Heuristic Search for the Orienteering Problem with Time-Varying Reward

Chao Cao[1], Jinyun Xu[1], Ji Zhang[1], Howie Choset[1], Zhongqiang Ren[2]

[1] Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA
[2] Shanghai Jiao Tong University, 800 Dongchuan Road, Minhang District, Shanghai, 200240 China
{ccao1, jinyunx, zhangji, choset}@andrew.cmu.edu, zhongqiang.ren@sjtu.edu.cn

Abstract

The Orienteering Problem (OP) seeks a path on a graph to maximize total rewards collected subject to a path length budget. Typically, a reward is achieved by visiting a vertex in the graph, and such a reward is constant for all time. This paper considers a variant of OP where the reward of each vertex is an arbitrary time-dependent function, and hence the name time-varying reward OP (TR-OP). To solve this problem, we develop a novel heuristic search algorithm called Reward Maximization A* (RMA*), which is guaranteed to find an optimal solution to TR-OP. We also develop a fast method to compute an admissible heuristic for RMA* that can effectively direct the search to save computational effort. Furthermore, we introduce a hyper-parameter in RMA* that trades off between solution quality and runtime efficiency for RMA*. We benchmark RMA* against a recent dynamic programming (DP) approach, which runs fast in practice, but has no guarantee of the solution optimality. In our tests, RMA* reduces the runtime by up to 70% compared to DP. By adjusting the hyper-parameter, RMA* is able to find solutions with up to 30% more rewards than those found by DP.

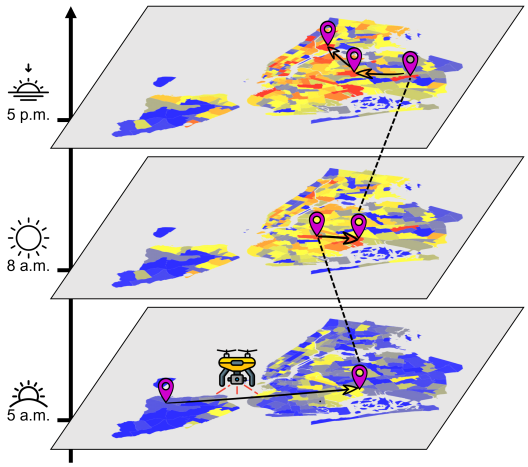

Figure 1: A motivating example of TR-OP, where a UAV plans a route to maximize the information collected in a traffic monitoring task in a large city.

Introduction

In a graph where each vertex has an associated reward value, expressed as a real number, the Orienteering Problem (OP) (Tsiligirides 1984) seeks to find a path for an agent that maximizes the total rewards collected along the path, while adhering to constraints on path length or travel time. This paper considers a generalized version of OP, namely Time-varying Reward OP (TR-OP), where the reward at each vertex is a known function that changes over time. TR-OP arises in applications such as logistics (Aringhieri et al. 2022), surveillance (Yu, Schwager, and Rus 2016), and transportation system (Martins et al. 2021). Consider an example (Fig. 1) where unmanned aerial vehicles (UAVs) are tasked to monitor traffic in a large city (Yu, Schwager, and Rus 2016). These robots have limited flying time due to the battery capacity. Meanwhile, traffic events, such as congestion, are time-varying, and the goal is to plan a best tour for the UAV so that the maximum amount of information can be collected per flight. Similar scenarios arise when autonomous marine vehicles are deployed to collect samples to monitor water pollution events such as oil spills subject to limited travel and time budgets.

OP is an NP-hard problem (Golden, Levy, and Vohra 1987) and so is TR-OP. The challenge in TR-OP lies in determining which subset of vertices to visit within the time budget, the order of these visits, and the timing of arrival, i.e., arrival time, at each vertex. TR-OP has been previously addressed by a dynamic programming (DP) approach (Ma et al. 2017), where a state space defined over both vertices and time steps is first constructed, and dynamic programming is then applied over this state space to greedily maximize the reward collected along a path from the starting vertex to any other vertex-time pair. This DP approach is not guaranteed to return an optimal solution but has been shown to run fast and provide near-optimal solution in practice.

This paper develops a new heuristic search approach called Reward Maximization A* (RMA*). RMA* solves a reward maximization problem and is guaranteed to find an optimal solution to TR-OP, i.e., a path with the maximum reward subject to the path length constraint. We also develop a fast method as part of RMA* to compute an admissible

heuristic that can effectively direct the search to save computational effort. Furthermore, we introduce a hyper-parameter in RMA* that tunes the searching behavior of RMA*, trading off solution quality for runtime efficiency. We show that the existing DP approach (Ma et al. 2017) is a special case of RMA* by fixing this hyper-parameter to 1.

In comparison with this DP approach (Ma et al. 2017), RMA* gains runtime efficiency due to the following reasons. First, it avoids the explicit construction of the state space over vertices and times, which is often a huge space and can be time-consuming to explicitly construct. Second, the search process of RMA* is guided by a heuristic, which allows RMA* to explore only a fraction of the state space that leads to a solution. Third, RMA* introduces a dominance pruning rule to eliminate partial solutions that cannot lead to an optimal solution during the search. We compare RMA* against the DP approach using both a public dataset on New York taxi routing and a synthetic dataset with randomly generated graphs. Our experimental results show that RMA* can reduce the runtime by up to 70% compared to the DP approach. By adjusting the hyper-parameter, RMA* is able to find solutions with up to 30% more rewards than those found by the DP approach.

Related Work

The Orienteering Problem (OP), initially introduced by (Tsiligirides 1984) has been extensively studied over the past few decades. Fundamentally, OP involves finding a single route to maximize total rewards by visiting a subset of vertices on a graph, adhering to a path-length limit. Practically, OP has wide applications ranging from planning optimal sightseeing tours in tourism (Gavalas et al. 2015) to enhancing delivery routes in supply chain logistics (Aringhieri et al. 2022).

