Greedy Algorithm in Books Arrangement Problem

Okiriza Wibisono / 13509018
Informatics Engineering
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jalan Ganesha 10 Bandung 40132, Indonesia
okiriza.wibisono@students.itb.ac.id

***Abstract*—Despite its simplicity, greedy algorithm is well known for its efficiency in many instances of optimization problems. One such instance is "The Integer Knapsack Problem", in which there is a knapsack, with certain maximum capacity, to be filled with things in such a way that maximizes the sum of value of all the things put in the knapsack. This paper discusses a problem similar to the integer knapsack problem: how to fill bookshelves in a bookstore to get the most profit.**

***Index Terms*—book, greedy algorithm, optimization, solution.**

I. INTRODUCTION

Imagine you have just opened a bookstore not far from your campus. For starters, you purchased two hundred books and five bookshelves. You may start to wonder: how should you arrange these books, probably not all of them, into the bookshelves so that people visiting would find your bookstore convenient and you would (hopefully) get the most profit?

Undoubtedly, a bad arrangement of books will cause confusion to customers and denial of potential income. A student probably won't like having to waste his time wandering all the way around your bookstore only to find a textbook. Moreover, a bad arrangement may make it difficult to monitor book stocks and to maintain a consistent database. A technique (or algorithm) needs to be devised to overcome these problems associated with bad arrangement of books.

Fig. 1. A good arrangement of books attracts more customers

In almost every bookstore, the books are arranged according to a certain parameter. For example, the books may be arranged alphabetically on their titles. While this may be the simplest arrangement, arranging books by their titles hinders a class of customers: impulsive customers. If this kind of arrangement is applied, customers who come to the bookstore without prior planning on what to buy will almost likely not find the books they like, as the books are sorted only on their titles. Also imagine what will happen if a newly stocked book must be put into an already full bookshelf. Should the new book be put elsewhere, or all the books with titles lexicographically greater than it rearranged to make some place? Surely this will cause other problems and won't make the customers or the bookstore owner happy.

An obvious, yet effective, arrangement is sorting books according to their categories. In this way a customer who is, say, a novel-addict can easily scan through numerous novels in a novels section and grab some which he likes. In fact, it is very common to find bookstores labeling their bookshelves with category names, not alphabet.

Another possible arrangement is sorting books on their categories plus their sets. For example, books of a certain trilogy should be put adjacent to each other. While certainly better than arrangement solely by category, it is more complex and therefore not discussed in this paper.

This paper walks through the application of greedy algorithm to solve books arrangement problem, precisely one which bases the arrangement on category. We will see in later sections how to get the best books arrangement given some bookshelves with a certain capacity, some book categories, and (of course) some books.

II. THEORETICAL BASIS

Greedy algorithm is the most popular algorithm for **optimization** problems. Optimization problem is defined as how to obtain the optimal solution given an instance of the problem. Optimization problem is further classified into 2 types: **maximization** and **minimization** problem.

Greedy algorithm takes as its principle the statement "take what you can get now". This means that greedy algorithm will immediately take (or choose) the best possible option at the moment. By taking the best option at each step (**local optimum**), it hopes to reach the best solution there is (**global optimum**).

In shorter words, greedy is an algorithm which:

1. Is stepwise,
2. Takes the best option available at a moment without considering the consequences, and
3. Hopes that taking all the local optimums leads to the global optimum.

Greedy algorithm is represented by 5 elements:

1. Candidate set, C,
2. Solution set, S,
3. Selection function,
4. Feasibility function, and
5. Objective function.

Candidate set (C) consists of all the elements which will build the solution. An element of candidate set may itself be a set, as we will see in this paper later. At each step, an element is taken from candidate set and inserted into **solution set** (S) (hence candidate set and solution set must be sets of the same type of elements). Solution set consists of the selected elements which, as a whole, represent the solution of the problem instance. Since its elements are taken from the candidate set, solution set is a subset of candidate set.

Selection function selects, at each step, an element from candidate set which will be inserted into the solution set. Based on the objective of the greedy algorithm, selection function assigns a value to each candidate in candidate set and selects one candidate which has the maximum, or minimum, value and removes it from candidate set. A greedy algorithm for one problem may have more than one selection functions.

