Mapping the Tail Assignment Problem to Vehicle Routing Problems using "Reduced Graph" Approach

Jordan Makansi

July 1, 2023

1 Introduction

The tail assignment problem (TAP) remains one of the most challenging problems in the aviation industry [**?**]. The problem is assigning a set of flights to aircraft, under practical constraints such as aircraft-route pairing, crew downtime, etc., while minimizing the cost. Existing approaches to solving TAP have primarily focused on framing the problem as a feasibility problem, and solve the problem using constraint programming [**?**]. The approach presented here makes 2 contributions: First, it poses the problem as a Min-Max problem, by minimizing the maximum cost incurred by any single aircraft and crew combination (which we refer to as an "agent"), which we believe to be more practical. The closest example to a min-max formulation of a relevant problem can be found in [1], where the objective is to minimize the maximum delay.. To our knowledge, no algorithm for solving the TAP when framed as a min-max problem, has ever been presented. Another methodological contribution of this work is that we map the TAP onto other canonical vehicle routing sub-problems in the literature that typically take place in a planar graph. By forming what we call the "Reduced Graph", we are able to map TAP to other canonical problems such as traveling salesman problem (TSP), Multi-Depot Vehicle Routing (MDVRP), and Capacitated Vehicle Routing (CVRP). This allows us to take advantage of specialized solvers for these canonical problems, rather than solving using generic MIP or IP approaches. Based on our survey of the literature, the only canonical problem that has been mapped under TAP is the exact cover problem [2].
We offer 2 heuristics to solve the Min-Max TAP: An initial heuristic that runs in worst-case $O(|E'|\log(|E'|)+ m(|E'|+2^{|E'|})$, and another that involves an outer loop that balances the burden on each aircraft, and an inner loop that solves one of the aforementioned canonical vehicle routing problems. We then present 3 different ways of embedding state-of-the-art algorithms for solving these canonical vehicle routing problems inside TAP, parameterized by "capacities" which are tuned by the outer loop. First we provide a Problem Formulation, and then for each algorihm, we offer some intuition and some analysis, before comparing them, and finally show computational results, and conclude.

2 Literature Review

3 Problem Formulation

3.1 MAEP-0

Notation and Terminology:

- $G = (V, E)$ = "complete graph", strongly connected graph representing connections between all nodes V' in demand graph.
- $G' = (V', E')$ = "demand graph", a directed multi-graph, i.e., it allows multiple edges between the same pair of nodes. This multiplicity represents the demand frequency on the corresponding segment.
- G'_j = subgraph of "demand graph" traversed by agent j

- $c(E'_j)$ = cost of arcs E'_j of demand graph, that must be covered by agent j. (Does not include cost of virtual arcs).
- $c(P_j)$ = cost of path of agent j in the complete graph, including the cost of virtual arcs, and demand arcs.
- $G_{j,\text{red}} = (V_{j,\text{red}}, E_{j,\text{red}})$ = "Reduced Graph" of Agent j.
- P_j = "route of agent j" - the path in G traversed by agent j, which covers all of its assigned arcs in G'_j, and may also include arcs in $G \setminus G'$.
- $P_{j,\text{red}}$ = path of agent j in the reduced graph $G_{j,\text{red}}$
- m = number of agents available to serve the demand
- edge $e \in E \setminus E'$ = "virtual arc"
- cost $c(e)$ of virtual arc e, and when necessary we denote $c(t_k, s_j)$ as the virtual arc cost between vertices t_k and s_j.
- cost $c(e')$ of demand arc e'
- y_j, capacity of agent j
- v_k, subpath with start and end vertices s_k, t_k.
- v_j^*, core subpath assigned to agent j
- d_j, node deviation for agent j
- N, number of subpaths.
- K, maximum number of arcs in a subpath.

Inputs:

- m
- G'
- $c(e') \qquad \forall e' \in E'$
- $c(e) \qquad \forall e \in E$

$$\min_{P_1,\ldots,P_m} \max_{j\in\{1,\ldots,m\}} c(P_j) \tag{1a}$$

$$\text{s.t.} \quad P_1 \cup \ldots \cup P_m = E' \tag{1b}$$

$$P_j = \{e_{i,j}\} \subseteq G \qquad \forall j = 1 \ldots m \tag{1c}$$

Description:

- Minimize the "burden" on any one agent (by minimizing the max cost of the subgraph G^j corresponding to the most overworked agent), while meeting the demand.
- The walk for agent j must be a connected walk in the complete graph G_j

Notice:

- G'_j need not be a connected graph.
- $c(P_j)$ = cost of demand arcs assigned to agent j plus cost of virtual arcs agent j traverses when going between subpaths of demand arcs, i.e. $c(P_j) = c(E'_j) + c(P_{j,\text{red}})$

4 Main Results

4.1 "Reduced Graph" Approach

First we describe the methodological step of forming the "Reduced Graph" by making several observations:

- Graph G' can be divided into N eulerian trails, each of which we call a "subpath" v_k. (Details for finding subpaths below).
- Each subpath has start and end vertices s_k, t_k.
- Notice that the end of one subpath t_k may be connected to the start of a different subpath s_j, by a virtual arc e.
- Given a set of subpaths $\{v_k\}$, finding the path of minimum cost to traverse all subpaths in the set $\{v_k\}$, becomes a problem of finding the minimum cost to visit all subpaths without needing to return to the starting subpath.

