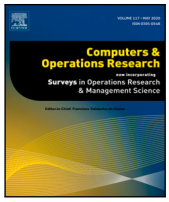
Computers & Operations Research 143 (2022) 105769

Contents lists available at ScienceDirect

Computers and Operations Research

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



Boosting ant colony optimization via solution prediction and machine learning

Yuan Sun [a,*], Sheng Wang [b], Yunzhuang Shen [c], Xiaodong Li [c], Andreas T. Ernst [a], Michael Kirley [d]

[a] School of Mathematics, Monash University, Clayton, VIC, 3800, Australia
[b] Center for Urban Science and Progress, New York University, New York, NY, 11201, USA
[c] School of Computing Technologies, RMIT University, Melbourne, VIC, 3000, Australia
[d] School of Computing and Information Systems, The University of Melbourne, Parkville, VIC, 3010, Australia

ARTICLE INFO

Keywords:
Meta-heuristic
Machine learning
Combinatorial optimization
Ant colony optimization
Optimal solution prediction

ABSTRACT

This paper introduces an enhanced meta-heuristic (ML-ACO) that combines machine learning (ML) and ant colony optimization (ACO) to solve combinatorial optimization problems. To illustrate the underlying mechanism of our ML-ACO algorithm, we start by describing a test problem, the orienteering problem. In this problem, the objective is to find a route that visits a subset of vertices in a graph within a time budget to maximize the collected score. In the first phase of our ML-ACO algorithm, an ML model is trained using a set of small problem instances where the optimal solution is known. Specifically, classification models are used to classify an edge as being part of the optimal route, or not, using problem-specific features and statistical measures. The trained model is then used to predict the 'probability' that an edge in the graph of a test problem instance belongs to the corresponding optimal route. In the second phase, we incorporate the predicted probabilities into the ACO component of our algorithm, i.e., using the probability values as heuristic weights or to warm start the pheromone matrix. Here, the probability values bias sampling towards favoring those predicted 'high-quality' edges when constructing feasible routes. We have tested multiple classification models including graph neural networks, logistic regression and support vector machines, and the experimental results show that our solution prediction approach consistently boosts the performance of ACO. Further, we empirically show that our ML model trained on small synthetic instances generalizes well to large synthetic and real-world instances. Our approach integrating ML with a meta-heuristic is generic and can be applied to a wide range of optimization problems.

1. Introduction

Ant colony optimization (ACO) is a class of widely-used meta-heuristics, inspired by the foraging behavior of biological ants, for solving combinatorial optimization problems (Dorigo et al., 1996; Dorigo and Gambardella, 1997). Since its introduction in early 1990s, ACO has been extensively investigated to understand both its theoretical foundations and practical performance (Dorigo and Blum, 2005; Blum, 2005). A lot of effort has been made to improve the performance of ACO, making it one of the most competitive algorithms for solving a wide range of optimization problems. Whilst ACO cannot provide any optimality guarantee due to its heuristic nature, it is usually able to find a high-quality solution for a given problem within a limited computational budget.

The ACO algorithm builds a probabilistic model to sample solutions for an optimization problem. In this sense, ACO is closely related to Estimation of Distribution Algorithms and Cross Entropy methods (Zlochin et al., 2004). The probabilistic model of ACO is parametrized by a so-called *pheromone matrix* and a *heuristic weight matrix*, which basically measure the 'payoff' of setting a decision variable to a particular value. The aim of ACO is to evolve the pheromone matrix so that an optimal (or a near-optimal) solution can be generated via the probabilistic model in sampling. Previously, the pheromone matrix is usually initialized uniformly and the heuristic weights are set based on prior domain knowledge. In this paper, we develop machine learning (ML) techniques to warm start the pheromone matrix or identify good heuristic weights for ACO to use.

Leveraging ML to help combinatorial optimization has attracted much attention recently (Bengio et al., 2021; Karimi-Mamaghan et al., 2022). For instance, novel ML techniques have been developed to prune

* Corresponding author.
E-mail addresses: yuan.sun@monash.edu (Y. Sun), swang@nyu.edu (S. Wang), s3640365@student.rmit.edu.au (Y. Shen), xiaodong.li@rmit.edu.au (X. Li), andreas.ernst@monash.edu (A.T. Ernst), mkirley@unimelb.edu.au (M. Kirley).

https://doi.org/10.1016/j.cor.2022.105769
Received 28 April 2021; Received in revised form 24 February 2022; Accepted 25 February 2022
Available online 7 March 2022
0305-0548/© 2022 Elsevier Ltd. All rights reserved.

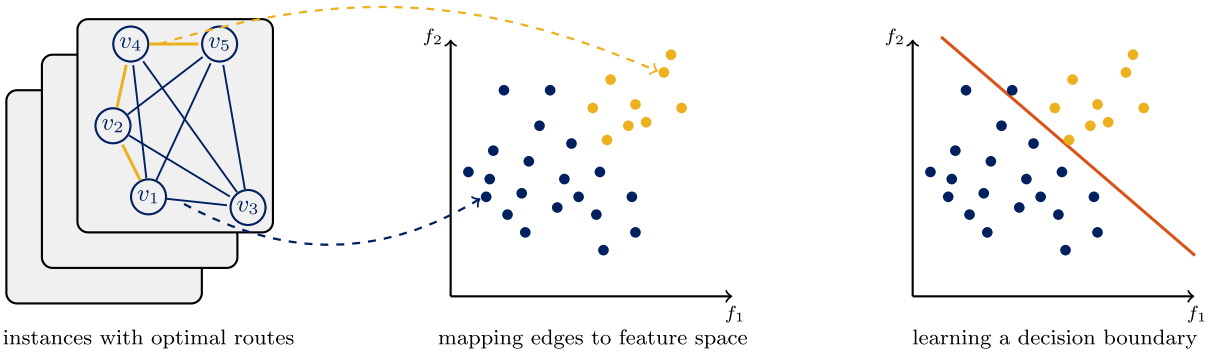


Fig. 1. An illustration of the training procedure of our ML model. First, a set of orienteering problem instances are solved, with the optimal route highlighted in yellow in the corresponding graph of orienteering problem instance (left figure). We then extract features (e.g., edge weight) to describe each edge of the graphs, and map each edge to the feature space as a training point (middle figure). Finally, we apply a classification algorithm to learn a decision boundary in the feature space to well separate edges (training points) that are part of the optimal routes from those which are not (right figure).

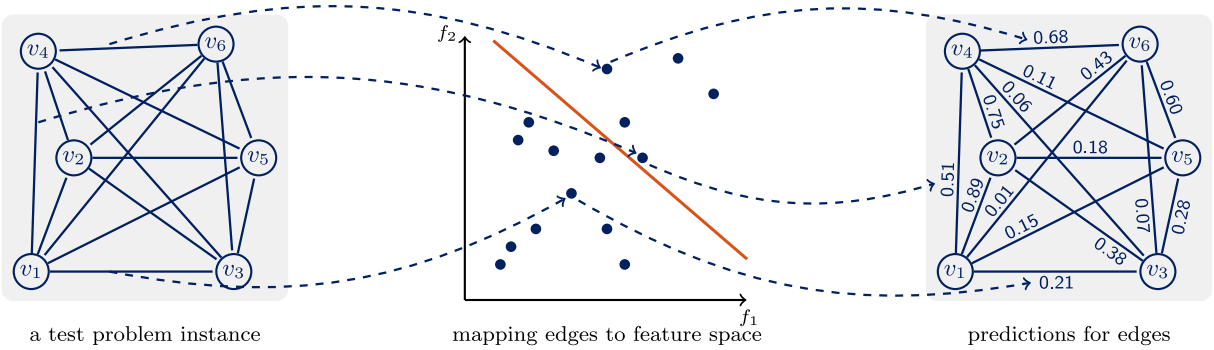


Fig. 2. The testing procedure of our ML model. Given an unsolved test orienteering problem instance (left figure), we first map each edge of the corresponding graph to a point in the feature space (middle figure). Based on the location of the points with respect to the decision boundary learned in training, we can compute for each edge a probability that it belongs to an optimal route (right figure). The predictions are then used to bias the sampling of ACO towards using the edges with a larger predicted probability value more often when constructing feasible routes.

the search space of large-scale optimization problems to a smaller size that is manageable by existing solution algorithms (Sun et al., 2021b; Lauri and Dutta, 2019; Sun et al., 2021a), to order decision variables for branch and bound or tree search algorithm (Li et al., 2018b; Shen et al., 2021), and to approximate the objective value of solutions (Fischetti and Fraccaro, 2019; Santini et al., 2021). There also exist ML-based methods that try to directly predict a high-quality solution for an optimization problem (Abbasi et al., 2020; Ding et al., 2020). The key idea of these methods is typically *solution prediction* via ML; that is aiming to predict the optimal solution for a given problem as closely as possible.

Building upon these previous studies, we propose an enhanced meta-heuristic named ML-ACO, that combines ML (more specifically solution prediction) and ACO to solve combinatorial optimization problems. To illustrate the underlying mechanism of our proposed algorithm, we first describe the orienteering problem, which is also used to demonstrate the efficacy of ML-ACO. The aim of orienteering problem is to search for a route in a graph that visits a subset of vertices within a given time budget to maximize the total score collected from the visited vertices (see Section 2.1 for a formal definition). The orienteering problem has many real-world applications (Vansteenwegen et al., 2011; Gunawan et al., 2016).

In the first phase of our ML-ACO algorithm, an ML model is trained on a set of optimally-solved small orienteering problem instances with known optimal route, as shown in Fig. 1. We extract problem-specific features as well as statistical measures (see Section 3.1) to describe each edge in the graphs of solved orienteering problem instances, and map each edge to a training point in the feature space. Classification algorithms can then be used to learn a decision boundary in the feature space to differentiate the edges that are in the optimal routes from those which are not. We have tested multiple existing classification algorithms for this task including graph neural networks (Kipf and Welling, 2017; Wu et al., 2021), logistic regression (Bishop, 2006) and support vector machines (Boser et al., 1992; Cortes and Vapnik, 1995). For an unsolved test orienteering problem instance, the trained ML model can then be used to predict the 'probability' that an edge in the corresponding graph belongs to the optimal route, as shown in Fig. 2.

In the second phase of our ML-ACO algorithm, we incorporate the probability values predicted by our ML model into the ACO algorithm, i.e., using the probability values to set the *heuristic weight matrix* or to initialize the *pheromone matrix* of ACO. The aim is to bias the sampling of ACO towards favoring the edges that are predicted more likely to be part of an optimal route, and hopefully to improve the efficiency of ACO in finding high-quality routes. In this sense, the idea of our ML-ACO algorithm is also related to the *seeding* strategies that are used to improve evolutionary algorithms (Liaw, 2000; Hopper and Turton, 2001; Friedrich and Wagner, 2015; Chen et al., 2018).

We use simulation experiments to show the efficacy of our ML-ACO algorithm on the orienteering problem. The results show that our ML-ACO algorithm significantly improves over the classic ACO in finding high-quality solutions. We also test the use of different classification algorithms, and observe that our solution prediction approach consistently boosts the performance of ACO. Finally, we show that our ML model trained on small synthetic problem instances generalizes well to large synthetic instances as well as real-world instances.

In summary, we have made the following contributions:

- This paper is the first attempt, to our knowledge, at boosting the performance of the ACO algorithm via solution prediction and ML.

- We empirically show that our proposed ML-ACO algorithm significantly improves over the classic ACO, no matter which classification algorithm is used in training.
- We also demonstrate the generalization capability of ML-ACO on large synthetic and real-world problem instances.

The remainder of this paper is organized as follows. In Section 2, we introduce the orienteering problem and the ACO algorithm. In Section 3, we describe the proposed ML-ACO algorithm. Section 4 presents our experimental results, and the last section concludes the paper and shows potential avenues for future research.

2. Background and related work

We first describe the orienteering problem and then introduce the ACO algorithm in the context of orienteering problem.

2.1. Orienteering problem

The orienteering problem finds its application in many real-world problems, such as tourist trip planning, home fuel delivery and building telecommunication networks (Vansteenwegen et al., 2011; Gunawan et al., 2016). Consider a complete directed graph $G(V, E, S, C)$, where $V = \{v_1, \dots, v_n\}$ denotes the vertex set, $E = \{e_{i,j} \mid 1 \le i \ne j \le n\}$ denotes the edge set, $S = \{s_1, \dots, s_n\}$ denotes the score of each vertex, and $C = \{c_{i,j} \mid 1 \le i \ne j \le n\}$ denotes the time cost of traveling through each edge. In this paper, we will only consider directed graphs with symmetry in time costs, i.e., $c_{i,j} = c_{j,i}$, however, the same idea can be applied to general directed graphs. Assume v_1 is the starting vertex and v_n is the ending vertex. The objective of the orienteering problem is to search for a path from v_1 to v_n that visits a subset of vertices within a given time budget $T_{\max}$, such that the total score collected is maximized. Thus, the orienteering problem can be viewed as a combination of traveling salesman problem and knapsack problem. We use u_i to denote the visiting order of vertex v_i, and use a binary variable $x_{i,j}$ to denote whether vertex v_j is visited directly after vertex v_i. The integer program of the orienteering problem can be written as:

$$\max_{\boldsymbol{x},\boldsymbol{u}} \sum_{i=1}^{n-1}\sum_{j=2}^{n} s_j x_{i,j}, \tag{1}$$

$$s.t. \sum_{j=1}^{n} x_{1,j} = \sum_{i=1}^{n} x_{i,n} = 1, \tag{2}$$

$$\sum_{i=1}^{n-1} x_{i,k} = \sum_{j=2}^{n} x_{k,j} \le 1, \qquad 2 \le k \le n-1; \tag{3}$$

$$u_i - u_j + 1 \le (n-1)(1 - x_{i,j}), \qquad 2 \le i, j \le n; \tag{4}$$

$$\sum_{i=1}^{n-1}\sum_{j=2}^{n} c_{i,j} x_{i,j} \le T_{\max}, \tag{5}$$

$$u_i \ge 0, \qquad 2 \le i \le n; \tag{6}$$

$$x_{i,j} \in \{0, 1\}, \qquad 1 \le i, j \le n. \tag{7}$$

The constraints (2) ensure that the path starts from vertex v_1 and ends in vertex v_n. The constraints (3) guarantee that each vertex in between can only be visited at most once. The constraints (4) are the Miller–Tucker–Zemlin subtour elimination constraints, and the constraint (5) satisfies the given time budget. Note that this formulation is not computationally efficient, and there are some relatively trivial ways to make it slightly stronger (Fischetti et al., 1998). However, this formulation is sufficient for logical correctness.

