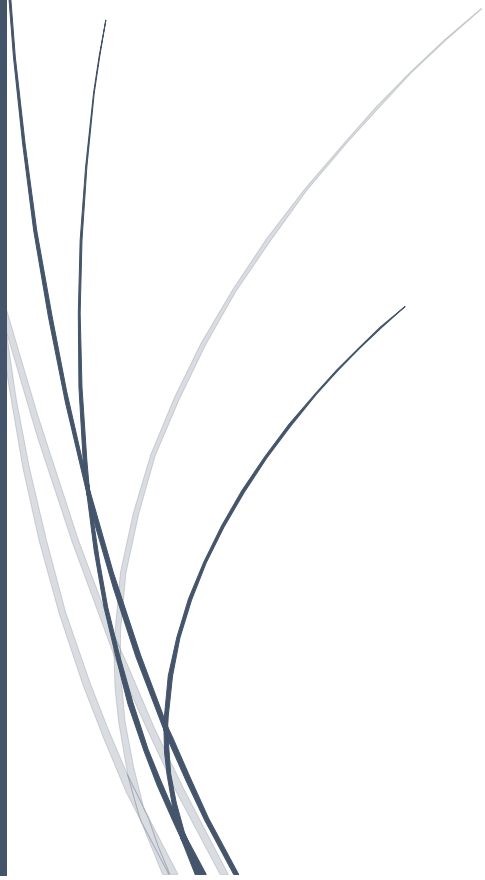
6/5/2019

Environmental Impact Study for Table to Table

A Food Rescue Organization



Jessica Ayers, Danielle Thomas, Erica Wiener
COMMUNITY CENTERED PROBLEM SOLVING AND DESIGN

Table of Contents

Introduction

Table to Table's (T2T) mission is to keep wholesome, edible food from going to waste by collecting it from donors and then distributing to those in need, which includes agencies that serve the hungry, homeless and at-risk populations. Their operations consist of food (fresh and perishable) retrieval and delivery of donated items to recipient agencies all in the same day. T2T recovers fresh, frozen, and prepared food that is donated from local restaurants, grocery stores, bakeries, schools, and businesses. The food is delivered to community social service agencies and low-income housing sites. Since the organization began in April 1966, they have rescued and redistributed over 18 million pounds of food.

In the last 50 years, it has been challenging to quantify T2T's impacts of food rescue to their local communities. T2T needs to have credible, fastidious calculations so they can apply for grants, market to potential donors, and argue the importance of food rescue in their neighborhoods. The University of Iowa (UI) team collaborated directly with Nicki Ross and Emily Meister, the Director and Program Coordinator of T2T, respectively. Through a series of emails and in-person meetings, T2T guided the goals, decisions, and inclusions of the project. The main goal of this work is to quantify T2T's food rescue impact so that they will have evidence of their specific net environmental gains. Specifically, they want to specifically quantify their impact through:

1. water used in food production (categories) that would have otherwise been disposed of
2. carbon dioxide equivalent emitted in the production of T2T food groups
3. greenhouse gas (GHG) reductions by diverting food waste from landfills
4. environmental inputs of the food rescue operations of T2T

Food Waste in Iowa City, Iowa

The Environmental Protection Agency (EPA) promotes sustainable management of food. They provide a systematic approach that seeks to reduce wasted food and its associated impacts over the entire life cycle, starting with the use of natural resources, manufacturing, sales and consumptions. The EPA defines wasted food as food that was not used for its

intended purpose, and it is managed in a variety of ways. Wasted food can be donated to feed people, converted to animal feed, composted, or sent to landfills or combustion facilities. Examples of wasted food could be unsold food from retail stores; plate waste, uneaten prepared food or kitchen trimmings from restaurants, cafeterias, and households; or by-products from food and beverage processing facilities.[1] The EPA's Food Recovery Hierarchy (Figure 1) prioritizes how different organizations can act to prevent and divert wasted food. Each tier of the hierarchy focuses on different management strategies from wasted food. They consider the top levels to be the best ways to prevent and divert wasted food because they create the largest societal and environmental benefit. T2T fits in on the second level of this hierarchy and their operations benefits Johnson County's economy and environment.

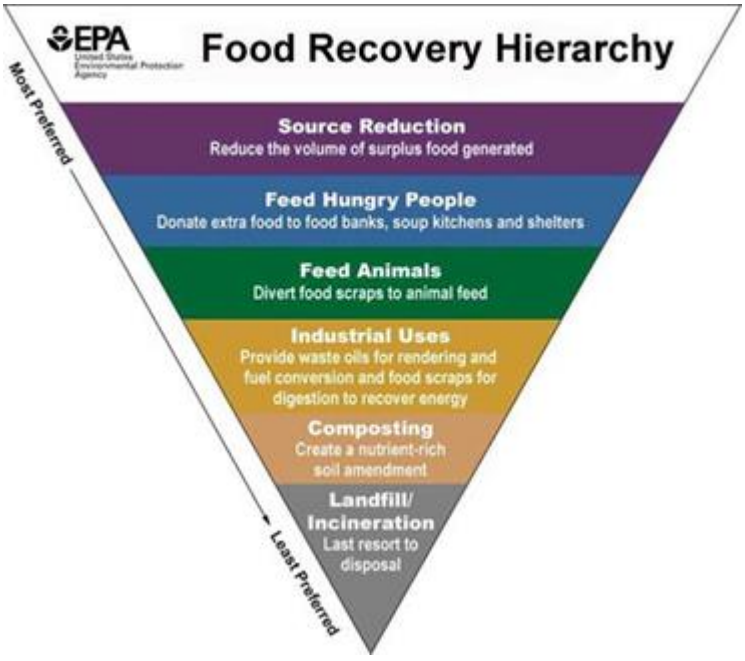


Figure 1. The EPA's Food Recovery Hierarchy for sustainable management of food.

There are many benefits of reducing food waste, including reducing the amount of methane in landfills, reducing resource use associated with food production, improving soil health and structure, increasing drought resistance, reducing the need for supplemental water, fertilizers and pesticides, and improving sanitation, public safety and health. Food waste reduction can increase people's access to food, and it can feed more people instead of landfills.

An estimated 50 million Americans do not have access to enough food. In Iowa, one out of nine people are food insecure and one out of every seven children faces hunger issues.[2] Organizations such as T2T can help reduce food sent to landfills; in 2018 they rescued about 2.2 million pounds of food. T2T works with donors and agencies to fight food insecurity by providing vulnerable populations with greater access to food. Recipients of donated food include 50 area organizations from daycares to food pantries, and feed over 19,000 people a year or 13% of Johnson County's population. The Iowa City Landfill and Recycling Center is used by residential and commercial haulers in Johnson County, Kalona and Riverside. The landfill takes about 135,000 tons of trash each year of which approximately 36% is organic waste.[3,4]

Methodology

Project Scope

T2T environmental impacts were assessed for two sections of the food production life cycle: pre-retail and post-retail. For this analysis, pre-retail was defined as all food production processes prior to sale that lead to water and carbon inputs. This includes raising crops and livestock, transport of goods, and inputs used to cook food goods at retail for prepared foods. While T2T does not directly affect pre-retail environmental impacts through their operations, these impacts help contextualize food waste in terms of the large input of natural resources needed to produce food. Post-retail is defined as the balance of direct reduction of GHG emissions from diverting food waste out of the local landfill and the carbon emission inputs from T2T vehicular use in completing weekly routes.

Data Management

T2T provided the following: monthly and yearly data files containing weights of collected food in different categories, vehicle information for collection vans, and metadata for collection routes. Data were provided for 2017 and 2018, but route data were only available for certain months of 2018. A series of data cleaning and processing steps were necessary prior to quantitative analysis. All data management steps were performed using Python programming language and R Software. First, the Microsoft Excel spreadsheets containing monthly and yearly

operations information were converted into comma separated value documents with each step of the route parsed out (Appendix C). Locations were assigned index number identifiers, and string matching was used to cross-reference pickup and drop-off locations and route names in different files as names were not necessarily consistent across files (Appendix A). String matching was performed using the FuzzyWuzzy Python package, which calculates the Levenshtein distance between two string sequences.[5] Calculating Levenshtein distance allows for string matching even when the two strings are not identical (e.g. partial string matching). Matches that were less than an 85% match were checked manually to confirm validity of the match, and corrections to location names were made if necessary. The indexed route distances were calculated using the Google Maps API to find driving distances between GPS coordinates of latitude and longitude as decimals (Appendix D).