The figure below illustrates forming the reduced graph: The original demand graph with vertices in blue and subpaths $\{v_k, v_j, v_l\}$ is shown in Figure 1, and the resulting reduced graph shown in Figure 2.

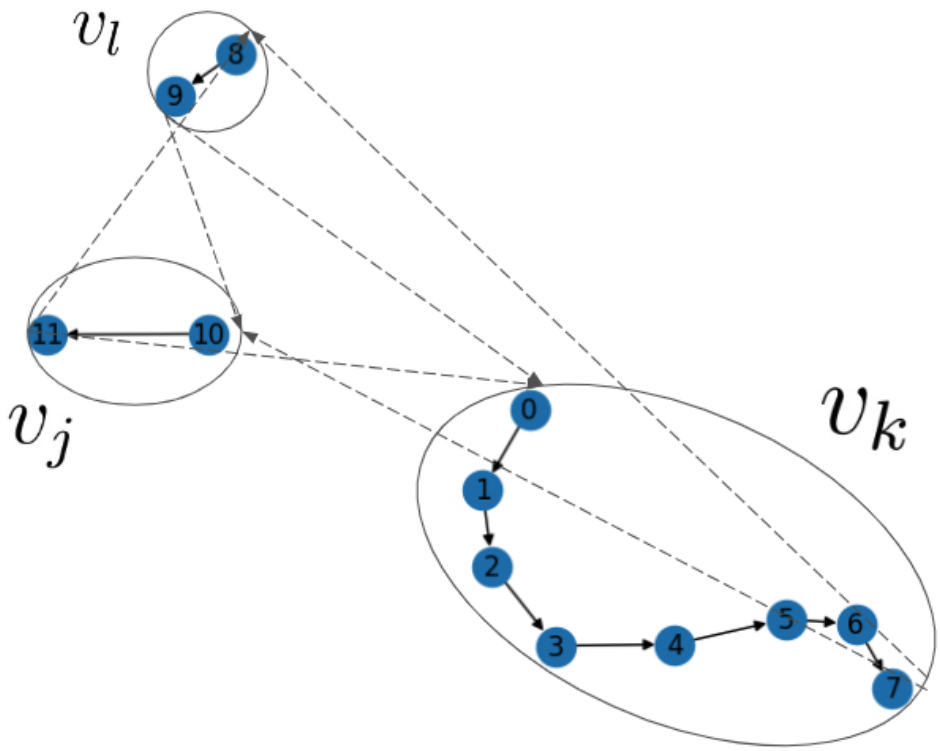

Figure 1: Demand graph with subpaths v_k, v_l, v_j

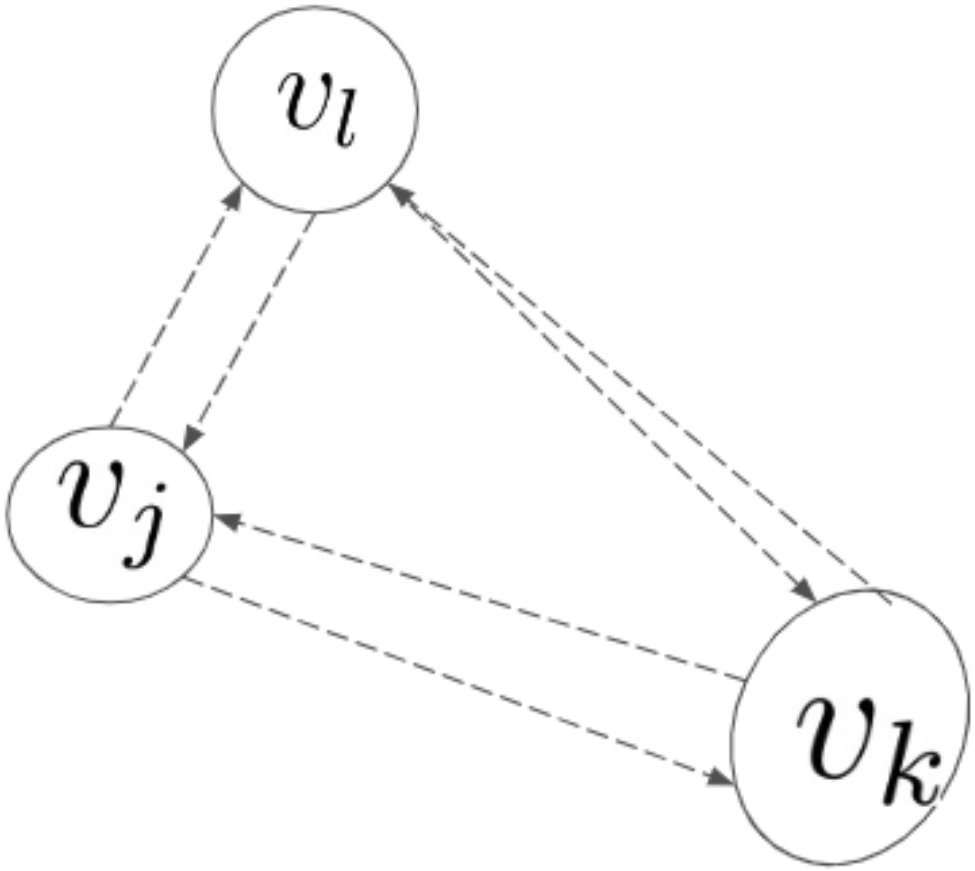

Figure 2: Reduced graph with vertices $V_{j,\text{red}} = \{v_k, v_l, v_j\}$, formed from demand graph G'

4.2 "Naive Reduced Graph" Approach

An informal description of the procedure to solve 1 using the "Reduced Graph" approach is given first, and then a more formal algorithm is presented.

- Assign arcs from E' to each agent by solving $\min_{E'_j} \max_{j=1....m} c(E'_j)$. Notice this objective is different from $c(P_j)$ in equation (1). We use it as an approximation by only considering arcs in the demand graph. The assigned arcs for each agent form G'_j.
  For each agent, do the following:
  - Find approximately the largest set of subpaths within the assigned arcs.
  - Form the "reduced graph", and solve the problem of visiting all subpaths exactly once.

The subpaths become vertices in the reduced graph. This allows us to encode the each agent's subproblem in a new graph. Each agent solves the problem of finding the shortest path that visits all vertices in the graph $G_{j,\text{red}}$. We denote the openTSP subproblem over vertices V as $\text{OPENTSP}(V)$.
Note:

- This allows us to solve each agent's subproblem in parallel.
- We take advantage of the objective to minimizing the maximum burden on any agent, by mirroring that in the computation, so that the computation time is only as long as the longest-running agent subproblem.

We transform the original problem 1 into solving a series of smaller problems of visiting all vertices as follows:

Algorithm 1 "Reduced Graph" Algorithm

Input: $G' = (V', E'),\ c(e')\forall e' \in E',\ G = (V, E),\ c(e)\forall e \in E$
Output: $P_j,\ \forall j \in \{1....m\}$

```
{E'_1, ...E'_j, ...E'_m} ← SolvePartitioning(E')
for j ∈ 1....m do
    {V_{j,red}} ← FindSubpaths(E'_j)
    Form the reduced graph G_{j,red}
    P_j ← openTSP(V_{j,red}, G)
end for
```

4.2.1 Partitioning Sub-problem

