Journal of Computational Design and Engineering 6 (2019) 368–386

Contents lists available at ScienceDirect

Journal of Computational Design and Engineering

journal homepage: www.elsevier.com/locate/jcde

A modified Ant Colony Optimization algorithm to solve a dynamic traveling salesman problem: A case study with drones for wildlife surveillance

Sudipta Chowdhury [a,*], Mohammad Marufuzzaman [b], Huseyin Tunc [c], Linkan Bian [b], William Bullington [b]

[a] Department of Civil and Environmental Engineering, University of Connecticut, Storrs, CT 06269, USA
[b] Department of Industrial and Systems Engineering, Mississippi State University, Starkville, MS 39759-9542, USA
[c] Institute of Population Studies, Hacettepe University, Beytepe, Ankara 06800, Turkey

ARTICLE INFO

Article history:
Received 23 June 2018
Received in revised form 10 October 2018
Accepted 13 October 2018
Available online 21 October 2018

Keywords:
Ant colony optimization
Dynamic traveling salesman problem
Immigrant schemes
Adaptive large neighborhood search
Drone routing
Wildlife surveillance

ABSTRACT

This study presents a novel Ant Colony Optimization (ACO) framework to solve a dynamic traveling salesman problem. To maintain diversity via transferring knowledge to the pheromone trails from previous environments, Adaptive Large Neighborhood Search (ALNS) based immigrant schemes have been developed and compared with existing ACO-based immigrant schemes available in the literature. Numerical results indicate that the proposed immigrant schemes can handle dynamic environments efficiently compared to other immigrant-based ACOs. Finally, a real life case study for wildlife surveillance (specifically, deer) by drones has been developed and solved using the proposed algorithm. Results indicate that the drone service capabilities can be significantly impacted when the dynamicity of deer are taken into consideration.

1. Introduction

The *Static Traveling Salesman Problem* (STSP) is considered as one of the most challenging $\mathcal{NP}$-complete combinatorial optimization problem that has been studied extensively for the last few decades (Applegate, Bixby, Chvatal, & Cook, 2011). The problem can be stated as follows: Given a set of available cities, STSP attempts to find the shortest path that starts from one city and visits other cities only once prior to the returning of the starting city. This problem can be represented by a fully connected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{0, \ldots, n\}$ is a set of vertices and $\mathcal{E} = \{(i,j) : i \neq j\}$ is a set of edges. Set $\mathcal{V}$ represents the collection of available cities n which are known in priori while the connections between the cities are represented by set $\mathcal{E}$. Each connection $(i,j) \in \mathcal{E}$ is associated with a non-negative value d_{ij} which represents the distance between cities $i \in \mathcal{V}$ and $j \in \mathcal{V}$. We define variable $\mathbf{X} := \{X_{ij}\}_{(i,j)\in\mathcal{E}} = 1$ if arc $(i,j) \in \mathcal{E}$ is used in the tour; 0 otherwise. Formally, the STSP can be described as follows:

$$f(x) := \underset{\mathbf{X}}{Minimize} \sum_{(i,j)\in\mathcal{E}} d_{ij}X_{ij} \tag{1}$$

Function $f(x)$ attempts to find a Hamiltonian tour of minimum total distance that starts and ends at source node 0. The literature on STSP is abundant and provides a wide spectrum of solution methods including exact methods (see e.g., Christofides, 1970; Dantzig, Fulkerson, & Johnson, 1954; Desrochers & Laporte, 1991; Tarjan, 1977) and heuristic algorithms (see e.g., Glover, 1977; Golden & Stewart, 1985; Lin, 1965; Or, 1977). However, in many real-life applications, transportation managers need to handle *stochasticity* and *dynamicity* in designing and managing a route plan. For instance, a technician plans to visit all her customers, *exactly once*, who are located in different cities. In the morning, she generates a schedule to visit all her customers while guaranteeing to come back to depot. However, due to a number of uncontrollable and unforeseeable events such as bad weather condition, road construction, traffic delays, and many more, the route plan for the technician have to be changed. Unfortunately, in most of the instances, the technician may not be able to realize these events prior to starting her trip. In practice, she may be on her way to

Peer review under responsibility of Society for Computational Design and Engineering.

* Corresponding author.
E-mail addresses: sudipta.chowdhury@uconn.edu (S. Chowdhury), maruf@ise.msstate.edu (M. Marufuzzaman), huseyin.tunc@hacettepe.edu.tr (H. Tunc), bian@ise.msstate.edu (L. Bian), wsb106@msstate.edu (W. Bullington).

https://doi.org/10.1016/j.jcde.2018.10.004
2288-4300/

serve a customer and experience such events (e.g., traffic delays, road construction). This may force the technician to change her initial route plan instantaneously in order to avoid respond to the realized events. Therefore, a new route needs to be generated by simply re-optimizing the underlying problem with realized parameters. The example implies that the distance d_{ij} in Eq. (1) cannot be treated as *constant* in dynamic and uncertain environments and they are in fact subject to change. The traveling salesman problem with varying parameters are referred to as *Dynamic Traveling Salesman Problem* (DTSP).

STSP is known to be $\mathcal{NP}$-complete. Therefore, solving it to optimality is a challenging task. Further, incorporating dynamic features into STSP makes the problem even more difficult (Emelogu, Chowdhury, Marufuzzaman, Bian, & Eksioglu, 2016). Also, the traditional solution methods provided for STSPs may not be applicable for DTSPs. As such, an expanding line of research has emerged on metaheuristic optimization algorithms such as swarm and evolutionary algorithms to solve DTSPs. These algorithms are inspired from natural self-organized systems and biological evolution, and essentially flexible in the sense that they can adapt themselves to changing environments through exploiting information obtained from earlier moves. The studies on this subject matter primarily concentrate on enhancing metaheuristic algorithms – originally developed for solving conventional static optimization problems. Examples of these algorithms include but not limited to Ant Colony Optimization algorithms (ACOs) (Mavrovouniotis & Yang, 2011a, 2012a, 2013; Sama, Pellegrini, D'Ariano, Rodriguez, & Pacciarelli, 2016), Genetic Algorithms (GAs) (Cheng & Yang, 2010a, 2010b, 2011), Particle Swarm Optimization algorithms (PSOs) (Du & Li, 2008; Eddaly, Jarboui, & Siarry, 2016), and Artificial Immune Systems (AISs) (Hart & Ross, 1999). The literature provides a variety of studies developing ACO algorithms to solve STSPs (see Cordon, De Viana, Herrera, & Moreno, 2000; Olivas, Valdez, & Castillo, 2015a, Olivas, Valdez, & Castillo, 2015b; Stutzle & Hoos, 1997, 2000) compared to other research domains (e.g., fuzzy control design (Castillo, Lizárraga, Soria, Melin, & Valdez, 2015a; Castillo, Neyoy, Soria, Melin, & Valdez, 2015b), image processing (Olivas, Valdez, & Castillo, 2008), and many more). Recent studies further extend the basic ACO algorithms to solve DTSPs. This is primarily due to the reason that ACO algorithms have inherent characteristics of being able to adapt to dynamic changes by transferring knowledge from past environments. Moreover, instance easiness, adaptability, efficiency, and many other features make ACO-based approaches very popular among researchers to solve DTSPs (Mavrovouniotis & Yang, 2013). Till now, a number of different variants of the ACO algorithms have been developed to solve DTSPs, examples include but not limited to local and global restart strategies (Guntsch & Middendorf, 2001), memetic algorithms (Mavrovouniotis & Yang, 2011b), memory-based schemes (Guntsch, Middendorf, & Schmeck, 2001), immigrant schemes (Mavrovouniotis & Yang, 2013), and pheromone manipulation schemes (Mavrovouniotis & Yang, 2010).

Among the different strategies discussed above, immigrant schemes arise as the most promising mechanisms due to their structural simplicity yet superiority in computational performance (Mavrovouniotis & Yang, 2010, 2011a). Liu proposes a *rank-based* ACO algorithm to solve DTSP (Liu, 2005). Essentially, the author improves the convergence of the basic ACO algorithm by utilizing a nonlinear selection function and a modified Q-learning method. Note that although the proposed method ensures diversity, introduced new parameters increase the computational complexity of the algorithm. Mavrovouniotis, Muller, and Yang (2015) propose a memetic algorithm that exploits the adaption capabilities of an ACO algorithm to solve a symmetric DTSP. In a follow-up paper, Mavrovouniotis, Müller, and Yang (2016) utilize local search operators to extend the earlier ACO-based memetic algorithm to solve an asymmetric DTSP. Guntsch and Middendorf (2001) propose three pheromone modification strategies, namely restart-strategy, η-strategy, and τ-strategy to investigate their performance in response to an insertion or deletion operation of a city in a DTSP. Eyckelhof and Snoek (2002) propose an ACO algorithm to solve a DTSP where the dynamicity is induced through traffic delays. The authors employ a novel technique termed as "shaking" to smoothen the pheromone trails after a dynamic change. Mavrovouniotis and Yang (2014) propose two multi-colony ACO algorithms to solve a DTSP. The authors show that the multi-colony ACO algorithms outperform the single colony ACO algorithm with respect to solution quality in solving a DTSP. Guntsch and Middendorf (2002) propose a memory-based ACO algorithm, namely, "P-ACO" where the best ants of each iteration are kept in the memory and used in updating the pheromone levels. The main drawback of P-ACO is that the algorithm may generate same ants multiple times which in turn may result in poor computational performance. To avoid redundancy within the ant population list, different extensions of P-ACO have been proposed. These extensions include Fitness Sharing P-ACO (FS-PACO) (Angus, 2009), Simple Crowding P-ACO (SC-PACO) (Angus, 2006), immigrant schemes with ACO (Mavrovouniotis & Yang, 2010, 2011a), and Memetic ACO (M-PACO) (Mavrovouniotis & Yang, 2011b) algorithms.

Apart from the studies discussed above which specifically focus on solving a classical DTSP, efforts are continued to develop algorithms for solving different variants of DTSP. Toriello, Haskell, and Poremba (2014) propose a DTSP with stochastic arc costs and provide an approximate dynamic programming heuristic to tackle this challenging problem. The authors assume that the vehicles are rerouted in real-time as congestion related information becomes available to the user. Although the cost of a decision can be known probabilistically beforehand, the true value is only revealed after the decision is executed. Groba, Sartal, and Vázquez (2015) develop a GA-based approach to solve a DTSP where the targets change their position with time. Eyckelhof and Snoek (2002) and Mavrovouniotis and Yang (2011a) propose a memory-based immigrants ACO algorithm (M-ACO) to solve a novel variant of DTSP where the traffic jams occur in a cyclic pattern. Kilby, Prosser, and Shaw (1998) and Montemanni, Gambardella, Rizzoli, and Donati (2005) propose ACO algorithms, which are extended later by Mavrovouniotis and Yang (2010, 2011b), to solve an extension of DTSP in which customer requests to route vehicles are generated incrementally. Most recently, a random immigrants ACO (R-ACO) and elitism-based immigrants ACO (E-ACO) are developed by Mavrovouniotis and Yang (2010) to solve a DTSP where the cities are exchanged between each other in a spare pool over time.

This paper extends the existing body of literature by developing an enhanced ACO-based metaheuristic to solve a DTSP. In particular, our efforts lie in developing *Adaptive Large Neighborhood Search*-based (ALNS) algorithm to generate immigrants inside an ACO framework with a goal of solving a DTSP efficiently. ALNS, originally proposed by Ropke and Pisinger (2006a) for solving vehicle routing problems, is based on iteratively destroying and repairing relatively large fractions of an incumbent solution. In its essence, it is an extension of large neighborhood search (LNS) proposed by Shaw (1998) and closely related to the ruin and recreate method by Schrimpf, Schneider, Stamm-Wilbrandt, and Dueck (2000). Detailed explanation of the ALNS procedure is provided in Section 3.4. Interested readers can refer to studies by Muller, Spoorendonk, and Pisinger (2012), Kovacs, Parragh, Doerner, and Hartl (2012), and Mancini (2016) for more details.

We compare the performance of our proposed algorithms with several peer algorithms to solve DSTPs, namely R-ACO, E-ACO, and M-ACO. A number of sensitivity analyses are carried out such as solution space diversity, ant replacement rate, and tuning various ALNS parameters to check the robustness of the proposed algorithms. Finally, we show how the proposed algorithms can be employed to route drones in surveying deer which are highly mobile in nature. The problem has similar characteristics as of a DTSP since mobility of deer are heavily reliant on feeding times, proximity to water and food sources, exposure to potential threats, and many others which eventually impact the surveillance time of a drone with a limited battery capability. We use a wildlife refuge national park in Mississippi to visualize and validate the algorithmic results.

In summary the contributions of this study are as follow:

- A novel metaheuristic is developed that leverages *Adaptive Large Neighborhood Search*-based (ALNS) algorithm to generate immigrants inside an ACO framework with a goal to solve a DTSP.
- Four variants of the proposed metaheuristic, i.e., ALNS-1, ALNS-2, ALNS-3, and ALNS-4 are developed to investigate the efficiency of each of them and determine their applicability under different circumstances. In a nutshell, these four variants can be expressed as
  - ALNS-1: ALNS-based ACO with *roulette wheel selection* strategy inside E-ACO framework
  - ALNS-2: ALNS-based ACO with *stochastic universal sampling* strategy inside E-ACO framework
  - ALNS-3: ALNS-based ACO with *roulette wheel selection* strategy inside M-ACO framework
  - ALNS-4: ALNS-based ACO with *stochastic universal sampling* strategy inside M-ACO framework
- A real life case study mirroring the behavior of DTSP is developed where drones are used for wildlife surveillance.

The rest of the paper is structured as follows. Section 2 reviews conventional ACO algorithms to solve STSP. Section 3 describes how the conventional ACO algorithms can be extended with different immigrant schemes to solve a DTSP. Later of this section we introduce our proposed ALNS-based ACO algorithm. Sections 4 and 5 provide experimental and case study results. Finally, Section 6 summarizes the key findings of this paper and provides some future research directions.

2. Basic ACO frameworks

2.1. Ant System (AS) to solve STSP

Ant Colony Optimization (ACO) algorithms emulate the behavior of real ant colonies when the ants work collaboratively to search good paths between their nest and food sources (Olivas et al., 2017; Yang et al., 2017). The ants deposit *pheromone* during their travel to mark certain favorable paths which the subsequent ants follow. To the end of this process, all the ants converge to the path with the largest deposition of pheromones (referred to as *shortest path*) and complete the "food-searching" task as efficiently as possible. The first algorithm to represent such system is called Ant System (AS), which was proposed to solve the well known STSP (Colorni, Dorigo, & Maniezzo, 1992; Dorigo, Maniezzo, & Colorni, 1996; Mostafa, Baykan, & Kodaz, 2015). Later, a number of different variants of the AS algorithm are proposed to solve STSP (Bullnheimer, Hartl, & Strauss, 1990; Mavrovouniotis & Yang, 2011a, 2012a, 2013).