Pre-Retail

Water footprint values and GHG emissions for pre-retail were referenced from the literature. For consistency in methodology, comprehensive papers that included a wide variety of food items were used. Specific notes and references for each food item are provided in Appendix F. In the T2T data files, collected foods were categorized as follows: Bakery, Dairy, Produce, Meat, Deli/Italian, Entrée, Grocery, Beverage, Prepared Food, and Other. However, water and carbon footprint values are typically estimated for specific food items; to calculate the water and carbon footprints of food collected within these categories, common items were assigned to each category (Table 1). Deli/Italian, Entrée, and Prepared Food were agglomerated into one "Prepared" category, assuming some common ingredients in prepared foods. The "Other" category was not included in analysis, as it often consists of miscellaneous items that are not necessarily food items (e.g. flowers). For each food category, we created five different possible scenarios for food items that may be collected. These scenarios weighted each food item differently to better represent variability of foods collected within each category (Appendix E). For example, for one scenario of the meat category, we considered it to be 50% chicken, 20% beef, 15% pork and 15% turkey. After the total water inputs and carbon emissions were calculated for each scenario, and then the average of the five different scenarios were calculated to assess the overall impact of each major category.

Table 1. Breakdown of each food group

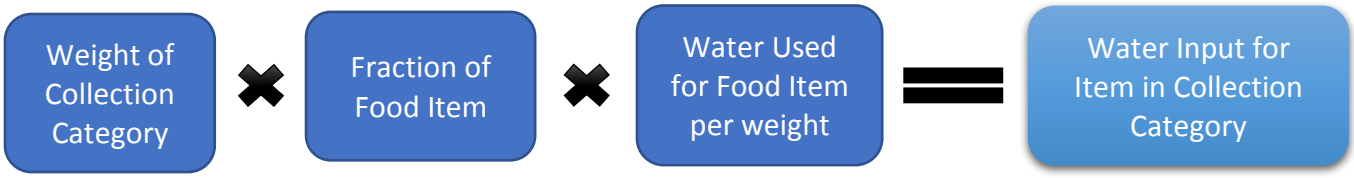
Meat	Produce	Dairy	Bakery	Grocery	Prepared	Beverage
Chicken	Apples	Eggs	Bread	Coffee	Wheat bread	Orange Juice
Beef	Bananas	Milk	Eggs	Cereal	Wheat flour	Apple Juice
Turkey	Oranges	Yogurt	Butter	Oil	Cheese	Tea
Pork	Potatoes	Cheese	Sugar	Oatmeal	Olive Oil	Milk
	Lettuce	Soy Milk	Wheat Flour	Beans	Tomatoes	Soy Milk
	Broccoli		Milk	Rice	Tomato Puree	
	Strawberries			Nuts	Lettuce	
	Avocados			Dry Pasta	Onions	
	Carrots			Juice	Dry pasta	
	Corn				Chicken	
					Beef	
					Pork	
					Ham	

Assumptions and Uncertainties

For pre-retail life cycle analyses (LCA), there can be high levels of variability of water inputs and carbon emissions depending on location of food production. The calculated values for these inputs are estimates based on literature values and do not necessarily reflect the exact inputs of producing the foods collected by T2T because it is beyond the scope of our study to trace production sources of foods collected. Additional sources of uncertainty include those on the data entry side. All data were manually inputted by different T2T volunteers, who may enter the same food item into different categories, which may lead to inconsistency in the data. Food items selected for each category (Appendix E) were assumed to be representative for that category throughout the year; realistically, there could be a high level of variability in food collected within each category. For example, the produce category may see a higher level of avocados and strawberries during the summer (growing season) as opposed to other types of food collected in the winter. In addition, not all food items collected are represented within the category. While weighting factors were meant to capture variability, not all situations may be represented in the five scenarios for collection categories.

Water Inputs

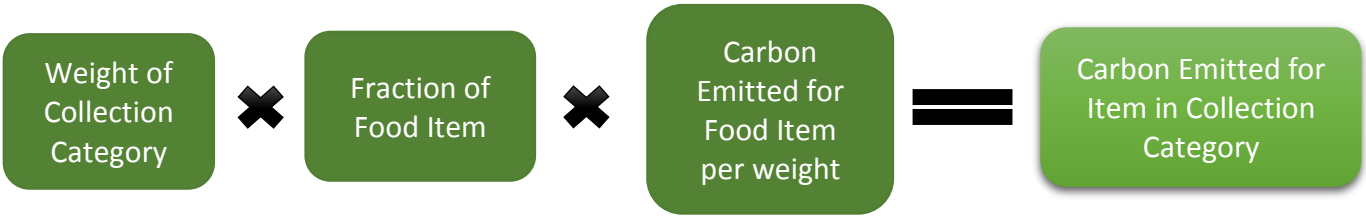
The total water footprint was considered as the sum of the grey, green and blue water footprints. The U.S. weighted averages of grazing, mixed, and industrial values were used for animal products while global averages were used for crops and crop products.[6,7] The water input of each food item was calculated for each scenario using the following equation:



The water inputs were then summed for each scenario and the category totals were averaged for the five scenarios.

GHG Emissions

Unless otherwise stated in Appendix F, global mean values were used from Poore et al. for carbon footprint factors of food items. This paper aggregated and analyzed a large set of LCA publications on a variety of foods to investigate variability and trends in environmental costs of producing food.[8] The carbon emissions from producing each food item in the categories were calculated using the following equation:



The carbon emissions were then summed for each scenario and the category totals were averaged for the five scenarios. For the "Prepared" food category, an additional input of cooking at retail was considered for the following meat items: chicken, beef, pork.[9] These inputs were calculated for each scenario and added to the GHG emission total for that scenario prior to calculating the total average for the "Prepared" category.

Post-Retail

To calculate post-retail emissions saved from the landfill, we calculated the amount of methane that would have been emitted for the mass of food collected. Assumptions and calculations followed those detailed in Chapter 10 of the *Environmental Engineering: Fundamentals, Sustainability, Design* textbook by Mihelcic and Zimmerman for methane emissions from organic waste in landfills.[10] First, the total mass of food collected was converted into a dry mass, assuming a 70% moisture content of food. The total masses of carbon, hydrogen, oxygen, and nitrogen in the food were calculated from the dry mass. We assume 60% of food waste is decomposed within the landfill. The moles methane produced per kg of food is calculated from the following equation:

$$\frac{4a + b - 2c - 3d}{8} = moles\ CH_4$$

Where a, b, c, and d are the masses of degraded carbon, hydrogen, oxygen, and nitrogen, respectively. The moles methane are then converted to a volume of methane using the ideal gas law (0.0224 m^3 gas/mole gas). Since the Johnson County landfill has an on-site gas recovery facility, we assumed that 80% of landfill gas is collected, and the remaining 20% is uncollected in the landfill. Of the 20% that is not collected, approximately 20% is oxidized within the landfill to produce methane emissions (overall, 16% of the original moles methane calculated is assumed to be emitted from the landfill.) The volume methane emitted is then converted to CO_{2e} by using a global warming potential of 25 for methane.

Route-based Calculations

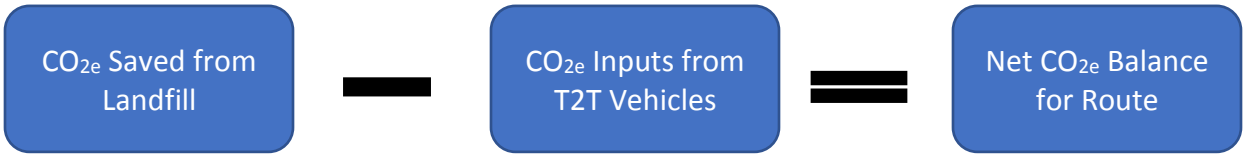
The CO_{2e} emissions from vehicles were calculated for each route using the metadata for the vehicles T2T uses and the driving distance of each route. Routes change every day of the week, and they vary from month to month. First, we used the GPS decimal coordinates for each pickup and drop-off locations (organized by route) to determine the distance traveled on each route using the Google Maps Application Programing Interfaces (API). The Google Maps API calculates driving distance, rather than the Euclidian distance (the shortest distance between two points). The route includes the distance from the T2T headquarters to the first pickup location and the distance from the last drop-off location back to the T2T headquarters. The

calculator uses the *distance_matrix* function from the Python Google Maps library. Each distance is multiplied by the corresponding Tailpipe CO_{2e} Emissions factor for the T2T van used for each route to calculate the route's emissions (Table 2). We calculated the number of times a route was driven for each month to obtain its total monthly emissions number.

Table 2. Vehicle Emission Factors

Vans	Year	Make	Emissions Factor
#1 Old Ford	1998	Ford	684
#2 Ford	2006	Ford	592
#3 Chevy	2008	Chevy	468
#4 3500	2013	Nissan	363
#5 200	2014	Nissan	363
#6 2500	2014	Nissan	363
#7 N200	2016	Nissan	353
#8 Transit	2016	Ford	548
Penske	2018	Penske	246

To match the driving distance calculations, we calculated the monthly sum of food collected for each route on a given weekday. We then calculated the CO_{2e} emissions that were saved using the same assumptions and equations as previously described. To find the net CO_{2e} for a route, we finally calculated:



The equation follows that a positive net balance indicates more CO_{2e} saved from the landfill than emitted by driving T2T vehicles (positive impact), and a negative net balance indicates that more CO_{2e} was emitted from driving T2T vehicles than saved from the landfill (negative impact).