Partitioning demand arcs into approximately equal total cost is the same as the k-way partitioning problem [3]. This is solved using the largest differencing method [4], which is linear in the number of values being partitioned, in our case $O(|E'|)$. For brevity, we only show the subroutine for finding the forward path. Finding the backward path is similar except traverse incoming arcs rather than outgoing arcs. Finding subpaths is done through the following procedure: The complexity of Algorithm 11 is linear in the number arcs assigned to agent j: $O(|E'_j|)$. The algorithm FindBackwardSubpath is similar to FindForwardSubpath except with backward traversal and has been ommitted for brevity. To form the reduced graph $G_{j,\text{red}}$ we need 2 things:

- Designated start and end vertex for each subpath P^k. The last vertex in the path forward is the end vertex. And, the first vertex in the path backward is the start vertex.
- Costs connecting the end vertices of each P^{k_1} to the start vertices of other subpaths P^{k_2}. Notice that these costs are inherited from the complete graph G.

Algorithm 2 FindSubpaths

function FindSubpaths(V', E', K)
  $s \leftarrow \emptyset$
  while $E' \neq \emptyset$ **do**
    $P, E' \leftarrow$ FindSubpath(V', E', K)
    $s \leftarrow s \cup \{P\}$
  end while
  return s
end function

Algorithm 3 FindSubpath

function FindForwardSubpath(V', E', K)
  $P_f \leftarrow \{(u', v')\}$ ▷ Start path forward with arbitrary arc
  while $\exists (v', w') \in E'$ **do**
    $P_f \leftarrow P_f \cup \{(v', w')\}$ ▷ Select arbitrary arc outgoing from vertex v'
    $E' \leftarrow E' \setminus \{(v', w')\}$ ▷ Remove arc (v', w') from E'
    $v' \leftarrow w'$
  end while
  return P_f, E'
end function

function FindSubpath(V', E')
  $P_f, E' \leftarrow$ FindForwardSubpath(E')
  $P_b, E' \leftarrow$ FindBackwardSubpath(E')
  $P \leftarrow P_f \cup P_b$
  return P, E'
end function

The openTSP($V_{j,\text{red}}$) subproblem is equivalent to the following problem:

$$\min_{P_{j,\text{red}}} c(P_{j,\text{red}}) \tag{2a}$$

$$\text{s.t. } P_{j,\text{red}} \cap V_{j,\text{red}} = V_{j,\text{red}} \tag{2b}$$

4.3 "Capacity Restriction" Algorithm

This approach differs from the previous approach in primarily three ways:

- Contiguous subpaths are identified first, and are later assigned to agents, so unlike the "Reduced Graph" Approach, there is a possibility that the agents will be assigned subpaths of different total costs.
- Assignment of subtours to agents is done by clustering subpaths together before assigning clusters to agents.
- There is an outer loop which rebalances the allocation of subpaths to agents, based on their cost relative to the average cost of all agents.