To solve STSP using AS, each ant is initially placed in a randomly chosen city. Ants are then allowed to move iteratively between the cities from their starting city. Let $\mathcal{N}_i^k$ be the set of unvisited cities for ant k. While being at city i, ant k may decide to visit city $j \in \mathcal{N}_i^k$ by using the following probabilistic rule:

$$\mathbf{p}_{ij}^k(t) = \frac{\left[\boldsymbol{\tau}_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{z\in\mathcal{N}}\left[\boldsymbol{\tau}_{iz}(t)\right]^{\alpha}\left[\eta_{iz}\right]^{\beta}} \quad \text{if } j \in \mathcal{N}_i^k \tag{2}$$

where $\boldsymbol{\tau}_{ij}(t)$ is the current pheromone trail between cities i and j; $\eta_{ij} = 1/d_{ij}$ is the heuristic information available a priori, where d_{ij} is the distance between cities $i \in \mathcal{V}$ and $j \in \mathcal{V}$; and α and β are two parameters which determine the relative influence of pheromone trail and heuristic information. If $\alpha = 0$, the selection probabilities $\mathbf{p}_{ij}^k(t)$ are proportional to $[\eta_{iz}]^{\beta}$ and the closest cities are more likely to be selected. In such a case, AS corresponds to a classical *stochastic greedy algorithm*. If $\beta = 0$, only pheromone amplification is at work, leading to a *stagnation* situation which result strong suboptimal solution. After each ant completes a tour, the pheromone trails are updated as follows:

$$\boldsymbol{\tau}_{ij}(t+1) = (1-\rho)\boldsymbol{\tau}_{ij}(t) + \sum_{k=1}^{m}\Delta\boldsymbol{\tau}_{ij}^k(t) \quad \forall (i,j) \in \boldsymbol{\mathcal{E}} \tag{3}$$

where m is the number of available ants and ρ is the pheromone trail evaporation rate ($0 < \rho \leqslant 1$) which is used to avoid accumulating unlimited pheromone on path $(i,j) \in \boldsymbol{\mathcal{E}}$. Further, ρ enables ACO algorithms to "forget" previously chosen *bad decisions*. The term $\Delta\boldsymbol{\tau}_{ij}^k(t)$ represents the amount of pheromone ant k deposits on each arc $(i,j) \in \boldsymbol{\mathcal{E}}$ and is defined as follows:

$$\Delta\boldsymbol{\tau}_{ij}^k(t) = \begin{cases} 1/L^k(t) & \text{if arc } (i,j) \in \boldsymbol{\mathcal{E}} \text{ is used by ant } k \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $L^k(t)$ represents the length of the kth ant's tour. In general, the shorter the length $L^k(t)$ for ant k is, the more pheromone will be deposited in that arc and thus more likely to be chosen in future iterations of the algorithm.

2.2. Extensions of Ant System to solve STSP

The basic Ant System algorithm has been extended later to solve STSP efficiently, namely, elitist strategy (Dorigo et al., 1996), rank-based system (Bullnheimer et al., 1990), $\mathcal{MAX} - \mathcal{MIN}$ AS (Stutzle & Hoos, 1997, 2000). In elitist strategy, an additional weight (i.e., in the form of additional deposition of pheromone) is given to the best tour since the starting of the algorithm (Dorigo et al., 1996). Therefore, the edges in Eq. (4) need to be revised as follows:

$$\Delta\boldsymbol{\tau}_{ij}^{gb}(t) = \begin{cases} e/L^{gb}(t) & \text{if arc } (i,j) \in \boldsymbol{\mathcal{E}}^{gb} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $\boldsymbol{\mathcal{E}}^{gb}$ represents global-best solution which can be reinforced with a positive integer quantity e and L^{gb} is the length of $\boldsymbol{\mathcal{E}}^{gb}$. On the other hand, rank-based AS allows only $(w-1)$ ants and the global-best ant to deposit pheromone at the end of each tour construction phase (Bullnheimer et al., 1990). Note that all the ants are sorted based on the lengths of their tours prior to this assignment. Let rth best ant of the colony contributes to the pheromone update with a weight given by $max\{0, w-r\}$ and w be the weight that reinforces the pheromone trails of the global-best tour. Therefore, Eq. (3) becomes:

$$\boldsymbol{\tau}_{ij}(t+1) = (1-\rho)\boldsymbol{\tau}_{ij}(t) + \sum_{r=1}^{w-1}(w-r)\Delta\boldsymbol{\tau}_{ij}^r(t) + w\Delta\boldsymbol{\tau}_{ij}^{gb}(t) \tag{6}$$

where $\Delta\boldsymbol{\tau}_{ij}^r(t) = 1/L^r(t)$ and $\Delta\boldsymbol{\tau}_{ij}^{gb}(t) = 1/L^{gb}(t)$.

Another extension of AS is $\mathcal{MAX}-\mathcal{MIN}$ AS ($\mathcal{MMAS}$) algorithm, where an upper (τ_{max}) and lower bound (τ_{min}) to the values of the pheromone trails are imposed i.e., $\tau_{min} \leqslant \tau_{ij} \leqslant \tau_{max};\ \forall (i,j) \in \mathcal{E}$. Further, in $\mathcal{MMAS}$ pheromone trails are initialized to the upper trail limit, causing a higher exploration at the start of the algorithm. Among different extensions of the AS algorithm, $\mathcal{MMAS}$ is recognized as one of the best performing ACO algorithm to solve STSP (Stutzle & Hoos, 1997, 2000).

3. Immigrant schemes for ACO

This section introduces few immigrant-based ACO algorithms to solve DTSP efficiently. We modify the framework of population-based ACO algorithm (P-ACO) (Guntsch & Middendorf, 2002) to adapt our proposed algorithms to a dynamic environment. In P-ACO, the best ant of every iteration is stored in a *population-list* which is denoted by $k_{long}(t)$ (size $\mathbf{K}_l$). However, due to the need to maintain diversity within the population and to transfer knowledge from previous environment to the new environment, instead of using a long-term memory, a short-term memory, denoted by $k_{short}(t)$, is used. The introduction of $k_{short}(t)$ is obvious since no ant can survive more than one iteration in a dynamic environment. This is because ants in iteration $t-1$ may not provide a feasible solution for iteration t due to the change in the environment. Therefore, computationally expensive repair mechanisms are required to handle $k_{short}(t)$ and as such the following immigrant schemes are applied. For the convenience of the readers, we list the key mathematical symbols used in this section in Table 1.

With immigrants scheme, the solution construction procedure is the same as the traditional ACO algorithms i.e., placing ants randomly on cities and select the next move of an ant probabilistically using Eq. (2). However, due to dynamicity, the pheromone matrix associated with $k_{short}(t)$ and $k_{short}(t+1)$ will differ and thus require adjustment in reflecting the true pheromone trails. Let T^k be the tour of ant k, $\mathbf{K}_s$ be the size of $k_{short}(t)$, and τ_{max} and τ_0 be the maximum and initial pheromone values, respectively. The pheromone trails of each kth immigrant ant are then added as follows (Mavrovouniotis & Yang, 2012a, 2012b):

$$\tau_{ij}(t+1) \leftarrow \tau_{ij}(t) + \Delta\tau_{ij}^{k}(t) \quad \forall (i,j) \in T^k \tag{7}$$

where $\Delta\tau_{ij}^{k} = (\tau_{max} - \tau_0)/\mathbf{K}_s$. Similarly, when the worst ants are replaced by immigrant ants, the pheromone trails of each kth worst ant are removed as follows:

$$\tau_{ij}(t+1) \leftarrow \tau_{ij}(t) - \Delta\tau_{ij}^{k}(t) \quad \forall (i,j) \in T^k \tag{8}$$

These pheromone updating techniques ensure that no ants will survive in two successive iterations due to the dynamic nature of the environment.

The above framework is now used to introduce four ACO algorithms, namely, random immigrants scheme(R-ACO), elitism-based immigrants scheme (E-ACO), memory-based immigrants scheme (M-ACO), and the proposed ALNS-based immigrants scheme to solve DTSP efficiently.

3.1. Random immigrants ACO (R-ACO)

In this scheme (R-ACO), immigrants are generated randomly and replaced with the worst ants in $k_{short}(t)$ to maintain the diversity of the population (Mavrovouniotis & Yang, 2010, 2013). This immigrants scheme is particularly suitable for the case when environments change abruptly and little information can be transferred from the prior environment. Let $\mathbb{R}$ be the set of $r \times \mathbf{K}_s$ immigrants that are generated randomly and are used to replace ants in $k_{short}(t)$. The replacement rate is denoted by r. When ants are added or removed, the corresponding pheromone trails can be updated using Eqs. (7) and (8). The process continues until a set of termination criterions are satisfied (e.g., maximum time or iteration limit is reached). A pseudo-code for R-ACO is presented in Algorithm 1.

Table 1
Summary of mathematical notations.

Symbol	Description
$k_{long}(t)$	Long-term memory at iteration t
$k_{short}(t)$	Short-term memory at iteration t
$\mathbf{K}_l$	Size of long-term memory
$\mathbf{K}_s$	Size of short-term memory
τ_{max}	Maximum pheromone value
τ_0	Initial pheromone value
T^k	Tour of ant k
r	Immigrant ant replacement rate
t_m	Iteration number at which the next memory update will occur
$\mathbb{R}$	Set of random immigrant ants
$\mathbb{R}_e$	Set of elitism-based immigrant ants
$\mathbb{R}_m$	Set of memory-based immigrant ants
$\mathbf{S}_{bs}$	Global best solution
$\mathbf{S}_{bf}$	Iteration best solution
$\mathbf{X}^e$	Best solution for a given environment
$\|k\|$	Ants population size

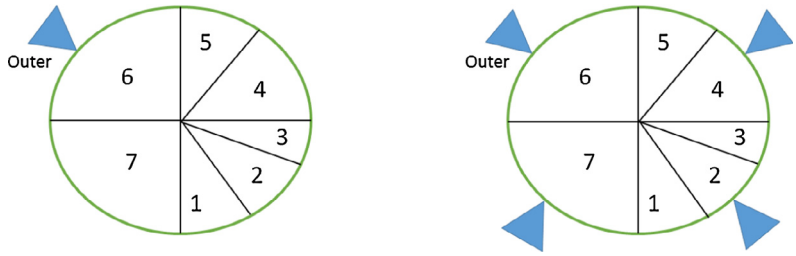
3.2. Elitism-based immigrants ACO (E-ACO)

Under mild environmental changes, R-ACO may misguide the ants and lead to poor computation performance. To alleviate this challenge, a guided diversity scheme, namely, *elitism-based immigrants* ACO (E-ACO) is introduced (Mavrovouniotis & Yang, 2010, 2013). In this scheme, diversity is introduced via transferring knowledge from the best ant of the previous iteration $\mathbf{P}(t-1)$. A pseudo-code for E-ACO is presented in Algorithm 2. Let $\mathbb{R}_e$ be the set of $r \times \mathbf{K}_s$ immigrants that are generated using an inner-over operator presented in Algorithm 3 (Mavrovouniotis & Yang, 2013). The underline concept of Algorithm 3 is to reverse the segment between two selected cities which are then used to replace the ants in $k_{short}(t)$. The process continues until a set of termination criterions are satisfied (e.g., maximum time or iteration limit is reached). Mavrovouniotis and Yang (2013) reports that E-ACO performs well under two specific conditions: (*i*) small changes between the environments and (*ii*) the population has sufficient time to converge to an optimal solution. Therefore, if too much information is transferred to the new environment, then E-ACO may get stuck to a local optimum solution.

3.3. Memory-based immigrants ACO (M-ACO)

Memory-based immigrants (M-ACO) scheme is a generalized version of the elitism-based immigrants scheme (Mavrovouniotis & Yang, 2010, 2013). In M-ACO, not only the best ant from the previous environment $\mathbf{P}(t-1)$, but also the best ants from several previous environments are considered to generate immigrants. Therefore, unlike R-ACO and E-ACO, M-ACO utilizes both short-term $k_{short}(t)$ and long-term $k_{long}(t)$ memories. In M-ACO, $k_{short}(t)$ is created and updated in the same way as R-ACO and E-ACO. However, $k_{long}(t)$ is initially populated with random ants which are later replaced with the *best-so-far* ants obtained from the previous environments. Further, in M-ACO, a metric of how close ant a is to ant b is used to replace old ants with new ants (Mavrovouniotis & Yang, 2013).

$$\mathbf{M}(a,b) = 1 - \left(\frac{E_{ab}}{n}\right) \tag{9}$$



(a) roulette-wheel sampling (b) stochastic universal sampling

Fig. 1. Different sampling strategies used in ALNS.

where n denotes the number of cities and E_{ab} is the number of common edges that exists between the solutions of ant a and b. A value of $\mathbf{M}(a,b)$ closer to 0 means that the ants are more similar to each other and are located nearby (Angus, 2006).

M-ACO follows a unique *ant replacement strategy* over other ACO algorithms, such as P-ACO. Unlike P-ACO, where ant is replaced on every iteration, M-ACO only replaces an ant when a dynamic change occurs in the environment. Therefore, the ants in $k_{long}(t)$ are re-evaluated in each iteration t to detect any change in the environment. Afterwards, the best ant from $k_{long}(t)$ is selected and used as a base to generate a set $\mathbb{R}_m$ of $r \times \mathbf{K}_s$ memory immigrants (using Algorithm 3). A pseudo-code for M-ACO is presented in Algorithm 4. Note that while applying M-ACO, it must be kept in mind that due to noise[1] there may be changes in the environment. As a result, preventive measure must be taken while updating $k_{long}(t)$ in M-ACO. Mavrovouniotis and Yang (2013) adopts an approach which dynamically updates $k_{long}(t)$ instead of updating them after a fixed number of iterations. This is illustrated in Algorithms 4 and 5 where it can be observed that a random number t_m is used to indicate the next updating time for $k_{long}(t)$. For instance, if the memory is updated in iteration t, the next update will occur at iteration $t_m \leftarrow t + rand[\underline{t}_m, \bar{t}_m]$ where $\underline{t}_m$ and $\bar{t}_m$ be the lower and upper limit to generate random numbers. Note that updating in $k_{long}(t)$ can still be made if dynamic changes occur before the pre-specified time as indicated by the random number t_m (shown in Algorithm 5).