Results and Discussion

The total weight of food collected in 2017 was 2,037,151 lb, and in 2018 was 2,396,094 lb. The distribution of weight within each T2T collection category for each year is displayed in Figures 2 and 3. The top three collection categories in 2017 were Bakery, Dairy, and Grocery, whereas the top three categories in 2018 were Bakery, Dairy, and Produce. Bakery was consistently one of the top collection categories, which is likely because many retail locations have baked fresh daily goods that are not sold the following day.

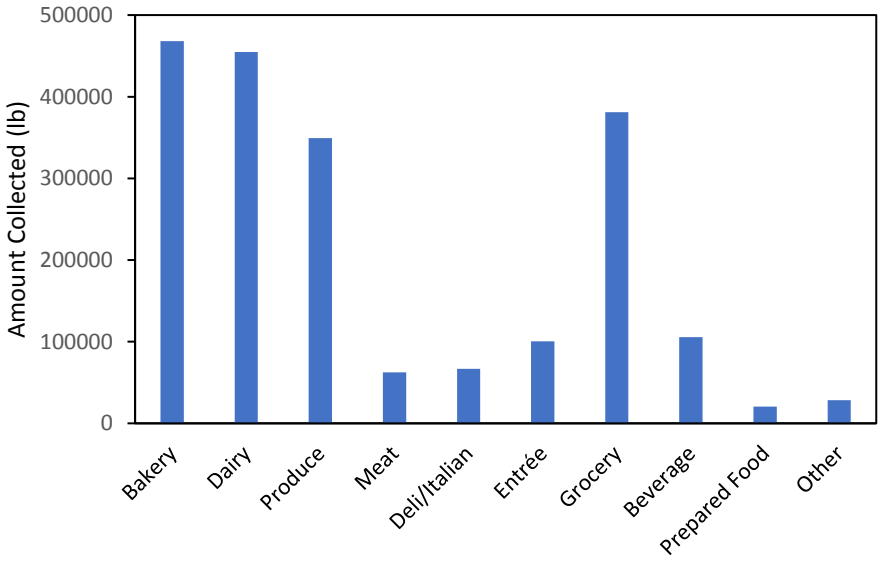


Figure 2. Distribution of the total weight of food collected in each category in 2017.

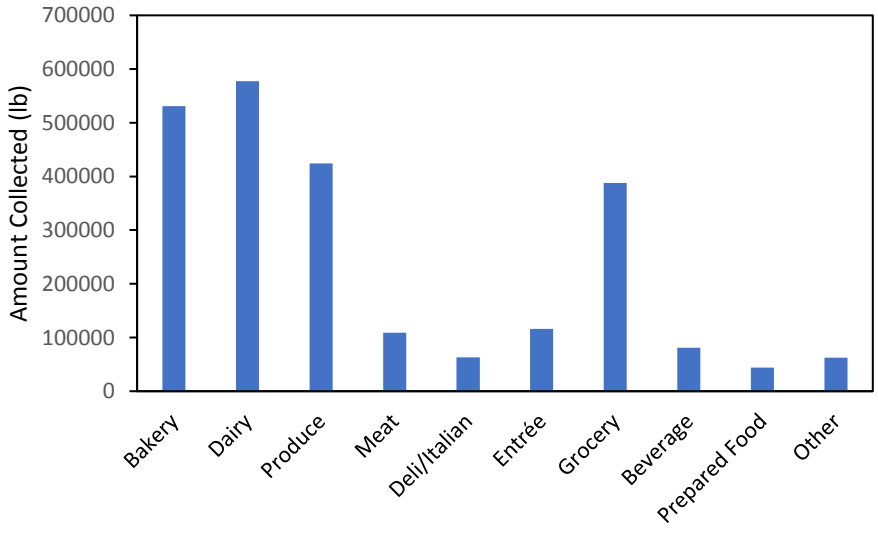


Figure 3. Distribution of the total weight of food collected in each category in 2018.

Pre-Retail

Water Footprint

The total embedded water for foods collected in 2017 was 2,831,705 m^3. This total value is the sum of categorical averages of the five scenarios detailed in Appendix E. The categorical averages of embedded water for the scenarios are displayed in Figure 4.

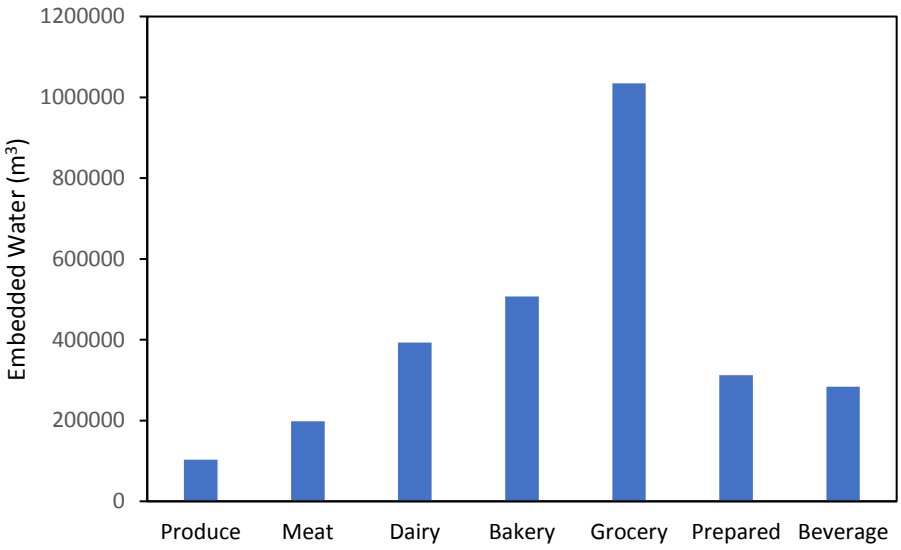


Figure 4. Average embedded water for each T2T collection category in 2017. Each average is for total embedded water for the five scenarios.

The Grocery category has the highest average total embedded water value of 1,034,266 m^3. T2T collects a larger amount (in weight) for the grocery category, and it includes food items with high embedded water footprints, for example, coffee. While meat items have high embedded water footprints, the collection weight was less than other categories, which resulted in a relatively low average total embedded water for the meat category. The total embedded water for food collected in 2018 was 3,536,867 m^3. The categorical averages of embedded water are displayed in Figure 5. Once again, Grocery had the highest average total embedded water value.

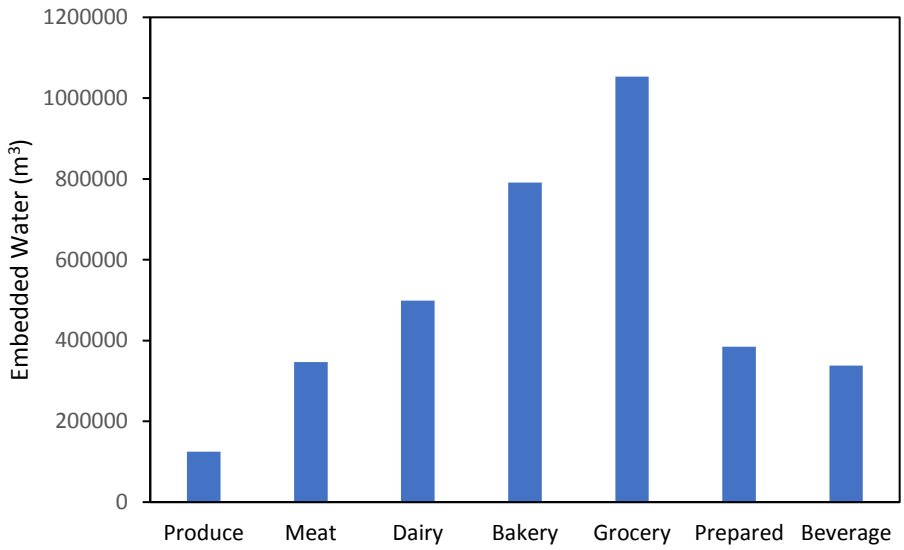


Figure 5. Average embedded water for each T2T collection category in 2018. Each average is for total embedded water for the five scenarios.

GHG Emissions

The total GHG emissions for producing food that was collected in 2017 was 5,459 tons CO_{2e}. This total value is the sum of categorical averages of the five scenarios detailed in Appendix E. The categorical averages of GHG emissions for the scenarios are displayed in Figures 6 and 7. The highest GHG emissions from producing food was in the Dairy category, with an average total of 1,466 tons CO_{2e} (Figure 5). High collection weight within the Dairy category as well as the high carbon footprint of specific items (e.g. cheese) cause large GHG emissions. In contrast to the water footprint, the Meat category has a high total average GHG emission relative to other categories considered in this report. The disproportionately high carbon footprint of meat items caused this value to be comparatively elevated; for example, beef has a 99.5 kg CO_{2e}/kg global average carbon footprint factor, which is the highest carbon footprint considered in this analysis.

