A TRIP TO EUROPE: A DYNAMIC APPROACH TO PLANNING THE BEST VACATION

21-393 OPERATIONS RESEARCH II

Katherine A. Yang
Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213
katheriy@andrew.cmu.edu

Olivia G. Husak
Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213
ogh@andrew.cmu.edu

Sophie C. Stoyer
Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213
sstoyer@andrew.cmu.edu

Thomas J. Pascucci
Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213
tjpascuc@andrew.cmu.edu

December 17, 2019

ABSTRACT

Planning a trip can be a daunting task if not done properly. Through research and thorough consideration however, outlining a fun and exciting trip can be easy. The objective of this project was to create a program that, when given user input regarding various European cities, would output a trip that would give the most utility to the individual traveling. A dynamic programming model was used to determine the trip with maximum utility for the individual given their preferences, budget, and time. For each of the team members' preferences, the designed program was able to output a feasible trip that maximized the designed utility from the given preferences.

Introduction

You hear the question all the time: if you could travel anywhere in the world, where would it be?

Travel is fun, but it can be difficult to figure out where to go or how long to stay there. And since people have different preferences, a trip that one person likes might not be good at all for someone else. As the number of choices increases, it only gets harder to plan the optimal vacation: people have a limited rational capacity, which means that balancing and optimizing all these options can be quite daunting. That's where operations research comes in: supply a person's preferences along with the necessary logistical constraints, and let a program figure out what the best travel plan is.

In our problem, we consider trips to Europe taken individually by the four of us. Our possible destinations are Amsterdam, Athens, Barcelona, Berlin, Budapest, Dublin, London, Paris, Rome, and Venice. We have T days and a budget of B for our trip. Each of us provided utility functions to model our preferences.

1 Dynamic Program Solution

In our formulation of the problem, we must make a choice every day about whether or not to stay where we are and where to go if we decide to leave. Staying in a city incurs a cost specific to that city that must be subtracted from the budget. Traveling between cities also incurs a cost based on the origin and destination cities, and this cost too is subtracted from the budget. And of course, every day that we are in any city must be added to the total number of days spent on vacation, which cannot exceed the T days allotted. Additionally, each person derives a certain utility from being in a certain city; however, this value is not constant even for a fixed person, as we derive different levels of

satisfaction from being in a city depending on how long we've been there. In general, the additional satisfaction coming from being in a city one more day decreases the longer we've been in that city. The utility functions model this, as we see below, and we sum the utility values for all the days of the trip to get a total utility value. That total value is what our dynamic program attempts to maximize.

1.1 Utility Functions

A utility function is a representation of individual preferences for certain activities beyond their monetary value. In our traveling problem, our utility function weights the expense, travel time, and enjoyment between each city. The function outputs 'utils', a measure of satisfaction with the decision, for our traveller.

Utility functions can take many different shapes and vary based on the preferences of each individual. For the sake of illustration, consider an example in which a banker is receiving money. In their simplest form, utility functions are linear, (i.e. the more money he receives, the corresponding amount greater his utility is). However, behavioral theories have uncovered that we do not exhibit such 'rational' proclivity for an increase in units. We make the three following observations in light of our example:

1. The theory of Diminishing Marginal Utility explains that for each additional increase in money, we observe a smaller corresponding increase in our utility. To illustrate, the improvement in happiness our banker experiences when he moves from owning \$10 to \$1,010 would be quite large. In contrast, the improvement in happiness he would experience in moving from \$100,000,000 to \$100,001,000 would be less significant. Based on this experience, the utility function is concave; as we increase the inputs more and more, the corresponding satisfaction improves by smaller margins.
2. We also observe a phenomenon known as loss aversion, or risk aversion in probabilistic utility, where an observed loss "hurts" more than a gain of similar magnitude. Our banker would be quite happy if he received \$1,000! But he would experience an even greater loss of satisfaction if he lost \$1,000.
3. Finally, there is a threshold where utility becomes discontinuous and an individual loses their satisfaction. For example, if all of the resources are used up, we experience 0, or $-\infty$, utility. For example, say our banker receives \$100 per hour that he works in a day. If a dynamic programming recommends that he work 25 hours or more per day, he would lose all of his utility because he has exceed his use of resources.

