A Branch-and-Bound Algorithm for the Ticket Restaurant Assignment Problem

Gabriele Cerizza

Università degli Studi di Milano
gabriele.cerizza@studenti.unimi.it
https://github.com/gabrielecerizza/orc_project

Introduction

In this report we detail a branch-and-bound exact method based on Lagrangean relaxation to solve the ticket restaurant assignment problem (TRAP), pursuant to the project specifications set out for the Operational Research Complements course of the Università degli Studi di Milano[1].

In Section **??** we illustrate the dataset used in the experiments. In Section **??** we briefly describe the algorithm and its implementation. In Section **??** we show the results of our experiments and provide comments on them. Finally, Section 4 contains our concluding remarks.

1 Ticket Restaurant Assignment Problem

In this section we define the TRAP (Section 1.1), we formalize it as a mathematical programming problem (Section 1.2) and we examine the relevant literature (Section 1.3).

1.1 Definition

The TRAP is defined as follows. A ticket company (TC) possesses two kinds of restaurant tickets: the low-profit tickets and the high-profit tickets. TC gives a certain amount of tickets to customer companies (CC). Each CC receives only one kind of tickets. Each CC has different sets of employees and each set uses the tickets to buy meals in a specific restaurant.

For each restaurant with which TC has a deal, a given ratio between low-profit and high-profit tickets must be observed. The TC must maximize the profit while complying with this constraint. Thus, maximizing the profit amounts to minimizing the number of low-profit tickets while ensuring that a given amount of low-profit tickets is assigned for each considered restaurant.

[1] https://homes.di.unimi.it/righini/Didattica/ComplementiRicercaOperativa/ComplementiRicercaOperativa.htm

1.2 Formalization

Let $I = \{1, \ldots, m\}$ be a set of restaurants, $J = \{1, \ldots, n\}$ be a set of customer companies, $b \in \mathbb{Z}_+^m$ be the vector representing the minimum amount of low-profit tickets to be assigned for each restaurant, and $A \in \mathbb{Z}_+^{m \times n}$ be the matrix representing the amount of low-profit tickets assigned to each given customer company for each given restaurant, so that a_{ij} is the amount of low-profit tickets assigned to customer company j for restaurant i. Then, the integer linear programming model of the ticket restaurant assignment problem is the following:

$$\begin{aligned} \min \quad & z = \sum_{j \in J} \left(\sum_{i \in I} a_{ij} \right) x_j && && (1) \\ \text{subject to} \quad & \sum_{j \in J} a_{ij} x_j \geq b_i && i = 1, \ldots, m\,, \\ & x_j \in \{0, 1\} && j = 1, \ldots, n\,, \end{aligned}$$

where each binary variable x_j indicates whether customer company j is assigned low-profit tickets. We define a cover a vector x such that $Ax \geq b$.

From (1), the following Lagrangean relaxation (LR) is obtained:

$$\begin{aligned} \min \quad & z_{\text{LR}} = \sum_{j \in J} \left(\sum_{i \in I} (1 - \lambda_i) a_{ij} \right) x_j + \sum_{i \in I} \lambda_i b_i && && (2) \\ \text{subject to} \quad & x_j \in \{0, 1\} && j = 1, \ldots, n\,, \\ & \lambda_i \geq 0 && i = 1, \ldots, m\,, \end{aligned}$$

where λ_i are the Lagrangean multipliers.

Finally, the dual of the linear programming (LP) relaxation of (1) is:

$$\begin{aligned} \max \quad & w = \sum_{i \in I} b_i y_i && && (3) \\ \text{subject to} \quad & \sum_{i \in I} a_{ij} y_i \leq c_j && j = 1, \ldots, n\,, \\ & y_i \geq 0 && i = 1, \ldots, m\,, \end{aligned}$$

where $c_j = \sum_{i \in I} a_{ij}$ for all $j \in J$. The dual of the LP relaxation may be exploited to find optimal Lagrangean multipliers (see Section 2.2).

