The Buyer Seller Problem

James McDonagh
Lakehead University
Thunder Bay, ON, Canada
jwmcdona@lakeheadu.ca

***Abstract*—This report explores practical approaches to solving combinatorial optimization problems with a focus on buyer-seller relationships. We first analyze classical optimization techniques and introduce a new method that prioritizes speed over accuracy. We then propose enhancements through the integration of Reinforcement Learning and Quantum Computing, specifically targeting improvements in runtime. Our results indicate that while significant progress is still needed, the integration of advanced techniques shows promise. This study underscores the potential of these methods to refine optimization strategies and lays out a framework for future research.**

***Index Terms*—Combinatorial Optimization, Weighted Bipartite Graphs, Hungarian Algorithm, Integer Linear Programming, Reinforcement Learning, Quantum Computing**

I. Introduction

Combinatorial Optimization is a field of mathematical optimization that seeks to identify the most effective solution from a discrete set of potential configurations or arrangements. This is incredibly helpful in many fields, as a common goal is often to find the most optimal solution for a given problem. The downside to such a powerful tool is time complexity. While many implementations can perform very well on certain sets of data, the worst case scenario can still result in an exponential time complexity. Our goal in this paper is to address these computational bottlenecks by introducing a technique aimed at improving both performance and accuracy. Specifically, we target the optimization processes in the trading sector, focusing on buyer-seller interactions, where efficient resource distribution is key. Through a combination of the Hungarian Algorithm and Integer Linear Programming, we seek to provide solutions for problems that would otherwise suffer from high computational costs. We provided a functional demo, made in Python, that showcases our work. Additionally, we propose leveraging Reinforcement Learning to further enhance the decision-making process, reducing reliance on traditional optimization techniques. Lastly, we explore the potential of Quantum Computing as a means to dramatically accelerate these computations.

The rest of this paper is organized as follows. Section II describes the fundamentals of a trading network and how we use the Hungarian Algorithm, as well as Linear Integer Programming, to solve basic examples of the Buyer Seller Problem. Section III begins to explore the concept of using Reinforcement Learning to achieve the same objective. Section IV details how Quantum Computing can further improve our technique. Finally section V concludes our thoughts and elaborates on what future work could be.

II. Trading Networks

To begin, we can represent a trade between two entities as two vertices with an edge between them. The edge representing a relationship between its two endpoints. Entities can be thought of as the individuals or companies that are buyer or selling some arbitrary product. The buyer vertices will simply contain the amount of money available for the buyer to spend, referred to as the buyer's total. The seller vertices will contain the number of units of said product the seller has to sell, the price per unit, and the total amount the seller will make if everything is sold, similarly referred to as the seller's total. These two types of vertices will be called BNodes and SNodes respectively, and are both defined as classes in our Python program.

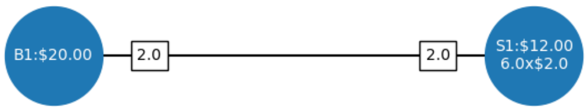

Fig. 1. Basic Trade

In figure 1, the left vertex represents a buyer who has $20 to spend on the product, and the right vertex represents a seller with 2 units of said product that they are selling for $6 per unit, totalling to $12. The edge that connects the two vertices has weight equal to the price per unit of the seller. These graphics are automatically made in our program based on the text within the input.txt file. We plan on creating a more convenient method for inputting test cases at a later date.

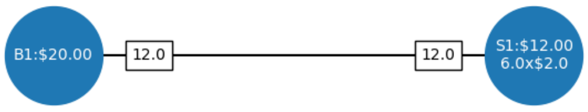

Fig. 2. Basic Brokering

In figure 2, the vertices stay the same, but this time the weight is the calculated by taking the buyer's total and dividing it by the price per unit of the seller, then flooring the resulting quotient. To ensure the buyer doesn't accidentally try to buy more units than the seller has, we take the minimum value between the previously calculated value and the seller's unit quantity. Finally we multiply the minimum by the seller's price per unit to determine the maximum amount of money a buyer can spend on a seller. We call this value the "brokered" value between a given buyer and seller:

$$
\begin{aligned}
b_n &: \text{buyer } n \\
s_m &: \text{seller } m \\
BR_{n,m} &: \text{brokered value of buyer } n \text{ and seller } m
\end{aligned}
$$

$$BR_{n,m} = \min\left(\left\lfloor \frac{b_{n_{total}}}{s_{m_{price}}} \right\rfloor, s_{m_{quantity}}\right) \times s_{m_{price}}$$

