, pre-schedule 3 and to pre-schedule 4.

5 Experimental Study

5.1 Experimental Setup

To investigate the effect of the colony size of $\mathcal{MM}$AS in dynamic environments, three TSP stationary benchmark instances (i.e., `kroA100.tsp`, `kroA150.tsp` and `kroA200.tsp`) were obtained from TSPLIB[1] and corresponding DOPs are generated using the benchmark generator (described in Sect. 2.3) with f set to 5000 and 50000 function evaluations, indicating quickly and slowly changing environments, respectively, and m set to 0.1, 0.25, 0.5 and 0.75, indicating slightly, to medium, to severely changing environments, respectively. Totally, a series of 8 dynamic test cases of DTSPs are constructed from each stationary benchmark instance to systematically investigate $\mathcal{MM}$AS algorithm with the proposed pre-scheduled colony size against standard fixed colony size.

The colony size of a traditional $\mathcal{MM}$AS was set to fixed values, i.e., $\omega \in \{1, 2, 5, 10, 25, 50, 100\}$, and the results are compared with the pre-scheduled variation of $\mathcal{MM}$AS. The remaining parameters were set to typical values for DOPs as follows: $\alpha = 1$, $\beta = 5$ and $\rho = 0.8$ from our preliminary experiments.

5.2 Performance Measurement

For each DTSP, 30 independent runs of the $\mathcal{MM}$AS were executed. For each run, 25 environments changes were allowed and the best so far ant after a dynamic change was recorded. The overall *offline performance* [11] is defined as follows:

$$\bar{P}_{offline} = \frac{1}{E} \sum_{i=1}^{E} \left(\frac{1}{R} \sum_{j=1}^{R} P_{ij}^{*} \right), \tag{6}$$

where E is the total number of function evaluations, R is the number of runs, P_{ij}^{*} is the best-so-far after a dynamic change of iteration i of run j.

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

5.3 Results and Discussion

The offline performance results of the $\mathcal{MM}$AS algorithm on DTSPs with fixed and pre-scheduled colony sizes are presented in Table 1. Pairwise comparisons between the best fixed colony variation (1, 2, 5, 10, 25, 50 and 100 ants) against the best pre-scheduled colony variation (1, 2, 3 and 4 pre-schedules) using Mann–Whitney statistical tests are performed. The best variation of one type with a bold value indicates that is significantly better than the best variation of the other type. In case both variations types are in bold it indicates insignificantly difference between them. In Figs. 2 and 3, the dynamic offline performance for quickly and slowly changing environments against the algorithmic evaluations are plotted for the first 10 environments to better understand the behaviour of $\mathcal{MM}$AS. From the experimental results, the following observations can be drawn.

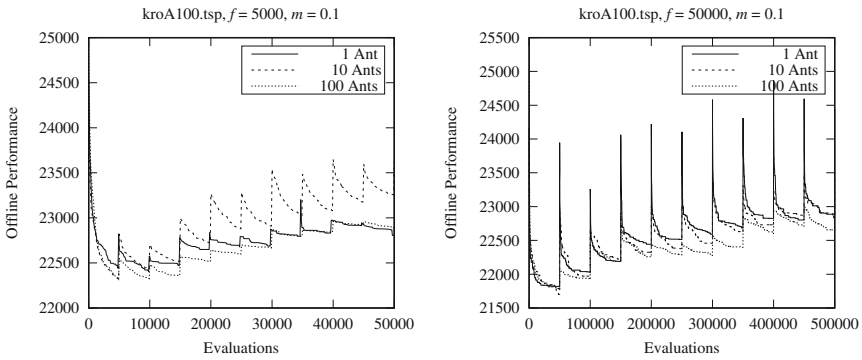

Fig. 2. Dynamic offline performance of $\mathcal{MM}$AS with different fixed number of ants for quickly changing (left) and slowly changing (right) DTSPs, respectively.

