Decisions in Finance from the Knapsack Problem Point of View

Jae-Moon Hwang, Matthew David Kearney,
Geir Agnarsson

George Mason University
Mason Experimental Geometry Lab

December 6, 2019

Starting off with a simple problem

- You are an investor and a client has given you a large, but finite, amount of money. You then compile a list of companies, their values, and their discrete value rate. For each company, you can either buy the whole company or not buy it.

- The question is: What is the best combination of companies in order to maximize your client's investment over time?

Knapsack Problem

- This type of problem is an example of the **knapsack problem**.
- Given a set, with each element having a weight w_i and value v_i, determine what combination of the elements gives the highest value while respecting a given constraint on the weights.
- We want to maximize

$$f(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} v_i x_i$$

while satisfying the condition

$$C(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} w_i x_i \leq W$$

where W is the maximum weight capacity.

Linear Programming

- **Linear Programming (LP)** is an optimization method that applies to mathematical models that can be characterized by linear relationships.
- With LP, the set of possible choices can be represented as a convex polytope due to the nature of using linear inequalities.
- Solutions are always at one of the vertices of this region. If more than one vertices yield the same maximum value, then the set of optimal solutions is a convex combination of these vertices.
- LP gives many possible algorithms in order to solve problems, many of which are efficient.

Integer Programming and Difficulties

- **Integer Programming (IP)** seeks to solve optimization problems in which variables are restricted to integer values.
- IP and LP are often at odds as restricting solutions to integers prevents LP from helping, as seen in certain variants.
- Compared to LP problems, IP problems are more complex, in the sense that the time required to solve an IP problem grows quickly as the number of items increases.

Solving the Knapsack Problem

- A general solution for the knapsack problem is not efficient, as it can only be solved using a brute force method. If we were to solve using a brute force method, it would be exponential in runtime.

- This is due to the constraint that we are choosing **discrete** items from a finite set. This means the knapsack problem is, evidently, an IP problem, and hence difficult to efficiently solve and optimize.

A bit of complexity theory

- In fact, the solving portion of the knapsack problem is part of a class of problems categorized as **NP-Complete**. This essentially means that there are no efficient algorithms better than brute force as the problem stands.

- However, while the general case is hard, we can examine variants that consider the problem with more, but reasonable, constraints.

Simplest Situation

- If we allowed x_i to be rational numbers between 0 and 1, then our algorithm could sort items by their value-to-weight ratio and take the most "valuable" items until capacity is reached.

- This LP algorithm can be done in linear time and is more efficient than the exponential-time, brute-force algorithm for the general case of the Knapsack Problem.

0-1 Knapsack Problem

- However, this is not as useful as we want to stay within discrete values. For our research, we instead look at a more well-known variant called the **0-1 Knapsack Problem**.

- This variation is what we focused on for our research, and it is analogous to the problem we posed at the beginning.

- For the 0-1 knapsack problem, we have the additional condition on the number of items:

$$C(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} w_i x_i \leq W \textbf{ and } x_i \in \{0, 1\}$$

Financial Point of View

- In our finance application, we look at the following scenario: An investor is looking to buy a collection of companies, c_1 to c_n, with corresponding prices, p_1 to p_n.
- For those prices, we assume the change in price, $\Delta p_i(t) = p_i(t+1) - p_i(t)$, is fixed around a neighborhood of time t.
- In this way, we want to maximize $\sum_i x_i \Delta p_i$ for which $\sum_i x_i p_i \leq B$ where B is the budget of the investor and $x_i \in \{0, 1\}$.