3.4. Proposed ALNS-based immigrant schemes (ALNS)

The proposed *Adaptive Large Neighborhood Search* (ALNS)-based immigrant schemes utilize the basic ALNS framework, proposed by Ropke and Pisinger (2006a), to generate a set $\mathbb{R}$ of $r \times \mathbf{K}_s$ immigrants from the ants in $k_{short}(t)$. For each iteration and ant in $k_{short}(t)$, a destroy and an insertion operator, among the set of destroy and repair operators, are used to generate a new ant route. The choice of an operator $o \in \mathcal{O}$ is governed by a *roulette-wheel* mechanism (Ropke & Pisinger, 2006a) or a *stochastic universal sampling* mechanism (Pencheva, Atanassov, & Shannon, 2009) where each operator o is assigned a weight ω_o whose value depends on the prior performance of the operator. Given $|\mathcal{O}|$ operators with weight ω_o, operator o' will be selected with a probability: $\omega'_o / \sum_{o=1}^{|\mathcal{O}|} \omega_o$. Each operator is assigned a weight and a score which are initially set equal to one and zero, respectively. The entire search is divided into a number of segments of ϕ iterations. At the end of each segment, the weights of all the operators $o \in \mathcal{O}$ are computed based on their performance in the last segment. The following scoring mechanism is used at the end of each iteration: the score of a selected operator $o \in \mathcal{O}$ is increased by σ_1, σ_2, and σ_3, respectively if the operator finds a new best solution, better than the incumbent, and not better but still an accepted solution. Clearly, $\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3 \geqslant 0$. The weights are then updated using the following equation:

$$\omega_o := \begin{cases} \omega_o & \text{if } \vartheta_{oo'} = 0 \\ (1-\eta)\omega_o + \eta \pi_o / \zeta_o \vartheta_{oo'} & \text{if } \vartheta_{oo'} \neq 0 \end{cases} \tag{10}$$

where $\eta \in [0,1]$ is a *reaction factor* which determines how quickly the weight adjustment reacts to changes in the operator performance; π_o and ζ_o, respectively represent the score of the operator $o \in \mathcal{O}$ and the number of times operator o is used in the last segment (Ropke & Pisinger, 2006a). Finally, a *normalized factor* $\zeta_o \geqslant 0$ is introduced to reflect the computational efforts needed to solve an operator (Coelho, Cordeau, & Laporte, 2012a, 2012b). This factor is multiplied with $\vartheta_{oo'}$ (a cooling rate factor which is typically $0 < \vartheta_{oo'} < 1$) in Eq. (10) to ensure that the weights of the time consuming operators are decreased and are less likely being selected by the roulette-wheel mechanism or the stochastic universal sampling mechanism. Note that roulette-wheel mechanism selects one operator in each iteration (see Fig. 1(a)). This may introduce biasness in the selection of the operators and may lead to premature convergence. To counter this effect, we further examine the impact of using stochastic universal sampling mechanism in solving DTSP. Note that stochastic universal sampling mechanism initially places multiple pointers in equal distances and selects more than one operator in each iteration compared to roulette-wheel mechanism, as illustrated in Fig. 1(b). Details about the benefits of using one mechanism over the other can be found in Section 4.

We use the same acceptance criterion as in simulated annealing (Ropke & Pisinger, 2006b) to decide whether a solution for an ant in $k_{short}(t)$ should be accepted. The algorithm always accepts an improved solution while an worse solution can still be accepted with a probability $\rho = \exp\left(\frac{-100}{T}\left(\frac{f(s')-f(s)}{f(s)}\right)\right)$, where $f(s)$ and $f(s')$ are the total distance of an ant in the current solution (s) and the new solution (s'), respectively. At iteration t, the temperature can be adjusted by the formula $T = T_t = cT_{t-1}$ and we initially set $T_0 = 0.3$ and $c = 0.995$. The algorithm terminates when one of the following criteria is satisfied: (i) the maximum iteration limit is reached or (ii) temperature drops to a certain limit. The details of the ALNS algorithm are provided in Algorithm 6. The following destroy ($\mathcal{O}^-$) and insertion ($\mathcal{O}^+$) operators ($\mathcal{O} = \mathcal{O}^- \bigcup \mathcal{O}^+$) are used in Algorithm 6 to obtain an ant route.

Destroy operators:

- **Random removal:** This operator randomly removes a node from the tour and is repeated for ρ times. This operator is useful for refining the solution. Depending on the size of ρ, the changes can be significant i.e., if ρ is small, the solution may change slightly, but yields a major transportation when ρ is large.
- **Worst distance removal:** This operator removes the nodes with the highest cost function. The cost function can be defined as

[1] Noises are artificially introduced in DTSP to indicate environmental changes.

the one which relates the distance from the proceeding and succeeding nodes in the tour. Mathematically, the node to be removed will be the one that gives the maximum value for $(d_{ij} + d_{jk})$ where j in the node under consideration and $i \in \mathcal{V}$ and $k \in \mathcal{V}$ are proceeding and succeeding nodes of j, respectively.

- **Proximity based removal:** This operator removes a node that are relatively close to each other. Specifically, it randomly selects one node and compute the minimum distance d_{min} of all the neighboring nodes. The node with the least d_{min} can be removed for the next iteration and the process can be repeated for ρ times.
- **Neighborhood removal:** This operator is based on the idea of deleting nodes that are extreme with respect to the average distance of a route. For instance, let us assume that a path consists of 5 nodes i.e., $R = 1, 2, 3, 4, 5$. Here, the average distance of the route can be expressed as $Average_R = \frac{\sum_{i=1}^{5}\sum_{j=1}^{5} d_{ij}}{5}$. Afterwards, any node, say node z, can be removed from the route if node $z^* := argmax\left\{Average_R - Average_{R\setminus z}\right\}$.

Insertion operators:

- **Random insertion:** Similar to the random removal operator, this operator serves the purpose of diversifying the search. This operator iteratively places a removed node randomly within an ant's sequence on each pass until all nodes have been placed.
- **Greedy insertion:** This operator follows the same principle as *greedy heuristic* where nodes are inserted in the best possible positions of a route. Let, δF_i be the change in distance before and after inserting a node i into the route. The goal of greedy insertion operator is to insert the node in a position of a route where δF_i is minimum.
- **Greedy insertion with noise:** This operator utilizes a *noise function* to determine the best position to insert a node. Let d^* be the distance of a route after inserting a node in its best position by using a greedy heuristic. The new distance $\mathbf{d}^{**}$ of the route for the same node can be computed as follows:

$$\mathbf{d}^{**} = d^* + d_{max} \times \mu \times \eta \tag{11}$$

where d_{max} is the maximum distance between nodes, μ is a diversifying constant (i.e., 0.1), and η is random number which is set between -1.0 and 1.0.

- **Regret insertion:** This operator is an improvement over the *greedy insertion* operator where a look ahead information is added to the insertion heuristic while selecting the insertion position of a node in the route. Let δf_i be the change in distance by inserting node i in a route. We further define δf_{i1} and δf_{i2} be the best and second best position of a node in a route, respectively. In each iteration, the regret insertion operator inserts the node i that maximizes the difference $(\delta f_{i1} - \delta f_{i2})$.
- **Regret insertion with noise:** This operator is an extension over the *regret insertion* operator with the addition of noise function used in the *greedy insertion with noise* operator.
- **Farthest insertion:** This operator selects a node i from a node removal list and then calculate the distance from this node to other nodes of a tour. Let Δf_{iR} be the distance of a tour where R be the number of nodes in the tour. Each node i should be placed in a position for which Δf_{iR} is maximum.
- **Farthest insertion with noise:** This operator is an extension over the *farthest insertion* operator with the addition of noise function used in the *greedy insertion with noise* operator.

To help the readers follow different ALNS-based immigrant schemes proposed in this study, we have used the following notations to represent the algorithms. Details about the computational experiments of the different ALNS-based immigrant schemes are discussed in Section 4.

- ALNS-1: ALNS-based ACO with *roulette wheel selection* strategy inside E-ACO framework (described in Algorithm 7)
- ALNS-2: ALNS-based ACO with *stochastic universal sampling* strategy inside E-ACO framework (described in Algorithm 7)
- ALNS-3: ALNS-based ACO with *roulette wheel selection* strategy inside M-ACO framework (described in Algorithm 8)
- ALNS-4: ALNS-based ACO with *stochastic universal sampling* strategy inside M-ACO framework (described in Algorithm 8)

4. Experimental study

4.1. Parameter settings

This section investigates the performance of the proposed ALNS-based immigrant schemes (ALNS-1, ALNS-2, ALNS-3, ALNS-4) over the peer ACO algorithms, namely R-ACO, E-ACO, and M-ACO. We use the three stationary benchmark STSP instances from TSPLIB,[2] namely, KroA100 (small), KroA150 (medium), and KroA200 (large), to generate instances for the DTSP. Two different variants of the DTSP, namely, *random* and *cyclic* DTSP, are developed to test the variability of the proposed algorithms. The definitions of the *random* and *cyclic* DTSP are provided in Definitions 1 and 2 and are discussed in details in Mavrovouniotis and Yang (2013).

Definition 1 (*Random DTSP*). This variant of the DTSP assumes that the dynamic changes occur randomly i.e., previous environments are not guaranteed to appear again in the future.

Definition 2 (*Cyclic DTSP*). This variant of the DTSP assumes that the dynamic changes occur with a cyclic pattern i.e., previous environments are guaranteed to appear again in the future.

Let f and m be the frequency and magnitude of dynamic change, respectively. This means at each f iterations, degree of environmental change/dynamicity is m. Smaller value of f indicates fast changing environment whereas smaller value of m corresponds to low degree (i.e., low probability) of environmental change. For experimental purposes, we vary $f \in \{10, 25, 50, 100, 200\}$ and $m \in \{0.10, 0.25, 0.75, 0.95\}$ to obtain 40 (20 for each DTSP types) different problem instances. Note that in random DTSP, in every f iterations, a random number $R \in [F_L, F_U]$ is generated to represent potential dynamicity, where F_L and F_U are the lower and upper bounds of the dynamic factor, respectively. As each edge has a probability m to have dynamicity, by generating a different R to represent high and low dynamic factors on different edges, random dynamic behavior is incorporated. On the other hand, in cyclic DTSP, DTSP environments are rotated in a cyclic fashion via a fixed logical ring. This means, at certain iterations a higher probability is given to generate R closer to F_U, whereas a higher probability is given to generate R closer to F_L at any other iteration, and this process is repeated in a cyclic fashion till the end of iterations. The primary difference between random DTSP and cyclic DTSP lies in the fact that in cyclic DTSP incorporation of level of randomness is known whereas in random DTSP it is mostly unknown.

The following offline indicator is used to evaluate the performance of the proposed algorithms (Jin & Branke, 2005):

$$\overline{A}_{Off} = \frac{1}{N'}\sum_{i=1}^{N'}\left(\frac{1}{N}\sum_{j=1}^{N} d_{ij}^*\right) \tag{12}$$

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

where d_{ij}^{*} denotes the distance of the tour of any ant in iteration i of run j. Further, N' and N represent the total number of iterations and number of independent runs, respectively and are set equal to $N' = 600$ and $N = 35$. Table 2 reports the base parameters used for different variants of the ACO algorithms.

4.2. Experimental results

4.2.1. Effectiveness of the proposed immigrant-based ACO algorithms

The first set of experiments compare the offline performance of the proposed ALNS-based immigrant schemes (e.g., ALNS-1, ALNS-2, ALNS-3, and ALNS-4) over the peer algorithms available in the literature, namely, R-ACO, E-ACO, and M-ACO. Tables 3–5 report the performance of the proposed algorithms for different dynamic instances (i.e., random and cyclic) with KroA100, KroA150, and KroA200 dataset, respectively. We vary $f \in \{10, 25, 50, 100, 200\}$ and $m \in \{0.10, 0.25, 0.75, 0.95\}$ in *random* and *cyclic* DTSP to obtain 40 different problem instances. The values highlighted in *bold* in Tables 3–5 indicate the best distance achieved by the "winner" algorithm within a pre-specified iteration limit (see $\overline{A}_{Off}$ in Eq. (12) for details). Apart from that, four additional indicators are added in Tables 3, 4 to enhance their respective readability. These indicators are: total count, average, (%) improvement, and Average (Avg.) improvement. Note that "total count" measures the total instances created for each of the algorithm types, average provides the mean of all the algorithmic solutions except the best one (in bold), (%) improvement demonstrate the improvement that the best algorithm shows over others, and finally average (Avg.) improvement provides an average measure obtained from all the improvement values achieved across all the problem instances. For instance, let us take Table 3 as an example. In Table 3, total count for random pattern is 20. When $f = 10$ and $m = 0.1$, Average provides the mean value of 28,726 considering the best solution obtained via R-ACO, E-ACO, M-ACO, ALNS-2, ALNS-3, and ALNS-4. Note that ALNS-1 is excluded from the calculation of Average as ALNS-1 provides the best solution for that problem instance. Furthermore, (%) improvement is calculated by $\frac{Average\text{-}Best\ known\ solution}{Average} * 100$, that provides a measure for how better the best algorithm (ALNS-1 for random pattern with $f = 10$, $m = 0.1$) performs compared to others. This value has been calculated to be 6.89% in this particular problem instance. Finally, Average (Avg.) improvement demonstrates the average performance improvement across all the problem instances which is 6.78% for random pattern. Note that as ALNS based algorithms always perform better for all problem instances in this study, Average (Avg.) improvement indicator can be interpreted as the overall performance indicator of ALNS-based algorithms as well.

We summarize the findings of this part of the computational study as follows.

- Out of 60 problem instances with varying datasets, f and m values for random pattern, ALNS-1 and ALNS-2 provide best solution in 34 and 26 instances, respectively. On the other hand, for cyclic DTSP, the result is much more evenly distributed. For instance, ALNS-1, ALNS-2, ALNS-3, and ALNS-4 provide best solution in 16, 18, 10, and 16 instances, respectively. Similarly, roulette wheel (ALNS-1 and ALNS-3) selection strategy comparatively provides better quality solutions compared to stochastic universal selection strategy (ALNS-2 and ALNS-4) for random DTSP. On the other hand, for cyclic DTSP, stochastic universal selection performs comparatively better than roulette wheel selection strategy. However, no conclusive evidence is found between the roulette wheel (ALNS-1 and ALNS-3) and stochastic universal (ALNS-2 and ALNS-4) selection strategy within our experimental settings for all DTSPs that unequivocally selects one "winner" over the other.
- Out of 60 problem instances with varying datasets, f, and m values for cyclic pattern, ALNS-1 and ALNS-2 collectively provide best solution in 34 instances while ALNS-3 and ALNS-4 collectively provide best solution in the remaining 26 instances. These results further demonstrate that irrespective of the type of pattern change (i.e., random and cyclic), ALNS-1 and ALNS-2 have the capability to adapt themselves in evolving conditions due to the presence in multiple types of insertion and deletion operators available in the ALNS schemes. Finally, it is observed that ALNS-1 and ALNS-2 are dominant especially in cyclic pattern change and in large datasets (KroA150 and KroA200) with low degree of dynamicity (f), whereas ALNS-3 and ALNS-4 provide better quality solutions under the same experimental settings but for higher degree of dynamicity (f).
- All proposed ALNS-based immigrant schemes (e.g., ALNS-1, ALNS-2, ALNS-3, and ALNS-4) provide faster solutions compared to R-ACO, E-ACO, and M-ACO. However, for brevity comparison in terms of computational time is provided for only two datasets, i.e., KroA100 and KroA200 and with only two values for magnitude of dynamic change, i.e.,$m = 0.25$ and $m = 0.95$. Results show that ALNS based algorithms are on average 75 times faster compared to existing immigrant schemes available in literature. Another key trend that can be observed is that ALNS based algorithms that perform better in Tables 3 and 5, also perform better in terms of CPU time. See Tables 12 and 13 in Appendices B for details where fastest computation time for each problem instance is put in bold.

