Algorithm Design and Analysis - Final Project

Carlos Perez-Guerra, Liu Yancen

July 28, 2019

Task 1

This task is a special case of the well-studied Vehicle Routing Problem (VRP) family. The VRP is a problem of enormous practical relevance and one of the most studied problems in Operations Research, with more than 780,000 Google Scholar entries, increasing at a rate of 20.000 new entries every year. Toth and Vigo (2001) and Golden, Raghavan, and Wasil (2008) are the standard references for different algorithmic approaches.

Notation

Table 1: Notation table

		General notation
S	$\triangleq$	Sets are denoted using capital Latin letters
$\mathcal{C}, \mathcal{G}$	$\triangleq$	Collections and graphs are denoted in capital calligraphic lettering
$\mathbf{M}$	$\triangleq$	Matrices are denoted in capital boldface
$\mathbf{p}$	$\triangleq$	Vectors are denoted in lowcase boldface
		Model
$\mathcal{G} = (V, E)$	$\triangleq$	Locations, as in the problem statement
v_A	$\triangleq$	Airport vertex (depot).
$l \in \mathbb{N}$	$\triangleq$	Amount of locations excluding depot. $l = \lvert V - \{v_0\}\rvert$
$n \in \mathbb{N}$	$\triangleq$	Amount of passengers
$m \in \mathbb{N}$	$\triangleq$	Amount of buses
$\mathbf{d} \in \mathbb{N}^l$	$\triangleq$	Passenger demand per destination. $\lvert\mathbf{d}\rvert_1 = \sum d_i = n$
$\mathbf{k} \in \mathbb{N}^m$	$\triangleq$	Capacity of buses of passengers per destination. $\lvert\mathbf{k}\rvert_1 \geq n$

Problem 1

We shall refer to the problem in this task as the Multiple Capacitated Bus Routing Problem, or MCBRP for short.

Lemma 1. *MCBRP is NP-Hard.*

Proof. To show that it is NP-Hard, we note that it has TSP as a special case: simply construct an instance where the demand vector is 1 for all vertices and the bus capacity (for a single bus) is $K_1 \geq l$. The solution will only contain one route, since the demand can be fit into a single bus of size l, which must be the shortest possible route that visits each vertex and returns to the origin vertex (the airport). Since the map and the origin vector can be chosen by us, we have TSP $\leq_P$ MCBRP. □

Remark. *Both TSP and MCBRP are NP-Hard but not NP-Complete, since we cannot verify in polynomial time whether a solution given by an oracle is optimal.*

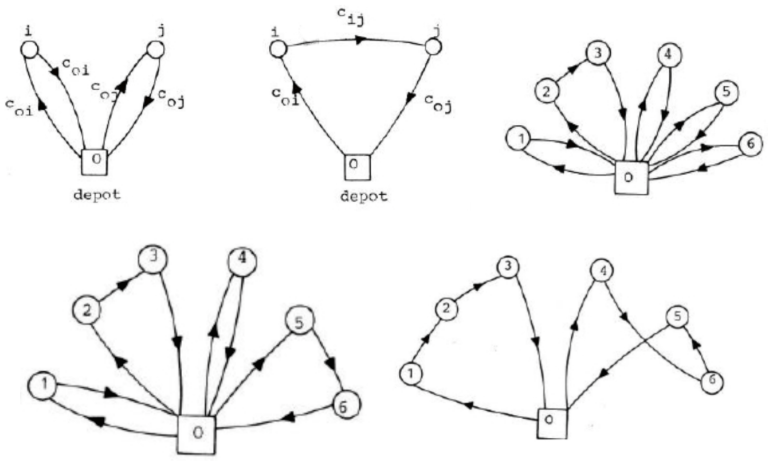

Figure 1: A pictorial summary of the Clarke-Wright algorithm

Problem 2

Exact algorithms, that guarantee to find the optimal solution, can reliably solve instances with up to 200 locations in a reasonable amount of computing time (Toth and Vigo (2001)). For larger instances, a large number of heuristics are available, of which the Clarke and Wright savings algorithm (Clarke and Wright (1964)) is one of the most widely used, and arguably one of the best-known heuristics in the entire field of Operations Research (Sörensen, Arnold, and Palhazi Cuervo (2019)).

The Clarke-Wright algorithm (Fig. 1) starts from a solution in which each of the l locations is visited in a separate tour. For each location pair, the algorithm then determines the saving that would result from visiting these locations directly and creates a savings array by sorting these $n\frac{n-1}{2}$ savings in decreasing order. Finally, the algorithm greedily connects the locations with the greatest savings if:

The locations are exterior (visited first or last before leaving or returning from the depot)	τ_1
They are not already visited in the same tour, and	τ_2
Capacity constraints are not violated.	τ_3

The performance guarantee for Clarke-Wright is known to be $O(\log n)$, although it often performs much better (Brecklinghaus and Hougardy (2015)). This result is very recent, and the proof is non-trivial. No better approximation algorithms are known, although heuristics with better empirical performance are also used in practice. Hence, MCBRP $\in$ log-APX.

