An exact method for shrinking pivot tables

Published Version:
An exact method for shrinking pivot tables / Boschetti M.A.; Golfarelli M.; Graziani S.. - In: OMEGA. - ISSN 0305-0483. - STAMPA. - 93:(2020), pp. 102044.1-102044.18. [10.1016/j.omega.2019.03.002]

Availability:
This version is available at: https://hdl.handle.net/11585/702179 since: 2020-03-04

Published:

DOI: http://doi.org/10.1016/j.omega.2019.03.002

An Exact Method for Shrinking Pivot Tables

Marco A. Boschetti[a,*], Matteo Golfarelli[b], Simone Graziani[b]

[a] *Department of Mathematics, University of Bologna, 47521 Cesena, Italy*
[b] *DISI, Department of Computer Science and Engineering, University of Bologna, 47521 Cesena, Italy*

Abstract

Pivot tables are one of the most popular tools for data visualization in both business and research applications. Although they are in general easy to use, their comprehensibility becomes progressively lower when the quantity of cells to be visualized increases (i.e., *information flooding problem*). Pivot tables are largely adopted in OLAP, the main approach to multidimensional data analysis. To cope with the information flooding problem in OLAP, the *shrink operation* enables users to balance the size of query results with their approximation, exploiting the presence of multidimensional hierarchies. The only implementation of the shrink operator proposed in the literature is based on a greedy heuristic that, in many cases, is far from reaching a desired level of effectiveness.

In this paper we propose a model for optimizing the implementation of the shrink operation which considers two possible problem types. The first type minimizes the loss of precision ensuring that the resulting data do not exceed the maximum allowed size. The second one minimizes the size of the resulting data ensuring that the loss of precision does not exceed a given maximum value. We model both problems as set partitioning problems with a side constraint. To solve the models we propose a dual ascent procedure based on a *Lagrangian pricing approach*, a Lagrangian heuristic, and an exact method. Experimental results show the effectiveness of the proposed approaches, that is compared with both the original greedy heuristic and a commercial general-purpose MIP solver.

Keywords: OLAP, Integer Linear Programming, Set Partitioning, Lagrangian Relaxation, Pricing

[*] Corresponding author, marco.boschetti@unibo.it

1. Introduction

Pivot tables are one of the most popular and powerful tools for data visualization in both business and research applications. Although they are in general easy to use, their comprehensibility becomes progressively lower when the quantity of cells to be visualized increases. Human operators have difficulties in understanding issues and effectively making decisions when they have too much information. This problem is known as *information flooding* and it can be solved by properly tuning the quantity of data to be visualized.

Pivot tables have been widely adopted in Business Intelligence (BI) systems, becoming the primary mode of viewing On-Line Analytical Processing (OLAP) data. In the context of BI data are mainly modeled using a multidimensional paradigm. Figure 1 shows a multidimensional cube, where events to be analyzed (e.g., census outcomes) are associated with multidimensional cube cells, while cube edges represent the analysis dimensions (e.g., RESIDENCE, TIME, OCCUPATION). For each cube cell, a value is given for each measure describing the event (e.g., citizen incomes, number of children). On top of each dimension, a hierarchy is built that defines groupings of its values. Figure 2 reports hierarchies associated with the dimensions of the cube shown in Figure 1. Multidimensional cubes are queried through OLAP queries, which typically ask for the values of one or more numerical measures (e.g., income of citizens) grouped by a given set of attributes in the hierarchies (e.g., City and Year), possibly with reference to a subset of dimensional values (e.g., State='FL'). The results of OLAP queries also take the form of multidimensional cubes and they are typically visualized through pivot tables, which usually consist of rows, columns, and data fields (see Figure 3).

As argued in more detail in [27], one of the critical issues affecting OLAP analyses, especially using pivot tables, is the achievement of a satisfactory compromise between the precision and the size of the data being visualized. In other words, the goal is to return the maximum quantity of information while avoiding information flooding. Queries that return results at a very fine-grained aggregation level (i.e., a cube with many cells) give more information, but they also require a greater effort from the user to analyze them. An excessive level of detail hinders the comprehension of the overall picture, which would be apparent when exploiting queries at coarse-grained

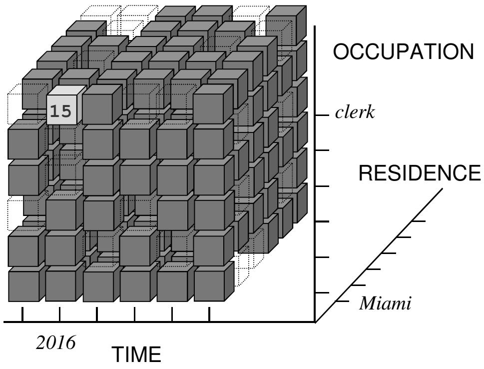

Figure 1: An example of a three dimensional cube.

