Aerial Vehicle Routing and Scheduling for UAS Traffic Management: A Hybrid Monte Carlo Tree Search Approach

Kenny Chour
Metis Technology Solutions
NASA Ames Research Center
Moffett Field, CA 94035, USA
kenny.chour@nasa.gov

Priyank Pradeep
USRA
NASA Ames Research Center
Moffett Field, CA 94035, USA
priyank.pradeep@nasa.gov

Alexey A. Munishkin
Aviation Systems Division
NASA Ames Research Center
Moffett Field, CA 94035, USA
alexey.a.munishkin@nasa.gov

Krishna M. Kalyanam
Aviation Systems Division
NASA Ames Research Center
Moffett Field, CA 94035, USA
krishna.m.kalyanam@nasa.gov

***Abstract*—We present the Multi-Route Weighted Package Delivery Problem (MRWPDP) and a scalable solution methodology as a major step towards enabling an airspace deconfliction service for drone delivery operations. The problem is motivated by *Strategic deconfliction* under the FAA's "Unmanned Aircraft Systems Traffic Management" Concept of Operations. MRWPDP falls under a class of vehicle routing and scheduling problems, and as such is NP-Hard. In MRWPDP, a graph network is given which consists of depots, drop-off sites, and multiple routes connecting the two. In addition, routes are weighted by the associated ground risk and total travel distance for package delivery. The goal is to optimally schedule the departure time and assign routes to a known set of vehicles at the depot. We propose a heuristic solution to the problem by borrowing techniques from Mixed Integer Linear Programming (MILP), Constraint Programming, and Monte Carlo Tree Search (MCTS). The resulting hybrid framework is MCTS with Bound-and-Prune (BP) and rapid simulated updates (U), or MCTS-BP-U. This approach is able to quickly provide a feasible solution for MRWPDP, even for large problem instances up to 1000 vehicles. We provide a MILP formulation of MRWPDP and compare its performance against MCTS-BP-U in terms of solution quality. An agent-based model simulation is conducted as a final step to validate the efficacy of our approach.**

***Index Terms*—Monte Carlo Tree Search, Mixed Integer Programming, Constraint Programming, Combinatorial Optimization, Scheduling and Routing**

NOMENCLATURE

C	Set of customers.
D	Set of depots.
E	Set of edges or routes.
O	Set of delivery orders.
O_d	Set of delivery orders at depot $d \in D$.
$\Omega(e)$	Set of edges that intersect with $e \in E$.
E_d	Set of edges connected to depot $d \in D$.
$a(e, i)$	Duration needed to arrive to an intersection (e, i), where $e, i \in E$.
t_i	Receipt time of delivery order i.
x_i	Scheduled departure time of delivery order i.
$y_{i,e}$	Indicator variable for edge e, 1 if $i \in O$ is assigned to it; 0 otherwise.
α	Non-negative importance weighting for risk.
β	Non-negative importance weighting for distance.
r_e	Risk value of edge $e \in E$.
c_e	Distance length of edge $e \in E$.
$b_s(e)$	Vehicle-to-Vehicle temporal separation along an edge $e \in E$.
$b_p(d)$	Package preparation lead time, $d \in D$.
$b_f(e, i)$	Temporal separation required at an intersection (e, i), where $e, i \in E$.
$b_t(d)$	Temporal separation during departure from a depot, $d \in D$.
M	Large non-negative value ("big-M"), i.e. $M \gg 0$.

I. INTRODUCTION

In recent years, rapid advances in unmanned aircraft systems (UASs) have led to ubiquitous civilian applications including crop monitoring, filming, inspection, and parcel delivery [1], [2]. Several commercial entities (e.g., Joby, Wisk) are already working on electric air taxis for passenger transport [3]. This explosive rise in UAS technology brings many potential benefits, but also introduces many challenges related to safe integration into the airspace. The current Air Traffic Control paradigm requires significant human resources and is likely not scalable to UAS operations. The UAS Traffic Management (UTM) ecosystem, first developed by NASA and later written into a formal Concept of Operations (ConOps) by the FAA serves as a promising first step towards addressing the aforementioned issues [4].

Under UTM, three layers of airspace conflict management are envisioned [5]: 1) Strategic deconfliction involves pre-flight scheduling and routing in order to minimize the likelihood of airborne conflicts later during flight. 2) Tactical deconfliction aims to maintain aircraft separation during flight especially with techniques such as speed adjustment. Lastly, 3) collision avoidance, i.e., detect and avoid or sense and avoid methods, are used to remove imminent conflicts. This work focuses on strategic deconfliction presented as a Multi-Route Weighted Package Delivery Problem (MRWPDP), which models an automated package delivery system involving multiple UAS operators. In MRWPDP, UASs are to depart from various depot nodes and travel along an assigned route to reach a customer drop-off node. A key challenge is maintaining separation distance between every pair of UAS, especially since different routes may intersect with each other. Thus, the goal is to optimally schedule departure times and also assign routes to each of the UAS such that they maintain separation at all times. Central to our problem is the use of Operational Volume Blocks (OVBs), which discretize pre-determined routes into consecutive polygonal volumes. By ensuring only 1 UAS occupies an OVB at a given time, spacing constraint values are automatically determined [6]. Belonging to the class of vehicle routing and scheduling problems, MRWPDP is unfortunately NP-Hard [7]. Thus, fast heuristics are required to scale the problem to real-world scenarios.