Careful investigation of the result shows that on average E-ACO outperforms both R-ACO and M-ACO for random DTSP, whereas M-ACO outperforms both R-ACO and E-ACO for cyclic DTSP. More specifically, for random DTSP, E-ACO outperforms both R-ACO and M-ACO when m is smaller and f is larger. This is because using guided immigrants, transferring the knowledge gained from the previous environment to the pheromone trails of the new environment will guide the population of ants to promising areas of the new environment quickly. However, too much passing of information in a shorter time frame (i.e., smaller f) reduces the effectiveness of E-ACO, as exploration of diversity becomes more important. In such scenarios, R-ACO is expected and is observed to perform better. However, as ALNS-1 and ALNS-2 essentially are improved versions of E-ACO, it is observed that they can outperform R-ACO, E-ACO, and M-ACO due to their ability to easily adapt to new environment. On the other hand, M-ACO alongside ALNS-3 and ALNS-4 are more useful in cyclic DTSP due to their capability to direct the population directly to a previously visited environment as they store the best solutions for all cyclic states in its k_{long}, which is useful for cyclic environment. Consequently, as ALNS-3 and ALNS-4 are improved versions of M-ACO; R-ACO, E-ACO, M-ACO, ALNS-1, or ALNS-2 fall short in providing better solution compared to them.

In overall, the results in Tables 3–5 indicate that the proposed ALNS-based immigrant schemes have the potential to outperform

Table 2
Base parameters used in the ACO algorithms.

ACO algorithm	$\|k\|$	α	β	$\mathbf{K}_l$	$\mathbf{K}_s$	r
R-ACO	28	1	5	0	6	0.6
E-ACO	28	1	5	0	6	0.6
M-ACO	25	1	5	3	6	0.6
ALNS-1	28	1	5	0	6	0.6
ALNS-2	28	1	5	0	6	0.6
ALNS-3	25	1	5	3	6	0.6
ALNS-4	25	1	5	3	6	0.6

Table 3
Performance of proposed algorithms for different dynamic instances with KroA100 dataset.

Pattern	f	m	R-ACO	E-ACO	M-ACO	ALNS-1	ALNS-2	ALNS-3	ALNS-4	Average	(%) Improvement
Random	10	0.10	29,992	28,658	29,511	**26,328**	26,372	27,454	27,669	28,726	6.89
		0.25	32,016	31,100	31,748	30,472	**30,184**	31,324	31,556	31,369	3.78
		0.75	33,884	33,287	33,451	**32,214**	32,548	32,789	32,967	33,154	2.84
		0.95	38,451	38,546	39,012	**36,258**	36,854	38,766	38,924	27,356	6.20
	25	0.10	28,956	27,896	28,120	25,985	**25,659**	26,369	26,811	30,256	6.23
		0.25	31,256	29,568	30,846	28,572	**28,371**	30,195	31,102	32,368	1.35
		0.75	33,568	31,956	32,140	**31,931**	31,988	32,073	32,484	29,282	4.89
		0.95	37,850	36,124	36,968	32,472	**32,265**	36,680	37,012	36,184	10.83
	50	0.10	27,150	26,226	26,988	**25,410**	25,950	26,311	26,754	26,563	4.34
		0.25	28,956	27,146	27,968	27,269	**26,891**	27,368	27,245	27,658	2.78
		0.75	30,126	29,540	29,886	27,916	**27,849**	29,869	28,356	29,282	4.89
		0.95	33,568	32,395	32,698	28,803	**28,729**	32,587	32,444	32,082	10.45
	100	0.10	27,008	25,859	26,872	**23,568**	23,968	26,058	26,202	25,994	9.33
		0.25	27,106	26,580	26,594	24,154	**24,103**	26,962	26,547	26,323	8.44
		0.75	29,412	28,360	28,416	25,652	**25,479**	28,654	28,221	28,119	9.39
		0.95	31,112	30,126	31,014	25,994	**25,946**	31,965	32,102	30,385	14.61
	200	0.10	25,568	25,012	25,196	**23,524**	23,549	25,154	25,106	24,930	5.64
		0.25	26,876	25,751	26,482	23,857	**23,824**	25,988	25,459	25,735	7.43
		0.75	26,952	26,104	26,702	**24,849**	25,521	26,586	26,548	26,402	5.88
		0.95	28,102	27,508	27,866	**25,142**	25,269	28,124	28,265	27,522	8.65
Total count					20				Avg. improvement		6.78
Cyclic	10	0.10	28,365	27,393	27,847	26,212	26,229	**26,111**	26,226	27,045	3.45
		0.25	31,214	30,214	30,548	30,129	30,251	28,991	**27,965**	29,901	6.48
		0.75	33,311	32,548	32,618	32,110	32,331	31,957	**30,748**	32,231	4.60
		0.95	36,315	36,547	36,124	36,229	36,215	33,822	**33,620**	35,553	5.44
	25	0.10	28,103	26,129	26,998	25,451	**25,386**	26,098	26,124	26,483	4.15
		0.25	30,665	29,556	30,245	28,465	28,224	29,044	**27,847**	29,149	4.47
		0.75	32,698	31,879	32,122	31,893	31,907	31,558	**30,692**	31,821	3.55
		0.95	36,549	35,352	35,102	32,390	**31,254**	32,025	32,459	33,979	8.02
	50	0.10	27,393	25,885	26,112	**25,106**	25,380	25,847	25,778	26,065	3.68
		0.25	29,659	28,458	28,996	27,056	**26,579**	27,136	27,102	28,067	5.30
		0.75	31,897	30,215	31,015	**27,813**	27,829	28,554	28,551	29,676	6.28
		0.95	35,376	34,225	33,879	28,643	28,431	**28,172**	28,935	31,581	10.80
	100	0.10	26,684	25,336	25,889	**23,345**	23,888	25,789	25,515	25,516	8.51
		0.25	28,549	27,657	27,469	24,082	**24,055**	26,854	26,459	26,845	10.39
		0.75	30,325	29,559	29,221	**24,875**	25,372	28,479	28,116	28,512	12.76
		0.95	34,215	33,268	32,243	**25,872**	25,896	31,958	28,412	30,998	16.54
	200	0.10	25,879	24,987	24,998	23,256	**23,129**	24,899	23,251	24,545	5.77
		0.25	27,818	26,019	26,879	23,666	**23,588**	26,102	25,272	25,959	9.13
		0.75	29,548	28,997	28,659	**24,564**	25,125	26,116	25,892	27,389	10.32
		0.95	33,369	32,261	31,669	25,029	**24,993**	28,011	27,968	29,717	15.90
Total count					20				Avg. improvement		7.78

some well known peer ACO algorithms, namely, R-ACO, E-ACO, and M-ACO under the given experimental settings.

4.2.2. Effect of replacement rate r on the immigrant-based ACO algorithms

This set of experiments study the effect of immigrant replacement rate r over the proposed ALNS-based immigrant schemes. To perform these experiments, we vary $r \in \{0.0, 0.2, 0.4, 0.6, 0.8\}, f \in \{10, 50, 200\}$, and $m \in \{0.10, 0.25, 0.75, 0.95\}$ to obtain 120 different problem instances (60 each for random and cyclic environments) as reported in Tables 6 and 7. Setting $\mathbf{K}_s = 5, r = 0.0, 0.2, 0.4, 0.6$, and 0.8 imply generation of zero, one, two, three, and four immigrant ants, respectively ($r \times \mathbf{K}_s$). We select ALNS-1 and ALNS-2 for random DTSP (shown in Table 6) and ALNS-3 and ALNS-4 for cyclic DTSP (shown in Table 7) due to their superiority in solving the respective instance sets as demonstrated in Tables 3–5. Dataset KroA100 is considered as a testing ground for this set of experiments. Results in Table 6 indicate that no specific preference can be made between *roulette wheel* (ALNS-1) and *stochastic universal* selection strategy (ALNS-2) in solving random DTSP. Out of 60 instances, ALNS-1 and ALNS-2 outperformed each other by 33 and 27 problem instances, respectively. However, the pattern changed completely for cyclic DTSP where it can be observed that ALNS-3 outperformed ALNS-4 by only 13/60 problem instances. Results indicate that selecting multiple operators, as in the case with *stochastic universal* selection strategy, demonstrate superiority in solving DTSP's that follow cyclic patterns.

4.2.3. Impact of ALNS operators on the proposed immigrant-based ACO algorithms

We now analyze the impact of individual operators (insertion and deletion operators) on the performance of the ALNS-based immigrant schemes. The insertion ($\mathcal{O}^+$) and deletion ($\mathcal{O}^-$) operators used in the ALNS-based immigrant schemes are summarized in Table 8. Experimental results in Tables 9 and 10 are reported based on the following settings: KroA100 dataset; $f \in \{10, 100\}$; random DTSP; and only ALNS-1 algorithm is used for demonstration purposes. The first set of experiments, reported in Table 9, study the changes in solution quality ($\overline{A}_{Off}(\%)$) due to the removal of an ALNS operator. Experimental results indicate that when $f = 10$, removing deletion operator: *worst distance removal* and

Table 4
Performance of proposed algorithms for different dynamic instances with KroA150 dataset.

Pattern	f	m	R-ACO	E-ACO	M-ACO	ALNS-1	ALNS-2	ALNS-3	ALNS-4	Average	(%) Improvement
Random	10	0.10	41,446	40,125	41,124	**39,124**	39,326	40,325	40,224	40,428	3.23
		0.25	43,458	42,541	42,981	40,993	**39,869**	42,968	42,889	40,638	6.49
		0.75	48,598	48,146	48,151	46,128	**45,669**	48,136	48,269	47,905	4.67
		0.95	53,508	53,778	54,213	**51,218**	51,369	53,651	53,996	53,419	4.12
	25	0.10	40,331	39,108	39,872	**37,996**	38,459	39,354	39,226	39,392	3.54
		0.25	42,764	41,808	42,352	41,021	**40,996**	41,995	41,857	41,966	2.31
		0.75	47,585	47,121	47,489	**44,256**	44,445	47,268	47,326	46,872	5.58
		0.95	53,066	53,008	53,262	48,659	**46,621**	49,965	49,811	51,295	9.11
	50	0.10	38,784	37,982	38,024	34,758	**34,669**	38,165	38,226	37,657	7.93
		0.25	41,215	40,159	41,018	36,554	**36,547**	40,658	40,649	40,042	8.73
		0.75	44,585	42,887	43,558	**39,784**	39,985	43,661	43,668	43,057	7.60
		0.95	50,128	49,884	49,982	42,587	**42,496**	49,959	49,833	48,729	12.79
	100	0.10	36,578	34,796	35,145	**32,552**	32,654	33,871	33,954	34,500	5.65
		0.25	38,758	35,487	37,854	**33,257**	33,529	34,968	35,065	35,944	7.47
		0.75	42,548	37,136	38,992	**34,987**	35,369	36,557	36,852	37,909	7.71
		0.95	47,859	39,102	42,846	38,528	**38,365**	40,965	39,764	41,511	7.58
	200	0.10	34,158	32,546	33,439	**31,024**	31,568	31,685	31,559	32,493	4.52
		0.25	36,847	35,214	35,829	**34,125**	34,859	34,765	34,559	35,346	3.45
		0.75	40,887	37,552	38,859	**34,589**	35,039	36,442	36,374	37,526	7.83
		0.95	45,857	41,254	43,319	**38,458**	36,829	38,664	39,558	40,914	6.00
Total count						20			Avg. improvement		6.32
Cyclic	10	0.10	40,236	38,452	39,458	38,859	**38,331**	39,578	39,856	39,407	2.73
		0.25	42,398	41,558	41,857	**39,543**	39,664	41,859	41,933	41,545	4.82
		0.75	48,258	47,211	47,101	45,226	**44,893**	47,996	47,838	47,272	5.03
		0.95	53,077	53,102	52,968	**50,266**	50,632	52,870	53,526	52,696	4.61
	25	0.10	38,458	37,933	37,996	**37,854**	37,896	39,126	38,961	38,395	1.41
		0.25	41,268	40,058	40,112	39,385	**39,172**	41,226	41,224	40,546	3.39
		0.75	46,115	45,986	45,344	43,955	**43,870**	47,006	47,108	45,919	4.46
		0.95	49,635	52,198	51,885	48,332	**46,785**	49,584	52,004	50,606	7.55
	50	0.10	38,144	37,112	37,345	34,559	**34,216**	37,945	38,415	37,253	8.15
		0.25	40,069	39,616	39,551	36,558	**36,411**	40,529	40,428	39,459	7.72
		0.75	43,689	42,105	41,849	39,448	**38,119**	42,889	43,582	42,260	9.80
		0.95	49,359	47,228	46,424	**41,659**	42,105	48,849	49,446	47,235	11.81
	100	0.10	35,127	34,501	34,445	32,549	32,541	32,527	**32,503**	33,615	3.31
		0.25	37,549	35,410	35,157	33,155	33,409	34,659	**33,058**	34,890	5.25
		0.75	38,528	36,991	36,738	35,486	35,128	35,196	**35,049**	36,345	3.56
		0.95	43,651	38,748	38,365	38,524	38,272	38,354	**38,129**	39,319	3.03
	200	0.10	33,265	31,654	31,254	30,945	31,289	**30,549**	30,866	31,546	3.16
		0.25	34,869	33,482	33,216	33,266	33,010	32,214	**32,015**	33,343	3.98
		0.75	37,698	35,268	34,254	34,210	34,282	**33,168**	33,426	34,856	4.84
		0.95	40,125	37,198	36,802	36,533	35,618	35,533	**35,452**	36,968	4.10
Total count						20			Avg. improvement		5.14

insertion operators: *regret insertion with noise*, *farthest insertion*, and *farthest insertion with noise* significantly drop the offline solution quality in solving DTSP. However, removing deletion operator: *random removal* and insertion operators: *random insertion*, *regret insertion with noise*, *farthest insertion*, and *farthest insertion with noise* significantly drop the offline solution quality when $f = 100$. Note that ALNS-1 uses a number of other insertion and removal operators (e.g., greedy insertion, greedy insertion with noise, regret insertion) whose removal essentially turn out to be an improvement in the solution quality. This indicates that ultimately only a handful number of operators are truly responsible for maintaining and/or increasing the solution quality. However, it is not possible to generalize which particular set of operators would be beneficial toward obtaining better solution quality since the selection of the operators are heavily dependent on frequency (f) and magnitude (m) of environmental changes.

