Chapter 2
The Knapsack Problem

John J. Bartholdi, III
Georgia Institute of Technology

The "knapsack problem" appears in many forms in economics, engineering, and business: any place where one must allocate a single scarce resource among multiple contenders for that resource. It has acquired the fanciful name "knapsack problem" because our common experience of packing luggage expresses something of the flavor of the problem: What should be chosen when space is limited?

Introduction

Alice and Bob were excited about the bicycle tour they had long planned. They were going to ride during the day, carrying only light supplies, and stay in hotels at night. Alice had suggested they coordinate packing to avoid duplication and extra weight.

Alice was to pack tools: a compass, spoke wrench, chain breaker, hub wrench, and spare tire tube; and Bob was to pack consumables: Water, toilet paper, sunscreen, trail mix, soap. They agreed that neither would pack more than $c = 7$ pounds of supplies.

On the night before the trip Alice gathered everything on the kitchen table and prepared to pack; but it quickly became clear that she had staged rather more than 7 pounds. Decision time.

Alice was a perfectionist. She realized the importance of packing the right stuff and so was prepared to think carefully about it. She methodically indexed the candidate items and listed the weight w_i of each item i (Table 2.1, 2nd row).

But what to pack? Alice saw that she could fill the knapsack to the limit with items 1, 2, and 4. But then again she could just as well fill it with items 3 and 5 or with items 2, 4, and 5. Which is best? Or might it be better to pack only items 2 and 3 even if this alternative leaves some residual capacity unused? Why not pack only items 1 and 5 or only 3 and 4?

Alice realized that each item would either be packed or left behind—2 choices, and so there were up to $2^5 = 32$ possible ways to pack the knapsack. Not all of these are possible or even desirable, but it seemed as if she would have to consider each possibility explicitly and choose the best, whatever that meant.

Alice saw that, to choose the best, she needed some way to distinguish the "value" of one packing from another. Was it better to pack the spoke wrench or the spare inner tube?

D. Chhajed and T.J. Lowe (eds.) *Building Intuition*: *Insights From Basic Operations Management Models and Principles*.
doi:10.1007/978-0-387-73699-0,

Table 2.1 Weight in Pounds of Each Item to be Packed

Item	1	2	3	4	5
Weight (lbs)	3	2	4	2	3

Table 2.2 Relative Values of Each Item to be Packed

Item	1	2	3	4	5
Value	10	6	11	4	5

She thought she could answer simple questions like this, which asked her to compare the values of individual items, but it seemed harder to compare entire packings.

A good rule of thumb in making hard decisions is to formalize the choices in some way, in hopes of clarifying them. Alice did this by assigning "values" v_i to each candidate item i (Table 2.2, 2nd row).

Furthermore, it seemed reasonable to measure the value of a packing by the sum of the values of the items packed. Under this assumption it became clear that the packing of items 2, 4, and 5, which has a total value of 6 + 4 + 5 = 15, is less desirable, even though it fills the knapsack, than the packing of items 2 and 3, which does not fill the knapsack but has total value 6 + 11 = 17.

Alice saw that she could search for the best items to pack by the following process. Write out all 32 possible subsets of items, sum the values of the items in each subset, eliminate those that are not feasible (exceed the weight limit), and choose the subset of largest value. Tedious perhaps, but simple, straightforward, and guaranteed to produce a packing of maximum value.

Of course, this instance of a knapsack problem is so small that it presents scarcely any challenge. If Alice had had to choose from among, say, 20 items; then there would have been 2^{20} = 1,048,576 possibilities and Alice would likely have been awake all night to evaluate them all. Things can become much worse quickly for larger versions of this problem: To choose from among 100 items requires evaluating 2^{100} = 1,267,650,600,228,229,401,496,703,205,376 possibilities and it is not likely Alice could evaluate all of them within her lifetime.

General Structure of a Knapsack Problem

In her professional life Alice is a fund manager who oversees investment portfolios. She is currently considering over 100 potential investments and has estimated the return expected from each one. But each investment has a certain cost and Alice may not exceed her budget. Which investments should she choose? You will recognize this as having the same structure as Alice's problem of packing for the bicycle trip: How to choose a subset of items of maximum value while not exceeding a limit on total "weight." For the bicycle trip, "value" was a subjective expression of importance and "weight" was physical weight. For the investment portfolio, "value" is estimated return, perhaps constructed via a financial or economic model; and "weight" is the

Table 2.3 Bang-for-Buck of Each Item to be Packed

Item	1	2	3	4	5
Bang-for-buck	10/3	6/2	11/4	4/2	5/3

cost of investment. The common structure of these problems of choice is called the "knapsack problem" and it is distinctive in the following ways.