Conventionally, research on OP and its variations has focused on a static model, where travel costs and reward collection are time-independent, as highlighted in several surveys (Vansteenwegen, Souffriau, and Van Oudheusden 2011; Gunawan, Lau, and Vansteenwegen 2016; Vansteenwegen and Gunawan 2019). Recently, some time-dependent variants of OP have been explored, particularly those accounting for variable travel costs between locations, mirroring fluctuating real-world traffic scenarios (Li et al. 2010; Gunawan, Yuan, and Lau 2014). In these studies, the travel cost between two locations typically depends on the time when the trip starts.

Studies incorporating time-varying rewards are relatively rare. These typically assume specific time-dependent reward variations at each location. Erkut and Zhang pioneered an OP variant featuring linearly decreasing vertex rewards, using a branch-and-bound algorithm and a greedy heuristic (Erkut and Zhang 1996). This concept was later expanded to include multiple agents, employing a clustering-based heuristics algorithm (Ekici and Retharekar 2013) and evolutionary local search methods (Afsar and Labadie 2013). Tang et al. introduced a piecewise-linear relationship between vertex reward and agent arrival time, and approached the problem using tabu search (Tang, Miller-Hooks, and Tomastik 2007). In a different vein, Erdogan and Laporte, along with Pietz and Royset, explored scenarios where rewards increase during an agent's stay at a location, i.e., service time (Pietz and Royset 2013; Yu et al. 2019). Yu et al. further extended this to rewards both decrease over arrival time and increase over service time (Yu et al. 2022).

The study most closely related to ours is the one conducted by Ma et al, where they consider arbitrary time-dependent rewards (Ma et al. 2017). They discretize time into a finite number of time intervals, and construct a spatial-temporal graph that encapsulates all possible transitions between locations over different time intervals. Given the unidirectional nature of time, this graph forms a directed acyclic graph (DAG). They then employ a dynamic programming approach to identify the path that accumulates the most rewards, essentially finding the "longest" path within the DAG. Empirical results show that this DP approach can produce near-optimal solutions with relatively short computation time. In the rest of this paper, we reference this method as DP for ease of discussion.

Finally, the proposed RMA* algorithm is related to multi-objective heuristic search (Hernández et al. 2023; Ren et al. 2022b) and heuristic search for maximization problems (Stern et al. 2014; Cohen, Stern, and Felner 2020), which is discussed after RMA* is presented.

Problem Statement

Let $G = (V, E)$ denote a finite directed graph, where each vertex $v \in V$ represents a static target location in the workspace to be visited by the agent, and each edge $e \in E$ denote the transition from one vertex to another. There is a global clock, and time is discretized into steps, i.e., $t \in T = \{0, 1, 2, \cdots, t_{max}\}$, where t_{max} is a finite time budget for the agent to move.

For each $t \in T$ and each vertex $v \in V$, there is a corresponding real number $r(v, t) \in \mathbb{R}$ (either positive or negative), which represents the time-varying reward that is collected by the agent if the agent visits v at time t. Each edge $(u, v) \in E$ is associated with a constant positive real number $d(u, v) \in \mathbb{R}^+$, which represents the amount of time (i.e., duration) for the agent to travel from u to v.[1] The agent is allowed to wait at a vertex $v \in V$ for an arbitrary number of time steps $\{t_1, t_1 + 1, t_1 + 2, \cdots, t_2\}$, where the agent collects all rewards associated with those time steps $\{t_1, t_1 + 1, t_1 + 2, \cdots, t_2\}$. Once the agent leaves a vertex v, the agent is not allowed to visit v again in its future path.[2]

The agent starts to move from its starting vertex $v_o \in V$ at time $t = 0$. Let $\pi(v_\ell, t_\ell) = \{(v_o, 0), (v_1, t_1), (v_2, t_2), \cdots, (v_\ell, t_\ell)\}$ denote a path of the agent from its depot v_o through vertices $v_1, v_2, \cdots, v_\ell$, while reaching these vertices at times $t_1, t_2 \cdots, t_\ell$ respec-

[1] If $d(u, v) = 0$, then u, v are at the same location in the workspace and u, v can be combined as a single vertex w with $r(w, t) = r(u, t) + r(v, t)$. We therefore only consider the case where $d(u, v) > 0$.

[2] We follow the convention in the literature (Vansteenwegen, Souffriau, and Van Oudheusden 2011; Gunawan, Lau, and Vansteenwegen 2016; Vansteenwegen and Gunawan 2019) where the agent can visit each vertex at most once.

tively. We refer to $\pi(v_\ell, t_\ell)$ simply as π when there is no ambiguity. Let $r(\pi)$ denote the sum of rewards collected by the agent along the path π, i.e., $r(\pi) := \sum_{k=1,2,\dots,\ell} r(v_k, t_k)$. Let $V_d \subseteq V$ denote a set of destination vertices where the agent is allowed to end its path. When $V_d = V$, it means the agent can stop at any vertex when $t = t_{max}$.

Definition 1 (TR-OP)**.** *The Time-Varying Rewards Orienteering Problem (TR-OP) seeks to find a path $\pi(v_\ell, t_\ell)$ that starts from v_o at time $t = 0$ and ends at a vertex $v_\ell \in V_d$ at some time $t_\ell \leq t_{max}$, while maximizing the total reward $r(\pi(v_\ell, t_\ell))$ collected by the agent along the path.*

Method

Concepts and Notations

Let $l = (v, t, g, A)$ denote a *label* that is a tuple of a vertex $v \in V$, a real number g, a time t and a set of vertices $A \subseteq V$. Intuitively, each label l identifies a path of the agent that reaches v at time t with accumulated reward g (i.e., reward-to-come) while visiting the set of vertices A along the path. To simplify the presentation, let $v(l), g(l), t(l), A(l)$ denote the respective component of label l. The components $t(l), g(l), A(l)$ together capture the history information of a path that reaches vertex $v(l)$, which are necessary for the search. In other words, two labels l_1, l_2 that reach the same vertex $v = v(l_1) = v(l_2)$ are different, if any one of their t, g, A components is different.