The orienteering problem is NP-hard (Golden et al., 1987). Many solution methods have been proposed to solve the orienteering problem and its variants, including exact solvers (Fischetti et al., 1998; El-Hajj et al., 2016; Archetti et al., 2016; Angelelli et al., 2017) and heuristics or meta-heuristics (Kobeaga et al., 2018; Santini, 2019; Hammami et al., 2020; Assunção and Mateus, 2021). Solving the orienteering problem to optimality using exact solvers may take a long time, especially for large instances. However, in some real-world applications such as tourist trip planning, we need to provide a high-quality solution to users within a short time. In this case, meta-heuristics are useful to search for a high-quality solution when the computational budget is very limited. In the next subsection, we describe the meta-heuristic, ACO, to solve the orienteering problem.

2.2. Ant colony optimization

The ACO algorithm is inspired by the behavior of biological ants seeking the shortest path between food and their colony (Dorigo et al., 1996; Dorigo and Gambardella, 1997). Unlike many other nature inspired algorithms, ACO has a solid mathematical foundation, based on probability theory. The underlying mechanism of ACO is to build a parametrized probabilistic model to incrementally construct feasible solutions. The parameters of this probabilistic model are evolved over time, based on the sample solutions generated in each iteration of the algorithm. By doing this, better solution components are reinforced, leading to an optimal (or near-optimal) solution in the end. The ACO algorithm has been demonstrated to be effective in solving various combinatorial optimization problems (Dorigo and Blum, 2005; Blum, 2005; Mavrovouniotis et al., 2016; Xiang et al., 2021; Jia et al., 2021; Palma-Heredia et al., 2021).

As our main focus is to investigate whether ML can be used to improve the performance of ACO, we simply test on two representative ACO models — Ant System (AS) (Dorigo et al., 1996) and Max–Min Ant System (MMAS) (Stützle and Hoos, 2000). The AS is one of the original ACO algorithms, and MMAS is a well-performing variant (Blum, 2005). Note that ACO has been applied to solve the orienteering problem variants, e.g., team orienteering problem (Ke et al., 2008), team orienteering problem with time windows (Montemanni et al., 2011; Gambardella et al., 2012) and time-dependent orienteering problem with time windows (Verbeeck et al., 2014, 2017). These works are typically based on one of the early ACO models, possibly integrated with local search methods. To avoid complication, we simply select the AS and MMAS models, which are sufficient for our study.

The **AS** algorithm uses a population of m ants to incrementally construct feasible solutions based on a parametrized probabilistic model. For the orienteering problem, a feasible solution is a path, consisting of a set of connected edges. In one iteration of the algorithm, each of the m ants constructs its own path from scratch. Starting from v_1, an ant incrementally selects the next vertex to visit until all the time budget is used up. Note that as v_n is the ending vertex, each ant should reserve enough time to visit v_n.

Suppose an ant is at vertex v_i, and V_i denotes the set of vertices that this ant can visit in the next step without violating the time budget constraint. The probability of this ant visiting vertex $v_j \in V_i$ in the next step is defined by

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}}{\sum_{k \in V_i} \tau_{i,k}^{\alpha} \eta_{i,k}^{\beta}}, \tag{8}$$

where $\tau_{i,j}$ is the amount of *pheromone* deposited by the ants for transition from vertex v_i to v_j; $\eta_{i,j}$ is the *desirability of transition* from vertex v_i to v_j; $\alpha \ge 0$ and $\beta \ge 0$ are control parameters. This means the probability of visiting vertex $v_j \in V_i$ from v_i is proportional to the product of $\tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}$.

The desirability of transition from vertex v_i to v_j, i.e., $\eta_{i,j}$, is usually set based on prior knowledge. In the orienteering problem, we can set $\eta_{i,j}$ to the ratio of the score collected at vertex v_j to the time required for traveling from vertex v_i to v_j: $\eta_{i,j} = s_j / c_{i,j}$. This computes the score that can be collected per unit time if traveling through edge $e_{i,j}$, and measures the 'payoff' of including edge $e_{i,j}$ in the path in terms of the objective value. By using these $\boldsymbol{\eta}$ values, high-quality edges (i.e., those

allowing for a large collected score per unit time) are more likely to be sampled.

The pheromone values $\boldsymbol{\tau}$ are typically initialized uniformly, and are gradually evolved in each iteration of the algorithm, such that the components (edges) of high-quality sample solutions gradually acquire a large pheromone value. This also biases the sampling to using high-quality edges more often. In each iteration, after all the m ants have completed their solution construction process, the pheromone values $\tau_{i,j}$, where $i, j = 1, \ldots, n$ and $i \neq j$, are updated based on the sample solutions generated:

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}, \tag{9}$$

where $\rho > 0$ is the *pheromone evaporation coefficient*, and $\Delta\tau_{i,j}^{k}$ is the amount of pheromone deposited by the kth ant on edge $e_{i,j}$. Let y_k denote the objective value collected by the kth ant, y^{best} be the best objective value found so far, and $Q > 0$ be a constant. We can define $\Delta\tau_{i,j}^{k} = y_k/(y^{\text{best}}Q)$, if edge $e_{i,j}$ is used by the kth ant; otherwise $\Delta\tau_{i,j}^{k} = 0$. The amount of pheromone deposited by an ant when it travels along a path is proportional to the objective value of the path. As we are solving a maximization problem, edges that appear in high-quality paths are reinforced (i.e., acquiring a large pheromone value), so that these edges are more likely to be used when constructing paths in the later iterations. This sampling and evolving process is repeated for a predetermined number of iterations, and the best solution generated is returned in the end.

The **MMAS** algorithm is a variant of AS, which uses the same probabilistic model (Eq. (8)) to construct feasible solutions. The key difference between MMAS and AS is how the pheromone matrix ($\boldsymbol{\tau}$) is updated. The MMAS algorithm only uses a single solution to update the pheromone matrix in each iteration, in contrast to AS which uses a population of m solutions. This single solution can either be the best solution generated in the current iteration (*iteration-best*) or the best one found so far (*global-best*). The use of a single best solution makes the search more greedy towards high-quality solutions, in the sense that only the edges in the best solution get reinforced. Let $\boldsymbol{x}^{\text{best}}$ denote the best solution and y^{best} be the objective value of $\boldsymbol{x}^{\text{best}}$. The pheromone values $\tau_{i,j}$ for each pair of $i, j = 1, \ldots, n$ and $i \neq j$ are updated as

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \Delta\tau_{i,j}^{\text{best}}, \tag{10}$$

where $\Delta\tau_{i,j}^{\text{best}} = 1/y^{\text{best}}$ if edge $e_{i,j}$ is in the best solution $\boldsymbol{x}^{\text{best}}$; otherwise $\Delta\tau_{i,j}^{\text{best}} = 0$.

The second key difference between MMAS and AS is that the pheromone values in MMAS are restricted to a range of $[\tau_{\min}, \tau_{\max}]$. After the pheromone values have been updated in each iteration using Eq. (10), if a pheromone value $\tau_{i,j} > \tau_{\max}$, we reset it to the upper bound $\tau_{\max}$. This avoids the situation where an edge accumulates a very large pheromone value, such that it is (almost) always selected in sampling based on the probabilistic model. The upper bound on pheromone values is derived as

$$\tau_{\max} = \frac{1}{\rho \cdot y^{\text{opt}}}, \tag{11}$$

where y^{opt} is the optimal solution of the problem instance. In practice, we often substitute y^{opt} with the best solution found so far to compute the upper bound, since we do not have y^{opt} before solving the problem. Similarly if a pheromone value $\tau_{i,j} < \tau_{\min}$, we reset it to the lower bound $\tau_{\min}$. This ensures the probability of selecting any edge in sampling does not reduce to zero. In this sense, the probability of generating the optimal solution via sampling approaches one if given an infinite amount of time. We simply set the lower bound to

$$\tau_{\min} = \frac{\tau_{\max}}{2n}, \tag{12}$$

where n is the problem dimensionality. This setting is consistent with the original paper (Stützle and Hoos, 2000) for solving the traveling salesman problem.

The MMAS algorithm also uses an additional mechanism called *pheromone trail smoothing* to deal with premature convergence. When the algorithm has converged, the pheromone values are increased proportionally to their difference to $\tau_{\max}$:

$$\tau_{i,j} = \tau_{i,j} + \delta \cdot (\tau_{\max} - \tau_{i,j}), \tag{13}$$

where $0 \leq \delta \leq 1$ is a control parameter. In the extreme case $\delta = 1$, it is equivalent to restarting the algorithm, in the sense that the pheromone values are reinitialized uniformly to $\tau_{\max}$. We activate the *pheromone trail smoothing* mechanism if there is no improvement in the objective value for a predetermined number of consecutive iterations T_{pts}.

3. Boosting ant colony optimization via solution prediction

This section presents the proposed ML-ACO algorithm, that integrates ML with ACO to solve the orienteering problem. The main idea of our ML-ACO algorithm is first to develop an ML model, aiming to predict the probability that an edge in the graph of an orienteering problem instance belongs to the optimal route. The training and testing procedures of our ML model are illustrated in Figs. 1 and 2 respectively. The predicted probability values are then leveraged to improve the performance of ACO in finding high-quality solutions.

In the first phase of our ML-ACO algorithm, we construct a training set from small orienteering problem instances (graphs), that are solved to optimality by a generic exact solver — CPLEX V12.8.0. We treat each edge in a solved graph as a training point, and extract several graph features as well as statistical measures to characterize each edge (Section 3.1). The edges that are part of the optimal route obtained by CPLEX are called *positive* training points and labeled as 1; otherwise they are *negative* training points labeled as −1. Note that the decision variables of the edges that do not belong to the optimal route have a value of zero in the optimal solution produced by CPLEX. We call these edges *negative* training points to be consistent with the ML literature. This then becomes a binary classification problem, where the goal is to learn a decision rule based on the extracted features to well separate the positive and negative training points. We will test multiple classification algorithms for this task. Given an *unsolved* orienteering problem instance, the trained ML model can then be applied to predict for each edge a probability that it belongs to the optimal route (Section 3.2).

In the second phase of ML-ACO, the probability values predicted by our ML model are then incorporated into the probabilistic model of ACO to improve its performance. The idea is to use the edges that are predicted more likely to be in an optimal route more often in the sampling process of ACO. By doing this, high-quality routes can hopefully be generated more quickly. We will use the predicted probability values to either seed the *pheromone matrix* or set the *heuristic weight matrix* of ACO (Section 3.3).

The general procedure of our ML-ACO algorithm can be summarized as follows:

1. Solve small orienteering problem instances to optimality using CPLEX;
2. Construct a training set from the optimally-solved problem instances;
3. Train an ML model offline to separate positive and negative training points (edges) in our training set;
4. Predict which edges are more likely to be in an optimal route for a test (unsolved) problem instance;
5. Incorporate solution prediction into ACO to boost its performance.

Note that our ML model is based on the edge representation of solutions for the orienteering problem, i.e., each edge in the graph belongs to a route (solution) or not. A more efficient way typically used by ACO to represent a route is using a sequence of vertices. A potential avenue for future research would be to develop an ML model based on the vertex representation to predict the order in which vertices are

visited in the optimal route. This will become a *multiclass* classification problem in contrast to the *binary* classification problem developed in this paper.

3.1. Constructing training set

We construct a training set from optimally-solved orienteering problem instances on complete graphs, where each edge is a training point. We assign a class label 1 to edges that belong to the optimal route and −1 to those who do not. Three graph features and two statistical measures are designed to characterize each edge, which are detailed below.

Recall that the objective of orienteering problem $G(V, E, S, C)$ is to search for a path that visits a subset of vertices within a given time budget $T_{\max}$, to maximize the total score collected. Three factors are relevant to the objective, i.e., vertex scores S, edge costs C and time budget $T_{\max}$. The first graph feature we design to describe edge $e_{i,j}$, is the ratio between edge cost $c_{i,j}$ and the time budget $T_{\max}$

$$f_1(e_{i,j}) = \frac{c_{i,j}}{T_{\max}}, \tag{14}$$

where $i, j = 1, \dots, n$ and $i \neq j$. Intuitively, if $c_{i,j} > T_{\max}$, the edge $e_{i,j}$ certainly cannot appear in any of the feasible solutions. A stronger preprocessing criterion would be to eliminate edge $e_{i,j}$ if $c_{1,i} + c_{i,j} + c_{j,n} > T_{\max}$, where v_1 is the starting vertex and v_n is the ending vertex. However, this type of exact pruning mechanism is not expected to eliminate many edges from a problem instance.

Another informative feature for describing edge $e_{i,j}$ is the ratio between vertex score s_j and edge cost $c_{i,j}$, which computes the score we can collect immediately from vertex v_j per unit time if taking the edge $e_{i,j}$. We normalize this ratio of edge $e_{i,j}$ by the maximum ratio of the edges that originates from vertex v_i,

$$f_2(e_{i,j}) = \frac{s_j / c_{i,j}}{\max_{k=1,\dots,n} s_k / c_{i,k}}. \tag{15}$$

This normalization is useful because it computes the *relative* payoff of selecting edge $e_{i,j}$, comparing to the alternative ways of leaving vertex v_i. Similarly, we also normalize the ratio of edge $e_{i,j}$ by the maximum ratio of the edges that ends in vertex v_j,

$$f_3(e_{i,j}) = \frac{s_j / c_{i,j}}{\max_{k=1\dots,n} s_j / c_{k,j}}. \tag{16}$$

This computes the *relative* payoff of visiting vertex v_j via edge $e_{i,j}$, comparing against other ways of visiting vertex v_j. These graph features are computationally very cheap, but they only capture local characteristics of an edge. Hence, we also adopt two statistical measures, originally proposed in Sun et al. (2021b), to capture global features of an edge.

The two statistical measures rely on random samples of feasible solutions (routes). We use the method presented in Appendix A.1 to generate m random feasible solutions, denoted as $\{\boldsymbol{x}^1, \boldsymbol{x}^2, \dots, \boldsymbol{x}^m\}$, and their objective values denoted as $\{y^1, y^2, \dots, y^m\}$. Each solution $\boldsymbol{x}$ is a binary string, where $x_{i,j} = 1$ if the edge $e_{i,j}$ is in the route; otherwise $x_{i,j} = 0$. The time complexity of sampling m feasible solutions for an n-dimensional problem instance is $\mathcal{O}(mn)$, which is proved in Appendix A.1. The sample size m should be larger than n, otherwise there will be some edges that are never sampled. We will set $m = 100n$ in our experiments, unless explicitly indicated otherwise.