- We are faced with a number of yes/no decisions.
- The decisions are independent, except that each yes decision consumes some known amount of a single, common, scarce resource.
- None of the data will change during the time we make our decisions.

The most important part of this problem is the single, scarce resource. We are challenged to use it wisely to achieve the greatest value. For the bicycle trip, that resource was weight capacity; for the investment portfolio, that resource is budget. The knapsack problem is a way of looking at decisions that focuses on that single constraint.

In real life there are not many decisions with single constraints; but there are many in which one constraint matters more than others, and so the knapsack model is much more widely applicable than the idea of a single constraint might suggest. For Alice and Bob there may be an additional constraint in the physical volume that can be packed in the knapsack, but they may be justified in ignoring this constraint for now, perhaps because from previous experience they expect to reach the 7-pound limit well before filling the knapsack. Other constraints might not have well defined thresholds. For example, the cost of supplies may be a "soft" constraint: As more items are packed, Alice and Bob might feel increasingly uncomfortable at the thought of all they must purchase, but there is no clear threshold as there is for the capacity of the knapsack. In such cases, one identifies the most important constraint and models that are in the knapsack formulation. Then, after solving, check that the remaining, unexpressed constraints are satisfied.

For the bicycle trip, Alice and Bob have decided that weight is the most significant issue to them and they have stipulated a limit on it. After finding a good packing, Alice may check the resultant volume to see whether it is acceptable. If not, she might re-pose her problem as one of packing the most valuable load subject to a limit on the total volume. This would be a knapsack problem as well, but with volume as the scarce resource.

The knapsack problem Alice faces as a fund manager is too large to solve by considering every possible solution (a process known as "total enumeration"); but the economic context gave her some insight that helped simplify her decision-making. In the business world, one hopes to achieve a high return on investment (ROI), because this indicates efficient use of financial resources. Accordingly, Alice prefers an investment that promises a greater return per dollar invested ("bang-for-buck"). For the bicycle trip, analogous reasoning suggests that she should prefer to pack items that have a large value-per-pound ratio v_i/w_i, Table 2.3.

Alice realized she could avoid the work of total enumeration by simply choosing those items with greatest bang-for-buck. More formally, we can summarize the decision process as follows.

Step 1: For each item compute its bang-for-buck v_i/w_i.
Step 2: Sort the items from greatest to least bang-for-buck.
Step 3: Choose items from the top of the list until the next item (call it item k) does not fit.

Using this procedure to pack for the trip, Alice would choose items 1 and 2, which together weigh 3 + 2 = 5 pounds and are of value 10 + 6 = 16. We know this is not the best possible solution—for such a small problem you can surely see better packings—but it was very easy to generate. Moreover, we have some reason to believe the solution might not be too far from the best because there is an undeniable logic behind the procedure: Choose those items that best use the scarce resource.

Bob Packs

Bob was to pack the consumables, which, for pedagogical convenience, had exactly the same weights as given in Table 2.1. He noticed immediately that he had more choices in packing than did Alice: Because his items were divisible, he could repackage any of them and take only a portion; therefore his problem was to decide not which to pack, all or nothing, but the fraction of each item to pack. Like Alice, he saw that some items were more important than others. For example, he valued water more than soap, which they could do without if necessary. Bob listed the relative values of each item, which, again for pedagogical convenience, were identical to those of Table 2.2.

Building Intuition The procedure defined by steps 1, 2, and 3 is of a type known as "greedy" because it simply sorts possible choices by some measure of attractiveness (in this case, bang-for-buck) and chooses from the top. It never reconsiders decisions. However, since this procedure is not guaranteed to find the best answer, we call it a "heuristic." Hence choosing items by bang-for-buck is a "greedy heuristic." Greedy procedures to find solutions to decision problems are appealing since, in general, they are intuitive.

Although a greedy procedure does not always optimally solve the knapsack problem, there are instances of decision problems in operations management where such a procedure does find the best solution. One such example is the weighted completion time problem discussed in Chap. 1. As is shown in that chapter, a sequence (ordering) of tasks that minimizes the sum of weighted completion times is found by computing, for each task, the ratio of task time divided by weight and then sequencing the tasks in order of nondecreasing ratio values.