Let $h(l)$ denote the heuristic value (h-value) of label l. Here, $h(l)$ is an estimate of the reward that the agent can possibly collect in the future (i.e., reward-to-go) by further extending the path represented by l. A heuristic is said to be *admissible* if it is an upper bound of the true maximum reward that the agent can possibly collect. Let $f(l) := g(l) + h(l)$ denote the f-value of label l, which is an estimate of the total reward that the agent can collect from the start to the goal via label l. Let OPEN denote a priority queue, where labels l are prioritized based on their keys, which are defined as tuples $(f(l), t(l))$ in lexicographic order as follows. For any two labels l_1 and l_2, the label with a larger f-value gets higher priority and is popped from OPEN earlier during the search. If $f(l_1) = f(l_2)$, then $t(l_1)$ and $t(l_2)$ are considered, and the label with a smaller t gets higher priority and is popped from OPEN earlier.

To compare labels, we introduce the following notion of label dominance.

Definition 2 (Label Dominance)**.** *Given two labels l_1, l_2 with the same vertex (i.e., $v(l_1) = v(l_2)$) and the same time (i.e., $t(l_1) = t(l_2)$), we say l_1 dominates l_2, if (i) $g(l_1) \geq g(l_2)$ and (ii) $A(l_1) \subseteq A(l_2)$.*

If l_1 dominates l_2, then we also say l_2 is dominated by l_1. If l_1 does not dominate l_2 and l_2 does not dominate l_1, then l_1, l_2 are *non-dominated* by each other. Labels are only comparable if they have the same vertex and the same time. During the search, there can be multiple non-dominated labels that reach the same v at the same time t, which need to be tracked when searching for an optimal solution. Let $\mathcal{F}(v, t), v \in V, t \in T$ denote the *front* set at vertex v at time t, which is a set of labels, where each label is non-dominated by any other label in this set.

Algorithm 1: Pseudocode for RMA*

```
1: l_o ← (v = v_o, g = 0, t = 0, A = ∅), parent(l_o) ← null
2: h(l_o) ← GetHeuValue(l_o), f(l_o) ← g(l_o) + h(l_o)
3: Add l_o to OPEN
4: F(v, t) ← ∅, ∀v ∈ V, ∀t ∈ T
5: l_d ← null
6: while OPEN ≠ ∅ do
7:     extract l from OPEN
8:     if IsPruned(l) then
9:         continue
10:    FilterAndAddFront(l)
11:    if (l_d = null or g(l) > g(l_d)) and (v(l) ∈ V_d) then
12:        l_d ← l
13:    U ← GetSuccessors(l)
14:    for v' ∈ U do
15:        if v' = v(l) then                          ▷ Wait in place
16:            t' ← t(l) + 1, g' ← g(l) + r(v', t'), A' ← A(l)
17:        else                                       ▷ Move to another vertex
18:            t' ← t(l) + d(v(l), v'), g' ← g(l) + r(v', t'),
       A' ← A(l) ∪ {v'}
19:        l' ← (v', g', t', A')
20:        h(l') ← GetHeuValue(l'), f(l') ← g(l') + h(l')
21:        if IsPruned(l') then
22:            continue
23:        else
24:            parent(l') ← l
25:            add l' to OPEN
26: return Reconstruct(l_d)
```

Algorithm Description

RMA* (Alg. 1) is a search algorithm that is guided by an admissible heuristic. To initialize the search, RMA* first creates an initial label $l_o \leftarrow (v = v_o, g = 0, t = 0, A = \emptyset)$, computes its f-value and adds l_o to OPEN. Furthermore, $\mathcal{F}(v, t), \forall v \in V, t \in T$ is initialized to be an empty set, and l_d is initialized to be an empty pointer, which will be used to store a label with the maximum g-value at any time during the search. In other words, l_d represents an incumbent solution path with the largest g-value during the search.

In each search iteration (Line 6-25), a label with the highest priority is first extracted from OPEN for processing. Then, procedure *IsPruned* (Alg. 3) checks if l should be discarded based on (i) whether $f(l)$ is smaller than the incumbent solution $g(l_d)$, (ii) whether l can still reach a destination vertex within the time budget, or (iii) whether l is dominated by any existing labels that have reached $v(l)$. If either condition among (i) (ii) or (iii) is true, then l cannot lead to an optimal solution and is thus pruned. The current search iteration ends. If l is not pruned, l is then added to $\mathcal{F}(v(l), t(l))$ in procedure *FilterAndAddFront*, where any existing label in $\mathcal{F}(v(l), t(l))$ is removed if it is dominated by l. Note that *FilterAndAddFront* differs from *IsPruned* as *FilterAndAddFront* seeks to remove the existing labels in $\mathcal{F}(v, t)$ and *IsPruned* seeks to remove l.

If l reached a destination vertex in V_d (Line 11), RMA* checks if the incumbent solution should be updated. If l has higher reward than the previous incumbent solution or there is no incumbent solution yet (i.e., $l_d = null$), l then becomes the incumbent solution and is assigned to be l_d.

Algorithm 2: Pseudocode for IsPruned(l)

```
1: if f(l) ≤ g(l_d) then
2:     return true
3: if t_max − t(l) > min_{u∈V_d} d(v(l), u) then
4:     return true
5: for l' ∈ F(v(l), t(l)) do
6:     if A(l') ⊆ A(l) and g(l') ≥ g(l) then
7:         return true
8: return false
```

Algorithm 3: Pseudocode for FilterAndAddFront(l)

```
1: for l' ∈ F(v(l), t(l)) do
2:     if A(l) ⊆ A(l') and g(l) ≥ g(l') then
3:         remove l' from F(v(l), t(l))
4: add l to F(v(l), t(l))
```

Afterwards, label l is expanded as follows. A set of successor labels of l is created. These successors are created by either extending the path represented by l to all adjacent vertices of $v(l)$ in G, or let the agent wait at $v(l)$ for one time unit. For each of these successors l', the corresponding $h(l')$ and $f(l')$ are computed (as elaborated in the Heuristic Computation section). Then, l' is checked for pruning using *IsPruned*, which is the same procedure as aforementioned. Finally, l' is added to OPEN if it is not pruned.