In this work, we present a heuristic based on the popular *Monte Carlo Tree Search* (MCTS) method to quickly provide feasible solutions to the MRWPDP. MCTS has been used in a variety of applications such as games [8], [9] aviation, [10], and combinatorial optimization problems [11]–[20]. A salient feature of MCTS is that it is able to look through a large problem space by using statistical information about each decision. As a result, MCTS carefully balances exploitation and exploration to avoid exhaustively searching through the entire problem space. To make use of MCTS, MRWPDP is formulated as a Markov Decision Process (MDP), which allows us to use Constraint Propagation techniques from Constraint Programming (CP) to find feasible solutions. We also incorporate lower bounding information from solving a Mixed Integer Linear Programming (MILP) relaxation of MRWPDP. The lower bound information allows us to effectively bound and prune branches of the search tree. Applying strategies from both CP and Integer Programming (IP) to MCTS, allows us quickly find solutions.

This work contributes to existing research in several ways. First, we provide a MILP formulation of the MRWPDP. Then, we provide a hybrid Monte Carlo Tree Search (MCTS) framework that neatly integrates Branch-and-Bound and Constraint Propagation techniques from MILP and CP, respectively. We provide computational results showing the efficacy of this framework as the problem size increases, and finally validate the results in an agent-based model simulation. The remainder of this paper is organized as follows. Some background is given to set the context of this work in Section II. An overview of the related works is presented in Section III. A formal problem definition and a MILP formulation are given in Section IV. A hybrid Monte Carlo Tree Search framework to solve the problem is presented in Section V, followed by computational results and discussion in Section VI.

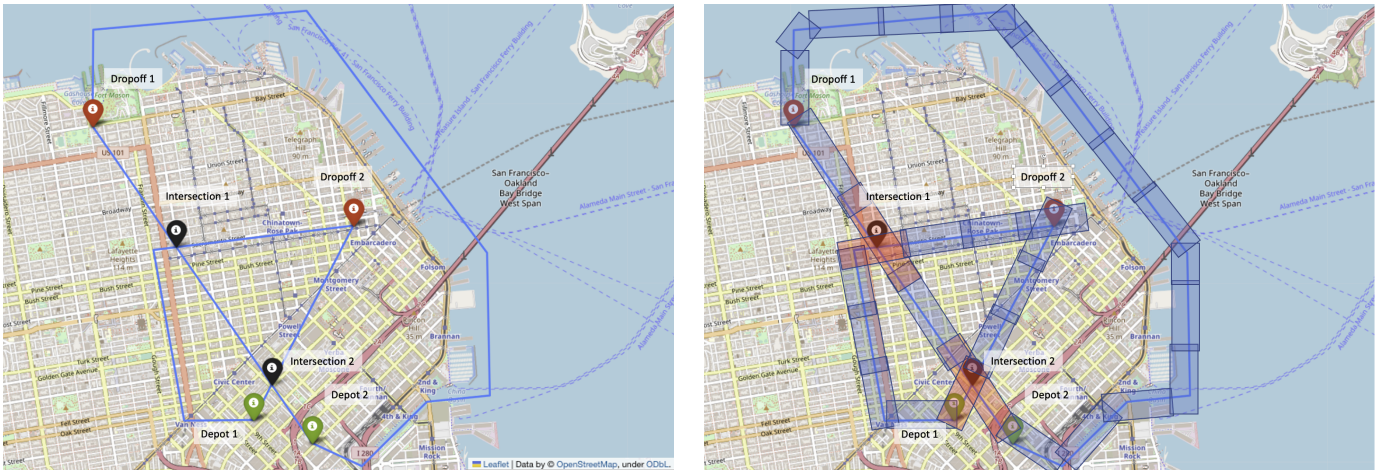
II. Background

In a Markov Decision Process (MDP), decisions are carried out in sequence and can be described by a 6-tuple (S, A, P, R, s_o, s_g) [21]: let S be the set of states, A the set of actions or *moves*, $P : S \times A \times S \to \mathbf{R}$ the transition model, $R : S \times A \times S \to \mathbf{R}$ the reward function, and $s_o, s_g \in S$ the initial and goal state, respectively. A policy π maps each state $s \in S$ to an action $a \in A$. A value function $V^{\pi}(s)$ returns the expected reward of a state $s \in S$ by following a policy π. In MDP problems, we seek an optimal policy π^* by finding the optimal value function $V^{\pi^*}(s)$. However, finding $V^{\pi^*}(s)$ is often difficult due to most problems having large state and action spaces. The optimal value function can be approximated along with a policy using strategies such as MCTS.

MCTS estimates the value function by performing *rollouts*, i.e. creating forward trajectories from a given state until a terminal condition (such as a goal state) is reached. The collection of trajectories help build a statistical notion of possible future rewards. More specifically, MCTS constructs a search tree where each node and action is synonymous with a state $s \in S$ and action $a \in A$, respectively. During each iteration of the algorithm, 4 steps are taken beginning with the root node: 1) in *Selection*, the child node s' with the best reward is selected recursively until the frontier (no more actions or children) is reached, 2) in *Expansion* new child nodes are created, 3) in *Simulation*, the reward of s' is estimated via rollouts, and 4) in *Backpropagation*, the reward and other statistical information is recursively returned to all direct ancestor nodes.

III. Related Prior Work

Over the years, there has been considerable research on similar aircraft deconfliction problems. In [23], a MINLP (nonlinear programming) is solved to deconflict up to 6 vehicles flying directly towards a common meeting point. Due to scalability issues, the authors develop a clustering-based heuristic, but only at most 10 vehicles were deconflicted. In [24], the authors solved an en-route planning problem for air traffic control that involves scheduling aircraft departures and arrivals. Though similar to our problem, the authors also incorporate fuel costs in their objective and deploy a custom heuristic that is not asymptotically optimal like ours. In [25], the authors also focus on strategic deconfliction by providing a MILP formulation. However, they deconflict aircrafts using both flight level selection and ground delays. Additionally, the authors propose solving the MILP in a rolling window fashion as a scalable heuristic. Other works have used MCTS in similar, but different contexts. In [26], MCTS is applied to create a guidance algorithm for on-demand, free-flight operations for a single aircraft. The authors later generalized their work to multiple aircrafts [27]. In [28], MCTS is used



(a) Scenario given by the downtown San Francisco area.