Now Bob's packing problem was similar to Alice's: How much of each item to pack so as to maximize the total value of a load that may not exceed 7 pounds.

Bob's Solution

Bob tried Alice's suggestion of choosing the items of greatest bang-for-buck; but when he had chosen items 1 and 2, he noticed that they weighed only 3 + 2 = 5 pounds, which left unused a capacity for 7 − 5 = 2 pounds. Because his items were divisible, Bob decided to go one step further and pack 2 pounds of item 3, which was the next-most-attractive. Two pounds of item 3 represented half its weight and so may be assumed to contribute half the value, or 11/2 = 5.5. This results in a packing that uses all available capacity and achieves a value of 21.5. Furthermore, as can be verified by inspection of this small example, this is the very best packing achievable. It provides the best packing possible in all instances because every single unit of the scarce resource is devoted to that item, or fraction thereof, that returns the greatest possible bang-for-buck.

Let us make Bob's decision process more specific, because, if we can formalize it, then we can program a computer to do it for us.

Step 1: For each item i, compute its bang-for-buck v_i/w_i.

Step 2: Sort the items from greatest to least bang-for-buck.

Step 3a: Choose items from the top of the list until the next item (call it item k) does not fit.

Step 3b: Take only as much of item k as will fill the remainder of the knapsack.

Note that, for any capacity and for any set of items selected in this manner, Bob will have to repackage at most a single item, the one selected in less than full quantity.

For this version of the knapsack problem, in which items are continuously divisible, the procedure is optimal: There is no alternative choice of items to pack that will have greater total value. Let us call the resultant value V^*. It is the largest possible value that can be realized from these candidate items subject to this weight limit.

Alice Tries Harder

Alice was pleased to see that choosing by bang-for-buck always generated the very best possible choices for Bob; but she saw that it would not work for her because her items were indivisible. Half a chain breaker would be useless: Each of her items had either to be packed in its entirety or else not at all and so she could not "top off" remaining capacity by adding just a little of another item.

Alice Finds a Guarantee

Alice saw that her procedure was similar to Bob's except that it stopped one step short[1]. (Step 3b). Surely her solution must be almost as good as Bob's, and Bob's was the best that could be hoped for. Surely, the value of the items greedily chosen could never be "too far" from V^*, the very best that would be achievable if Alice's items were divisible. Alice reasoned thusly: Suppose the heuristic halts, unable to pack item k entirely in the remaining capacity of the knapsack. If item k were divisible, so that I could pack it to fill the remaining capacity, it would add value $(v_k/w_k) \times$ (the remaining available capacity of the knapsack) to what had been already packed and this would be the best possible packing. But my items are not divisible and so this solution is an ideal, possibly unachievable. If I stop here, omitting item k, then the value of what I have packed must be within

$$(v_k/w_k) \times \text{(the remaining available capacity of the knapsack)} \qquad (1)$$

of V^*.

In short, Alice cannot have missed optimal by more than the value of that k-th item, the first that that did not fit.

How Wrong Can Alice Be?

Expression (1) bounds the opportunity Alice might forfeit by accepting an easy-to-compute solution rather than insisting on an optimal solution. Here is an example that embarrasses the greedy heuristic: Consider a knapsack of capacity c for which two items contend. One is of weight 1 and value 1; the other is of weight c and value just shy of c, call it c-ε. The greedy heuristic would choose the first item and halt, because the second item no longer fits. The total value achieved would be 1; yet if the second item had been chosen the total value would have been c-ε. Since c could be any large number, the error, both relative and absolute could be arbitrarily large.

That is the worst case. What might we actually expect? In some circumstances this worst-case error is small, perhaps small enough that we would feel comfortable accepting it. For example, we would expect the worst-case error (1) to be small and the solution to be quite close to V^* whenever any of the terms of (1) are small; that is, in any of the following situations.

- If v_k is small compared to the total value packed; or
- If v_k/w_k is small compared to the average bang-for-buck of the packed items; or
- If the remaining capacity of the knapsack is a small fraction of the total.

[1]Alice's procedure could be made a little more effective by having it try to fit each successive candidate item into the knapsack rather than stopping at the first failure. This will not affect our subsequent discussion.