At the end of the search, all labels l' in OPEN are pruned due to the incumbent solution l_d (since $f(l) \leq g(l_d)$) and OPEN depletes. Then RMA* terminates and the path represented by l_d is reconstructed by iteratively following the parent points within procedure Reconstruct. The resulting path is guaranteed to have the maximum accumulative reward. If l_d is $null$, Reconstruct returns an empty path which means the given problem instance is unsolvable.

Heuristic Computation

We introduce a pre-processing procedure, which is conducted before Alg. 1 in order to pre-compute a table H that can be looked up when computing the heuristic value for a label l. The key of this table H is $(v, t), \forall v \in V, \forall t \in T$, and the value is $\max_{\tau=t+1,t+2,\cdots,t_{max}} r(v, \tau)$, the maximum reward at vertex v for any future time step greater than t. Denote $H(v, t)$ to be the value in this table.

For any label l, we compute its heuristic as follows.

$$h(l) := (t_{max} - t(l)) \max_{u \notin A(l)} H(u, t(l)) \tag{1}$$

This heuristic is an upper bound of the maximum reward-to-go in the future and thus admissible.

Lemma 1 (Admissible Heuristic)**.** *The heuristic in Equation (1) is admissible.*

To compute the table H, for each vertex, we only need to iterate backwards from t_{max} down to 0 in order to find $\max_{\tau=t+1,t+2,\cdots,t_{max}} r(v, \tau)$ via dynamic programming, since $H(v, t) = \max\{H(v, t+1), r(v, t+1)\}$. The entire table takes $O(|V||T|)$ time to compute. During the search, to evaluate Equation (1) for a label l, it takes at most $O(|V|)$ time to find the maximum $H(u, t)$ for any $u \notin A(l)$.

Remark 1. *The computation of $h(l)$ for any label l in Equation (1) depends on the time $t(l)$ and the history, i.e., the vertices that are visited by l, which is stored in $A(l)$. Here, we avoid defining the consistency of a heuristic, which is not required by RMA*.*

Truncated Frontier Set

RMA* can be modified to balance between the runtime efficiency and solution quality. The $\mathcal{F}(v, t)$ for any $v \in V, t \in T$ can be a large set, since any two labels l, l' can represent paths with different subset of vertices $A(l), A(l')$ visited, and neither of $A(l), A(l')$ is a subset of the other. As a result, many labels are non-dominated during the search and need to be expanded. Since the goal of the problem is to maximize the reward, we therefore propose using a truncated frontier set $\mathcal{F}_K(v, t)$ to replace $\mathcal{F}(v, t)$, which stores the top K labels that have the largest g-values during the search and ignores the set of vertices visited by the label. This variant of RMA* does not guarantee returning an optimal solution at termination, however, in practice, the returned solution is often near optimal.

Remark 2. *When using truncated frontier set $\mathcal{F}_K(v, t)$ with $K = 1$, RMA* becomes a greedy approach that always stores the label with the best g-value during the search and there is only one label in each $\mathcal{F}_K(v, t)$. This greedy approach RMA* ($K = 1$) produces equivalent results to the existing dynamic programming approach in DP. However, RMA* ($K = 1$) is more computationally efficient as it tends to explore a smaller state space compared to DP, due to the heuristic guided search. We compare the differences between different RMA* variants and DP in the Experiments section.*

Discussion

This section elaborates on a few design choices behind Alg. 1 and its relationship to other heuristic search techniques. First, the existing research in longest simple path problem (Cohen, Stern, and Felner 2020) suggested that depth-first methods (such as depth-first branch-and-bound) often run fast for maximization problems (Stern et al. 2014). Here, we choose to first investigate a best-first search method, since it is hard to obtain an accurate heuristic for estimating the reward-to-go, and an inaccurate heuristic can lead the depth-first search to inefficiently explore many branches.

Second, the check for pruning on Line 8 is needed for the following reason. The frontier set $\mathcal{F}(v, t)$ stores only expanded labels, and resembles the "closed" set in A*. The check on Line 21 does not compare a newly generated label against other existing labels in OPEN. A label l in OPEN may be dominated by another label l' generated and added to OPEN later. In this case, l' will be selected first from OPEN for expansion. When l is then selected from OPEN, it will be pruned by Line 8 and thus will not be expanded. This design choice is similar to the idea of "lazy check" in bi-objective A* (Hernández et al. 2023).

Third, the label dominance in Def. 2 only compares labels at the same time $t(l_1) = t(l_2)$ and can not be modified to $t(l_1) \leq t(l_2)$, since the reward $r(v, t)$ can be negative,

and arriving earlier is not always better. When assuming the reward is always positive, the dominance can consider $t(l_1) \leq t(l_2)$. When the time-varying reward has a piecewise pattern (Tang, Miller-Hooks, and Tomastik 2007), the notion of safe-interval can be potentially used (Phillips and Likhachev 2011) in combination with dominance (Ren et al. 2022a) to expedite the search.

Finally, the widely adopted relaxation of traveling salesman problems such as minimum spanning trees can not be readily adapted here to provide heuristics, since it is hard to estimate which subset of vertices will be visited within the time limit. We therefore use a simple yet fast-running heuristic to guide the search.

Analysis

Intuitively, RMA* enumerates all possible paths (represented by labels) from v_o to any other vertex in an intelligent way by using the heuristic to guide the search, while attempting to prune the labels that can not lead to an optimal solution. At termination, all reachable labels are either pruned or kept as the incumbent solution l_d, which is an optimal solution.

