Branch and Bound Algorithm Implementation on Zomato Restaurant Data for Food Tourism Planning

I Putu Gede Wirasuta
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
wirasutat@gmail.com 13517015@std.stei.itb.ac.id

***Abstract*—Food tourism has been on the rise lately along with the growth of food and restaurant supportive social media. However, the process of making food tourism available for more audience are often hampered by lack of budget. Therefore, it can be modeled as an optimization problem with limitations, particularly a 0/1 knapsack problem. Optimization problem is one of the most common problem in computer science with many ways to solve. In this paper, the author will discuss the implementation of branch and bound algorithm to optimize food tourism planning modeled as 0/1 knapsack problem with the help of Zomato API as the restaurant data provider.**

Keywords—Branch and bound, optimization problem, food tourism, zomato API

I. Introduction

Tourism has become an integral part of humanity for a long time. It's in human nature to seek recreation from its routine. According to Britannica [1], tourism we knew today is a product of modern society, beginning in western Europe in the 17th century while the concept of "recreation seeking" itself dates back to the classical era. It is no surprise that tourism has grown to be associated with other activities such as business, sport, or culinary.

According to a 2019 report by the World Food Travel Association [2], 80% of leisure travelers visit to a destination are motivated by culinary attraction. Furthermore, their experts state that the number one reason why food tourism is important to the tourist is because it provide an authentic and sustainable experience to visitors. The report believed that culinary activity is one of the core experience of tourism. Food tourism also generate positive economic impact and helped protect local heritage. On the interaction between visitors and locals, the report suggests that food and drink enable the two parties to connect. Based on the report, food tourism is undoubtedly one of the most important and fastest growing part of tourism industry.

The growth of food tourism and food related social media is inseparable. Some of the food tourism supporting social media are instagram, facebook, path. Among these platform, one rises because of its specialization in food attraction. Zomato is a platform focused on food attraction discovery. It features location based search, discovery, reviews, blogs, and many more food attraction related features. Other than its website and application, Zomato also provide other means of interacting to its services, by REST API. Their API provides most of the features available on its website and application such as search and culinary attraction info.

Provided these information, it is possible to model food tourism planning as an optimization problem. The optimization problem is on of classic computer science problem that it had many ways to be solved. A more detailed explanation on tourism planning optimization will be discussed on this paper.

II. Theoretical Framework

A. Optimization Problem

The study of computer science has been around since Claude Shannon published his paper titled "A Mathematical Theory of Communication" published in 1948 [3]. In it, Shannon described multiple theorem that would be known as the beginning of computer science. As it progresses, the scope of computer science span further than just theoretical. It started to model real-life problem to computable theoretical model. In these real-life modelling, experts realize that they keep coming back to the same problem. Referring to "Classic Computer Science Problems" by David Kopec [4], few of the classical computer science problem are constraint satisfaction problems, solving graph problems, k-means clustering, and adversarial clustering.

Optimization problem is one of many constraint satisfaction problems which focuses on finding the best solution from constrained conditions. It can further be classified as discrete or continuous optimization problem. While continuous optimization problem deals with continuous variables, discrete optimization problem deals with discrete concepts such as combination and permutations.

Let's consider an example of discrete optimization problem. Imagine a map of Australia that will be colored by state. A constraint might be no two adjacent regions should share a color. Then, the optimization problem is to use as few colors as possible. This is a trivial task for human as it is possible to be solved through trial and error and pattern recognition. Meanwhile, to be able to solved by computer the problem has to be modelled to a computable graph model. In this example, the

problem is a graph coloring problem which has a high correlation to permutation.

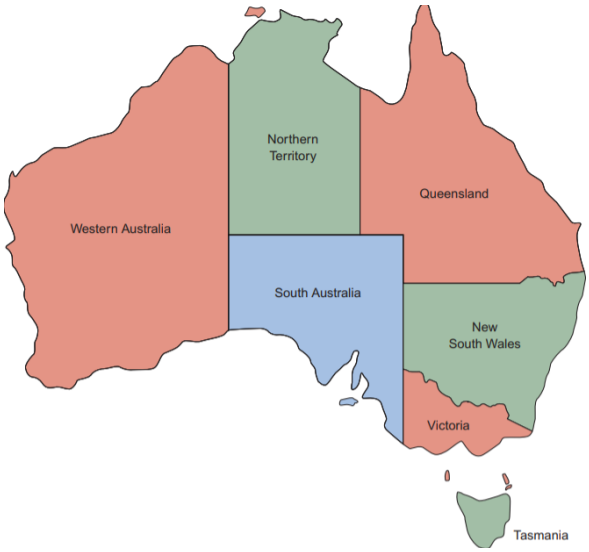

Figure 1

In the following subchapter, a specific discrete optimization problem, the 0/1 knapsack problem, will be explained in depth.

