HIGH SCHOOL EDITION
Journal of Student Research

Volume 12 Issue 1 (2023)

A Graph Algorithm to Eliminate Hunger and Food Waste by Matching Excess Food to Demand

Neil Makur[1] and Maya Mathews[#]

[1]Fremont Christian High School, Freemont, CA, USA
[#]Advisor

ABSTRACT

Millions of tons of food from grocery stores go to waste every year in the US, while many people consistently depend on community food banks. Having a plan to transport food surplus to the needy will eliminate both food waste and hunger at the same time. This problem is modeled as a bipartite graph with one set of vertices representing food donors with certain excess food of various types, and the other set representing food banks with demand of different types of food, and a cost of transportation on each edge. We create four heuristic algorithms to find low-cost transportation plans that satisfy all food needs, and compare them in terms of plan cost and runtime. We find that the basic greedy algorithm does the best in both aspects.

Introduction

Large quantities of food go to waste every day, including food surplus from retailers [REFED] while several local community food banks see a growing need of food [DAILYBOWL]. Several retailers are open to donating extra food to the local food banks. Food banks have partnerships with food donors, but food banks don't typically talk to each other. Similarly, food donors don't have visibility into what other donors may have to contribute. These blind spots lead to wasted food because of reasons such as food not reaching required food bank before it spoils, or too much of one kind of food at one food bank.

In this paper, we will implement and analyze different algorithms that generate a food distribution plan that sends all the surplus food to the right place where it is needed, thus eliminating both food waste and hunger in the local community. We aim to find an algorithm that generates a plan that distributes all food, minimizes transportation cost, and can be generated in a timely manner every day.

Related Work

There has been prior research that solves food distribution for food delivery where sources and sinks are not interchangeable services [Joshi et al.][Gupta et al.], for general supply chain for big firms like Walmart and Amazon-not including groceries[Chiles et al.], or for long-term relationships with food vendors [Michelson et al.] but no research that holistically looks at daily local food surplus and demand for optimum distribution, that this paper does. Bipartite graphs [Asratian et al.] used as the model in this paper has been used for food distribution[Joshi et al.][Gupta et al.], robot work distribution[Jain et al.] but not for solving a problem such as the one here where an optimal plan requires playing around with multiple parameters - edges, their cost, and supply/demand at vertices. By taking a holistic approach to process local food distribution problem, this paper presents various algorithms and determines their feasibility and application for small as well as large areas. Moreover, it builds a foundation that can be used for other distribution problems.

There are also several graph algorithms available for different types of problems (e.g. Max-Flow, Max-Flow/Min-Cost, Fixed Charge Transportation etc). However, the food distribution problem in this paper is fundamentally different. Max-flow problem is based on edge capacity alone, not on edge cost and can be solved in Polynomial time using Edmonds-Karp. Max-flow/min-cost takes cost into account but the cost is per unit of flow, unlike the problem presented in this paper where the cost of an edge is fixed. The fixed charge transportation problem, which is proven to be NP-Hard[Hochbaum et al.], can be used for edges with fixed cost. However, it only distributes one kind of item. To solve the local food distribution problem, we need consider multiple food types from each food donor to bank. This effectively makes it a fixed-charge-transportation with multiple graphs superimposed on each other to create one graph.

While matching for bipartite graphs is a well studied problem via theorems such as Hall's Theorem and algorithms such as the Hungarian Algorithm [Vassilev et al.], the food distribution problem in this paper is different from the bipartite graph matching problem because it has different quantities of food of various food types that need to be transported.

Food Distribution Problem as a Bipartite Graph

In order to study the problem from ground up and find solutions, we first restrict the problem with two attributes. Firstly, we only consider produce (fruits and vegetables) and not meat or dairy, as they require special handling such as cold storage and considering expiry dates. Produce also has a shelf life, but it is more uniform and manageable. Secondly, we only consider local area (~10 mile radius) so as to have transportation by road (vans/trucks). This is a reasonable limitation for timely food surplus distribution to local food banks.

The Model

Suppose that we have d places that donate food at the end of each day, b food banks that will accept food to distribute to the needy, and t food types (potatoes, lettuce, etc.). We call the food donors $1_D, 2_D, \dots, d_D$, the food banks $1_B, 2_B, \dots, b_B$, and number the types of food $1, 2, \dots, t$. Suppose that the donor i_D outputs d_i^x items of food type x, that the bank j_B needs at most b_j^x items of food type x, and that $\sum_{i=1}^{d} d_i^x = \sum_{j=1}^{b} b_j^x$ for $1 \leq x \leq t$. Suppose further that there is a cost $c_{i,j}$ for the road between i_D and j_B.

We can make this into a bipartite graph with $X = [d]$ and $Y = [b]$. The set of vertices on the left represents food donors, and the set of vertices on the right represents food banks. The edge between a donor and food bank has a cost of transporting food along that edge, which can be a function of distance.

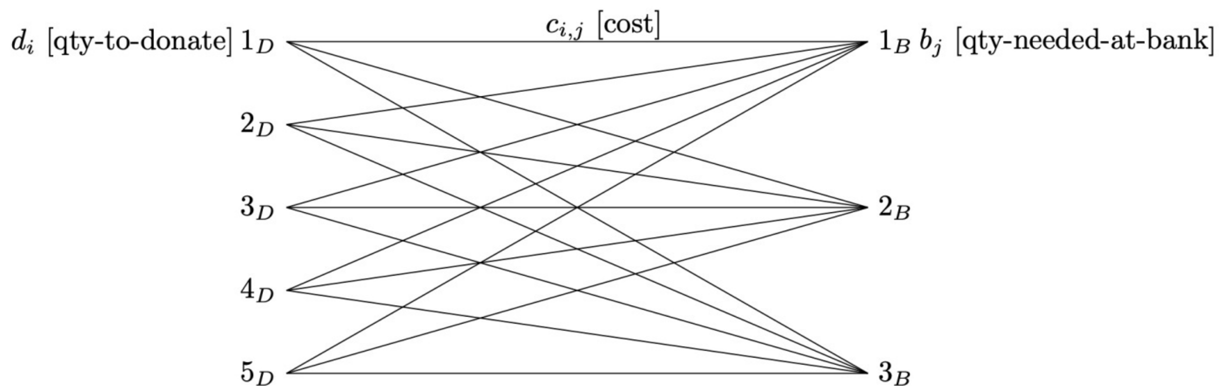

Figure 1. Representation of the bipartite graph with d=5 and b-=3

Definitions

Definition 1: *Distribution Problem:* A (d, b, t)-distribution problem is a tuple $(\mathcal{D}, \mathcal{B}, \mathcal{C})$, such that $\mathcal{D}: [d] \to (\mathbb{Z}^{\geq 0})^t$ is a function mapping i to a tuple $D_i = (d_i^1, d_i^2, \ldots\ldots d_i^t)$, $\mathcal{B}: [b] \to (\mathbb{Z}^{\geq 0})^t$ is a function mapping j to a tuple $B_j = (b_i^1, b_i^2, \ldots\ldots b_j^t)$, $\sum_{i=1}^{d} \mathcal{D}(i) = \sum_{j=1}^{b} \mathcal{B}(j)$, and $\mathcal{C} : [d] \times [b] \to \mathbb{Z}^{\geq 0}$ is a function mapping (i, j) to a value $c_{i,j}$.