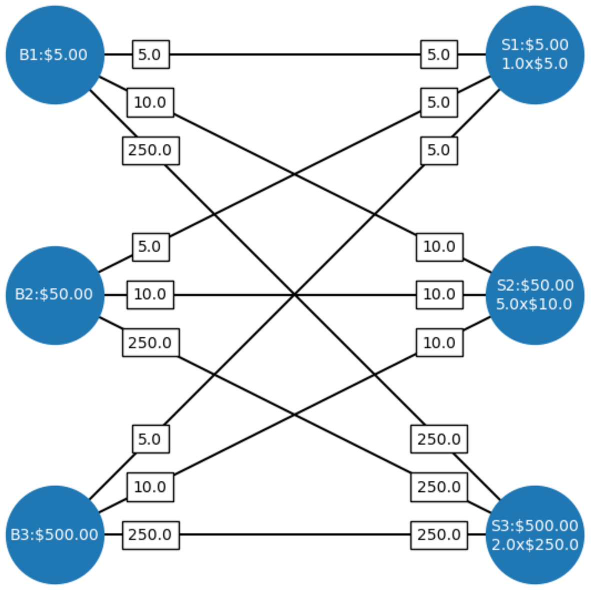

Fig. 3. Multiple Traders

In figure 3, we have a more complex trading structure with three buyers and three sellers. Like figure 1, the edges represent the price per unit of the seller.

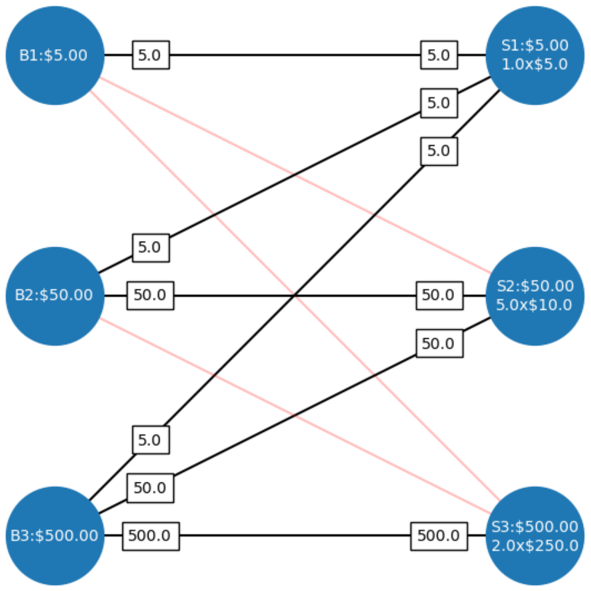

Fig. 4. Multiple Brokers

In figure 4, we have a more complex brokering structure. Notice that edges where the seller's price per unit is greater than the buyer's total are red because the buyer would not be able to afford a transaction. A red edge will indicate that a relationship is not possible.

A. Purpose

Say we want to figure out the optimal arrangement of edges such that the trading network minimizes the amount of leftover money after all of the trades are performed. We refer to this leftover value as the waste produced by a given solution.

We define waste to be any leftover buyer or seller totals. We can calculate the minimal amount of waste by taking the absolute value of the sum of all buyer totals and subtract the sum of all seller totals:

$$B_T : \text{Sum of all buyer totals}$$
$$S_T : \text{Sum of all seller totals}$$
$$W : \text{Waste}$$
$$W_{\min} : \text{Minimum waste}$$
$$W_{\min} = |B_T - S_T|$$

In figure 1 the minimal waste would be $|20 - 12| = 6$ and in figure 3 the minimal waste would be $|555 - 555| = 0$. Being able to verify the accuracy of our solution with such efficiency is one of the biggest strengths of our structure.

B. Fundamental Data Structure

Our visualizations reveal a structure known as a weighted bipartite graph, which effectively models our data by distinguishing buyers and sellers as two distinct sets of vertices. In cases where an entity functions as both a buyer and a seller, referred to as a prosumer, we assign it a vertex in each set and enforce a rule that prevents edges from forming between the corresponding vertices.

