Student 2938740 – Optimisation Assignment

Part 1 – Maximising a Function

See codebook – Part 1.ipynb

This solution was a combination of random search, hill climb coupled with an unrealised venture into basic evolutionary algorithm techniques. The code was run manually to gain an intuition as to what was happening and gain finer control of the process of improvement.

The process developed as:

1. Run several 10 million step brute force style random search
2. Identify results with potential and make them compete against each other
3. Run iterative random search / manual simulated annealing by feeding back x, y, z coordinates into the random search
4. Log results

The brute force random search identified two peaks of interest, similar X and Z coordinates but with one Y in the positive and the other in the negative.

fbest	X	Y	Z
11.1211676	1.62772776	1.925139571	1.41367972
11.10044207	1.63010322	-1.84464909	1.40923697

More Random Searches were then performed against the two peaks of interest and iteratively smaller increments being applied to the X, Y, Z coordinates in the fashion of manual simulated annealing. The second of the peaks of interest (negative Y) improved to level worse than the original positive Y so it was decided to only proceed with the positive Y version.

The best result for the positive Y peak is as follows:

fbest	X	Y	Z
11.16922792	1.63047038	1.92681881	1.41364397

Results can be seen in the codebook.

Part 2.1 –Single Objective Optimisation

See codebook – Part 2.ipynb

As per part 1, I kept the finer points of this manual to test the iterative approach against the optimisation hill climb algorithm.

A series of hill climbs were performed to try and get as close to the £50000 limit as possible. As the iterations grew closer and stalled in the 49000s, I started to feed the solution vector back into the algorithm so that the starting point was near the top of the hill instead of at the bottom of the hill. It took less than 5 attempts before the algorithm maxed out at the full £50000 investment and an all-time high return of £61400.

The results of the final hill climb are given here and in the code book:

```
Optimal solution - Investment: 50000, Return: 61400
--------------------------------------------------
Opportunities selected:
[1, 3, 5, 7, 8, 9, 10, 13, 16, 17, 19, 20, 22, 23, 24, 25, 26, 30, 31,
32, 33, 38, 40, 42, 43, 44, 47, 49, 50, 51, 52, 55, 57, 61, 62, 63, 64,
65, 66, 67, 68, 71, 74, 75, 76, 77, 79, 80, 81, 82, 83, 84, 86, 89, 90,
91, 93, 94, 96, 99, 100]
--------------------------------------------------
Solution vector:
[1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1
, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
1, 0, 0, 1, 1]
--------------------------------------------------
```

The solution vector relates to the list of investments (1 represents an investment) and was multiplied by the Investment and Return vector to verify that the solution was valid. The vector was found to be valid.

Having proved the theory of iterative feedback manually I feel that I have a much better intuition for how these algorithms would be working if automated, and more importantly I have been able to learn where the silly coding errors are likely to be found when unexpected results start to appear.

Part 2.2 –Multi Objective Optimisation

See codebook – Part 2.ipynb

Solving the multi-objective problem starts with some basic analysis and an assumption that our answer to the single objective problem in part 2.1 was correct. This assumption is made on the basis that our client has asked for both solutions and if we do not use our solution from 2.1 then how can we expect the client to have faith in us?

Firstly, the data is analysed to see if any constraints can be found to reduce the size of the search space. The investments were ranked by lowest cost, highest return, highest profit, and highest profit ratio to see how the profit related to the number of investments. The limit of 50k was adhered to in each case. The analysis and optimisation use a profit metric which is calculated as "Cost – Return".

Approach	Num Items	Profit
Select lowest cost investments	70	10400
Select highest return investments	21	6700
Select highest profit investments	32	9700
Select highest profit ratio investments	61	11500
Select solution to part 2.1 Maximise investments	61	11400

The results are very interesting, we see that the optimal investment in part 2.1 was only £100 lower profit than selecting the highest profit ratio investments (however the total return is lower, this is calculated for profit). The lower bound of 21 items is disregarded due to the low profit margin in favour of 32 items with a profit of £9700 which is relatively close to the maximum profit margins. We now have 2 constraints for our search space:

- Lower bound: Selecting a minimum of 32 items yields a maximum profit of £9700
- Upper bound: Selecting a maximum of 61 items yields a maximum profit of £11500

To create a range of options for our client the starting point was to take the solution vector for the 32 investment lower bound and iteratively increase the number of items from 32 to 61 to see if the profit could be increased towards the upper bound limit of £11500.