(b) Operational Volume Blocks overlaid on scenario.

Fig. 1: OpenStreetMap [22] view of the downtown San Francisco area used to generate problem instances. (Left): The depots, customers, and intersections are green (bottom two), red (top two), and black (middle two) pins, respectively. The solid blue lines are available routes to take. (Right): Operational volume blocks are shown overlaid on the left figure. As part of ConOps, the blocks at the intersections should never be active at the same time (red).

to resolve conflicts during flight, after pre-flight planning has already occurred. Our work however is primarily focused on using MCTS to solve strategic deconfliction. The closest related work can be found in [29]. The authors pose the strategic deconfliction problem as a MDP and develop a custom algorithm called fastMDP to solve. They show that FastMDP is competitive to MCTS and superior in some cases. However, their problem involves planning in free-flight (absence of tubes or corridors) and it is not clear how many flight plans are being generated simultaneously. Additionally, the FastMDP algorithm is only applicable for very specific problem setups, so generalization may be difficult.

Our work differs from existing research in the following ways: 1) To the best of our knowledge, there is little prior work that has attempted to incorporate both CP and MILP knowledge into MCTS to solve the strategic deconfliction problem as we have, 2) few works have shown their heuristics methods to be as scalable as MCTS, and 3) many strategic deconfliction works are done in free-flight instead utilizing pre-defined routes.

IV. Problem Definition and Formulation

We are interested in solving a multi-depot multi-customer site package delivery drones scheduling problem that maximizes efficiency and also minimizes both conflict in the air (between vehicles) and risk to people on the ground. In MRWPDP, a graph network is given, which consists of several depots (source) and customer (sink) nodes. Between the two node types, multiple routes connect the depots to customer nodes, i.e., hyperedges are allowed. We will use the term route and edge interchangeably. The motivation behind hyperedges is to allow for edges with varying degrees of risk and distance values. For example, a shorter and more risky route mimicks flying over residential areas. In addition, routes are weighted by both ground risk and *distance to customer site* values. The goal is to determine the earliest *conflict free* departure time for a known set of package-carrying drones at the depots, while also choosing the best route (w.r.t. risk, distance, and time) to reach their assigned customer site. We assume fixed ground speeds for the vehicles and *conflict free* refers to a complete path from depot to destination that satisfies all inter-vehicle spacing constraints. Note that fixed ground speed implies that we can only control the take-off time and there is no opportunity to change the flight thereafter. This assumption vastly simplifies the optimization problem (which is NP-hard already). We can always introduce additional waypoints/nodes where the vehicle can hover/hold, however, this comes at the additional (computational) cost of dealing with bigger graphs. We deal with two types of spacing constraints: 1) between adjacent vehicles on the same edge (or route/path) and 2) between two *consecutive* vehicles on two different paths that intersect. By consecutive, we imply that the two vehicles arrive at a common waypoint (intersection) one immediately after the other (in any order).

Figure 1a shows an example scenario of MRWPDP. A 2D map consisting of 2 depots (green) and 2 customers (red) are shown, along with 2 routes (blue) per depot-customer pair, and 2 intersection points (black). The routes with more risk are the short, direct connections between any depot-customer pair, while the longer routes are the opposite. On the right, Fig. 1b shows routes being discretized into approximately equal blocks. In MRWPDP, package orders will arrive to the depot in an exponentially distributed fashion. They are then collected into a batch, and some time is needed to prepare and load them onto the UASs. Finally, the UASs are scheduled for departure, along with a route assignment. Upon reaching the destination (customer), the UAS is removed. In this work, we

aim to solve a deterministic version of the problem where a batch of package orders is known. Thus, the MRWPDP can be formulated as a MILP.

The objective is shown in Eq. (1), where the aim is to minimize total ground delay as well as a weighted combination of risk and total travel distance. Let O be the set of all package orders, D the set of all depots, E_d the set of all outgoing edges from depot $d \in D$, and O_d, the set of package orders at depot d. The first term minimizes total ground delay, where x_k is the scheduled departure time, while t_k is the package order's known arrival time at a depot. Let the set of all package orders be given by O. The second term minimizes the weighted combination of risk and distance, represented by r_e and c_e, respectively. α, β are non-negative real value weights which control the trade-off between risk and distance, respectively. The binary variable $y_{k,e}$ is 1 if package order k is assigned to edge e, but 0 otherwise.

$$\min_{x,y} \sum_{k \in O} (x_k - t_k) + \sum_{\substack{k \in O_d, \\ e \in E_d, \\ d \in D}} (\alpha r_e y_{k,e} + \beta c_e y_{k,e}). \tag{1}$$

Next, the first set of separation constraints are formulated. Equation (2) ensures temporal separation between two consecutive UASs that are assigned to the same edge. With some abuse of notation, we can prove this constraint is correct. In the case both UASs are assigned to the same edge, the left binary variable $y_{k,e_u} = 1$ and $\sum y_{q,e_v} = 0$, and $x_k \geq x_q + b_s(e_u)$. On the other hand, if neither vehicles are assigned to the same edge, the constraint becomes either $x_k \geq x_q$ or $x_k \geq x_q - b_s(e_u)$.