Before the candidate selected by selection function is put into solution set, **feasibility function** tests whether it is feasible to put that candidate into solution set. By feasible it means that, together with all the candidates in solution set, the new candidate does not violate an existing constraint on the solution set. A "feasible" selected candidate will be inserted into solution set. If the selected candidate fails to pass the feasibility test, another will be taken from the candidate set. This process is repeated until there are no more elements left in candidate set, or until a certain goal has been achieved by the current solution set. The last function, **objective function**, is a function which maximizes (or minimizes) the sum of value of the solution.

Here is the general scheme of greedy algorithm:

1. Initialize solution set as an empty set.
2. While solution set doesn't represent the solution wanted and candidate set is not empty, perform the steps below.
3. Remove a candidate from candidate set (by calling selection function).
4. Test the selected candidate with feasibility function. If it passes, add it to solution set.
5. Repeat step 2.

Finally, it is important to note that **for some problems greedy algorithm does not yield the best solution**, since two things apply:

1. Greedy algorithm bases its decisions only on the current available options, not considering the consequences or alternative solutions.
2. There may be more than one selection functions; therefore we must choose the correct one if the algorithm is to give the best solution possible.

While this makes greedy algorithm not completely reliable, it surely keeps greedy algorithm simple and efficient to implement, thus its popularity in optimization problems.

III. APPLICATION

As our problem seeks to maximize the sum of value of all the books in the arrangement, it is a good candidate for applying greedy algorithm. Before we define the elements of the greedy algorithm as explained in the previous section, we should define the problem statement.

"Suppose there are $A_1, A_2, \cdots, A_p$ bookshelves, each may contain up to L books. Suppose there are $n_1, n_2, \cdots, n_m$ books of category, in exact order, $K_1, K_2, \cdots, K_m$. Each bookshelf may contain books of one and only one category. Let b_{ij} denote the j^th^ book of category i, and v_{ij} denote its value ($1 \le i \le m$ and $1 \le j \le n_i$). How do we put the books in the bookshelves so that we get the biggest sum of value of all the books in the bookshelves?"

From the problem statement above, we can define the elements of the greedy algorithm:

1. Candidate Set

   The candidate set is all sets of books, with each set containing books of the same category. Mathematically written, the candidate set is

$$C = \left\{ A \,\middle|\, A = \left\{ b_{ij} \,\middle|\, \begin{matrix} 1 \le i \le m, 1 \le j \le n_i, \\ \forall\, b_{i1j1}, b_{i2j2}, i_1 = i_2 \end{matrix} \right\} \right\}$$

   Though looks a bit complicated, this equation simply states that C is all the subsets of all the books that haven't been placed into a shelf, with each subset consisting of books of the same category. For instance, $A_1 = \{b_{13}, b_{17}, b_{18}\}$ is an element of C but $A_1 = \{b_{13}, b_{25}, b_{18}\}$ is not, since A_2 consists of books of two categories, category 1 and category 2.

2. Solution Set

   The solution set is probably the hardest element to express. It contains the set of bookshelves which gives the maximum value for the problem instance, such that each bookshelf passes the feasibility function (described below) and contains books of one and only one category, and (naturally) a book cannot exist in more than one bookshelves.

$$S = \left\{ A \left| \begin{matrix} A = \left\{ b_{ij} \middle| \begin{matrix} 1 \leq i \leq m, 1 \leq j \leq n_i, \\ \forall b_{i1j1}, b_{i2j2}, i_1 = i_2 \end{matrix} \right\}, \\ \forall b_{i1j1} \in A_1, b_{i2j2} \in A_{2,} b_{i1j1} \neq b_{i2j2}, \\ F(A) \end{matrix} \right. \right\}$$

where $F(A)$ denotes that A passes the feasibility function F.

3. Selection Function

   Since we will focus on how to merely arrange the books based on their category, we do not seek to compare a number of selection functions. We can then give value to each book as we see fit. For example, the value of a book might be its price, its number of inquiries in the last month, other properties of the book, or some function of them. Thus, the selection function will only select the bookshelf with the maximum sum of that value.
4. Feasibility Function

   From the problem statement, we can easily infer the feasibility function: a candidate bookshelf may contain up to L books, or "the maximum number of elements in a candidate is L".
5. Objective Function

   The objective function is to maximize the sum of value of all the bookshelves. The value of a bookshelf itself is the sum of value of all the books in it. In other words, the objective function is to maximize

   $$O = \sum_{i=1}^{p} v(A_i) = \sum_{i=1}^{p} \sum_{j=1}^{|A_i|} v_{ij}$$

   where $v(A_i)$ denotes the value of the ith bookshelf.