The routing of agents in the inner loop may still be run in parallel, similar to the "Reduced Graph" approach.

Algorithm 4 "Capacity Restriction" Algorithm

Input: $G' = (V', E'),\ c(e')\forall e' \in E',\ G = (V, E),\ c(e)\forall e \in E, T(\text{max iterations}), c, r$
Output: $P_j,\ \forall j \in \{1....m\}$ ▷ Paths for all agents

$\{v_1, ..v_k, ...v_N\} \leftarrow$ FindSubpaths(E') ▷ Different input from Reduced Graph approach.
$\{v_1, ...v_m\} \leftarrow$ IdentifyCores(V_{red}) ▷ Identify core subpaths
$y_j^t \leftarrow (N - m)$ ▷ $\sum y_j^t \geq N - m$ req'd for feasibility of assignment problem
Construct reduced graph G_{red}, using subpaths v_k found.
while $t < T$ and $\frac{c(P)_{min} - c(P)_{max}}{c(P)_{max}} \geq r$ **do**
    $(V_{1,\text{red}}, ..., V_{j,\text{red}}, ...V_{m,\text{red}}) \leftarrow$ SolveAssignment($v_1, ...v_k, ...v_N$)
    for $j \in 1....m$ **do**
        $P_j \leftarrow$ openTSP($V_{j,\text{red}}, G$)
    end for
    $y_j^t \leftarrow$ Rebalance($(P_1, ...P_j, ...P_m), c$)
    $(P_1, ...P_j, ...P_m) \leftarrow$ RedOversizedSubpath($(P_1, ...P_j, ...P_m), c$)
end while

We now describe each of the sub-routines.

4.3.1 IdentifyCores

- Core subpaths v_j^* are identified by selecting the m largest subpaths, and assigning one to each agent.
- The justification for this is without knowing the costs of the virtual arcs connecting the subpaths, the m most costly subpaths $\{v_1^*, ..., v_j^*, ...v_m^*\}$ will have the most impact on the agent's cost.

4.3.2 Assignment sub-problem

First we define the "worst-case distance".

Definition 4.1 (Worst-case Distance)**.** The worst-case distance $c_{w.c.}(k, l)$ from subpath v_k to core v_l with start and end vertices $(s_k, t_k), (s_l, t_l)$ in the demand graph, respectively, is

$$c_{w.c.}(k, l) = \max\{c(t_k, s_l), c(t_l, s_k)\}$$

The sub-problem of assigning subpaths v_k to cores/agents is based on our "worst-case triangle inequality". The idea is that given any three subpaths, the the worst-case distance between each pair of them is likely to satisfy the triangle inequality (details are left to the Analysis section). We feel that using the maximum as a distance metric is reasonable because our objective is to minimize the maximum cost since ultimately the openTSP sub-problem solved by each agent will determine which of the two directions between two subpaths is more optimal in P_j. If the "worst-case triangle inequality" holds, then under a Min-Max problem, a reasonable heuristic is assigning two subpaths v_k, v_l to the same core v_j^*. This allows considering their distance to v_j^*, without directly considering their distance to each other.
The worst-case triangle inequality allows us to solve the assignment sub-problem by heuristically mapping it to some canonical vehicle routing problems, which typically take place in a planar graph. In its most general form, the sub-problem of assigning subpaths to cores/agents each with capacities, can be solved by casting the subproblem as an assignment problem. The assignment problem will be explained through the lense of the common task/worker interpretation:

- The subpaths v_k are the tasks.
- The cores v_j^* are the workers.
- The capacities y_j are the limits on total size that core v_j^* can carry.
- The cost of assigning a subpath v_k to core subpath v_j^* is $c_{w.c.}(k, j)$ - the "worst-case distance" between subpaths v_k and v_j^*.
- Objective is to minimize the sum of costs. Stated in terms of our notation:

$$\min_{x_{k,j}} \sum_{k,j} c_{w.c.}(k, j) x_{k,j} \tag{3a}$$

$$\text{s.t.} \sum_{j} x_{k,j} = 1 \ \forall k \tag{3b}$$

$$\text{s.t.} \sum_{k} c(v_k) x_{k,j} \leq y_j \ \forall j \tag{3c}$$

The assignment sub-problem is parameterized by the capacities y_j which are controlled by the outer loop of the CapacityRestriction algorithm. In this work, we explore 3 variations of this assignment problem. The first is solving 3 at each iteration, which is an integer program (IP). The second is a simpler version, where the costs of the subpaths are assumed to be approximately equal, and the assignment problem reduces to a min cost flow (MCF) problem which can be solved using a linear program. The third is an iterative procedure which attempts to solve the assignment problem by a clustering algorithm that runs in $O(m^2)$.