Lemma 2. *For an unsolvable instance, RMA* terminates in finite time and returns failure.*

Proof. The graph G is finite and the time budget t_{max} is finite. Therefore, there is a finite number of possible paths from v_o to any other vertex within the time budget, and there is a finite number of labels to be generated during the search. For any label l, RMA* never generates another label l' that is the same as l ($l' = l$), since such a l' will be pruned due to IsPruned(l'). Therefore, for unsolvable instances, RMA* terminates in finite time after generating and expanding a finite number of labels, and returns failure. □

Lemma 3. *Labels that are pruned due to the IsPruned procedure cannot lead to either a feasible solution or an optimal solution if one exists.*

Proof. IsPruned(l') only removes successor labels l' that (i) cannot reach any goal vertex within the time budget (Line 3 in Alg. 3) or (ii) leads to a path with less reward than an existing path (Line 1 and Line 6 in Alg. 3). Here, for Line 1 in Alg. 3, the correctness of the pruning relies on the admissibility of the heuristic (Lemma 1). For case (i), the successor label l' cannot lead to a feasible solution path for the problem and can thus be pruned. For case (ii), an existing better path (than the path represented by l') is already found and the successor label l' can be pruned. □

Theorem 1 (Completeness)**.** *RMA* terminates in finite time by either returning a solution path if the instance is solvable, or returning failure if the instance is unsolvable.*

Proof. With Lemma 2, RMA* terminates in finite time and returns failure if the instance is unsolvable. If the instance is solvable, then RMA* considers all possible labels that may lead to a solution and keeps the labels with higher rewards (Lemma 3). There is only a finite number of possible labels to be generated during the search. RMA* terminates in finite time and returns a feasible solution. □

Theorem 2 (Solution Optimality)**.** *For a solvable instance, the path returned by RMA* is an optimal solution.*

Proof. For a solvable instance, with Lemma 3, the search considers all possible labels and prunes labels that cannot lead to an optimal solution. RMA* always selects a label l with the maximum f-value in OPEN for expansion, and stores the label with the maximum reward during the search as the incumbent solution l_d. When the search terminates, OPEN is empty and all labels in OPEN are discarded due to IsPruned, and l_d is an optimal solution. □

Experiments

We conducted numerical evaluations to validate the computational efficiency and effectiveness of RMA*. In these tests, our algorithm was benchmarked against the DP approach in (Ma et al. 2017). As pointed out in (Ma et al. 2017), the DP approach has significant superiority over classic OP algorithms, notably the "center-of-gravity" heuristic (Golden, Levy, and Vohra 1987) when dealing with time-varying rewards. For this reason, our comparison focused solely on DP. Both RMA* and DP were implemented in C++ for a fair comparison. The experiments were conducted on a laptop with an Intel i7 3.6Ghz processor and 32 GB of RAM.

To verify the computational efficiency and solution quality of RMA*, we employed two types of experiments. The first utilized a public dataset on city-like road networks, specifically recorded New York Taxi Data, to test the algorithm in a practical, real-life scenario. The second experiment involved a simulated environment, using a randomly generated graph where vertices are assigned with randomly generated time-varying rewards. Both experiments allowed us to evaluate RMA* under both real and controlled conditions, demonstrating its computational efficiency and effectiveness in solving TR-OP.

Taxi Routing in New York City

Similarly to (Ma et al. 2017), this experiment leverages the taxi trip records of for-hire vehicles in New York City, covering the period from January to June 2023 (NYC Taxi and Limousine Commission 2023).[3] This dataset segments New York City into 260 distinct taxi zones, as illustrated in Figure 2. In this experiment, we focus solely on the pick-up locations of all trips, following the conventions in (Ma et al. 2017).

We address the taxi routing challenge as a TR-OP. In our model, each taxi zone within the city is represented as a vertex in a graph, defined by its geographical coordinates. To determine travel costs between these taxi zones, we calculate the geodesic distance between their centroids. This distance is then used to estimate travel time between zones, assuming an average taxi speed of 15 kilometers per hour in New York City (New York City Department of Transportation 2023).

[3]Only seven months' data of 2023 was available during our experiments. The dataset is available at: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

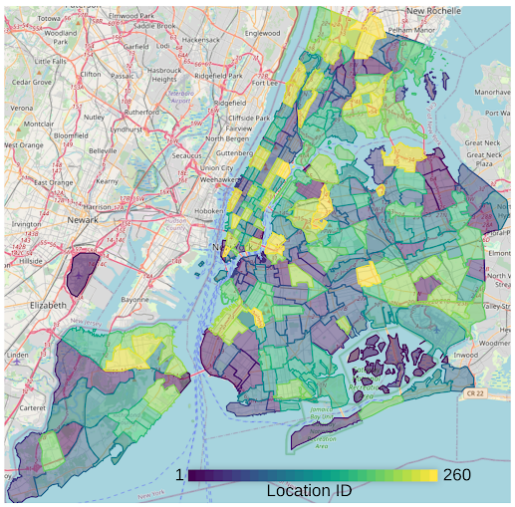

Figure 2: NYC taxi zones overlaid on a geographic map. Colors denote the zone IDs from 1 to 260.

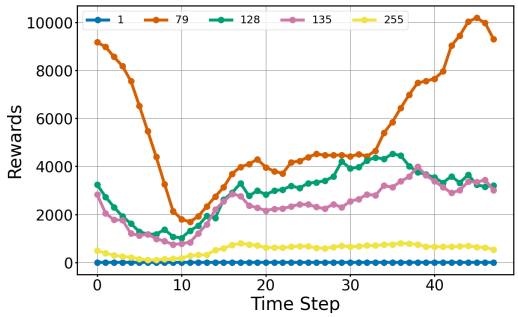

Figure 3: Passenger pick-up call volume (rewards) in June 2023 at key locations: Newark Airport (1), East Village, Manhattan (79), Inwood Hill Park, Manhattan (128), Kew Gardens, Queens (135), and Williamsburg (North Side), Brooklyn (255).