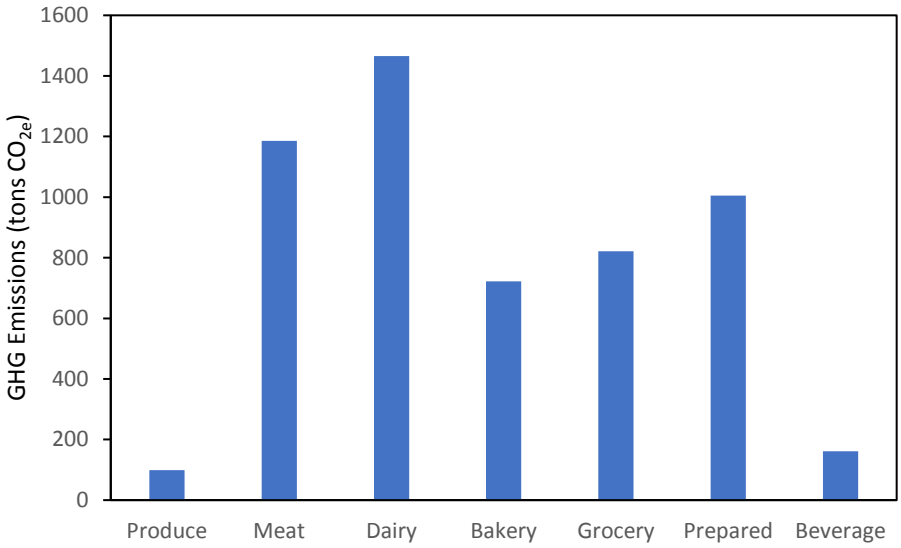


Figure 6. Average GHG emissions for each T2T collection category in 2017. Each average is for total GHG emissions for the five scenarios.

In 2018, the total GHG emissions for producing collected foods was 6,952 tons CO_{2e} (Figure 6). T2T collected more meat in 2018 than in 2017, resulting in the Meat category having the highest GHG emissions associated with pre-retail, followed by the Dairy category.

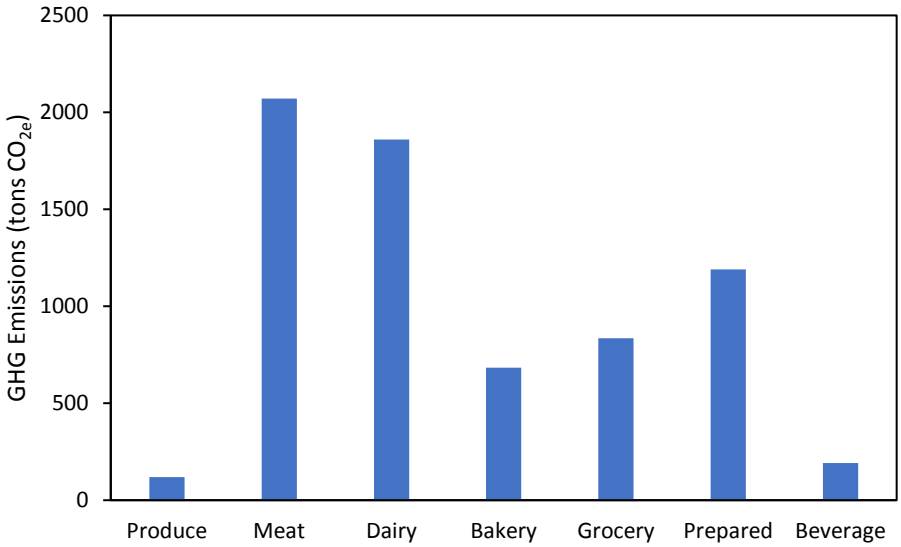


Figure 7. Average GHG emissions for each T2T collection category in 2018. Each average is for the total GHG emissions for the five scenarios.

Post-Retail

In 2017, the amount of carbon emissions that would have occurred if food had not been collected by T2T was estimated to be 26.9 tons CO_{2e}. In 2018, the carbon emissions were

estimated to be 31.7 tons CO_{2e}. From the Iowa City Climate Action Plan, total municipal emissions in 2015 were 44,194 tons CO_{2e}.[4] Approximately 54% of these emissions were due to solid waste decomposition in the landfill (23,997 tons CO_{2e}). Although the fraction of emissions saved from the landfill is relatively small, the weight of collected food represents just a small proportion of total tons of organic waste collected by the landfill (approximately 2-3%). Additionally, comparisons reported here are approximations because data from the Johnson County Landfill for 2017 and 2018 were not available.

Route-based Calculations

Route-based balances were calculated for each month in 2018 (Table 3). Positive balances indicate more emissions saved from landfill than emitted from driving T2T vehicles for that route. On average, the US Foods-UNFI route had the best emissions balance and it had the highest total emissions saved from the landfill (after subtracting the vehicle's emissions). Some routes, such as the Iowa City Walmart/Aldi route, had consistently negative emission balances, which may be from the specific car's emissions for that route (0.02 ton CO_{2e}) which were higher compared to other routes.

Table 3. The average monthly balance of emissions saved with T2T vehicle emissions deducted and the yearly total balance for 2018.

Route Name	Average Monthly Emissions Balance (tons CO_{2e})	Yearly Total Emissions Balance (tons CO_{2e})	Yearly Total Emissions Balance (pounds CO_{2e})
Coralville	0.10	1.20	2400
Costco	0.10	1.14	2280
Dorms-UIHC	0.01	0.11	220
FM-Kalona	-0.01	-0.08	-160
Fareway West	0.03	0.16	320
IC Kum-Go	0.00	0.00	0
IC Wal-Aldi	-0.04	-0.53	-1060
Iowa City	0.15	1.78	3560
NDH-LM	-0.02	-0.21	-420
School District	0.00	0.01	20
Trader Joe's	0.14	1.68	3360
US-UNFI	0.77	9.23	18469
Waterfront HyVee	0.25	3.05	6100
T2T Total (Sum)	**1.48 tons CO_{2e}**	**17.54 tons CO_{2e}**	**35,089 lbs. CO_{2e}**

Conclusion

Our pre-retail analysis confirms the large amount of resources needed to produce food. T2T's operations help ensure that more food goes towards feeding people so that these resource inputs are not "wasted". Understanding the amount of resources that go into producing food can help people understand the environmental impacts of the food they eat.

Our post-retail findings suggest that T2T has a net positive environmental impact for GHG emissions. In both 2017 and 2018, the Nissan vehicles emit less CO_{2e} than the Ford Transit models. For that reason, our recommendation is to choose Nissan vans when utilizing vehicles for route delivery. In addition, we would suggest purchases cars that have a lower emission rate. Implementing these practices would help to further decrease T2T's carbon footprint and their environmental impact in the long run. Additionally, the shortest routes with larger donations are the most efficient, which is expected. The yearly total emissions saved after T2T vehicular inputs is equivalent to the CO_2 emissions from 17,400 pounds of coal burned or more than 2 million smartphones charged or about 39,000 miles driven by an average passenger vehicle.[11]

References

(1) US EPA. Food Recovery Hierarchy https://www.epa.gov/sustainable-management-food/food-recovery-hierarchy (accessed Jun 5, 2019).

(2) Feeding America: Hunger in Iowa https://www.feedingamerica.org/hunger-in-america/iowa (accessed Jun 5, 2019).

(3) Landfill and Recycling Center https://www.icgov.org/city-government/departments-and-divisions/landfill-and-recycling-center (accessed Jun 5, 2019).

(4) *Iowa City Climate Action and Adaptation Plan*; 2018.

(5) Gilleland, M. Levenshtein Distance https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein Distance.htm (accessed May 9, 2019).

(6) Mekonnen, M.; Hoekstra, A. Y. *The Green, Blue and Grey Water Footprint of Animals and Animal Products*; Unesco-IHE Institute for Water Education: Delft, 2010.

(7) Mekonnen, M. M.; Hoekstra, A. Y. The Green, Blue and Grey Water Footprint of Crops and Derived Crop Products. *Hydrol. Earth Syst. Sci* **2011**, *15*, 1577–1600. https://doi.org/10.5194/hess-15-1577-2011.

(8) Poore, J.; Nemecek, T. Reducing Food's Environmental Impacts through Producers and Consumers. *Science (80-.).* **2018**, *360* (6392), 987–992. https://doi.org/10.1126/SCIENCE.AAQ0216.

(9) Hamerschlag, K.; Venkat, K. *Meat Eaters Guide to Climate Change and Health Life Cycle Assessments: Methodology and Results*; 2011.