$$\begin{aligned} x_k \geq x_q + (y_{k,e_u} - \sum_{e_v \in E_d \setminus \{e_v\}} y_{q,e_v}) b_s(e_u), \\ \forall d \in D, k \in O_d, e_u \in E_d, \\ q \in \{h \in O_d | t_h \leq t_k\}. \end{aligned} \tag{2}$$

Equation (3) ensures temporal separation between UASs arriving at an intersection between two different edges. Since it is not known ahead of time which edge a vehicle is assigned to, we have to consider all possible conflicts it may encounter. When two different vehicles are encountering a conflict, all the binary variables $y = 1$. As a result, the big-M term goes away on the right-hand side; what remains is $b_f(e, j)$. Thus, we have $|x_k + a(e, j) - (x_q + a(j, e))| \geq b_f(e, j)$. On the other hand, if two vehicles will never be in conflict (due to being assigned to two non-intersecting edges), the constraint needs to turn off. The right hand side becomes $\approx -M$.

$$\begin{aligned} |x_k + y_{k,e} a(e, j) - (x_q + y_{q,j} a(j, e))| \\ \geq (y_{q,j} + y_{k,e} - 1) b_f(e, j) + (y_{q,j} + y_{k,e} - 2) M, \\ \forall d \in D, k \in O_d, e \in E_d, \\ j \in \Omega(e), q \in \{h \in O_{\sigma(j)} | t_h \leq t_k\}, \\ \text{where } \sigma : E \to D. \end{aligned} \tag{3}$$

Note that Eq. (3) is a nonlinear expression. We can linearize it using an additional binary variable z in conjunction with a big-M as such using Eq. (4), (5), (6). Note that $(.)$ indicates the right-hand side of Eq. (3).

$$\begin{aligned} x_k + y_{k,e} a(e, j) - (x_q + y_{q,j} a(j, e)) \\ \geq (.) + z_{d,k,e,j,q} M, \end{aligned} \tag{4}$$

$$\begin{aligned} x_q + y_{q,j} a(j, e) - (x_k + y_{k,e} a(e, j)) \\ \geq (.) + (1 - z_{d,k,e,j,q}) M, \end{aligned} \tag{5}$$

$$z_{d,k,e,j,q} \in \{0, 1\}, \tag{6}$$

$$\begin{aligned} &\forall d \in D, k \in O_d, e \in E_d, \\ &j \in \Omega(e), q \in \{h \in O_{\sigma(j)} | t_h \leq t_k\}, \\ &\text{where } \sigma : E \to D. \end{aligned}$$

The next constraint, Eq. (7), ensures vehicles undergo a preparation lead time.

$$x_k - t_k \geq b_p(d), \qquad \forall d \in D, k \in O_d \tag{7}$$

Equation (8) ensures vehicles depart with a minimum separation of $b_t(d)$ from a given depot d.

$$\begin{aligned} x_k - x_q \geq b_t(d), \\ \forall d \in D, k \in O_d, q \in \{h \in O_d | t_h \leq t_k\} \end{aligned} \tag{8}$$

Finally, Eq. (9), (10) ensures every vehicle is assigned to exactly 1 outgoing edge, while Eq. (11) ensures departure times are non-negative.

$$\sum_{e \in E_d} y_{k,e} = 1, \qquad \forall d \in D, k \in O_d \tag{9}$$

$$y_{k,e} \in \{0, 1\}, \quad \forall k \in O_d, e \in E_d, d \in D \tag{10}$$

$$x_k \in \mathbb{R}^+, \quad \forall k \in O \tag{11}$$

Consequently, the MILP formulation is given by Eq. (1)-(11) (with the exception of Eq. (3)). Note that solving the MILP optimally scales poorly with growing problem size; the number of constraints required grows exponentially with each additional aircraft due to conflicts. As an alternative, we opted to find sub-optimal solutions by reformulating the problem as a MDP and solving it with an MCTS-based approach.

We briefly discuss the reformulation of MRWPDP as an MDP, while implementation details will be given in the next

section. Let each state $s \in S$ be defined as an ordered collection of actions $a \in A$ taken so far. The initial state s_0 is empty, while there are many goal states s_g (no more possible actions). Since all package orders are known ahead of time and are "queued up" in a line at each depot according to their arrival time t_k, we want to maintain a notion of fairness. The action space at s, A_s is given by the following: 1) choice among the package orders at the front of each depot line, and 2) choice among which route to assign to that package, and 3) the departure time, x_k. Since x_k is a continuous variable, the action space becomes infinite dimensional. However, we can avoid this issue by simply selecting over the discrete choices, which fully specifies the next t_k. Finally, the probability of selecting any action is uniformly distributed, while the reward of selecting any action is the expected objective function value.

V. Monte Carlo Tree Search with Branch/Pruning and Simulated Updates

Let each node in the search tree consists of several attributes: children, actions, state, parent, value, and count. For our problem of consideration, *actions* is the set of legal actions or "action space", where each individual action is defined as a 3-tuple, $(o, x, e) : o \in O, x \in R^+, e \in E$, where o is the delivery order id, x the departure time, and e the assigned edge. The *state* attribute is implemented as an increasing list of actions. The *parent* or *child* attribute is a pointer to another node instance in the tree. Once a node is created, its partial state is considered fixed. Additionally, a node has "*descendants*" as long as either actions or children are non-empty. The *value* is the "score" of a node, and the count is the number of times a node has been traversed. The algorithm maintains the best solution so far, called the *incumbent* node. This node gets updated each time a better solution is encountered. Unlike basic MCTS, we introduce two features: 1) *Bound-and-Prune* helps prune the tree when lower bound information is available or when no more descendants are possible, while 2) *Simulated Updates* allows the roll-out procedure to directly update the incumbent solution. Alg 1 shows the overall algorithm with our modifications; it is performed repeatedly until either a computational budget is exhausted or the root node no longer has any descendants.