A straightforward extension of such a greedy algorithm is to add a controlled randomization in the greedy selection rule, to allow the algorithm to generate a different solution at each iteration. This idea has been popularized in the GRASP metaheuristic, which stands for Greedy Randomized Adaptive Search Procedure (Feo and Resende (1995)). In the CW case, the GRASP flavour (Algorithm 1) modifies the selection so that one of the top δ savings are picked at random instead of the best one. The algorithm is then run several times and the best solution returned. Sörensen, Arnold, and Palhazi Cuervo (2019) showed that CW-GRASP usually returns the optimal solution for instances under 50 locations through experiments with well-known datasets. In our own Python implementation we found that, on average, the CW-GRASP heuristic reduces cost by 8.2% with regards to the original CW solution, with 2.4% standard deviation, evaluated on 100 randomly generated runs of ~300 passengers.

Figure 2: Left: Solution of a random instance with ~300 orders and capacity 100 by CW-GRASP. Total cost was 2.14. Right: Initial solution to the same instance by deterministic CW. Cost was 2.99. Hence, in this case, the GRASP metaheuristic reduced the cost by 11.3%,

Algorithm 1 Clarke-Wright GRASP Algorithm,

function CW-GRASP-Base($\mathcal{G}, \delta, K$)
  $S \in \mathbb{R}^{l \times l}$
  for i $\in \{1, 2, .., l\}$ **do**
    for j $\in \{1, 2, .., l\}$ **do**
      if i=j **then**
        $S(i, j) = \infty$
      else
        $S(i, j) = d(v_A, v_i) + d(v_A, v_j) - d(v_i, v_j)$
  $\mathcal{R} \leftarrow \varnothing$
  for i $\in \{1, 2, .., l\}$ **do**
    Make-Set(v_i) *Disjoint Set Forest implementation*
    $r_i \leftarrow \{v_i\}$
    $\mathcal{R} \cup \{r_i\}$
  $S' \leftarrow$ Make S into a 1-dimensional array and sort in decreasing order (omitting ∞ values).
  while $S' \neq \varnothing$ **do**
    $i, j, s_{ij} \leftarrow$ Randomly pop an element from head($S', n = \delta$)
    if $\tau_1 \wedge \tau_2 \wedge \tau_3$ **then**
      $\tilde{i}, \tilde{j} \leftarrow$ Find-Set[i],Find-Set[j]
      Merge-Routes($\tilde{i}, \tilde{j}$)
  return $\mathcal{R}$

function CW-GRASP($\mathcal{G}, \delta, K$,iterations)
  $\mathcal{R} \leftarrow$ CW-GRASP-Base($\mathcal{G}, \delta = 1, K$)
  for $i \in \{1, 2, ...,$iterations$\}$ **do**
    $\mathcal{R}' \leftarrow$ CW-GRASP-Base($\mathcal{G}, \delta, K$)
    if $\mathcal{R}'$ is better than $\mathcal{R}$ **then**
      $\mathcal{R} \leftarrow \mathcal{R}'$
  return $\mathcal{R}$

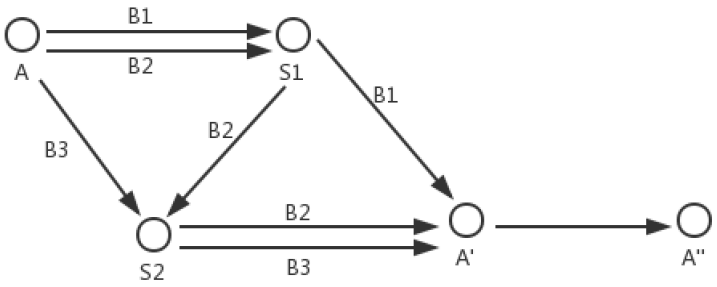

Figure 3: Example: Graph Reconstruction Step(1)

Successive Approximation Heuristic For the Heterogeneous Bus Routing Problem

The case where each vehicle is allowed a different capacity constraint is known as the heterogeneous vehicle routing problem. A commonly used heuristic for this case is to consider vehicles with higher capacities first. The base algorithm, in this case CW-GRASP, is called to generate all routes for the vehicle with the highest capacity, then the route that is the most full (equivalently, least seats are empty) is retained (Algorithm 2). The algorithm is successively called with vertices from previous solutions removed. An approximation bound for this heuristic is not known (Yaman (2006)).

Algorithm 2 Clarke-Wright GRASP Algorithm for Heterogeneous Bus Routing,

function CW-GRASP-HETEROGENEOUS($\mathcal{G}, \delta, \mathbf{k}$,iterations)
  $\mathcal{R} \leftarrow \varnothing$
  $\mathbf{k}' \leftarrow$ Sort $\mathbf{k}$ in decreasing order
  for $k_j \in \mathbf{k}'$ **do**
    $\mathcal{R}' \leftarrow$ CW-GRASP($\mathcal{G}, \delta, k_j$)
    $R_j \leftarrow$ Route with least seats remaining $\in \mathcal{R}'$
    $\mathcal{G} \leftarrow (V - \{\text{vertices in } R_j\}, E)$
    $\mathcal{R} \leftarrow \mathcal{R} \cup R_j$
  return $\mathcal{R}$

Task 1 Simulation Results

We tested the performance of CW vs. CW-GRASP. CW-GRASP improved the performance 8% on average, with 4% standard deviation.

Acknowledgements

- Geographical data courtesy of Beijing City Lab (www.beijingcitylab.com).

