FINAL PROJECT REPORT

Analysis of Solving Methods for Practical Capacity Optimization Problem in Manufacturing

阮春科秀 – Tu – B10735011

1. Introduction

The globalization of the economy and market led to the appearance and domination of multinational companies. As an inevitability, the increased diversity of customers' requirements to the companies has caused an increase in the complexity of manufacturing processes. These convoluted operations require complex logistics and supply chains both in-plant and out-plant of the manufacturing.

The optimization of logistic systems is the key to the company's economical operations. Generally, most logistical problems are so complicated that a simple model cannot specify them. Hence, many studies had come up with multiple appropriate models and approaches to reduce the computational time, human resources, and save money.

2. Problem Definition

For my project, I have decided to replicate the knapsack solving method studies in optimizing capacity management in manufacturing and evaluate their practical performance.

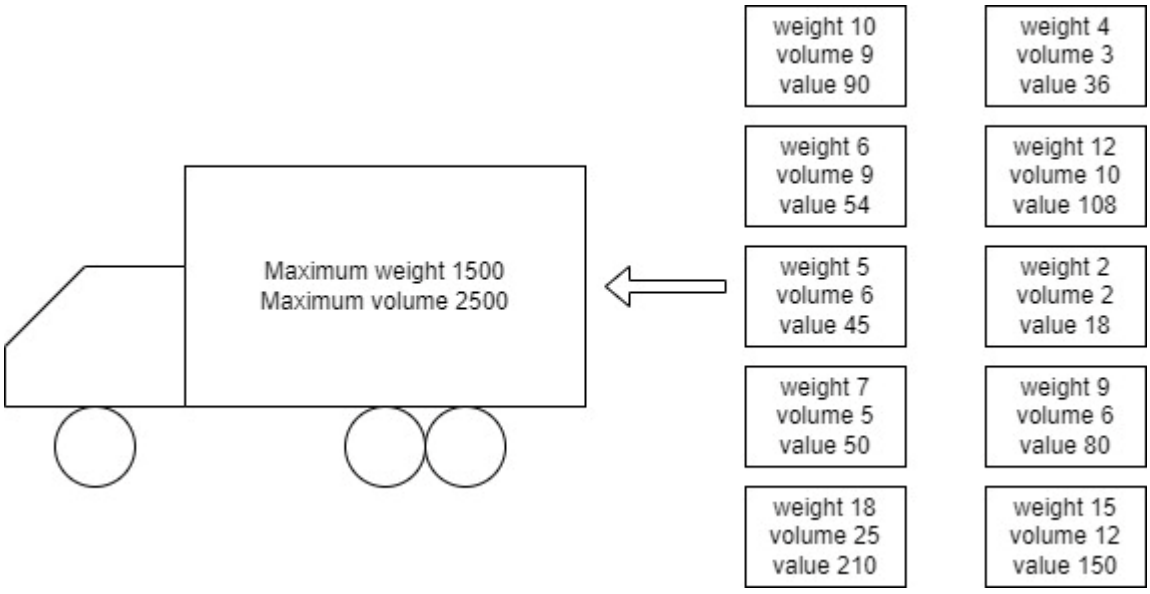
Capacity problems have an essential effect on every field of manufacturing, mostly in purchasing, distribution, and production. Many examples can be mentioned here which are loading and unloading products, storage, warehouse capacity, resources and materials handling, et cetera.

To deal with these problems, the company can consider improving the capacity or enhancing the rate of transportation methods. However, it will take them huge investments to upgrade buildings, facilities, vehicles, and machines. If they include the profit factor, there will be capacity optimization left as the most efficient method.

- **Example**

  A capacity optimization problem can be presented as a knapsack problem, which focuses on a solution to specify different kinds of products to maximize profit within a limited space. An example of this problem which I am going to demonstrate is represented below:

A manufacturing company needs to solve a logistic problem with its delivery process, in which they have to pack different crates of post-production components into a container and then distribute them to different assembly factories.
Each crate has its weight, volume and value, it is required to find a solution for optimizing the capacity while achieving the highest value.
The factory can provide at most 20 waiting-for-deliver similar crates at one delivery.