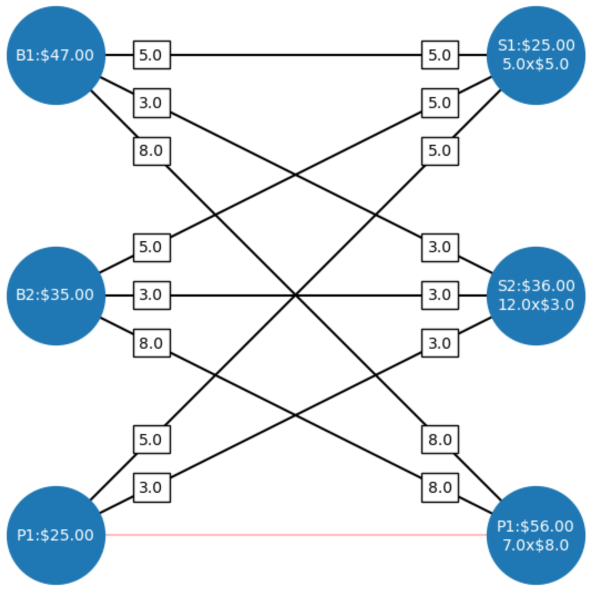

Fig. 5. Prosumer

In figure 5, we have an example of a trading matrix that includes a prosumer. There is a red colored edge between the vertices with the same label ”P1” to show that an entity cannot trade with themselves.

Given that our structure is a graph, we can also use an adjacency matrix to represent the graph's edges in a tabular format:

$$\text{Multiple Traders (see figure 3)} \rightarrow \begin{array}{c|ccc} & b_1 & b_2 & b_3 \\ \hline s_1 & 5 & 10 & 250 \\ s_2 & 5 & 10 & 250 \\ s_3 & 5 & 10 & 250 \end{array}$$

$$\text{Multiple Brokers (see figure 4)} \rightarrow \begin{array}{c|ccc} & b_1 & b_2 & b_3 \\ \hline s_1 & 5 & 0 & 0 \\ s_2 & 5 & 50 & 0 \\ s_3 & 5 & 50 & 250 \end{array}$$

These matrices will allow us to store and interact with the data so that we can perform algorithms on our graphs. As you can see, we are able to represent many different relationships for any given simulation by simply altering what our edges represent, and updating our matrices.

C. Hungarian Algorithm

Now we can begin trying to solve the problem. The idea of matching buyers to sellers strongly relates to the Assignment Problem. The goal of which is to assign agents to tasks in order to maximize efficiency or minimize cost. One popular approach to solving the assignment problem is the Hungarian Algorithm. The Hungarian Algorithm has a time complexity of $O(n^3)$ and is able to solve this matching problem through clever matrix manipulation.

Looking back at Figure 4, we have already constructed the necessary matrix for the Hungarian Algorithm to work. We want to apply the algorithm to the brokered edge matrix as our goal is to determine the optimal way to assign pairs, such that the waste leftover is minimal. Each edge in the matrix represents the most amount of money that can be exchanged between the given buyer and seller. Which means if we compute the sum of all of the edges in the sub-graph generated by the Hungarian algorithm, we can calculate the amount of waste produced.

To perform this calculation we need three values: the minimal waste, the sub-graph summation, and the minimum of the buyer total and seller total. The minimum of the buyer total and seller total represents the maximum summation we can produce from our sub-graph. This is because the only time our waste can be zero, is when the buyer total and seller total are zero. We subtract our sub-graph summation from this maximum, then add the minimum waste to get the waste produced:

$$W = W_{\min} + \left[\min(B_T, S_T) - \sum E\right]$$

One requirement of the Hungarian Algorithm is that there must be an equal number of vertices in each set. We can get around this by creating filler vertices that have a total of zero until the graph is balanced. Like our workaround for prosumers, we will enforce that no edges can form between our filler vertices and our actual vertices.