The algorithm works by dropping a random investment from the minimum lower bound solution. The algorithm then adds as many as required to reach the next investment level. As the original vector contained all the highest profit items it automatically creates a lower profit which gives algorithm space to swap combinations of investments to find a higher profit. I tried a tighter approach that never went lower than the original £9700 lower bound profit but it kept stalling in the mid-50s so this more flexible approach was required.

The run-time is no more than a few seconds.

Presented overleaf is the histogram of results for one the better results showing the client what returns they could expect if they invest anything from 32 to 61 items. Note that the results are sub-optimal if compared to part 2.1 however we could always try to gain a better solution by running the algorithm for longer to generate more potential solutions.

For this given instance, the following results can be plotted and shown to the client so they can make an informed choice. On the limited run-time tested, a good optimisation run can hit £10000 profit around the early 50s and usually maxes out around £10700, or 93% of the observed global optima at 61 items. Running for longer may improve results as might a backwards sweep, starting from 61 items and gradually descending. As we know the global maxima already we can allow the optimiser to throw up suggestions for 61 items:

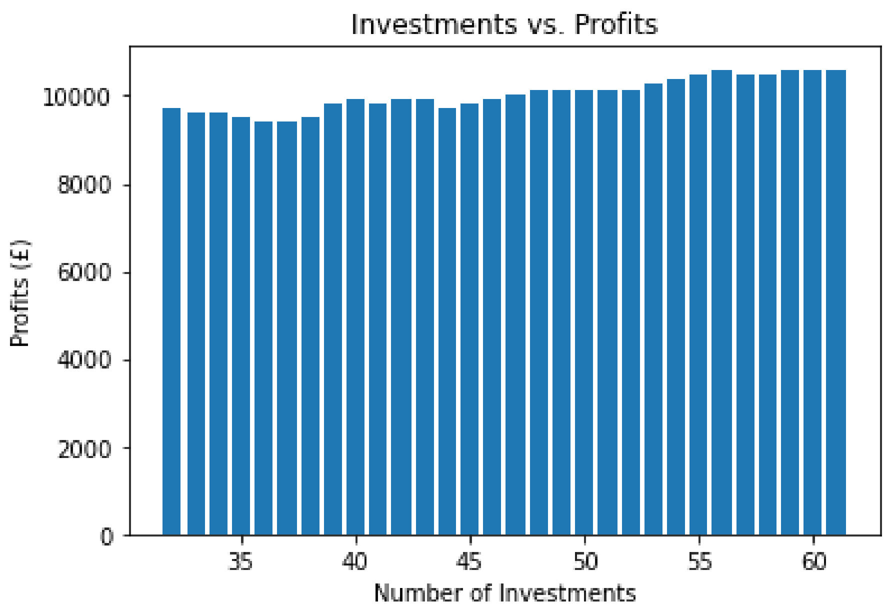


Figure 1: Maximise Profits vs. Minimise Investments

The code throws out a summary for each number of investments possible:

(Remember that the initial solution is always the 32 items minimised lower bound)

```
Initial solution - Investment: 49791, Return: 59491, Profit: 9700
-----------------------------------------------------------------
Best solution for 33 investments - Inv: 48420, Ret: 58020, Prof: 9600
Best solution for 34 investments - Inv: 49259, Ret: 58859, Prof: 9600
…
Best solution for 61 investments - Inv: 49614, Ret: 60214, Prof: 10600
```

The final vector stores an array from 0-29 corresponding to items 32-61, shown here is the portfolio of investments making up the 61 item solution:

```
# Print the investments in 61 items (final[29])
print([x for i,x in enumerate(opportunities) if final[29][i]])

[3, 4, 5, 7, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26,
27, 30, 32, 33, 36, 38, 39, 40, 41, 46, 48, 50, 52, 55, 57, 59, 61, 63, 64,
65, 66, 67, 68, 69, 71, 76, 77, 78, 79, 80, 81, 82, 83, 84, 86, 87, 89, 91,
93, 94, 100]
```

Part 3 – Travelling Salesperson Challenge

See codebook – Part 3.ipynb

Choosing the hill climb approach I adapted the algorithm from the practical sessions and ran a few test scenarios on one lorry to get a feel for the data. The results showed that perhaps it would be better to bisect the delivery areas long the vertical and send a lorry from each warehouse. This led to improved results. During the next iteration I identified some clusters close to the warehouses. I deployed 3 vans to pick off these closer stores due to the low mileage cost and sent a single lorry out to the further away stores that lies to the north of the warehouses. I ran two tests concurrently:

- 1 Lorry and 1 van departing warehouse #24, and 2 vans departing warehouse #25
- 1 van departing warehouse #24, and 1 lorry and 2 vans departing warehouse #25