The first statistical measure for characterizing edge $e_{i,j}$ is computed based on the ranking of sample solutions

$$f_r(e_{i,j}) = \sum_{k=1}^{m} \frac{x_{i,j}^k}{r^k}, \tag{17}$$

where r^k denotes the ranking of the kth sample in terms of its objective value in descending order. This ranking-based measure assigns a large score to edges that *frequently* appear in *high-quality* sample solutions, in the hope that these edges may also appear in an optimal solution. We normalize the ranking-based score of each edge by the maximum score in a problem instance to alleviate the effects of different sample size m

$$f_4(e_{i,j}) = \frac{f_r(e_{i,j})}{\max_{p,q=1,\dots,n} f_r(e_{p,q})}. \tag{18}$$

The other statistical measure employed is a correlation-based measure, that computes the Pearson correlation coefficient between each variable $x_{i,j}$ and objective values y across the sample solutions:

$$f_c(e_{i,j}) = \frac{\sum_{k=1}^{m}(x_{i,j}^k - \bar{x}_{i,j})(y^k - \bar{y})}{\sqrt{\sum_{k=1}^{m}(x_{i,j}^k - \bar{x}_{i,j})^2}\sqrt{\sum_{k=1}^{m}(y^k - \bar{y})^2}}, \tag{19}$$

where $\bar{x}_{i,j} = \sum_{k=1}^{m} x_{i,j}^k / m$, and $\bar{y} = \sum_{k=1}^{m} y^k / m$. As the orienteering problem is a maximization problem, edges that are highly positively correlated with the objective values are likely to be in an optimal route. Similarly, we normalize the correlation-based score of each edge by the maximum correlation value in a problem instance:

$$f_5(e_{i,j}) = \frac{f_c(e_{i,j})}{\max_{p,q=1,\dots,n} f_c(e_{p,q})}. \tag{20}$$

The time complexity of directly computing these two statistical measures based on the binary string representation $\mathbf{x}$ is $\mathcal{O}(mn^2)$, as we need to visit every bit in each of the m binary strings. To improve the time efficiency, we adopt the method proposed in Sun et al. (2021b), which represents the sample solutions using sets instead of strings. We then are able to compute the statistical measures in $\mathcal{O}(mn + n^2)$ time. The details of how to efficiently compute these measures are presented in Appendix A.2.

In summary, we have extracted five features (f_1, f_2, f_3, f_4, f_5) to characterize each edge (training point). For a problem instance with n vertices, we can extract $n(n-1)$ training points, as there are $n(n-1)$ directed edges in the corresponding complete graph. We use multiple solved problem instances to construct our training set $\mathbb{S} = \{(\boldsymbol{f}^i, l^i) \mid i = 1, \dots, n_t\}$, where $\boldsymbol{f}^i$ is the 5-dimensional feature vector; $l^i \in \{-1, 1\}$ is the class label of the ith training point; and n_t is the number of training points.

3.2. Training and solution prediction

After we have obtained a training set $\mathbb{S}$, our goal is then to learn a decision boundary to separate positive (label 1) and negative (label −1) training points in $\mathbb{S}$ as well as possible. This is a typical binary classification problem, that can be solved by any off-the-shelf classification algorithm. To see the effects of using different classification algorithms, we compare three alternatives for this task, namely, support vector machine (SVM) (Boser et al., 1992; Cortes and Vapnik, 1995), logistic regression (LR) (Bishop, 2006), and graph convolutional network (GCN) (Kipf and Welling, 2017; Wu et al., 2021). SVM and LR are well-known traditional algorithms with a solid mathematical foundation, and GCN is a popular deep neural network based on graph structure of a problem. This comparison is interesting, because it sheds light on whether a 'deep' model outperforms a 'shallow' model in the context of solution prediction for combinatorial optimization. A brief description of the three learning algorithms can be found in Appendix A.3.

In our training set, the number of positive training points is much smaller than that of negative training points. Considering an orienteering problem instance with n vertices, the number of edges appearing in an optimal route is less than n, and the total number of edges in the directed complete graph is $n(n-1)$. Hence, the ratio between positive and negative edges is less than $1/(n-2)$. In this sense, our training set is highly imbalanced, and classification algorithms tend to classify negative training points better than the positive points. To address this issue, we penalize misclassifying positive training points more by using a larger regularization parameter r^+, in contrast to that of negative training points r^- (see the loss functions (25), (28) and (30) of the

classification algorithms in Appendix A.3). In our experiments, we set $r^- = 1$ and $r^+ = n_{-1}/n_1$, where n_{-1} and n_1 are the number of negative and positive points in our training set.

In the testing phase, we can apply the trained model to predict a scalar $z_{i,j}$ for each edge $e_{i,j}$ in an unseen orienteering problem instance, where $i, j = 1, \ldots, n$ and $i \neq j$. For GCN, $z_{i,j}$ is the output of the last layer. For SVM and LR, $z_{i,j}$ is computed as $z_{i,j} = \boldsymbol{w}_*^T \boldsymbol{f}^{i,j} + b_*$, where $(\boldsymbol{w}_*, b_*)$ are the optimized parameters, and $\boldsymbol{f}^{i,j}$ is the feature vector of edge $e_{i,j}$. We then feed the predicted value $z_{i,j}$ into the logistic function to normalize it to a range of $[0, 1]$:

$$p_{i,j} = \frac{1}{1 + e^{-z_{i,j}}}. \tag{21}$$

The value of $p_{i,j}$ approaches 1 if $z_{i,j}$ approaches infinity; and $p_{i,j}$ approaches 0 when $z_{i,j}$ approaches negative infinity. In this sense, $p_{i,j}$ can be interpreted as the probability of edge $e_{i,j}$ belonging to an optimal solution. In the next subsection, we will explore multiple ways of incorporating the predicted probability values $p_{i,j}$ into ACO to guide its sampling process.

3.3. Incorporating solution prediction into ACO

Recall that the probabilistic model of ACO heavily depends on the *heuristic weight matrix* $\boldsymbol{\eta}$, as shown in Eq. (8). The $\eta_{i,j}$ value is a 'quality' measure of edge $e_{i,j}$, indicating if it is beneficial to include edge $e_{i,j}$ in a solution in order to obtain a large objective value. The $\boldsymbol{\eta}$ values are usually set based on a heuristic rule, for instance in the orienteering problem we can set $\eta_{i,j} = s_j/c_{i,j}$, where s_j is the score of vertex v_j and $c_{i,j}$ is the cost of edge $e_{i,j}$. Here, we use the probabilities ($\boldsymbol{p}$) predicted by our ML model to set $\boldsymbol{\eta}$ values: $\eta_{i,j} = p_{i,j}$, and compare it against the heuristic rule: $\eta_{i,j} = s_j/c_{i,j}$. We also explore a hybrid approach that sets the $\boldsymbol{\eta}$ values to the product of our ML prediction and the heuristic rule: $\eta_{i,j} = p_{i,j} \cdot s_j/c_{i,j}$, for each pair of $i, j = 1, \ldots, n$ and $i \neq j$.

The *pheromone matrix* $\boldsymbol{\tau}$ is another important parameter of ACO. The $\boldsymbol{\tau}$ values are usually initialized uniformly, and are evolved in each iteration of ACO. Instead, we initialize the $\boldsymbol{\tau}$ values by our predicted probabilities, i.e., $\tau_{i,j} = p_{i,j}$. By doing this, better $\boldsymbol{\tau}$ values hopefully can be evolved more quickly, and thus high-quality solutions can be constructed earlier. As the pheromone values of the MMAS algorithm are restricted to a range of $[\tau_{\min}, \tau_{\max}]$, we re-scale the predicted probabilities $\boldsymbol{p}$ to $[\tau_{\min}, \tau_{\max}]$. In addition, if the *pheromone trail smoothing* mechanism is triggered, we re-initialize the $\boldsymbol{\tau}$ values to the rescaled probabilities.

To summarize, we consider three different ways of incorporating our solution prediction into ACO:

1. Set the $\eta_{i,j}$ value to the predicted probability value $\eta_{i,j} = p_{i,j}$, and initialize $\tau_{i,j}$ uniformly;
2. Set the $\eta_{i,j}$ value to the product of the predicted probability and a heuristic rule: $\eta_{i,j} = p_{i,j} \cdot s_j/c_{i,j}$, and initialize $\tau_{i,j}$ uniformly;
3. Set the $\eta_{i,j}$ value by the heuristic rule: $\eta_{i,j} = s_j/c_{i,j}$, and initialize $\tau_{i,j}$ based on the predicted probability value: $\tau_{i,j} = p_{i,j}$.

In our experiments, we will compare our ML-enhanced ACO with the classic ACO that sets the $\eta_{i,j}$ value by the heuristic rule ($\eta_{i,j} = s_j/c_{i,j}$) and initializes $\tau_{i,j}$ uniformly.

4. Experiments

We empirically show the efficacy of our ML models for enhancing the performance of ACO via solution prediction for solving the orienteering problem. Specifically, we explore different ways of integrating ML prediction and ACO in Section 4.1 and compare the effects of using different ML algorithms for solution prediction in Section 4.2. We then test the generalization capability of our model to large synthetic, benchmark, and real-world problem instances in Sections 4.3, 4.4, and 4.5, respectively. Finally, we compare our method against the state-of-the-art algorithms in Section 4.6.

Our source code is publicly available online at https://github.com/yuansuny/MLACO. For the ML algorithms, we use the SVM model implemented in the LIBSVM library (Chang and Lin, 2011), and the LR model implemented in the LIBLINEAR library (Fan et al., 2008). For GCN, we implement it using TensorFlow (Abadi et al., 2015). Our experiments are conducted on a high performance computing server at Monash University — MonARCH, using a NVIDIA Tesla P100 GPU and multiple types of CPUs that are at least 2.40 GHz. Each CPU is equipped with 4 GB memory.

To construct a training set, we generate 100 orienteering problem instances with 50 vertices. For each vertex, we randomly generate a pair of real numbers between 0 and 100 as its coordinates in the Euclidean space. We assign a score of 0 to the starting and ending vertices, and generate a random integer between 0 and 100 as the score for each of the other vertices. The total distance budget (or time budget) is set to an integer randomly generated between 100 and 400 for each problem instance. We then use CPLEX to solve these 100 problem instances, among which 90 are solved to optimality within a cutoff time 10,000 s given to each instance. The total time taken to solve the 90 problem instances to optimality is about 11.5 h if using a single CPU, and the time can be significantly reduced if using multiple CPUs. To train the 'deep' GCN model, we construct a large-sized training set using all the 90 solved problem instances which contains 220,500 training points. To train the 'shallow' LR and SVM models, we only use the first 18 solved problem instances, as using more training data cannot further improve the performance of these models.

4.1. Efficacy of integrating machine learning into ACO

We investigate whether the performance of ACO can be improved by solution prediction. To do so, we train a linear SVM model on our training set, that takes about 31 s. For testing, we generate 100 problem instances, each with 100 vertices, in the same way as we generate the training instances. We use the trained SVM model to predict a probability $p_{i,j}$ for each edge $e_{i,j}$ in a test problem instance. The prediction time is about 0.7 s, which is negligible.

We explore three different ways of incorporating the solution prediction into the probabilistic model of ACO, as shown in Section 3.3. We denote these hybrid models as (1) SVM-ACO$_\eta$, that sets $\eta_{i,j} = p_{i,j}$; (2) SVM-ACO$_{\hat{\eta}}$ that sets $\eta_{i,j} = p_{i,j} \cdot s_j/c_{i,j}$; and (3) SVM-ACO$_\tau$ that initializes $\tau_{i,j}$ based on $p_{i,j}$. We test two ACO variants, AS and MMAS, which are detailed in Section 2.2. The default parameter settings for AS and MMAS are: $\alpha = 1$, $\beta = 1$, $\rho = 0.05$, $\delta = 0.5$, $T_{\text{pts}} = 100$, and $Q = 100$. The values for $\tau_{\max}$ and $\tau_{\min}$ are computed based on Eqs. (11) and (12). For MMAS, the iteration-best solution is used to update the pheromone matrix. These parameter values are selected based on the original papers (Dorigo et al., 1996; Stützle and Hoos, 2000) and our preliminary experimental study (see Appendix A.4 for details).

To show the efficacy of our ML prediction, we first compare the initial probabilistic models of SVM-ACO$_\eta$ and SVM-ACO$_{\hat{\eta}}$ against that of the classic ACO algorithm without ML enhancement. Note that the initial probabilistic model of SVM-ACO$_\tau$ is the same as that of SVM-ACO$_{\hat{\eta}}$ under our parameter settings. Moreover, the initial probabilistic models of the two ACO variants, AS and MMAS are also identical. We use the initial probabilistic models to sample 10,000 solutions for each test problem instance, and plot the distribution of averaged normalized objective values in Fig. 3. The objective values of the sample solutions are normalized by the mean objective value generated by the classic ACO algorithm. The normalized objective values are then averaged across 100 test problem instances. We can observe that the average objective values generated by SVM-ACO$_\eta$ is about 40% better than that of the classic ACO algorithm without ML enhancement. The only difference between these two algorithms is that SVM-ACO$_\eta$ sets $\eta_{i,j}$ based on predicted probability $p_{i,j}$, while ACO sets $\eta_{i,j}$ based on a

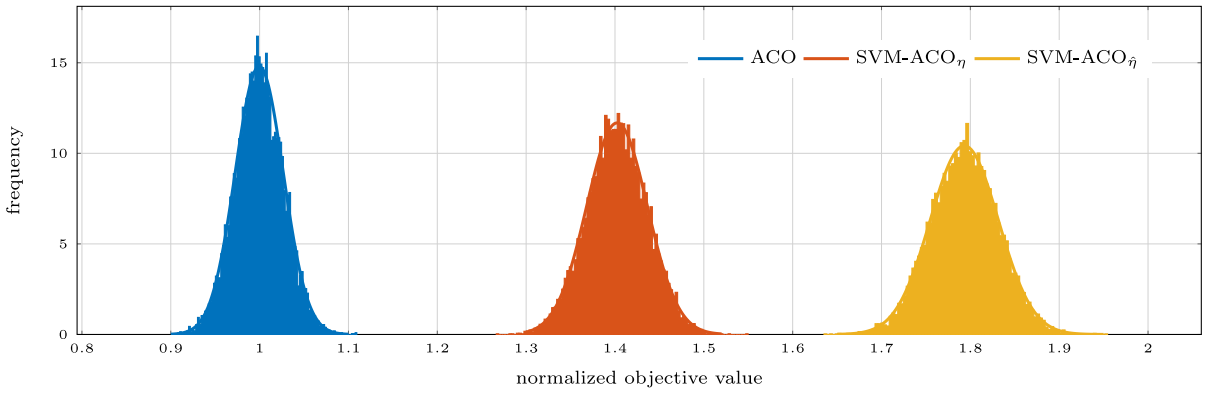


Fig. 3. The distribution of the objective values generated by the ACO, SVM-ACO$_\eta$, and SVM-ACO$_{\hat{\eta}}$ algorithms in the first iteration when tested on the orienteering problems of size 100. The objective values are normalized by the mean objective value generated by ACO.