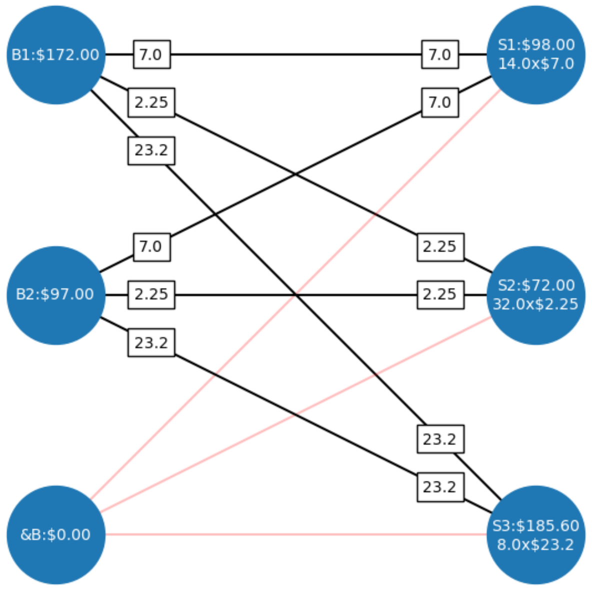

Fig. 6. Unbalanced Simulation

In figure 6, there are 2 buyers and 3 sellers. Every single edge that connects to the filler buyer vertex is red to demonstrate that none of the sellers can trade with it.

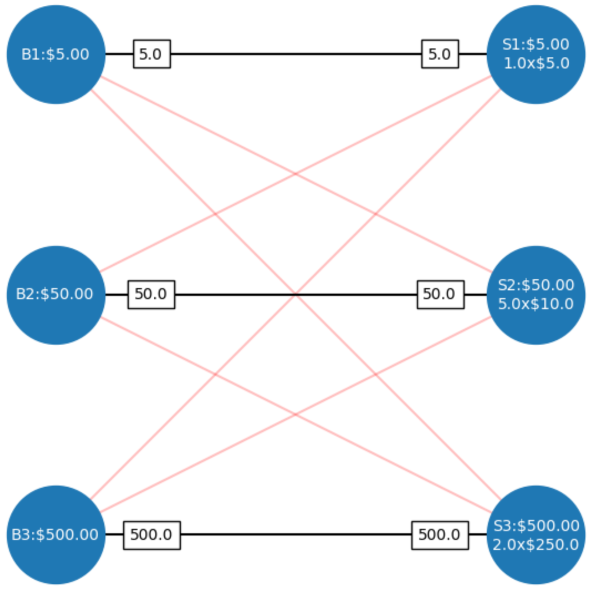

Fig. 7. Hungarian Solution Sub-graph

In figure 7, we can see the sub-graph that has been produced after using the Hungarian algorithm.

Sadly, the Hungarian algorithm doesn't provide us with a complete solution. To obtain the optimal solution for the majority of problems we need to check every possible arrangement of edges and weights to determine which one will lead to the ideal solution. To do this, we will use Integer Linear Programming.

D. Integer Linear Programming (ILP)

In all ILP problems, there is an objective function, decision variables, and contraints. In our case we have the following matrix and variables:

$$p_m : \text{seller } m\text{'s price per unit}$$
$$q_{n,m} : \text{quantity buyer } n \text{ is buying from seller } m$$

$$\begin{array}{c|ccccc}
 & s_1 & s_2 & s_3 & \cdots & s_m \\
\hline
b_1 & p_1 \cdot q_{1,1} & p_2 \cdot q_{1,2} & p_3 \cdot q_{1,3} & \cdots & p_m \cdot q_{1,m} \\
b_2 & p_1 \cdot q_{2,1} & p_2 \cdot q_{2,2} & p_3 \cdot q_{2,3} & \cdots & p_m \cdot q_{2,m} \\
b_3 & p_1 \cdot q_{3,1} & p_2 \cdot q_{3,2} & p_3 \cdot q_{3,3} & \cdots & p_m \cdot q_{3,m} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
b_n & p_1 \cdot q_{n,1} & p_2 \cdot q_{n,2} & p_3 \cdot q_{n,3} & \cdots & p_m \cdot q_{n,m}
\end{array}$$

Where p_m is the coefficient for the corresponding decision variable $q_{n,m}$.