All of these conditions, and especially the third one, may be expected to hold if most items are small with respect to the capacity of the knapsack. In such case we are packing lots of small items into a large container, which, as we know from experience, is easy to do well (think socks in a suitcase). Many items will fit before the first one fails, and consequently that item may be expected to have a relatively small bang-for-buck v_k/w_k. This is so for the simple reason that we will have already packed those items that have the greatest bang-for-buck. Moreover, because the items are small with respect to the capacity of the knapsack, we would expect any capacity unused after Step 3 to be small. Consequently we would expect the solution to be quite close to V^*, the upper bound. For example, if no item weighs more than x% of the weight limit, then we can be sure that a greedy packing will always fill the knapsack to within x% of its capacity and provide total value within x% of the maximum possible. Statistics are on our side here.

Is It Good Enough?

Is it "good enough" to know that your solution is close to the best possible? The answer to this question is the ubiquitous "it depends." In particular, it depends on how much you are willing to pay in money, time, and effort to get a better solution. It is probably not worth worrying over if you are packing a knapsack for a bike trip; but it may well be worth anguish if you are loading a space vehicle for an expedition to Mars.

There are some situations in which it makes sense to accept an approximate solution, even if you wish for the best. For example, the best may not be well defined when the data are uncertain. For the bike trip, the weights of the items to be packed can easily be measured and everyone can agree on them, assuming we have a scale of sufficient accuracy. It is not so clear how to measure value. It may be that Alice and Bob disagree on the value of sunscreen; indeed, Alice might be uncertain herself exactly what number to assign to the value of a compass. When the data are uncertain, the expense of careful optimization may not be worthwhile.

Alice Is Difficult to Please

Alice was a perfectionist. She took pride in owning a top-of-the-line touring bicycle, which had been fine-tuned for lightness and strength. She was dissatisfied that the greedy solution to her knapsack problem might not be the best possible. She wanted to know for sure whether it was the best; and if it was not … well, then she wanted the best. As we saw, Alice could be sure by methodically evaluating all possible solutions and choosing the best. But this is practical only when the number n of candidate items is quite small because the work to evaluate 2^n possibilities increases very rapidly in n and so is impractical for any but the smallest instances of the knapsack problem.

It is possible to compute an optimal solution for fairly large problems by trying carefully to fit items into the knapsack, adding one after another, possibly removing some if we have come to an impasse, and resuming. We can formalize such a process by representing it as a network that summarizes the sequence of decisions about what to pack, the resultant state of the partially packed knapsack, and the remaining choices available to us. For example, Fig. 2.1 shows the network of decisions representing Alice's problem if she, unlike Bob, could not split any item.

There are $1 + n = 6$ columns of vertices indexed by $i = 0, 1,\ldots,n$; the first vertex corresponds to the start (an empty knapsack) and each of the next 5 columns corresponds to the disposition of an item. Each column is composed of $1 + c = 8$ rows indexed by $w = 0, 1,\ldots,c$, each corresponding to one additional unit of weight. Each vertex (i,w) represents a possible state of a partially packed knapsack, the state in which items $1,\ldots,i-1$ have been considered and either packed or rejected, resulting in a knapsack of weight w. Each edge is directed from left to right, bottom to top, and represents the inclusion or exclusion of an item. In particular:

- Any edge from $(i-1,w)$ to (i,w) represents the exclusion of item i: We have considered item i but not packed it and so the cumulative weight of the partially filled knapsack remains w. Such an edge is assigned length 0.