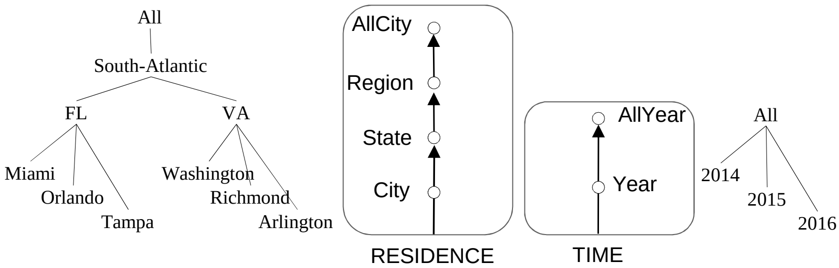

Figure 2: Two examples of hierarchies showing both their values and their aggregation structures (see [15])

aggregation levels, thus losing some precision.

In contrast with the general case, the presence of hierarchies in multidimensional cubes permits to deal with the problem of information flooding in pivot tables through an optimization process. Hierarchies define how dimensional values (e.g., *Miami*, *Arlington*) can be grouped to create semantically relevant clusters of elements (e.g., *Tampa* and *Orlando* can be grouped since they are both located in Florida). Compliance with the hierarchy structure when grouping dimensional values is not only recommended, but it is also mandatory for ensuring summarizzability [23]. This is a core property of OLAP applications that ensures the correctness and meaningfulness of values when progressive grouping is applied (e.g., income values for the cluster of citizens including *Miami* and *Arlington* can not be used to calculate the income of *FL* citizens). Based on this property/constraint and on the observation that approximation is a key to balance data precision with data size, in [15] the authors proposed a novel OLAP operation called *shrink*. Shrink

City	Year 2014	2015	2016
Miami	47	45	50
Orlando	44	43	52
Tampa	39	50	41
Washington	47	45	51
Richmond	43	46	49
Arlington	—	47	52

Shrink

City	Year 2014	2015	2016
Miami, Orlando	45.5	44	51
Tampa	39	50	41
VA	45	46	50.6

Figure 3: A pivot table resulting from an OLAP query (left), and its shrunk version when applied to the RESIDENCE dimension (right).

can be applied to the dimensions of a cube resulting from an OLAP query to decrease its size while controlling the loss in precision. The main idea is to fuse/cluster those dimensional values whose cells have similar values and replace them with a single representative satisfying the constraints imposed by hierarchies. As a consequence, the corresponding slices of cells will be fused and the measure values will be substituted by an approximated value, computed as their average.

We propose a simple example to help readers that are not familiar with OLAP queries. Let us suppose that an OLAP query has been issued against the CENSUS cube in Figure 1. The query asks for the average citizen incomes for each city in different years. Since the returned pivot table (Figure 3 left) is too large, due to the high number of cities, the user applies the shrink operator to the RESIDENCE dimension. This permits to fuse rows related to those cities that show similar values and comply with the structure of the hierarchy to be shrunk. The right side of Figure 3 shows a possible result. *Miami* and *Orlando* are clustered, since they show similar average incomes and belong to the same state. Similarly, all the cities in *Virginia* have been clustered and their names have been replaced by the state name to improve readability. Finally, *Tampa* remains a singleton since the income behavior differs too much from the other ones. Overall, the number of cells to be visualized drops from 18 to 9. As a side effect, we have a loss of precision.

The shrink operation can have two different but related goals:

- **Size-bound shrink**: minimize the loss in precision without exceeding the maximum size allowed for the resulting data.

- **Loss-bound shrink**: minimize the size of the resulting data without exceeding the maximum loss of precision.

The shrink operation is ruled by a parameter expressing either the maximum size allowed for the resulting data (i.e., the maximum number of returned cells) or the maximum loss of precision. Due to hierarchical constraints (better defined in Section 2) not all the slice fusions are feasible for shrinking, thus an additional constraint must be defined.

The shrink implementation proposed in [15] is based on a simple greedy algorithm, which is able to find a solution in a small amount of computing time. Unfortunately, as shown in Section 7, the greedy heuristic may generate solutions too weak for some target applications as the percentage gap from the optimal solution could be of some units. In this paper we propose:

- An original formulation of the problem as a set partitioning problem with side constraints.

- A new *matheuristic* algorithm (see [6, 26]) based on a dual ascent procedure that exploits pricing and Lagrangian relaxation. The dual ascent procedure provides a near optimal solution for the dual problem and the Lagrangian heuristic generates feasible solutions of very good quality. The percentage gap from the optimal solution value of the proposed approach is much better than that of the greedy heuristic and for many instances the best solution found is also optimal.

- An exact method which solves the problem using only a limited subset of variables generated by a pricing procedure based on the dual solution found by the dual ascent procedure. This exact method performs better than IBM ILOG CPLEX, a commercial general-purpose MIP solver, that also fails in solving some very large instances.

Our contributions create a bridge between BI and optimization techniques. This approach has become very common in recent years since BI approaches have become more sophisticated and often require the support of optimization techniques for an effective implementation, as witnessed by several papers on the subject proposed in the literature.