The idea is that $\mathcal{D}$ gives the amount of food surplus at a given donor, $\mathcal{B}$ gives the amount of food need at a given bank, and $\mathcal{C}$ gives the cost of travel between a given donor and bank.

Definition 2: *Distribution Plan:* Let$(\mathcal{D}, \mathcal{B}, \mathcal{C})$ be a (d, b, t)-distribution problem. A distribution plan is a pair $P = (E, \mathcal{S})$, where $E \subseteq [d] \times [b]$ and $\mathcal{S} : E \to (\mathbb{Z}^{\geq 0})^t$, such that

$$\sum_{\substack{1 \leq j \leq b \\ (i,j) \in E}} \mathcal{S}(i, j) \leq \mathcal{D}(i) \quad i \in [d]$$

$$\sum_{\substack{1 \leq i \leq d \\ (i,j) \in E}} \mathcal{S}(i, j) \leq \mathcal{B}(j) \quad j \in [b]$$

We write

$$D(i) = \mathcal{D}(i) - \sum_{\substack{1 \leq j \leq b \\ (i,j) \in E}} \mathcal{S}(i, j) \quad i \in [d]$$

$$B(j) = \mathcal{B}(j) - \sum_{\substack{1 \leq i \leq d \\ (i,j) \in E}} \mathcal{S}(i, j) \quad j \in [b].$$

The idea of a distribution plan is that E represents which donors send food to which banks, and that $\mathcal{S}$ represents how much food is sent from each donor to each back. At any given point in time, $D(i)$ and $B(j)$ are the remaining supply and need respectively. We see from the definition that $D(i), B(j) \geq 0$.

Definition 3: *Full Distribution Plan:* Let $(\mathcal{D}, \mathcal{B}, \mathcal{C})$ be a (d, b, t)-distribution problem, and let $P = (E, \mathcal{S})$ be a distribution plan. We say that P is a *full distribution plan* if the following two conditions hold.

$$\sum_{\substack{1 \leq j \leq b \\ (i,j) \in E}} \mathcal{S}(i, j) = \mathcal{D}(i) \quad i \in [d]$$

$$\sum_{\substack{1 \leq i \leq d \\ (i,j) \in E}} \mathcal{S}(i, j) = \mathcal{B}(j) \quad j \in [b]$$

Alternatively, P is a full distribution plan if $D(i) = B(j) = 0$ for $i \in [d], j \in [b]$. A full distribution plan is one in which all supply is used and all need is fulfilled.

Definition 4: *Cost of a Plan:* Let $(\mathcal{D}, \mathcal{B}, \mathcal{C})$ be a (d, b, t)-distribution problem, and let $P = (E, \mathcal{S})$ be a distribution plan. The cost of P, $C(P)$ is

$$\sum_{(i,j) \in E} \mathcal{C}(i, j)$$

We therefore want to find a full distribution plan P such that $C(P)$ is minimized.We can also prove some results about our definitions that we will use later to prove that the algorithms are correct.

Proposition 5: Let $(\mathcal{D}, \mathcal{B}, \mathcal{C})$ be a (d, b, t)-distribution problem, and let $P = (E, \mathcal{S})$ be a distribution plan. *Then*$\sum_{1 \leq i \leq d} D(i) = \sum_{1 \leq j \leq b} B(j)$

Proof: We have that

$$\sum_{i=1}^{d} D(i) = \sum_{i=1}^{d}\left(\mathcal{D}(i) - \sum_{\substack{1\le j\le b\\(i,j)\in E}} \mathcal{S}(i,j)\right) = \sum_{i=1}^{d}\mathcal{D}(i) - \sum_{(i,j)\in E}\mathcal{S}(i,j)$$
$$= \sum_{j=1}^{b}\mathcal{B}(j) - \sum_{(i,j)\in E}\mathcal{S}(i,j) = \sum_{j=1}^{b}\left(\mathcal{B}(j) - \sum_{\substack{1\le i\le d\\(i,j)\in E}}\mathcal{S}(i,j)\right) = \sum_{j=1}^{b} B(j).$$

■

Corollary 6: Let $(\mathcal{D},\mathcal{B},\mathcal{C})$ be a (d,b,t)-distribution problem, and let $P=(E,\mathcal{S})$ be a distribution plan. Then, P is not a full distribution plan if and only if there exist $i\in[d]$ and $j\in[b]$ with $\min(D(i),B(j))\neq 0$.
Proof: Suppose that P is not a full distribution plan. Then, there is either some $i\in[d]$ with $D(i)\neq 0$, or some $j\in[b]$ with $B(j)\neq 0$. Without loss of generality, take $i\in[d]$ with $D(i)\neq 0$, and let $D(i)^x \neq 0$ in particular. Then, $\sum_{1\le i\le d} D(i) = \sum_{1\le j\le b} B(j)$, so that $0 \neq \sum_{1\le i\le d} D(i)^x = \sum_{1\le j\le b} B(j)^x$. Thus, there must be some j with $B(j)^x\neq 0$. Thus, $\min(D(i),B(j)^x)\neq 0$, so that $\min(D(i),B(j))\neq 0$. Conversely, suppose that there are $i\in[d]$, $j\in[b]$ with $\min(D(i),B(j))\neq 0$. Then, in particular, $D(i)\neq 0$, so we have that P cannot be a full distribution plan. ■

Building Block Algorithms

Before diving into any full algorithms, we present two algorithm modules that we will use as building blocks in all full algorithms and prove some theorems related to those. It is always understood that we have a (d,b,t)-distribution problem $(\mathcal{D},\mathcal{B},\mathcal{C})$.

Algorithm 1 Send All Food Along $\mathcal{E}$

```
Require: 𝓔 ⊆ [d] × [b] is ordered
 1: E ← ∅
 2: D ← 𝓓
 3: B ← 𝓑
 4: for (i, j) = e ∈ 𝓔 do
 5:     if min(D(i), B(j)) ≠ 0 then
 6:         E ← E ∪ {e}
 7:         𝓢(e) ← min(D(i), B(j))
 8:         D(i) ← D(i) − 𝓢(e)
 9:         B(j) ← B(j) − 𝓢(e)
10:     end if
11: end for
12: return (E, 𝓢)
```

Algorithm 1. The send-food algorithm module that is used as a building block in Algorithms 3,4,5,6, and 7. This algorithm takes in a set of edges, and returns a distribution plan using the given edges

Note that lines 7-9 are executed t times, as the computations are done for each type of food. We can find that this algorithm is $O(dbt)$. As t(number of types of food) is not expected to grow as much as d (number of donors) and b (number of food banks), we can consider this $O(db)$.