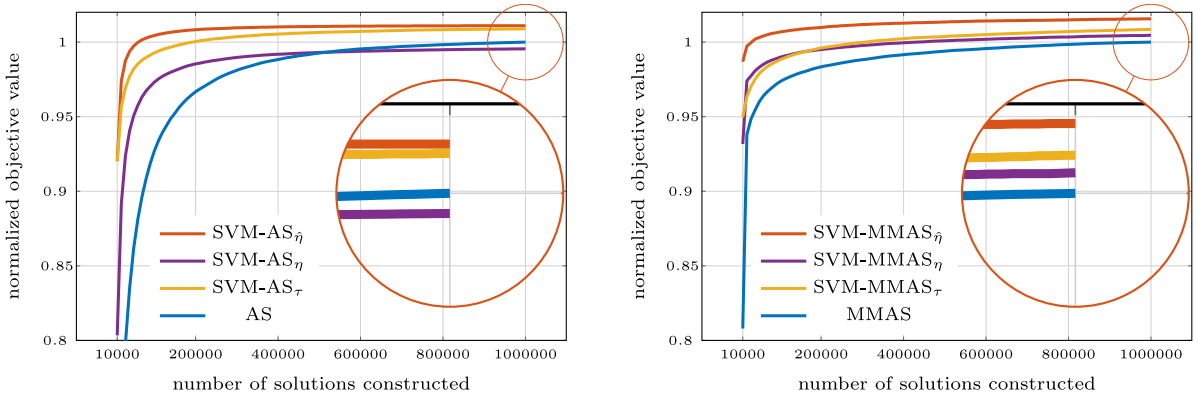


Fig. 4. The convergence curves of the ACO (i.e., AS or MMAS), SVM-ACO$_\tau$, SVM-ACO$_\eta$ and SVM-ACO$_{\hat{\eta}}$ algorithms when used to solve the orienteering problems of size 100. The objective values are normalized by the best objective value found by ACO and are averaged across 100 instances.

heuristic rule $s_j/c_{i,j}$. In this sense, our ML prediction is more 'greedy' than the heuristic rule. Furthermore, by setting $\eta_{i,j}$ to the product of our predicted probability and the heuristic rule, the resulted algorithm SVM-ACO$_{\hat{\eta}}$ improves over ACO by 80% in terms of the objective values generated in the first iteration.

We then compare the ACO algorithms (AS or MMAS) enhanced by ML prediction against the classic ACO algorithms, when solving the test problem instances. The number of solutions to be constructed is set to $10\,000n$ for each algorithm, where n is problem dimensionality. The population size of AS is set to $100n$ and that of MMAS is n, because MMAS only uses a single best solution to update the pheromone matrix in each iteration, and thus it benefits more from relatively small population size and more iterations. The objective values generated by each algorithm are normalized by the best objective value found by AS (or MMAS), and are averaged across 100 problem instances and 25 independent runs. The curves of normalized objective value v.s. number of solutions constructed is shown in Fig. 4. These convergence curves can show not only the final objective values generated by the algorithms but also their converging speed. We can observe that the performances of both AS and MMAS in finding high-quality solutions are greatly enhanced by ML prediction. Significantly, the solution generated by SVM-MMAS$_{\hat{\eta}}$ at 4% of computational budget is already better than the final solution produced by MMAS. Furthermore, the enhanced AS and MMAS algorithms are generally able to find a better solution at the end of a run, except for the SVM-AS$_\eta$ algorithm that may have an issue of premature convergence. We note that the hybridization SVM-ACO$_{\hat{\eta}}$ works the best; it improves over the classic ACO by more than 1% in terms of the final solution quality generated. This improvement is larger if less computational budget is allowed. Hence, we will only employ the hybridization ($\hat{\eta}$) that sets $\eta_{i,j} = p_{i,j} \cdot s_j/c_{i,j}$ in the rest of the paper.

4.2. Sensitivity to machine learning algorithms

We take the SVM-ACO algorithm and replace SVM by LR and GCN to see if the performance of our hybrid algorithm is sensitive to the ML algorithm used in training. We train a separate model with LR and GCN on our training set. For GCN, we use 20 layers and each hidden layer has 32 neurons. The learning rate is set to 0.001 and the number of epochs is 100. The training time for LR is about 27 s and that for GCN is about 1000 s.

Similar as before, we compare the initial probabilistic models of the SVM-ACO, LR-ACO, GCN-ACO and ACO algorithms, and plot the distribution of objective values generated by each probabilistic model in Fig. 5. We can observe that no matter which one of the three learning algorithms is used, the ACO enhanced by solution prediction significantly improves over the classic ACO by more than 50% in terms of the objective values generated in the first iteration. Among the three learning algorithms, the LR performs the worst and SVM is the best. This is a bit surprising as the simple linear SVM model performs slightly better than the deep GCN model in this context. Note that we have not done any fine-tuning for the GCN model, and we suspect that the performance of GCN may be further improved by tuning hyper-parameters. However, a thorough evaluation along this line requires significantly more computational resources and is beyond the scope of this paper.

We also compare the performance of the four algorithms for solving the test problem instances, and the averaged convergence curves are shown in Fig. 6. The results show that the ACO algorithms (i.e., AS or MMAS) enhanced by different ML predictions consistently outperform the classic ACO in finding high-quality solutions. Whilst the initial objective values found by SVM-ACO, LR-ACO, and GCN-ACO are different,

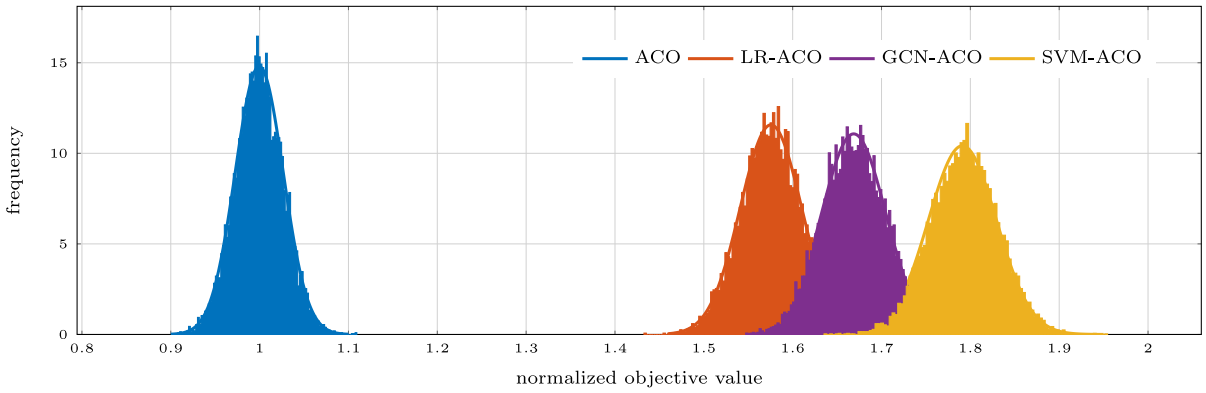


Fig. 5. The distribution of objective values generated by the ACO, SVM-ACO, GCN-ACO, and LR-ACO algorithms in the first iteration when tested on the orienteering problems of size 100. The objective values are normalized by the mean objective value generated by ACO.

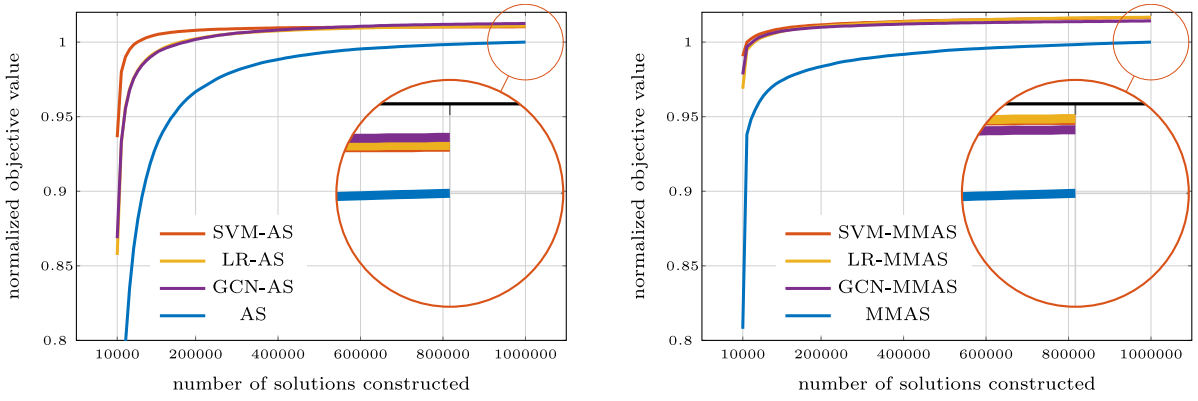


Fig. 6. The convergence curves of the ACO (i.e., AS or MMAS), SVM-ACO, GCN-ACO, and LR-ACO algorithms when used to solve the orienteering problems of size 100. The objective values are normalized by the best objective value found by ACO and are averaged across 100 instances.

Table 1
The best objective values obtained by the AS, SVM-AS, MMAS and SVM-MMAS algorithms averaged across 100 problem instances in each problem set. A series of Wilcoxon signed-rank tests are performed between each pair of algorithms (i.e., AS vs. SVM-AS and MMAS vs. SVM-MMAS), and the p-values are reported. The statistically significantly better results are highlighted in bold (p-value < 0.05).

Dataset	AS	SVM-AS	p-value	MMAS	SVM-MMAS	p-value
size 200	3043.22	**3102.74**	3.02e−12	3068.50	**3254.87**	3.39e−15
size 300	3674.04	**3739.04**	3.42e−12	3637.37	**3944.72**	8.86e−17
size 400	4278.65	**4392.26**	4.05e−15	4207.71	**4665.61**	4.01e−18
size 500	4752.65	**4866.93**	6.10e−14	4612.65	**5167.52**	1.05e−17

the final solutions generated by these algorithms are of a similar quality after many iterations of ACO sampling. In the rest of this paper, we will only use SVM as our ML model.

4.3. Generalization to larger problem instances

We test the generalization of our SVM-ACO algorithm to larger orienteering problem instances. To do so, we randomly generate larger problem instances with dimensionality 200, 300, 400 and 500, each with 100 problem instances, for testing. We then apply the SVM-ACO model, which is trained on small problem instances of dimensionality 50, to solve each of the larger test problem instance, compared against the classic ACO. The parameter settings for the two ACO variants, AS and MMAS are the same as before.

The best objective values obtained by each algorithm averaged across 100 problem instances for each problem set are presented in Table 1, and the averaged convergence curves are shown in Fig. 7. First, we can observe that our ML model trained on small problem instances generalizes very well to larger test problem instances, in the sense that it consistently boosts the performance of both AS and MMAS when solving the larger problem instances. Furthermore, this improvement becomes more significant as the problem dimensionality increases from 200 to 500. The SVM-MMAS algorithm is clearly the best performing one among the four algorithms tested. Significantly, SVM-MMAS improves over MMAS by more than 10% in terms of the best objective values generated for the problem instances of dimensionality 500.

4.4. Generalization to benchmark problem instances

We evaluate the generalization capability of our SVM-ACO algorithm on a set of benchmark instances used in Chao et al. (1996). Each instance has 66 vertices, the locations of which form a square shape. The distance budget is varied from 5 to 130 in increments of 5, resulting in 26 instances in total. We apply the SVM-ACO algorithm, trained on randomly generated problem instances, to solve the square-shaped benchmark problem instances, compared against the classic ACO under the same parameter settings.

The averaged convergence curves of the algorithms when used to solve the benchmark problem instances are shown in Fig. 8. The results show that our ML model trained on randomly generated instances generalizes well to the square-shaped benchmark instances. In particular, the SVM-MMAS algorithm is able to find a high-quality solution at the very early stage of the search process, comparing to MMAS. The mean and standard deviation of the best objective values generated by each algorithm across 25 runs for each problem instance are presented in Table 2. The statistically significantly better results obtained by the SVM-ACO and ACO algorithms are highlighted in bold, based on the

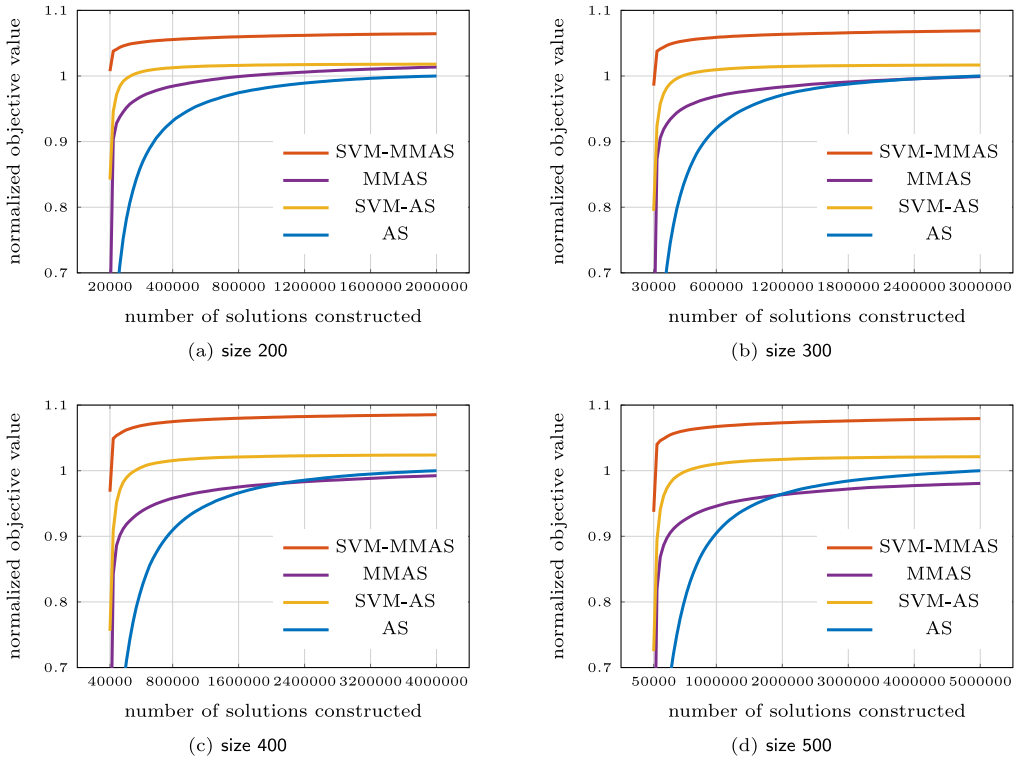


Fig. 7. The convergence curves of the AS, SVM-AS, MMAS and SVM-MMAS algorithms, when used to solve the larger orienteering problem instances. The objective values are normalized by the best objective value found by AS and are averaged across 100 instances.

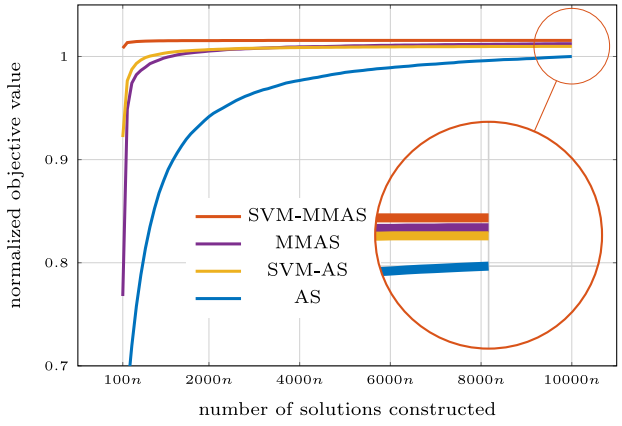


Fig. 8. The convergence curves of the AS, SVM-AS, MMAS and SVM-MMAS algorithms, when used to solve the benchmark problem instances. The objective values are normalized by the best objective value found by AS and are averaged across 26 instances.