The next set of experiments, reported in Table 10, test whether removing any particular pair of operators have any significant impact on the solution quality of the proposed ALNS-1 algorithm. The results further reveal that no particular generalization can be made regarding the selection of the pair of operators to remove. For instance, when both *random removal* destroy operator and *random insertion* repair operator are removed, the solution quality increases significantly for $f = 10$ but drops when $f = 100$. However, when both *neighborhood removal* destroy operator and *regret insertion* repair operator are removed, the solution quality drops significantly for $f = 10$ but increases when $f = 100$. In summary, selection of the insertion and deletion operators is heavily dependent on the frequency (f) and magnitude (m) of the environmental changes in DTSP and hence, cannot be generalized. Note that although the experiments for this particular sensitivity analysis is performed for Kro A100 dataset with random DTSP, it shows similar trend across all the datasets, ALNS algorithms, and DTSP types (i.e., cyclic). One key insight that can be drawn from these experiments is that if an operator individually provides a deteriorated solution, combining it with other similar effect operators further worsens the solution quality. The combined effect is always far worse compared to their individual effect. Same trend has been observed for both random and cyclic patterns. However, unless exhaustive simulation studies are performed to determine which combination provide better or worse quality results, it is not prudent to generalize and declare certain operators to be better per-

Table 5
Performance of proposed algorithms for different dynamic instances with KroA200 dataset.

Pattern	f	m	R-ACO	E-ACO	M-ACO	ALNS-1	ALNS-2	ALNS-3	ALNS-4	Average	(%) Improvement
Random	10	0.10	46,993	44,924	45,998	**43,754**	43,998	44,872	44,669	45,242	3.29
		0.25	51,684	49,857	50,211	45,914	**45,865**	50,896	50,362	49,821	7.94
		0.75	58,374	55,487	56,354	50,533	**50,368**	55,869	55,298	55,319	8.95
		0.95	64,234	61,874	63,681	**54,652**	55,368	62,846	61,485	61,581	11.25
	25	0.10	44,100	42,125	42,998	**40,859**	41,135	42,354	42,884	42,599	4.09
		0.25	48,254	46,457	47,025	**42,556**	42,688	47,326	47,935	46,614	8.71
		0.75	53,124	52,114	52,995	46,378	**46,259**	49,622	54,228	51,410	10.02
		0.95	58,978	57,954	57,994	**53,102**	53,622	54,669	55,941	56,526	6.06
	50	0.10	42,259	40,958	41,775	**36,458**	36,994	38,762	38,983	39,995	8.75
		0.25	46,885	42,766	45,325	40,254	**39,857**	40,655	41,102	42,831	6.94
		0.75	50,025	48,996	49,356	**43,365**	43,855	45,136	45,867	47,206	8.14
		0.95	53,142	51,974	52,658	47,859	**47,393**	49,545	49,988	50,861	6.82
	100	0.10	41,015	40,821	40,958	36,910	**36,859**	37,322	38,958	39,331	6.28
		0.25	44,254	42,668	43,215	39,965	**39,311**	39,559	40,885	41,758	5.86
		0.75	46,598	44,584	45,129	**42,795**	43,012	42,997	44,746	44,511	3.86
		0.95	48,758	46,958	47,441	**46,017**	46,825	46,986	46,872	47,307	2.73
	200	0.10	40,012	40,254	40,921	**35,921**	36,353	36,256	36,445	38,023	5.53
		0.25	43,847	42,187	42,778	**37,458**	38,256	38,239	38,785	40,221	6.87
		0.75	45,872	43,254	45,022	**39,371**	39,421	39,951	39,729	41,803	5.82
		0.95	48,012	45,329	47,201	**42,098**	43,102	42,872	42,742	44,479	5.35
Total count						20			Avg. improvement		6.66
Cyclic	10	0.10	44,546	42,625	42,254	**40,587**	43,979	44,821	44,523	43,791	7.32
		0.25	47,391	49,254	47,523	45,888	**45,763**	48,556	50,122	48,122	4.90
		0.75	55,263	52,328	52,514	**50,512**	55,844	55,498	55,211	54,443	7.22
		0.95	62,834	58,648	58,569	**54,651**	55,100	62,540	60,225	59,653	8.38
	25	0.10	43,612	40,124	42,211	**40,051**	41,108	42,108	42,773	41,989	4.62
		0.25	46,873	43,292	46,254	**42,510**	42,554	47,229	47,892	45,682	6.94
		0.75	54,268	52,067	51,359	46,336	**46,108**	49,368	54,126	51,254	10.04
		0.95	58,549	57,239	56,964	**53,009**	53,529	54,549	55,893		
	50	0.10	41,872	40,020	40,991	36,412	36,872	36,283	**36,115**	38,742	6.78
		0.25	46,121	43,106	45,103	40,153	**39,452**	40,028	40,059	42,428	7.01
		0.75	49,685	48,829	48,124	43,221	43,744	**43,108**	43,126	46,112	6.53
		0.95	52,392	51,106	51,053	47,102	47,869	**47,003**	47,999	49,587	5.21
	100	0.10	40,871	39,873	39,215	37,875	38,104	**36,102**	36,988	38,821	7.00
		0.25	43,356	42,315	42,159	39,892	40,105	39,488	**38,857**	41,219	5.73
		0.75	45,125	44,103	43,648	42,593	42,668	42,547	**39,986**	43,447	7.97
		0.95	47,221	46,108	45,326	44,025	43,897	43,257	**42,169**	44,972	6.23
	200	0.10	40,629	39,847	38,349	35,879	36,224	36,129	**35,506**	37,843	6.18
		0.25	42,993	41,266	41,015	37,432	37,985	**36,998**	37,129	39,637	6.66
		0.75	44,867	43,030	42,211	38,918	39,128	**38,785**	38,887	41,174	5.80
		0.95	46,996	44,991	44,852	41,556	43,225	**38,669**	38,948	43,428	10.96
Total count						20			Avg. improvement		6.85

Table 6
Offline performance of ALNS-1 and ALNS-2 on varying r in random DTSP.

f	$r, m \Rightarrow$	ALNS-1				ALNS-2			
		0.1	0.25	0.75	0.95	0.1	0.25	0.75	0.95
10	0.0	**26,352**	30,492	**32,231**	**36,279**	26,388	**30,195**	32,599	36,868
	0.2	**26,339**	30,483	**32,224**	36,269	26,398	**30,192**	32,568	**35,859**
	0.4	**26,321**	30,478	**32,218**	36,262	26,366	**30,191**	32,551	**35,851**
	0.6	**26,328**	30,472	**32,214**	**36,258**	26,372	**30,184**	32,548	36,854
	0.8	**26,366**	30,497	**32,244**	**36,282**	26,393	**30,206**	32,609	36,889
50	0.0	**25,432**	27,281	27,921	28,818	25,989	**26,912**	**27,889**	**28,768**
	0.2	**25,421**	27,279	27,913	28,811	25,961	**26,897**	**27,863**	**28,745**
	0.4	**25,417**	27,274	27,912	28,808	25,948	**26,895**	**27,853**	**28,737**
	0.6	**25,410**	27,269	27,916	28,803	25,950	**26,891**	**27,849**	**28,729**
	0.8	**25,433**	27,282	27,928	28,817	25,978	**26,907**	**27,884**	**28,773**
200	0.0	**23,548**	23,879	**24,866**	**25,159**	23,577	**23,849**	25,547	25,281
	0.2	**23,526**	23,861	**24,847**	**25,151**	23,558	**23,826**	25,539	25,272
	0.4	**23,539**	23,853	**24,858**	**25,144**	23,552	**23,829**	25,536	25,267
	0.6	**23,524**	23,857	**24,849**	**25,142**	23,549	**23,824**	25,521	25,269
	0.8	**23,555**	23,871	**24,871**	**25,159**	23,587	**23,856**	25,552	25,287

Table 7
Offline performance of ALNS-3 and ALNS-4 on varying r in cyclic DTSP.

f	$r, m \Rightarrow$	ALNS-3				ALNS-4			
		0.1	0.25	0.75	0.95	0.1	0.25	0.75	0.95
10	0.0	**26,199**	29,029	31,998	33,869	26,296	**28,026**	**30,795**	**33,704**
	0.2	**26,115**	29,009	31,979	33,847	26,251	**27,991**	**30,778**	**33,668**
	0.4	**26,102**	28,997	31,966	33,858	26,248	**27,978**	**30,756**	**33,647**
	0.6	**26,111**	28,991	31,957	33,822	26,226	**27,965**	**30,748**	**33,620**
	0.8	**26,184**	29,038	31,991	33,901	26,267	**28,031**	**30,784**	**33,721**
50	0.0	25,921	**27,198**	28,617	**28,229**	**25,877**	27,237	**28,591**	28,975
	0.2	25,877	27,145	28,584	**28,205**	**25,819**	**27,125**	**28,574**	28,953
	0.4	25,859	27,166	**28,541**	**28,191**	**25,784**	**27,118**	28,559	28,947
	0.6	25,847	27,136	28,554	**28,172**	**25,778**	**27,102**	**28,551**	28,935
	0.8	25,958	**27,105**	28,607	**28,341**	**25,866**	27,154	**28,596**	28,977
200	0.0	24,954	26,148	27,155	31,054	**23,124**	**25,321**	**25,914**	**25,998**
	0.2	24,937	26,134	27,135	31,038	**23,084**	**25,305**	**25,888**	**27,979**
	0.4	24,918	26,114	27,119	31,025	**23,069**	**25,288**	**25,904**	**27,961**
	0.6	24,899	26,102	27,116	31,011	**23,051**	**25,272**	**25,892**	**27,968**
	0.8	24,958	26,151	27,141	31,051	**23,116**	**25,335**	**25,918**	**28,014**

Table 8
Representation of the ALNS operators.

ALNS Operator	Number
Destroy operators ($\mathcal{O}^-$):	
Random removal	1
Worst distance removal	2
Proximity based removal	3
Neighborhood removal	4
Insertion operators ($\mathcal{O}^+$):	
Random insertion	5
Greedy insertion	6
Greedy insertion with noise	7
Regret insertion	8
Regret insertion with noise	9
Farthest insertion	10
Farthest insertion with noise	11

forming than others. Note that the goal of Section 4.2.3 is to demonstrate different operator combinations have varying effect on solution quality, rather than attempting to provide a "ground truth" on the operators capability.

5. Case study

This section presents a real-life case study that demonstrate the applicability of the proposed ALNS-based immigrant schemes in solving a DTSP. To do so, we have developed a case concerning the use of *drones* for *wildlife surveillance*. Wildlife surveillance has been universally recognized over the last decade due to increasing pace in spread of wildlife diseases (see e.g., Garcia-Parra & Tapia, 2014; Grogan et al., 2014; Kuiken, Ryser-Degiorgis, Gavier-Widen, & Gortázar, 2011; Ryser-Degiorgis, 2013) and rising need for monitoring distribution animals because of rapid ecosystem shifts mostly due to pollution (see e.g., Brinkman, Deperno, Jenks, Haroldson, & Osborn, 2005). Traditional methods for surveillance of wildlife includes but not limited to tagging, using GSP collar, and conducting camera surveys (Webb, Gee, Strickland, Demarais, & DeYoung, 2010; Webb, Riffell, Gee, & Demarais, 2009). These methods in general are shown to be effective in action, yet researchers have long been emphasizing on the need of developing advanced wildlife surveillance methods due to their serious drawbacks, such as potential to harm animals and lack of accuracy. Therefore, a more secure and accurate form of surveillance methods are needed to monitor wildlife effectively (Goetzman, 2011). Use of drones in wildlife surveillance becoming more common and convenient in recent years (Hodgson, Baylis, Mott, Herrod, & Clarke, 2016; Treacy, 2016; The conversation, 2016). Therefore, in our case study, we investigate the use of drones in wildlife surveillance particularly centered on deer population in Mississippi.

As drones are slowly becoming more common, there has been a growing trend in the use of drones for wildlife conservation purposes; from watching out for poachers in Africa to looking for orangutans in Indonesia. Drones are able to monitor areas that are out of reach for humans. Further, they have a much wider view than that of someone on the ground (Chowdhury, Emelogu, Marufuzzaman, Nurre, & Bian, 2017; Treacy, 2016). Drones are capable of flying in lower altitudes and this enable them to collect

Table 9
Change in solution quality (in %) due to the removal of the ALNS operators.

	$f = 10$					$f = 100$			
	$\overline{A}_{Off}(\%)$					$\overline{A}_{Off}(\%)$			
Operator	$m \Rightarrow$0.1	0.25	0.75	0.95	Operator	$m \Rightarrow$0.1	0.25	0.75	0.95
1	0.43	0.59	0.39	0.48	1	−0.39	−0.82	−0.22	−0.18
2	−0.18	−0.23	−0.22	−0.19	2	0.31	0.16	0.19	0.18
3	−0.02	−0.60	0.06	0.03	3	−0.03	−0.44	0.11	0.09
4	−0.17	0.09	0.40	−0.20	4	0.10	0.11	0.42	0.22
5	0.66	0.22	0.54	0.90	5	−0.28	−0.16	−0.15	−0.13
6	0.69	0.72	0.53	0.33	6	0.37	0.48	0.37	0.28
7	0.48	0.63	0.26	0.31	7	0.22	0.39	0.23	0.19
8	−0.33	0.05	0.46	0.39	8	0.28	0.12	0.13	0.11
9	−0.22	−0.36	−0.35	−0.11	9	−0.16	−0.12	−0.14	−0.22
10	−0.12	−0.14	−0.23	−0.18	10	−0.17	−0.23	−0.21	−0.11
11	−0.15	−0.22	−0.15	−0.33	11	−0.24	−0.17	−0.17	−0.22

Table 10
Change in solution quality (in %) due to the removal of two ALNS operators.