Finally, I identified 5 clusters and despatched vans to these clusters with the algorithm determining the optimal route for each cluster. I could have used branch and bound given the limited number of destinations but to do this 5 times would have been time consuming and the algorithms were already set up. Results are as follows, full details in the codebook:

Solution	Start End Warehouse	Mileage	Cost	Fleet
2 Lorries	Same	591	£1773	One lorry departs from and returns to each warehouse with map segmented vertically along the centre line
1 Lorry + 3 Vans	Lorry W24	594	£1308	1 van leaves W24 and 2 vans leave W25 and return Lorry leaves W24 and returns
1 Lorry + 3 Vans	Lorry W25	544	£1158	1 van leaves W24 and 2 vans leave W25 and return Lorry leaves W25 and returns
5 Vans	Same	632	£632	2 Vans leave W24 and 4 vans leave W25. All return to starting point. Stores broken into 5 identifiable clusters.
2 Lorries	Different	522	£1566	One lorry departs from each warehouse with map segmented horizontally along the centre line Lorries return to the other warehouse

Lorries often record the lowest mileage however vans are just far more cost effective with 5 vans costing less than half of a single lorry. As well as the cost savings by deploying higher numbers of vans customers would likely receive their deliveries earlier in the day due to the limited numbers of deliveries each van can make.

Suggested optimal routes for each of the 5 vans

Warehouse 24 – 9 – 18 – 6 – 12 – 17 – Warehouse 25 (112 miles)

Warehouse 24 – 7 – 1 – 4 – 3 – 14 – Warehouse 25 (186 miles)

Warehouse 25 – 19 – 15 – 5 – 20 – Warehouse 25 (65 miles)

Warehouse 25 – 16 – 21 – 2 – 13 – Warehouse 25 (100 miles)

Warehouse 25 – 10 – 23 – 11 – 22 – 8 – Warehouse 25 (169 miles)

Part 4 – Optimising Stay Durations for a Holiday Park

Background

The problem I have chosen for this question comes from my professional life. My last client runs a chain of nationwide holiday parks and I was hired to be their "eyes" on a new software suite being delivered by an external consultancy. As well as understanding the architecture, technical aspects, and an expectation to perform code quality appraisals I was expected to be able to figure out and train colleagues on how the software operated, in lieu of me accepting a full-time role with the company.

The software suite included revenue and fleet optimisation models that would allow the company to start automating the optimal price at which to sell holidays based on previous sales, forecasted sales and availability within the confines of the optimal booking curve, which describes the art of increasing the price as customers scramble for last minute availability amid a booking frenzy.

The fleet optimisation component consisted of analysing the mix of caravans and lodges that the client had on their sites and assigning the optimal mixture of 3, 4 or 7 nights stays with the twin goals of maximising revenue and minimising wastage with wastage defined as bookings that led to empty days between the standard Friday, Saturday, or Monday arrival slots.

The delivered software turned out to be extremely inefficient and poorly coded, yet we were all surprised that it appeared to work for the revenue optimisation module however the fleet optimisation module was so poorly built that it was put to one side and deemed a waste of time to decode and would need rebuilding from scratch. To put it in perspective, the optimisation task took 30 hours to run and operated as a grid search with zero search optimisation. Technically, it consisted of multiple nested loops, massive functional overhead from using too many high-level R functions and committed every sin listed on the "common mistakes made by R programmers" list. The code was simply unreadable, and the project was deemed as a very expensive proof-of-concept that would be rebuilt when time/politics allowed. There was also some overly complicated mathematics but zero knowledge of computer science.

Aim

The aim of this mini-project is to start looking at how I would approach this problem if I were to return to the client and offer to work on a solution. To even attempt this is folly and it will need to be massively simplified for this question. In full the main components of this problem are:

- There are multiple caravan and lodge classes (2/3/4 bed, patio, sea/lake view, pets, smoking, ...)
- Standard holidays are Friday 7 night (F7), Saturday 7 night (S7), Monday 7 night (M7)
- Standard breaks are Friday 3 night (F3), Monday 4 night (M4)
- Seasonality of bookings including peak seasons of half-terms, summer, local events, easter
- Easter is particularly difficulty to code due to the ever changing date on which it falls
- Bookings and cancellations that may break the optimal path before the fleet mix can be updated
- Random events such as gypsy invasions can cause a holiday park to close temporarily

Clearly assumptions will have to be made!