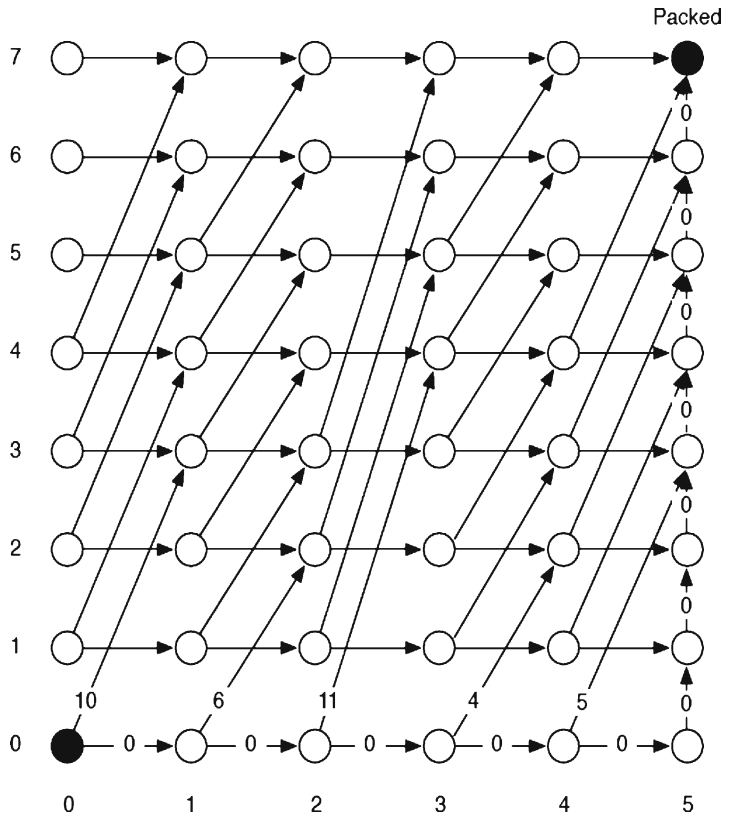

Fig. 2.1 Alice's decision modeled as a problem of finding the longest path through a network

- Any edge from *(i-1,w)* to *(i,w + w_i)* represents the inclusion of item *i*: We have considered item *i* and packed it and so the cumulative weight of the knapsack has increased by w_i. Such an edge is assigned length v_i.
- Any edge from *(n,w)* to *(n,w + 1)* represents the decision to leave one unit of capacity of the knapsack unused. Such an edge is assigned length 0.

NOTE: To avoid clutter in the figure, only a very few edges have lengths indicated on them. The complete figure would have a length indicated on each edge.

Here is the key insight: Any path from the bottom-left origin *(0,0)* to the vertex on the top right *(n,w)* corresponds to a sequence of decisions of what to put in the knapsack and what to omit. We want a selection of items that gives greatest value, which means we want the longest-path from *(0,0)* to *(n,c)*.

Fortunately, there are simple methods to compute the longest path in a network such as this, in which all edges point in the same direction (so that there are no cycles). The standard method is called dynamic programming and it works like this:

- Start at the last column and label each vertex in this column with 0, which is the length of the longest path from it to vertex *(n,c)*.
- Working backward to preceding columns, label each vertex in the current column with the largest value of: the length of an edge departing this vertex plus the label of the vertex on the other end of that edge. This will be the length of the longest path from the current vertex to vertex *(n,c)*.

When we have finished labeling the first column, the label of vertex *(0,0)* will give the length of the longest path to vertex *(n,c)*, which will also, by the clever way we have built the network, be the maximum value of any way of packing the knapsack. The path of this length, which we can discern by the pattern of labeling, tells us exactly which items to pack and which to leave, Fig. 2.2.

As you will have surmised, this method is tedious for a human but simple for a computer. One can think of it as merely a clever way of organizing a search. It reduces the work by taking advantage of the observation that the best-packed knapsack of capacity *c* must also contain the same loads as the best-packed knapsacks of smaller capacity. In any event, this methodical search takes substantially less time than evaluating every one of the 2^n possible loads.

The Cost of Finding Optimum

To find the longest path on the network above requires us to examine about *nc* vertices[2], and so we may take this as an estimate of the total work required. Notice that this depends on the number of items to be packed, which seems reasonable; but it also depends on the capacity *c* of the knapsack. This is more troubling because it seems

[2]Some vertices are clearly spurious to the solution, such as those in the upper left-hand corner, but it is generally not worth the effort to identify those that can be ignored.