Wilcoxon signed-rank tests with a significance level of 0.05. We can observe that on easy instances in which the distance budget is small, both the SVM-ACO and ACO algorithms are able to find the optimal solutions by the end of a run consistently. However, on hard instances in which the distance budget is large, the solution quality generated by SVM-ACO is statistically significantly better than that by the classic ACO algorithm.

4.5. Generalization to real-world problem instances

We further test the generalization of our SVM-ACO algorithm to real-world problem instances — tourist trip planning, where the goal is to plan an itinerary that visits a subset of attractions in a city within a given distance (or time) budget such that the total collected 'popularity score' is maximized. We use six datasets published in a trajectory-driven tourist trip planning system (Wang et al., 2018) to test our model. Each

Table 2
The best objective values generated by the AS, SVM-AS, MMAS and SVM-MMAS algorithms on the benchmark problem instances. The statistically significantly better results generated by SVM-AS as opposed to AS (and SVM-MMAS as opposed to MMAS) are highlighted in bold, according to Wilcoxon signed-rank tests with a significance level of 0.05.

Datasets	Budget	AS		SVM-AS		MMAS		SVM-MMAS	
		Mean	std	Mean	std	Mean	std	Mean	std
set_66_1_005	5	10.00	0.00	10.00	0.00	10.00	0.00	10.00	0.00
set_66_1_010	10	40.00	0.00	40.00	0.00	40.00	0.00	40.00	0.00
set_66_1_015	15	120.00	0.00	120.00	0.00	120.00	0.00	120.00	0.00
set_66_1_020	20	205.00	0.00	205.00	0.00	205.00	0.00	205.00	0.00
set_66_1_025	25	290.00	0.00	290.00	0.00	290.00	0.00	290.00	0.00
set_66_1_030	30	400.00	0.00	400.00	0.00	400.00	0.00	400.00	0.00
set_66_1_035	35	465.00	0.00	465.00	0.00	465.00	0.00	465.00	0.00
set_66_1_040	40	575.00	0.00	575.00	0.00	575.00	0.00	575.00	0.00
set_66_1_045	45	647.60	2.55	**649.20**	1.87	648.60	2.29	**650.00**	0.00
set_66_1_050	50	730.00	0.00	730.00	0.00	729.20	3.12	730.00	0.00
set_66_1_055	55	820.40	2.00	**823.60**	3.39	823.00	2.50	**824.80**	1.00
set_66_1_060	60	904.80	6.84	**910.20**	5.68	914.20	2.36	915.00	0.00
set_66_1_065	65	972.40	5.02	**976.60**	5.72	980.00	0.00	980.00	0.00
set_66_1_070	70	1055.00	8.42	**1065.20**	8.72	1069.60	1.38	1070.00	0.00
set_66_1_075	75	1126.00	9.46	**1138.60**	4.45	1140.00	0.00	1140.00	0.00
set_66_1_080	80	1188.20	10.09	**1206.80**	8.52	1209.20	6.24	**1215.00**	0.00
set_66_1_085	85	1242.80	6.47	**1265.20**	6.84	1264.60	3.20	**1270.00**	0.00
set_66_1_090	90	1302.60	9.03	**1322.80**	10.91	1329.80	6.84	**1340.00**	0.00
set_66_1_095	95	1354.20	10.58	**1379.40**	10.24	1386.60	6.25	**1394.80**	1.00
set_66_1_100	100	1405.60	8.33	**1436.40**	11.23	1447.20	9.36	**1464.60**	2.00
set_66_1_105	105	1452.00	11.64	**1484.40**	10.64	1508.20	8.65	**1519.20**	2.77
set_66_1_110	110	1496.60	10.48	**1529.60**	9.46	1549.40	7.82	**1560.00**	0.00
set_66_1_115	115	1544.00	10.80	**1576.80**	9.45	1583.80	8.07	**1594.80**	1.00
set_66_1_120	120	1582.40	8.55	**1617.40**	9.03	1623.00	8.29	**1634.60**	2.00
set_66_1_125	125	1615.40	6.91	**1654.60**	9.46	1654.60	5.94	**1670.00**	0.00
set_66_1_130	130	1649.40	5.83	**1675.80**	4.72	1675.00	3.82	**1680.00**	0.00

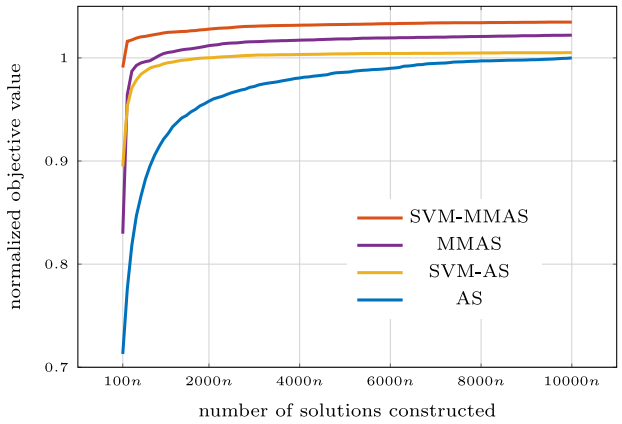


Fig. 9. The convergence curves of the AS, SVM-AS, MMAS and SVM-MMAS algorithms, when used to solve the real-world problem instances. The objective values are normalized by the best objective value found by AS and are averaged across six instances.

Table 3
The best objective values generated by the AS, SVM-AS, MMAS and SVM-MMAS algorithms on the real-world problem instances. The statistically significantly better results generated by SVM-AS as opposed to AS (and SVM-MMAS as opposed to MMAS) are highlighted in bold, according to Wilcoxon signed-rank tests with a significance level of 0.05.

Datasets	Dimension	AS		SVM-AS		MMAS		SVM-MMAS	
		Mean	std	Mean	std	Mean	std	Mean	std
Berlin	97	188.39	1.08	**191.00**	1.70	192.63	0.93	**195.46**	0.64
Copenhagen	81	227.28	1.90	228.01	2.14	232.21	1.61	**235.62**	1.79
Istanbul	154	206.11	1.68	**208.36**	1.79	209.11	1.25	**215.12**	1.25
London	114	172.88	0.92	172.52	0.70	175.82	0.64	176.02	1.52
Paris	117	153.81	0.76	153.47	2.21	158.83	1.41	159.19	1.78
Prague	78	242.73	1.34	**244.52**	1.62	248.48	1.26	**251.73**	1.20

dataset corresponds to a city in Europe. The starting and ending vertex of a dataset is a hotel randomly chosen from the corresponding city, and the other vertices are attractions for visiting. The coordinates of each vertex are its geographic location: latitude and longitude; and the distance between two vertices is the geographical distance between them. The popularity score of an attraction is calculated based on how many people have visited that attraction: $s_i = \log_2(n_i+1)$, where n_i is the number of trajectories that have been to that attraction. The trajectory data was originally crawled from the Triphobo website by Wang et al. (2018). The dimensionality of these six datasets varies from 81 to 154. The total distance budget is set to 50 kilometers for each dataset. We take the SVM-ACO model trained on synthetic problem instances and test it on these real-world problem instances.

The average convergence curves of the AS, SVM-AS, MMAS and SVM-MMAS algorithms when used to solve the real-world problem instances are shown in Fig. 9. The results show that our ML model trained on synthetic problem instances generalizes well to real-world problem instances; the ML model speeds up both AS and MMAS in finding high-quality solutions for the real-world instances. Overall, the SVM-MMAS algorithm achieves the best solution quality, and improves over MMAS by 1.24% on average. The mean and standard deviation of the best objective values generated by each algorithm across 25 runs for each problem instance are presented in Table 3. We can see that both AS and MMAS consistently find an equally good or statistically significantly better solution when enhanced by our ML prediction.

4.6. Comparison with state-of-the-art algorithms

We take the best-performing algorithm SVM-MMAS and compare it against three state-of-the-art heuristics, Evolutionary Algorithm for Orienteering Problem (EA4OP) (Kobeaga et al., 2018), GRASP with Path Relinking (GRASP-PR) (Campos et al., 2014) and 2-Parameter Iterative Algorithm (2P-IA) (Silberholz and Golden, 2010). These three state-of-the-art algorithms all use effective local search methods. For a fair comparison, we also use a local search method to improve the solutions sampled by SVM-MMAS. Specifically, we use the well-known 2-opt local search method (Lin, 1965) to reduce the path length of the best solution constructed in each iteration of SVM-MMAS. Candidate vertices (i.e., those have not been visited) are greedily inserted into the path until a local optimum is found. Note that the 2-opt local search method is repeatedly applied when a candidate vertex is inserted into the path, attempting to reduce the path length.

We compare the performance of our algorithm (denoted as SVM-MMAS-LS) with the state-of-the-art heuristics on a set of benchmark problem instances (Fischetti et al., 1998). These instances (generation 3) were generated based on the TSP library, and are available from the OP library (https://github.com/bcamath-ds/OPLib). For each problem instance, we run our SVM-MMAS-LS algorithm 10 times, and record the best solution found and the average runtime used, following (Kobeaga et al., 2018). The parameters of our algorithm are set as before with four exceptions:

1. We use a different termination criterion for our SVM-MMAS-LS algorithm, i.e., if the best solution found cannot be improved for T_{ter} consecutive iterations, the algorithm is terminated. We will test two different values for T_{ter}: 200 and 500.
2. We have tested the use of both the iteration-best solution and the global-best solution to update the pheromone matrix, and found that using the global-best solution can generate an optimality gap that is 44% better than that of using the iteration-best solution on average. Therefore, we will use the global-best solution to update the pheromone matrix.
3. We have tested three different population sizes {50, 100, n} and found that using a larger population size can generate a slightly smaller optimality gap but significantly increases the runtime. Hence, we will set the population size to 50.
4. We use the SVM model (primal formulation) implemented in the LIBLINEAR library, which is faster than that of the LIBSVM library (dual formulation) in prediction. Furthermore, we reduce the sample size m to $10n$, to gain computational efficiency.

The results for the medium-sized instances of which the number of vertices ranges from 48 to 400 are presented in Table 4. Note that the results of EA4OP, GRASP-PR and 2P-IA are taken from Kobeaga et al. (2018), which are generated on a workstation with Intel(R) Xeon(R) E5-2609 v3 @ 1.90 GHz Processor and 4 GB RAM, while the results of our SVM-MMAS-LS algorithm are obtained on a server with Xeon-Gold-6150 @ 2.70 GHz Processor and 4 GB RAM. Hence, the runtimes of the algorithms cannot be directly compared. In terms of solution quality found, we can observe that our SVM-MMAS-LS algorithm with $T_{\text{ter}} = 200$ is already competitive with the three state-of-the-art heuristics. On average, SVM-MMAS-LS achieves a smaller optimality gap than the other three algorithms. When T_{ter} is increased from 200 to 500, the average optimality gap can be further reduced, but at an expense of longer runtime. For larger problem instances (with $|V| > 400$), our SVM-MMAS-LS algorithm can generate better solution quality than the 2P-IA and GRASP-PR algorithms but does not outperform the EA4OP algorithm in general (see Appendix A.5 for the detailed results).

Finally, we would like to remark that our primary aim is *not* to fine-tune our proposed method to outperform the state-of-the-art algorithms for solving the orienteering problem. Instead, our aim is to boost the performance of ACO in general via solution prediction and ML. Therefore, our experiments have mainly focused on showing whether the ML-enhanced ACO algorithm outperforms the classic ACO. In fact, our ML-enhanced ACO is not confined to solving the orienteering problem. In Appendix B, we adapt our ML-enhanced ACO algorithm to solve the maximum weighted clique problem and show that it is competitive compared to the state-of-the-art algorithms.

Table 4
The comparison between our SVM-MMAS-LS algorithm and three state-of-the-art heuristics on the medium-sized benchmark problem instances. The column 'Opt' presents the optimal objective value. For each algorithm, the best objective value found, optimality gap (%) and the average runtime (in second) are presented. The best optimality gap is highlighted in bold. The last row denotes the number of instances on which an algorithm achieves the best optimality gap. Note that the results for 2P-IA, GRASP-PR and EA4OP are taken from Kobeaga et al. (2018), and thus the runtimes are not comparable.