B. 0/1 Knapsack Problem

As mentioned in the previous subchapter, knapsack problem is a specific problem of discrete optimization problems: given a set of items, each with its own weight and value, determine the quantity of each item to be included so that the total weight is less than or equal to the given limit and the value is as large as possible. The problem has been studied for quite a long time with the first work of the problem done as early as 1897 [5].

While this paper would focus on 0/1 knapsack problem, there are three variant of the problem in the context of discrete values, each with their own definition and constraint. The bounded knapsack problem is a knapsack problem where there is more than only one of each item (x_i) but under a maximum c. Formally,

maximize $\sum_{i-1}^{n} v_i x_i$

subject to $\sum_{i=1}^{n} w_i x_i \leq W$ and $0 \leq x_i \leq c$

Then, the unbounded knapsack problem removes the constraint that an item has maximum number of c copies. Formally it is described as,

maximize $\sum_{i-1}^{n} v_i x_i$

subject to $\sum_{i=1}^{n} w_i x_i \leq W$ and $x_i \geq 0$

Lastly, the 0/1 knapsack problem limit the copy of each items to zero or one. It is formally written as,

maximize $\sum_{i=1}^{n} v_i x_i$

subject to $\sum_{i=1}^{n} w_i x_i \leq W$ and $x_i \in \{0,1\}$

Other than the three mentioned above, there are actually a lot more variation when the problem domain is extended to more than discrete and integer quantity.

Consider the following example for 0/1 knapsack problem with available items in the table below.

Table 1 : 0/1 Knapsack Problem Example Data

ID	Weight	Value
1	10	5
2	15	9
3	7	3
4	12	6

Figure 2

With maximum weight of 23, the maximum value possible based on the table is 12 by taking item with ID 2 and ID 3. The 0/1 knapsack problem is solvable by brute force, dynamic programming, or branch and bound algorithm. The problem also often approximated by using a polynomial time algorithm such as greedy algorithm.

C. Branch and Bound Algorithm

State space tree building and traversing is one of many algorithms commonly used in computer science. Two of the most widely known such algorithm are breadth first search (BFS) and depth first search (DFS). While the two algorithms are excellent in their own way, they are not optimized to solve optimization problem. Branch and bound algorithm is a variant of state space tree building and traversing algorithm to maximize or minimize objective function with constraints.

Branch and bound algorithm is one of the most excellent algorithm in solving optimization problem as it is more concise and efficient than brute force while at the same time produce equally good or better result than greedy algorithm. It is possible because of two major component of branch and bound algorithm:

- Cost function, denoted with $\hat{c}(i)$ is the approximated lowest cost getting to the target node via node i
- Bounding function, is a way of bounding nodes that if expanded wouldn't result in a goal node. Generally, there are three [6] criteria of a bounding function:
    - The value of the node is worse than current best node value
    - The node represents constraint(s) violation
    - The feasible solution at the node is only comprised of one point

In a nutshell, branch and bound algorithm is BFS implementation with modification to node expansion mechanism. While in BFS, the first queued node gets expanded first, in branch and bound the node to be expanded is determined by the nodes costs. The algorithm of branch and bound is as follows

1. Insert the root node into queue Q. If the root node is equals to the goal node, then the solution has been found. Stop the program
2. If Q is empty, then no solution exists. Stop the program
3. If Q is not empty, then choose node i from queue Q with the lowest cost, if more than one node has the lowest cost choose a random node
4. If node i chosen before is equals to the goal node, then the solution has been found. Stop the program
5. If node i doesn't have any child, then return to step 2
6. If node i is not equals to the goal node, then calculate the cost of its every child and insert them into queue Q.
7. Return to step 2

Branch and bound is a good algorithm to be used to solve 0/1 knapsack problem as it falls under optimization problem. The algorithm works on 0/1 knapsack problem by using maximum possible value using fraction knapsack problem as the bounding function and comparing it with current maximum value. It is one of the better algorithm to solve the problem as its complexity is $O(n^2)$. The algorithm is as follows:

1. Sort the items to be optimized in a descending order based on value per weight
2. Initialize maxProfit with 0
3. Initialize a new list L with dummy first node with zero weight and value and level set to -1
4. Get the first element from L as e
5. Compute the profit of e, if it is greater than current maxProfit then update the maxProfit with the profit of e
6. Compute the bound of e, if it is greater than current maxProfit then it is possible to generate a higher profit so take e into the knapsack, then add e to the list L.
7. Add a new element v to the list L describing e not taken.