The TRAP is a generalization of the set covering problem (SCP), as explained in the following section. The SCP is NP-hard [4], hence so is the TRAP.

1.3 Related Works

The problem at hand is equivalent to the SCP when A is a binary matrix and b is an all-ones vector. The SCP has been treated extensively in literature and many of the ideas developed within that context can be leveraged to solve the

TRAP. For the SCP, both exact and heuristic algorithms have been devised (see the survey in [4]). We focus only on exact algorithms for our problem.

The following works in the field of SCP are of particular interest to us. Balas and Ho [2] adopted a branch-and-cut procedure, comprising a primal heuristic to find upper bounds and subgradient optimization to find lower bounds. Beasley [3] adopted a branch-and-bound procedure, computing upper bounds with a greedy heuristic, computing lower bounds with dual ascent and subgradient optimization, and devising a number of problem reduction techniques. Balas and Carrera [1] adopted a dynamic subgradient-based branch-and-bound procedure to solve the SCP, additionally employing variable fixing techniques and heuristics to obtain upper bounds.

We fashioned our branch-and-bound procedure over the algorithms described in these works, adapting their solutions to account for the peculiarities of the TRAP. In particular, we note that most of the preprocessing techniques exploited for the SCP are unsuited for the TRAP.

A problem more closely related to the TRAP is the one called multicovering problem (MCP) [7, 8], defined as follows:

$$\begin{aligned} \min \quad & z = c^\top x \\ \text{subject to} \quad & Ax \geq b\,, \\ & x_j \in \{0,1\} \qquad \forall j = 1, \ldots, n\,, \end{aligned}$$

where A is a binary matrix and b is a vector of positive integers. This problem differs from the one at hand in that A is binary and not simply non-negative.

Notably, Hall and Hochbaum [8] adopted a branch-and-cut procedure to solve the MCP, using a primal heuristic to find upper bounds and combining a dual heuristic with subgradient optimization to find lower bounds. From this work we derived the main primal heuristic for our branch-and-bound algorithm (see Sections 2.1 and 3.3).

Furthermore, a generalization of the TRAP may be identified in the so-called covering integer problem (CIP) [9, 10], defined as follows:

$$\begin{aligned} \min \quad & z = c^\top x \\ \text{subject to} \quad & Ax \geq b\,, \\ & x_j \leq d_j \qquad \forall j = 1, \ldots, n\,, \\ & x_j \in \mathbb{Z}_+ \qquad \forall j = 1, \ldots, n\,, \end{aligned}$$

where all the entries in A, b, c and d are non-negative. This problem is equivalent to the TRAP when the multiplicity constraints $x_j \leq d_j$ force x to be binary. Unfortunately, we found only approximation algorithms for the CIP (see [9, 10]).

2 Branch-and-Bound

In this section we describe the different aspects of the branch-and-bound algorithm we experimented upon to solve the TRAP: the primal heuristics used to

find upper bounds (Section 2.1); the techniques used to find lower bounds (Section 2.2); the branching rules (Section 2.3); and the variable fixing strategies (Section 2.4).

2.1 Primal Heuristics

Greedy heuristic. This method selects greedily the variables of the cover, picking the row i with the largest ratio between the sum $\sum_{j \in J} a_{ij}$ and b_i and then picking the column with the largest coefficient. Each time a variable is selected, the coefficients on the left-hand side (LHS) and right-hand side (RHS) are decreased accordingly.

In this way we select the row which is easiest to cover, since a larger ratio means that more solutions might satisfy the constraint. Indeed, when the sum $\sum_{j \in J} a_{ij}$ is equal to b_i, we must select all variables with non-zero entries in A_i in order to satisfy the constraint, thus restricting the number of possible solutions. By picking the column with the largest coefficient we aim to satisfy the covering constraint in the fastest way.

Dobson heuristic. This heuristic is taken from Dobson [6] and generalizes an heuristic for the SCP from Chvatal [5]. This heuristic picks the column j that minimizes $c_j \sum_{i=1,\dots,m} a_{ij}$. Furthermore, any a_{ij} larger than b_i is lowered down to b_i.