Min Cost Flow (MCF).If the costs of subpaths $c(v_k)$ are assumed to be approximately equal, then the capacities y_j in 3 correspond to limiting the *number of subpaths*, rather than the subpath cost. Notice this assumption does not hold true in general because the output of FindSubpaths is non-deterministic.

$$\min_{x_{k,j}} \sum_{k,j} c_{w.c.}(k,j)x_{k,j} \tag{4a}$$

$$\text{s.t. } \sum_{j} x_{k,j} = 1 \ \forall k \tag{4b}$$

$$\text{s.t. } \sum_{k} x_{k,j} \leq y_j \ \forall j \tag{4c}$$

Remark 1. *If something is known about the topology of the demand graph G' so that a sub-routine can be called instead of* FindSubpaths *and can output subpaths $\{v_1, ...v_k, ...v_N\}$ such that $c(v_j) = c(v_k)\ \forall k \neq j$, then the sub-routine* ReduceOversizedSubpath *is not necessary.*

Clustering Notice that the assignment problem considers the costs of subpaths, whereas MCF does not. So, we would expect it to perform better, especially in cases where the costs of subpaths are quite different. However, the assignment problem requires solving an IP, whereas MCF solves an LP. We repurpose a clustering algorithm used in capacitated vehicle routing problem (CVRP) [5], which considers the costs of subpaths in its optimization, but runs in $O(m^2)$. The routine for accomplishing this is shown in 5.

Algorithm 5 "Clustering" Algorithm

Input: $(P_1, ...P_j, ...P_m), \{v_1, ..v_k, ...v_N\}$ ▷ Paths for all agents
Output: $\{V_{1,\text{red}}, ..., V_{j,\text{red}}, ...V_{m,\text{red}}\}$ ▷ New subpath assignments

for $j_1 \in 1....m$ **do**
  for $v_k \in V_{j,\text{red}}$ **do**
    for $j_2 \in 1....m$ **do**
      if $j_1 \neq j_2$, and $c_{w.c.}(k, j_2) \leq c_{w.c.}(k, j_1)$, and $c(P_{j_2}) + c(v_k) \leq y_{j_2}$ **then** ▷ If subpath v_k is closer by w.c. distance to agent j_2, and has sufficient available capacity, then reassign to agent j_2.
        $V_{j_2,\text{red}} = V_{j_2,\text{red}} \cup \{v_k\}$
      end if
    end for
  end for
end for
Return $\{V_{1,\text{red}}, ..., V_{j,\text{red}}, ...V_{m,\text{red}}\}$

4.3.3 Rebalancing sub-problem

Now we describe the rebalancing procedure:

- For each agent, compute the "cost deviation" $d_j = \frac{c(P_j) - \bar{c}(P)}{\bar{c}(P)}$ where $\bar{c}(P)$ is the average cost of all of the agents.
- For agents whose cost is above average, decrease their capacities y_j by $c * d_j$, where c is an algorithm parameter, starting with the agent with highest cost.
- If too much total capacity was decreased, then the assignment problem will become infeasible: $\sum_{j=1}^{m} y_j < N - m$. In this case sort agents with negative cost deviations (below the average) in ascending order, and add to each of their capacities until the sum of capacities $\sum_{j=1}^{m} y_j = N$.

4.3.4 ReduceOversizedSubpath

First we justify the necessity for this subroutine:

- The approaches RG and CR differ in how they identify subpaths. Since the number and size of subpaths v_k is unknown at the start of the algorithm, given a set of demand arcs (in the case of RG, a subset of E'_j is given for each agent), from the demand graph a greedy approach is used to find subpaths, starting at an arbitrary arc.

- In larger graphs however, this can create an oversized subpath that is disproportionally high cost as compared to the other subpaths.
- In RG the consequences of this potential pitfall are less severe because it divides the demand arcs amongst the agents initially, before finding subpaths. In CR however, subpaths are found first, which can lead to a single oversized subpath assigned to one agent.

We use the following procedure to detect, and dissolve oversized subpaths.

- If in an iteration of Algorithm 4 there is an agent j^* who is assigned only a single subpath v_N, and that agent's cost is currently the largest, then dissolve subpath v_N.
- Dissolve v_N into m segments: One segment of cost $c(v_N^*) = \frac{\sum c(P_j) - c(v_N)}{m-1}$ and the remaining $m-1$ segments of cost $\frac{c(v_N) - c(v_N^*)}{m-1}$ (justification below). Dissolving the subpaths is accomplished by calling the FindForwardSubpath, but instead of starting at an arbitrary arc, start at s_N, the start vertex of the subpath. We know a subpath of size $c(v_N^*)$ exists, starting at s_N, because subpaths were created by FindSubpaths, which calls sub-routine FindForwardSubpath. Input s_N and output t_N are not shown in 6, but may be considered optional.
- Assign v_N^* to agent j^*, and assign one of each of the remaining $m-1$ segments to each of the other agents arbitrarily.