Solving the 0-1 Knapsack Problem

- In our research, we looked at simple cases where we limit the number of distinct weight values.
- If we put the restraint on the weights that

$$|\{w_1, w_2, ..., w_n\}| = 2$$

  then we can easily find an efficient solution.
- Additionally, this can be extended to n distinct weights, although even this process will get inefficient as n grows.
- The algorithm for 2 weights works as follows:

Algorithm

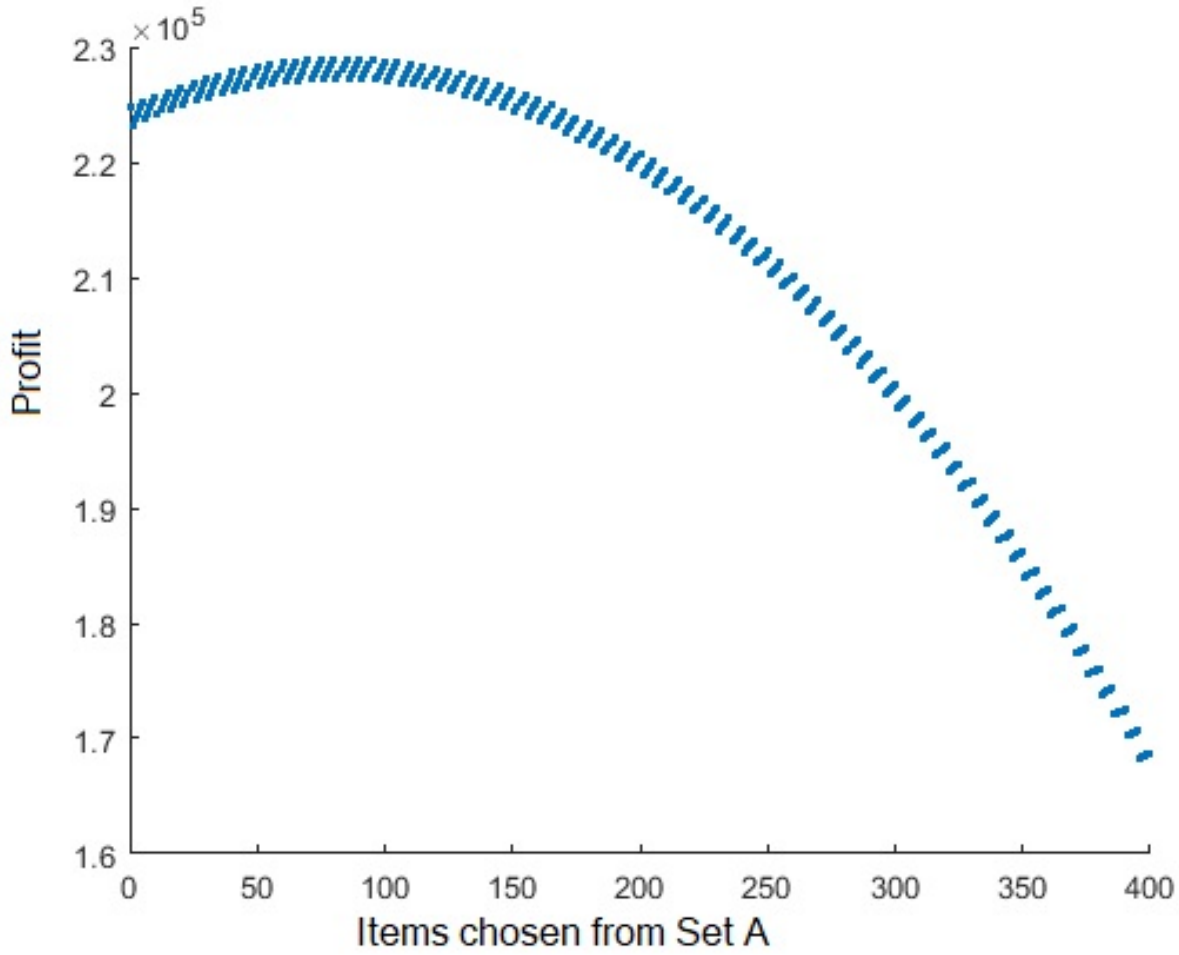
Data: Two arrays, A and B, where one holds the low weight items and the other hold the high weight items.
Result: The optimal combination of items to maximize value given a capacity.

sortByValue(A);
sortByValue(B);
while $i = 0 < min\{A.length, floor(capacity/A.length)\}$ **do**
    j $= min\{B.length, floor(capacity - i * low\_cost)/high\_cost\}$;
    for $s < i$ **do**
        bag_sum += $A[s]$;
    end
    for $s < j$ **do**
        bag_sum += $B[s]$;
    end
    if $bag\_sum > current\_max$ **then**
        current_max = bag_sum;
        opt_combination = (i, j)
    end
end
return opt_combination;