To solve this example, I will consider the integer-valued knapsack problem model:

$$f(x) = max \ \sum_{j=1}^{n} c_j \cdot x_j \quad (1)$$

$$\sum_{j=1}^{n} w_j \cdot x_j \ \leq W \quad (2)$$

$$\sum_{j=1}^{n} v_j \cdot x_j \ \leq V \quad (3)$$

$$0 \ \leq \ x_j \ \leq \ r_j \quad and \quad x_j \in Z \quad and \quad j = 1 \dots n \quad (4)$$

where c_j is the value, w_j is the weight, v_j is the volume, and W, V is the capacity limit (Weight, Volume). The x_j is the decision quantity variable that depends on different types of knapsack problems (0-1 knapsack problems, bounded knapsack problems, unbounded knapsack problems) and the r_j is bounded quantity. For instance, by the example above I mean the case where r_j=20 for all *j*.

The problem is represented as the form of the maximized value function (1), subject to the limited capacity (2) in the range of r_j integer quantity and the range of (1, n) objects (3).

I have found 2 methods to solve this problem: the first one is using the simplex method for Linear Programming and the second is applying Harmony Search. Below I will give a briefly explain and analysis of each method.

3. Solving by the simplex method for Linear Programming

Linear Programming is the name given to computing the best solution to a problem model as a set of linear relationships. The example above gives us two linear constraints with weight and volume, so we can definitely solve them using linear program methods.

The method I choose to implement is the simplex method, which remains the most dominant LP algorithm and is provided various implementing tools

- Pseudocode for the simplex algorithm:
    1. Set up the simplex tableau.
        - Initiate a 2-d array representing the tableau
    2. Find the positive index of the last row, and increase the corresponding variable from the basis of zero.
    3. Repeat step 2 until the last row is non-positive.
    4. Output the last column.

- My implementation:
    - I use Google's open-source linear programming solver (GLOP), created by Google's Operations Research Team as a tool to solve this problem.

- Result:

```
Number of variables =  10
{0: x[0], 1: x[1], 2: x[2], 3: x[3], 4: x[4], 5: x[5], 6: x[6], 7: x[7], 8: x[8], 9: x[9]}
Number of constraints =  2

Solution:
Objective value = 14753.333333333336
x[0]  =  20
x[1]  =  20
x[2]  =  20
x[3]  =  20
x[4]  =  20
x[5]  =  20
x[6]  =  0
x[7]  =  7
x[8]  =  20
x[9]  =  20
Maximum weight reached: 1500.0000000000002
Maximum volume reached: 1560.0000000000002

Problem solved in 0.000000 milliseconds
Problem solved in 0 iterations
--- 0.997305 milliseconds ---


Process finished with exit code 0
```

- Analysis:
  - For this problem, it gives us m inequality constraints and n number of items.
  - The Simplex algorithm operates mostly in the size of the simplex tableau with m rows of constraints and n columns of decisions variables, so the time complexity is O(m*n), and the space complexity is O(m*n)
  - Depends on the pivot rules so that the elapsed time can be reduced while executing.
  - With the GLOP, the primal-dual simplex algorithm is applied with the LU decomposition technique, which helps to reduce the elapsed time to almost 1ms.
  - The OR-Tools iteration count built-in function return 0 iteration, meanwhile it is too difficult to implement another custom iteration count function, so the real-time per iteration is undetermined.
  - The algorithm depends on the tableau. It means a more complicated problem with more constraints and decision variables will increase runtime.

4. Harmony Search

The Harmony Search algorithm is based on the performance process of jazz musicians. The group members of the jazz band try to find the best pitches to create a good harmony. Similar to it, the designers of engineering systems try to find the best parameters to create an optimized system.

In the optimization process of the Harmony Search algorithm, the musicians of the band can be replaced with each decision variable of the physical parameters of the system.

Thus, in this example, each item will be counted as a musician in the algorithm.

- The Harmony Search has several specific parameters, which require custom input from the user:
  - Harmony Memory (HM): stores objective function.
  - Termination criteria (*t*): is a pre-defined number of iterations for the algorithm.
  - Harmony Memory consideration rate (HMCR): is a random number generated in [0, 1], denotes the probability of using historical values stored in HM for playing a note.
  - Pitch adjustment rate (PAR): is a random number generated in [0, 1], and gives each value improvised from the HM a chance to be replaced by a value located in the vicinity of the selected value from HM.
  - Bandwidth (*bw*): If the pitch adjustment mechanism is selected, the value of *bw* controls the step size of movement. By selecting a large value of *bw*, the distance between the new value and the HM value increases.