Optimisation Goals

- Maximise available holidays by revenue
- Minimise days wasted (customers spend a lot of money in bars, restaurants, and entertainment)

This task will focus on maximising the holiday admixture to ensure the maximum number of days in a month that a caravan or lodge can be booked. By this, I will take one month of 31 days with the first day of the month being randomly selected. I will attempt to take a limited subset of caravan/lodge mixtures and mould them into the optimal number of holidays and breaks that reduces days wasted to a minimum. Some simplification assumptions for this are as follows:

- A holiday availability **cannot** run over the end of the month into the start of the next
- There are no bookings in the system, this is day 1 of a fresh holiday park if you like
- Only standard bookings will be used (F7, S7, M7, F3, M4)
- Seasonality will be ignored
- Holiday prices are static and not linked to external factors such as demand or current bookings

Data

Whilst I do have access to the company data as I am still on their books, it would be unprofessional to do so. I shall construct an artificial dataset from what I can remember using two types of accommodation (caravan, lodges) with different prices per type. The various holiday duration shall also be randomly priced. In total, there will be 10 accommodations split as 5 caravans and 5 lodges. There will be the 5 standard holiday types and each will be priced differently.

Model

The model will operate within the parameters of a 31 day month that begins on a Thursday (chosen by the excel function RANDBETWEEN(1, 7)). The task will be to take the 10 accommodations and assign them a mixture of 5 different holiday types (F7, S7, M7, F3, M4). This will be repeated over epochs so that the revenue is maximised and days wasted is minimised. The constraints are:

- The holiday duration must not extend outside of the month, there must be enough days to book
- Start of month is static and cannot be changed to suit better booking patterns
- 5% penalty will be applied per wasted day to encourage the minimisation of wasted days
- The only hard-coding allowed is where a day needs to be identified by the ‘F’, ‘S’ or ‘M’ codes

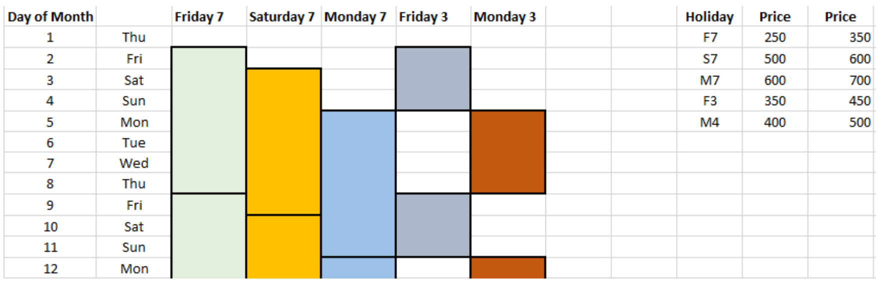
The model will use a vectorised approach to recording bookings due to the speed and ease at which vectors can be manipulated. All variables are parameterised into dictionaries for scalability.

The holidays are priced such that the Saturday 7 weeks and Monday 4 breaks are in direct competition. This is because a Friday 7 can be broken down into a Friday 3 and Monday 4, whereas a Saturday 7 cannot be broken down without incurring wastage. Breakdowns are future phase.

The 5% penalty will be calculated from the optimised revenue generated by each accommodation. This is not what happened in the real-life system but it is an interesting way of encouraging the optimiser to generate better solutions with less wastage. By penalising wastage and generating higher revenue solutions the benefit is synergised with On Park Spend by customers in real-life terms.

Solving the Problem

After struggling to decide where to begin, I drew a quick visual in Excel to aid the creative process. Here you can see how the different booking types might fill up a calendar:



Day of Month		Friday 7	Saturday 7	Monday 7	Friday 3	Monday 3
1	Thu					
2	Fri	🟩			▨	
3	Sat	🟩	🟨		▨	
4	Sun	🟩	🟨		▨	
5	Mon	🟩	🟨	🟦		🟧
6	Tue	🟩	🟨	🟦		🟧
7	Wed	🟩	🟨	🟦		🟧
8	Thu	🟩	🟨	🟦		🟧
9	Fri	🟩	🟨	🟦	▨	
10	Sat	🟩	🟨	🟦	▨	
11	Sun	🟩	🟨	🟦	▨	
12	Mon	🟩	🟨	🟦		🟧

Holiday	Price	Price
F7	250	350
S7	500	600
M7	600	700
F3	350	450
M4	400	500

Figure 2: Setting up the problem visually

Optimisation Method

The visualisation above helped me to realise that a random search would be ideal for this problem. By randomly choosing the bookings and storing them in a booking vector it would be easy to randomly select a holiday type and start date, and then check whether there are enough contiguous days to make the holiday viable. This led to a hybrid approach where initially the month is filled with randomly selected holiday types until small gaps are left, then to achieve minimisation of wasted days a mathematical approach takes over to analyse and select the correct break (F3 or M4) to fill the gaps. This minimisation approach came on the second iteration development after initial testing had highlighted a problem sometimes up to 80 iterations failing to randomly select the required holiday type and start date to fill the gap. Gaps of 3 days or less are not penalised as there are no holiday types to fill these gaps (an artificial constraint prevents spilling over into the next month).