Instance	Opt	2P-IA			GRASP-PR			EA4OP			SVM-MMAS-LS (200)			SVM-MMAS-LS (500)		
		Best	Gap	Time	Best	Gap	Time	Best	Gap	Time	Best	Gap	Time	Best	Gap	Time
att48	1 049	1 049	**0.00**	0.13	1 049	**0.00**	0.18	1 049	**0.00**	0.26	1 049	**0.00**	0.22	1 049	**0.00**	0.39
gr48	1 480	1 480	**0.00**	0.07	1 480	**0.00**	0.20	1 480	**0.00**	0.13	1 480	**0.00**	0.21	1 480	**0.00**	0.43
hk48	1 764	1 764	**0.00**	0.09	1 764	**0.00**	0.14	1 764	**0.00**	0.22	1 764	**0.00**	0.23	1 764	**0.00**	0.36
eil51	1 399	1 399	**0.00**	0.12	1 399	**0.00**	0.17	1 398	0.07	0.22	1 399	**0.00**	0.25	1 399	**0.00**	0.50
berlin52	1 036	1 036	**0.00**	0.19	1 036	**0.00**	0.30	1 034	0.19	0.64	1 036	**0.00**	0.20	1 036	**0.00**	0.39
brazil58	1 702	1 702	**0.00**	0.13	1 702	**0.00**	0.33	1 702	**0.00**	0.71	1 702	**0.00**	0.29	1 702	**0.00**	0.48
st70	2 108	2 108	**0.00**	0.24	2 108	**0.00**	0.37	2 108	**0.00**	0.31	2 108	**0.00**	0.36	2 108	**0.00**	0.79
eil76	2 467	2 461	0.24	0.30	2 462	0.20	0.44	2 467	**0.00**	0.36	2 462	0.20	0.51	2 467	**0.00**	1.05
pr76	2 430	2 430	**0.00**	0.26	2 430	**0.00**	0.56	2 430	**0.00**	0.57	2 430	**0.00**	0.48	2 430	**0.00**	1.12
gr96	3 170	3 170	**0.00**	0.39	3 153	0.54	1.07	3 166	0.13	1.41	3 170	**0.00**	0.93	3 170	**0.00**	1.49
rat99	2 908	2 896	0.41	0.47	2 881	0.93	0.80	2 886	0.76	0.78	2 870	1.31	0.54	2 908	**0.00**	1.32
kroA100	3 211	3 211	**0.00**	0.30	3 211	**0.00**	1.16	3 180	0.97	0.38	3 206	0.16	0.76	3 211	**0.00**	1.69
kroB100	2 804	2 804	**0.00**	0.46	2 804	**0.00**	1.34	2 785	0.68	0.51	2 804	**0.00**	0.64	2 804	**0.00**	1.29
kroC100	3 155	3 155	**0.00**	0.38	3 149	0.19	0.86	3 155	**0.00**	0.44	3 149	0.19	0.70	3 140	0.48	1.20
kroD100	3 167	3 123	1.39	0.65	3 167	**0.00**	1.18	3 141	0.82	0.58	3 147	0.63	0.58	3 151	0.51	1.55
kroE100	3 049	3 027	0.72	0.56	3 049	**0.00**	1.48	3 049	**0.00**	0.47	3 049	**0.00**	0.58	3 049	**0.00**	1.15
rd100	2 926	2 924	0.07	0.62	2 924	0.07	0.90	2 923	0.10	0.48	2 926	**0.00**	0.71	2 926	**0.00**	1.85
eil101	3 345	3 333	0.36	0.46	3 322	0.69	0.76	3 345	**0.00**	0.56	3 335	0.30	0.92	3 322	0.69	2.02
lin105	2 986	2 986	**0.00**	0.54	2 986	**0.00**	1.89	2 973	0.44	2.09	2 986	**0.00**	0.68	2 986	**0.00**	1.92
pr107	1 877	1 877	**0.00**	0.29	1 877	**0.00**	1.15	1 802	4.00	0.82	1 875	0.11	0.46	1 877	**0.00**	0.76
gr120	3 779	3 736	1.14	0.96	3 745	0.90	1.15	3 748	0.82	1.36	3 687	2.43	1.23	3 765	**0.37**	3.07
pr124	3 557	3 517	1.12	0.62	3 549	**0.22**	2.41	3 455	2.87	0.88	3 549	**0.22**	0.98	3 549	**0.22**	1.63
bier127	2 365	2 356	0.38	1.08	2 332	1.40	2.07	2 361	**0.17**	2.62	2 336	1.23	1.57	2 347	0.76	3.62
pr136	4 390	4 390	**0.00**	0.93	4 380	0.23	2.56	4 390	**0.00**	1.13	4 312	1.78	1.31	4 299	2.07	2.85
gr137	3 954	3 928	0.66	1.13	3 926	0.71	1.89	3 954	**0.00**	1.88	3 932	0.56	1.80	3 934	0.51	2.64
pr144	3 745	3 633	2.99	0.77	3 745	**0.00**	3.36	3 700	1.20	2.41	3 745	**0.00**	0.96	3 745	**0.00**	2.11
kroA150	5 039	5 037	**0.04**	1.26	5 018	0.42	3.06	5 019	0.40	1.07	5 011	0.56	1.56	5 034	0.10	3.71
kroB150	5 314	5 267	0.88	1.31	5 272	0.79	2.31	5 314	**0.00**	1.04	5 177	2.58	1.68	5 253	1.15	4.12
pr152	3 905	3 557	8.91	0.80	3 905	**0.00**	4.07	3 902	0.08	3.62	3 905	**0.00**	1.37	3 905	**0.00**	2.85
u159	5 272	5 272	**0.00**	1.33	5 272	**0.00**	4.46	5 272	**0.00**	0.94	5 214	1.10	1.48	5 218	1.02	3.97
rat195	6 195	6 174	**0.34**	2.22	6 086	1.76	3.06	6 139	0.90	2.00	6 127	1.10	3.00	6 125	1.13	6.46
d198	6 320	5 985	5.30	1.86	6 162	2.50	5.86	6 290	**0.47**	7.14	6 258	0.98	2.71	6 212	1.71	5.46
kroA200	6 123	6 048	1.22	2.73	6 084	0.64	4.64	6 114	**0.15**	1.72	5 971	2.48	2.39	6 028	1.55	7.14
kroB200	6 266	6 251	**0.24**	2.79	6 190	1.21	5.46	6 213	0.85	1.77	6 163	1.64	3.17	6 226	0.64	4.78
gr202	8 616	8 111	5.86	2.05	8 419	2.29	9.12	8 605	**0.13**	10.45	8 464	1.76	3.79	8 542	0.86	9.68
ts225	7 575	7 149	5.62	1.47	7 510	0.86	6.15	7 575	**0.00**	1.14	7 486	1.17	3.12	7 575	**0.00**	9.80
tsp225	7 740	7 353	5.00	2.38	7 565	2.26	5.04	7 488	3.26	2.58	7 607	1.72	3.74	7 681	**0.76**	11.50
pr226	6 993	6 652	4.88	1.97	6 964	**0.41**	15.50	6 908	1.22	8.01	6 950	0.61	3.37	6 937	0.80	6.10
gr229	6 328	6 190	2.18	4.42	6 205	1.94	9.03	6 297	**0.49**	11.65	6 135	3.05	5.98	6 197	2.07	17.40
gil262	9 246	8 915	3.58	5.68	8 922	3.50	6.07	9 094	1.64	3.94	9 090	1.69	7.57	9 116	**1.41**	22.60
pr264	8 137	7 820	3.90	3.98	7 959	2.19	17.88	8 068	0.85	3.62	8 118	**0.23**	4.21	8 086	0.63	7.44
a280	9 774	8 719	10.79	4.53	9 426	3.56	9.42	8 684	11.15	3.22	9 609	**1.69**	4.83	9 587	1.91	13.55
pr299	10 343	10 305	**0.37**	6.07	10 033	3.00	19.61	9 959	3.71	3.95	10 227	1.12	5.40	10 254	0.86	16.01
lin318	10 368	9 909	4.43	7.57	9 758	5.88	12.18	10 273	**0.92**	6.33	10 155	2.05	9.05	10 271	0.94	21.95
rd400	13 223	12 828	2.99	14.49	12 678	4.12	16.46	13 088	**1.02**	7.74	12 552	5.07	12.15	12 848	2.84	53.39
Average	4 724	4 601	1.69	1.80	4 646	0.96	4.18	4 661	0.90	2.31	4 661	0.88	2.19	4 683	**0.58**	5.90
# best	–	–	20	–	–	19	–	–	22	–	–	18	–	–	**24**	–

5. Conclusion

We have proposed a new meta-heuristic called ML-ACO that integrates machine learning (ML) with ant colony optimization (ACO) to solve the orienteering problem. Our ML model trained on optimally-solved problem instances, is able to predict which edges in the graph of a test problem instance are more likely to be part of the optimal route. We incorporated the ML predictions into the probabilistic model of ACO to bias its sampling towards using the predicted 'high-quality' edges more often when constructing solutions. This in turn significantly boosted the performance of ACO in finding high-quality solutions for a test problem instance. We tested three different classification algorithms, and the experimental results showed that all of the ML-enhanced ACO variants significantly improved the classic ACO in terms of both speed of convergence and quality of the final solution. Of the three ML models, the SVM based predictions produced the best results for this application. The best results were obtained by using the prediction to modify the heuristic weights rather than just for the initial pheromone matrix. Importantly, our ML model trained on small synthetic problem instances generalized very well to large synthetic and real-world problem instances.

We see great potential of the integration between ML (more specifically solution prediction) and meta-heuristics, and a lot of opportunities for future work. First, there is a large family of meta-heuristics, that can potentially be improved by solution prediction. Second, it would be interesting to see if this integrated technique also works on other combinatorial optimization problems as well as continuous, dynamic or multi-objective optimization problems. In particular, we expect this integrated technique would be more effective in solving a dynamic problem where the optimal solution changes over time. Based on the results shown in this paper, solution prediction is very greedy, and therefore can potentially adapt quickly to any changes occurring in a problem. Third, there is a large number of ML algorithms that can be used for solution prediction. Meta-heuristics will certainly benefit more from this type of hybridization, if we can further improve the accuracy of solution prediction. This paper shows that SVM, one of the simpler ML models, is already highly effective. However, given the large number of advanced ML methods developed in recent years, there

may be others that are even more effective in this context of boosting meta-heuristics.

CRediT authorship contribution statement

Yuan Sun: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing – original draft. **Sheng Wang:** Software, Investigation, Writing – original draft. **Yunzhuang Shen:** Software, Investigation, Writing – original draft. **Xiaodong Li:** Methodology, Writing – original draft, Funding acquisition. **Andreas T. Ernst:** Methodology, Writing – original draft, Funding acquisition. **Michael Kirley:** Methodology, Writing – original draft.

Acknowledgment

This work was supported by an ARC (Australian Research Council) Discovery Grant (DP180101170).

Appendix A. Supplementary methodology and experimental results

A.1. A random sampling method for the orienteering problem

Consider an orienteering problem instance $G(V, E, S, C)$ with a given time budget $T_{\max}$. Without loss of generality, we assume v_1 is the starting vertex and v_n is the ending vertex. The main steps of our random sampling method to generate one feasible solution (route) are:

1. Initialize a route with the starting vertex v_1;
2. Generate a random permutation of the candidate vertices $\{v_2, \dots, v_{n-1}\}$ that can be visited;
3. Consider the vertices in the generated permutation one by one, and add the vertices to the sample route which does not violate the time budget constraint;
4. Add the ending vertex v_n to the sample route.

The pseudocode of the random sampling method is presented in Algorithm 1. It is obvious that the time complexity of generating one sample route by using this method is $\mathcal{O}(n)$, where n is the number of vertices in a problem instance. Hence, the total time complexity of generating m sample routes is $\mathcal{O}(mn)$. Furthermore, the sample size m should be larger than n; otherwise there will be some edges that are never sampled. This is because the number of edges in the directed complete graph is $n(n-1)$, and the total number of edges in m sample routes is no more than mn. Therefore, each edge is expected to be sampled no more than $m/(n-1)$ times.

Algorithm 1 Random Sampling Method

Require: vertex set V, vertex score set S, edge cost set C, time budget $T_{\max}$, number of samples to generate m.

for k from 1 to m **do**
  Initialize the route P^k with the starting vertex v_1;
  Initialize the object value $y^k \leftarrow S[v_1]$;
  Initialize the current vertex $v_c \leftarrow v_1$;
  Initialize the time used so far $t_c \leftarrow 0$;
  Generate a random permutation of $\{v_2, \dots, v_{n-1}\}$;
  for v_j in the generated random permutation **do**
    if $t_c + C[v_c, v_j] + C[v_j, v_n] \le T_{\max}$ **then**
      Add v_j to the route P_k;
      Update $y^k \leftarrow y^k + S[v_j]$;
      Update $t_c \leftarrow t_c + C[v_c, v_j]$;
      Update $v_c \leftarrow v_j$;
  Add v_n to the route P_k;
  Update $y^k \leftarrow y^k + S[v_n]$;
return $\{P^1, \dots, P^m\}$ and $\{y^1, \dots, y^m\}$.

A.2. An efficient method for computing the statistical measures

In the main paper, we used a binary string $\boldsymbol{x}$ to represent a sample solution (route), where $x_{i,j} = 1$ if the edge $e_{i,j}$ is in the route; otherwise $x_{i,j} = 0$. We have shown that directly computing the ranking-based measure and correlation-based measure based on the binary string representation $\mathbf{x}$ costs $\mathcal{O}(mn^2)$. Here, we adapt the method proposed in Sun et al. (2021b) to efficiently compute the two statistical measures based on set representation P, which only stores the edges appearing in the corresponding sample route.

Let $\{P^1, \dots, P^m\}$ be the set representation of the m randomly generated solutions; $\{\boldsymbol{x}^1, \dots, \boldsymbol{x}^m\}$ be the corresponding binary string representation; and $\{y^1, \dots, y^m\}$ be their objective values. Because $x^k_{i,j}$ are binary variables, we can simplify the calculation of Pearson correlation coefficient by using the following two equalities:

$$\sum_{k=1}^{m}(x^k_{i,j} - \bar{x}_{i,j})^2 = \bar{x}_{i,j}(1 - \bar{x}_{i,j})m, \tag{22}$$

Algorithm 2 Computing Statistical Measures

Require: samples $\mathbb{P}$, objective values Y, number of samples m, number of vertices n, and edge set E.

1: Sort the samples in $\mathbb{P}$ based on objective value Y; and use r^k to denote the ranking of kth sample P^k;
2: Compute mean objective value: $\bar{y} \leftarrow \sum_{k=1}^{m} y^k/m$;
3: Compute objective difference: $y_d \leftarrow \sum_{k=1}^{m}(y^k - \bar{y})$;
4: Compute objective variance: $\sigma_y \leftarrow \sum_{k=1}^{m}(y^k - \bar{y})^2$;
5: Initialize f_r, $\bar{x}_{i,j}$ and $s^1_{i,j}$ to 0, for each $e_{i,j} \in E$;
6: **for** k from 1 to m **do**
7: **for** idx from 1 to $|P^k| - 1$ **do**
8: $i \leftarrow P^k[idx]$, $j \leftarrow P^k[idx+1]$;
9: $f_r(e_{i,j}) \leftarrow f_r(e_{i,j}) + 1/r^k$;
10: $\bar{x}_{i,j} \leftarrow \bar{x}_{i,j} + 1/m$;
11: $s^1_{i,j} \leftarrow s^1_{i,j} + (y^k - \bar{y})$;
12: **for** i from 1 to n **do**
13: **for** j from 1 to n and $j \neq i$ **do**
14: $\sigma_{c_{i,j}} \leftarrow (1 - \bar{x}_{i,j})s^1_{i,j} - \bar{x}_{i,j}(y_d - s^1_{i,j})$;
15: $\sigma_{x_{i,j}} \leftarrow \bar{x}_{i,j}(1 - \bar{x}_{i,j})m$;
16: $f_c(e_{i,j}) \leftarrow \sigma_{c_{i,j}} / \sqrt{\sigma_{x_{i,j}}\sigma_y}$;
17: **return** f_r and f_c.

$$\sum_{k=1}^{m}(x^k_{i,j} - \bar{x}_{i,j})(y^k - \bar{y}) = (1 - \bar{x}_{i,j})s^1_{i,j} - \bar{x}_{i,j}s^0_{i,j}, \tag{23}$$

where $\bar{x}_{i,j} = \sum_{k=1}^{m} x^k_{i,j}/m$, $\bar{y} = \sum_{k=1}^{m} y^k/m$ and

$$s^1_{i,j} = \sum_{\substack{1\le k\le m \\ x^k_{i,j}=1}} (y^k - \bar{y}); \text{ and } s^0_{i,j} = \sum_{\substack{1\le k\le m \\ x^k_{i,j}=0}} (y^k - \bar{y}). \tag{24}$$

The proof of these two equalities can be found in Sun et al. (2021b). We then are able to compute the two statistical measures in $\mathcal{O}(mn + n^2)$ by using Algorithm 2. Computing our ranking-based measure f_r based on the set representation is straightforward, i.e., scanning through the edges in each sample route P to accumulate the rankings. To compute the correlation-based measure, we first iterate through the edges in each sample route P to accumulate $\bar{x}_{i,j}$ and $s^1_{i,j}$, i.e., line 6 to 11 in Algorithm 2. Our correlation-based measure f_c can then be easily computed based on $\bar{x}_{i,j}$ and $s^1_{i,j}$ (line 12 to 16 in Algorithm 2).