Task 2

Problem 1

For the first algorithm design, we reconstruct a new graph. For example, there are two stations named S_1 and S_2. And 3 bus lines $B_1 = \{Airport, S_1, Airport\}$, $B_2 = \{Airport, S_1, S_2, Airport\}$ and $B_3 = \{Airport, S_2, Airport\}$. To reconstruct the graph, add vertex A' and A'', A' represent that all the bus lines will return to Airport at last, and A'' is useful in the algorithm. As you can see in Fig. 3.

Then allocate capacity to the graph. I use $Cap(S_i)$ to denote the amount of customers whose destination is S_i. As you can see in Fig. 4, the capacity of edges between any two nodes in $\{A, S_1, S_2\}$ is determined by

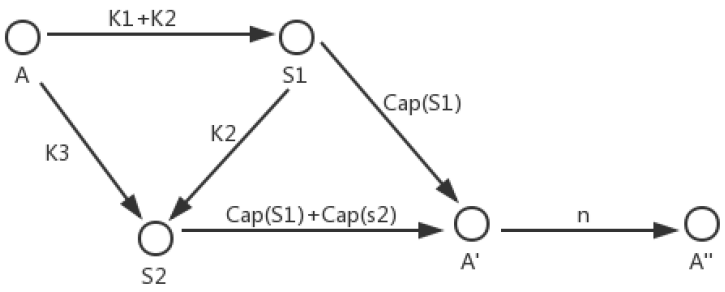

Figure 4: Example: Graph Reconstruction Step(2)

Figure 5: Example: Graph Reconstruction

the bus lines pass through it. For example, there are B_1 and B_2 pass through $edge\{A, S_1\}$, so the capacity of $edge\{A, S_1\}$ is $K_1 + K_2$. And for the edges between $\{S_1, S_2\}$ and A', because the paths ended with S_2 passed through S_1 and S_2, so the capacity of $edge\{S_2, A'\} = Cap(S_1) + Cap(S_2)$. The edges ended with S_1 only passed through S_1, so the capacity of $edge\{S_1, A'\} = Cap(S_1)$. As for $edgeA', A''$, its capacity is equal to the total amount of customers, which is n. This edge serves for the algorithm.

There may also be situation like Fig. 5, bus line B_1 passed through $\{S_1, S_2\}$ and bus line B_2 passed through $\{S_2, S_1\}$. In the graph reconstructed, the two directions remained, and the capacity of $edge\{S_1, S_2\}$ is K_1 and the capacity of $edge\{S_2, S_1\}$ is K_2. But as far as the algorithms I've looked at, there are few algorithms for solving the maximum flow problem in bidirectional graphs. So I also assume that this will not happen.

So the rule to reconstruct a new graph is: 1. Add two new vertex A' and A''. 2. Reconnect all the edges that from the last station return to A to A'. 3. Add edge from A' to A''. 4. Merge all the edge between two vertexes and have the same direction to one edge, the capacity of the new edge is the sum of the capacity of bus line corresponding to the original edges before merging. But for edges between vertexes and A', also merge them, but the capacity is the sum of capacity of stations passed by the bus line corresponding to the original edges before merging. 5. The capacity of $edge\{A', A''\}$ is the total number of passengers, which is exactly n.

Then we run Ford-Fulkerson algorithm on the reconstructed graph to get the maximum number of people.

Problem 1 Simulation Results

As for the simulation part, I assume that there are 350 passengers and there are 10 buses. Each bus will go through 3 to 5 stations and all 30 stations will be included. The capacity of each bus is set to 35, and the destination of each passenger is randomly generated. In this simulation we get the maximum capacity of 122.

Problem 2a

Lemma 2 (Worst case if no bound on minimum capacity)**.** *If there is no bound on the minimum capacity, then no online algorithm can perform better than* $\frac{1}{2}$*-competitively.*

Proof. Consider a fleet of m buses all of which have the same capacity 1. They all go through some common vertex v_0 and then each proceed to a unique endpoint vertex, denoted $v_1, ..., v_m$ (see Fig. 6).Suppose the algorithm receives $\frac{m}{2}$ orders destined to to v_0; after they are assigned (it doesn't matter how) there are $\frac{m}{2}$

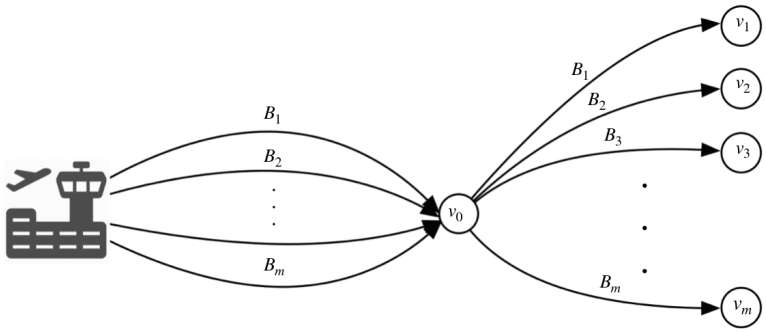

Figure 6: In the example, every bus goes through the same central node, then continues to some unique endpoint (edges returning to the airport are omitted for clarity).