A. Selection Policy

As in basic MCTS, the selection policy (Lines 8-18) is used to recursively select the most promising child node for expansion among its leaf nodes. However, if a leaf node is still expandable (its set of possible actions are non-empty) or has no children, the while loop will break early to allow for possible expansion in the subsequent phase. Line 18 is used to update the incumbent node n^* (the best solution found so far) with the selected node n, in the event n has a terminal state with a better objective value. In Line 17, U is used to "score" each child node; its aim is to maintain a proper balance between exploitation and exploration. A common implementation is given by the *Upper Confidence Bounds applied to Trees (UCT)* [8], [20]. Let n be a node

Algorithm 1: MCTS-BP-U

```
1  root.children ← ∅
2  root.actions := set of possible actions
3  root.state := state of node n
4  root.parent ← NULL // Parent node of n.
5  root.value ← ∞ // Objective value.
6  root.count ← 1
7  n* ← root // Incumbent node; deep copy.
8  while
     Computational budget > 0 and Descendants(root) ≠ ∅
     do
       // Selection Policy
9      n ← root
10     while n.state is not terminal do
11       if n has no children or n is expandable then
12         break
13       else
           // Bound and Prune
14         while (Bound(n) > n*.value ∨
            Descendants(n) ≠ ∅)∧n.parent ≠ NULL
            do
15           remove n from its parent
16           n ← n.parent
17         n ← argmax_{c∈n.children} U(c) if n.children ≠ ∅
18     update n* with n if n.value < n*.value and
        n.state is terminal
19     if if n is expandable then
         // Expansion
20       a ← randomly pop an action from n.actions
21       s' ← Move(a, n.s)
22       n' ← create an empty node
23       n'.actions ← GetLegalActions(s')
24       n'.state ← s'
25       n'.parent ← n
26       add n' to n.Children
         // Simulation Policy
27       s ← n'.s
28       for i ∈ 1 . . . numRollouts do
29         while s is not terminal do
30           A ← GetLegalActions(s)
31           a ← randomly pop an action from A
32           s ← Move(a, s)
33         n'.value ←
            min{n'.value, ComputeObjective(s)}
34         update n* with current node n' if
            n'.value < n*.value
         // Backpropagation
35       n ← n'
36       while n ≠ NULL do
37         n.value ← min{n.value, n'.value}
38         n.count+ = 1
39         n ← n.parent
40 return n*
```

from the MCTS search tree, p be the parent node of n, and $N(n) : n \mapsto n.count$ be the number of times a node was previously visited. Then, the UCT expression is:

$$U(n) = Q(p, n) + C\sqrt{\frac{\ln \mathrm{N}(p)}{\mathrm{N}(n)}} \tag{12}$$

The first term is responsible for "exploitation" behavior and is a function of the node's objective value (see Eq. (1)). We ensure $Q \in [0, 1]$ by normalizing each node's value relative to its siblings total objective. For minimization problems, each node's value may need to be negated. Additionally, the second term is responsible for "exploration"; note that as n is visited more often, the term begins to vanish. Finally, the constant C is usually recommended to be $\sqrt{2}$, but is often empirically tuned.

B. Expansion

A node is expanded (Lines 19-26) by first taking a random action from the node's action space. Next, a new consistent state s' is created using *Move*, while *GetLegalActions* creates an action space for n'. Both functions utilize *domain reduction* via inference, though exact implementation is domain dependent. By carefully implementing both functions, the search space can be reduced significantly. Our implementations of both for the MRWPDP are given as follows.

1) GetLegalActions: Consider a FIFO queue at each depot $\mathcal{Q}_d,\ \forall d \in D$. We push all UAS orders onto $\mathcal{Q}_d$ in the order of their arrival time $t_k,\ \forall k \in O_d, d \in D$. Then, the set of all legal actions is given by $\{(x, y) : x \in \mathcal{Q}_d.peek()$ **and** $y \in E_d,\ \forall d \in D\}$.

2) Move: We propagate constraints in a sequential fashion in order of greatest constraint "tightness". First, the preparation constraint of Eq. (7) is propagated. It is easy to ensure this by modifying the latest action a's time attribute. Next, the departure constraint of Eq. (8) is propagated. This can be done by checking the latest action a against all previous actions in s: for each action that belongs to Eq. (8)'s index set, the departure time can be adjusted. A similar approach follows for the edge separation constraint, Eq. (2). For the intersection separation constraint Eq. (3), the propagation is more involved. We begin by scanning over all previous actions that are in conflict with a. Each time we adjust a, we have to re-scan over the previous actions and re-propagate. This is continued until a can no longer be adjusted. At this point, create a new state by deep copying s and then assigning $s' \leftarrow s \cup \{a\}$. Finally return s'.

C. Bound and Prune

When strong bounds are available, we can prune entire branches of the search tree similar to *branch and bound* (Lines 13-16). In other words, if a better solution cannot be found through n', it is pruned. We briefly check a child node's (n') bound against the incumbent solution; If the bound is worse, we remove the n' from its parent node. Additionally, we recursively remove nodes without any children or legal actions, such as terminal state nodes. For MRWPDP, bounds are given by the LP-relaxation of the MILP formulation.

D. Simulation Policy and backpropagation

The simulation policy is used to estimate a node's value by performing several rollouts until some terminal state is reached. Its primary advantage is that it does not require any heuristic information. In most implementations, the value is averaged over many rollouts, but we found that using the best value over all rollouts to be more effective.