A.3. A brief introduction of the machine learning algorithms used

Support Vector Machine (SVM): Given a training set $\mathbb{S} = \{(\boldsymbol{f}^i, l^i) \mid i = 1, \ldots, n_t\}$, the aim of SVM is to find a decision boundary ($\boldsymbol{w}^T \boldsymbol{f} + b = 0$) in the feature space to maximize the so-called geometric margin, defined as the smallest distance from a training point to the decision boundary (Boser et al., 1992; Cortes and Vapnik, 1995). We use an L2-regularized linear SVM model, that finds the optimal decision boundary by solving the following quadratic programming with linear constraints:

$$\min_{\boldsymbol{w},b,\xi} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + r^+ \sum_{l^i=1} \xi^i + r^- \sum_{l^i=-1} \xi^i, \tag{25}$$

$$s.t. \quad l^i\left(\boldsymbol{w}^T \boldsymbol{f}^i + b\right) \geq 1 - \xi^i, \quad i = 1, \ldots, n_t, \tag{26}$$

$$\xi^i \geq 0, \quad i = 1, \ldots, n_t, \tag{27}$$

where $r^+ > 0$ and $r^- > 0$ are the regularization parameters for positive and negative training points; and ξ^i, $i = 1, \ldots, n_t$ are slack variables.

Logistic Regression (LR) uses a loss function derived from the logistic function $g(x) = 1/(1 + e^{-x})$, whose output is in between $[0, 1]$ and can be interpreted as probability. LR aims to separate positive and negative training points by maximum likelihood estimation (Bishop, 2006). We use an L2-regularized LR model that fits its parameters ($\boldsymbol{w}$, b) by solving the following optimization problem:

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + r^+ \sum_{l^i=1} \log_2(1 + e^{-\boldsymbol{w}^T \boldsymbol{f}^i - b}) + r^- \sum_{l^i=-1} \log_2(1 + e^{\boldsymbol{w}^T \boldsymbol{f}^i + b}). \tag{28}$$

Graph Convolutional Network (GCN) is a convolutional neural network that makes use of graph structure when classifying vertices in a graph (Kipf and Welling, 2017). Consider a simple GCN model with only two layers: the input layer contains feature vectors $\boldsymbol{f}$ and the output layer is a predicted scalar z for a vertex in a graph. To compute z_i for vertex v_i in a graph, GCN aggregates its feature vector $\boldsymbol{f}^i$ with that of its neighbors $\mathcal{N}_i$:

$$z_i = \boldsymbol{w}_0 \boldsymbol{f}^i + \boldsymbol{w}_1 \sum_{v_j \in \mathcal{N}_i} \sqrt{d_i d_j}\, \boldsymbol{f}^j, \tag{29}$$

where d_i and d_j are the degrees of vertex v_i and v_j; $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ are the weights to be optimized. This two-layer GCN model is a simple linear classifier, which is not expected to work well in practice. Thus, we usually use multiple hidden layers between the input and output layers, and each hidden layer can have multiple 'neurons'. A hidden layer basically takes the output of its previous layer as input, and performs a linear transformation of its input. The intermediate output of the linear transformation is then filtered by an activation function to make GCN a non-linear classifier. In our experiments, the GCN model consists of 20 layers and each hidden layer has 32 neurons. The activation function used is the ReLU function (Nair and Hinton, 2010), defined as $\text{ReLU}(x) = \max(0, x)$. The weights ($\boldsymbol{w}$) of GCN are optimized via stochastic gradient descent with L2-regularized cross-entropy loss function (Kipf and Welling, 2017). In the case of binary classification, the cross-entropy loss function is identical to the loss function of the LR algorithm:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + r^+ \sum_{l^i=1} \log_2(1 + e^{-z_i}) + r^- \sum_{l^i=-1} \log_2(1 + e^{z_i}), \tag{30}$$

where $\boldsymbol{w}$ is a vector of all GCN's weights to be optimized, and z_i is the output (prediction) of GCN for the ith training point. Because our training points are edges instead of vertices in the graphs of the orienteering problem instances, the GCN model cannot be directly applied to make predictions for edges in the graphs. To tackle this, we transfer the original graph (G) to its line graph ($\bar{G}$) such that the edges in G are now vertices in $\bar{G}$ and the neighboring edges (i.e., edges sharing a common vertex) in G are now neighboring vertices (i.e., vertices sharing a common edge) in $\bar{G}$. We then can apply the GCN model on the line graph $\bar{G}$ to make predictions for the edges in the original graph G.

Table 5
The mean and standard deviation of the best objective values generated by the AS and MMAS algorithms with different parameter settings on the test instances. The default parameter values are in italics. The p-values are generated by comparing each parameter value to the default, and the ones with statistical significance (< 0.05) are highlighted in bold.

Parameter	Values	Mean	Std	P-Value
ρ (AS)	0.01	1831.06	661.47	**2.11e−04**
	0.05	1839.06	671.25	–
	0.1	1838.27	673.21	**6.22e−03**
Q (AS)	10	1808.77	656.42	**8.33e−18**
	100	1839.06	671.25	–
	200	1841.58	674.85	4.91e−01
T_{pts} (MMAS)	50	1883.69	698.39	**6.11e−03**
	100	1886.90	697.88	–
	200	1888.32	704.85	2.49e−01

A.4. Parameter setting for AS and MMAS

The default settings for α, β, δ, τ_{max} and τ_{min} are selected based on the original AS and MMAS models (Dorigo et al., 1996; Stützle and Hoos, 2000). Here we tune the parameters ρ, Q and T_{pts} using the 100 randomly generated instances (with $|V| = 100$). We test three different values for each parameter and select the value that generates the best solution quality as the default (see Table 5 for the results). In the case that there is no statistically significant difference between the parameter values, e.g., $Q = 100$ and $Q = 200$, we simply select the smaller value as the default. The stopping criterion, population size and global-best vs iteration-best for updating the pheromone matrix are tuned in Section 4.6.

A.5. Experimental results for the large benchmark problem instances

We further compare our SVM-MMAS-LS algorithm (with $T_{\text{ter}} = 200$) to the three state-of-the-art heuristics on a set of large benchmark problem instances (generation 3) with $|V| > 400$ (Fischetti et al., 1998), and the results are shown in Table 6. Note again that the results for EA4OP, GRASP-PR and 2P-IA are taken from Kobeaga et al. (2018). Further, there is a cutoff (18 000 s) imposed for the 2P-IA, GRASP-PR, and EA4OP algorithms, while our SVM-MMAS-LS algorithm is terminated only when the best solution found cannot be improved for $T_{\text{ter}} = 200$ consecutive iterations. Hence, the runtimes of the algorithms cannot be directly compared, and we will mainly focus on the comparison of the solution quality generated. Overall, the EA4OP algorithm performs the best on these instances, while our SVM-MMAS-LS algorithm generally finds better solutions than the GRASP-PR and 2P-IA algorithms. It is noteworthy that our SVM-MMAS-LS algorithm is still able to find better solutions than EA4OP for seven instances, even though our algorithm has not been fine-tuned for these large problem instances.

Appendix B. Adapting ML-ACO to solve the maximum weighted clique problem

As our ML-ACO algorithm is a generic approach, we apply it to solve another combinatorial optimization problem, the maximum weighted clique problem (MWCP). The MWCP is a variant of the maximum clique problem, which is a fundamental problem in graph theory with a wide range of real-world applications (Wu and Hao, 2015; Malladi et al., 2017; Letchford et al., 2020; Blum et al., 2021). Solving the MWCP is NP-hard, and a large number of solution methods have been developed for this problem recently. These include exact solvers such as branch-and-bound algorithms (Jiang et al., 2017, 2018; Li et al., 2018a; San Segundo et al., 2019) and local search methods (Wang et al., 2016; Cai and Lin, 2016; Zhou et al., 2017; Nogueira and Pinheiro, 2018; Wang et al., 2020).

Table 6
The comparison between our SVM-MMAS-LS algorithm and three state-of-the-art heuristics on the large benchmark problem instances. The column 'Overall best' denotes the best objective value found by the four algorithms. For each algorithm, the best objective value found, the gap (%) to the overall best solution and the average runtime (in second) are presented. The best gap is highlighted in bold. Note that the results for 2P-IA, GRASP-PR and EA4OP are taken from Kobeaga et al. (2018), and thus the runtimes are not comparable.

Instance	Overall	2P-IA			GRASP-PR			EA4OP			SVM-MMAS-LS		
	Best	Best	Gap	Time	Best	Gap	Time	Best	Gap	Time	Best	Gap	Time
fl417	14 186	12 792	9.83	12.49	13 709	3.36	73.99	14 186	**0.00**	12.45	14 010	1.24	11.95
gr431	10 817	10 735	0.76	17.18	10 500	2.93	103.88	10 817	**0.00**	54.50	10 727	0.83	32.15
pr439	15 097	13 006	13.85	15.35	14 694	2.67	153.19	15 097	**0.00**	10.96	14 952	0.96	28.27
pcb442	14 565	14 446	0.82	11.57	14 206	2.46	31.56	14 522	0.30	6.58	14 565	**0.00**	19.39
d493	24 981	21 458	14.10	15.15	23 362	6.48	197.43	24 981	**0.00**	19.18	24 505	1.91	33.56
att532	15 342	15 178	1.07	23.23	14 573	5.01	75.56	15 342	**0.00**	22.75	15 187	1.01	52.79
ali535	9 328	8 884	4.76	26.04	8 672	7.03	162.88	9 328	**0.00**	94.09	9 261	0.72	47.28
pa561	14 034	13 662	2.65	35.44	13 271	5.44	36.99	14 034	**0.00**	21.35	13 778	1.82	34.14
u574	19 691	19 368	1.64	34.05	18 747	4.79	44.60	19 691	**0.00**	19.77	19 216	2.41	46.40
rat575	19 879	19 669	1.06	33.87	19 007	4.39	47.82	19 879	**0.00**	18.03	19 251	3.16	38.61
p654	24 249	22 303	8.03	30.94	24 221	0.12	284.87	24 130	0.49	18.54	24 249	**0.00**	24.00
d657	23 792	22 401	5.85	32.94	21 893	7.98	69.62	23 772	0.08	21.89	23 792	**0.00**	78.17
gr666	16 902	15 561	7.93	46.77	15 545	8.03	227.61	16 902	**0.00**	143.87	16 387	3.05	93.76
u724	27 932	27 072	3.08	58.19	26 665	4.54	150.49	27 932	**0.00**	29.26	27 294	2.28	94.25
rat783	26 870	26 870	**0.00**	80.85	25 591	4.76	153.06	26 797	0.27	30.64	26 269	2.24	80.96
dsj1000	30 943	30 043	2.91	183.75	28 822	6.85	781.25	30 943	**0.00**	79.18	30 063	2.84	149.73
pr1002	38 762	37 244	3.92	115.38	35 808	7.62	485.46	38 762	**0.00**	47.30	37 354	3.63	184.91
u1060	36 570	35 649	2.52	179.48	34 873	4.64	689.68	36 570	**0.00**	75.88	35 816	2.06	204.28
vm1084	37 508	36 170	3.57	167.56	36 121	3.70	813.15	37 508	**0.00**	54.21	36 488	2.72	394.14
pcb1173	40 069	38 284	4.45	301.94	37 506	6.40	477.29	40 069	**0.00**	66.16	38 749	3.29	240.44
d1291	40 706	36 419	10.53	212.23	36 063	11.41	1 288.60	38 132	6.32	299.87	40 706	**0.00**	423.03
rl1304	41 214	37 562	8.86	362.55	37 859	8.14	1 000.74	41 214	**0.00**	81.11	40 261	2.31	647.91
rl1323	46 641	43 029	7.74	323.11	42 990	7.83	904.51	46 641	**0.00**	93.53	45 395	2.67	336.84
nrw1379	43 972	42 412	3.55	418.50	40 170	8.65	870.77	43 972	**0.00**	124.75	40 696	7.45	545.26
fl1400	57 226	57 131	0.17	471.99	55 269	3.42	7 075.68	57 226	**0.00**	599.81	57 140	0.15	1 046.48
u1432	46 657	45 806	1.82	305.25	45 084	3.37	1 291.16	46 657	**0.00**	138.02	45 507	2.46	551.09
fl1577	45 692	44 188	3.29	428.11	44 062	3.57	9 751.32	45 692	**0.00**	295.62	44 661	2.26	772.68
d1655	59 092	55 771	5.62	600.91	54 121	8.41	3 487.68	58 728	0.62	674.25	59 092	**0.00**	787.98
vm1748	70 958	67 785	4.47	1 280.00	68 976	2.79	6 251.95	70 958	**0.00**	225.29	69 209	2.46	1 676.85
u1817	63 639	60 751	4.54	738.77	59 783	6.06	4 171.11	63 639	**0.00**	1 302.35	61 813	2.87	1 382.51
rl1889	68 422	64 660	5.50	1 260.33	62 538	8.60	5 535.04	68 422	**0.00**	244.97	66 131	3.35	1 556.93
d2103	80 940	78 084	3.53	1 585.02	73 034	9.77	18 000.00	77 333	4.46	1 168.90	80 940	**0.00**	2 401.62
u2152	73 400	71 469	2.63	1 326.63	68 152	7.15	9 579.31	73 400	**0.00**	1 619.61	70 920	3.38	2 217.11
u2319	78 319	78 319	**0.00**	1 210.42	76 250	2.64	6 496.30	78 113	0.26	569.76	76 317	2.56	3 982.97
pr2392	84 094	79 704	5.22	1 496.23	78 364	6.81	8 624.02	84 094	**0.00**	422.73	80 697	4.04	5 311.21
pcb3038	104 667	100 660	3.83	4 491.30	97 596	6.76	18 000.00	104 667	**0.00**	917.39	102 194	2.36	9 188.42
fl3795	99 121	95 675	3.48	6 867.61	–	NA	18 000.00	97 707	1.43	3 158.89	99 121	**0.00**	16 848.43
fnl4461	164 201	158 654	3.38	11 047.56	–	NA	18 000.00	164 201	**0.00**	3 248.64	137 559	16.23	27 916.33
rl5915	199 336	189 096	5.14	15 139.79	–	NA	18 000.00	199 336	**0.00**	5 593.23	193 173	3.09	63 131.86
rl5934	207 385	198 428	4.32	16 384.22	–	NA	18 000.00	207 385	**0.00**	5 881.87	200 213	3.46	51 098.55
pla7397	320 744	303 425	5.40	18 000.00	–	NA	18 000.00	320 744	**0.00**	18 000.00	310 795	3.10	163 213.50
Average	59 949	57 312	4.53	2 082.26	38 280	5.63	4 814.36	59 744	**0.35**	1 109.93	57 913	2.45	8 705.53
# best	–	–	2	–	–	0	–	–	**32**	–	–	7	–

Table 7
The best objective values generated by our ML-ACO algorithm and four state-of-the-arts for solving the MWCP. The best results are highlighted in bold.