The justification for the cost division is in an effort to maintain approximately equal costs across agents:

- $\sum c(P_j) - c(v_N)$ is the burden at the current iteration incurred by all agents except j^*. If we assume that the rebalancing procedure has approximately leveled the cost across all agents except agent j^*, then we can assign approximately the same buren to agent j^* by assigning this largest segment to agent j^*.
- The remaining demand arcs in subpath v_N are divided into segments of approximately equal cost and assigned to the agents.

5 Analysis

For our analysis we have the following practical assumptions.

Assumption 1. *The cost of virtual arcs $c(e)$ have the following properties:*

1. *The cost of the virtual arc between two vertices i, j is higher than the cost of the demand arc between them:*
$$c(e_{i,j}) > c(e'_{i,j}),\ \forall e' \in E', \forall e \in E$$

2. *The cost of virtual arcs obey the triangle inequality:*
$$c(v_i, v_k) \leq c(v_i, v_j) + c(v_j, v_k),\ \forall v_i, v_j, v_k \in V$$

3. *The cost of the virtual arcs are i.i.d. and distributed uniformly with upper/lower bounds $0 < a_v < b_v$.*
$$c(e) \sim U[a_v, b_v]$$

Algorithm 6 "Reduce Oversized Subpath" Algorithm

Input: $(P_1, ...P_j, ...P_m), \{v_1, ..v_k, ...v_N\}$ ▷ Paths for all agents
Output: $\{V_{1,\text{red}}, ..., V_{j,\text{red}}, ...V_{m,\text{red}}\}$ ▷ New subpath assignments

if There does not exist an agent j^* such that: $V_{j^*,\text{red}} = \{v_N\}$, and $c(v_N) = \max_j c(P_j)$ **then**
  Return $\{V_{1,\text{red}}, ..., V_{j,\text{red}}, ...V_{m,\text{red}}\}$ ▷ No oversized subpath exists, so return subpaths unchanged
end if
$c(v_N^*) \leftarrow \frac{\sum c(P_j) - c(v_N)}{m-1}$
$\{v_N^*, E_N'^*, t_N^*\} \leftarrow$ FindForwardSubpath(E_N', s_N) ▷ Dissolve subpath v_N into new segment of cost $c(v_N^*)$ starting at s_N.
$V_{j^*,\text{red}} = \{v_N^*\}$ ▷ Assign newly reduced size subpath to agent j^*
$s_k \leftarrow t_N^*$ ▷ Terminal vertex of previous subpath becomes starting vertex
for $j \in 1....m-1$ **do**
  $\{v_k, E_k', t_k\} \leftarrow$ FindForwardSubpath(E_k', s_k) ▷ Find new subpath starting at s_k
  $V_{j,\text{red}} = V_{j,\text{red}} \cup \{v_k\}$
  $s_k \leftarrow t_k$
end for
Return $\{V_{1,\text{red}}, ..., V_{j,\text{red}}, ...V_{j^*,\text{red}}\}$

5.1 Single-Agent

The following lemmas give intuition for the RG algorithm for solving **??** by find subpaths, and then solving an openTSP sub-problems for connecting each agent's subpaths.

Lemma 1. *For a single agent, in expectation it is lower cost to traverse as few virtual arcs as possible. More formally, if $|P_{j,red}^1| < |P_{j,red}^2|$ then in expectation on the cost of virtual arcs, $c(e)$,*

$$\mathbb{E}\left[c(E_j') + c(P_{j,red}^1)\right] \leq \mathbb{E}\left[c(E_j') + c(P_{j,red}^2)\right]$$

Proof. The total cost of virtual arcs $c(P_{j,\text{red}}^1) = \sum_{k=1}^{|P_{j,\text{red}}|} c(e_k)$, on average will be higher if there are more virtual arcs. □

Lemma 2. *Given a sequence of 2 demand arcs, $\{e_1', e_2'\}$, where $e_1' = (u_1', v_1')$, $e_2' = (u_2', v_2')$, it is always lower cost to go directly from end of one demand arc to start of another demand arc, without any intermediate vertices.*

Proof. If an intermediate vertex v were visited in going from v_1' to u_2', then because of assumption 1, part 2, the cost will always be greater. □

5.2 "Reduced Graph" vs "Capacity Restriction"

The main observations in this section are:

1. Without the ReduceOversizedSubpath subroutine, RG may outperform CR if the sub-routine FindSubpath(E') finds a subpath v_k whose cost $c(v_k)$ is sufficiently high.
2. By adding subroutine ReduceOversizedSubpath, which detects and "dissolves" large subpaths v_k, CR can avoid this potential pitfall at nominal extra computational cost.