The application of the booking penalty (5% of final revenue per wasted day) will make optimisations with too many gaps less profitable and unappealing to the optimiser's best optimisation check. Due to the limited time available it was not possible to develop the algorithm to remove troublesome bookings or turn weeks into breaks as would be required in real-life. It would not be overly difficult, just time consuming. With more time it would also be possible to utilise weightings to skew random selection in favour of higher priced holidays but this would require a degree of tuning.

Results

The algorithm worked better than expected after some teething problems. The booking vector typically filled up in 10-15 iterations before the iteration count would expire before smaller gaps of less than 7 days could be filled. The targeted mathematical approach that was implemented to scan for 3 and 4 day contiguous blocks that matched the F3 and M4 holiday types was incredibly successfully in reducing wastage to less than 3 days in most cases. Due to the artificial constraint of not allowing bookings to spill into the next month, having a day wasted at the start of the month and 2 at the end was acceptable and hard to avoid but with the mathematical approach filling almost all the gaps it was an acceptable result. The results can be found in the accompanying codebook. There is a weakness where 5 or 6 remaining days fails to trigger the gap catcher but this was left in to encourage penalisation and prevent the minimiser from doing any more work than necessary. This can happen when multiple groups of 1-2 day gaps cannot be penalised unfairly. Once assumptions are backed out for more advanced algorithms these would likely resolve naturally.

For each accommodation, the optimum is likely to be local as so much depends on how the initial bookings are chosen. As the booking vector fills rapidly, I opted for fewer steps and more epochs to maximise the chance of a good revenue selection. In real-life it is far easier to justify price increases if you are close to selling out but price increases are not variable in this model.

Conclusion

The model ran incredibly quickly for the 10 accommodations. With each park holding a couple of hundred accommodations for a medium sized park and a little of 60 active parks in total I would be optimistic that this could scale without the performance issues seen with the original solution. With the application of a specialised python library like numba even faster speeds could be achieved.

The main weakness of the model is the simplicity however the concept works well and the architecture of the design accommodates the removal of assumptions without too much effort:

- Accommodations and prices are stored in dictionaries for easy access
- Booking vectors can be extended without performance issue
- Constraint dictionaries can be added and scaled without issue (such as minimum holiday types)
- Existing bookings can be vectorised and added as a pre-populated booking vector

Whilst we spent a lot of time criticising the quality of the original work delivered it can be seen from this simple example that this really is an incredibly complex and difficult task to optimise, none more so than as a single person competing against what a team of a half dozen professionals were able to pour their combined abilities into. The model developed would be incredibly flawed if offered up as a final solution. Having said that, without the ability to efficiently search and break down the holiday stay durations into the optimal layout to fill the holiday park it would be pointless to add huge layers of complexity on top of this problem. Some of the assumptions that would need backing out to make this more useful would be:

1. The start day of the month is not constant and constantly changes, as do the prices
2. Bookings exist and need to be factored into the optimisation plan
3. This cannot be performed on an isolated month as the holiday year consists of 4 peak seasons that flow into each other, particularly around Easter and May half-term where the shoulder seasons heavily overlap even when they approach maximal separation
4. A minimum percentage of fleet admixture is required to give the holiday park a full feeling and to attract the long term repeat visitors that synergises the holiday experience
5. This would need to interact with other systems that control which bookings are allocated to the accommodations

There are four immediate changes I would make if time were not an issue at present:

1. Develop the solution in Object Oriented python for computational efficiency and abstraction
2. Add increased variability to the prices by differing by arrival date (this is simply data engineering)
3. Instigate a simple method of randomly removing a 7 day holiday around a small block of available days. The consultancy preferred to calculate every permutation of knock-on effects within a 3 week radius of the removed allocation. Taking a full grid search approach on a problem this size was the biggest issue with the efficiency of their approach (coding aside).
4. The optimiser sometimes returns 4 day block of zeros at the start of the month. This is because the first available day is a Thursday with no standard holiday type for that scenario. I would add a function that crawls through irregular contiguous patterns to try and fit a short break (e.g., F3).