First, the offline performance of fixed colony variations of $\mathcal{MM}$AS with a larger size is better in most test cases. Only on few cases, i.e., when $m = 0.75$, a smaller colony size has better performance. These results were expected for slowly changing DTSPs, i.e., $f = 50000$, because more ants perform better in long runs. However, a large colony also performs better for quickly changing DTSPs, i.e., $f = 5000$. This is contradictory with the observations in Fig. 1, where a small colony size performed better in a shorter run (corresponds to a quickly changing environment). From Fig. 2, it can be observed that the performance up to the first environment (before any change occurs) the results match the one in Fig. 1. When a dynamic change occurs $\mathcal{MM}$AS with fewer ants perform worst. This is possibly because the pheromone trails generated by fewer ants of the previous environment may not promote exploration when they are used in the new environment.

Second, the offline performance of pre-scheduled colony variations of $\mathcal{MM}$AS that increase the colony size faster, e.g., Pre-schedule 3 and Pre-schedule 4,

Table 1. Comparison of $\mathcal{MM}$AS variations regarding the results of the offline performance

	$f = 5000$				$f = 50000$			
$m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
# ants	`kroA100.tsp`							
(1 ant)	23452	23565	23417	23584	22993	23125	22986	23159
(2 ants)	23378	23523	23381	23538	22967	23119	23004	23157
(5 ants)	23395	23575	23458	23648	23014	23152	23034	23188
(10 ant)	23390	23565	23410	23558	22984	23151	23007	23184
(25 ants)	22831	23030	23037	23085	22937	23135	23002	23134
(50 ants)	22892	23072	22933	23150	22880	23098	22975	23109
(100 ants)	22845	23016	22964	23128	22790	23092	23006	23112
Pre-schedule 1	23199	23385	23226	23284	22826	23055	22929	23055
Pre-schedule 2	23294	23318	23126	23193	22781	23038	22952	23081
Pre-schedule 3	22804	23050	**22740**	**22830**	**22609**	**22913**	**22803**	**22916**
Pre-schedule 4	**22707**	**22887**	23071	23144	22814	23066	22963	23090
# ants	`kroA150.tsp`							
(1 ant)	28892	29276	29473	29538	**28404**	**28797**	**28975**	**29004**
(2 ants)	28874	29260	29492	29588	28436	28842	29029	29060
(5 ants)	29020	29338	29610	29727	28558	28907	29082	29146
(10 ant)	28993	29308	29466	29568	28596	28922	29088	29188
(25 ants)	28600	28885	29229	29419	28578	28902	29057	29158
(50 ants)	28575	28897	29236	29300	28542	28861	29056	29156
(100 ants)	28596	28810	29237	29302	28508	28883	28966	29195
Pre-schedule 1	28832	29065	29163	29313	28474	28870	**28960**	29161
Pre-schedule 2	28856	29040	**29019**	29309	28491	28845	28995	29116
Pre-schedule 3	28566	28930	29189	29286	**28414**	28824	28994	29176
Pre-schedule 4	**28405**	**28701**	29334	**29143**	28500	28843	29004	29146
# ants	`kroA200.tsp`							
(1 ant)	31799	32405	32722	32829	31240	31712	32044	32103
(2 ants)	31635	32178	32588	32620	31219	31684	32029	32061
(5 ants)	31602	32148	32561	32592	31165	31656	32011	32017
(10 ant)	31543	31958	32409	32327	31131	31641	31992	31980
(25 ants)	31156	31529	32171	32105	31077	31583	31969	31987
(50 ants)	31191	31505	32138	32129	31055	31552	31973	31914
(100 ants)	31158	**31493**	32032	32113	**30982**	31468	31915	31802
Pre-schedule 1	31326	31692	32134	32086	31005	31503	31860	31895
Pre-schedule 2	31243	31534	31991	**31850**	**30978**	31463	31841	31825
Pre-schedule 3	31084	**31466**	**31882**	32101	31002	**31345**	**31806**	**31718**
Pre-schedule 4	**30935**	31643	31155	32248	31017	31549	31861	31858