In light of these observations, we are able to make the following connections to the travelling problem at hand. First, our traveller will exhibit diminishing marginal utility per day spent in a location. The first day in Rome might be spectacular, but after spending three days there, the traveller may experience a greater benefit from moving on. In the same light, the traveller experiences loss aversion, i.e. the amount of time spent travelling would affect their utility by a greater magnitude negatively than spending that same time touring a city would affect their utility positively. Furthermore, if a traveller uses up their resources, that is if they run out of money or time, they will experience no more satisfaction in travelling. Such an occurrence is factored into the dynamic program via constraints on the function.

Based on the parameters and the above behavioral observations, we craft the following utility functions that have been researched to be representative of human preference. For an individual of moderate risk-aversion, (i.e. the middle ground of concavity), a negative quadratic utility function will accurately reflect such preferences. Let $a > 10$ be the relative intercept that reflects an individual's utility preferences for a given city, and $i > 0$ represent the number of days already spent at the current location p. Then the utility gained on that day is:

$$u = a - (i_p - 1)^2$$

Every traveler will have their own utility functions for each city based on the time spent in that city, which will contribute to the overall utility function. For simplicity, a linearly decreasing function with respect to time can be used to determine the added utility of another day spent in a city. The total utility of the trip would be given by the summation of the daily utilities from the trip length m, i.e:

$$u_{total} = \sum_{k=0}^{m} u_k$$

1.2 Gathering Data

We gathered the following data regarding distances between any two cities in order to calculate the cost of different trips. The numbers represent the absolute distance between the row and column cities in miles. This was a simpler metric than actual flight distances as those would verify from airline to airline.

	Ams	Ath	Bar	Ber	Bud	Dub	Lon	Par	Rom	Ven
Amsterdam	0	3082	1639	649	714	471	358	267	806	586
Athens	3082	0	1169	1121	698	1777	1489	1305	655	781
Barcelona	1639	1169	0	930	933	915	708	517	532	882
Berlin	649	1121	930	0	428	817	578	545	734	491
Budapest	714	698	933	428	0	1177	901	775	504	351
Dublin	471	1777	915	817	1177	0	287	483	1170	997
London	358	1489	708	578	901	287	0	212	889	708
Paris	267	1305	517	545	775	483	212	0	687	526
Rome	806	655	532	734	504	1170	889	687	0	245
Venice	586	781	882	491	351	997	708	526	245	0

1.3 Problem Formulation

We started by coming up with a dynamic programming solution to a simple version of the problem.
Let:

- $p =$ current destination
- $i =$ number of days already spent at p
- $b =$ remaining budget
- $t =$ remaining time
- $D =$ set of destinations already visited
- $x = \delta(p, d) =$ cost to travel from p to d
- $y = c(d) =$ cost of staying at d
- $u_d(i) =$ utility of staying at d, having already been there i days

We then define the overall utility function f as follows:

$$f(p, i, b, t, D) = \max \begin{cases} \max\limits_{d \notin D} [u_d(0) + f(d, 1, b - x - y, t - 1)] \\ u_p(i) + f(p, i + 1, b - y, t - 1) \end{cases}$$

We let $f(p, i, b, 0, D) = f(p, i, 0, t, D) = 0$ as base cases. Note that the budget does not consider the cost of traveling to the starting destination or traveling home from the final destination.