(10) Mihelcic, J. R.; Zimmerman, J. B. *Environmental Engineering: Fundamentals, Sustainability, Design*, 2nd ed.; John Wiley & Sons, Inc., 2013.

(11) US EPA. Greenhouse Gas Equivalencies Calculator https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator (accessed Jun 5, 2019).

(12) Vergé, X. P. C.; Maxime, D.; Dyer, J. A.; Desjardins, R. L.; Arcand, Y.; Vanderzaag, A. Carbon Footprint of Canadian Dairy Products: Calculations and Issues. *J. Dairy Sci.* **2013**, *96* (9), 6091–6104. https://doi.org/10.3168/JDS.2013-6563.

(13) Nette, A.; Wolf, P.; Schlüter, O.; Meyer-Aurich, A.; Nette, A.; Wolf, P.; Schlüter, O.; Meyer-Aurich, A. A Comparison of Carbon Footprint and Production Cost of Different Pasta Products Based on Whole Egg and Pea Flour. *Foods* **2016**, *5* (4), 17. https://doi.org/10.3390/foods5010017.

(14) Hu, A. H.; Chen, C.-H.; Huang, L. H.; Chung, M.-H.; Lan, Y.-C.; Chen, Z. Environmental Impact and Carbon Footprint Assessment of Taiwanese Agricultural Products: A Case Study on Taiwanese Dongshan Tea. *Energies* **2019**, *12* (1), 138. https://doi.org/10.3390/en12010138.

Appendix A: Python code to match location names

```
"""
Created on Sat Apr  6 16:16:55 2019

@author: Erica
"""

import pandas
from fuzzywuzzy import process
import numpy
import os

COL_NAMES = ['P1', 'P1id', 'P2', 'P2id', 'P3', 'P3id', 'P4', 'P4id', 'P5', 'P5id', 'P6', 'P6id', 'D1', 'D1id', 'D2', 'D2id', 'D3', 'D3id', 'D4', 'D4id', 'D5', 'D5id', 'D6', 'D6id']

def read_file(fname):
    """ Reads in csv file """

    if os.path.exists(fname):
        file_df = pandas.read_csv(fname)
        return file_df

    else:
        print("Error: Route data file does not exist. Please check file path")
        return

def create_route_dfcopy(route_df):
    """ Creates a copy of the route dataframe to populate with matched location ID numbers

    Inputs
    ------
    route_df : pandas DataFrame

    Returns
    -------
    pandas DataFrame
    """

    route_df_copy = route_df.copy()
    # Iterate through global variable list column names and insert id columns
    for i in range(len(COL_NAMES)):
        colname = COL_NAMES[i]
        if "id" in colname:
            # all IDs will have a default of -1
            route_df_copy.insert(i, colname, numpy.nan, allow_duplicates=True)
        else:
            continue

    return route_df_copy

def match_pickuploc_id(route_df, loc_df):
    """ Uses fuzzywuzzy string matching to match location id in route file to id in the indexing file

    Inputs
    ------
    route_df : pandas DataFrame
```

```
    loc_df : pandas DataFrame

    Returns
    -------
    matched_df : pandas DataFrame
    unmatched_set : set
    """

    # Creating a dictionary from the indexed locations mapping loc name to ID
    loc_dict = dict(zip(loc_df.Location, loc_df.ID))
    loc_dict = {k.lower(): locid for k, locid in loc_dict.items()}
    loc_names = loc_dict.keys()
    bad_list = []
    route_columns = route_df.columns
    matched_df = create_route_dfcopy(route_df)
    counter = 0

    # Iterate through the route ID file rows to try and match location names
    for row in route_df.itertuples(index=False):

        # Go cell by cell to match each location with its best match in the locations file
        for icell in range(len(route_columns)):
            icolname = route_columns[icell]
            cell = row[icell]

            # Matching the location name in route file to closest name in locations file
            if cell != "NA" and type(cell) is str:

                # Pre-treating location names by making lowercase and removing leading and ending whitespaces

                mod_loc = cell.lower()
                mod_loc = mod_loc.lstrip()
                mod_loc = mod_loc.rstrip()

                # Fixing specific cases where matching won't work
                if mod_loc[:2] == "cv":
                    mod_loc = "coralville" + mod_loc[2:]

                if mod_loc == "nl food pantry":
                    mod_loc = "north liberty" + mod_loc[2:]

                if mod_loc == "dvip":
                    mod_loc = "domestic violence intervention program"

                if "community serving community" in mod_loc:
                    mod_loc = "community serving community child care center"

                    if mod_loc == "coralville kum&go":
                        mod_loc = "2nd st. kum & go"

                    # Find the closest match using fuzzywuzzy
                    high_match = process.extractOne(mod_loc, loc_names)

                    # set a minimum match threshold below which the match is considered unacceptable. 100 is a perfect match
                    min_match = 85
                    if high_match[1] <= min_match:
                        bad_list.append((mod_loc, high_match))
```

```
                            continue

                        # if match is considered acceptable, modify the match dataframe to have the location index
                        matched_id = loc_dict[high_match[0]]
                        idcolname = icolname + 'id'
                        matched_df.at[counter, idcolname] = matched_id

                    else:
                        continue
                # increase the row counter
                counter += 1

            # Create a unique set of the unacceptable matches
            unmatched_set = set(bad_list)

            return matched_df, unmatched_set


        def main():
            tt_homelocfile = "C:/Users/Erica/OneDrive - University of Iowa/2019/Classes/Spring2019/CEE5993_CCPSD/Data/eawcopy_Locations_Indexed_v2.csv"
            tt_homeroutefile = "C:/Users/Erica/OneDrive - University of Iowa/2019/Classes/Spring2019/CEE5993_CCPSD/Data/eawcopy_Route_Total_ID.csv"
            tt_officelocfile = "C:/Users/ewiener/OneDrive - University of Iowa/2019/Classes/Spring2019/CEE5993_CCPSD/Data/eawcopy_Locations_Indexed_v2.csv"
            tt_officeroutefile = "C:/Users/ewiener/OneDrive - University of Iowa/2019/Classes/Spring2019/CEE5993_CCPSD/Data/eawcopy_Route_Total_ID.csv"
            tt_matchedfile = "C:/Users/Erica/OneDrive - University of Iowa/2019/Classes/Spring2019/CEE5993_CCPSD/Data/eaw_matchedindices_042819.csv"
            # When running from office
            #route_df = read_file(tt_officeroutefile)
            #loc_df = pandas.read_csv(tt_officelocfile, usecols=[0,1])

            # When running from home
            route_df = read_file(tt_homeroutefile)
            loc_df = pandas.read_csv(tt_homelocfile, usecols=[0,1])

            ploc = route_df.columns.get_loc("P1")
            dloc = route_df.columns.get_loc("D6")
            route_df_mod = route_df.iloc[:,ploc:dloc+1]

            print("matching indices now")
            matched_df, unmatched_set = match_pickuploc_id(route_df_mod, loc_df)
            print("finished matching successfully")

            print("unmatched locations are: ", unmatched_set)
            print("writing matched file")
            matched_df.to_csv(tt_matchedfile, index=False)

        main()
```

Appendix B: Python code for analyzing monthly files

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 20 13:09:38 2019

@author: ewiener
"""

import pandas
import os
import calendar
import math

# GLOBAL VARIABLES

PICKUP_LOCNAMES = []
WEEKDAY_DICT = {0: "mon", 1: "tue", 2: "wed", 3: "thu", 4: "fri", 5: "sat", 6: "sun"}
ROUTE_DICT = {'Waterfront': 'Waterfront Hyvee', 'Iowa City': 'Iowa City', 'Coralville':
 'Coralville', 'Costco': 'Costco', 'IC Walmart/Aldi': 'IC Wal-Aldi', 'N': 'NDH-
LM', 'Kalona': 'FM-
Kalona', "Trader Joe's": 'Trader Joes', 'School District': 'School District', 'Fareway
West': 'Fareway West'}
LB_TO_KG = 0.4536
MW_H = 1.008
MW_C = 12.011
MW_O = 15.999
MW_N = 14.007


def calculate_foodmethane(foodwt):
    """ Calculates the CO2e emitted from landfill given weight of food in lb

    Inputs
    ------
    foodwt : float

    Returns
    -------
    float
    """

    # percentage of elements in dry mass of food
    perc_C = 0.48
    perc_H = 0.064
    perc_O = 0.376
    perc_N = 0.026

    # convert food weight to kg and dry mass
    foodwt_kg_dry = 0.3 * foodwt * LB_TO_KG

    # calculate degraded element masses
    degr_C = 0.6 * perc_C * foodwt_kg_dry
    degr_H = 0.6 * perc_H * foodwt_kg_dry
    degr_O = 0.6 * perc_O * foodwt_kg_dry
    degr_N = 0.6 * perc_N * foodwt_kg_dry