- The Objective Function is the sum of each entry in the matrix.

$$\max z = p_1 \cdot q_{1,1} + \ldots + p_m \cdot q_{n,m}$$

- The sum of each row forms the inequality for the corresponding buyer's constraint equation.

$$p_1 \cdot q_{n,1} + p_2 \cdot q_{n,2} + p_3 \cdot q_{n,3} + \ldots + p_m \cdot q_{n,m} \leq b_{n_{total}}$$

- The sum of each column forms the inequality for the corresponding seller's constraint equation.

$$p_m \cdot q_{1,m} + p_m \cdot q_{2,m} + p_m \cdot q_{3,m} + \ldots + p_m \cdot q_{n,m} \leq s_{m_{total}}$$

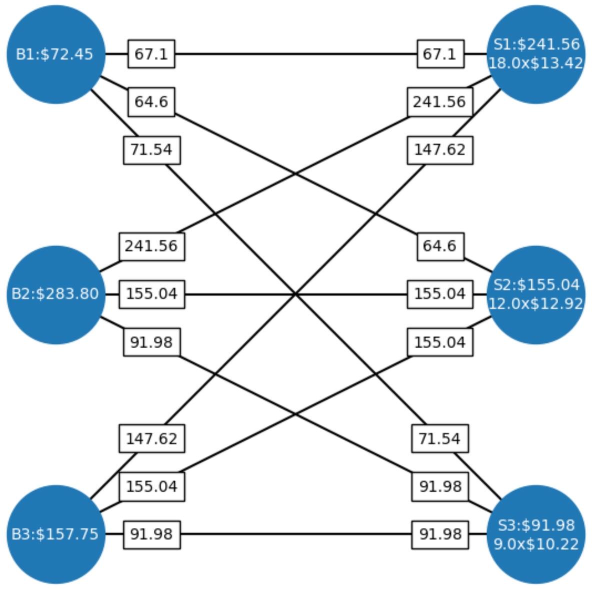

Fig. 8. ILP Example: Brokered Matrix

In figure 8, we have another trading simulation that involves three buyers and three sellers. We will apply the Hungarian algorithm as well as the ILP method and compare the sub-graphs that are produced.

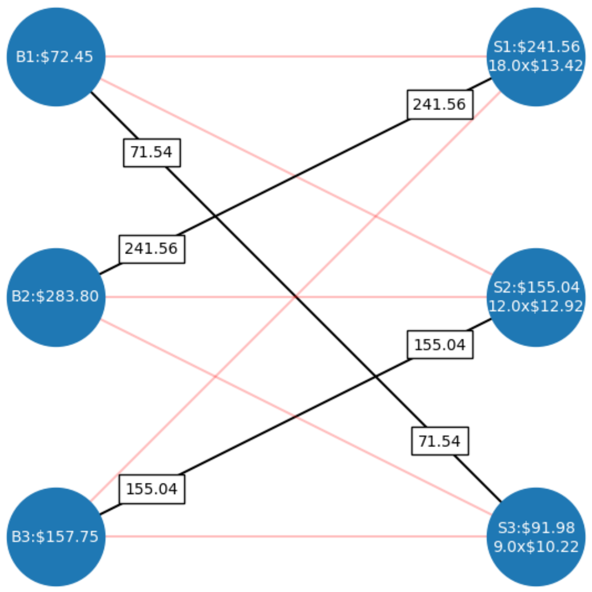

Fig. 9. ILP Example: Hungarian Solution

In figure 9, we have the sub-graph produced by the Hungarian algorithm. The total waste of this sub-graph is:

$$W = 25.42 + [\min(514, 488.58) - 468.14] = 45.86$$

It is important to make note of the edges that were selected. One hypothesis we have is that the sub-graph from the Hungarian algorithm will always be a sub-graph of an optimal solution.