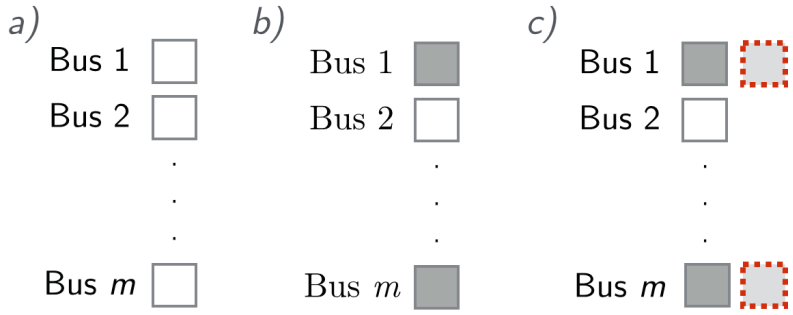

Figure 7: *(a)* Each bus has a capacity of 1; *(b & c)* After $\frac{m}{2}$ assignments, $\frac{m}{2}$ of the vertices are unreachable and a 'malignant god' can assign the remaining items so that they are impossible to allocate

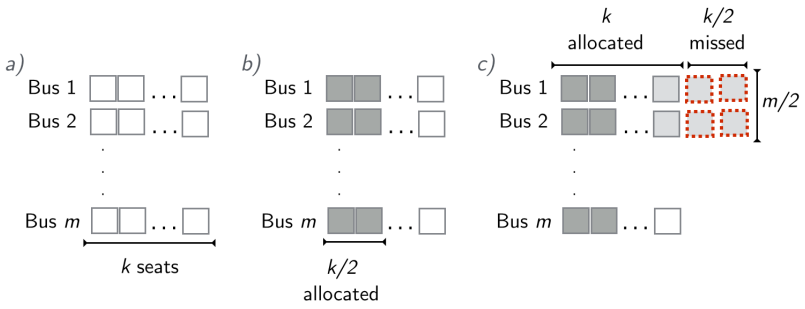

Figure 8: *(a)* Each bus has a capacity of k; *(b & c)* After $\frac{mk}{2}$ assignments, $\frac{m}{2}$ of the vertices are have remaining capacity of no more than $\frac{k}{2}$ each. A 'malignant god' can assign the remaining items so that there are $\frac{mk}{4}$ misses by assigning $fracm2$ orders of capacity k that are uniquely reachable by buses that are half full.

vertices which are now impossible to reach, hence an adversarial entity can create $\frac{m}{2}$ passengers destined to the unreachable vertices which will be missed.

This is illustrated in Fig. 7. □

Lemma 3 (Worst case for any minimum capacity)**.** *A trivial worst case competitive ratio for any online algorithm for this task is* $\alpha = \frac{3}{4}$

Proof. Consider the same example as in the previous lemma, but with arbitrary minimum capacity k. Suppose the first $\frac{mk}{2}$ passengers are all destined to v_0 and they are allocated uniformly across all buses. Then the minimum remaining seats for any bus is $\frac{k}{2}$. An adversarial entity can still send $\frac{m}{k}2$ passengers, suppose each k of them are now destined to vertices only reachable by one bus (in other words, not v_0). Then, there will be $fracm2$ such orders, each of size k, but only $frack2$ can be allocated per order. Hence there will be $\frac{mk}{4}$ misses in total, so $\frac{\frac{3}{4}mk}{mk} = \frac{3}{4}$ is the upper bound on competitiveness for any online algorithm.

This is illustrated in Fig 8. □

Remark. *In this example, if the initial allocation were not uniform (i.e. some buses were allocated more passengers than others) then the bound would be worse!*

Problem 2b

Lemma 4 (Competitiveness of MRSF)**.** *MRSF is* $\frac{1}{2}$*-competititve for any minimum bus capacity.*

Proof. If the minimum capacity is 1, then we can simply use the lemma from part (a).

For a larger minimum capacity, let us continue using the graph from Fig. 6. Denote the minimum capacity as k, and let there be $m > k$ buses, all of them with capacity k except for the first one, which has capacity mk (this is illustrated in Fig. 9a). Dispatch $(m-1)k$ orders, each with destination v. By construction they will be accepted by bus 1, since it has the most remaining capacity. Then dispatch mk orders with destination v'. Only k can be accepted, and $(m-1)k$ will be missed (see Fig. 9b). Hence:

$$\alpha = \frac{mk}{(2m-1)k} \geq \frac{1}{2}$$

□

Problem 2c

Let us follow the same line of thought as in the previous lemma and try to generalize. Recall that the maximum capacity in the proof example was $2mk$. Consider the allocation of the first $(m-1)k$ passengers. If they are allocated in Bus 1, then we are in the same situation as in the previous lemma (see Fig. 9b). If they are distributed among the remaining $m-1$ buses then an adversarial entity can construct the remaining $(m-1)k$ orders to be in those buses which are now blocked (see Fig. 9c). However if we allocate $\frac{(m-1)k}{2}$ passengers to bus 1 and $\frac{(m-1)k}{2}$ to the remaining buses then no adversarial entity can dispatch new orders that can make the algorithm perform worse than $\frac{3}{4}$-competitively:

Consider the next $(m-1)k$ orders. If they are assigned to Bus 1, then it is at most $\frac{1}{2}$ full, which implies there will be $\frac{(m-1)k}{2}$ misses (see Fig. 9f). Hence $\alpha = \frac{\frac{3}{2}(m-1)k}{2(m-1)k} = \frac{3}{4}$. If they are assigned to locations in the remaining buses then we know by the lemma in part a that this will also result in $\frac{3}{4}$-competitiveness.

These cases are illustrated in Fig. 9.