    # calculate moles methane
    mol_C = degr_C * 1000.0 / MW_C
    mol_H = degr_H * 1000.0 / MW_H
```

```
    mol_O = degr_O * 1000.0 / MW_O
    mol_N = degr_N * 1000.0 / MW_N
    mol_methane = float((4*mol_C+mol_H-2*mol_O-3*mol_N) / 8.0)

    # calculate volume methane emitted and convert to CO2e
    vol_methane_emitted = 0.0024 * mol_methane * 0.16
    equiv_emitted = vol_methane_emitted/22.4*16.0*25.0

    return equiv_emitted

def calc_monthly_routemethane(route_sum_dict):
    """ Calculates methane emitted for each route by iterating through a dict for monthly data

    Inputs
    ------
    route_sum_dict : dict
        dict mapping route names to weekday sums of food

    Returns
    -------
    dict
    """

    route_methane_dict = {}
    for route in route_sum_dict.keys():
        if route not in route_methane_dict.keys():
            route_methane_dict[route] = {}
        for key in route_sum_dict[route].keys():
            methane = calculate_foodmethane(route_sum_dict[route][key])
            route_methane_dict[route][key] = methane

    return route_methane_dict

def get_routes_monthly(file_list):
    """ Returns a set of route name strings based on the sheets of monthly files """

    route_list = []
    for fname in file_list:
        f_monthlyobj = MonthlyData(fname)
        f_monthlyobj.read_monthly_files()

        print(fname)
        for route_name in f_monthlyobj.xl_file.keys():
            # checking if the sheet is a route sheet and not a totals sheet
            if "by date" in route_name.lower() or "by type" in route_name.lower():
                        continue

                    # checking that the route is not one of the special routes
                    if route_name != "FRANK" and route_name != "OTHER" and route_name != "Donors":
                        route_list.append(route_name)

            return set(route_list)

        def read_index_file(index_fname):

            if os.path.exists(index_fname):
                index_file = pandas.read_excel(index_fname)
            else:
                print("Error: Index file does not exist. Please check file path")
```

```
114.                index_file = {}
115.
116.            return index_file
117.
118.        def match_rname_rid(index_file):
119.            """ Create a dictionary that allows for matching routes between monthly file
    s and route data
120.
121.            Inputs
122.            ------
123.            index_file : Pandas object
124.                The route index file read in previously
125.
126.            Returns
127.            -------
128.            dict"""
129.
130.            rid_dict = {}
131.            for row in index_file.itertuples():
132.                rname = getattr(row, "Rname")
133.                rname_mod = rname.strip()
134.                rid = getattr(row, "ID")
135.                day = getattr(row, "Day")
136.                month = getattr(row, "month")
137.                year = getattr(row, "year")
138.                emissions = getattr(row, "Emissions")
139.                if rname_mod in ROUTE_DICT.keys():
140.                    rid_dict[(ROUTE_DICT[rname_mod].lower(), month.lower(), year, day)]
    = (rid, emissions)
141.                if rname_mod == 'UNFI/US Foods' or rname_mod == 'UNFI' or rname_mod == '
    US Foods':
142.                    rid_dict[('us-unfi', month.lower(), year, day)] = (rid, 5.73)
143.
144.            day_list = ['mon', 'tue', 'wed', 'thu', 'fri']
145.            month_list = ['jan', 'feb', 'mar', 'apr', 'may', 'aug', 'sep', 'nov', 'dec']

146.            for month in month_list:
147.                for day in day_list:
148.                    rid_dict[('dorms-uihc', month, 2018, day)] = (3000, 2.3595)
149.                    rid_dict[('ic kum-go', month, 2018, day)] = (3000, 1.089)
150.
151.            return rid_dict
152.
153.        def get_carbon_inputs(route_count_dict, rid_dict, month, year):
154.            """ Calculates the amount of carbon emissions associated with driving routes
     """
155.
156.            carbon_dict = {}
157.            if month <= 3:
158.                route_mon = 2
159.            if month > 3 and month <= 6:
160.                route_mon = 5
161.            if month > 6 and month <= 8:
162.                route_mon = 7
163.            if month > 8 and month <= 12:
164.                route_mon = 9
165.            month_abbrev = calendar.month_abbr[route_mon]
166.            for route in route_count_dict.keys():
167.                if not route in carbon_dict.keys():
168.                    carbon_dict[route] = {}
169.                for weekday in route_count_dict[route].keys():
```

```
                    key = (route.lower(), month_abbrev.lower(), year, weekday)
                    if key not in rid_dict.keys():
                        continue
                    if not weekday in carbon_dict[route].keys():
                        carbon_dict[route][weekday] = 0
                    carbon = route_count_dict[route][weekday] * rid_dict[key][1]
                    carbon_dict[route][weekday] = carbon

            return carbon_dict

        def carbon_balance(input_dict, methane_dict):
            """ Calculate the balance of T2T inputs with emissions saved from landfill f
    rom food collected """

            route_balance_dict = {}
            bad_list = []
            for route in methane_dict.keys():
                if not route in route_balance_dict.keys():
                    route_balance_dict[route] = 0
                total_balance = 0
                for weekday in methane_dict[route].keys():
                    if weekday not in input_dict[route].keys():
                        continue
                    if math.isnan(methane_dict[route][weekday]):
                        bad_list.append(("monthly", route, weekday))
                    if math.isnan(input_dict[route][weekday]):
                        bad_list.append(("routes", route, weekday))

                    balance = methane_dict[route][weekday] - input_dict[route][weekday]

                    total_balance += balance
                route_balance_dict[route] = total_balance

            print(set(bad_list))
            return route_balance_dict

        class MonthlyData(object):

            """
            """

            def __init__(self, data_fname):
                """"""

                # Check if the file names are valid
                self.valid_datafname = False
                self.data_fname = data_fname
                self.xl_file = {}

                if os.path.exists(data_fname):
                    self.valid_datafname = True
                else:
                    if not os.path.exists(data_fname):
                        print("Error: Monthly data file does not exist. Please check fil
    e path")
                        return


            def read_monthly_files(self):
                """
                Reads the monthly data file into an OrderedDict object
```

```
228.                 """
229.
230.                 if self.valid_datafname == False:
231.                     return
232.
233.                 self.xl_file = pandas.read_excel(self.data_fname, sheet_name=None)
234.
235.
236.             def monthly_typesum(self, type_bool):
237.                 """ Calculates the monthly sum of weight of food collected for each rout
    e and sorts into weekdays.
238.
239.                 Inputs
240.                 ------
241.                 type_bool : boolean
242.                     True if summing based on types, False if doing a total sum
243.
244.                 Returns
245.                 -------
246.                 dict
247.                     Maps the route name to a dictionary mapping (weekday, food type) tup
    les  or just weekdays to the collected sum for the month
248.                    {route_str: {(weekday_str, type_str): sum, # times collected}}
249.                 """
250.
251.                 typesum_dict = {}
252.                 route_count_dict = {}
253.
254.                 # Iterate through each sheet in the monthly data file
255.                 for route_name in self.xl_file.keys():
256.                     food_sheet_dict = {}
257.                     count_sheet_dict = {}
258.
259.                     # checking if the sheet is a route sheet and not a totals sheet
260.                     if "by date" in route_name.lower() or "by type" in route_name.lower(
    ):
261.                         continue
262.
263.                     # checking that the route is not one of the special routes
264.                     if route_name != "FRANK" and route_name != "OTHER" and route_name !=
     "Donors":
265.                         route_sheet = self.xl_file[route_name]
266.                         if not route_name in typesum_dict.keys():
267.                             typesum_dict[route_name] = {}
268.
269.                         # iterate through each row of the sheet and find the sum of each
     type
270.                         day_set = []
271.                         for row in route_sheet.itertuples():
272.                             date_ts = getattr(row, "Date")
273.                             type_name = getattr(row, "Type")
274.                             pickup = getattr(row, "Pickup")
275.                             if type(date_ts) is not float and type(date_ts) is not str:

276.                                 weekday = date_ts.weekday()
277.                             else:
278.                                 continue
279.
280.                             # Finding the sum based on the weekday since routes are the
    same each weekday
281.                             if weekday in WEEKDAY_DICT.keys():
```

```
                                if type_bool == True:
                                    row_key = (WEEKDAY_DICT[weekday], type_name)
                                else:
                                    row_key = WEEKDAY_DICT[weekday]
                                collected_amt = getattr(row, "Collected")
                                if not row_key in food_sheet_dict.keys():
                                    food_sheet_dict[row_key] = 0
                                if not math.isnan(collected_amt):
                                    food_sheet_dict[row_key] += collected_amt
                                else:
                                    food_sheet_dict[row_key] += 0

                                if not row_key in count_sheet_dict.keys():
                                    count_sheet_dict[row_key] = 0
                                if date_ts not in day_set:
                                    count_sheet_dict[row_key] += 1
                                    day_set.append(date_ts)