The idea of this method is to pick the column that covers the most while costing the least. In the case of the TRAP, however, the cost of each column is equal to the sum of the coefficients of that column, which means that all columns could be equivalently selected. The only exception to this occurs when a column j is enough to cover the remaining part of b_i for any i. In this case, a_{ij} would be lowered and therefore $c_j \sum_{i=1,\dots,m} a_{ij}$ would also be lowered.

In short, this method, applied to the TRAP, prioritizes the columns whose coefficients are enough to cover the remaining part of b_i for any i.

Hall-Hochbaum heuristic. This method is adapted from one of the heuristics conceived by Hall and Hochbaum [8]. Specifically, this method picks the column j that maximizes $\frac{1}{c_j} \sum_{i \in L} \frac{b_i a_{ij}}{\text{space}(i)}$, where $L = \{i \in I : b_i > 0\}$ and $\text{space}(i) = \sum_{j \in J} a_{ij} - b_i$.

2.2 Lower Bounds

Lagrangean relaxation. A lower bound to the optimal value of problem (1) can be obtained by solving the LR defined in (2). When the Lagrangean multipliers λ are given, the objective function of (2) is trivially minimized by setting x_j to 1 when $\sum_{i \in I}(1 - \lambda_i)a_{ij} < 0$ and to 0 otherwise. The only problem left is to find the optimal Lagrangean multipliers, yielding the highest lower bound.

Algorithm 1: Subgradient optimization

Input: A, b, z_{LB}, z_{UB}; f, k, ϵ, ω
$t \leftarrow 1$
$\lambda_i^t = 0 \ \forall i \in I$
$\lambda_{\text{best}} \leftarrow \lambda^t$
$z_{\text{best}} \leftarrow z_{\text{LB}}$
while $z_{UB} > z_{LB}$ **do**
  for $j \in J$ **do**
    if $\sum_{i \in I}(1 - \lambda_i^t)a_{ij} < 0$ **then**
      $x_j \leftarrow 1$
    end
    else
      $x_j \leftarrow 0$
    end
  end
  $L(\lambda^t) \leftarrow \sum_{j \in J} \left(\sum_{i \in I}(1 - \lambda_i^t)a_{ij} \right) x_j + \sum_{i \in I} \lambda_i^t b_i$
  if $L(\lambda^t) > z_{LB}$ **then**
    $z_{\text{LB}} \leftarrow L(\lambda^t)$
    $z_{\text{best}} \leftarrow z_{\text{LB}}$
    $\lambda_{\text{best}} \leftarrow \lambda^t$
  end
  $g(\lambda^t) \leftarrow b - A^\top x$ `/* gradients */`
  if z_{LB} *unchanged for* k *iterations* **then**
    $f \leftarrow \frac{f}{2}$
  end
  $\sigma^t \leftarrow \frac{f(z_{\text{UB}} - z_{\text{LB}})}{\|g(\lambda^t)\|^2}$ `/* step length */`
  for $i \in I$ **do**
    $\lambda_i^{t+1} \leftarrow \max(0, \lambda_i^t + \sigma^t g_i^t(\lambda^t))$
  end
  if $\sigma^t < \epsilon \vee t > \omega$ **then**
    break
  end
  $t \leftarrow t + 1$
end
return z_{best}, λ_{best}

Subgradient optimization is a popular algorithm to find the optimal Lagrangean multipliers. The version of subgradient optimization we employed is mostly taken from [1] and is described in Algorithm 1.

When searching for the best Lagrangean multipliers in (2), we cannot assume that the Lagrangean multipliers can be confined within the range $[0, 1]$.

Lemma 1. *The optimal Lagrangean multipliers may not lie in the range* $[0, 1]$.