D. *Zomato API*

As mentioned in the previous chapter, Zomato API is an interface to interact with Zomato's services to obtain culinary attraction related information. Referring to its documentation, Zomato API are divided into three main categories: common, location, and restaurant. Interfaces in the common category consists of six endpoints. Most of them are used to retrieve restaurant metadata, including its locations. Two endpoints in the location category are used to retrieve location based on Zomato's format. The remaining four endpoints in the restaurant category are used to search and retrieve information about a restaurant such as its name, average price for two, and user rating.

Using the Zomato API works by making a get request to the desired endpoint with access key as the header and required parameters as URL encoded form data. The endpoint would then return a JSON containing the requested data if no error occurred. The limitation of this API is that it only provide 20 restaurant data per request, so that the planning could not be done with a broader culinary attraction variance. Depicted below is the response of request made to /cities endpoint with "jakarta" as the query.


```
{
  "location_suggestions": [
  {
    "id": 74,
    "name": "Jakarta",
    "country_id": 94,
    "country_name": "Indonesia",
    "country_flag_url":
"https://b.zmtcdn.com/images/countries/flags/cou
ntry_94.png",
       "should_experiment_with": 0,
       "discovery_enabled": 0,
       "has_new_ad_format": 1,
       "is_state": 0,
       "state_id": 0,
       "state_name": "",
       "state_code": ""
    }
  ],
  "status": "success",
  "has_more": 0,
  "has_total": 0
}
```


III. Implementation

In this paper, the food tourism planning will be modeled as a 0/1 knapsack problem with restaurant rating multiplied by vote count as the value and average price for two as the weight. The program will accept trip budget as the constraint and optimize the available budget for the highest rating.

The program fetch data from Zomato API prior to the suggestion so that it always recommend based on the latest data. The fetching make use of python's request library with the get method. The fetched data will then be decoded into associative array, which will then be converted into list of Restaurant objects. The code snippet below is the function to fetch the data

```
def fetch():
    # City Codes
    JAKARTA = 74
    BANDUNG = 11052
    BALI = 170
    url =
"https://developers.zomato.com/api/v2.1/search"
```

```
    data = {
        'entity_id' : ...,
        'entity_type' : 'city'
    }
    res = requests.get(url, params=data,
headers={"Accept":"application/json", "user-
key":SECRET})
    res = res.json()
    resto_list = list()
    for resto in res['restaurants']:

resto_list.append(Restaurant(resto['restaurant']
['name'],resto['restaurant']['average_cost_for_t
wo'],resto['restaurant']['user_rating']['aggrega
te_rating'],resto['restaurant']['user_rating']['
votes']))
    return resto_list
```

As previously stated, the food tourism planning will be modeled as a 0/1 knapsack problem with restaurant rating multiplied by vote count as the value and average price for two as the weight. Refering to the branch and bound algorithm on chapter three, below is the algorithm for calculating bounding function.

```
def bound(node, max_weight, resto_list):
    n = len(resto_list)
    if (node['weight'] >= max_weight):
        return 0
    bound_value = node['profit']
    j = node['level'] + 1
    sum_weight = node['weight']

    while ((j < n) and (sum_weight +
resto_list[j].getWeight() <= max_weight)):
        sum_weight    +=
resto_list[j].getWeight()
        bound_value += resto_list[j].getValue()
        j += 1

    if (j < n):
        bound_value += (max_weight - sum_weight)
* (resto_list[j].getValue() /
resto_list[j].getWeight())
```

The bounding function above is then applied to the 0/1 knapsack algorithm as described in chapter three. Below is the code snippet for 0/1 knapsack problem solving algorithm

```
def knapsack(W, resto_list):
    n = len(resto_list)
    Q = [{'level' : -1, 'profit' : 0, 'weight' :
0, 'bound' : 0, 'name' : 'init'}]
    v = {}
    w = {}
    maxProfit = 0
    while len(Q) > 0:
        u = Q.pop()

        if (u['level'] == n-1):
            continue

        v['level'] = u['level'] + 1
        w['level'] = u['level'] + 1
        v['weight'] = u['weight'] +
resto_list[v['level']].getWeight()
        v['profit'] = u['profit'] +
resto_list[v['level']].getValue()
        v['name'] =
resto_list[v['level']].getName()
        w['name'] =
resto_list[w['level']].getName()

        if (v['weight'] <= W and v['profit'] >
maxProfit):
            maxProfit = v['profit']

        v['bound'] = bound(v, W, resto_list)
        if (v['bound'] > maxProfit):
            v['prev'] = u['name']
            print("Taken")
            print(v)
            Q.append(v)

        w['weight'] = u['weight']
        w['profit'] = u['profit']
        w['bound'] = bound(w, W, resto_list)
        if (w['bound'] > maxProfit) :
            w['prev'] = u['name']
            print("Not taken")
            print(w)
            Q.append(w)
```

```
    return maxProfit
```