How could the examples in the last two lemmas be allocated optimally? In Lemma 2, we would like all the initial orders to be allocated to buses other than 1, since their only destination is v, however in Lemma 3, we saw that if we allow small buses to have unique final destinations then this would be a $\frac{1}{2}$ strategy.

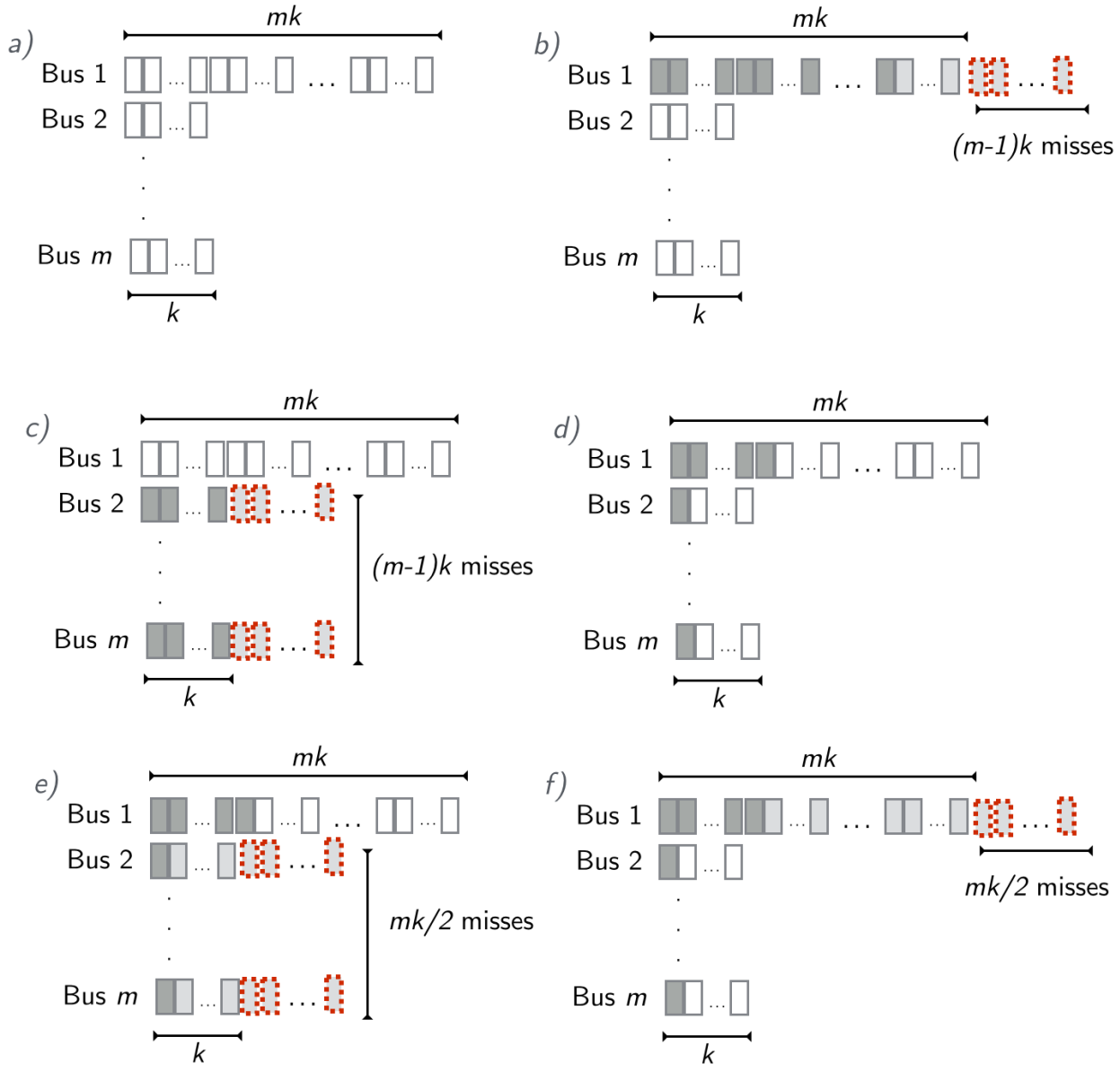

Figure 9: *(a)* The first bus has a capacity m times greater than the other buses; *(b & c)* If an online algorithm assigns the first half of the passengers to either the larger bus or the smaller buses then an online algorithm cannot perform better than $\frac{1}{2}$-competitively *(d, e & f)* The best an online algorithm can do in this scenario is to distribute passengers equally between the large bus and the smaller buses, resulting in half the misses as in the previous examples ($\frac{3}{4}$-competitive)

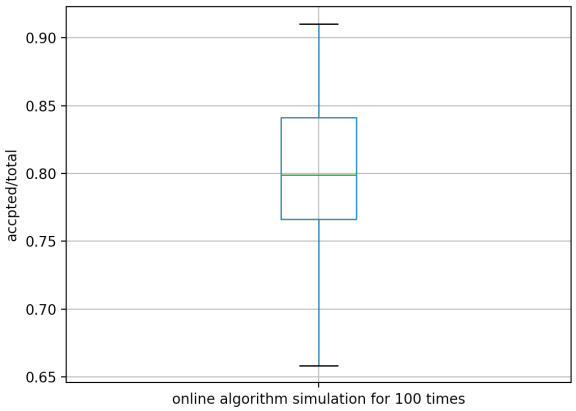

Figure 10: 100 time simulations for the online algorithm of Problem 2c

One way to realize $\frac{3}{4}$-competitiveness is through randomization. Suppose we assign each order j according to a discrete distribution where

$$P(\text{Bus } j \text{ is assigned order } i) = \frac{\text{remaining seats in } B_j}{\text{total remaining seats for } d_i}$$

Then the expected allocation for the previous example would be optimal. This is realized in Algorithm 3.