IV. EXAMPLE

Having defined the elements of the greedy algorithm, we can see how we apply the algorithm in a real example.

Suppose we have 3 bookshelves with capacity of 4 books ($p = 3$ and $L = 4$). Our list of books is summarized in the table below:

Table 1. Books List

Number	Category	ID	Value
1	1	1	57
2	1	2	37
3	1	3	24
4	1	4	9
5	1	5	92
6	2	1	50
7	2	2	99
8	2	3	65
9	2	4	89
10	2	5	53
11	2	6	39
12	2	7	98
13	2	8	62
14	3	1	37
15	3	2	11
16	3	3	45
17	3	4	22
18	3	5	77
19	3	6	6
20	3	7	0

From this table, we can infer that there are 3 categories of books ($m = 3$), with 5 books of category 1 ($n_1 = 5$), 8 books of category 2 ($n_2 = 8$), and 7 books of category 3 ($n_3 = 7$). As there are 3 bookshelves, the greedy algorithm will be done in 3 steps.

1. Step 1

At the beginning of each step, we construct the candidate set. In this step, the candidate set is:

Table 2. Candidate Set for Step 1

Candidate	Sum of Value
{ }	0
$\{b_{11}\}$	57
$\{b_{12}\}$	37
…	…
$\{b_{11}, b_{12}\}$	94
…	…
$\{b_{11}, b_{12}, b_{13}, b_{14}, b_{15}\}$	219
$\{b_{21}\}$	50
$\{b_{22}\}$	99
…	…
$\{b_{22}, b_{23}, b_{24}, b_{27}\}$	351
…	…
$\{b_{21}, b_{22}, b_{23}, b_{24}, b_{25}, b_{26}, b_{27}, b_{28}\}$	555
$\{b_{31}\}$	37
…	…
$\{b_{31}, b_{32}, b_{33}, b_{34}, b_{35}, b_{36}, b_{37}\}$	198

It would be too long to enumerate all the candidates, so I only show a few of them.

After constructing the candidate set, we proceed to select one which gives the maximum sum of value, which is $\{b_{21}, b_{22}, b_{23}, b_{24}, b_{25}, b_{26}, b_{27}, b_{28}\}$ (555 sum of value), and remove it from the candidate set. Before we insert it into the solution set (which is still an empty set now), we must test it with the feasibility function. Since it has 8 elements, it does not pass the feasibility function, and we should repeatedly seek to find another candidate. Among all candidates that pass the feasibility function, the candidate which has the greatest sum of value is $\{b_{22}, b_{23}, b_{24}, b_{27}\}$ (351 sum of value). Therefore, we insert it into the solution set and conclude step 1. The current solution set is $\{\{b_{22}, b_{23}, b_{24}, b_{27}\}\}$.

One last task in this step is removing the books $b_{22}, b_{23}, b_{24}, b_{27}$ from our list of books (and putting them into the first bookshelf). We will not consider these books in the remaining steps.

2. Step 2

Using the same method as in step 1, we select the candidate $\{b_{11}, b_{12}, b_{13}, b_{15}\}$ (210 sum of value). Be wary that the candidate set in each step is different, as we do not consider books that have been put into a bookshelf. After inserting the candidate into the solution set and removing the books in it from our list of books, we can proceed to the final step.

The solution set is now $\{\{b_{22}, b_{23}, b_{24}, b_{27}\}, \{b_{11}, b_{12}, b_{13}, b_{15}\}\}$ (561 value total).

3. Step 3

The best candidate for this step is another set of books from category 2: $\{b_{21}, b_{25}, b_{26}, b_{28}\}$ (value = 204). Adding it to the solution set, we get the final solution set, $\{\{b_{22}, b_{23}, b_{24}, b_{27}\}, \{b_{11}, b_{12}, b_{13}, b_{15}\}, \{b_{21}, b_{25}, b_{26}, b_{28}\}\}$, which gives us 765 sum of value. The books arrangement can then be summarized in the table below.