Graph	$\|V\|$	ML-ACO	FastWClq	LSCC	WLMC	TSM
p_hat1000-1	1000	1 514	1 514	1 514	1 514	1 514
p_hat1000-2	1000	5 777	5 777	5 777	5 777	5 777
p_hat1000-3	1000	**8 111**	8 058	**8 111**	8 076	**8 111**
p_hat1500-1	1500	1 619	1 619	1 619	1 619	1 619
p_hat1500-2	1500	**7 360**	7 327	**7 360**	**7 360**	**7 360**
p_hat1500-3	1500	**10 321**	10 057	**10 321**	9 846	10 119
DSJC1000.5	1000	2 186	2 186	2 186	2 186	2 186
san1000	1000	1 716	1 716	1 716	1 716	1 716
C1000.9	1000	9 191	8 685	**9 254**	7 317	7 341
C2000.5	2000	**2 466**	**2 466**	**2 466**	2 360	2 407
C2000.9	2000	10 888	9 943	**10 964**	7 738	8 228
C4000.5	4000	2 776	2 645	**2 792**	2 383	2 402
MANN_a45	1035	34 209	34 111	34 243	**34 265**	34 259
hamming10-2	1024	50 512	50 512	50 512	50 512	50 512
hamming10-4	1024	5 127	4 982	**5 129**	4 738	4 812
Average	–	10 252	10 107	10 264	9 827	9 891

Here, the aim of our ML model is to predict the 'probability' of a vertex being part of the maximum weighted clique. To train our ML model, we construct a training set using eighteen optimally-solved small graphs ($|V| < 1000$) from the standard DIMACS library. The original DIMACS graphs are unweighted, and thus we assign a weight $w_i = (i \mod 200) + 1$ to the vertex v_i ($i = 1, \ldots, |V|$), following the previous works (Wang et al., 2016; Cai and Lin, 2016; Jiang et al., 2017, 2018). Each training instance corresponds to a vertex in a training graph. We extract six features to characterize a vertex, including graph density, vertex weight, vertex degree, an upper bound and two statistical features described in Sun et al. (2021b). A training instance is labeled as 1 if the corresponding vertex belongs to the maximum weighted clique; otherwise it is labeled as −1. We then train a linear SVM to classify whether a vertex belongs to the maximum weighted clique or not.

We use fifteen larger graphs ($|V| \geq 1000$) from the DIMACS library as our test problem instances. For each problem instance, we use the trained ML model to predict a probability value p_i for each vertex $v_i \in V$. The predicted values (p_i) are then incorporated into the MMAS algorithm to guide its sampling towards larger-weighted cliques. More specifically, we use p_i to set the heuristic weight: $\eta_i = p_i \cdot w_i$, for each $v_i \in V$. The parameter settings for MMAS and linear SVM are the same as before. We compare our ML-ACO algorithm against two exact solvers — TSM (Jiang et al., 2018) and WLMC (Jiang et al., 2017) as well as

two heuristic methods LSCC (Wang et al., 2016) and FastWClq (Cai and Lin, 2016) for solving the MWCP. The cutoff time for each algorithm is set to 1000 s.

The best objective values obtained by each algorithm in 25 independent runs are presented in Table 7. We can observe that our ML-ACO algorithm is comparable to the state-of-the-art algorithms for solving the MWCP. On average, the best objective values found by our ML-ACO algorithm are significantly better than those found by FastWClq, WLMC and TSM. The LSCC algorithm performs the best and generates slightly better results than our ML-ACO algorithm. These results are interesting, because generic solution methods such as our ML-ACO algorithm are not often expected to be as competitive as specialized solvers.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: https://www.tensorflow.org/ software available from tensorflow.org.

Abbasi, B., Babaei, T., Hosseinifard, Z., Smith-Miles, K., Dehghani, M., 2020. Predicting solutions of large-scale optimization problems via machine learning: A case study in blood supply chain management. Comput. Oper. Res. 119, 104941.

Angelelli, E., Archetti, C., Filippi, C., Vindigni, M., 2017. The probabilistic orienteering problem. Comput. Oper. Res. 81, 269–281.

Archetti, C., Corberán, Á., Plana, I., Sanchis, J.M., Speranza, M.G., 2016. A branch-and-cut algorithm for the orienteering arc routing problem. Comput. Oper. Res. 66, 95–104.

Assunção, L., Mateus, G.R., 2021. Coupling feasibility pump and large neighborhood search to solve the steiner team orienteering problem. Comput. Oper. Res. 128, 105175.

Bengio, Y., Lodi, A., Prouvost, A., 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. European J. Oper. Res. 290 (2), 405–421.

Bishop, C.M., 2006. Pattern Recognition and Machine Learning. Springer.

Blum, C., 2005. Ant colony optimization: Introduction and recent trends. Phys. Life Rev. 2 (4), 353–373.

Blum, C., Djukanovic, M., Santini, A., Jiang, H., Li, C.-M., Manyà, F., Raidl, G.R., 2021. Solving longest common subsequence problems via a transformation to the maximum clique problem. Comput. Oper. Res. 125, 105089.

Boser, B.E., Guyon, I.M., Vapnik, V.N., 1992. A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory. pp. 144–152.

Cai, S., Lin, J., 2016. Fast solving maximum weight clique problem in massive graphs. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. In: IJCAI'16, AAAI Press, pp. 568–574.

Campos, V., Martí, R., Sánchez-Oro, J., Duarte, A., 2014. GRASP with path relinking for the orienteering problem. J. Oper. Res. Soc. 65 (12), 1800–1813.

Chang, C.-C., Lin, C.-J., 2011. LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2, 27:1–27:27.

Chao, I.-M., Golden, B.L., Wasil, E.A., 1996. A fast and effective heuristic for the orienteering problem. European J. Oper. Res. 88 (3), 475–489.

Chen, T., Li, M., Yao, X., 2018. On the effects of seeding strategies: a case for search-based multi-objective service composition. In: Proceedings of the Genetic and Evolutionary Computation Conference. In: GECCO '18, Association for Computing Machinery, pp. 1419–1426.

Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20 (3), 273–297.

Ding, J., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., Song, L., 2020. Accelerating primal solution findings for mixed integer programs based on solution prediction. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, Vol. 34. pp. 1452–1459.

Dorigo, M., Blum, C., 2005. Ant colony optimization theory: A survey. Theor. Comput. Sci. 344 (2-3), 243–278.

Dorigo, M., Gambardella, L.M., 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. 1 (1), 53–66.

Dorigo, M., Maniezzo, V., Colorni, A., 1996. Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. B 26 (1), 29–41.

El-Hajj, R., Dang, D.-C., Moukrim, A., 2016. Solving the team orienteering problem with cutting planes. Comput. Oper. Res. 74, 21–30.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J., 2008. LIBLINEAR: A library for large linear classification. J. Mach. Learn. Res. 9 (Aug), 1871–1874.

Fischetti, M., Fraccaro, M., 2019. Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks. Comput. Oper. Res. 106, 289–297.

Fischetti, M., Gonzalez, J.J.S., Toth, P., 1998. Solving the orienteering problem through branch-and-cut. INFORMS J. Comput. 10 (2), 133–148.

Friedrich, T., Wagner, M., 2015. Seeding the initial population of multi-objective evolutionary algorithms: A computational study. Appl. Soft Comput. 33, 223–230.

Gambardella, L.M., Montemanni, R., Weyland, D., 2012. Coupling ant colony systems with strong local searches. European J. Oper. Res. 220 (3), 831–843.

Golden, B.L., Levy, L., Vohra, R., 1987. The orienteering problem. Nav. Res. Logist. 34 (3), 307–318.

Gunawan, A., Lau, H.C., Vansteenwegen, P., 2016. Orienteering problem: A survey of recent variants, solution approaches and applications. European J. Oper. Res. 255 (2), 315–332.

Hammami, F., Rekik, M., Coelho, L.C., 2020. A hybrid adaptive large neighborhood search heuristic for the team orienteering problem. Comput. Oper. Res. 123, 105034.

Hopper, E., Turton, B.C.H., 2001. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European J. Oper. Res. 128 (1), 34–57.

Jia, Y.-H., Mei, Y., Zhang, M., 2021. A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem. IEEE Trans. Cybern. 1–14. http://dx.doi.org/10.1109/TCYB.2021.3069942.

Jiang, H., Li, C.-M., Liu, Y., Manya, F., 2018. A two-stage MaxSAT reasoning approach for the maximum weight clique problem. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32. pp. 1338–1346.

Jiang, H., Li, C.-M., Manya, F., 2017. An exact algorithm for the maximum weight clique problem in large graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 31. pp. 830–838.

Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A.M., Talbi, E.-G., 2022. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. European J. Oper. Res. 296 (2), 393–422.

Ke, L., Archetti, C., Feng, Z., 2008. Ants can solve the team orienteering problem. Comput. Ind. Eng. 54 (3), 648–665.

Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations. pp. 1–14.

Kobeaga, G., Merino, M., Lozano, J.A., 2018. An efficient evolutionary algorithm for the orienteering problem. Comput. Oper. Res. 90, 42–59.

Lauri, J., Dutta, S., 2019. Fine-grained search space classification for hard enumeration variants of subset problems. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence. pp. 2314–2321.

Letchford, A.N., Rossi, F., Smriglio, S., 2020. The stable set problem: Clique and nodal inequalities revisited. Comput. Oper. Res. 123, 105024.

Li, Z., Chen, Q., Koltun, V., 2018b. Combinatorial optimization with graph convolutional networks and guided tree search. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. In: NIPS'18, Curran Associates Inc., Red Hook, NY, USA, pp. 537–546.

Li, C.-M., Liu, Y., Jiang, H., Manyà, F., Li, Y., 2018a. A new upper bound for the maximum weight clique problem. European J. Oper. Res. 270 (1), 66–77.

Liaw, C., 2000. A hybrid genetic algorithm for the open shop scheduling problem. European J. Oper. Res. 124 (1), 28–42.

Lin, S., 1965. Computer solutions of the traveling salesman problem. Bell Syst. Tech. J. 44 (10), 2245–2269.

Malladi, K.T., Mitrovic-Minic, S., Punnen, A.P., 2017. Clustered maximum weight clique problem: Algorithms and empirical analysis. Comput. Oper. Res. 85, 113–128.

Mavrovouniotis, M., Müller, F.M., Yang, S., 2016. Ant colony optimization with local search for dynamic traveling salesman problems. IEEE Trans. Cybern. 47 (7), 1743–1756.

Montemanni, R., Weyland, D., Gambardella, L., 2011. An enhanced ant colony system for the team orienteering problem with time windows. In: 2011 International Symposium on Computer Science and Society. IEEE, pp. 381–384.

Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning. pp. 807–814.

Nogueira, B., Pinheiro, R.G., 2018. A CPU-GPU local search heuristic for the maximum weight clique problem on massive graphs. Comput. Oper. Res. 90, 232–248.

Palma-Heredia, D., Verdaguer, M., Molinos-Senante, M., Poch, M., Cugueró-Escofet, M., 2021. Optimised blending for anaerobic co-digestion using ant colony approach: Besòs river basin case study. Renew. Energy 168, 141–150.

San Segundo, P., Furini, F., Artieda, J., 2019. A new branch-and-bound algorithm for the maximum weighted clique problem. Comput. Oper. Res. 110, 18–33.

Santini, A., 2019. An adaptive large neighbourhood search algorithm for the orienteering problem. Exp. Syst. Appl. 123, 154–167.

Santini, A., Viana, A., Klimentova, X., Pedroso, J.P., 2021. The probabilistic travelling salesman problem with crowdsourcing. pp. 1–31, Preprint. URL: http://www.optimization-online.org/DB_HTML/2021/08/8563.html.

Shen, Y., Sun, Y., Eberhard, A., Li, X., 2021. Learning primal heuristics for mixed integer programs. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. http://dx.doi.org/10.1109/IJCNN52387.2021.9533651.

Silberholz, J., Golden, B., 2010. The effective application of a new approach to the generalized orienteering problem. J. Heuristics 16 (3), 393–415.

Stützle, T., Hoos, H.H., 2000. MAX–min ant system. Future Gener. Comput. Syst. 16 (8), 889–914.

Sun, Y., Ernst, A., Li, X., Weiner, J., 2021a. Generalization of machine learning for problem reduction: a case study on travelling salesman problems. OR Spectr. 43 (3), 607–633.

Sun, Y., Li, X., Ernst, A., 2021b. Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems. IEEE Trans. Pattern Anal. Mach. Intell. 43 (5), 1746–1760.

Vansteenwegen, P., Souffriau, W., Van Oudheusden, D., 2011. The orienteering problem: A survey. European J. Oper. Res. 209 (1), 1–10.

Verbeeck, C., Sörensen, K., Aghezzaf, E.-H., Vansteenwegen, P., 2014. A fast solution method for the time-dependent orienteering problem. European J. Oper. Res. 236 (2), 419–432.

Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.-H., 2017. The time-dependent orienteering problem with time windows: a fast ant colony system. Ann. Oper. Res. 254 (1-2), 481–505.

Wang, Y., Cai, S., Chen, J., Yin, M., 2020. SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem. Artif. Intell. 280, 103230.

Wang, Y., Cai, S., Yin, M., 2016. Two efficient local search algorithms for maximum weight clique problem. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30. pp. 805–811.

Wang, S., Li, M., Zhang, Y., Bao, Z., Tedjopurnomo, D.A., Qin, X., 2018. Trip planning by an integrated search paradigm. In: Proceedings of the 2018 International Conference on Management of Data. In: SIGMOD '18, Association for Computing Machinery, New York, NY, USA, pp. 1673–1676.

Wu, Q., Hao, J.-K., 2015. A review on algorithms for maximum clique problems. European J. Oper. Res. 242 (3), 693–709.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S., 2021. A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. 32 (1), 4–24.

Xiang, X., Tian, Y., Zhang, X., Xiao, J., Jin, Y., 2021. A pairwise proximity learning-based ant colony algorithm for dynamic vehicle routing problems. IEEE Trans. Intell. Transp. Syst. 1–12.

Zhou, Y., Hao, J.-K., Goëffon, A., 2017. PUSH: A generalized operator for the maximum vertex weight clique problem. European J. Oper. Res. 257 (1), 41–54.

Zlochin, M., Birattari, M., Meuleau, N., Dorigo, M., 2004. Model-based search for combinatorial optimization: A critical survey. Ann. Oper. Res. 131 (1-4), 373–395.