To effectively manage time in our model, we segment each day into 30-minute intervals, resulting in 48 distinct time periods per day. The potential reward from a taxi zone during a specific interval is measured by the volume of taxi calls recorded in that period. We assume that higher taxi call volumes indicate increased opportunities for revenue, making call volume an appropriate metric for rewards in our optimization process.

Figure 3 shows the average taxi call volumes throughout the day for June 2023, focusing on five key locations: Newark Airport, Inwood Hill Park, East Village, Kew Gardens Hills, and Williamsburg. This figure illustrates the dynamic and fluctuating nature of demand, and how the demand shifts over time and across different locations. Understanding these shifts in demand is crucial for the optimization of taxi routing decisions, as it allows for better alignment of taxi availability with passenger needs.

We evaluate the computational runtime of our method against the DP approach, using the New York taxi data.

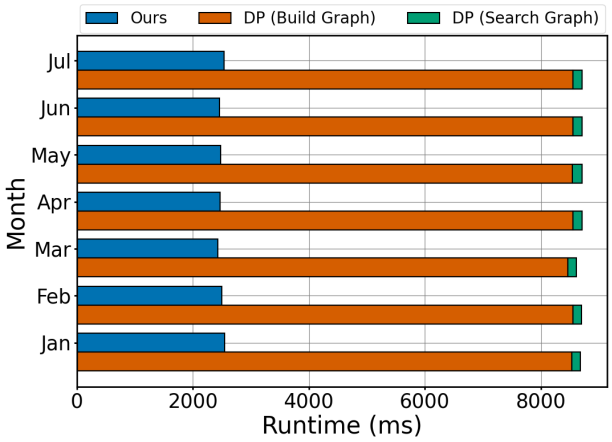

Figure 4: Average computational runtime for both our method and the DP method on the New York Taxi Dataset. The DP method involves two steps: graph building and searching. Notably, the graph-building step demands a significantly longer processing time.

Specifically, we adjust the Truncated Frontier Set size to 1 ($K = 1$), as detailed in the Truncated Frontier Set section, ensuring that our method yields solutions equivalent to those of DP. Both methods were tested using seven months of data, with the starting taxi zone varying from 1 to 260. We measure the computational runtime in two dimensions: physical time and the number of iterations required. Here, we define an iteration in our method as the expansion of a node within the state space (i.e., when RMA* reaches Line 13 in Alg. 1), and in the DP approach, as the number of times where the parent pointer of a vertex in the DAG is updated.

The result shows that our method outperforms the DP approach significantly in terms of computation speed, being 71.44% faster as illustrated in Figure 4. Despite the fast searching speed of DP, the construction of an explicit spatial-temporal graph demands a considerable amount of time. As highlighted in Table 1, while both methods collect the same rewards with similar iteration counts, our method consistently demonstrates shorter computational runtime compared to DP. Note that the iteration counts of DP solely depends on the number of graph vertices and time intervals. DP has the same iteration counts for all months except for March. This is due to the absence of data from one location, the Newark Airport (location 1), which results in 259 graph vertices instead of 260 as in other months. An exemplary route for June 2023 is visualized in Figure 5. In the visualization, the time dimension is represented perpendicularly to the 2D plane, and the taxi call volumes are color-coded across different time intervals.

Reward Maximization on Random Graphs

We evaluated the effectiveness of our method using a dataset consisting of randomly generated graphs with time-varying rewards. Each instance in the dataset contains a fully connected graph with 50 vertices. These vertices are randomly

Month	Average Reward	Runtime (ms)		Iteration	
		Our	DP	Our	DP
Jan	388961.98	2549.35 ± 21.05	8687.60 ± 11.24	12145.82 ± 3.19	12221.00 ± 0.00
Feb	367748.68	2493.13 ± 18.45	8707.62 ± 23.84	12104.38 ± 5.07	12221.00 ± 0.00
Mar	434418.21	2432.12 ± 19.00	8618.12 ± 8.58	12040.83 ± 10.07	12174.00 ± 0.00
Apr	418235.92	2465.41 ± 21.17	8708.82 ± 20.15	12094.35 ± 10.04	12221.00 ± 0.00
May	466301.60	2479.23 ± 20.82	8710.00 ± 8.44	12088.82 ± 11.89	12221.00 ± 0.00
Jun	434847.82	2456.12 ± 21.99	8708.34 ± 19.95	12084.17 ± 9.48	12221.00 ± 0.00
Jul	413033.79	2534.25 ± 20.54	8709.01 ± 12.22	12137.50 ± 7.30	12221.00 ± 0.00

Table 1: Comparison of computational runtime and iteration count between our method and the DP method on the New York Taxi Dataset. Both methods achieve the same rewards in all runs. Our method exhibits lower computational runtime compared to DP, albeit having a similar iteration count.

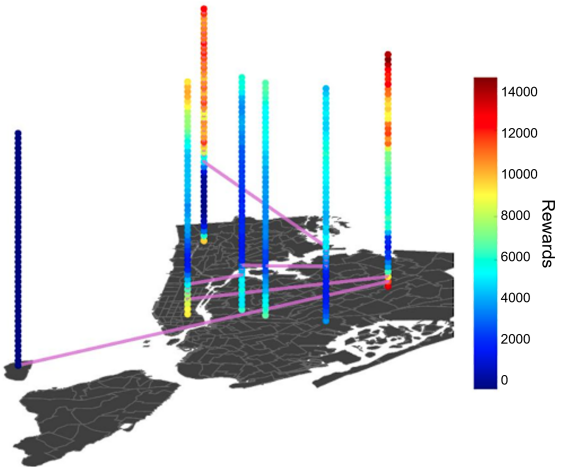

Figure 5: Sample results of running our method on the New York City taxi dataset: all taxi zones are mapped onto the XY plane, with time intervals extending along the Z axis. The computed path, starting at location 1 and finishing at location 136, is shown as the purple line. Additionally, rewards associated with locations along the path are visualized and color-coded.