We want to show that $P=(E,\mathcal{S})$ generated by Algorithm 1 is a distribution plan.

Lemma 7: *After executing Algorithm 1,* $D(i)\ge 0$ *for all* $i\in[d]$*, and* $B(j)\ge 0$ *for all* $j\in[b]$.

Proof: The only time the value of D changes is in line 8, so we only need to examine this line. $S(e)$ is defined as $S(e) = \min(D(i), B(j)) \leq D(i)$, meaning that $D(i) - \mathcal{S}(e) \geq 0$. Hence, getting a $D(i) \ngeq 0$ cannot happen in line 8, and so cannot happen at all. The same argument applies for B. ■

Proposition 8: *Algorithm 1 produces a distribution plan.*

Proof: For $i \in [d]$, $D(i)$ is given by $D(i) - \sum_{\substack{1\le j\le b \\ (i,j)\in E}} \mathcal{S}(i,j)$. Lemma 7 gives us that this is ≥ 0, so that

$$\sum_{\substack{1\le j\le b \\ (i,j)\in E}} \mathcal{S}(i,j) \leq \mathcal{D}(i) \;\; i \in [d].$$

Similarly,

$$\sum_{\substack{1\le i\le d \\ (i,j)\in E}} \mathcal{S}(i,j) \leq \mathcal{B}(j) \;\; j \in [b].$$

■

Thus, Algorithm 1 produces a distribution plan. We can also find full distribution plans.

Proposition 9: *In the case that $\mathcal{E}$is a permutation of $[d] \times [b]$, Algorithm 1 produces a full distribution plan.*

Proof: Suppose that $(E, \mathcal{S})$ is not a full distribution plan, and take i, j with $\min(D(i), B(j)) \neq 0$ by Corollary 6. Since $\mathcal{E}$ is a permutation of $[d] \times [b], (i,j) \in \mathcal{E}$, and so $e = (i,j)$ for some iteration. Let $D'(i)$ and $B'(j)$ be the values of $D(i)$and $B(j)$ right after the iteration $e = (i,j)$, and $D''(i)$ and $B''(j)$ be the values right before the iteration $e = (i,j)$. We have that $D'(i) = D''(i) - \min(D''(i), B''(j))$ and that $B'(j) = B''(j) - \min(D''(i), B''(j))$. Thus, for each x, at least one of $D'(i)^x$ and $B'(j)^x$is 0, so that $\min(D'(i), B'(j)) = 0$ (running Algorithm 1 on all edges in $\mathcal{E}$ before e shows that $D''(i), B''(j) \geq 0$ so that $D'(i), B'(j) \geq 0$).. However, $D'(i) \geq D(i)$ and $B'(j) \geq B(j)$, so that $\min(D'(i), B'(j)) \geq \min(D(i), B(j)) \neq 0$, a contradiction. ■

The following algorithm will check if an edge e can be removed from a set of edges $\mathcal{E}$ while still providing a full distribution plan.

Algorithm 2 Can Remove e From $\mathcal{E}$

Require: $\mathcal{E} \subseteq [d] \times [b]$ is sorted
Require: $e \in \mathcal{E}$
Require: Executing "Send All Food Along $\mathcal{E}$" results in a full distribution plan

```
1: E ← E \ {e}
2: Send All Food Along E                                  ▷ Algorithm 1
3: for i ∈ [d] do
4:     if D(i) ≠ 0 then
5:         return FALSE
6:     end if
7: end for
8: return TRUE
```

Algorithm 2. Algorithm module to check if an edge can be removed

The costliest step of Algorithm 2 is the call to Algorithm 1. Thus, this is also $O(dbt)$. Since t (the number of types of food) is not expected to grow as much as d (number of donors) and b (number of food banks), we can consider this $O(db)$.

Algorithms for Optimal Cost

We will try five algorithms. Four of these algorithms will prioritize edges based on some heuristic, and one of them will be a complete brute-force algorithm. The four heuristic algorithms are devised based on whether they start with an empty set of edges or all edges, and what heuristic they use to pick an edge to add to the set, or remove an edge from the set. This is demonstrated in the following table:

Table 1. Description of four heuristic algorithms: Take-Cheap, Take-Expensive-Delta, Avoid-Expensive, and Avoid-Cheap-Delta

	Algorithm Converging Method	
	Pick	Remove
	Start with an empty set, *Add one edge at a time until all food is distributed.*	*Start with all edges,* *Remove edges from the set one at a time until no more can be removed.*
1st order	**Take Cheap** Pick cheapest of the remaining edges	**Avoid Expensive** Remove costliest of remaining edges
2nd order	**Take Expensive Delta** Pick the one with the most gap with next cheapest	**Avoid Cheap Delta** Remove the one with least gap with next costliest

Take-Cheap

In this algorithm, we order the edges between donors and banks from least to most expensive. Then we send all food across each edge in order, as long as there is food left to send. Note that since there could be various different types of foods, picking an edge does not mean that all the food available at one donor (or needed by one bank) will be transported by that edge. Rather, for each food type, the minimum of the available amount of food and the needed amount of food will be transported.

Algorithm 3 Take-Cheap

1: $\mathcal{E} \leftarrow [d] \times [b]$
2: Sort $\mathcal{E}$ such that $\mathcal{C}$ is non-decreasing
3: $(E, \mathcal{S}) \leftarrow$ Send All Food Along $\mathcal{E}$ ▷ Algorithm 1
4: **return** $(E, \mathcal{S})$

Algorithm 3. The Take-Cheap Algorithm

This algorithm iterates over each edge, as well as doing computations for each food type, and so is $O(db(t + log(db))$. Since t (the number of types of food) is not expected to grow as much as d (number of donors) and b (number of food banks), we can consider this $O(db \cdot log(db))$.