A bit more explanation

- This approach is easy to understand and is efficient. Simple analysis gives that this algorithm has a worst case run time of $O(n \log n)$. The solution is actually found in linear time, but we do have to sort which is bounded by $n \log n$.
- This algorithm can easily be modified to deal with any amount of weights as mentioned before.
- For 3 weights, we have an additional nested loop, so we get an upper bound of $O(n^2)$.
- This is because, if the sets of items are Set A, Set B, and Set C (items of which have weight a, weight b, or weight c), then the algorithm must consider every possible pair of n_a (items from set a) with n_b (items from set b)

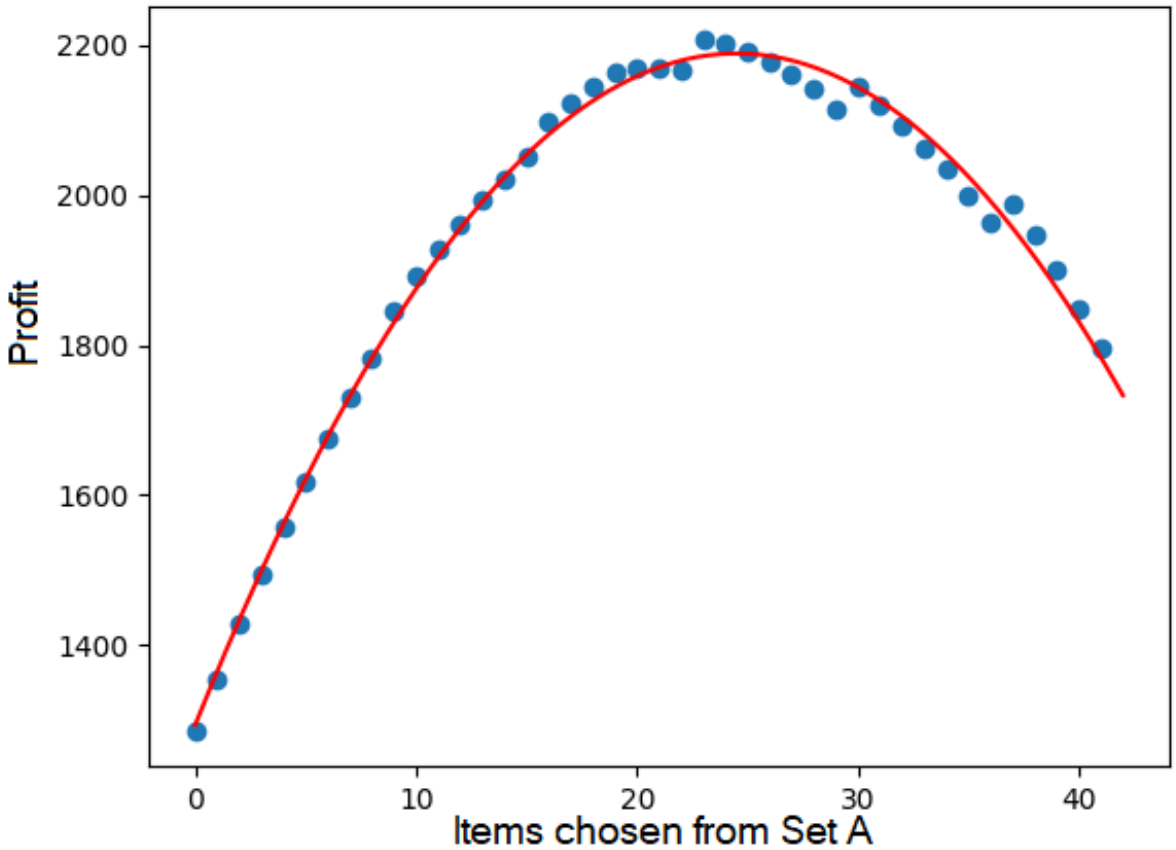
Geometry for 2 Weights



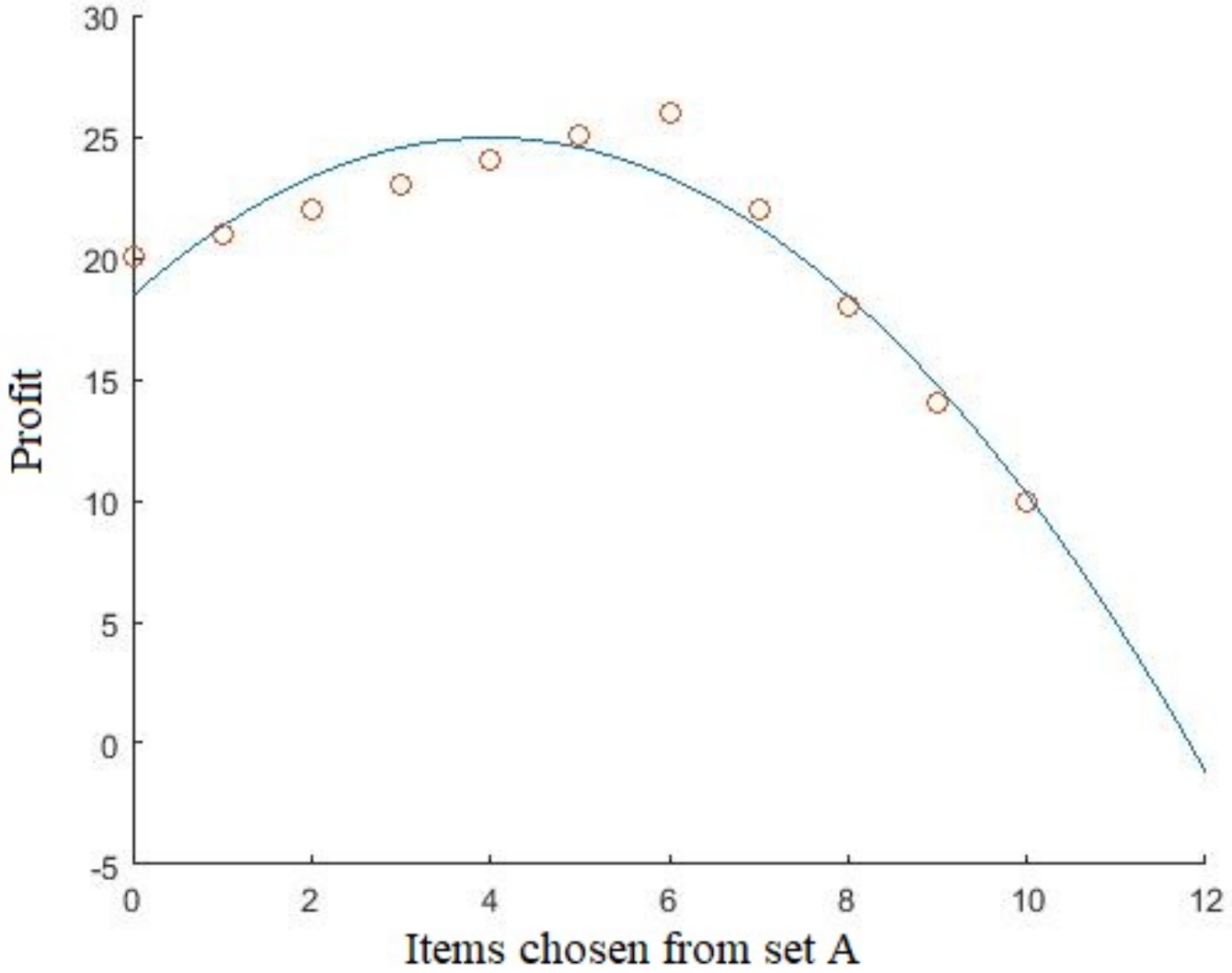
More Geometry

- We graphed the maximum profit as a function of the number of items chosen from array A, as in the algorithm.
- In the above case, the values of the items in array A and array B vary linearly, but the graph of the solutions can be visualized as a parabola.
- While the solutions form a parabolic shape, a parabola is not a reliable model for the output of the 2-weight algorithm.
- The maximum of the curve of best fit does not reliably model the best solution, even for large number of items.
- A similar algorithm for 3 distinct weights can be graphed.