Algorithm 3 Online allocation

function OL-ALLOCATION((t_i, d_i): Passenger order)
  $\Pi \leftarrow \varnothing$
  $\Omega \leftarrow 0$
  for $B_j \in \mathcal{B}$ **do**
    if $(d_i \in S_j) \wedge (t_i \geq t_j)$ **then**
      Π.append$(B_j : \tilde{K_j})$ $\tilde{K_j}$ *denotes remaining seats in* B_j
      $\Omega \leftarrow \Omega + \tilde{K_j}$
  if $\Omega = 0$ **then**
    return nil
  else
    $\Delta \leftarrow$ makeDiscreteDistribution$(Pr(B_j) = \frac{\pi_j}{\Omega}$ for π_j in $\Pi)$
    $B_{\text{choice}} \leftarrow \Delta$.randomDraw()
    return B_{choice}

Please note that, as with all Monte Carlo algorithms, this bound is on the expected value. # Task 3

Task 2 Simulation Results

For the simulation, several buses were filled with orders (optimally), and then the passenger orders were randomly passed to the online algorithm. The performance of the online algorithm was divided by the cost of the optimal solution and recorded. This procedure was repeated 100 times. The performance was found to be under the $\frac{3}{4}$ for around 80% of the experiments, although there were a few outliers (this is to be expected in a randomized algorithm).

Task 3

Problem 1

Our Bus Allocation with Priorities problem is essentially a bin packing or multiple knapsack problem with the added complexity that our 'bins' (buses) have some additional feasibility constraints on the 'items' (passengers) that they can carry (namely, the destination must be in the itinerary). Hence, it makes the most sense to show polynomial reducibility from Multiple Knapsack:

Definition (Multiple Knapsack Problem)**.** *Given a set of items S, each with a weight and a value (denoted w_i, v_i respectively), find a collection $\mathcal{C}$ of m pairwise disjoints subsets $S_i \subseteq S$ such that the total weight is bounded by a constant (a different one for each subset) $\forall S_i.(\sum_{j \in S_i} w_j \leq W_i)$ and the total value $\sum_{S_i \in \mathcal{C}} \sum_{j \in S_i} v_{ij}$ is as large as possible.*

Lemma 5. *BUS-ALLOCATION-WITH-PRIORITY is NP-Hard*

Proof (by gadget). We will proceed by MULTIPLE-KNAPSACK $\leq_P$ BUS-ALLOCATION-WITH-PRIORITY, since Multiple Knapsack has Knapsack as a special case, which is known to be NP-Hard.

Suppose we are given a multiple knapsack instance and we have an oracle $\mathcal{O}_B$ that can solve the Bus Allocation with Priority problem. Then we can construct an instance of the bus allocation problem like this:

For each knapsack: create a bus with destination v and capacity equivalent to the weight constraint on the knapsack.

For each item: construct an order with passengers equal to the weight and the item density (that is $\frac{v_i}{w_i}$) as the priority.

Then the solution to that bus-allocation optimization instance is equivalent to the M-Knapsack optimization problem. The proof is by structural induction:

- No bus can exceed the capacity of a knapsack because of the bus capacity constraints, so allocations will be feasible solutions for the Knapsack problem
- Passengers must be allocated together, and the objective function will scale the priority by the amount of passengers per order, so when a passenger order is allocated it will take up the same space as in the knapsack case and count the same towards the objective function.

□

Problem 2

This is the same problem as 2c with a priority and a size constraint on the passengers. We will build on the intuition from Kosuch, Letournel, and Lisser (2017) on Stochastic Multiple Knapsack Problems:

First, let us discuss the passenger size constraint c_i. We have the following lemma:

Lemma 6 (No competitive online algorithm exists for passenger orders with unbounded c_i)**.** *Suppose passenger orders are of the form (t_i, c_i, d_i, p_i) and there is no restriction on the maximum value c_i can take. Then the competitive ratio for any online algorithm can be arbitrarily bad.*

Proof. Consider our earlier examples from lemmas 2 and 3 where each bus had a unique vertex v_j that was only reachable in its route. In the example from Lemma 3, let the first passenger order be $(0, 1, v_1, p_{max})$ and the second one be $(0, mk, v_1, p_{max})$. The first order will be assigned to B_1, because it is of maximum priority; and the second one will not because there is no space left. Hence, the competitive ratio is $\frac{1}{mk}$, which is the worst theoretical ratio considering $n = mk + 1$. Since we can choose both m and k, the algorithm can be made to perform arbitrarily badly. □

Remark. *Suppose c_{max} is the maximum size of any passenger order, and $\forall B_j \in \mathcal{B}.(K_j \gg c_{max})$. Then each passenger order can take up at most $\frac{c_{max}}{K_{min}}$ of the bus, which is a small proportion. This implies that the randomized algorithm from 2c can still be used with the same asymptotic bound of $\frac{3}{4}$-competitiveness.*

Now, let us address the priority constraint:

Lemma 7 (No competitive online algorithm exists for passenger orders with unbounded p_{max})**.** *Let p_{min} and p_{max} denote the minimum and maximum priority. If p_{max} can be arbitrarily large, then an online algorithm can be made to perform arbitrarily badly.*

Proof. The proof is similar to that of the previous lemma. An adversarial entity sends passenger orders. When the first order is assigned to bus B_j, the next order is destined to v_j, so that it is uniquely reachable by B_j, but has arbitrarily large priority and size smaller than the original bus capacity but larger than the current remaining capacity. Since the order was feasible and of the highest priority, it would have been in the optimal solution, but it cannot be picked by the online algorithm. Since p_i can be arbitrarily large, the competitive ratio can be arbitrarily bad. □

In view of the previous two lemmas, we will propose a solution with a competitive ratio of $\frac{3}{4(\ln \frac{p_{max}}{p_{min}}+1)}$, under the assumption that p_{min}, p_{max} are fixed and known by the algorithm and the minimum allowed bus capacity is much greater than the maximum passenger order size: $\forall B_j \in \mathcal{B}.(K_j \gg c_{max})$.

Under these assumptions, we build up to the final algorithm by first solving the problem of allocating passengers to a *single* bus with a $\frac{1}{\ln \frac{p_{max}}{p_{min}}+1}$-competitive algorithm (this result will make the generalization to multiple buses quite straightforward). The algorithm itself is very simple, inspired by Zhou, Chakrabarty, and Lukose (2008), it uses a monotonically increasing function $\Psi(z)$ to prevent overfilling:

Algorithm 4 Online allocation with priority algorithm for a single bus

```
function OL-Single-Priority-Allocation((t_i, c_i, d_i, p_i): Passenger order)
    Ψ(z) := (e p_max / p_min)^z (p_min / e)
    if (d_i ∈ S) ∧ (remaining capacity of B ≥ c_i) ∧ (t ≥ t_i) then
        z ← (remaining capacity of B) / (total capacity of B)
        if p_i ≥ Ψ(z) then
            return Accept passenger
    return Reject passenger
```

Theorem 1 (α-competitiveness of Algorithm 4)**.** *Algorithm 4 is $\frac{1}{\ln \frac{p_{max}}{p_{min}}+1}$-competitive*

Proof. Let us start by introducing some notation:

σ	$\triangleq$	Some sequence of passenger orders
S^*, S	$\triangleq$	Optimal solution set with allocated orders, respective set produced by algorithm
$\text{OPT}(\sigma)$	$\triangleq$	Optimal solution (value of the objective function): $\sum_{i \in S^*} c_i p_i$
$\mathcal{A}(\sigma)$	$\triangleq$	Algorithm solution (value of the objective function): $\sum_{i \in S} c_i p_i$
C	$\triangleq$	Size of orders in both optimal and algorithm solution: $\sum_{i \in S \cap S^*} c_i$
P	$\triangleq$	Priority of orders in both optimal and algorithm solution: $\sum_{i \in S \cap S^*} c_i p_i$
Z	$\triangleq$	Proportion of seats empty in bus at the end of the sequence.

Now we are going to define some very basic bounds:

For any order not picked by the algorithm, we must have that:

$$\forall i \in \sigma - S.(p_i \leq \Psi(z_i) \leq \Psi(Z)) \tag{1}$$

This implies that:

$$OPT(\sigma) \leq P + \Psi(Z)(K - C) \tag{2}$$

Now, let us use the abbreviation: $p(S - S^*) \triangleq \sum_{i \in S - S*} c_i p_i$, then we also have:

$$\mathcal{A}(\sigma) = P + p(S - S^*) \implies \frac{OPT(\sigma)}{A(\sigma)} \leq \frac{P + \Psi(Z)(K - C)}{P + p(S - S^*)} \tag{3}$$

Now, because of the way the algorithm selects orders we can find two more bounds: $P \leq \sum_{i \in S \cap S^*} \Psi(z_i) c_i$. Also $A(\sigma) \geq \sum_{i \in S} \Psi(z_i) c_i$ because $\Psi(z_i) \geq \frac{p_i}{c_i}$. We can update our competitive ratio:

$$\frac{OPT(\sigma)}{A(\sigma)} \leq \frac{\sum_{i \in S \cap S^*} \Psi(z_i) c_i + \Psi(Z)(K - C)}{\sum_{i \in S} \Psi(z_i) c_i} \leq \frac{\Psi(Z) K}{\sum_{i \in S} \Psi(z_i) c_i} = \frac{\Psi(Z)}{\sum_{i \in S} \Psi(z_i) \frac{c_i}{K}} \tag{4}$$

Now, by our assumption that $K \gg c_i$, we can approximate the bound through integration:

$$\sum_{i \in S} \Psi(z_i) \frac{c_i}{K} \approx \int_0^Z \Psi(z) dz = \int_0^Z (\frac{e p_{max}}{p_{min}})^z \frac{p_{min}}{e} dz = \frac{p_{min}}{e} \left[\frac{(\frac{e p_{max}}{p_{min}})^z}{\ln \frac{e p_{max}}{p_{min}}} \right]_0^Z \tag{5}$$

$$\approx \frac{p_{min}}{e} \frac{(\frac{e p_{max}}{p_{min}})^Z}{\ln \frac{p_{max}}{p_{min}} + 1} = \frac{\Psi(Z)}{\ln \frac{p_{max}}{p_{min}} + 1} \implies \frac{OPT(\sigma)}{A(\sigma)} \leq \frac{\Psi(Z)}{\frac{\Psi(Z)}{\ln \frac{p_{max}}{p_{min}} + 1}} = \ln \frac{p_{max}}{p_{min}} + 1 \tag{6}$$