Take-Expensive-Delta

There are some cases where picking the cheapest edge may not be the most optimal strategy. Consider the following (2,2,1)-distribution problem:

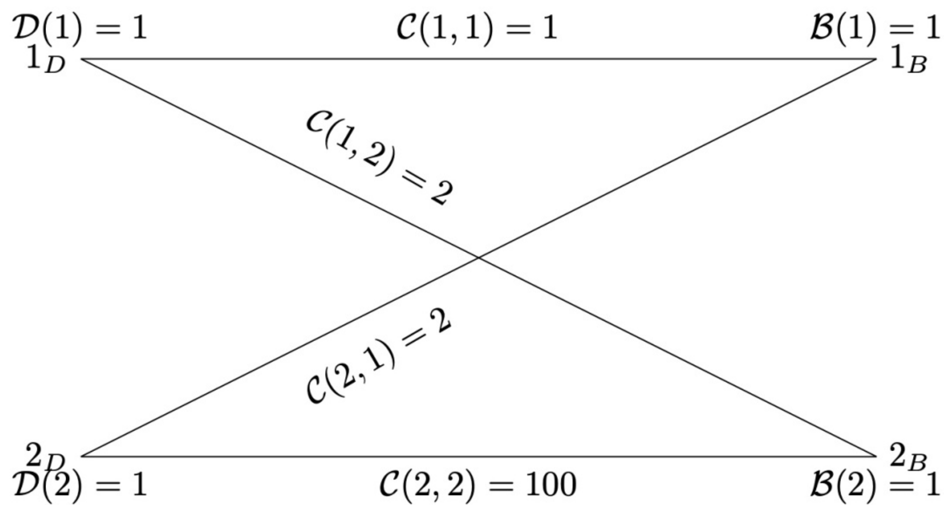

Figure 2. Example for the Take-Expensive-Delta Algorithm

The Take-Cheap algorithm will pick edge of cost 1, followed by that of cost 100, giving a total cost of 101. However, picking the edges of cost 2 will make a better plan with total cost of 4. The core reason for this is that locally choosing the cheapest edge can be globally costly if the remaining edge choices are much more expensive. We try to approach this by considering an ordering on the delta between different costs. For the example in Figure 2, the costs in ascending order are $\{1,2,2,100\}$ and the differences are$\{1,0,98,-1\}$ (we define the last difference to be -1 because there is nothing worse than it). Sorting the edges in descending order of their difference with the next edge gives us $\{2,1,2,100\}$, and picking edges in this order gives us the solution $\{2,2\}$, which is optimal in this case.

The intuitive way of thinking about this is that we pick the edge that, if not picked, will have the most consequence down the line because the next edge is going to be a lot more expensive.

In this algorithm, we order the edges between donors and banks from least to most expensive, and calculate the difference in cost between each edge and the following edge. Then order the edges from highest difference to lowest difference, and send all food across each edge in order if there is still food left to send.

Algorithm 4 Take-Expensive-Delta

```
1: ℰ ← [d] × [b]
2: Sort ℰ such that 𝒞 is non-decreasing
3: for e = (i, j) ∈ ℰ do
4:     if e is not the last edge then
5:         e′ ← edge after e
6:         Δ(e) ← 𝒞(e′) − 𝒞(e)
7:     else
8:         Δ(e) ← −1
9:     end if
10: end for
11: Sort ℰ such that Δ is non-increasing
12: (E, 𝒮) ← Send All Food Along ℰ                ▷ Algorithm 1
13: return (E, 𝒮)
```

Algorithm 4. The Take-Expensive-Delta Algorithm

This algorithm also iterates over each edge, as well as doing computations for each food type, and so is $O(db(t + log(db))$. Since t (the number of types of food) is not expected to grow as much as d (number of donors) and b (number of food banks), we can consider this $O(db \cdot log(db))$.

Avoid-Expensive

This algorithm takes the opposite of Take-Cheap approach and removes the edges one by one until no more edges can be removed. We order the edges between donors and banks from most to least expensive, and then remove each edge in order, as long as it is possible to do so and still send all food across.

Algorithm 5 Avoid-Expensive

1: $\mathcal{E} \leftarrow [d] \times [b]$
2: Sort $\mathcal{E}$ such that $\mathcal{C}$ is non-increasing
3: **for** $e = (i, j) \in \mathcal{E}$ **do**
4:     **if** Can Remove e From $\mathcal{E}$ **then** ▷ Algorithm 2
5:         $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e\}$
6:     **end if**
7: **end for**
8: $(E, \mathcal{S}) \leftarrow$ Send All Food Along $\mathcal{E}$ ▷ Algorithm 1
9: **return** $(E, \mathcal{S})$

Algorithm 5. The Avoid-Expensive Algorithm

This algorithm is $O(d^2b^2t)$. Since t (the number of types of food) is not expected to grow as much as d (number of donors) and b (number of food banks), we can consider this $O(d^2b^2)$.

Avoid-Cheap-Delta

Following the same intuition as Take-Expensive-Delta, we try Algorithm 5 but instead of removing the costliest edge, we remove the edge that is the most close in cost to the next edge. This edge, if not avoided, will have the most consequence down the line because the next edge is cheaper and can likely lead to a satisfactory plan as well. We order the edges between donors and banks from least to most expensive, and calculate the difference in cost between each edge and the following edge. Then, we order the edges from lowest difference to highest difference, and remove each edge in order, as long as it is possible to do so and still send all food across.

Algorithm 6 Avoid-Cheap-Delta

```
1: 𝓔 ← [d] × [b]
2: Sort 𝓔 such that 𝓒 is non-decreasing
3: for e = (i, j) ∈ 𝓔 do
4:     if e is not the last edge then
5:         e′ ← edge after e
6:         Δ(e) ← 𝓒(e′) − 𝓒(e)
7:     else
8:         Δ(e) ← −1
9:     end if
10: end for
11: Sort 𝓔 such that Δ is non-increasing
12: for e = (i, j) ∈ 𝓔 do
13:     if Can Remove e From 𝓔 then                ▷ Algorithm 2
14:         𝓔 ← 𝓔 \ {e}
15:     end if
16: end for
17: (E, 𝓢) ← Send All Food Along 𝓔                 ▷ Algorithm 1
18: return (E, 𝓢)
```

Algorithm 6. The Avoid-Cheap-Delta Algorithm

We find that this is $O(d^2b^2t)$ Since t (the number of types of food) is not expected to grow as much as d (number of donors) and b (number of food banks), we can consider this $O(d^2b^2)$.

Brute-Force

In this algorithm, we try all possible combinations of transportation, calculate the cost of each, and choose the combination with the minimum cost. This will ensure that the best combination is found.

Algorithm 7 Brute-Force

```
1: COST-MIN← ∞
2: for 𝓔 is a permutation of [d] × [b] do
3:     (E, 𝓢) ← Send All Food Along 𝓔              ▷ Algorithm 1
4:     COST← 0
5:     for e ∈ E do
6:         COST←COST+𝓒(e)
7:     end for
8:     if COST<COST-MIN then
9:         E-MIN← E
10:        𝓢-MIN← 𝓢
11:        COST-MIN←COST
12:    end if
13: end for
14: return (E-MIN, 𝓢-MIN)
```

Algorithm 7. The Brute Force Algorithm

This is $O(dbt \cdot (db)!)$. Since t (the number of types of food) is not expected to grow as much as d (number of donors) and b (number of food banks), we can consider this $O(db \cdot (db)!)$

We summarize the order of various algorithms in Table 2 below.