Proof. Consider a matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 4 \\ 2 & 2 & 2 \end{bmatrix}$ and a vector $b = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$. If we run Algorithm 1 with parameters $f = 2$, $k = 5$, $\epsilon = 0.005$, $\omega = 200$ and we constrain each Lagrangean multiplier within the range $[0, 1]$, we obtain $z_{\text{LB}} = 8$ and $\lambda = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. Running the algorithm with the same parameters, but without bounds on the Lagrangean multipliers, we obtain $z_{\text{LB}} = 10.498$ and $\lambda = \begin{bmatrix} 0.004 \\ 2.256 \\ 0 \end{bmatrix}$. □

Another option to find the optimal Lagrangean multipliers is to solve the dual of the LP relaxation of 1, which we defined in Section 1.2. In the context of the SCP, this method was adopted, for instance, in [2,3]; and it is mentioned as a viable approach in [4], due to the fact that the dual of the LP relaxation has the integrality property.

LP relaxation. Finally, one could consider solving the LP relaxation of (1) to obtain a lower bound. In fact, it is observed in [4] that "the lower bound determined by Lagrangian or alternative relaxations is much worse than the optimal solution value of the LP relaxation". The same Authors remark that exact algorithms may behave better with LP relaxation, whereas Lagrangean relaxation may be better suited for heuristic algorithms.

2.3 Branching Rules

Reduced costs branching. For this branching we compute the reduced costs r of the variables with the formula $r = (1 - \lambda) \cdot A$. Then, we complete the partial solution of the current node by setting to 0 all the variables not fixed to 1. After that, we select the variable x_j with the minimum reduced cost and a non-zero coefficient in the row with the largest violation, considering the completed solution. Finally, we generate two children nodes, fixing x_j equal to 0 in the first and equal to 1 in the second.

Cost branching for LP. This branching rule follows the same strategy as the reduced costs branching, except for the fact that we select the variable with the minimum cost instead of the variable with the minimum reduced cost. This modification allows to employ this rule with the LP relaxation, since in that case we do not have the Lagrangean multipliers (or dual variables) λ.

Beasley branching. This is the branching rule used in [3] and differs from the reduced costs branching rule only for the fact that we select the row whose corresponding Lagrangean multiplier has the largest value instead of the row with the largest violation.

2.4 Reduction

Lagrangean penalties. Following [3], we used the reduced costs to fix variables to 0 or to 1 to reduce the size of the problem instances.

Let $r = (1 - \lambda) \cdot A$ be the vector of reduced costs, z_{LB} be the lower bound of the current node and z_{UB} be the best incumbent upper bound. Then we can fix x_j to 0 when $r_j \geq 0$ and $z_{\text{LB}} + r_j > z_{\text{UB}}$, and we can fix x_j to 1 when $r_j < 0$ and $z_{\text{LB}} - r_j > z_{\text{UB}}$.

Column inclusion. This reduction method is adapted from [3] and consists in fixing to 1 all the unassigned variables with a non-zero coefficient in an uncovered row, when said row cannot be covered otherwise.

3 Computational Results

In this section we briefly describe the machine and the technologies used to run the experiments (Section 3.1), we illustrate how the problems were generated (Section 3.2), and then we show the results of our tests (Section 3.3).

3.1 Technologies and Hardware

We carried out the computational analysis of the branch-and-bound procedure on a machine with 16 gigabytes of RAM and a CPU Intel(R) Core(TM) i7-9700K 3.60GHz with 8 cores. The code was implemented in Python 3.11.2.

We included in our comparison the results obtained with Gurobi 10.0.2[2], in order to gauge the effectiveness of our solution in contrast to a commercial solver. It is worth noting that Gurobi, unlike our implementation, exploits multi-threading. Moreover, the Gurobi engine is written in C with API for Python and other programming languages. These factors should be taken into account when evaluating the results.

3.2 Problem Generation

The TRAP instances were randomly generated by taking in input the number of desired rows (m), columns (n) and the density of the matrix. The density is the percentage of non-zero coefficients in each row of the matrix A.

For each row, the value of b_i was picked uniformly at random between n and n^2. Then, the sum of the coefficients on the LHS was picked uniformly at

[2] https://www.gurobi.com/

random between $2b_i$ and $5b_i$. The number of columns with a non-zero coefficient was then determined using the density parameter and their indices were picked uniformly at random. Finally, the sum of the coefficients on the LHS was distributed uniformly at random among the selected column indices.