IV. Case Studies

In this chapter, the food tourism planning algorithm will be executed on a few case study. The test will be done for culinary attraction in Jakarta and Bandung, testing for budget of 500.000 IDR and 1.000.000 IDR. The table below is the result of executing the algorithm for Jakarta

Table 2 : Result for Jakarta

Budget	
500.000 IDR	1.000.000 IDR
1.Sate Taichan "Goreng" 2.WAKI Japanese BBQ Dining 3.Sushi Go! **Price Total** : 470.000	1.Sate Taichan "Goreng" 2.WAKI Japanese BBQ Dining 3.Sushi Hiro **Price Total** : 770.000

Figure 3

Table 3 : Top Culinary Attraction in Jakarta

Name	**Avg Rating**	**Vote**	**Avg Price for Two**
Sate Taichan "Goreng"	4.9	3491	70000
WAKI Japanese BBQ Dining	4.9	1330	300000
Kintaro Sushi	4.8	1008	200000
Gurih 7	4.1	523	150000
Katsu YA	4.4	470	300000
Sushi Hiro	4.9	1607	400000
The Cutt Grill House	4.7	734	300000
Sushi Hiro	4.9	4515	400000
Magal Mapogalmegi BBQ	4.9	1345	300000
Gyu - Kaku Japanese BBQ	4.9	1096	350000
The Garden	4.6	1072	250000
Sushi Go!	4.9	1418	100000
Sushi Go!	4.7	338	100000
Sushi Go!	4.7	1959	100000
Sushi Hiro	4.8	2030	400000

Figure 4

With the same algorithm, the result of executing the program for Bandung has the following result

Table 4 : Result for Bandung

Budget	
500.000 IDR	1.000.000 IDR
1.Sate Taichan Goreng 2.Se'i Sapi Lamalera 3.Miss Bee Providore 4.One Eighty Coffee **Price Total** : 360.000	1.Gormeteria 2.Ambrogio Patisserie 3.One Eighty Coffee 4.Shabu Hachi 5.Sejiwa Coffee **Price Total** : 930.000

Figure 5

Table 5 : Top Culinary Attraction in Bandung

Name	**Avg Rating**	**Vote**	**Avg Price for Two**
Gyu - Kaku Japanese BBQ	4.9	3491	70000
Gyu - Kaku Japanese BBQ	4.9	1330	300000
Sate Taichan Goreng	4.8	1008	200000
Se'i Sapi Lamalera	4.1	523	150000
Gormeteria	4.4	470	300000
Miss Bee Providore	4.9	1607	400000
Ambrogio Patisserie	4.7	734	300000
One Eighty Coffee	4.9	4515	400000
Shabu Hachi	4.9	1345	300000
Hachi Grill	4.9	1096	350000
Sushi Hiro	4.8	2030	400000

Figure 6

Based on the two case studies above, it can be concluded that the algorithm works as expected. It is unfortunate that Zomato API only provide 20 restaurant data per request resulting in a not so optimized use of budget. For example, in table 2 for 1.000.000

IDR the total price is only 770.000 IDR, leaving a lot of headroom to be utilized. This also highlight how culinary attraction in Jakarta is far more expensive as there is no culinary attraction that could fill the 230.000 IDR gap. Meanwhile, culinary attractions in Bandung are far more affordable concluded from the 1.000.000 IDR case that could fit 5 culinary attraction and managed to have total price as high as 930.000 IDR.

V. Conclusion

Jakarta and Bandung each has its own culinary attractions characteristic. Jakarta has a more expensive overall culinary attraction average price, while Bandung is more affordable. From the technical prespective, branch and bound algorithm proved to be optimal on solving 0/1 knapsack problem. The integration of Zomato API with 0/1 knapsack solving enable the user to plan food tourism ahead of time with their desired budget.

VI. Acknowledgment

The author would like to thank God the Almighty for grace and blessings. The author would also thank Dr. Ir. Rinaldi, M.T. as the lecturer of algorithm design (IF2211). The author also express gratitude to family and friends for their support in the making of this paper

References

[1] https://www.britannica.com/topic/tourism

[2] 2019 State of the Food Travel Industry. http://bite.worldfoodtravel.org/lt.php?s=c3a2318e0fcdbadb9ebd376d63e89f5e&i=5A9A1A9

[3] Lohit and Jagadish, A NEW LOSSLESS METHOD OF IMAGE COMPRESSION AND DECOMPRESSION USING HUFFMAN CODING TECHNIQUES. Hubli, India. 2005.

[4] https://classicproblems.com/

[5] Mathews, G. B. (25 June 1897). "On the partition of numbers" (PDF). Proceedings of the London Mathematical Society. 28: 486–490. doi:10.1112/plms/s1-28.1.486.

[6] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Branch-&-Bound-(2018).pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

I Putu Gede Wirasuta - 13517015