1 Motivation

2 Mathematical Backgrounds

The Airplane Refueling Problem is combinatorial problem that is neither known to be NP-complete nor polynomially solvable.

Definition 2.1. *Given n planes each with a tank w_j and consumption rate p_j, what is the longest time/distance that this fleet of planes can fly, if they can refuel each other?*

An intuitive idea is consuming the tank of only one plane at a time, and drop it out after its tank is empty. This approach finds a *drop out ordering* σ that maximizes the objective function

$$\max \sum_{j=1}^{n} \frac{w_{\sigma(j)}}{\sum_{k=j}^{n} p_{\sigma(k)}}$$

The problem can also be formalized as a scheduling problem, proposed by Woeginger. In context of scheduling, σ can be viewed as permutation, the reverse of ordering, while we can also change the objective function to a simpler form.

$$\sum_{j=1}^{n} \frac{w_{\pi(j)}}{\sum_{k=1}^{j} p_{\pi(k)}} = \sum_{j=1}^{n} \frac{w_j}{C_j}$$

On the basis of this, we are also able to introduce a mathematical programming formulation.

$$\max \sum_{j=1}^{n} \frac{w_j}{C_j} \tag{A}$$
$$\text{s. t. } C_{\pi(j+1)} - C_{\pi(j)} = p_{\pi(j)} \forall j$$

As we know, a linear program or a convex semi-definite program is solvable using various interior-point methods. Since the mathematical program is neither linear nor convex semi-definite, our goal is to find a reduction the realizes this goal. On the otherhand, we are also interested in formulate the problem as an integer program, for it also provides us with some other possible solution approaches.

3 First Attempt in Theory

Our first attempt focused on the fact that the solution space of the formulation A is a polytope based on permutations. Each permutation generates a tuple of $(1/C_1, 1/C_2, ..., 1/C_n)$. Since w_j's are given constants, a straightforward solution is simply traversing all $n!$ permutations.

3.1 Permutahedron

A permutahedron is a polytope that is defined by permutations. In the permutahedron of the k-th order P_k, each vertex is a permutation of k elements. Hence there are $k!$ many vertices in P_k. For example, a hexagon is a second-order permutahedron.
Although the number of faces is not polynomial in n, some symmetric property was discovered, and can be used for optimization. Some researchers(cite later) have found an extended formulation $\mathbb{R}^{k^2+2k} \to \mathbb{R}^k$, which allows us to solve the LP on P_k in polynomial time.

3.2 Details in attempts

Elaborate formulation later: the symmetric property does not exists in our polytope. Using the similar formulation, we end up with a semi-definite program that has a convex feasible region, but concave objective function. We did not go further on this direction. However, two possiblility remains to be explored:

- Can we find an extended formulation by changing the formulation for permutahedron slight?
- Can we try some classical methods like interior-point method on this formulation, what is the result?

4 Second Attempt by Computation

After this attemtp in theory, we decided to move to do some computation work. For simplicity, python was chosen to generate some randomized inputs and scripts for the computation tool, polymake.

4.1 Input Generation

As we have analyzed in the first attempt, we want to compute the polytope of the all solution vectors

$$s = (\frac{1}{C_1}, \frac{1}{C_2}, ..., \frac{1}{C_n})$$

Since the completion time is based on the permutation π, we need a few steps to generate all possible vectors.

1. Generate a randomized/special consumption time p_j
2. Pick a permutation π, arrange the consumption time in the order of π
3. Compute the completion time by $C_1 = p_1$ and $C_{i+1} = C_i + p_{i+1}$
4. Compute the inverse of C_j
5. Repeat until all permutations are visited

This results in $n!$ vertices in $\mathbb{R}^n$. A straightforward traveral on the vertices is absolutely not polynomial-time, but it may help us to find some polynomial-time structure within the polytope.

4.2 Usage of Polymake

4.3 Results

Although the computation of the polytope is polynomial-time in the number of vertices, our computation takes a significant amount of time. The reason is that the number of vertices is factorial in n. The computation of a 7-dimensional polytope does not terminate within 24 hours. Thus, we establish a large amount of computation only in $n = 4$ and $n = 5$. Some computation of $n = 6$ was also done, but a large amount of computation is not applicable. Starting from $n = 4$, the number of facets is no longer a unique number. Hence we guess that the number of facets is somehow related to the input chosen. Unfornately, a direct pattern cannot be observer for $n = 5$. Thus, we try to use machine learning to help us find some pattern.

n	Number of Facets	Number of Unique Values
2	2	1
3	8	1
4	67, 68, 69, 70	4
5	$700 \sim 800$	≥ 66
6	$10000 \sim 12000$	Unknown

Table 1: Number of facets in the polytope for $n \leq 6$

5 Third Attempt by Machine Learning

In this section, we would propose some machine learning techniques to see if we can find some property that either justifies the our polynomial-time prediction or gives some analytical description of the exponential-time behavior. Fortunately, the problem is already geometric, simple unsupervised learning techniques like clustering can tell us some direct information. In addition, the behavior of the polytope is also related to the input chosen as consumption rate. Supervised learning techniques like decision tree, may help us to find a pattern that either helps us decide the behavior or a subclass of problems that is solvable in polynomial-time.