Table 2. Comparison of all algorithms for d=5, b=3.

Algorithm	Take Cheap	Take Expensive Delta	Avoid Expensive	Avoid Cheap Delta	Brute Force
Algorithm Order	$O(db \cdot log(db))$	$O(db \cdot \log(db))$	$O(d^2b^2)$	$O(d^2b^2)$	$O(db \cdot (db)!)$

Methods and Experiment Setup

A simulator is used to generate random integers for five inputs within a max range defined in Table 3. The food quantity for each food type is distributed among donors and banks by randomly choosing one donor/bank to increase the supply/demand repeatedly. We generated 50 inputs, and each experiment was repeated 10 times, and the average was taken.

Table 3. Integer range of different generated variables

Variable	d	b	t	D, B	c_{ij}
Description	*Number of donor*	*Number of food banks*	*Number of food types*	*Food quantity*	*Edge Cost*
Range	1-100	1-100	1-10	1-10,000	1-100

Each experiment outputs:

- Runtime in nanoseconds
- Total cost of the plan
- Number of edges selected in the plan

We use the above output to generate the following secondary outputs:

- Runtime in milliseconds/seconds
- Plan cost as a percentage of max cost
- Number of selected edges as a percentage of total number of edges

Here is a sample of what the input and output fields look like:

Algorithm	Algo	Number of donors	Number of banks	Types of food	Total number of edges (number of donors * number of banks)	Total cost of all edges	Cost of selected edges	Number of edges selected	Runtime (nanoseconds)	% selected edges	%cost	Runtime (ms)	Runtime (s)
0	Take Cheap	26	5	3	130	6335	1124	35	1066700	26.92	17.74	1.07	0.00
1	Take Expensive Delta	26	5	3	130	6335	1360	32	522200	24.62	21.47	0.52	0.00
2	Avoid Expensive	26	5	3	130	6335	1935	35	272634700	26.92	30.54	272.63	0.27
3	Avoid Cheap Delta	26	5	3	130	6335	1414	32	216830700	24.62	22.32	216.83	0.22
0	Take Cheap	11	15	2	165	7732	524	26	83700	15.76	6.78	0.08	0.00
0	Take Cheap	11	15	2	165	7732	524	26	70900	15.76	6.78	0.07	0.00
1	Take Expensive Delta	11	15	2	165	7732	928	25	105200	15.15	12.00	0.11	0.00
1	Take Expensive Delta	11	15	2	165	7732	928	25	294300	15.15	12.00	0.29	0.00
2	Avoid Expensive	11	15	2	165	7732	1148	26	146092600	15.76	14.85	146.09	0.15
2	Avoid Expensive	11	15	2	165	7732	1148	26	254042500	15.76	14.85	254.04	0.25
3	Avoid Cheap Delta	11	15	2	165	7732	1301	25	320224400	15.15	16.83	320.22	0.32
3	Avoid Cheap Delta	11	15	2	165	7732	1301	25	169515300	15.15	16.83	169.52	0.17
0	Take Cheap	5	41	5	205	10736	1426	51	311500	24.88	13.28	0.31	0.00
1	Take Expensive Delta	5	41	5	205	10736	1951	50	288700	24.39	18.17	0.29	0.00
2	Avoid Expensive	5	41	5	205	10736	3219	48	1341707500	23.41	29.98	1341.71	1.34
3	Avoid Cheap Delta	5	41	5	205	10736	2862	47	1105341500	22.93	26.66	1105.34	1.11
0	Take Cheap	7	30	3	210	10681	820	37	944200	17.62	7.68	0.94	0.00

Figure 3. Sample Data

All experiments were run on HP laptop 2.90 GHz core 11th Gen Intel Core i7-1195G7 with32 GB of RAM using C++.

Results & Analysis

Comparison of Runtime and Plan Cost

We started with 2 experiments for $d = 3, b = 3$ and $d = 5, b = 3$. These experiments made clear that it is not feasible to run any more experiments using Brute-Force Algorithm, which runs through all permutations and checks them, making it exponential in computational complexity. Thus, it could only run to completion for a very small dataset (d=3, b=3). As shown in Table 4 and Table 5, the other algorithms complete much faster and also generate reasonable plans.

Table 4. Comparison of all algorithms for d=3, b=3.

Input Data Size: d=3, b= 3, t=3					
Algorithm	Take Cheap	Take Expensive Delta	Avoid Expensive	Avoid Cheap Delta	Brute Force
Cost of plan	290	264	375	302	225
Number of edges	5	6	5	5	5
Runtime (ns)	25,700	27,300	2,791,200	2,690,800	67,613,496,400

The Take-Cheap and Take-Expensive-Delta algorithms finish in milliseconds, Avoid-Expensive and Avoid-Cheap-Delta finish in seconds. The Brute Force Algorithm takes hours but generates the best plan. The second best plan (Take-Expensive-Delta) is 17.33% worse than the best plan.

Table 5. Comparison of all algorithms for d=5, b=3.

Input Data Size: d=5, b= 3, t=3					
Algorithm	Take Cheap	Take Expensive Delta	Avoid Expensive	Avoid Cheap Delta	Brute Force
Cost of plan	628	580	461	573	542
Number of edges	12	11	10	10	9
Runtime (ns)	61,200	45,100	5,940,500	4,827,800	>3 days

The Take-Cheap and Take-Expensive-Delta algorithms finish in milliseconds, Avoid-Expensive and Avoid-Cheap-Delta finish in seconds. The Brute Force Algorithm did not finish in 3 days. By the time it was terminated (in 3 days), it had not found the optimal plan (we know that there is at least one better plan available: the plan generated by Avoid-Expensive algorithm).

As Brute-Force algorithm would not have run in a reasonable time for it to be practical, we decided to omit that from further experiments. The next 3 subsections focus on the four heuristic algorithms and find the best one.

Comparing Heuristic Algorithms

Each algorithm was run 10 times on 50 randomly generated data points. The runtime was slightly different and was averaged for the purpose of analysis in this section.

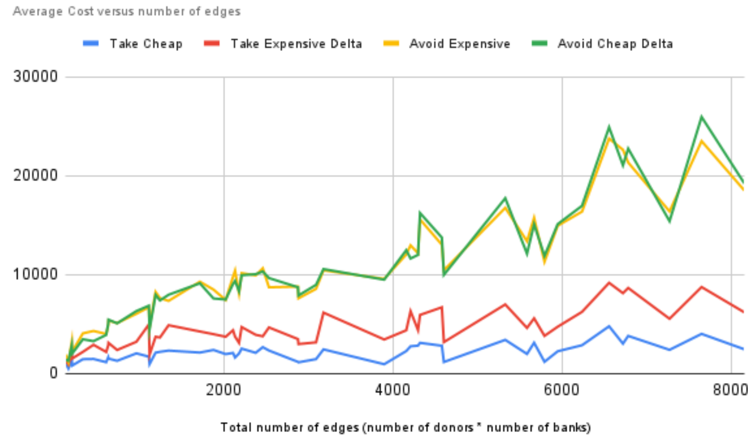

Figure 4. Cost of the plan generated by various algorithms for different number of edges in the graph.