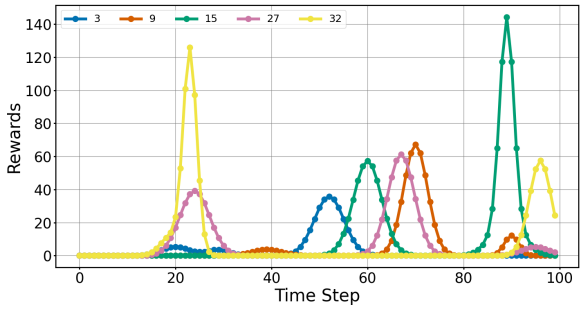

Figure 6: The time-varying rewards of five vertices in a randomly generated graph.

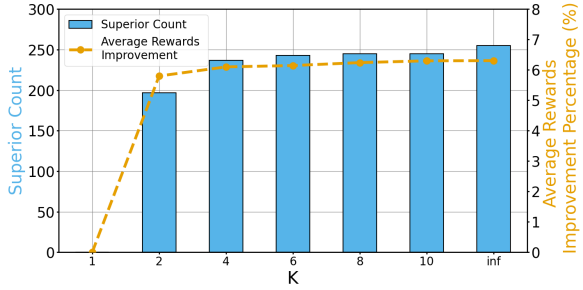

Figure 7: Superior count and reward improvement percentage increase as K increases from 1 to ∞. Superior count refers to the number of instances where our method collects more rewards than DP.

positioned on a two-dimensional plane, confined within a defined area that spans from the coordinates $[0, 0]$ to $[100, 100]$. The rewards for each vertex span over 100 time intervals. Values of the rewards were determined by composing three Gaussian distributions, with randomly generated mean ranging from [1, 100] and variance ranging from [1, 10]. Each Gaussian distribution is also randomly scaled (in the range of $[1, 500]$) to diversify the level of rewards across time intervals and vertices. Figure 6 shows the randomly generated rewards of a few vertices over 100 time steps.

We compared our method against DP across 1000 instances. We vary the hyper-parameter K in our method in the set of $\{1, 2, 4, 6, 8, 10, \infty\}$ to demonstrate the effect of the hyper-parameter on both the solution quality and computational runtime. Note that when $K = \infty$, our method produces optimal solutions. Figure 7 shows the effect of different K values on the solution quality. In the figure, as the K increases, the number of instances where our method finds solutions with higher rewards than DP, namely, superior count, increases. In addition, the average improvement percentage over the DP method increases as well, as shown by the yellow dotted line. In particular, when $K = \infty$, our method collects 30% more rewards on average in 255 out of 1000 instances. Figure 8 presents the number of cases where our method, with $K = \infty$, achieves higher rewards, and the average improvement percentage in these cases. For example, Figure 8 shows that in 25 cases, our method achieves 5% higher rewards compared to the baseline method.

Method		Runtime (ms)	Iteration
Baseline	DP	267.42 ± 2.23	2501.09 ± 434.66
With Heuristic	RMA* ($K = 1$)	102.53 ± 31.95	2584.52 ± 683.82
	RMA* ($K = 2$)	180.03 ± 59.76	4671.72 ± 1273.08
	RMA* ($K = 4$)	311.64 ± 115.66	8020.12 ± 2335.19
	RMA* ($K = 6$)	422.50 ± 171.94	10671.63 ± 3362.29
	RMA* ($K = 8$)	520.68 ± 228.31	12866.93 ± 4287.84
	RMA* ($K = 10$)	603.94 ± 283.55	14703.86 ± 5160.77
	RMA* ($K = \infty$)	5600.69 ± 12400.22	35908.61 ± 27269.25
Without Heuristic	RMA* ($K = 1$)	118.38 ± 35.86	2995.02 ± 728.36
	RMA* ($K = 2$)	213.51 ± 66.83	5543.98 ± 1366.60
	RMA* ($K = 4$)	385.36 ± 135.06	9844.73 ± 2510.77
	RMA* ($K = 6$)	542.17 ± 206.23	13489.66 ± 3557.30
	RMA* ($K = 8$)	684.99 ± 280.62	16653.62 ± 4627.16
	RMA* ($K = 10$)	821.48 ± 359.95	19435.54 ± 5718.59
	RMA* ($K = \infty$)	46586.03 ± 286675.07	73901.12 ± 74615.48

Table 2: Comparison of computational runtime between the DP method and our method using different frontier set sizes (K).

Node Number	Superior Count	Improvement
10	8	12.7123%
30	15	6.1760%
50	25	6.3043%
70	33	7.9023%
100	40	5.2333%

Table 3: Experiments on random graphs with different node numbers. As the node number increases, our method finds higher rewards in more instances.

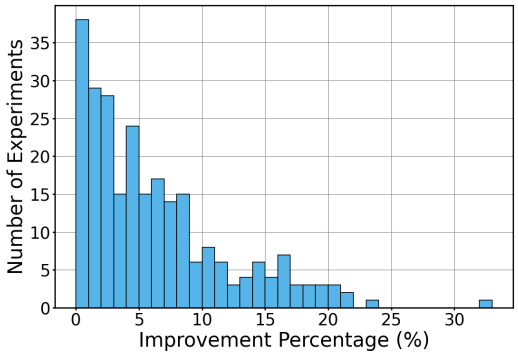

Figure 8: Percentage of solution quality improvement v.s. number of instances.

Table 2 presents the average runtime and iteration counts for the DP method and our method with different K values. Additionally, we report the runtime and iteration counts of our method running without the heuristic. In this scenario, the heuristic value $h(l)$ of all state-space nodes, which estimates the potential rewards that can be collected in the remaining time, is set to a large constant. As shown in Table 2, the absence of heuristic guidance leads to a significantly higher expansion of nodes compared to when a heuristic is used, which highlights the efficacy of our proposed heuristic.