$f = 10$						$f = 100$					
Removal	Insertion	$\overline{A}_{Off}(\%)$				Removal	Insertion	$\overline{A}_{Off}(\%)$			
Operator	Operator	$m \Rightarrow 0.1$	0.25	0.75	0.95	Operator	Operator	$m \Rightarrow 0.1$	0.25	0.75	0.95
1	5	0.97	0.82	0.93	0.113	1	5	−0.13	−0.3	−0.5	−0.8
	6	0.62	0.82	0.87	0.62		6	0.00	0.1	−0.2	0.00
	7	0.65	0.69	0.79	0.52		7	0.2	0.3	0.2	−0.1
	8	0.94	0.57	0.79	0.50		8	0.9	0.6	0.8	0.5
	9	0.68	0.93	0.43	0.40		9	−0.2	−0.4	−0.2	−0.4
	10	0.42	0.68	0.60	0.29		10	−0.4	−0.7	−0.6	−0.3
	11	0.15	0.69	0.74	0.45		11	−0.2	−0.7	−0.7	−0.4
2	5	0.06	0.91	0.99	0.14	2	5	0.1	0.9	0.1	0.1
	6	0.50	0.59	0.02	0.52		6	0.5	0.6	0.00	0.5
	7	0.29	0.97	0.37	0.23		7	0.3	0.10	0.4	0.2
	8	−0.76	−0.78	0.34	−0.01		8	−0.8	−0.8	0.3	0.00
	9	−0.93	−0.31	−0.01	−0.20		9	−0.9	−0.3	0.00	−0.02
	10	−0.26	−0.09	−0.95	−0.89		10	0.03	−0.1	−0.10	0.9
	11	−0.77	−0.96	−0.44	−0.01		11	−0.08	−0.10	−0.4	0.00
3	5	0.012	0.64	0.06	0.09	3	5	−0.01	−0.6	−0.3	−0.4
	6	0.08	0.38	0.57	0.74		6	0.01	0.4	0.6	0.7
	7	0.07	0.51	0.76	0.53		7	0.01	0.1	0.7	0.5
	8	−0.17	−0.72	−0.47	−0.21		8	−0.2	−0.7	0.5	−0.2
	9	−0.100	−0.48	0.59	0.20		9	−0.10	−0.5	−0.2	−0.1
	10	−0.20	−0.13	−0.58	−0.63		10	−0.2	0.00	−0.6	0.00
	11	−0.27	−0.19	−0.68	−0.24		11	−0.3	−0.2	−0.7	−0.2
4	5	0.89	0.62	0.92	0.49	4	5	−0.9	0.6	0.9	0.5
	6	0.25	0.20	−0.57	0.08		6	0.3	0.2	0.6	0.9
	7	0.39	0.19	0.02	0.43		7	0.4	0.2	0.00	0.4
	8	−0.74	−0.39	−0.99	−0.09		8	0.7	0.4	0.10	0.1
	9	−0.75	−0.86	−0.20	−0.51		9	−0.7	−0.9	0.2	−0.5
	10	−0.04	−0.98	0.91	−0.93		10	0.00	−0.10	0.9	−0.09
	11	−0.06	0.70	−0.34	−0.55		11	−0.10	0.7	−0.3	−0.25

more accurate ground information than even the satellites. Satellites, in general, are very powerful but the image quality can be significantly hampered by the presence of clouds in the sky (UNEP, 2013). In a relatively recent study, Hodgson et al. (2016) demonstrate that drones are much more precise at monitoring the size of wildlife than more traditional ground counts. Not only their exceptional precision, but their ability to survey hard-to-reach populations makes drones an effective tool in wildlife monitoring projects. Drones have also been used for wildlife conservation especially in reducing illegal animal trade that is approximately a $10 billion industry. Since 2013, drones have also been used to reduce poaching of rhinos in Africa (The conversation, 2016). In this specific application, researchers develop analytical models to determine the potential location of animals at different parts of the day and night and send drones to those nodes specifically to monitor the region. Since the inception of integrating drone technology into the wildlife monitoring network, poaching incidents in the Hluhluwe-Imfolozi Park in KwaZulu-Natal is reduced by 92% (SouthAfrica.info., 2014).

5.1. Data description

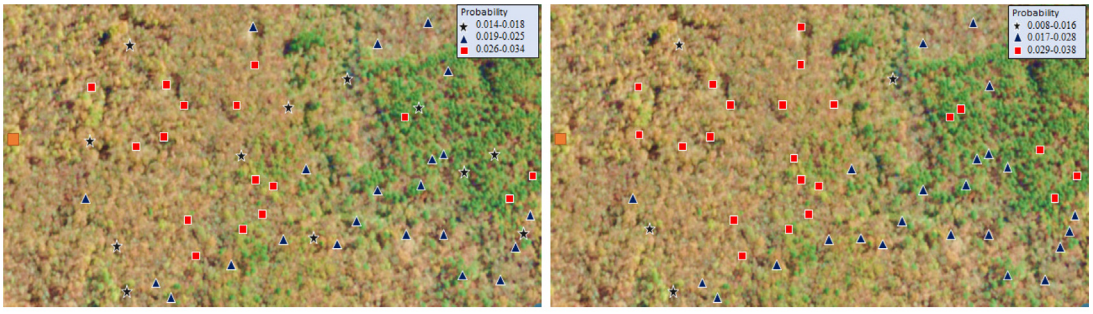
We choose Sam D. Hamilton Noxubee National Wildlife Refuge[3] because of its rich deer population as well as geographical proximity. Battery life is a crucial parameter that can significantly affect the utilization of drones in wildlife surveillance. Due to limited battery life consideration of the drone, we consider a 1.86 square mile area within this refuge. For experimental purposes, we consider a drone with 17 lb payload capacity, battery life of 12.43 miles, and flight time of 45 min (Airborne drones, 2016).

[3] Available from: https://www.fws.gov/refuge/Noxubee/.

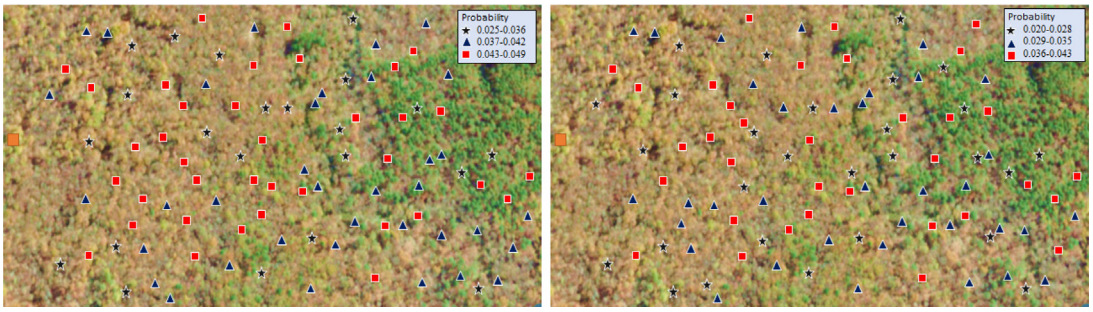
Peak activity times for deer are their feeding times at dawn as well as dusk. In the absence of predators, deer movement patterns are primarily governed by water and food acquisition (Massé & Côté, 2009). On the other hand, deer usually keep away from areas where they are more exposed to potential threats. Although the exact location of a deer population in a given time period cannot be known in advance, it is not difficult to make an estimate on the basis of movement trends and habitat selection. Here, we follow a probabilistic approach where we provide an estimate for probability of finding a deer in a given time and location. To do so, we randomly generate a set of nodes within the studied area and then examine each of these nodes to determine their proximity to water, roads, cover, and other things that would either attract or repel deer. We use the deer expert opinions from the Mississippi State University Deer Lab (http://www.msudeerlab.com/index.asp) to determine the likelihood of deer in a given location at different time periods of the day (e.g., feeding and non-feeding times of deer). The responses are then collected in a scale of 1–10, with 1 being lowest and 10 being the highest chances, and then normalized to determine an approximate probability of deer for a given time and location. Fig. 2 presents the probability estimation of deer in different of their feeding and non-feeding times. We generate 50 (Fig. 2(a) and (b)) and 100 (Fig. 2(c) and (d)) different hypothetical nodes in our testing region where the drones can be routed to maximize the number of deer surveyed.

5.2. Experimental results

We now present a set of experimental results demonstrating how drones can be used to survey deer in a selected area of Sam D. Hamilton Noxubee National Wildlife Refuge in Mississippi. Due to the mobility nature of deer, the problem on hand can be considered as a special case of a DTSP where the surveying time



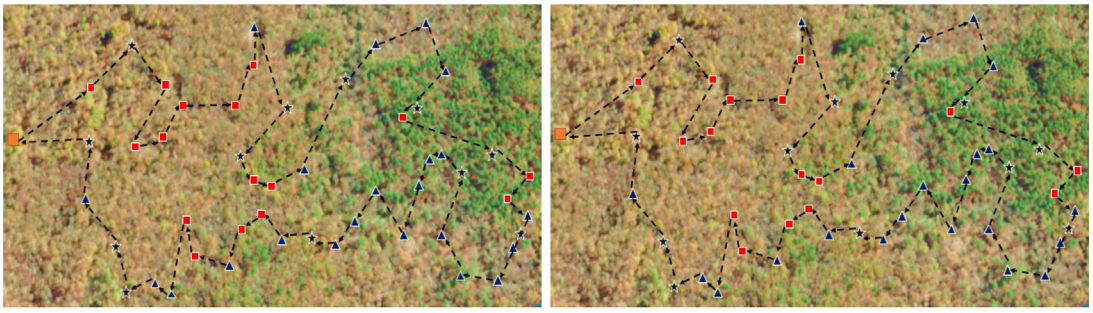
(a) Feeding time (50 nodes) (b) Non-feeding time (50 nodes)



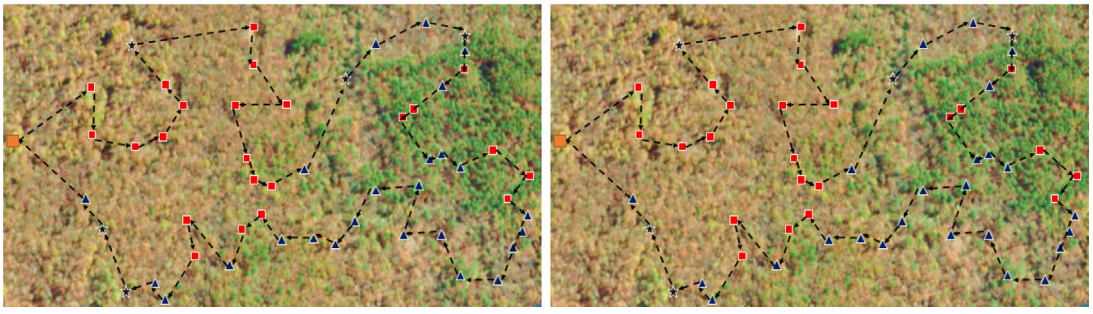
(c) Feeding time (100 nodes) (d) Non-feeding time (100 nodes)

Fig. 2. Feeding and non-feeding time probability estimation for deer (50 and 100 nodes).

of a node by drones vary significantly depending on the availability of deer on that given node. Further, the probability of servicing a node changes dynamically given that the deer may leave/join to/from the neighboring nodes. We assume that a drone may spend more time in a node considering the availability of deer on that given node i.e., more service time required due to capturing additional images for deer and vice versa. We realized from Fig. 2 that the availability of deer in different locations vary significantly



(a) STSP (feeding time) (b) DTSP (feeding time)

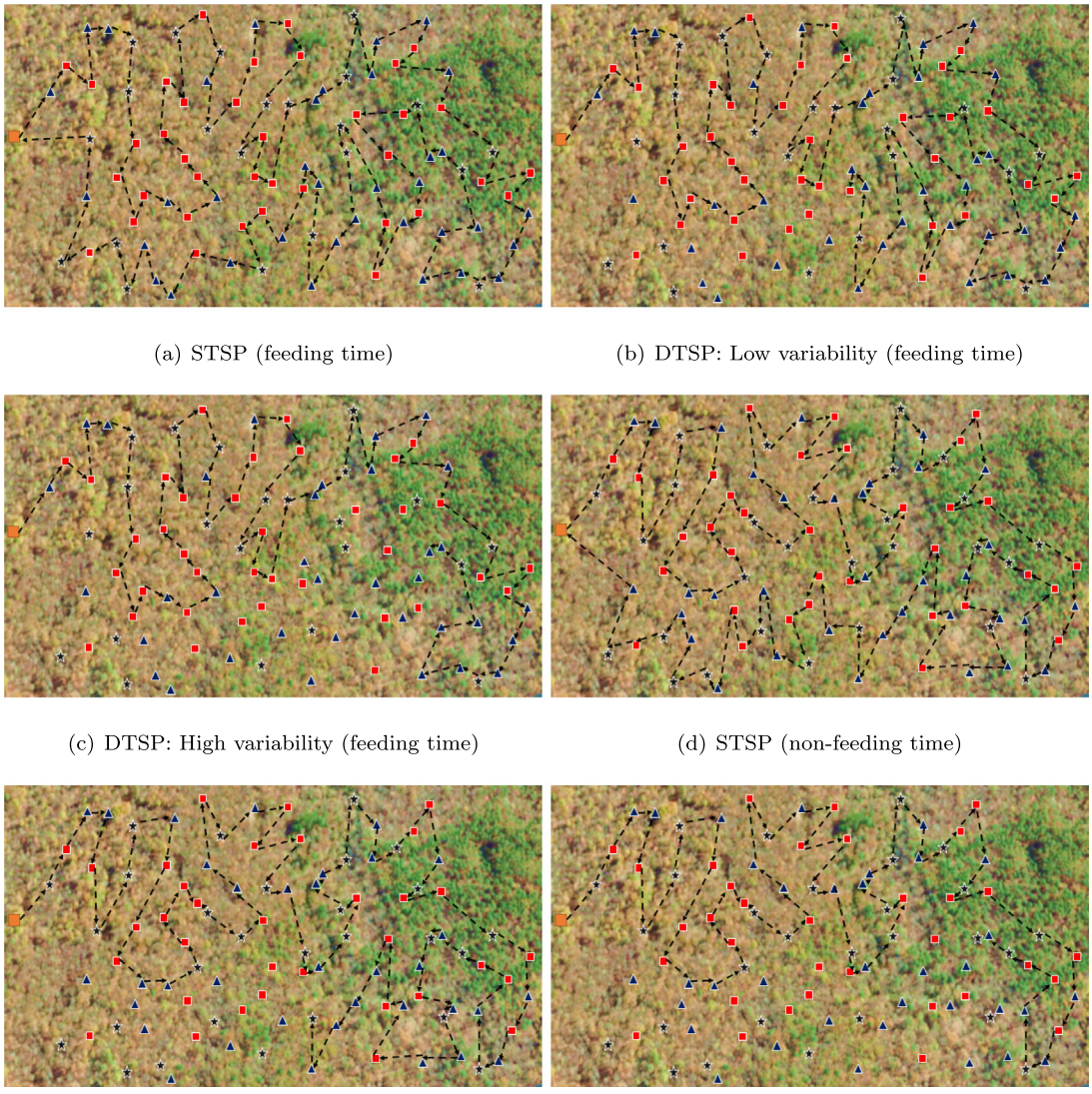


(c) STSP (non-feeding time) (d) DTSP (non-feeding time)

Fig. 3. Drone routing in STSP and DTSP for 50 nodes.

during feeding and non-feeding times of deer in a day. Thus, different drone routing strategies are needed in different time periods of the day (i.e., feeding vs. non-feeding time) to accurately survey deer in the tested area. Figs. 3 and 4 illustrate the drone routing paths for 50 and 100 points, respectively. Note that we use ALNS-1 to generate the solutions for DTSP.

Table 11 summarizes the experimental results used in this section. Clearly, the routes for drone are not significantly impacted



(a) STSP (feeding time)

(b) DTSP: Low variability (feeding time)

(c) DTSP: High variability (feeding time)

(d) STSP (non-feeding time)

(e) DTSP: Low variability (non-feeding time)

(f) DTSP: High variability (non-feeding time)

Fig. 4. Drone routing in STSP and DTSP for 100 nodes.

Table 11
Summary of experimental results.