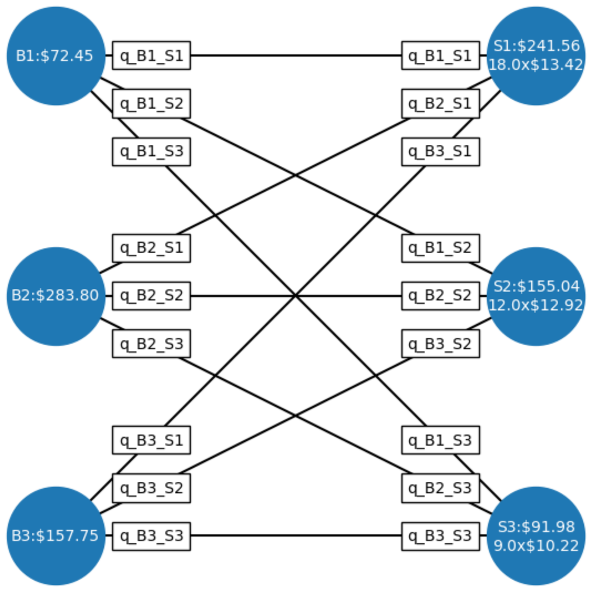

Fig. 10. ILP Example: ILP Variable Matrix

In figure 10, we have the visual representation of the matrix used to setup all of the key components to the ILP method.

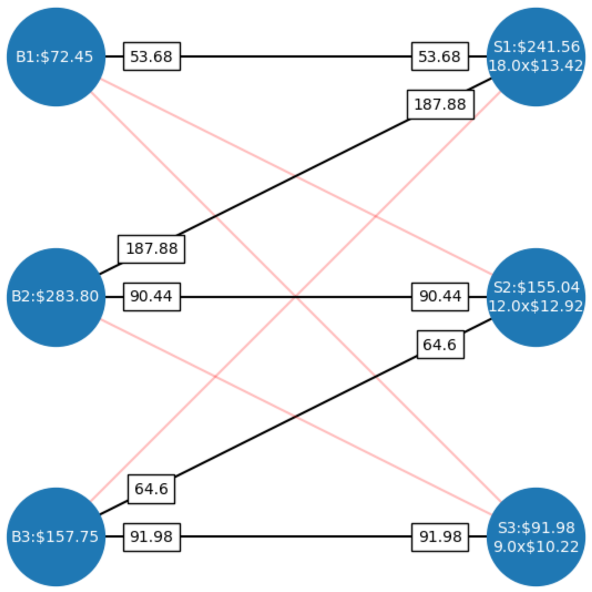

Fig. 11. ILP Example: ILP Solution

In figure 11, we have the sub-graph produced by the ILP method. The total waste of the sub-graph is:

$$W = 25.42 + [\min(514, 488.58) - 488.58] = 25.42$$

Here we see what an optimal solution looks like. In larger systems the number of multi-edged vertices will only increase as the number of edges increases in by a combinatoric order. We have not determined the exact amount of inaccuracy the Hungarian algorithm produces in comparison to the ILP method.

Notice that the sub-graph produced by the ILP method (Fig. 11) does not contain the graph from the Hungarian algorithm (Fig. 9). When using the ILP method to solve optimization problems, there can be times where there are multiple optimal solutions. Which means that even if the Hungarian algorithm does produce a sub-graph of an optimal solution, there's no guarantee the one we compute will match. This makes testing our hypothesis difficult as we would need to find all of the optimal solutions, and check to see if the Hungarian solution is a sub-graph of one of them.

E. Hungarian Influenced Integer Linear Programming (HILP)

To properly evaluate the graphs produced by the Hungarian algorithm, we create a second instance of the ILP method with an extra constraint: for every edge that appeared in the graph produced from the Hungarian algorithm, its corresponding decision variable in the ILP method must be greater than or equal to one. This guarantees that the sub-graph produced by the Hungarian algorithm will always be a sub-graph of the solution computed by the HILP method. We can then compare the two solutions to verify if our hypothesis is correct.