Table 3. Final Books Arrangement and Total Value

Bookshelf	Category	Books	Value
1	2	$b_{22}, b_{23}, b_{24}, b_{27}$	351
2	1	$b_{11}, b_{12}, b_{13}, b_{15}$	210
3	2	$b_{21}, b_{25}, b_{26}, b_{28}$	204
Total			765

Analysis

Though this example doesn't quite represent a real-world example (where can you find a bookstore which has only 20 books?), we can still perform some analysis.

We see that it is possible to put books of the same category in two or more bookshelves (category 2 in this example), as long as their sum of value is the greatest of all the candidates that pass the feasibility function. It is also possible that books of a certain category are not put in a bookshelf at all (category 3 in this example).

A curious person might ask, "Does this greedy algorithm always yield the best solution for the books arrangement problem?" The answer is yes, since we select, at each step, the best candidate available, hence for each element (bookshelf) in the solution set, its value is greater than the value of every remaining books candidate (which passes the feasibility function) after the books are removed from the list of books.

Finally, there is an improvement to the algorithm, although this improvement will make the algorithm deviate slightly from the original greedy algorithm. By rewriting the selection function, we can safely not construct the candidate set and still get the best candidate in each step. Moreover, the selected candidate is guaranteed to pass the feasibility function and all the books in the candidate will automatically be removed from the list of books. Here is the pseudo-code for the selection function.

```
function selection(books: list of book; numCateg, capacity: integer) → candidate
{Returns the candidate which has the maximum sum of value from the list of books. The candidate returned is guaranteed to pass the feasibility function.}

LIBRARY
bookCategs : list of candidate {initially each candidate is an empty set}
bs, maxBooks : candidate
b, min : book
i: integer
vals : array of integer [1..numCateg]

ALGORITHM
i traversal 1..numCateg
   vals[i] ← 0

i traversal 1..books.size
   b ← books[i]
   bs ← bookCategs[b.category]
   if (bs.size < capacity) then
      bs.add(b)
      vals[b.category] ← vals[b.category] +
         b.value
   else
      min ← findMin(bs)
      if (b.value > min.value) then
         bs.remove(min)
         bs.add(b)
         vals[b.category]← vals[b.category]
            - min.value + b.value

maxBooks ← bookCats[maxIdx(vals)]
i traversal 1..maxBooks.size
   books.remove(maxBooks[i])

→ maxBooks
```

```
function findMin(books: candidate) → book
{Returns the book with the least value in books}

LIBRARY
min : book
i : integer

ALGORITHM
min ← books[0]
i traversal 2..books.size
   if (books[i].value < min.value) then
      min ← books[i]
→ min
```

```
function maxIdx(vals: array of integer) → integer
{Returns the index of the category which has the maximum sum of value.}
```

```
LIBRARY
i, max : integer

ALGORITHM
if (vals.length = 1) then
   → 0
else
   max ← 0
   i traversal 1..vals.length
      if (vals[i] > vals[max]) then
         max ← i
   → max
```

V. SUMMARY

From the description and the example in previous sections, we can summarize several points:

1. Greedy algorithm for the books arrangement problem always gives the optimal solution.
2. In solving the books arrangement problem with this algorithm, our focus is not on comparing selection functions. We assume that each book already has some value prior to the application of the algorithm. The selection function merely selects a candidate that has the greatest value. Thus, given some number of books and their values, we only seek to construct the solution set, that is, finding which books to be put in which bookshelf so we get the maximum sum of value of these books.
3. There is a polynomial time algorithm for the selection function (given as pseudo-code in section IV). This selection function also saves us the work of constructing the candidate set prior to the selection function call.
4. Although we have only discussed the application of the algorithm for arranging books based on their categories, we can extend the algorithm to arrange books based on anything; for example, based on their titles.

REFERENCES

[1] Munir, Rinaldi. *Diktat Kuliah IF3051 Strategi Algoritma*. Institut Teknologi Bandung, 2009, pages 26-31.

STATEMENT

Hereby I declare that this paper is my own writing, not an adaptation, a translation, nor a plagiarism.

Bandung, December 8th 2011

[illegible]

OKIRIZA WIBISONO
13509018