#nodes	TSP	Figure	Time	Distance (miles)	#nodes served
50	STSP	3(a)	feeding	6.24	50
	DTSP	3(b)	feeding	7.98	50
	STSP	3(c)	non-feeding	6.87	50
	DTSP	3(d)	non-feeding	8.10	50
100	STSP	4(a)	feeding	11.97	100
	DTSP: Low variability	4(b)	feeding	12.41[a]	85
	DTSP: High variability	4(c)	feeding	12.39[a]	65
	STSP	4(d)	non-feeding	12.11	100
	DTSP: Low variability	4(e)	non-feeding	12.40[a]	81
	DTSP: High variability	4(f)	non-feeding	12.37[a]	65

[a] Out of range.

when surveying deer for only 50 nodes. It is observed that drone is capable of serving all 50 nodes within the drone flying range (for both feeding and non-feeding times). However, drones fly 27.2% more in feeding and 17.9% more in non-feeding times, respectively in DTSP over STSP. The changes in drone serviceability in DTSP over STSP are more pronounced when 100 nodes are taken into consideration. Fig. 4(a) and (d) illustrates that all 100 nodes can be served within the drone flying range in STSP (for both feeding and non-feeding times). However, this is not the case for DTSP where significant portion of the nodes remain unserved under both feeding and non-feeding times. We consider two different service cases for drones: (i) low service variability and (ii) high service variability by setting $\epsilon = 10\%$ and $\epsilon = 20\%$, respectively and generating them uniformly for each node $i \in \mathcal{V}$ using $[\bar{s}_i(1-\epsilon), \bar{s}_i(1+\epsilon)]$. Note that $\bar{s}_i$ represents mean service time by a drone at each node $i \in \mathcal{V}$ of the service area which we set $\bar{s}_i = 25$ s. As discussed earlier, the variation in $\bar{s}_i$ occurs due to the arrival or leaving deer in a given node $i \in \mathcal{V}$ at a particular time period of the day and cannot be accurately predicted in advance. Fig. 4 depicts the drone routing paths (100 nodes) for both feeding and non-feeding times under different service variation levels. Results clearly indicate that the drone is not capable of serving all the points within the flying range when the dynamicity of deer are taken into consideration. In overall, out of 100 nodes, 85 and 81 nodes can be served by drones under low service variability levels and 65 each under high service variability levels for both feeding and non-feeding times. Most importantly, DTSP provides an infeasible path to serve 100 demand points under different service variability levels for both feeding and non-feeding times which is apparently not the case under STSP.

6. Conclusions

This paper proposes different ALNS-based immigrant schemes inside an ACO framework to solve a dynamic traveling salesman problem efficiently. The performance of the proposed ALNS-based immigrant schemes are compared with the existing ACO-based immigrant schemes available in the literature, namely R-ACO, E-ACO, and M-ACO. Results demonstrate the superiority of the proposed algorithms in handling dynamic environments over the peer algorithms. Further, a number of sensitivity analyses are carried out such as diversity of the solution space, ant replacement rate, and tuning various ALNS parameters to check the robustness of the proposed algorithms. It is observed that irrespective of the type of pattern change (i.e., random and cyclic), ALNS-based immigrant schemes have the capability to adapt themselves in evolving conditions. This is primarily due to the presence of multiple types of operators (insertion and deletion) available in the ALNS schemes. Finally, we show how the proposed algorithms can be employed to route drones in surveying deer which are highly mobile in nature. We use a wildlife refuge national park in Mississippi to visualize and validate the algorithmic results. Results demonstrate that the drone is not capable of serving all the points within it's flying range when the dynamicity of deer are taken into consideration. We further realize that the paths provided by solving a STSP may no longer considered feasible for routing drones that capture mobile objects. Our proposed algorithms can handle this dynamicism by providing reliable paths that can survey mobile population (e.g., wildlife) efficiently.

This research can be extended in several directions. First, proposing new ALNS-based immigrant schemes to solve a DTSP that contains *dynamic time-linkage* property (Nguyen, Yang, & Branke, 2012). Second, extending the immigrant schemes to solve a *Dynamic Vehicle Routing Problem* (DVRP). Third, improved immigrant schemes can be applied to other evolutionary algorithms to determine whether algorithmic performance can be improved. Finally, new operators inside the ALNS-based immigrant schemes need to be identified to solve DTSP and DVRP efficiently. These issues will be addressed in future studies.

Conflicts of interest

The authors declared that there is no conflict of interest.

Appendix A. Algorithms

Algorithm 1. Random immigrants ACO (R-ACO)

1: $t \leftarrow 0$
2: Initialize: $\mathbf{P}(0)$ and τ_0
3: $k_{short}(0) \leftarrow \phi$ and $\mathbf{S}_{bs} \leftarrow \phi$
4: **while** (termination criteria not satisfied) **do**
5: $\quad \mathbf{P}(t) \leftarrow$ ConstructAntSolutions
6: $\quad k_{short}(t) \leftarrow$ AddBestAnts($\mathbf{K}_s$)
7: $\quad$ **if** $(t = 0)$ **then**
8: $\quad\quad$ UpdatePheromone($k_{short}(t)$)
9: $\quad$ **else**
10: $\quad\quad \mathbb{R} \leftarrow$ GenerateRandomImmigrants(r)
11: $\quad\quad k_{short}(t) \leftarrow$ ReplaceAntsWithImmigrants($\mathbb{R}$)
12: $\quad\quad$ UpdatePheromone($k_{short}(t)$)
13: $\quad$ **end if**
14: $\quad \mathbf{S}_{bf} \leftarrow$ FindBest($\mathbf{P}(t)$)
15: $\quad$ **if** $(f(\mathbf{S}_{bf}) < f(\mathbf{S}_{bs}))$ **then**
16: $\quad\quad \mathbf{S}_{bs} \leftarrow \mathbf{S}_{bf}$
17: $\quad$ **end if**
18: $\quad t \leftarrow t + 1$
19: **end while**
20: **return** $\mathbf{S}_{bs}$

Algorithm 2. Elitism-based immigrants ACO (E-ACO)

1: $t \leftarrow 0$
2: Initialize: $\mathbf{P}(0)$ and τ_0
3: $k_{short}(0) \leftarrow \phi$ and $\mathbf{S}_{bs} \leftarrow \phi$
4: **while** (termination criteria not satisfied) **do**
5: $\quad \mathbf{P}(t) \leftarrow$ ConstructAntSolutions
6: $\quad k_{short}(t) \leftarrow$ AddBestAnts($\mathbf{K}_s$)
7: $\quad$ **if** $(t = 0)$ **then**
8: $\quad\quad$ UpdatePheromone($k_{short}(t)$)
9: $\quad$ **else**
10: $\quad\quad \mathbf{X}^e \leftarrow$ FindBest($\mathbf{P}(t-1)$)
11: $\quad\quad \mathbb{R}_e \leftarrow$ GenerateGuidedImmigrants($\mathbf{X}^e, r$) using Algorithm 3
12: $\quad\quad k_{short}(t) \leftarrow$ ReplaceAntsWithImmigrants($\mathbb{R}_e$)
13: $\quad\quad$ UpdatePheromone($k_{short}(t)$)
14: $\quad$ **end if**
15: $\quad \mathbf{S}_{bf} \leftarrow$ FindBest($\mathbf{P}(t)$)
16: $\quad$ **if** $(f(\mathbf{S}_{bf}) < f(\mathbf{S}_{bs}))$ **then**
17: $\quad\quad \mathbf{S}_{bs} \leftarrow \mathbf{S}_{bf}$
18: $\quad$ **end if**
19: $\quad t \leftarrow t + 1$
20: **end while**
21: **return** $\mathbf{S}_{bs}$

Algorithm 3. GenerateGuidedImmigrants($\mathbf{X}^{elite}, r$)

$\mathbf{X}_n^e \leftarrow \mathbf{X}^e$
$r_c \leftarrow$ SelectRandomCity$(\mathbf{X}_n^e)$
while (termination criteria not satisfied) **do**
  if $(rand[0.0, 1.0] \leqslant v)$ **then**
    $r_c' \leftarrow$ SelectNewRandomCity$(\mathbf{X}_n^e)$
  else
    $\mathbf{X}^r \leftarrow$ SelectRandomAnt$(P(t))$
    $r_c' \leftarrow$ NextCity$(r_c + 1, \mathbf{X}^r) \in \mathbf{X}^r$
  end if
  if $((r_c' = r_c + 1 \in \mathbf{X}_n^e)$ or $(r_c' = r_c - 1 \in \mathbf{X}_n^e))$ **then**
    break
  else
    Inversion$(r_c + 1, r_c') \in \mathbf{X}_n^e$
    $r_c \leftarrow r_c'$
  end if
end while
return $\mathbf{X}_n^e$

Algorithm 4. Memory-based immigrants ACO (M-ACO)

$t \leftarrow 0$
Initialize: $\mathbf{P}(0), \boldsymbol{\tau}_0$, and $k_{long}(0)$
$k_{short}(0) \leftarrow \phi$ and $\mathbf{S}_{bs} \leftarrow \phi$
$t_m \leftarrow rand[\underline{t}_m, \bar{t}_m]$
while (termination criteria not satisfied) **do**
  $\mathbf{P}(t) \leftarrow$ ConstructAntSolutions
  $k_{short}(t) \leftarrow$ AddBestAnts$(\mathbf{K}_s)$
  if $(t = t_m$ or dynamic change is detected) **then**
    UpdateMemory$(k_{long}(t))$ using Algorithm 5
    $t_m \leftarrow t + rand[\underline{t}_m, \bar{t}_m]$
  end if
  if $(t = 0)$ **then**
    UpdatePheromone$(k_{short}(t))$
  else
    $\mathbf{X}^e \leftarrow$ FindBest$(k_{long}(t))$
    $\mathbb{R}_m \leftarrow$ GenerateGuidedImmigrants$(\mathbf{X}^e, r)$ using Algorithm 2
    $k_{short}(t) \leftarrow$ ReplaceAntsWithImmigrants$(\mathbb{R}_m)$
    UpdatePheromone$(k_{short}(t))$
  end if
  $\mathbf{S}_{bf} \leftarrow$ FindBest$(\mathbf{P}(t))$
  if $(f(\mathbf{S}_{bf}) < f(\mathbf{S}_{bs}))$ **then**
    $\mathbf{S}_{bs} \leftarrow \mathbf{S}_{bf}$
  end if
  $t \leftarrow t + 1$
  $k_{long}(t) \leftarrow k_{long}(t-1)$
end while
return $\mathbf{S}_{bs}$

Algorithm 5. UpdateMemory$(k_{long}(t))$

if $(t = t_m)$ **then**
  $\mathbf{S}_{mb} \leftarrow$ FindBest$(P(t))$
end if
if (dynamic change is detected) **then**
  $\mathbf{S}_{mb} \leftarrow$ FindBest$(P(t-1))$
end if
if (still any random ant in $k_{long}(t)$) **then**
  ReplaceRandomWithBest$(\mathbf{S}_{mb}, k_{long}(t))$
else
  $\mathbf{S}_{cm} \leftarrow$ FindClosest$(\mathbf{S}_{mb}, k_{long}(t))$
  if $(f(S_{mb}) < f(S_{cm}))$ **then**
    $S_{cm} \leftarrow S_{mb}$
  end if
end if

Algorithm 6. ALNS

Let s be an initial solution
Initialize scores: σ_1, σ_2, and σ_3
Initialize best solution: $s^* \leftarrow s$
repeat
  $s' \leftarrow s$
  Select destroy $(\mathcal{O}^-)$ and insertion $(\mathcal{O}^+)$ operators according to roulette-wheel/stochastic universal sampling mechanism
  Generate a new solution s' from s by applying $\mathcal{O}^-$ and $\mathcal{O}^+$
  if $(f(s') < f(s))$ **then**
    $s \leftarrow s'$
    if $(f(s) < f(s^*))$ **then**
      $s^* \leftarrow s$
      increase the score of the operators by σ_1
    else
      increase the score of the operators by σ_2
    end if
  else
    if $(f(s') > f(s))$ **then**
      $s \leftarrow s'$
      increase the score of the operators by σ_3
    end if
  end if
  if the iteration count is a multiple of φ **then**
    update the weights of all operators and reset their scores.
  end if
until the stopping condition is met
return s^*

Algorithm 7. ALNS-1/ALNS-2

$t \leftarrow 0$
Initialize: $\mathbf{P}(0)$ and $\boldsymbol{\tau}_0$
$k_{short}(0) \leftarrow \phi$ and $\mathbf{S}_{bs} \leftarrow \phi$
while (termination criteria not satisfied) **do**
  $\mathbf{P}(t) \leftarrow$ ConstructAntSolutions
  $k_{short}(t) \leftarrow$ AddBestAnts$(\mathbf{K}_s)$
  if $(t = 0)$ **then**
    UpdatePheromone$(k_{short}(t))$
  else
    $\mathbf{X}^e \leftarrow$ FindBest$(\mathbf{P}(t-1))$
    $\mathbb{R}_e \leftarrow$ GenerateGuidedImmigrants$(\mathbf{X}^e, r)$ using Algorithm 6

(continued on next page)

```
    k_short(t) ← ReplaceAntsWithImmigrants(ℝ_e)
    UpdatePheromone(k_short(t))
  end if
  S_bf ← FindBest(P(t))
  if (f(S_bf) < f(S_bs)) then
    S_bs ← S_bf
  end if
  t ← t + 1
end while
return S_bs
```

Algorithm 8. ALNS-3/ALNS-4

```
t ← 0
Initialize: P(0), τ_0, and k_long(0)
k_short(0) ← φ and S_bs ← φ
t_m ← rand[t̲_m, t̄_m]
while (termination criteria not satisfied) do
  P(t) ← ConstructAntSolutions
  k_short(t) ← AddBestAnts(K_s)
  if (t = t_m or dynamic change is detected) then
    UpdateMemory(k_long(t)) using Algorithm 5
    t_m ← t + rand[t̲_m, t̄_m]
  end if
  if (t = 0) then
    UpdatePheromone(k_short(t))
  else
    X^e ← FindBest(k_long(t))
    ℝ_m ← GenerateGuidedImmigrants(X^e, r) using Algorithm 6
    k_short(t) ← ReplaceAntsWithImmigrants(ℝ_m)
    UpdatePheromone(k_short(t))
  end if
  S_bf ← FindBest(P(t))
  if (f(S_bf) < f(S_bs)) then
    S_bs ← S_bf
  end if
  t ← t + 1
  k_long(t) ← k_long(t − 1)
end while
return S_bs
```

Appendix B. Tables

See Tables 12 and 13.

Table 12
Comparison of different algorithms in CPU time for KroA100 dataset.