The graph in Figure 4 shows the cost that each algorithm generates. It is evident that the basic greedy algorithm ($Take\ Cheap$) generates the best possible plan consistently. It also shows that the algorithms that reverse greedy ($Avoid\ Expensive$and $Avoid\ Cheap\ Delta$) produce a much worse plans than the other two algorithms and are probably not worth considering more.

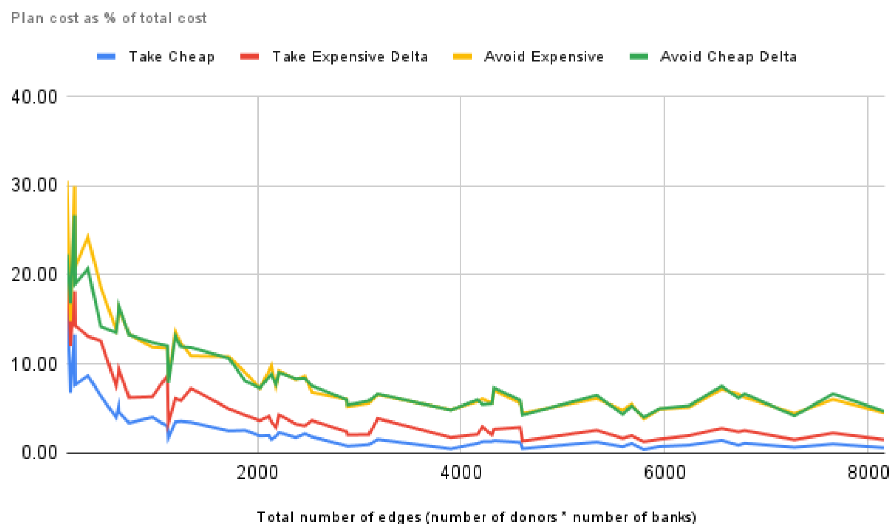

Figure 5. Cost of the plan generated by various algorithms as a percentage of maximum cost (sum of all edges)

The graph in Figure 5 shows the same data as the previous one, except that the cost is expressed as percentage of the total possible cost (i.e. if all the $d \times b$ edges were used to transport the food). This analysis is interesting because food banks and donors operate blindly today, without any knowledge of what is available at a donor and what is needed at a bank. Thus, they might be incurring the total possible cost. If we compare the different algorithms, the $Take\ Cheap$ algorithm does the best again. Another interesting insight from this graph is that all algorithms seem to converge as the number of donors and banks get large.

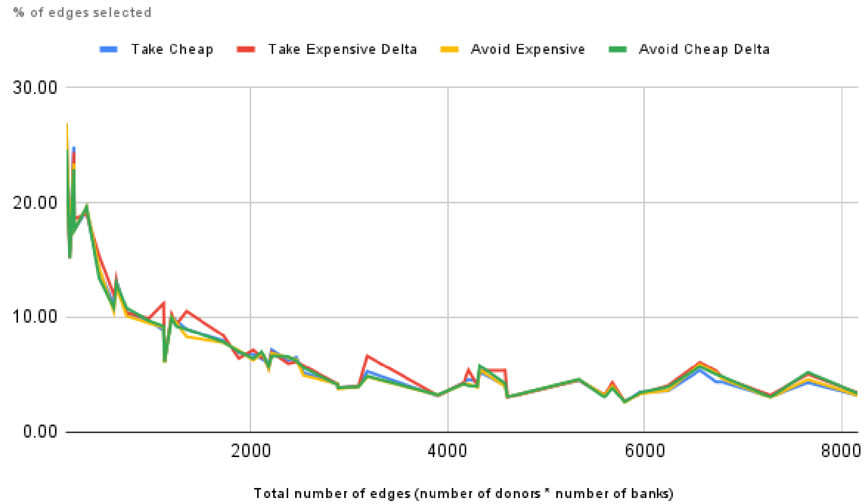

Figure 6. Number of edges selected by various algorithms as a percentage of maximum cost (sum of all edges)

The graph in Figure 6 shows the edges selected by a plan as a percentage of total number of edges $(d\ X\ b)$. Surprisingly, all algorithms select about the same number of edges. Since the cost of these algorithms is different, as shown by last two graphs, it means that the algorithms like $Take\ Cheap$ do a better job of selecting which edges to select.

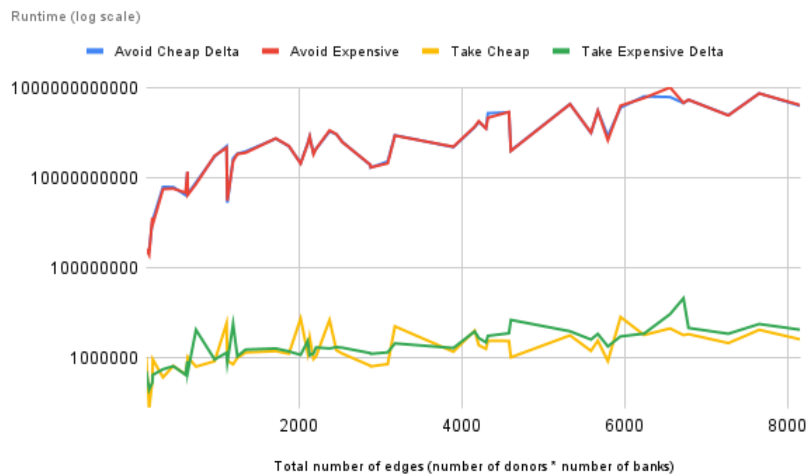

Figure 7. Runtime comparison for various algorithms

The graph in Figure 7 compares the runtime of various algorithms at log scale. The $Avoid$ algorithms are much more expensive. This is because the part of the algorithm that checks if edges can be removed makes it $O(d^2b^2)$, while the other algorithms are $O(db \cdot \log{(db)})$.

Further Analysis of "Take Cheap"

The previous section conclusively proves that the basic greedy $Take\ Cheap$ algorithm outperforms other algorithms both in runtime and finding the best possible plan. We do some further analysis on this algorithm to see how it would behave in the local area.

Based on Google Maps search results, our local urban area of 3 neighboring cities has about 110 grocery stores and 12 food banks. If we assume about 30% of the grocery stores participate in surplus food donation, a reasonable upper bound for number of donors would be 33, and number of banks would be 12.

We run two more experiments to analyze the “Take Cheap” algorithm on a small number of donors and banks.