Lemma 3. *Assuming:*

- *The degree of all vertices v in the demand graph is less than or equal to 2: $d(v) \leq 2$.*
- *G' is connected.*

The solution given as output from ReducedGraph$(G', c(e), ...)$ *is bounded above by:*

$$\mathbb{E}\left[\text{ReducedGraph}(G', c(e), ...)\right] \leq \frac{c(E')}{m} + \left[|E'| - m\right] \frac{a_v + b_v}{2}$$

where the expectation is taken with respect to $c(e)$*.*

Proof. The proof is as follows:

- The demand arcs are initially divided amongst the agents equally, so that each agent incurs a cost of $\frac{c(E')}{m}$.
- SolvePartitioning may have non-deterministic output since there may be multiple solutions to the multi-way partitioning problem. We consider the worst case in terms of the cost of any single agent:
- If G' were strongly connected, then since $d(v) \leq 2$, then $|E'| = 2$, the maximum number of disconnected arcs that could be assigned to an agent would be 1. If G' were weakly connected, then since $d(v) \leq 2$, then G' consists of a line of weakly connected arcs. In a worst-case scenario, one of the agents j is assigned $|E'| - (m-1)$ arcs, none of which are connected, and the remaining $m-1$ arcs are assigned one per agent for the remaining $m-1$ agents.
- This makes the objective value for the problem openTSP$(V_{j,\text{red}})$ solved by each agent dependent on the size of $V_{j,\text{red}}$. In the worst case scenario described above, $|V_{j,\text{red}}| = |E'| - (m-1)$. Let $\bar{V}$ be the set of vertices in the reduced graph of size $|E'| - (m-1)$.
- Even if the costs of virtual arcs were known a priori the following inequality may not hold for all j in general

  $$\text{openTSP}(V_{j,\text{red}}) \leq \text{openTSP}(\bar{V}) \ \forall j$$

  since more vertices as input to openTSP does not necessarily imply a higher cost. However, the inequality does hold in expectation on the cost of the virtual arcs:

  $$\mathbb{E}\left[\max_j \text{openTSP}(V_{j,\text{red}})\right] \leq \mathbb{E}\left[\text{openTSP}(\bar{V})\right]$$

  and because a uniform distribution was assumed $\mathbb{E}\left[\text{openTSP}(\bar{V})\right] = \left[|E'| - m\right] \frac{a_v + b_v}{2}$. Finally, combined with the cost of the demand arcs:

  $$\mathbb{E}\left[\text{ReducedGraph}(G', c(e), ...)\right] \leq \frac{c(E')}{m} + \left[|E'| - m\right] \frac{a_v + b_v}{2}$$

□

Notice nothing was assumed about the cost of the demand arcs $c(e')$.

Remark 2. *The above analysis was conservative and impractical since it assumes that* $d(v) \leq 2$ *for all vertices.*

Notice that the following cannot be guaranteed to hold in general:

$$c_{w.c.}(k, l) \leq c_{w.c.}(k, j) + c_{w.c.}(j, l)$$

However, based on the distributions of the costs of virtual arcs, we can compute the probability with which it holds:

Remark 3. *For subpaths* v_k, v_l, v_j *as* b_v *increases the following approaches 1:*

$$\mathbb{P}\left[c_{w.c.}(k, l) \leq c_{w.c.}(k, j) + c_{w.c.}(j, l)\right]$$

This has been mentioned as a remark since the below analysis is incomplete. However, simulation results show that for any fixed a_v, this inequality holds as $b_v \to \infty$.

- The probability distribution for the maximum of two i.i.d. uniformly-distributed random variables $c(k,l) \sim U[a_v, b_v], c(k,j) \sim U[a_v, b_v]$

$$f_{c_{w.c.}}(x) = \frac{2(x - a_v)}{(b_v - a_v)^2}$$

  where $x \in [2a_v, 2b_v]$.

- (to be completed).

Remark 4. *The above assumptions are conservative. However, if something is known about the distribution of the costs of the subpaths $c(v_k)$, then if the following holds, then the worst case triangle inequality also holds:*

$$c_{w.c.}(k,l) + c(v_k) \leq c_{w.c.}(k,j) + c_{w.c.}(j,l)$$

*Because of assumption 1 then it might be practical that $c(v_k)$ is not very costly as compared to the virtual arcs **connecting** the subpaths. An practical example of this is in aircraft/crew maximum duration constraints. Crews and aircraft can only fly flight segments consecutively for a limited time before the crew and/or aircraft need downtime or maintenance, respectively. A close example of this can be found in [1], in which they seek balanced flight durations.*

6 Simulation Results

To our knowledge, no known algorithms exist for solving TAP as a Min-Max problem. However, in searching for similar Min-Max benchmarking problems, we did notice that CR-MCF embeds the problem Min-Max Multi-Depot Vehicle Routing Problem (MDVRP). So, we give due credit to the authors of [6]. Our approach is slightly different, since we adjust capacities based on total cost rather than just distance, but the rebalancing procedure is very much the same. It would be nice to benchmark using an algorithm that solves the TAP specifically, but no existing algorithms could be found which solve TAP as a Min-Max problem. In future versions of this work, we will benchmark using an algorithm that solves the TAP as a minimization problem, except we will reformulate it to minimize an upper bound on the cost incurred by one aircraft. The main observations this section are:

- As expected, CR outperforms RG, with an objective function value at least 50% lower for any of CR algorithm variants.
- CR-Assignment outperforms CR-MCF and CR-Clustering on moderately-sized graphs, except when the cost distribution for virtual arcs is sufficiently wide, e.g. $c(e) \sim U[10, 500]$.
- For larger graphs, CR-Assignment is frequently unable to solve the assignment problem after 6 hours, indicated by "–".
- Even without guaranteeing that the sizes of subpaths $c(v_k)$ are approximately equal, in all scenarios CR-MCF is no more than 34% less optimal (averaged over 10 runs) as compared to the most optimal algorithm.

The simulation results were run under the following conditions:

- Implemented in python 2.7, run on USC's HPC Cluster using a single 64-bit core with 2.6GHz.
- Averaged over 10 runs, using parameter settings shown below.

The table below shows results for each of the three approaches, for various problem sizes and distributions on $c(e)$, with the lowest objective function value in **bold**. We use $V' - E' - m$ to denote size of the demand graph G'.

Algorithm	c	r
CR-Assignment	10	0.1
CR-MCF	2	0.1
CR-Clustering	2	0.1

Scenario	$c(e')$ Dist'b'n	$c(e)$ Dist'b'n	RG	CR-Assignment	CR-MCF	CR-Clustering
30-60-5	$U[1,5]$	$U[10,20]$	159	**79**	85	80
30-60-5	$U[1,5]$	$U[40,50]$	367	**153**	172	175
30-60-5	$U[1,5]$	$U[10,50]$	335	**84**	113	110
30-60-5	$U[1,5]$	$U[10,500]$	1115	399	**388**	390

Scenario	$c(e')$ Dist'b'n	$c(e)$ Dist'b'n	RG	CR-Assignment	CR-MCF
100-2000-20	$U[1,5]$	$U[10,20]$	1346	**275**	280
100-2000-20	$U[1,5]$	$U[10,50]$	913	–	**415**
100-2000-20	$U[1,5]$	$U[40,50]$	1819	–	**497**
100-2000-20	$U[1,5]$	$U[10,500]$	3031	–	**712**

Scenario	$c(e')$ Dist'b'n	$c(e)$ Dist'b'n	RG	CR-Assignment	CR-MCF
200-2000-50	$U[1,5]$	$U[10,50]$	674	–	**331**
200-2000-50	$U[1,5]$	$U[40,50]$	1332	1409	**593**
200-2000-50	$U[1,5]$	$U[10,500]$	3165	–	**965**

Solution times below:

Scenario	RG	CR-Assignment	CR-MCF	CR-Clustering
30-60-5	0.08	1.66	0.11	0.22
100-2000-20	0.6	117	3	–
200-2000-50	4.13	41085	2.49	–

As mentioned above, even without guaranteeing that the sizes of subpaths $c(v_k)$ are approximately equal, in all graph scenarios CR-MCF is no more than 34% less optimal (averaged over 10 runs) as compared to the most optimal algorithm. We believe this is for two reasons: Firstly, because the ReduceOversizedSubpath sub-routine limits the sizes of subpaths. Secondly, the capacities are adjusted based on the total cost incurred by an agent $c(P_j) = c(V_{j,\text{red}}) + c(E_{j,\text{red}})$, which includes the costs of subpaths, so in adjusting the capacities in Rebalance, it if an agent is overburdened due to a disproportionally large/small subpath, then the capacity y_j would be adjusted accordingly in the next iteration.

7 Conclusion

In conclusion, using this Reduced Graph approach to solving the Min-Max TAP can be effective and solved efficiently.

References

[1] Michael Jünger, Matthias Elf, and Volker Kaibel. Rotation planning for the continental service of a european airline. 2003.

[2] Pontus Vikstål, Mattias Grönkvist, Marika Svensson, Martin Andersson, Göran Johansson, and Giulia Ferrini. Applying the quantum approximate optimization algorithm to the tail-assignment problem. *Physical Review Applied*, 14(3):034009, 2020.

[3] Stephan Mertens. The easiest hard problem: Number partitioning, 2003.

[4] Narendra Karmarkar and Richard M Karp. *The differencing method of set partitioning.* Computer Science Division (EECS), University of California Berkeley, 1982.

[5] Kwangcheol Shin and Sangyong Han. A centroid-based heuristic algorithm for the capacitated vehicle routing problem. *Computing and Informatics*, 30:721–732, 01 2011.

[6] Xingyin Wang, Bruce Golden, and Edward Wasil. The min-max multi-depot vehicle routing problem: heuristics and computational results. *The Journal of the Operational Research Society*, 66(9):1430–1441, 2015.