2 Code Implementation

2.1 Base Solution Code

After designing the dynamic programming objective function, the next step was to implement the program in code. Python was chosen as the coding language due to its imperative nature, but also due to its ease of understanding for those just looking at the code. The code needed to perform three main steps: testing each of the destinations ($d \notin D$) for the maximization step, gathering all the costs associated with either staying in a location or traveling, and then calculating the objective function at that stage and the optimal future stages. The code we developed is shown below.

```
def travel(p, days, b, t, dList, itin):
    x, y = 0, 0
    xitin, yitin = itin, itin
    if b > 0 and t > 0:
        maxX = 0
        maxItin = []
        for i in range(len(dests)):
            print(i, b, t)
            if dests[i] not in dList and i != p:
                ux = u[i](0)
                newX, tryItin = travel(i, 1, b - 3 * planes[p][i] - costs[i], t - 1, dList + [dests[i]], itin + [(dests[i], time - t)])
                tryX = ux+newX
                if tryX > maxX:
                    maxX = tryX
                    maxItin = tryItin
        uy = u[p](days)
        newY, yitin = travel(p, days + 1, b - costs[p], t - 1, dList, itin + [(dests[p], time - t)])
        y = uy+newY
        x = maxX
        xitin = maxItin
    if max(x,y) == x: return (x, xitin)
    else: return y, yitin


def europe():
    final = travel(0, 1, budget, time, ["Amsterdam"], [])
    print(final)
    return final
```

2.2 Program Interface

To make the user experience as seamless as possible, we integrated a user interface into the program. The interface of the application allows the user to easily enter information, especially the constraints, of their upcoming trip. This includes:

1. The budget they have for the trip (in USD)
2. The length of their trip (in days)
3. The starting location of their trip

After this, the user can specify their preferences of cities to visit. They can distinguish preferences between all of them or just the top few if they become indifferent after those. The user can always go back into the program and change these preferences if the outputted trip isn't to their liking, as perhaps they misrepresented their true preferences originally.

Using the information given by the user, the program outputs the optimal trip. This includes two things, an itinerary and a map, to best illustrate the trip to the user. The itinerary, found in the top right corner, displays the city the traveller will be in on any given day of their trip. This overlays the map, which visually displays the trip for the user. Red nodes represent each of the cities available as options, and black edges represent the travelling undertaken by the traveller. Examples of this output are provided in the next section.

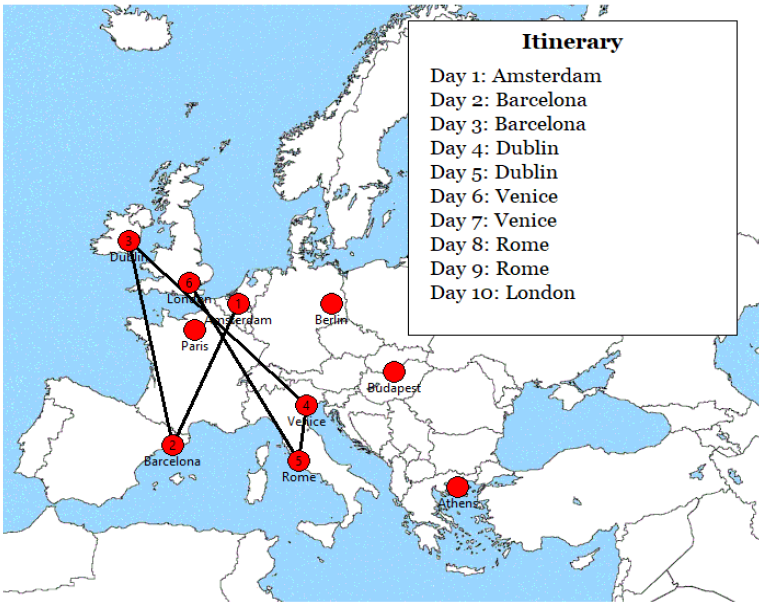
3 Finding Solutions

Once we had functioning code, we could experiment with different utility functions to generate different tours based on different preferences.

3.1 Identical, Simple Utility Functions

A very simple version of the problem is a traveler having the same utility function for each city. In this situation, the choices of where to go are based entirely on the costs of traveling, which comes down to distances. This version of the problem is then essentially a typical Traveling Salesman problem.