                        # add the route to sums to the dictionary
                        typesum_dict[route_name] = food_sheet_dict
                        route_count_dict[route_name] = count_sheet_dict

                return typesum_dict, route_count_dict


        def main():

            #main fn here

            # Retrieve all monthly files in directory
            monthly_file_dirpath = "C:/Users/ewiener/OneDrive - University of Iowa/2019/Classes/Spring2019/CEE5993_CCPSD/Data/Monthly/2018/"
            monthly_filelist = [os.path.join(monthly_file_dirpath, fname) for fname in os.listdir(monthly_file_dirpath) if os.path.isfile(os.path.join(monthly_file_dirpath, fname))]
            print(monthly_filelist)

            tt_officeindicesfile = "C:/Users/ewiener/OneDrive - University of Iowa/2019/Classes/Spring2019/CEE5993_CCPSD/Data/eaw_route_inputs.xlsx"
            index_file = read_index_file(tt_officeindicesfile)
            rid_dict = match_rname_rid(index_file)
            type_bool = False
            agg_balance_list = []

            # iterate through all monthly files in the given directory
            for fname in monthly_filelist:
                datestr = fname[-11:-5]
                month_num = int(datestr[:2])
                year_num = int(datestr[-4:])
                month_abbrev = calendar.month_abbr[month_num]
                print(month_abbrev)

                tt_obj = MonthlyData(fname)

                tt_obj.read_monthly_files()
                monthlysumdict, count_sheet_dict = tt_obj.monthly_typesum(type_bool)

                print(count_sheet_dict)
                monthlymethanedict = calc_monthly_routemethane(monthlysumdict)
                print(monthlymethanedict)
```

```
338.                route_carbon_dict = get_carbon_inputs(count_sheet_dict, rid_dict, month_
    num, year_num)
339.                print(route_carbon_dict)
340.                balance_dict = carbon_balance(route_carbon_dict, monthlymethanedict)
341.                #print(balance_dict)
342.                agg_balance_list.append(balance_dict)
343.
344.            # aggregate all monthly data calculated balances
345.            agg_df = pandas.DataFrame(agg_balance_list)
346.
347.            # write aggregated data
348.            final_fname = "C:/Users/ewiener/OneDrive - University of Iowa/2019/Classes/S
    pring2019/CEE5993_CCPSD/Data/eaw_balance.csv"
349.            agg_df.to_csv(final_fname)
350.
351.
352.
353.            tt_officetestfile = "C:/Users/ewiener/OneDrive - University of Iowa/2019/Cla
    sses/Spring2019/CEE5993_CCPSD/Data/Monthly/2018/TTMonthly_022018.xlsx"
354.
355.
356.        main()
```

Appendix C: R code for extracting route information

```
##########################################
####### T2T Impact Study Data Any ########
#######   Vehicle Info CSV Files  ########
##########################################

# This script extracts each route for each month along with vehicle IDs

rm(list=ls(all=TRUE)) ; cat("\014")

library(sp); library(raster); library(ggplot2); library(rgdal);
library(plyr); library(tidyr); library(data.table);

pathL =
"D:/Courses/CEE_5993_Com_Centered_Prob_Solv_Design/Vehicle_Information_CSV_Files"  #Use
pathfile rather than setwd if saving files in different locations
all = list.files(path = pathL,pattern="*.csv",full.names=T)
#number = 2

FUN1 <- function(number){ # fun to extract information from each file within fold
  car=read.csv(file = all[number],header = F ,sep=",")

  month =  as.character(substr(unlist(strsplit(all[number],"/"))[[5]],27,29))
  year = as.numeric(substr(unlist(strsplit(all[number],"/"))[[5]],31,34))
  carcar = car[,-3]
  mon = car[1:11,]
  tue = car[13:24,]
  wed = car[25:36,]
  thu = car[37:48,]
  fri = car[49:60,]
  sat = car[61:72,]

  week <- list(mon,tue,wed,thu,fri,sat)

  days = c("mon","tue","wed","thu","fri","sat")

  weekly <- list()

  list <- list()
  #drops <- list()
  #picks <- list()

  for(d in 1:length(days)){

  #d = 3
  day = week[[d]]

  day$Day = paste(days[d])
  #dayday = day[-c(1,2),]
  colnames(day) = c("Route","PickUp","DropOff","Additional","Vols","Van","Day")
  day$Route = as.character(day$Route)
```

```
    day$Rnumb = sapply(strsplit(as.character(day$Route), "\\."), `[`, 1)
    day$Rname = sapply(strsplit(as.character(day$Route), "\\."), `[`, 2)

    if(any(is.na(day$Rname))){day <- day[-which(is.na(day$Rname)),]} # remove values with
  NA


    day$P1 = sapply(strsplit(as.character(day$PickUp), "\\+"), `[`, 1)
    day$P2 = sapply(strsplit(as.character(day$PickUp), "\\+"), `[`, 2)
    day$P3 = sapply(strsplit(as.character(day$PickUp), "\\+"), `[`, 3)
    day$P4 = sapply(strsplit(as.character(day$PickUp), "\\+"), `[`, 4)
    day$P5 = sapply(strsplit(as.character(day$PickUp), "\\+"), `[`, 5)
    day$P6 = sapply(strsplit(as.character(day$PickUp), "\\+"), `[`, 6)

    day$D1 = sapply(strsplit(as.character(day$DropOff), "\\+"), `[`, 1)
    day$D2 = sapply(strsplit(as.character(day$DropOff), "\\+"), `[`, 2)
    day$D3 = sapply(strsplit(as.character(day$DropOff), "\\+"), `[`, 3)
    day$D4 = sapply(strsplit(as.character(day$DropOff), "\\+"), `[`, 4)
    day$D5 = sapply(strsplit(as.character(day$DropOff), "\\+"), `[`, 5)
    day$D6 = sapply(strsplit(as.character(day$DropOff), "\\+"), `[`, 6)

    day$monthmonth = month
    day$yearyear = year

    #pickups = gather(day[,10:15], key ="Order",value="Location")
    #dropoffs = gather(day[,16:21], key ="Order",value="Location")

    #drops[[d]] <- dropoffs
    #picks[[d]] <- pickups
    list[[d]] <- day
    totals <- rbindlist(list)
    }
   # weekly <- rbindlist(weektotals)
  #  return(weekly)
    weekly[[d]] <- totals

  }

    twoT = lapply(1:length(all), function(i){ FUN1(i)  }) #Use Lapply for a list of the
  same length as x(temp) using function (FUN1)
    allT = rbindlist(twoT)

    allT$ID <- seq.int(nrow(allT))

    write.csv(allT,
  "D:/Courses/CEE_5993_Com_Centered_Prob_Solv_Design/Vehicle_Information_CSV_Files/Route_
  Total_ID.csv", row.names = F)

    #totals <- do.call(rbind,drops)
  #  unis <- trimws(totals$Location, which = "both")
   # unis <- as.data.frame(unis)
  #  unique(unis)

```

Appendix D: Python code for distance calculations

```
"""
Finds the distances of T2T routes and calculates the carbon dioxide equivalents
of the respective routes.

"""


import pandas as pd
import googlemaps
from numpy import loadtxt
import numpy as np

#function to upload files
def file_uploader (filename):
    global new_variable
    df = pd.DataFrame.from_csv(filename)
    df = df.values.tolist()
    new_variable = []
    for i in range(len(df)):
        cleanedList = [x for x in df[i] if str(x) != 'nan' or '']
        new_variable.append(cleanedList)
    return new_variable

#Import routes
file_uploader("pickups.csv")
pickups = new_variable
file_uploader("dropoffs.csv")
dropoffs = new_variable

#Import decimal coordinates of all locations
loc_name, loc_id, loc_add, loc_lat, loc_long = loadtxt(
        "location_coords.csv", delimiter=',', dtype={'names': ('location', 'id', 'addre
ss', 'lat', 'long'),
          'formats': ('|S15', np.int, '|S15', np.float, np.float)}, skiprows=1, unpack=
True)
loc_id = np.ndarray.tolist(loc_id)

#Reads in routes with van assignments
df = pd.read_csv('eaw_route_with_vans.csv')

#Reads vehicle emission rates
vehicles = pd.read_csv('vehicles.csv')


#Convert location indices to integer type
for i, pick in enumerate(pickups):
    for j in range(len(pickups[i])):
        pickups[i][j] = int(pickups[i][j])
    for k in range(len(dropoffs[i])):
        dropoffs[i][k] = int(dropoffs[i][k])

#Join pickups with dropoffs in consecutive order
routes = pickups
for i, route in enumerate(dropoffs):
    for place in route:
        routes[i].append(place)

#Perform request to use the Google Maps API web service
```

```
API_key = 'AIzaSyCHtMeESjdJFqFb8TP8UI-fEKPy4zQzhys' #enter Google Maps API key
gmaps = googlemaps.Client(key=API_key)