A random rollout consists of randomly choosing a legal action from the node's action space and then constructing a new state using *Move*. At the end of a single rollout, the best value of n' is kept. Repeating the rollouts at a greater frequency increases the likelihood of correctly estimating a node's value, but also introduces overhead. A change to the basic MCTS algorithm is the addition of *Simulation-Based Updates* given in Line 34. Since we are able reach a terminal state during each rollout, we can immediately update the incumbent solution as necessary.

After completing the simulation policy, it is necessary to perform *backpropagation* (Lines 35-39) to update the statistics of all direct ancestor nodes. These statistics directly affect the score calculation given by Eq. (12).

E. Analysis of the algorithm

Proposition 1. *Given an arbitrary sequence $< O_1, O_2, \ldots, O_n >$ on the set of delivery orders, such that $t_1 \leq t_2 \leq \ldots t_n$ is true, a feasible solution to the MILP can be easily inferred.*

Proof. We can always introduce enough ground delay for subsequent delivery orders to avoid conflicts at intersections and between each consecutive delivery order. □

Lemma 2. *"Move(a, s)" returns a* ***locally consistent*** *state s'.*

Move takes advantage of proposition 1 by incrementally filtering a state in sequence.

Theorem 3. *Alg. 1 is probalistically complete and will return an asymptotically optimal solution to MRWPDP as computational budget tends to ∞.*

VI. Computational Results

A. Setup

All the algorithms were implemented in Python 3.11 on a MacOS-based laptop, with a 2.6 GHz 6-Core Intel Core i7 processor and 16 GB of memory. Several problem instances (SF20, SF30, SF40, SF100, SF1000) based on a representative downtown San Francisco package delivery scenario were created to compare MCTS-BP-U against typical MILP solvers (Fig. 1a). In each scenario, there are two source and two sink nodes. In addition, there are two different routes available for drone delivery between each pair of source and sink nodes. Based on the problem instance, there are a total of $|O| = \{20, 30, 40, 100, 1000\}$ delivery vehicles initially queued up at the sources. Ground risk values were based on those obtained from the NASA Ground Risk Assessment Service Provider (GRASP) tool [30]. Additionally, the design of the Operational Volume Blocks in Fig. 1b followed the

Instance	MILP			
	Obj. Value	Low. Bound	Rel. Gap (%)	Comp. Time (s)
sf20	148,004.075	144,965.455	2.053	60
sf30	233,679.754	218,864.763	6.340	60
sf40	324,422.069	296,334.932	8.658	60
sf100	1,084,638.745	817,155.605	24.661	100
sf1000*	-	9,829,576.709	-	334

TABLE I: Results are shown from solving the MILP formulations given a computational time limit. *The solver did not produce a result for sf1000 in the allotted time of 120s. Instead, a lower bound is obtained from the LP relaxation after 334s. The values shown are unit-less unless otherwise stated.

	MCTS-BP-U		
Instance	Initial		
	Obj. Value	Comp. Time (s)	Rel. Gap (%)
sf20	170,936.701	0.232	15.19
sf30	267,103.437	0.421	18.06
sf40	373,509.599	0.679	20.66
sf100	1,173,619.542	3.908	30.37
sf1000	41,285,410.072	13.797	76.19
Instance	Final		
	Obj. Value	Comp. Time (s)	Rel. Gap (%)
sf20	149,167.610	60	2.82
sf30	237,442.770	60	7.82
sf40	333,849.968	60	11.24
sf100	1,112,903.899	100	26.57
sf1000	40,109,336.091	120	75.49

TABLE II: Initial and final results are shown from applying the heuristic MCTS-BP-U; the relative gap is shown with respect to the lower bound presented in the previous table.