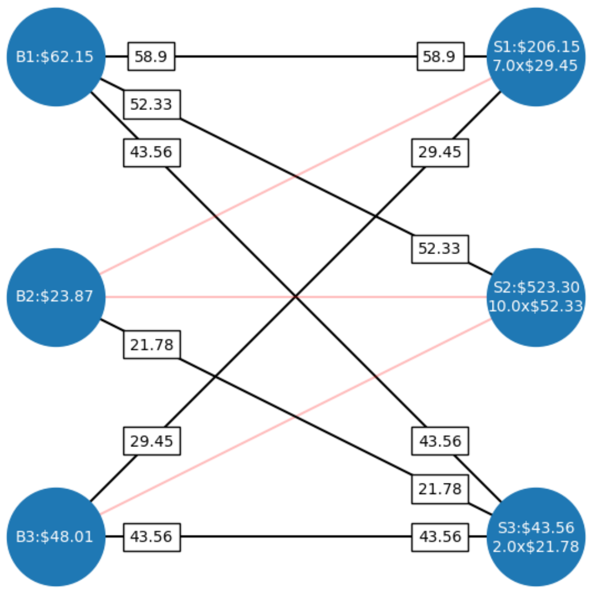

Fig. 12. Brokered Matrix

In figure 12, we have the sub-graph produced by the HILP method. The total waste of the sub-graph is:

$$W = 25.42 + [\min(514, 488.58) - 488.58] = 25.42$$

which is the same value as the ILP solution. This means that our hypothesis is correct for this example. Sadly this is not the case for all simulations.

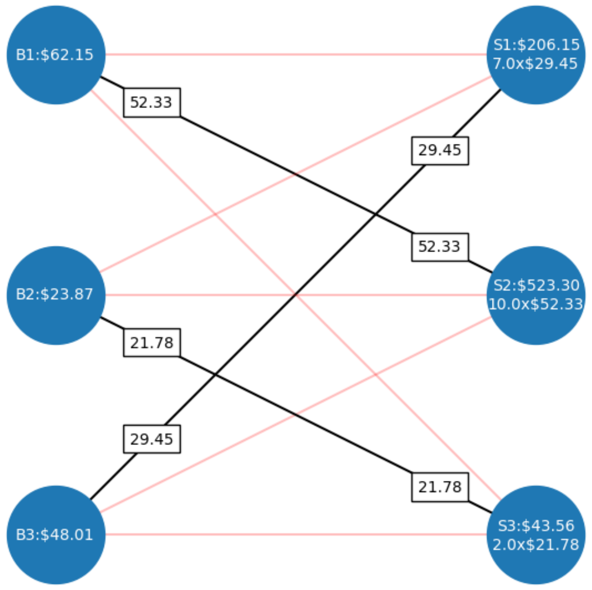

Fig. 13. Hungarian Solution

In figure 13, we have the sub-graph produced by the Hungarian algorithm. The total waste of the sub-graph is:

$$W = 638.98 + [\min(134.03, 773.01) - 103.56] = 669.45$$

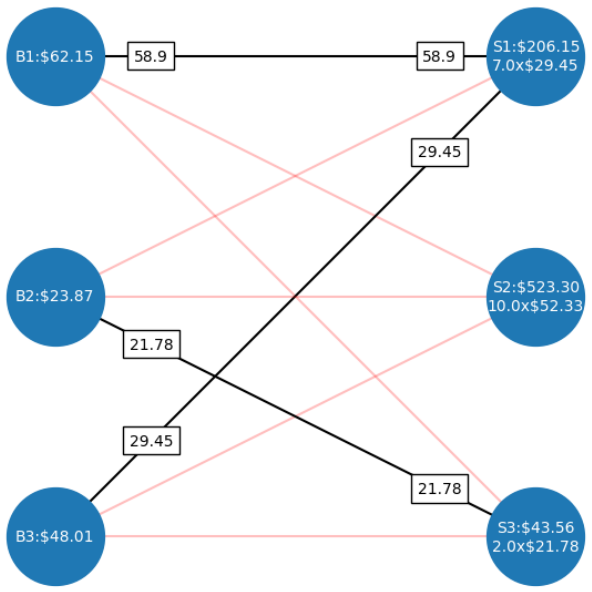

Fig. 14. ILP Solution

In figure 14, we have the sub-graph produced by the ILP method. The total waste of the sub-graph is:

$$W = 638.98 + [\min(134.03, 773.01) - 110.13] = 662.88$$

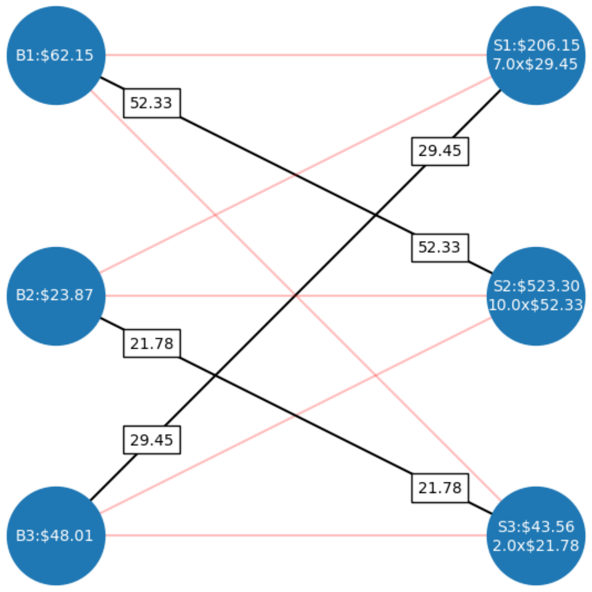

Fig. 15. HILP Solution

In figure 15, we have the sub-graph produced by the HILP method. The total waste of the sub-graph is:

$$W = 638.98 + [\min(134.03, 773.01) - 103.56] = 669.45$$

Here we see that enforcing the constraint that the Hungarian algorithm's graph must be a sub-graph of the ILP solution has made the solution sub-optimal. While this does mean that we cannot use the Hungarian algorithm to predict the shape of the optimal solution graph, this doesn't mean our technique has no use.

III. Reinforcement Learning

Reinforcement Learning allows us to train an agent to achieve our goal of minimizing the amount of waste our simulation produces. This agent can be rewarded based on how well it is able to choose edges, and their respective weights. This is where our ability to calculate the ideal solution in a simple constant time complexity function is incredibly powerful. We can check to see how close the sub-graph is to the optimal solution and make choices that leads towards that target value.

Here is where the Hungarian algorithm comes into play. Firstly, we can use it to create a sub-graph that, while sub-optimal, still helps teach the agent to prioritize the weights of the initial edges. This draws inspiration from the Bellman equation in Q-Learning. Which would translate to a low discount factor, as we are only looking at the first edge of each vertex, and not placing much importance on how it will affect the future outcomes.

We could also follow a more sequential algorithm, where we sort the buyers and sellers by totals and greedily form edges until we aren't able to make any more. However, this would not be a polynomial time complexity algorithm. Our technique will leverage the weighted bipartite graph, and Hungarian algorithm to speed up the time it takes for us to create an agent that is able to produce feasible solutions. The main drawback being the amount of inaccuracies this would method would produce.

IV. Quantum Computing

Quantum computing introduces a completely different method of computation that has the potential to significantly accelerate solving complex combinatorial optimization problems like the BSP. Unlike classical computing, which processes information in bits (either 0 or 1), quantum computing uses qubits that can represent both 0 and 1 simultaneously due to superposition. This allows quantum algorithms to explore multiple solutions in parallel.

In the context of the Buyer-Seller Problem, the exponential growth of possible solutions makes classical methods inefficient as the number of buyers and sellers increases. Quantum algorithms, such as Grover's algorithm for search or the Quantum Approximate Optimization Algorithm, can potentially provide faster convergence to an optimal or near-optimal solution by leveraging superposition and entanglement. This quantum advantage is particularly promising for problems where time complexity is a bottleneck, as is often the case with large-scale combinatorial optimizations.

While classical RL algorithms can efficiently train an agent to make decisions through interactions with the environment, they can still be slow when faced with large-scale, high-dimensional problems, as they require extensive exploration of the solution space. Quantum Reinforcement Learning (QRL) can overcome this limitation. With QRL, the quantum agent can explore multiple states and actions simultaneously, accelerating the learning process and reducing the time required to find near-optimal solutions.

For the Buyer-Seller Problem, where the goal is to minimize leftover resources (waste) while maximizing efficiency, QRL can enable a faster search for optimal buyer-seller pairings. By leveraging quantum-enhanced RL algorithms, such as Quantum Q-Learning or Quantum Policy Gradient methods, we can improve the agent's ability to explore a broader range of trade configurations in less time, leading to more efficient optimization.

V. Conclusion and Next Steps

It is only a matter of time before more tools and techniques using Quantum Computing emerge. We believe that more time spent focusing on quantum reinforcement learning will be the best way to demonstrate the strategies detailed in this paper.