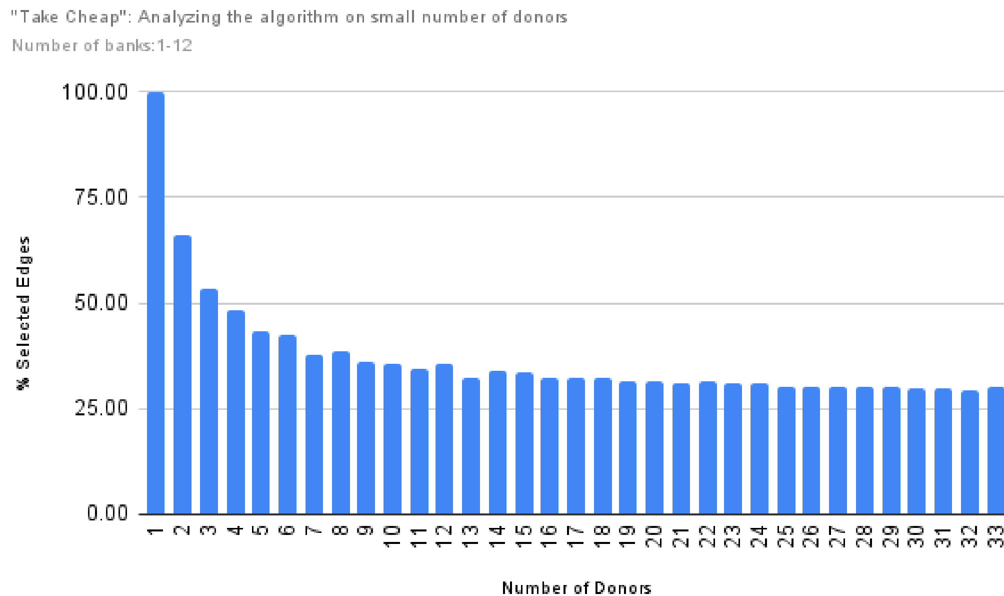

Figure 8. Take Cheap - studying convergence on small data by donors

The graph in Figure 8 plots the average (for number of banks $1-12$) of percent of selected edges for different number of donors. It shows that starting at $d = 4$, 50% or less edges are selected, making it meaningful to use the algorithm to determine a plan.

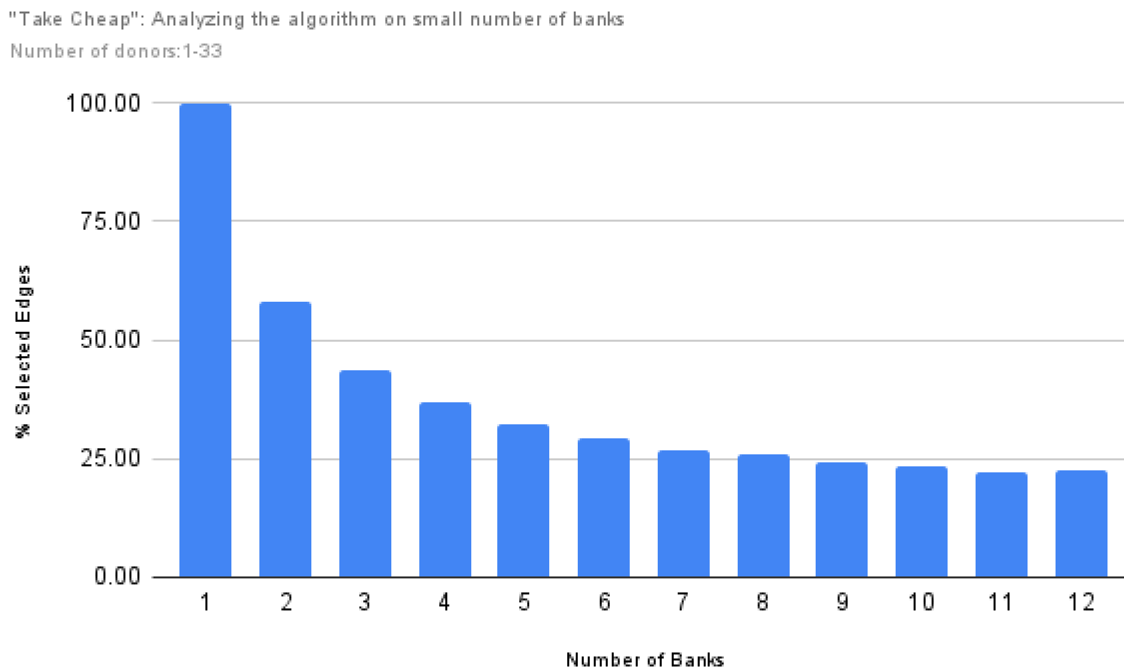

Figure 9. Take Cheap - studying convergence on small data by banks

The graph in Figure 9 plots the average (for number of donors $1-33$) of percent of selected edges for different number of donors. It shows that starting at $b=3$, 50% or less edges are selected, making it meaningful to use the algorithm to determine a plan.

Based on this data, it seems that once d or b gets above a fairly small threshold (< 5), under half the of the edges tend to be used.

Detailed Runtime Analysis of Heuristic Algorithms

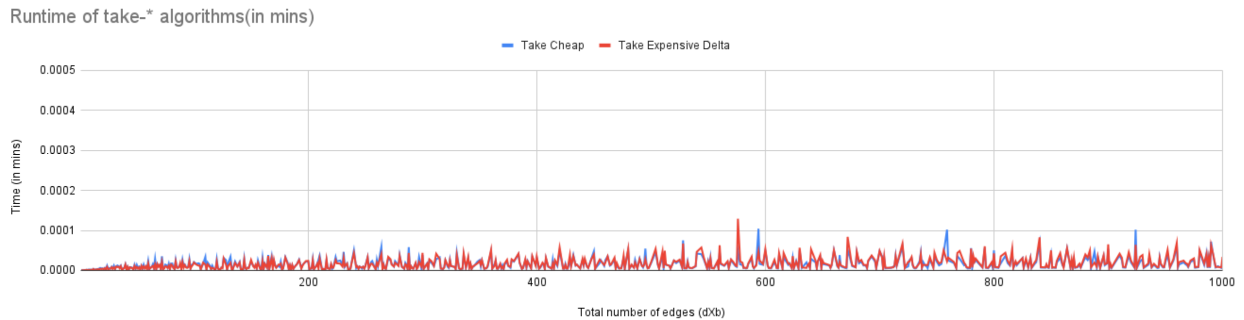

Figure 10. Runtime of Take $-$ Cheap and Take $-$ Expensiv $-$ Delta algorithms - all under 0.0001 minutes