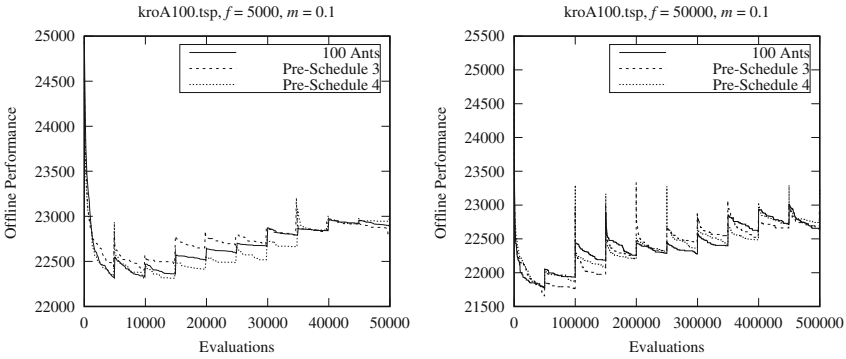

Fig. 3. Dynamic offline performance of the best fixed and pre-schedule $\mathcal{MM}$AS variations for quickly changing (left) and slowly changing (right) DTSPs, respectively.

perform better in most test cases. Only in `kroA150.tsp` problem instance when $f = 50000$ none of the pre-scheduled variations perform better. This is because the results of fixed variation in Table 1 show that 1 ant outperforms other fixed variations. Therefore, a pre-schedule that increases the colony size will not be helpful. For the remaining cases, either Pre-schedule 3 or Pre-schedule 4 performs better. This shows that the speed of increasing the colony size is problem dependent.

Finally, the comparisons between the fixed and pre-scheduled variation show that with the exception of `kroA150.tsp` when $f = 50000$, the best performing pre-schedule variation outperforms the best performing fixed variation in many DTSPs. From Fig. 3, it can be observed that a pre-scheduled colony size is able to maintain a better offline performance than a fixed colony size.

6 Conclusions

The optimal colony size of $\mathcal{MM}$AS algorithms varies at different stages of the optimization process. More precisely, a small colony size works better for short runs and a large colony size works better for long runs in stationary environments. This paper, investigates different pre-schedules for DTSPs, where $\mathcal{MM}$AS begins with a single ant and gradually increase its colony size in dynamic environments. When a dynamic change occurs, the colony is reset back to a single ant. The key idea of the pre-schedule is to combine the benefits of small and large colonies. The experiments for different DTSP test cases showed that a varying colony size has a promising performance when compared with a fixed colony size. However, the performance of the pre-schedule of the varying colony strongly depends on the properties of the DTSP. Hence, a direct future work would be to self-adapt the colony size of $\mathcal{MM}$AS. In this way, a possibly automatic pre-schedule will be generated for different DTSPs.

Acknowledgement. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant EP/K001310/1.