We ran this through our algorithm with utility functions for each destination of $10 - (x - 1)^2$ where x is the number of days spent at that destination. For a tour of 10 days starting in Amsterdam with a budget of $5,000, we generated the following tour:

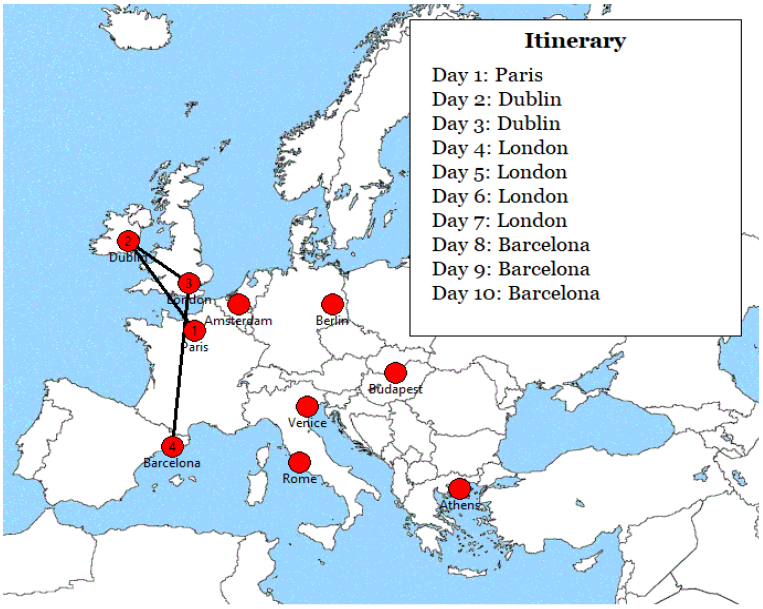


3.2 Individual Tours

A more complex version of the problem involves changing the utility functions in order to match each of the authors' preferences. For consistency, we used a budget of $5,000 and a 10-day trip starting in Paris for each person, so the only differences were in the utility functions. As a additional measure to keep things consistent, we had each traveler have a first, second, and third most preferred city and all other cities tied as fourth most preferred.

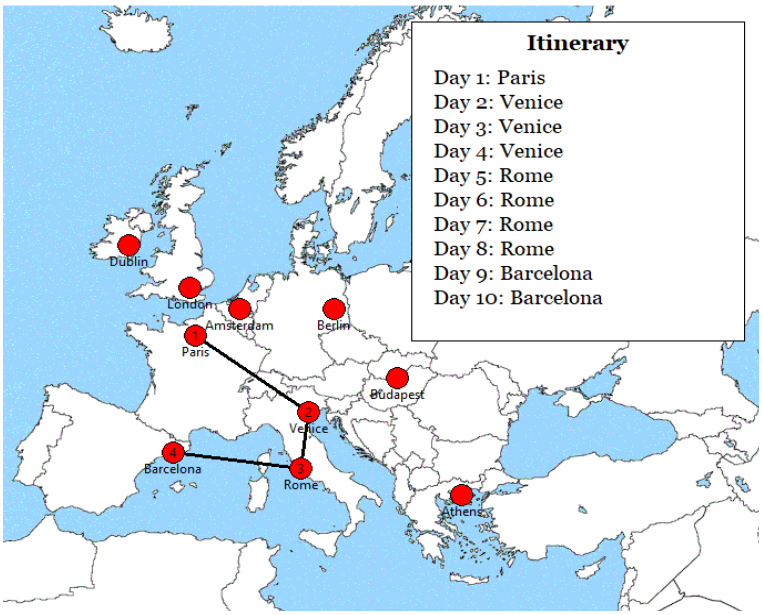
3.2.1 Katherine

Katherine gets the most utility (in order) from visiting Dublin, London, and Barcelona. To account for this, we assigned these cities utility functions of $18 - (x-1)^2$, $17 - (x-1)^2$, and $15 - (x-1)^2$ respectively. This generated the following tour:



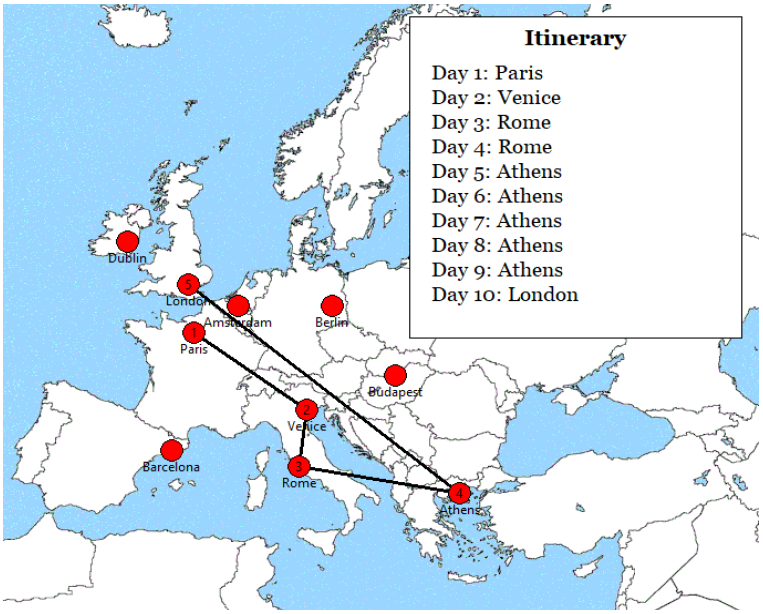
3.2.2 Olivia

Olivia gets the most utility (in order) from visiting Rome, Venice, and Barcelona. To account for this, we assigned these cities utility functions of $20-(x-1)^2$, $18-(x-1)^2$, and $13-(x-1)^2$ respectively. This generated the following tour:



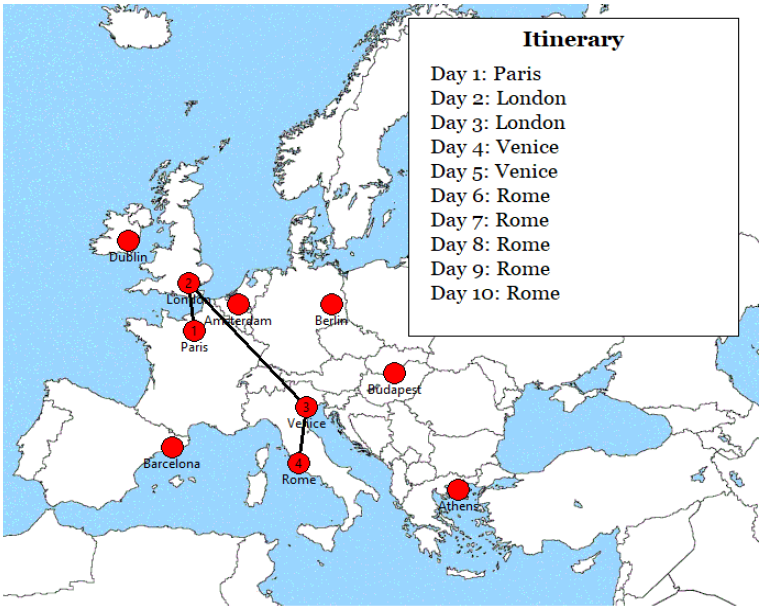
3.2.3 Sophie

Sophie gets the most utility (in order) from visiting Athens, London, and Paris. To account for this, we assigned these cities utility functions of $20-(x-1)^2$, $15-(x-1)^2$, and $12-(x-1)^2$ respectively. This generated the following tour:



3.2.4 TJ

TJ gets the most utility (in order) from visiting Rome, Paris, and London. To account for this, we assigned these cities utility functions of $20-(x-1)^2$, $18-(x-1)^2$, and $13-(x-1)^2$ respectively. This generated the following tour:



4 Discussion

Based off of the recorded distances, the dynamic model used, the preferences of each author, and the code designed to solve this problem, we were able to develop several tours to best match each author's individual preferences. On a whole, the program seemed successful. The program largely prioritized the cities that were ranked higher according to an individual's preferences while still ensuring that several cities were seen. For each of the generated tours, the top two prioritized cities for that person were visited for at least six days combined. This showed that the program was successful.