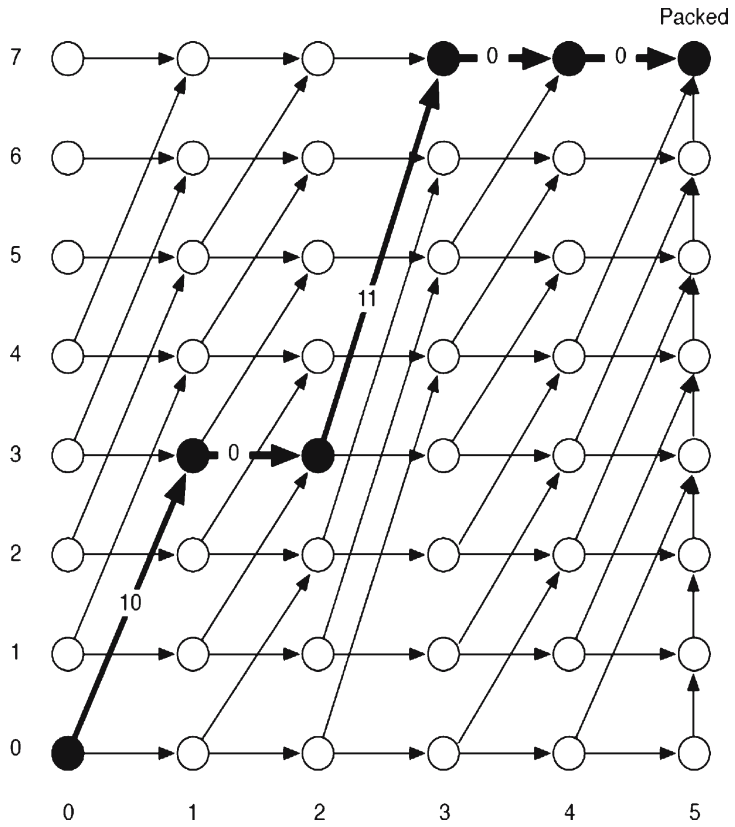

Fig. 2.2 The longest path corresponds to packing items 1 and 3 and omitting the remainder. The total value is 21, the length of the path

to suggest that the work can depend on how accurately we measure the capacity. For example, the network to represent Alice's problem would have been larger if she measured all weights to the nearest ounce, and larger still if she measured to the nearest tenth of an ounce. This observation works in the other direction as well. Alice could reduce the size of the network, and presumably the work of computing a solution, by measuring less accurately: to the nearest kilogram instead of to the nearest pound. This reveals an interesting feature of the knapsack problem: The greatest determinant of the work to construct an optimal solution is the precision of the weights w_i more so than the number of items n. The more precise the data, the more work is required to take advantage of that precision. The work increases exponentially with any increase in precision but only linearly in the number of items. For example, if capacity of the knapsack had been specified as 7.01 pounds instead of 7 then the network would require 701 rows of vertices and the work to solve would have increased by a factor of 100.

Pedaling Into the Sunset

As Alice and Bob have determined, we can approach any problem with knapsack structure via the most appropriate of the following:

- If there are few items from which to select, we can simply enumerate all possibilities and choose the most valuable subset of items.
- If there are a moderate number of items from which to select or if the precision of the weights is not too great then we can compute the optimal solution by dynamic programming, as on the network above.
- If there are very many items contending for selection or if the data are very precise (so that it is impractical to compute an optimal solution) or if the data are very imprecise (so that an optimum solution is not to be relied upon), then the greedy heuristic—choosing items of greatest bang-for-buck—is suitable.

In many practical applications, there is much merit in Alice's greedy heuristic if the potential shortfall in quality of solution can be tolerated. Besides ease of computation, Alice's solution has the advantage that it is simply a sorted list of all the items, ranked from greatest to least bang-for-buck. If the bang-for-buck of an item were to change, it is a simple matter to move this item to its new position in the sorted list. This allows us to incrementally adjust the knapsack solution as data changes, so that the greedy solution can support *dynamic decision-making*. That is, we might monitor items and remove or insert them as their values or weights change or as the capacity of the knapsack changes. This would be useful, for example, in maintaining an investment portfolio that must adapt to changes in the financial marketplace, or in keeping the right product stored in the right locations of a distribution center even as the patterns of customer orders change.

Applications of the Knapsack Problem

Among the straightforward applications of the knapsack problem are, unsurprisingly, problems of loading shipping containers especially when one of weight or volume is known in advance to be the limiting constraint. These issues can be quite significant when space is scarce, as is the case when manufactured product is shipped to the US from China in advance of the holiday selling season.

Another common application is to cutting stock problems. For example, paper mills produce huge rolls of paper, the dimensions of which are determined by the manufacturing process. These roles are subsequently sliced into smaller rolls to fill customer orders. The value of the smaller rolls depends on the selling price. A similar problem is faced by manufacturers of fiber optic cable, who must decide how to cut lengths of cable to satisfy customer orders while extracting the greatest value from each length of cable.