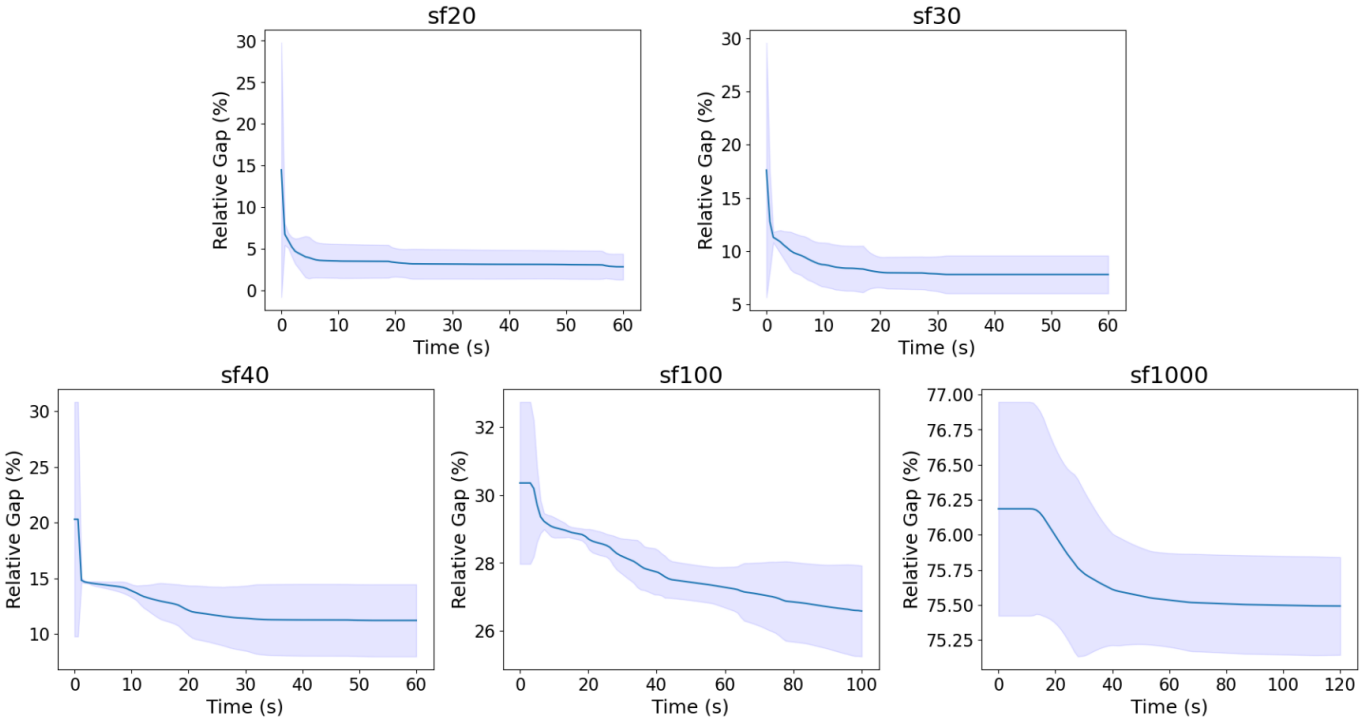


Fig. 2: The figures show the relative gap wrt. time using the heuristic, MCTS-BP-U, to solve MRWPDP. In each of the problem instances, results are averaged over 10 trials; the mean is the thin blue line, while the shaded regions represent 2 std deviations (95% confidence interval).

procedures established in [6]. We compared quality of the solutions over an allotted computational time using the relative gap, i.e. the relative difference between the upper and lower bounds. All optimization formulations (MILP, LP) were implemented in the Pyomo modeling language [31], [32]. As a final step, we validated solutions obtained by implementing them in an agent-based model [33].

B. MILP

Table I shows the average ($N = 10$) results for the MILP formulation. For all instances, the MILP solver was not able to find an optimal solution in the allotted CPU time of 120s. Instead, the best achievable objective value was recorded. Additionally, in SF1000, the MILP solver was not able to find a feasible solution in the allotted time of 120s. The quality of the objective value decreases with larger problem instances as evidenced by the growing relative gap from 2.053 to 24.661. Clearly, the MILP formulation does not scale well with problem size as shown in our example scenario in Fig. 1.

C. MCTS-BP-U

Table II shows both the initial and final average results of MCTS-BP-U with respect to allotted CPU time. The initial results correspond to the first instant a feasible solution was found. MCTS-BP-U can find a feasible solution significantly faster than the MILP solver. In the final results (bottom table), the solution quality has furthered increased. However, the MILP formulation holds a slight edge in terms of solution quality for small and intermediate problem sizes. For large problem sizes (SF1000), MCTS-BP-U can still find a solution in a relative short amount of time as opposed to the MILP solver which does not.

The average relative gap of MCTS-BP-U over time is shown in Fig. 2 with a 95% confidence interval shading. As expected, the relative gap decreases with more CPU time. This decrease is less dramatic for large problem sizes (SF1000). However for smaller problems, the relative gap becomes flat in a short amount of time suggesting diminishing returns for large computational times.

D. Validation with Agent-Based Model Simulation

An agent-based model (ABM) was developed to quickly validate solutions obtained from either using the MILP formulation or MCTS-BP-U. The relevant parts of the solution include the route assignment and take-off time variables for each UAS, which will be handed off to a waypoint following controller.

In the ABM, the UAS are modeled as mass-less points without any kinematic constraints. During each loop of the ABM simulation, every UAS listens for a velocity command from the waypoint following controller. After enough simulation time has passed, the controller will issue commands to the relevant UASs so that they begin following a predefined route. In addition, several "Operational Volume Blocks" (OVB) of uniform size have been defined over each route. As required by FAA ConOps [4], no two overlapping OVBs belonging to separate UAS operations, should be "active" at the same time. We henceforth refer to as a *conflict*. Each time there is a conflict, we increment a counter. We simulate the ABM using solution outputs from both algorithmic methods, while measuring the makespan and total number of conflicts. A table showing the metrics for the simulation can be found in Table III. The makespan values are relatively close, suggesting the benefits of using MCTS-BP-U over the MILP formulation. Additionally, there are no conflicts indicating correctness of our mathematical models and proposed approach.

Algorithm	Makespan (s)	Total Conflict Count
MCTS-BP-U	11857	0
MILP	11835	0

TABLE III: The simulation makespans between MCTS-BP-U and MILP is shown. It is defined as the final vehicle's delivery completion time.

VII. Conclusions

The rapid growth of UAS technology necessitates new algorithmic approaches to resolving potential conflicts among UASs and maintain the safety of airspace operations. In this work, we have proposed the MRWPDP, which can be seen as step towards modeling an automated package delivery scheduling service involving multiple UAS operators.

Taking inspiration from recent work in applying MCTS to combinatorial optimization as well as observing growing interest in fusing IP and CP techniques, we have created a hybrid framework called MCTS-BP-U that nicely blends the two. The resulting framework is able to solve large instances of the problem, for which the MILP formulation did not produce any feasible solution.

There are a number of ways to improve the solution methods as a future research direction. First, lower bounds are currently not being strengthened using cutting plane methods; we believe this may have a modest impact on performance by reducing the size of the search space, though it is not clear how to best incorporate branch-and-cut. Second, we did not explore parallelization of MCTS, which in theory would allow for greater utilization of CPU/GPU cores to more thoroughly search through the problem space. Though some literature suggests speed up is only modest.