□

Remark. *Since this is a bounded online knapsack problem,* $\frac{1}{\ln \frac{p_{max}}{p_{min}} + 1}$ *is the upper bound on competitiveness for this problem (Zhou et al. (2008)), so no other online algorithm can achieve a better bound.*

We now generalize to multiple buses by using this result and our result from Task 2c:

Algorithm 5 Online allocation with priority algorithm

```
function OL-Priority-Allocation((t_i, c_i, d_i, p_i): Passenger order)
    ψ(z) := (Ue/L)^z (L/e)
    B' ← ∅
    for j ← 1 to m do
        if (d_i ∈ S_j) ∧ (remaining capacity of B_j ≥ c_i) ∧ (t_j ≥ t_i) then
            z_j ← (remaining capacity of B_j) / (total capacity of B_j)
            if p_i ≥ ψ(z_j) then
                B' ∪ {B_j}
    if B' ≠ ∅ then
        return OL-Allocation(d_i, c_i, B')
    else
        return nil
```

Lemma 8 (α-competitiveness of Algorithm 5)**.** *Algorithm 5 is* $\frac{3}{4(\ln \frac{p_{max}}{p_{min}} + 1)}$*-competitive*

Proof. The proof is a simple product of bounds. By the bound for the single bus case, we know that deciding which buses are feasible is done in a $\frac{1}{\ln \frac{p_{max}}{p_{min}} + 1}$-competitive way, whereas allocating passengers to an actual bus is $\frac{3}{4}$-competitve by the remark to Lemma 6. As with all Monte Carlo algorithms, this bound is on the expected value. □

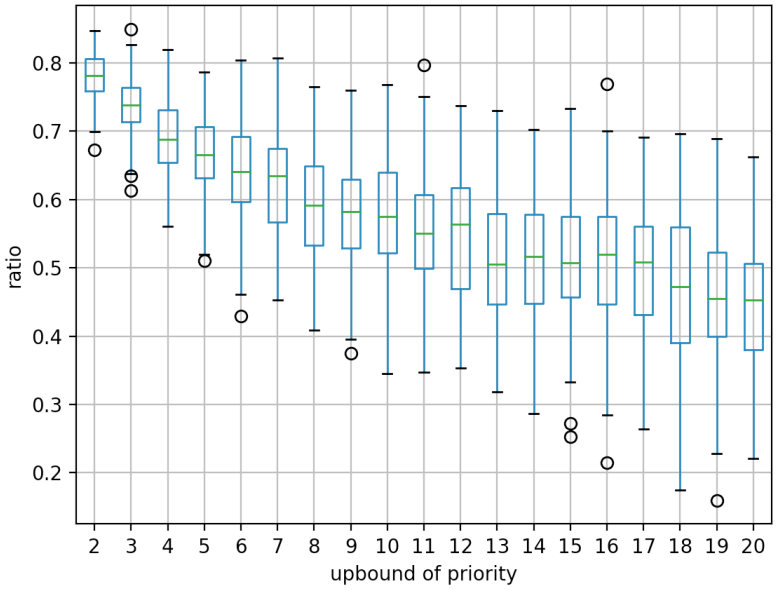

Figure 11: Box graph for the simulation

Task 3 Simulation Results

The result of the simulation of the online algorithm for task 3 is shown in Fig. 11. The lower bound on the minimum priority is fixed to $p_{max} = 1$, whereas p_{max} is allowed to vary from 2 to 20. With the exception of a few outliers, most results are under the bound from the previous lemma and the expectation is far better than the bound.

References

Brecklinghaus, Judith, and Stefan Hougardy. 2015. "The Approximation Ratio of the Greedy Algorithm for the Metric Traveling Salesman Problem." *Operations Research Letters* 43 (3): 259–61.

Clarke, Geoff, and John W Wright. 1964. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points." *Operations Research* 12 (4): 568–81.

Feo, Thomas A, and Mauricio GC Resende. 1995. "Greedy Randomized Adaptive Search Procedures." *Journal of Global Optimization* 6 (2): 109–33.

Golden, Bruce, S. Raghavan, and Edward Wasil, eds. 2008. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-77778-8_4.

Kosuch, Stefanie, Marc Letournel, and Abdel Lisser. 2017. "Stochastic Knapsack Problem: Application to Transportation Problems." *Pesquisa Operacional* 37 (3): 597–613.

Sörensen, Kenneth, Florian Arnold, and Daniel Palhazi Cuervo. 2019. "A Critical Analysis of the 'Improved Clarke and Wright Savings Algorithm'." *International Transactions in Operational Research* 26 (1): 54–63.

Toth, Paolo, and Daniele Vigo, eds. 2001. *The Vehicle Routing Problem*. Philadelphia, PA, USA: Society for Industrial; Applied Mathematics.

Yaman, Hande. 2006. "Formulations and Valid Inequalities for the Heterogeneous Vehicle Routing Problem." *Mathematical Programming* 106 (2): 365–90.

Zhou, Yunhong, Deeparnab Chakrabarty, and Rajan Lukose. 2008. "Online Knapsack Problems." In *International Workshop on Internet and Network Economics*, 566–76. Springer.