We also evaluate our method against DP on random graphs with different node numbers, ranging from 10 to 100 nodes. The results are presented in Table 3. We run 100 trials for each method for each graph size and calculate the superior counts and reward improvement. As shown in the table, as the node number increases, there are more instances where the path computed by our method collects higher rewards.

Conclusion and Future Work

This paper presents a method to solve the Orienteering Problem with Time-varying rewards (TR-OP). The method uses a heuristic-guided search to explore the state space, which can produce an optimal solution to TR-OP. In addition, a hyper-parameter is introduced in the method to balance the trade-off between solution quality and computational runtime. We provide completeness and optimality proof of our method. In experiments, we evaluate our method against a dynamic programming (DP) approach using two datasets: a real-world New York Taxi Dataset and a synthetic dataset featuring randomly generated graphs. Our method reduces the runtime by 70% compared to DP. By adjusting the hyper-parameter, our method achieves up to 30% higher rewards than DP solutions.

We plan to extend the method to a multi-agent scenario in the future, where multiple agents are set to collect the maximum rewards on a graph, subject to each agent's travel budget. One can also develop new heuristics, and investigate fast dominance checking techniques, as in multi-objective search (Ren et al. 2022b; Hernández et al. 2023), for TR-OP to speed up the search.

References

Afsar, H. M.; and Labadie, N. 2013. Team orienteering problem with decreasing profits. *Electronic Notes in Discrete Mathematics*, 41: 285–293.

Aringhieri, R.; Bigharaz, S.; Duma, D.; and Guastalla, A. 2022. Novel applications of the team orienteering problem

in health care logistics. In *Optimization in Artificial Intelligence and Data Sciences: ODS, Rome, Italy, September 14-17, 2021*, 235–245. Springer.

Cohen, Y.; Stern, R.; and Felner, A. 2020. Solving the longest simple path problem with heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 75–79.

Ekici, A.; and Retharekar, A. 2013. Multiple agents maximum collection problem with time dependent rewards. *Computers & Industrial Engineering*, 64(4): 1009–1018.

Erkut, E.; and Zhang, J. 1996. The maximum collection problem with time-dependent rewards. *Naval Research Logistics (NRL)*, 43(5): 749–763.

Gavalas, D.; Konstantopoulos, C.; Mastakas, K.; Pantziou, G.; and Vathis, N. 2015. Heuristics for the time dependent team orienteering problem: Application to tourist route planning. *Computers & Operations Research*, 62: 36–50.

Golden, B. L.; Levy, L.; and Vohra, R. 1987. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3): 307–318.

Gunawan, A.; Lau, H. C.; and Vansteenwegen, P. 2016. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2): 315–332.

Gunawan, A.; Yuan, Z.; and Lau, H. C. 2014. A mathematical model and metaheuristics for time dependent orienteering problem. PATAT.

Hernández, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; Koenig, S.; and Salzman, O. 2023. Simple and efficient bi-objective search algorithms via fast dominance checks. *Artificial Intelligence*, 314: 103807.

Li, J.; Wu, Q.; Li, X.; and Zhu, D. 2010. Study on the time-dependent orienteering problem. In *2010 International Conference on E-Product E-Service and E-Entertainment*, 1–4. IEEE.

Ma, Z.; Yin, K.; Liu, L.; and Sukhatme, G. S. 2017. A spatio-temporal representation for the orienteering problem with time-varying profits. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6785–6792. IEEE.

Martins, L. d. C.; Tordecilla, R. D.; Castaneda, J.; Juan, A. A.; and Faulin, J. 2021. Electric vehicle routing, arc routing, and team orienteering problems in sustainable transportation. *Energies*, 14(16): 5131.

New York City Department of Transportation. 2023. New York City Mobility Report. https://www.nyc.gov/html/dot/html/about/mobilityreport.shtml. Accessed: 2023-11-15.

NYC Taxi and Limousine Commission. 2023. TLC Trip Record Data. https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page. Accessed: 2023-11-15.

Phillips, M.; and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE international conference on robotics and automation*, 5628–5635. IEEE.

Pietz, J.; and Royset, J. O. 2013. Generalized orienteering problem with resource dependent rewards. *Naval Research Logistics (NRL)*, 60(4): 294–312.

Ren, Z.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022a. Multi-Objective Safe-Interval Path Planning With Dynamic Obstacles. *IEEE Robotics and Automation Letters*, 7(3): 8154–8161.

Ren, Z.; Zhan, R.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022b. Enhanced multi-objective A* using balanced binary search trees. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 162–170.

Stern, R.; Kiesel, S.; Puzis, R.; Felner, A.; and Ruml, W. 2014. Max is more than min: Solving maximization problems with heuristic search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 5, 148–156.

Tang, H.; Miller-Hooks, E.; and Tomastik, R. 2007. Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E: Logistics and Transportation Review*, 43(5): 591–609.

Tsiligirides, T. 1984. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35: 797–809.

Vansteenwegen, P.; and Gunawan, A. 2019. Orienteering problems. *EURO Advanced Tutorials on Operational Research*.

Vansteenwegen, P.; Souffriau, W.; and Van Oudheusden, D. 2011. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1): 1–10.

Yu, J.; Schwager, M.; and Rus, D. 2016. Correlated orienteering problem and its application to persistent monitoring tasks. *IEEE Transactions on Robotics*, 32(5): 1106–1118.

Yu, Q.; Adulyasak, Y.; Rousseau, L.-M.; Zhu, N.; and Ma, S. 2022. Team orienteering with time-varying profit. *INFORMS Journal on Computing*, 34(1): 262–280.

Yu, Q.; Fang, K.; Zhu, N.; and Ma, S. 2019. A matheuristic approach to the orienteering problem with service time dependent profits. *European Journal of Operational Research*, 273(2): 488–503.