References

[1] B. Canis, "Unmanned aircraft systems (UAS): Commercial outlook for a new industry," 2015.

[2] I. H. Beloev, "A review on current and emerging application possibilities for unmanned aerial vehicles," *Acta technologica agriculturae*, vol. 19, no. 3, pp. 70–76, 2016.

[3] S. O'Neill, "Electric air taxis create megadeal buzz," pp. 5–8, 2022.

[4] "UTM Concept of Operations Version 2.0 (UTM ConOps v2.0)," https://www.faa.gov/researchdevelopment/trafficmanagement/utm-concept-operations-version-20-utm-conops-v20.

[5] P. Kopardekar, J. Rios, T. Prevot, M. Johnson, J. Jung, and J. E. Robinson, "Unmanned aircraft system traffic management (UTM) concept of operations," in *AIAA Aviation and Aeronautics Forum (Aviation 2016)*, no. ARC-E-DAA-TN32838, 2016.

[6] P. Pradeep, A. A. Munishkin, K. M. Kalyanam, and H. Erzberger, "Strategic deconfliction of small unmanned aircraft using operational volume blocks at crossing waypoints," in *AIAA SCITECH 2023 Forum*, 2023, p. 1654.

[7] J. K. Lenstra and A. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[8] L. Kocsis and C. Szepesvári, "Bandit based Monte Carlo planning," in *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*. Springer, 2006, pp. 282–293.

[9] A. Cornuéjols and I. Mia, "An introduction to Monte Carlo tree search," p. 52.

[10] P. Razzaghi, A. Tabrizian, W. Guo, S. Chen, A. Taye, E. Thompson, A. Bregeon, A. Baheri, and P. Wei, "A survey on reinforcement learning in aviation applications," *arXiv preprint arXiv:2211.02147*, 2022.

[11] R. Munos *et al.*, "From bandits to Monte Carlo tree search: The optimistic principle applied to optimization and planning," *Foundations and Trends® in Machine Learning*, vol. 7, no. 1, pp. 1–129, 2014.

[12] V. Antuori, E. Hebrard, M.-J. Huguet, S. Essodaigui, and A. Nguyen, "Combining Monte Carlo tree search and depth first search methods for a car manufacturing workshop scheduling problem," p. 16.

[13] J. P. Pedroso and R. Rei, "Tree search and simulation," in *Applied Simulation and Optimization*, M. Mujica Mota, I. F. De La Mota, and D. Guimarans Serrano, Eds. Springer International Publishing, pp. 109–131.

[14] M. Shimomura and Y. Takashima, "Application of monte-carlo tree search to traveling-salesman problem," p. 5.

[15] M. A. Nguyen, K. Sano, and V. T. Tran, "A Monte Carlo tree search for traveling salesman problem with drone," vol. 6, p. 100028. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2185556020300286

[16] N. R. Sabar and G. Kendall, "Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems," vol. 314, pp. 225–239. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025514010287

[17] R. Rei, "Monte Carlo tree search applied to combinatorial optimization."

[18] A. I. Garmendia, J. Ceberio, and A. Mendiburu, "Exploratory analysis of the Monte Carlo tree search for solving the linear ordering problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, pp. 1433–1441. [Online]. Available: https://dl.acm.org/doi/10.1145/3449726.3463163

[19] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte Carlo tree search: a review of recent modifications and applications." [Online]. Available: https://link.springer.com/10.1007/s10462-022-10228-y

[20] A. Sabharwal, H. Samulowitz, and C. Reddy, "Guiding combinatorial optimization with UCT," in *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28–June1, 2012. Proceedings 9*. Springer, 2012, pp. 356–361.

[21] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[22] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," https://www.openstreetmap.org; https://www.openstreetmap.org/copyright, 2017.

[23] S. Cafieri and N. Durand, "Aircraft deconfliction with speed regulation: new models from mixed-integer optimization," *Journal of Global Optimization*, vol. 58, pp. 613–629, 2014.

[24] A. Akgunduz, B. Jaumard, and G. Moeini, "Deconflicted air-traffic planning with speed-dependent fuel-consumption formulation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1890–1901, 2017.

[25] H. Tang, Y. Zhang, and J. Post, "Predeparture flight planning to minimize operating cost for urban air mobility," *Journal of Air Transportation*, pp. 1–11, 2023.

[26] X. Yang and P. Wei, "Autonomous on-demand free flight operations in urban air mobility using monte carlo tree search," in *International Conference on Research in Air Transportation (ICRAT), Barcelona, Spain*, vol. 8, 2018.

[27] ——, "Scalable multi-agent computational guidance with separation assurance for autonomous urban air mobility," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 8, pp. 1473–1486, 2020.

[28] D. Sui and K. Liu, "A framework for optimising flight efficiency of a crossing waypoint by balancing flight conflict frequency and flight-level usage benefits," *The Aeronautical Journal*, pp. 1–28, 2023.

[29] J. Bertram, P. Wei, and J. Zambreno, "Scalable fastmdp for pre-departure airspace reservation and strategic de-conflict," in *AIAA Scitech 2021 Forum*, 2021, p. 0779.

[30] E. Ancel, T. Helsel, and C. M. Heinich, "Ground risk assessment service provider (GRASP) development effort as a supplemental data service provider (SDSP) for urban unmanned aircraft system (UAS) operations," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, 2019, pp. 1–8.

[31] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, 3rd ed. Springer Science & Business Media, 2021, vol. 67.

[32] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[33] S. C. Bankes, "Agent-based modeling: A revolution?" *Proceedings of the National Academy of Sciences*, vol. 99, no. suppl_3, pp. 7199–7200, 2002.