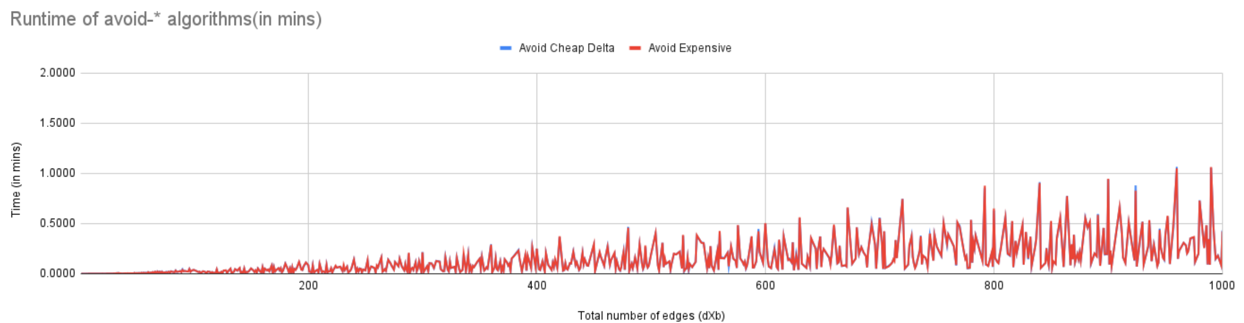

Figure 11. Runtime of $Avoid - Expensive$ and $Avoid - Cheap - Delta$ algorithms - all under 1.1 minutes

For $\mathrm{d.b} \leq 1000$, all 4 heuristic algorithms (Take $-$ Cheap, Take $-$ Expensive $-$ Delta, Avoid $-$ Expensive, and Avoid $-$ Cheap $-$ Delta) run in a very reasonable time (under 1.1 minutes).

Future Work

The closest existing graph problem that is similar to the one presented in this paper is Fixed Charge Transportation, which is proven to be NP-Hard [Hochbaum et al.]. Researchers have tried heuristics and linear programming solutions for this problem [Kim et al.] [Vyve] [KimHJ et al.]. Our problem is fundamentally different from Fixed Charge Transportation problem as it takes into account various types of food, which can independently vary in quantity for each transport along an edge. We can also try to formulate this problem as a linear programming problem or a mixed-integer linear programming problem and see if it can find better solutions in the same amount of time. We can additionally try a randomized algorithm with probabilities based on costs to try and find an algorithm that does better than Take-Cheap.

Conclusion

We found a new mathematical formulation of the seemingly well-known problem of distributing food from donors to food banks, which can be used as a basis to design further algorithms for optimal distribution. Further, the same formulation can be used in any problem in which multiple quantities need to be distributed from sources to sinks while minimizing a fixed-edge cost.

The purpose of this paper was to find an algorithm that generates a plan to distribute all food, minimizing the transportation cost. Our result provided useful insights into the various algorithms tried. Trying every possible solution in a brute-force manner is infeasible, as it takes days. Our heuristic algorithms all run in a reasonable amount of time, even for large number of donors and banks. They also generate reasonably cheap plans, with the best one being the $Take - Cheap$ algorithm. This algorithm tends to provide cheaper plans than our other three algorithms.

In the real world, this can be applied to a local area where food banks can input their daily needs based on what they see in the local community. Food donors can put in information about excess food at the end of each day, with which the algorithm can quickly generate a plan and communicate to the food banks for pickup from various donors.

Acknowledgments

I would like to thank volunteers at Tri City Food Bank (Fremont, CA) and employees from Safeway (Newark, CA), Sprouts (Newark, CA), Whole Foods (Fremont, CA), Smart and Final (Fremont, CA), Raley's (Fremont, CA), Trader Joe's (Fremont, CA) for taking the time to speak with me about how food is distributed in the local area.

References

[Asratian et al.] Asratian, A.S., Denley, T.M.J., &Haggkvist, R. [2008] Bipartite Graphs and Their Applications

[Chiles et al] Chiles, C.R., &Dau, M.T (2005). An analysis of current supply chain best practices in the retail industry with case studies of Wal-Mart and Amazon.com. http://hdl.handle.net/1721.1/33314

[DAILYBOWL] Daily Bowl https://dailybowl.org

[Gupta et al.] Gupta, A., Yadav, R., Nair, A., Chakraborty, Abhijnan.,Ranu, S.,Bagchi, A. (2022). Fairfoody: Bringing in fairness in food delivery. *In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36*, pages 11900-11907. https://doi.org/10.48550/arXiv.2203.08849

[Hochbaum et al.] Hochbaum, D.S., & Segev, A. [1989] Analysis of a flow problem with fixed charges. In Network, An International Journal Volume 19, Issue 3. https://doi.org/10.1002/net.3230190304

[Jain et al.] Jain, P., Goodrich, M.A. (2022). Processes for a Colony Solving the Best-of-N Problem Using a Bipartite Graph Representation. *In: Matsuno, F., Azuma, Si., Yamamoto, M. (eds) Distributed Autonomous Robotic Systems. DARS 2021. Springer Proceedings in Advanced Robotics, vol 22. Springer, Cham.*https://doi.org/10.1007/978-3-030-92790-5_29

[Joshi et al.] Joshi, M., Singh, A., Ranu, S, Bagchi, A., Karia, P. & Kala, P. (2021). Batching and matching for food delivery in dynamic road networks. *In 2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2099-2104.https://doi.org/10.48550/arXiv.2008.12905

[Kim et al.] Kim, D., &Pardalos, P.M.,A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure[1999]. *In Operations Research Letters*,Volume 24, Issue 4,Pages 195-203,ISSN 0167-6377,https://doi.org/10.1016/S0167-6377(99)00004-8

[KimHJ et al.] Kim, HJ., &Hooker, J.N. Solving Fixed-Charge Network Flow Problems with a Hybrid Optimization and Constraint Programming Approach. *In Annals of Operations Research* 115, 95–124 (2002).https://doi.org/10.1023/A:1021145103592

[Michelson et al.] Michelson, H., Boucher, S., Cheng, X., Huang, J., & Jia, X. (2018). Connecting supermarkets and farms: The role of intermediaries in Walmart China's fresh produce supply chains. In *Renewable Agriculture and Food Systems, 33(1)*, 47-59. https://doi.org/10.1017/S174217051600051X

[REFED] ReFED Insights Engine https://insights-engine.refed.org

[Vassilev et al.] Vassilev, T.S., Huntington, L. Algorithms for Matchings in Graphs. *In Algorithms Research , Vol. 1 No. 4, 2012, pp. 20-30 (2012).*

[Vyve] Van Vyve, M. (2011). Fixed-Charge Transportation on a Path: Linear Programming Formulations. *In Günlük, O., Woeginger, G.J. (eds) Integer Programming and Combinatoral Optimization. IPCO 2011*. Lecture Notes in Computer Science, vol 6655. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-20807-2_33