#Adjoin latitude and longitude of all locations into nested lists
coordinates = [[] for i in range(73)]
for i, coord in enumerate(loc_lat):
    coordinates[i].append(loc_lat[i])
    coordinates[i].append(loc_long[i])

#Convert nested lists to a list with the coordinates in the form as a tuple
#location coordinates
coordinates_tup = []
for i, coords in enumerate(coordinates):
    tup = (coordinates[i][0], coordinates[i][1])
    coordinates_tup.append(tup)

#route coordinates
route_coordinates = [[] for i in range(len(routes))]
for i, route in enumerate(routes):
    for j in range(len(route)):
        index = loc_id.index(routes[i][j])
        route_coordinates[i].append(coordinates_tup[index])

#Calculate route distances by finding distance from one location to the next,
#Once the last dropoff is reached, it adds the distance from the last location
#To T2T and the the first location from T2T
distance_list = []
table_to_table = (41.655520, -91.536250)
for i in range(len(route_coordinates)):
    #Initial distance from T2T
    origin = table_to_table
    destination = route_coordinates[i][0]
    result = gmaps.distance_matrix(origin, destination, mode='driving')["rows"][0]["ele
ments"][0]["distance"]["value"]
    for j in range(len(route_coordinates[i])):
        if j < (len(route_coordinates[i]) - 1):
            origin = route_coordinates[i][j]
            destination = route_coordinates[i][j+1]
            result += gmaps.distance_matrix(origin, destination, mode='driving')["rows"
][0]["elements"][0]["distance"]["value"]
        if j == len(route_coordinates[i]):
            #Ending distance back to T2T
            origin = route_coordinates[i][j]
            destination = table_to_table
            result += gmaps.distance_matrix(origin, destination, mode='driving')["rows"
][0]["elements"][0]["distance"]["value"]

           result = result / 1609.344 #Convert meters to miles
           distance_list.append(result)

       #Add distances to the route dataframe
       df['Distance'] = distance_list

       #Calculate route emissions
       emission_results = []
       for i in range(len(df['Van'])):
           for j in range(len(vehicles['Vans'])):
               if df['Van'][i] == vehicles['Vans'][j]:
                   distance = distance_list[i]
                   rate = vehicles['Tailpipe CO2 Emissions'][j]
                   emission = distance * rate / 1000
```

```
            emission_results.append(emission)

        #Export dataframe
        df['Emissions [CO2 kg]'] = emission_results
        df.to_csv('calculated_inputs.csv', sep=',', index=None)
```

Appendix E: Food Category Scenarios

Produce	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Apples	0.1	0.2	0.15	0.1	0.3
Bananas	0.1	0.2	0.15	0.3	0.1
Oranges	0.1	0.2	0.15	0.1	0.3
Potatoes	0.1	0.2	0.15	0.3	0.1
Lettuce	0.1	0.2	0.15	0.1	0.2
Broccoli	0.1	0	0.05	0.02	0
Strawberries	0.1	0	0.05	0.02	0
Avocados	0.1	0	0.05	0.02	0
Carrots	0.1	0	0.05	0.02	0
Corn	0.1	0	0.05	0.02	0
Total	1	1	1	1	1

Meat	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Chicken	0.25	0.5	0.3	0.2	0.7
Beef	0.25	0.2	0.3	0.5	0.3
Turkey	0.25	0.15	0.3	0.15	0
Pork	0.25	0.15	0.1	0.15	0
Total	1	1	1	1	1

Dairy	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Milk	0.2	0.5	0.25	0.3	0.6
Eggs	0.2	0.5	0.25	0.3	0.3
Yogurt	0.2	0	0.2	0.05	0.05
Cheese	0.2	0	0.2	0.3	0.05
Soy Milk	0.2	0	0.1	0.05	0
Total	1	1	1	1	1

Bakery	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Wheat Bread	1	0	0.7	0	0.6
Sugar	0	0.2	0.1	0.2	0.12
Eggs	0	0.2	0.04	0.15	0.04
Milk	0	0.2	0.03	0.05	0
Butter	0	0.2	0.03	0.1	0.04
Wheat Flour	0	0.2	0.1	0.5	0.2
Total	1	1	1	1	1

Grocery	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Coffee	0	0.2	0.05	0.111111	0
Cereal	0.2	0	0.1	0.111111	0.1
Canned beans	0.1	0.1	0.05	0.111111	0
Oatmeal	0.2	0.2	0.05	0.111111	0.1
Olive oil	0.05	0.05	0.1	0.111111	0.1
Rice	0.2	0.2	0.1	0.111111	0.3
Nuts	0.1	0.1	0.15	0.111111	0.1
Dry pasta	0.1	0.05	0.2	0.111111	0.3
Juice	0.05	0.1	0.2	0.111111	0
Total	1	1	1	1	1

Prepared	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Wheat bread	0.1	0.2	0	0.2	0.076923
Wheat flour	0.05	0	0.2	0.05	0.076923
cheese	0.05	0.05	0.15	0.05	0.076923
Olive Oil	0.05	0	0.1	0.05	0.076923
Tomatoes	0.1	0.15	0	0	0.076923
Tomato Puree	0.05	0	0.2	0.05	0.076923
Lettuce	0.15	0.3	0	0	0.076923
Onions	0.1	0.1	0.05	0.1	0.076923
Dry pasta	0.15	0	0	0.1	0.076923
Chicken	0.05	0	0	0.1	0.076923
Beef	0	0	0	0.1	0.076923
Pork	0.05	0	0	0.1	0.076923
Hams (cured)	0.1	0.2	0.3	0.1	0.076923
Total	1	1	1	1	1

Beverage	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Orange juice	0.2	0	0.3	0.2	0.4
Apple juice	0.2	0.3	0.4	0.2	0
Tea	0.2	0.3	0	0.3	0.2
Soy Milk	0.2	0	0.3	0.15	0.2
Milk	0.2	0.4	0	0.15	0.2
Total	1	1	1	1	1

Appendix F: Water and Carbon Footprint Factors

Product	Water Footprint (m^3/ton)	Pre-Retail GHG Emissions (kg CO2e/kg)	Reference	Notes
Apples	822	0.4	[7,8]	
Bananas	790	0.9	[7,8]	
Oranges	560	0.4	[7,8]	Carbon: "citrus fruits"
Potatoes	287	0.5	[7,8]	
Lettuce	237	0.5	[7,8]	Carbon: "Other vegetables"
Broccoli	285	0.5	[7,8]	Carbon: "brassicas"
Strawberries	347	1.5	[7,8]	
Avocados	1981	0.5	[7,8]	Carbon: "Other vegetables"
Carrots	195	0.4	[7,8]	Carbon: "Root vegetables"
Corn	1222	0.5	[7,8]	Carbon: "Other vegetables"
Chicken	2218	9.9	[6,8]	Carbon: "Poultry"
Beef	14191	99.5	[6,8]	Carbon: "Bovine meat (beef herd)"
Turkey	2218	9.9	[6,8]	Carbon: "Poultry"
Pork	5508	12.3	[6,8]	Water: "Swine cuts, fresh or chilled"
Milk	796	3.2	[6,8]	
Eggs	1566	4.7	[6,8]	
Yogurt	924	1.5	[6,12]	
Cheese	3945	23.9	[6,8]	
Soy Milk	3763	1	[7,8]	
Wheat Bread	1608	1.6	[7,8]	
Sugar	1782	3.2	[7,8]	Values are for refined cane sugar
Wheat Flour	1849	1.6	[7,8]	
Butter	4326	7.3	[6,12]	
Coffee	18925	28.5	[7,8]	Water: "coffee, roasted"
Cereal	1081	1.7	[7,8]	Water: "Maize (corn) groats and meal" Carbon: "Maize (meal)"
Canned beans	5053	1.8	[7,8]	Water: "beans, dry" Carbon: "other pulses"
Oatmeal	2416	2.5	[7,8]	Carbon: "oats, rolled or flaked grains"
Olive oil	14726	5.4	[7,8]	Water: "olive oil, refined"

Rice	1673	4.5	[7,8]	Water: “rice, paddy”
Nuts	12294	0.4	[7,8]	Water: average of almonds (shelled), walnuts (shelled), cashews, pistachios, and hazelnuts
Dry pasta	1849	1.79	[7,13]	Carbon: “Egg pasta”
Orange Juice	1000	0.4	[7,8]	Carbon: “other citrus”
Apple Juice	1100	0.4	[7,8]	Carbon: “apples”
Tea	8856	3.806	[7,14]	Carbon: sum of raw materials, manufacture, and distribution
Ham (cured)	5131	12.3	[6,8]	Water: “Hams, shoulders and cuts thereof, of swine bone in, cured”
Tomato puree	713	2.1	[7,8]	Water: “Tomato, puree” Carbon: “Tomatoes”
Tomatoes	214	2.1	[7,8]	
Onions	272	0.5	[7,8]	Water: “Onions (incl. shallots), green” Carbon: “Onions and Leeks”