5.1 Regression

5.2 Clustering

5.3 Decision Trees

6 Progress during the vacation

To be discussed in the meeting by the start of November

6.1 Machine Learning

Tree models worked pretty well. The visualization results seem to be interesting. Would be interesting to employ some quadratic classifier. Default quadratic discrimamt analysis do not work well
Discovery in data set: $x_i - x_j = 0$ separates all vertices by a half for all i, j.

6.2 Formulation with Permutahedron

$$
\begin{aligned}
&\max w^T x \text{ s.t.} \\
&\quad x_i y_i = 1 \ \forall i \\
&\sum_{i \in [n]} z_{ij} = 1 \forall j \\
&\sum_{j \in [n]} z_{ij} = 1 \forall i \\
&\quad \Pi = (z_{ij}), S = \begin{pmatrix} 1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 1 & \cdots & 1 \end{pmatrix} \\
&\quad y = \Pi^T S \Pi p
\end{aligned}
$$

This is a convex program, if $0 \leq z_{ij} \leq 1$. Apparently, computing this program in polynomial time gives us an approximation in some form. Is it worth doing so?

6.3 Formulation with Vasquez's Idea

Local and Global precedence can be computed upon getting input in polynomial time. For the simple case, where the global precedence can be decided directly, we don't need to take a second look. However, for the last case, where the global precedence is decided by the t^* a binary selection is need. Based on this idea, we can also formulate a convex program.
In this formulation, we assume all global precedence cannot be decided. z_{ij} stands for the precedence. If $z_{ij} = 1$ then $i \prec_g j$ and vice versa.

$$
\begin{aligned}
&\max \sum w^T x \text{ s.t.} \\
&x_i y_i = 1 \\
&(y_i - t^*_{ij} - p_i) z_{ij} \leq 0, z_{ij} \in \{-1, 1\} \\
&2y_i = \{2p_i\} + \sum_{j \neq i} p_j (1 + z_{ij})
\end{aligned}
$$

Again this is a discrete problem. Notice that z_{ij} can be transformed into the sign function of $t^*_{ij} - y_i$. Here is what we do

$$\begin{aligned}
&\max \sum w^T x \text{ s.t.} \\
&x_i y_i = 1 \\
&z_{ij} = y_i - t^*_{ij} - p_i \forall i < j \\
&z_{ij} = -z_{ji} \forall i > j \\
&2y_i = \{2p_i\} + \sum_{j \neq i} p_j (1 + \frac{z_{ij}}{|z_{ij}|})
\end{aligned}$$

We are able to simplify the last constraint even a bit more by removing y's and z's

$$\begin{aligned}
&\max \sum w^T x \text{ s.t.} \\
&(\frac{1}{x_i} - t^*_{ij} - p_i)(\frac{1}{x_j} - t^*_{ij} - p_j) \leq 0 \forall i < j \\
&\frac{2}{x_i} = \{2p_i\} + \sum_{j \neq i} p_j \big(1 + \frac{\frac{1}{x_j} - t^*_{ij} - p_j}{|\frac{1}{x_j} - t^*_{ij} - p_j|}\big)
\end{aligned}$$

It is easy to see that the second constraints are polynomial functions w.r.t. x_i only. And there n solutions for each of x. More importantly, the first quadratic constraints can be replaced with permutahedra again.

6.4 A final formulation?

$$\frac{2}{x_i} = \{2p_i\} + \sum_{j \neq i} p_j \big(1 + \frac{\frac{1}{x_j} - t^*_{ij} - p_j}{|\frac{1}{x_j} - t^*_{ij} - p_j|}\big)$$

Suppose we solve all of these equations to obtain all n solutions for each x_i. If arranged in decreasing order, they will correspond to their position in the permutation. This where we can employ the formulation of permutahedra again.

For the j-th solution of x_j in decreasing order, we donote it as x_{ij}

$$\begin{aligned}
&\max \sum w^T x \text{ s.t.} \\
&0 \leq z_{ij} \leq 1 \forall i, j \\
&\sum_{i \in [n]} z_{ij} \leq 1 \forall j \\
&\sum_{j \in [n]} z_{ij} \leq 1 \forall i \\
&x_i = \sum_{j \in [n]} x_{ij} z_{ij}
\end{aligned}$$

This is a linear program, and can be solved in polynomial time. Recall that x_{ij} are n^2 constant values obtained in polynomial time. If this formulation is indeed correct, we then over. Thus, we now need a computational proof for it.