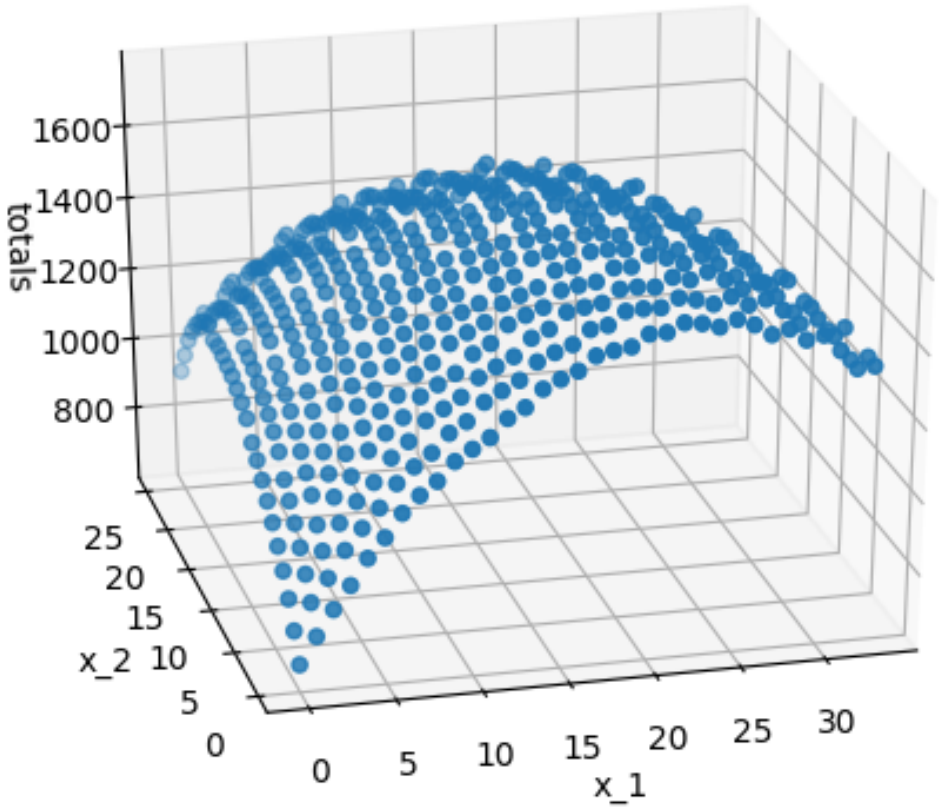
More jagged solutions



Curve of best fit is not really the best fit



Three Weights



A few observations

- As seen, the solutions are not monotonic as they increase or decrease and have a form of jaggedness to them.
- This is due to the variables being restricted to integers and the gaps in the weight difference allows for local "cups" where a previous sum is greater than the next and then jumps to a higher value again.
- This is then has a periodic nature as eventually the weights line up to fully utilize the capacity.
- This jaggedness also gets more drastic as the difference in weight values get smaller.
- This situation also happens in the three weight case as we get cups and a more jagged surface in a similar manner.

In the Long Run

- Another question we examined is about the long run profit of buying and selling companies.
- While we examined the case when we fix ourselves at a certain time t, what is the best way to buy and sell for all times?
- More specifically, can we maximize the profit from repeated buying and selling of companies by knowing each $p_i(t) = \sum_{\tau=1}^{t} \Delta p_i(\tau)$ over a long period of time?
- This would be a future task for our project to look closer into.

References

- "G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995."

- Alexander Schrijver, Theory of linear and integer programming. Wiley-Interscience Series in Discrete Mathematics. A Wiley-Interscience Publication. John Wiley and Sons, Ltd., Chichester, 1986. xii+471 pp. ISBN: 0-471-90854-1