- The algorithm includes 5 main steps:
  1. Initialize a harmonious memory.
     - Iterate i -> HMS
       Iterate j -> n (number of decision variables)
       Initialize x_{ij} in HM
  2. Improvised a new harmony.
     - Iterate j -> n
       if rand(0,1) < HCMR
       Let x_j be the *j*th dimension of a selected HM member.
       if rand(0,1) < PAR
       Apply pitch adjustment distance *bw* to mutate x_j
       x_j = x_j + rand(0,1) * *bw*
       else
       Let x_j be a random value in the range of decision variable
     - Evaluate the fitness of objective function f(x)
       Calculate the total_capacity of f(x)
       if total_capacity > max_capacity
       return 0
  3. The new harmony is included or excluded.
     - If f(x) is better than the fitness of the worst HM member
       Replace the worst HM member and its decision variable with f(x)
  4. Repeat steps 2 and 3 until the stopping criterion is met.
  5. Return the best harmony stored in harmonious memory.
     - If the problem requires non-decimal decision variables, we have to round the variables before printing them out.

- My implementation
  - I use a build-in library *pyHarmonySearch* as a tool for my implementation to solve this problem.
  - It was studied that HCMR = 80%, PAR = 15%, *bw* in the range of 0.15 – 0.35 is the optimized value, so I assigned these parameters to my program.

- Result:

```
Elapsed time: 0:00:15.378062 seconds
Best harmony (Selected items): [18, 19, 20, 19, 18, 17, 0, 12, 20, 20]
Best fitness (Maximum value): 14747.98706375701

Weight: 1496
Volume: 1543
Time per iteration: 0.515403 milliseconds

Process finished with exit code 0
```

- Analysis
  - Harmony Memory Size is based on the number of objective functions. For this example, because there is only one objective function, the HM is equal to 1.
  - Because the algorithm runs step (2) and step (3) totally *t* iterations, and each iteration takes n times of improvising harmony, the time complexity is O(t*n).
  - The algorithm has to create m arrays for m number of constraints, 2 arrays for the lower bound and upper bound of decision variables, and HMS*n size array for the Harmony Memory, so the space complexity should be: O(m*n + HMS*n + 2) = O((m + HMS)* n)
  - For the example above, I configured the program to run in 30000 iterations. Consequently, the elapsed time reached ~15s, and the real-time for each iteration is ~0.5ms.
  - The more iterations implemented, the more accuracy of the optimization value it can achieve.
  - For the problems which require a higher quantity and number of decision variables, it is recommend to increase the maximum iterations.

5. Summary

- There are a lot of methods to solve knapsack problems in practice. The simplex method and Harmony Search are only two ways of them.
- Throughout both practical and theoretical analysis, it can be said that the simplex method for knapsack solving is quite more efficient than Harmony Search.
- In terms of logical ideas, Harmony Search is more straightforward so that it is easier to implement the algorithms.

6. References

Péter Veres, Tamás Bányai, and Béla Illés (2017). ***Optimization problems of networking manufacturing processes****.* MATEC Web of Conferences 112, 06026.

Péter Veres, Tamás Bányai, and Béla Illés (2013). ***Optimization of Knapsack Problem with MATLAB, based on Harmony Search algorithm.*** *"Advanced Logistic Systems"*, Vol. 7, No.1, pp. 13 – 20.

Alireza Askarzadeh, Esmat Rashed (2017). ***Harmony Search algorithm: Basic Concepts and Engineering Application.*** *"Recent Developments in Intelligent Nature-Inspired Computing",* pp. 1 – 36*.*

X.Z. Gao, V. Govindasamy, H.Xu, X. Wang, and K. Zenger (2015). ***Harmony Search Method: Theory and Applications.*** *"Computational Intelligence and Neuroscience",* Vol. 2015, Article ID 258491.

Advanced LP Solving.
(URL: https://developers.google.com/optimization/lp/lp_advanced)

7. Appendix

For the source code, please refer to my Github repository

➔ https://github.com/khoaxuantu/Solving-Knapsacks-in-LinearProg-vs-HarmonySearch