Heuristics in Analytics

LOG 734 2021

Assignment 1
Construction Heuristic

Vladislav Klyuev
Mark Drozd
Sviatlana Lapitskaya

Logistics Analytics
Høgskolen i Molde

February 2021

Introduction

With this report we make an attempt to develop a construction heuristic described in *Heuristic description* part for the multidimensional knapsack problem (MKP). The definition of the problem is located in *Problem statement* part. Implementation language is C++.

Problem statement

Multidimensional knapsack problem can be formulated in the following way:

$$maximize \sum_{j=1}^{n} c_j x_j$$

$$s.t \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, ..., m$$

$$x_j \in 0, 1, \quad j = 1, ..., n$$

where the objective is to maximise the value of items we manage to fit into the knapsack. This knapsack is *multi-dimensional*, meaning it is limited by the numbers b_i in each of it's dimensions. The family of knapsack constraints represent the limitation of the knapsack volume. Thus, we strive to achieving the maximum value within a size-limited volume.

Heuristic description

A construction heuristic aims to find an initial primal-feasible solution by iteratively assigning values to the variables of a problem formulation. The approach produces feasible solutions, however it doesn't yet care about it's quality. The main goal here is to construct a solution from scratch, upon which improved solutions arise when using other approaches (s.a. genetic algorithm from [1]).
However, we are also interested in how good the acquired solution may be in terms of objective function. The motivation for this is that we may be interested in using the construction heuristic as a standalone one, that which produces results quite satisfiable on average and standing not very far from the optimum. The other thing is that the quality of *any* initial solution matters, since we aim to cut off unwanted branches in the BranchBound tree utilising the already known record (we are interested in a record that is as close to the dual bound as possible), or to spare time for the next improvement heuristic stages. Of course, due to the heuristic nature of the approach, we can not guarantee any quality level of an obtained solution, but we may build up our assumptions on it's usefulness based on the results we obtain and comparing them to other heuristics as well as others' implementation of them.
After a few experiments concerned mainly with the variable choice criteria (that which is to be assigned next) and therefore with how we compute weights, we implement the following approach:

Algorithm 1 Construction Heuristic

!

1: $V^{\#} := \{1, ..., n\}$
2: $g_i := 0, \forall i \in \{1, ..., m\}$
3: **while** $V^{\#} \neq$ **do**
4: Choose variable $j^* := argmax_{j \in V^{\#}} \{c_j \times (\sum_{i=1}^{m} \frac{a_{ij}}{(b_i - g_i)^2})^{-1}\}$
5: **if** $\exists i \in \{1, ..., m\} : g_i + a_{ij^*} \geq b_i$ **then**
6: $x_{j^*} := 0$
7: **else**
8: $x_{j^*} := 1$
9: $g_i := g_i + \sum_{i=1}^{m} a_{ij^*}$
10: **end if**
11: $V^{\#} := V^{\#} \setminus j^*$
12: **end while**
13: *Output* $f = \sum_{j=1}^{n} c_j \times x_j$

The next chosen variable is examined on whether the solution feasibility is kept if we set it to zero. If it is, we proceed with updating the row slacks and going to the next iteration. If it is not, the variable is set to zero: feasibility is saved and the process goes on to the next variable.
The entire point of the algorithm is how the variables are chosen. We assign weights equal to the corresponding objective coefficient multiplied by the sum of row slack squares divided by the matrix coefficients. Computing these weights, we strive to choose variables that are less likely to fill row slacks if we set them to 1 (dividing by smaller matrix coefficients gives out bigger weights) and are most promising to improve the objective. The process of making the final heuristic is absolutely creative and cannot be described with only logical decisions as occasional decisions could influence the results. However, we tried to structure the way to the results obtained. Initially a similar heuristic to the one given in lectures ($\frac{c_j}{a_j}$), changed into multidimensional($\frac{c_j}{\sum_{i=1}^{m} a_{ij}}$), was implemented. It had slightly worse results than ones in the previous years.
However, we decided to go further and make it even better, supplying these weights with more information, thus, the correction on the space left after insertion of a variable into the solution was made ($c_j \times (\sum_{i=1}^{m} \frac{a_{ij}}{b_i - g_i})^{-1}$), improving the results.
Nevertheless, after a while, we decided to give our heuristic even more with a similar to variance principle, squaring the difference, therefore enhancing the left space significance (large values become larger, and small values decrease). It can be vividly seen from the weight function, that such an approach with free space information has a value. It is logical that if we have much space left within the b-limit (look at the final version of the weight function), than the whole sum becomes smaller and as we divide by this sum, the final weight for a j-variable becomes larger, opposed to this sum being large and having almost no free space left. It was a small improvement, yet it has bettered the results and we decided to stop on developing anything further.

Implementation

Heuristic was implemented using C++ programming language based on the example presented in the tutorial. Instances were read from file and stored in vectors corresponding to the right hand side coefficients, the left hand side coefficients and the objective function coefficients. There are five functions implemented, not including main: for reading from file, finding index of the element with max weight, heuristic algorithm, assigning weights to all the variables and checking feasibility of the solution. For each instance time was measured using clock() function.

Results

The computations were done utilising the 2,2 GHz Quad-Core Intel Core i7 Processor with 16 GB 1600 MHz DDR3 Memory. The results are presented in the following table.

```
100-5-01.txt    0.096   milliseconds,    objective function value is   24049
100-5-02.txt    0.102   milliseconds,    objective function value is   23970
100-5-03.txt    0.102   milliseconds,    objective function value is   23317
100-5-04.txt    0.112   milliseconds,    objective function value is   22979
100-5-05.txt    0.107   milliseconds,    objective function value is   23826
250-10-01.txt    1.11   milliseconds,    objective function value is   58284
250-10-02.txt   0.989   milliseconds,    objective function value is   58293
250-10-03.txt   1.415   milliseconds,    objective function value is   57390
250-10-04.txt   1.092   milliseconds,    objective function value is   60467
250-10-05.txt   1.124   milliseconds,    objective function value is   57325
500-30-01.txt   14.482  milliseconds,    objective function value is  114839
500-30-02.txt   10.261  milliseconds,    objective function value is  113662
500-30-03.txt   11.388  milliseconds,    objective function value is  115805
500-30-04.txt   12.822  milliseconds,    objective function value is  114203
500-30-05.txt   10.029  milliseconds,    objective function value is  115471
```

At the end of the algorithm, we make an additional feasibility check to be sure that the solutions are correct.

References

[1] P. C. Chu and J. E. Beasley. "A Genetic Algorithm for the Multidimensional Knapsack Problem". In: *Journal of Heuristics* 4.1 (June 1998), pp. 63–86. ISSN: 1572-9397. DOI: 10.1023/A:1009642405419. URL: https://doi.org/10.1023/A:1009642405419.