3.3 Comparative Analysis

Primal heuristics.

Subgradient optimization parameters. We took the params from Balas? F=2 and halve is something from Held and Karp I think. Finally, we settled for 200 as the maximum number of iterations, since we found that the optimal value of the Lagrangean multipliers was already obtained by that point in a set of problems of different sizes (see Figure).

3.4 Runtime and number of nodes.

Gurobi logs report using and the cuts

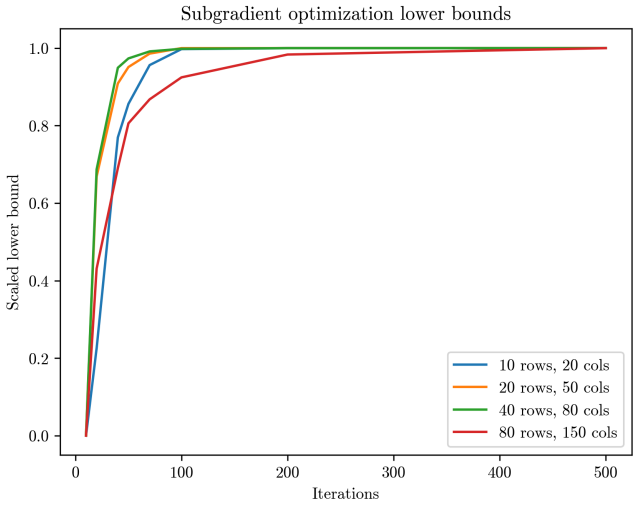

Fig. 1. Sample of images from the USPS dataset. The label is indicated below each image.

4 Conclusions

In this work we investigated the performance of the Pegasos algorithm described in Section **??** on a multiclass classification task over the USPS dataset described

in Section **??**. The algorithm was evaluated with 5-fold stratified cross-validation across various settings of hyperparameters. The best performance (0.026 test error with zero-one loss) was achieved by using a polynomial kernel of degree 3, 50000 iterations and a regularization coefficient of 1. A similar performance (0.027 test error) was achieved by using a Gaussian kernel with γ equal to 2, 25000 iterations and a regularization coefficient of 1e-5.

The algorithm proved to be an effective classification tool, although the task was not particularly challenging. Moreover, the algorithm showed a noteworthy rate of convergence, bordering a test error of 0.070 after only 1000 iterations when using the Gaussian kernel with γ equal to 2.

Further research might explore a wider set of hyperparameters, especially in the direction of lower regularization coefficient values and higher values of γ when using a Gaussian kernel.

References

1. Balas, E., and Carrera, M. C. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research 44*, 6 (1996), 875–890.
2. Balas, E., and Ho, A. *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1980, pp. 37–60.
3. Beasley, J. An algorithm for set covering problem. *European Journal of Operational Research 31*, 1 (1987), 85–93.
4. Caprara, A., Toth, P., and Fischetti, M. Algorithms for the Set Covering Problem. *Annals of Operations Research 98*, 1 (December 2000), 353–371.
5. Chvatal, V. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research 4*, 3 (1979), 233–235.
6. Dobson, G. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research 7*, 4 (1982), 515–531.
7. Hall, N. G., and Hochbaum, D. S. A fast approximation algorithm for the multicovering problem. *Discret. Appl. Math. 15*, 1 (1986), 35–40.
8. Hall, N. G., and Hochbaum, D. S. The multicovering problem. *European Journal of Operational Research 62*, 3 (1992), 323–339.
9. Kolliopoulos, S. Approximating covering integer programs with multiplicity constraints. *Discrete Applied Mathematics 129* (08 2003), 461–473.
10. Kolliopoulos, S. G., and Young, N. E. Approximation algorithms for covering/packing integer programs. *J. Comput. Syst. Sci. 71*, 4 (nov 2005), 495–505.