References

1. Angus, D., Hendtlass, T.: Ant colony optimisation applied to a dynamically changing problem. In: Hendtlass, T., Ali, M. (eds.) IEA/AIE 2002. LNCS (LNAI), vol. 2358, pp. 618–627. Springer, Heidelberg (2002). doi:10.1007/3-540-48035-8_60
2. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Vaerla, F., Bourgine, P. (eds.) Proceedings of the European Conference on Artificial Life, pp. 134–142. Elsevier Publishing (1991)
3. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B Cybern. **26**(1), 29–41 (1996)
4. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
5. Dorigo, M., Stützle, T.: Ant colony optimization. MIT Press, Cambridge (2004)
6. Eyckelhof, C.J., Snoek, M.: Ant systems for a dynamic TSP. In: Dorigo, M., Caro, G., Sampels, M. (eds.) ANTS 2002. LNCS, vol. 2463, pp. 88–99. Springer, Heidelberg (2002). doi:10.1007/3-540-45724-0_8
7. Gambardella, L.M., Taillard, E.D., Agazzi, C.: MACS-VRPTW: a multicolony ant colony system for vehicle routing problems with time windows. In: New Ideas in Optimization, pp. 63–76 (1999)
8. Gueta, L., Chiba, R., Ota, J., Arai, T., Ueyama, T.: A practical and integrated method to optimize a manipulator-based inspection system. In: IEEE International Conference on Robotics and Biomimetics, ROBIO 2007, pp. 1911–1918, December 2007
9. Guntsch, M., Middendorf, M.: Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: Boers, E.J.W. (ed.) EvoWorkshops 2001. LNCS, vol. 2037, pp. 213–222. Springer, Heidelberg (2001). doi:10.1007/3-540-45365-2_22
10. Guntsch, M., Middendorf, M.: Applying population based ACO to dynamic optimization problems. In: Dorigo, M., Caro, G., Sampels, M. (eds.) ANTS 2002. LNCS, vol. 2463, pp. 111–122. Springer, Heidelberg (2002). doi:10.1007/3-540-45724-0_10
11. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments-a survey. IEEE Trans. Evol. Comput. **9**(3), 303–317 (2005)
12. Kang, L., Zhou, A., McKay, B., Li, Y., Kang, Z.: Benchmarking algorithms for dynamic travelling salesman problems. In: Congress on Evolutionary Computation, CEC2004, vol. 2, pp. 1286–1292, June 2004
13. Li, C., Yang, M., Kang, L.: A new approach to solving dynamic traveling salesman problems. In: Wang, T.-D., Li, X., Chen, S.-H., Wang, X., Abbass, H., Iba, H., Chen, G.-L., Yao, X. (eds.) SEAL 2006. LNCS, vol. 4247, pp. 236–243. Springer, Heidelberg (2006). doi:10.1007/11903697_31
14. Mavrovouniotis, M., Yang, S.: A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. Soft Comput. **15**(7), 1405–1425 (2011)
15. Mavrovouniotis, M., Yang, S., Yao, X.: A benchmark generator for dynamic permutation-encoded problems. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012. LNCS, vol. 7492, pp. 508–517. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32964-7_51

16. Mavrovouniotis, M., Yang, S.: Adapting the pheromone evaporation rate in dynamic routing problems. In: Esparcia-Alcázar, A.I. (ed.) EvoApplications 2013. LNCS, vol. 7835, pp. 606–615. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37192-9_61
17. Mavrovouniotis, M., Yang, S.: Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors. Appl. Soft Comput. **13**(10), 4023–4037 (2013)
18. Mavrovouniotis, M., Yang, S.: Empirical study on the effect of population size on max-min ant system in dynamic environments. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC 2016), pp. 853–860 (2016)
19. Melo, L., Pereira, F., Costa, E.: Multi-caste ant colony algorithm for the dynamic traveling salesperson problem. In: Tomassini, M., Antonioni, A., Daolio, F., Buesser, P. (eds.) ICANNGA 2013. LNCS, vol. 7824, pp. 179–188. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37213-1_19
20. Psaraftis, H.: Dynamic Vehicle Routing Problems, pp. 223–248. Elsevier (1988)
21. Simões, A., Costa, E.: CHC-based algorithms for the dynamic traveling salesman problem. In: Chio, C., et al. (eds.) EvoApplications 2011. LNCS, vol. 6624, pp. 354–363. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20525-5_36
22. Stützle, T., Hoos, H.: $\mathcal{MAX}$-$\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In: IEEE International Conference on Evolutionary Computation, pp. 309–314 (1997)
23. Stützle, T., Hoos, H.H.: $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000)
24. Stützle, T., López-Ibáñez, M., Pellegrini, P., Maur, M., de Oca, M.M., Birattari, M., Dorigo, M.: Parameter adaptation in ant colony optimization. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 191–215. Springer, Heidelberg (2012)
25. Tinós, R., Whitley, D., Howe, A.: Use of explicit memory in the dynamic traveling salesman problem. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, pp. 999–1006. ACM, New York (2014)
26. Younes, A., Calamai, P., Basir, O.: Generalized benchmark generation for dynamic combinatorial problems. In: Proceedings of the 2005 Genetic and Evolutionary Computation Conference, pp. 25–31. ACM Press (2005)