Pattern		KroA100										
	m	0.25						0.95				
	f	10	25	50	100	200		10	25	50	100	200
		CPU time (seconds)						CPU time (seconds)				
Random	R-ACO	877	894	860	835	842	R-ACO	942	909	855	823	813
	E-ACO	959	893	861	894	884	E-ACO	965	910	938	880	888
	M-ACO	894	931	913	868	827	M-ACO	960	952	928	830	870
	ALNS-1	**464**	475	474	459	440	ALNS-1	497	486	521	420	429
	ALNS-2	481	503	**358**	**322**	**307**	ALNS-2	495	519	**403**	**396**	**373**
	ALNS-3	501	**474**	511	428	466	ALNS-3	**486**	**475**	518	508	463
	ALNS-4	556	515	525	476	466	ALNS-4	550	551	516	472	468
Cyclic	R-ACO	846	876	781	823	732	R-ACO	941	923	882	899	841
	E-ACO	924	860	869	839	829	E-ACO	981	960	899	868	855
	M-ACO	843	854	906	789	800	M-ACO	882	916	865	827	798
	ALNS-1	477	452	445	356	405	ALNS-1	**484**	468	492	441	387
	ALNS-2	**448**	494	**373**	**337**	**254**	ALNS-2	564	**420**	**355**	**369**	**341**
	ALNS-3	499	**422**	414	387	432	ALNS-3	495	534	458	464	498
	ALNS-4	528	530	455	399	434	ALNS-4	575	505	497	534	413

Table 13
Comparison of different algorithms in CPU time for KroA200 dataset.

Pattern		KroA200										
	m	0.25						0.95				
	f	10	25	50	100	200		10	25	50	100	200
		CPU time (seconds)						CPU time (seconds)				
Random	R-ACO	1091	1080	1062	1037	1020	R-ACO	1109	1114	1085	1060	1043
	E-ACO	1111	1090	1080	1065	1050	E-ACO	1145	1115	1124	1093	1068
	M-ACO	1108	1089	1068	1049	1027	M-ACO	1152	1118	1110	1069	1047
	ALNS-1	**684**	669	644	620	599	ALNS-1	**713**	689	682	656	626
	ALNS-2	689	**661**	**545**	**527**	**506**	ALNS-2	721	**699**	**589**	**564**	**544**
	ALNS-3	707	687	671	647	629	ALNS-3	725	707	699	669	658
	ALNS-4	721	699	680	668	656	ALNS-4	756	730	700	700	699
Cyclic	R-ACO	1072	1085	1049	1015	1001	R-ACO	1115	1111	1094	1054	1030
	E-ACO	1109	1091	1083	1062	1055	E-ACO	1154	1123	1113	1076	1096
	M-ACO	1090	1088	1063	1047	1006	M-ACO	1122	1129	1087	1070	1022
	ALNS-1	**670**	668	633	621	598	ALNS-1	694	684	656	639	612
	ALNS-2	681	**646**	**536**	**536**	**480**	ALNS-2	**689**	**658**	**582**	**553**	**524**
	ALNS-3	715	682	645	616	623	ALNS-3	737	728	683	635	661
	ALNS-4	720	708	675	667	641	ALNS-4	746	732	685	710	657

References

Airborne drones (2016). Wildlife and game monitoring. Available from: <http://www.airbornedrones.co/pages/farming-drones>.

Angus, D. (2006). Niching for population-based ant colony optimization. In *Second IEEE international conference on e-science and grid computing*.

Angus, D. (2009). Niching for ant colony optimisation. *Biologically-inspired optimisation methods* (Vol. 210, pp. 165–188). Springer.

Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. J. (2011). *The traveling salesman problem: A computational study*. Princeton University Press.

Brinkman, T. J., Deperno, C. S., Jenks, J. A., Haroldson, B. S., & Osborn, R. G. (2005). Movement of female white-tailed deer: Effects of climate and intensive row-crop agriculture. *Journal of Wildlife Management, 69*(3), 1099–1111.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1990). A new rank based version of the ant system-a computational study. *Central European Journal for Operations Research in Economics, 7*(1), 25–38.

Castillo, O., Lizárraga, E., Soria, J., Melin, P., & Valdez, F. (2015a). New approach using ant colony optimization with ant set partition for fuzzy control design applied to the ball and beam system. *Information Sciences, 294*, 203–215.

Castillo, O., Neyoy, H., Soria, J., Melin, P., & Valdez, F. (2015b). A new approach for dynamic fuzzy logic parameter tuning in ant colony optimization and its application in fuzzy control of a mobile robot. *Applied Soft Computing, 28*, 150–159.

Cheng, H., & Yang, S. (2010). Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks. In *European conference on the applications of evolutionary computation*.

Cheng, H., & Yang, S. (2011). Genetic algorithms with elitism-based immigrants for dynamic load balanced clustering problem in mobile ad hoc networks. In *2011 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*.

Cheng, H., & Yang, S. (2010a). Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks. *Engineering Applications of Artificial Intelligence, 23*(5), 806–819.

Chowdhury, S., Emelogu, A., Marufuzzaman, M., Nurre, S. G., & Bian, L. (2017). Drones for disaster response and relief operations: A continuous approximation model. *International Journal of Production Economics, 188*, 167–184.

Christofides, N. (1970). The shortest hamiltonian chain of a graph. *SIAM Journal on Applied Mathematics, 19*(4), 689–696.

Coelho, L. C., Cordeau, J. F., & Laporte, G. (2012a). The inventory-routing problem with transshipment. *Computers & Operations Research, 29*, 2537–2548.

Coelho, L. C., Cordeau, J. F., & Laporte, G. (2012b). Consistency in multi-vehicle inventory-routing. *Transportation Research Part C, 24*, 270–287.

Colorni, A., Dorigo, M., & Maniezzo, V. (1992). *Toward a practice of autonomous systems: Proceedings of the first eEuropean conference on artificial Life*. Mit Press, p. 134.

Cordon, O., De Viana, I. F., Herrera, F., & Moreno, L. (2000). A new ACO model integrating evolutionary computation concepts: The best-worst Ant System.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America, 2*(4), 393–410.

Desrochers, M., & Laporte, G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters, 10*(1), 27–36.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26*(1), 29–41.

Du, W., & Li, B. (2008). Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences, 178*(15), 3096–3109.

Eddaly, M., Jarboui, B., & Siarry, P. (2016). Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem. *Journal of Computational Design and Engineering, 3*(4), 295–311.

Emelogu, A., Chowdhury, S., Marufuzzaman, M., Bian, L., & Eksioglu, B. (2016). An enhanced sample average approximation method for stochastic optimization. *International Journal of Production Economics, 182*, 230–252.

Eyckelhof, C. J., & Snoek, M. (2002). Ant systems for a dynamic TSP. In *International workshop on ant algorithms*.

Garcia-Parra, C., & Tapia, W. (2014). Marine wildlife health surveillance. In *The Galapagos islands: First year results of the rapid response network*. Available from: <http://www.galapagos.org/wp-content/uploads/2015/09/GalapagosReport2013-2014-13-Garcia-Parra-89-94.pdf>.

Glover, F. (1977). Heuristic for integer programming using surrogate constraints. *Decision Sciences, 8*, 156–166.

Goetzman, K. (2011). When tagging wild animals goes wrong. Available from: <http://www.utne.com/environment/when-tagging-wild-animals-goes-wrong.aspx>.

Golden, B. L., & Stewart, W. R. (1985). The traveling salesman problem. *Empirical Analysis of Heuristics, 4*, 207–249.

Groba, C., Sartal, A., & Vázquez, X. H. (2015). Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices. *Computers & Operations Research, 56*(6), 22–32.

Grogan, L. F., Berger, L., Rose, K., Grillo, V., Cashins, S. D., & Skerratt, L. F. (2014). Surveillance for emerging biodiversity diseases of wildlife. *PLoS Pathog, 10*(5).

Guntsch, M., & Middendorf, M. (2001). Pheromone modification strategies for ant algorithms applied to dynamic TSP. In *Workshops on applications of evolutionary computation*.

Guntsch, M., & Middendorf, M. (2002). A population based approach for ACO. In *Workshops on applications of evolutionary computation*.

Guntsch, M., Middendorf, M., & Schmeck, H. (2001). An ant colony optimization approach to dynamic TSP. In *Proceedings of the 3rd annual conference on genetic and evolutionary computation*.

Hart, E., & Ross, P. (1999). An immune system approach to scheduling in changing environments. In *Proceedings of the 1st annual conference on genetic and evolutionary computation*.

Hodgson, J. C., Baylis, S. M., Mott, R., Herrod, A., & Clarke, R. H. (2016). Precision wildlife monitoring using unmanned aerial vehicles. *Scientific Reports, 6*.

Jin, Y., & Branke, J. (2005). Evolutionary optimization in uncertain environments—A survey. *IEEE Transactions on Evolutionary Computation, 9*(3), 303–317.

Kilby, P., Prosser, P., & Shaw, P. (1998). *Dynamic VRPs: A study of scenarios* Technical Report. University of Strathclyde, pp. 1–11.

Kovacs, A., Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling, 15*(5), 579–600.

Kuiken, T., Ryser-Degiorgis, M. P., Gavier-Widen, D., & Gortázar, C. (2011). Establishing a european network for wildlife health surveillance. *Revue Scientifique et Technique-OIE, 30*(3), 755.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal, 44*(10), 2245–2269.

Liu, J. (2005). Rank-based ant colony optimization applied to dynamic traveling salesman problems. *Engineering Optimization, 37*(8), 831–847.

Mancini, S. (2016). A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. *Transportation Research Part C: Emerging Technologies, 70*, 100–112.

Massé, A., & Côté, S. D. (2009). Habitat selection of a large herbivore at high density and without predation: Trade-off between forage and cover? *Journal of Mammalogy, 90*(4), 961–970.

Mavrovouniotis, M., & Yang, S. (2010). Ant colony optimization with immigrants schemes in dynamic environments. In *International conference on parallel problem solving from nature*.

Mavrovouniotis, M., & Yang, S. (2011). Memory-based immigrants for ant colony optimization in changing environments. In *European conference on the applications of evolutionary computation*.

Mavrovouniotis, M., & Yang, S. (2012). Ant colony optimization with immigrants schemes for the dynamic vehicle routing problem. In *European conference on the applications of evolutionary computation*.

Mavrovouniotis, M., & Yang, S. (2012). Ant colony optimization with memory-based immigrants for the dynamic vehicle routing problem. In *IEEE congress on evolutionary computation (CEC)*.

Mavrovouniotis, M., Yang, S., & Yao, X. (2014). Multi-colony ant algorithms for the dynamic travelling salesman problem. In *IEEE symposium on computational intelligence in dynamic and uncertain environments*.

Mavrovouniotis, M., Muller, F. M., & Yang, S. (2015). An ant colony optimization based memetic algorithm for the dynamic travelling salesman problem. In *Genetic and evolutionary computation conference (GECCO15)* (pp. 49–56).

Mavrovouniotis, M., Müller, F. M., & Yang, S. (2016). Ant colony optimization with local search for dynamic traveling salesman problems. *IEEE Transactions on Cybernetics*.

Mavrovouniotis, M., & Yang, S. (2011b). A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Computing, 15*(7), 1405–1425.

Mavrovouniotis, M., & Yang, S. (2013). Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors. *Applied Soft Computing, 13*(10), 4023–4037.

Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization, 10*(4), 327–343.

Mostafa, M., Baykan, O. K., & Kodaz, H. (2015). A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. *Applied Soft Computing, 30*, 484–490.

Muller, L. F., Spoorendonk, S., & Pisinger, D. (2012). A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research, 218*(3), 614–623.

Nguyen, T. T., Yang, S., & Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation, 6*, 1–24.

Olivas, F., Valdez, F., & Castillo, O. (2015). Dynamic parameter adaptation in Ant Colony Optimization using a fuzzy system for TSP problems. In *16th World Congress of the International Fuzzy Systems Association (IFSA): 9th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT)*.

Olivas, F., Valdez, F., & Castillo, O. (2008). An ant colony optimization algorithm for image edge detection. *IEEE Congress on Evolutionary Computation*.

Olivas, F., Valdez, F., & Castillo, O. (2015a). Ant colony optimization with parameter adaptation using fuzzy logic for TSP problems. *Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization*, 593–603.

Olivas, F., Valdez, F., Castillo, O., Gonzalez, C. I., Martinez, G., & Melin, P. (2017). Ant colony optimization with dynamic parameter adaptation based on interval type-2 fuzzy logic systems. *Applied Soft Computing, 53*, 74–87.

Or, I. (1977). *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. Northwestern University Evanston IL.

Pencheva, T., Atanassov, K., & Shannon, A. (2009). Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets. In *Proceedings of the tenth international workshop on generalized nets, Sofia*.

Ropke, S., & Pisinger, D. (2006a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science, 40*(4), 455–472.

Ropke, S., & Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research, 171*(3), 750–775.

Ryser-Degiorgis, M. (2013). Wildlife health investigations: Needs, challenges and recommendations. *BMC Veterinary Research, 9*(1), 223.

Sama, M., Pellegrini, P., D'Ariano, A., Rodriguez, J., & Pacciarelli, D. (2016). Ant colony optimization for the real-time train routing selection problem. *Transportation Research Part B: Methodological, 85*, 89–108.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics, 159*(2), 139–171.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*.

SouthAfrica.info (2014). Hi-tech boost for fight against rhino poaching. Available from: <http://www.southafrica.info/business/trends/innovations/rhino-drones-shaya-security-171214.htm.V9MJNfkrKUk>.

Stutzle, T., & Hoos, H. (1997). MAX-MIN ant system and local search for the traveling salesman problem. In *IEEE international conference on evolutionary computation*.

Stützle, T., & Hoos, H. H. (2000). MAX–MIN ant system. *Future Generation Computer Systems, 16*(8), 889–914.

Tarjan, R. E. (1977). Finding optimum branchings. *Networks, 7*(1), 25–35.

The conversation (2016). Satellites, mathematics and drones take down poachers in Africa. Available from: <http://theconversation.com/satellites-mathematics-and-drones-take-down-poachers-in-africa-36638.

Toriello, A., Haskell, W. B., & Poremba, M. (2014). A dynamic traveling salesman problem with stochastic arc costs. *Operations Research, 62*(5), 1107–1125.

Treacy, M. (2016). Study says drones are better at wildlife monitoring than humans. Available from: <http://www.treehugger.com/gadgets/study-says-drones-are-better-wildlife-monitoring-humans.html.

United Nations Environment Programme (UNEP) (2013). UNEP Global Environmental Alert Service (GEAS). Available from: <http://www.unep.org/pdf/UNEP-GEAS-MAY-2013.pdf>.

Webb, S. L., Gee, K. L., Strickland, B. K., Demarais, S., & DeYoung, R. W. (2010). Measuring fine-scale white-tailed deer movements and environmental influences using GPS collars. *International Journal of Ecology*.

Webb, S. L., Riffell, S. K., Gee, K. L., & Demarais, S. (2009). Using fractal analyses to characterize movement paths of white-tailed deer and response to spatial scale. *Journal of Mammalogy, 90*(5), 1210–1217.

Yang, Q., Chen, W., Yu, Z., Gu, T., Li, Y., Zhang, H., & Zhang, J. (2017). Adaptive multimodal continuous ant colony optimization. *IEEE Transactions on Evolutionary Computation, 21*(2), 191–205.