These tours do, however, reveal some apparent flaws in our logic in programming it. First, only three preferences were recorded and had their utility functions changed from those of the simplest example (3.1). This update to the utility function was significant enough to cause a larger amount of time to be spent specifically in the most preferred city. If utility functions were more evenly distributed along with preparing a trip of more days, more cities would have been included in the tours. In addition, there is a concern about the travel costs used in this model. While the distances between cities are accurate, the actual cost of travel may not be exactly proportional to the travel distance. The constant factor applied to the travel distance could also be on the higher end, which would cause the program to favor solutions that visit fewer cities. With this in mind, there are plenty of alterations that could be made to improve our model. We discuss some of these next.

5 Further Exploration

The problem we formulated was relatively simple. It makes several assumptions to simplify the problem, and it leaves out several things that would make finding solutions more complicated but would also make those solutions more optimal and the overall situation more realistic. In this section we discuss some of these areas in which our solution could be improved.

5.1 More Cities

The problem we solve here uses only ten potential cities. The methods we used and the code we wrote do not rely on there being ten cities, and in fact both the methods and the code work for an arbitrary number of cities. Thus in theory, our solution can be applied to much larger lists of cities. However, the dynamic programming approach relies on going through and testing each possible solution to find the best one; while this is not terribly time consuming or computationally heavy with our simple model and ten cities, it would quickly become infeasible as more and more cities were added. If we wanted to solve a problem with a significantly larger number of cities and do so in a reasonable amount of time, we would have to consider other methods. Since it wouldn't be possible to try every solution, we might consider heuristics such as genetic algorithms, simulated annealing, or tabu search.

5.2 Activities within Cities

When we originally thought about this project, we considered making choices not only about which city to be in when and for how long but also about what to do in specific cities while we were there. For example, if we decide to stay in Paris for a day, do we see the Eiffel Tower? The Louvre? Notre Dame? In making these decisions, we would have to consider the satisfaction a person derives from each activity along with any associated costs (travel within the city, admission fees, etc). It was eventually determined that we should focus on making the simpler model work first, as these additional decisions would make the problem correspondingly more difficult. However, now that we have a base solution method, it would be interesting to see how these additional choices could be incorporated.

5.3 Different Methods of Traveling

In our model, we considered one way to travel directly between two cities and did our calculations based on that. In reality, there are multiple ways to travel between most destinations: plane, car, bus, train, ferry. A more complete model would include some of these other travel options where they are feasible and allow the traveler to choose how they want to get to each city. Different methods of travel of course cost different amounts, and each traveler would derive different satisfaction from using different transportation methods. For example, someone who values speed of travel greatly might prefer to fly from Berlin to Amsterdam, but someone else might prefer to drive between the cities and see the scenery. A model allowing for different travel methods would have to use utility functions that take these additional costs and preferences into account. This would make the model and its solution more complicated, but also more accurate to the real world situation they represent.

5.4 Initial and Final Travel Costs

Our program formulates a solution starting with the traveler being in a European city and leaving them in the final European city they visit. As such, it does not make any considerations for how the traveler gets to the starting city (which is one of the vacation cities and not their home city) or for how the traveler leaves the final city. This simplifies the problem, but isn't a realistic representation of the situation, as the traveler must have a way to get to and from their home. In a more complete solution, we would take those travels into account when calculating utility and optimal routes, as there would of course be different costs for the trips to and from home depending on what the starting and ending cities were.

References

[1] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[2] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.