As suggested earlier, the knapsack problem is a basic tool of portfolio optimization, where budget is almost always the most important constraint. Many portfolio

optimization problems generalize the knapsack problem in ways that more exactly model economic phenomena. Thus it may be possible to purchase 0, 1, 2, or more shares of an investment; and the additional shares may bring diminishing returns. Many of the ideas mentioned by Alice and Bob can be extended to account for such additional complexities.

In a warehouse or distribution center (DC), space is frequently a scarce resource because of inevitable growth in the number of products handled. There is competition among the stockkeeping units for storage in the most convenient areas of the DC, where customer orders can be filled most quickly. This can be modeled as a knapsack with space as the limited resource and labor-savings as the value of storing a product in a convenient location.

Many scheduling problems can be posed as knapsack problems with machine time the most important scarce resource. This is especially appropriate when the machine represents a large capital investment. In such case it can make economic sense to schedule the machine, perhaps by solving a knapsack problem, and then purchasing whatever additional resources, such as workforce or transportation or raw materials, as are necessary to meet the schedule.

The knapsack appears as a sub-problem in many, more complex mathematical models of real world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy ignored constraints.

Historical Background

The knapsack problem seems to have first been identified in print in 1957, in two important publications. One was a paper by George Dantzig (1957), one of the developers of linear programming and a creator of the field of Operations Research. He showed that the continuous version of the knapsack problem (the one faced by Bob) is perfectly maximized by choosing items by bang-for-buck.

The problem must have been a topic of conversation amongst the early specialists in discrete optimization because in the same year Richard Bellman, another important early figure in Operations Research, described how to use dynamic programming to solve the knapsack problem. (This is equivalent to the method above in which we find the longest path on a special network.) Very quickly thereafter the knapsack model was applied to a range of applications, including most of those listed above.

Throughout the 1980s there was much work on approximation algorithms, solution techniques that cannot be guaranteed to produce optimal solutions because they strategically give up some quality to achieve speed of execution. The knapsack problem was a popular target for the development of such approximation methods. Indeed, one of the first "polynomial approximation schemes" was developed for the knapsack problem by Sahni (1975). Such schemes may be thought of as solution methods in which one may specify a desired guarantee for the quality of solution in advance of solving. As to be expected, the work to solve increases quickly with the quality required. Immediately afterwards, this was improved by Ibarra and Kim

(1975) to a "fully polynomial approximation scheme," which makes the trade-off between quality of solution and effort slightly more favorable. Specialized solution techniques to solve the knapsack problem and its variants are provided by Martello and Toth (1990). More recently, researchers, such as Kellerer et al. (2005) have worked on ways to exactly solve ever larger instances of the knapsack problem. Many of these involve solving some "core" of the problem and then building this partial solution to a full solution.

Interestingly, the knapsack problem figured prominently as the first suggested basis for a public key encryption system. This early work is described in Diffie and Helman (1976) and Merkle and Helman (1978). It should be noted that the approach was later "cracked" by cryptographers and replaced by more resistant schemes.

As an example of the flexibility conferred by the simplicity of the greedy algorithm, Bartholdi and Hackman (2006) make extensive use of the knapsack problem as part of a larger model to cache product in a distribution center.

Selected Bibliography

Bartholdi, J. J. III and S.T. Hackman, (2006). *Warehouse and Distribution Sciences*. Georgia Institute of Technology, Altanta, GA. This book is freely available at www.warehouse-science.com.

Bellman, R. (1957) *Dynamic Programming*, Princeton University Press Princeton, N.J., Reprinted (2003) Dover Publications, Mineola, N.Y.

Dantzig, G. (1957). "Discrete variable extremum problems", *Operations Research* 5,266–277.

Diffie, W. and M. Helman, (1976) "New directions in cryptography", IEEE Transactions on Information Theory 22(6):644–654.

Ibarra, O. H. and C. E. Kim, (1975) "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems", Journal of ACM 22,463–468.

Kellerer, H., U. Pferschy and D. Pisinger (2005) *Knapsack Problems*, Springer, Berlin, Germany.

Martello, S. and P. Toth, (1990). *Knapsack Problems: Algorithms and Computer Implementation,* John Wiley, Ltd. This book is freely available at www.or.deis.unibo.it/knapsack.html.

Merkle, R. and M. Helman, (1978). "Hiding information and signatures in trapdoor knapsacks", IEEE Transactions on Information Theory 24(5),525–530.

Sahni, S. (1975). "Approximate algorithms for the 0–1 knapsack problem", *Journal of ACM* 22:115–124.