One of the classical application of optimization in BI is the design of learning algorithms, where classification, clustering, and regression problems must be solved (e.g., [21], [34]). An interesting introduction to operations research and data mining can be found in the special issue [31] and in the survey [32]. Some mathematical formulations and challenges are also discussed in [10] and [33]. Operational research inspired techniques have been also adopted during the design of BI solutions; for example the problem of selecting the most effective subset of materialized views in Data Warehouse is discussed in [25] and [36]. Operations research is also very useful for optimizing query execution (e.g., [22], [24]) or data visualization and discretization (e.g., [1], [18], [19]). This article focuses on the latter topic. Among the proposed solutions to this problem there are those that make use of On-Line Analytical Mining (OLAM) techniques. OLAM corresponds to an OLAP paradigm that is coupled with data mining techniques to create an approach where multidimensional data can be mined "on-the-fly" to extract concise patterns for user evaluation, but at the price of an increased computational complexity and an overhead for analyzing the generated patterns (see [16]).

A problem that shares some similarities with shrink operation (SH) is *microaggregation* (MA), which is a statistical disclosure control technique aimed at producing groups of microdata records with cardinality greater than a given parameter k, such that an intruder cannot identify individuals. For each variable in the microdata, the average value over the group is reported. The goal is to obtain groups that are as homogenous as possible. For this reason, an optimal MA minimizes within-group sum squared error. Several different formulations of the basic problem have been proposed:

- Fixed vs Variable group size: in the fixed version, all the groups must have the same size k.
- Univariate vs Multivariate aggregation: in the univariate version, grouping is based on a single variable.

While optimal MA for univariate data relies on polynomial algorithms [17], the optimal solution for multivariate one has been proven to be NP-hard [30]. The MA formulation that is closest to the SH one involves multivariate data and produces groups with variable size. The main difference between SH and MA is that, while the SH limits the overall number of groups or the overall error, MA constrains the cardinality of each single group. Since the cardinality of an optimal cluster is not constrained between k and $2k$, there exist a much higher number of feasible solutions. Differences in constraints make most of the findings reported in [14] not applicable to the SH problem. Consequently, while the heuristic proposed in [14] for multivariate MA relies on the same clustering principle as the one we proposed in [15], the differences in constraints make the specific optimizations proposed in [14] not applicable in [15].

To the best of our knowledge, no commercial OLAP tools implements techniques similar to our ones to address the visualization of pivot tables, therefore, the problem is open. The main approaches adopted so far are: (i) splitting the tables in several parts through selection predicates and visualizing each of them separately; (ii) representing the pivot table through smart visualization techniques [20] that exploit colors and shapes to increase the readability when many data are represented. Solution (i) directly represents pivot tables but lacks in providing an overall picture of the data. Solution (ii) typically provides an overall picture of the data but requires further analysis steps to obtain numeric details (e.g., zoom in, details-on-demand operators [3]). Conversely, the proposed matheuristic algorithm based on a Lagrangian relaxation does not make use of expensive commercial solvers and provides effective results, therefore it is a good candidate to be integrated in a commercial OLAP tool.

The paper outline is as follows. Section 2 introduces the shrink operator together with its greedy implementation [15]. In Section 3 we define the set partitioning formulation of the problem, whereas the dual ascent procedure and the Lagrangian heuristic are described in Sections 4 and 5, respectively. The exact method is presented in Section 6. In Section 7 we discuss the computational results and in Section 8 we draw the conclusions.

2. OLAP Shrink Operation

Given a multidimensional cube and chosen one of its dimensions, the shrink operation works by merging the dimensional values, together with the corresponding slices of cells. The aim is to obtain a compact representation of the input data while minimizing the approximation error (or size) and satisfying a given size (or error) threshold. The resulting representation must also be compliant with the constraints imposed by the structure of the involved hierarchy. Before describing the greedy implementation of the shrink operation proposed in [15], we need to briefly introduce the concept of *hierarchy compliance* and how we compute the approximation error (for an exhaustive and formal definition see [15]).

City	Year 2014	2015	2016
Miami, Orlando	45.5	44	51
Tampa	39	50	41
VA	45	46	50.6

(a)

	Year 2014	2015	2016
South-Atlantic	44	46	49.2

(b)

Figure 4: Two reductions of the same cube

Intuitively, given a cluster C composed by the values of a dimension on top of which a hierarchy h is built, we say that C is hierarchy-compliant (or h-compliant) if and only if the elements of C are values of h belonging to a same level and with the same parent. The complete enumeration of the h-compliant clusters associated to the RESIDENCE hierarchy of Figure 2 is listed in Table 1. An example of a non h-compliant cluster is instead $\{Miami, Washington\}$, because in order to have *Miami* and *Washington* in the same cluster it would be necessary to also merge together *Orlando*, *Tampa*, *Richmond*, and *Arlington*. When a cluster includes all and only the children of one or more elements of the parent level, it can be represented as the set of the corresponding parent values (i.e., $\{Miami, Washington, Orlando, Tampa, Richmond, Arlington\} \simeq \{FL, VA\}$).

Each hierarchical value at the finest level of detail (i.e., a dimensional value) has an associated slice of cells, e.g., with reference to Figure 1, the slice associated with the value *Miami* of the City attribute is composed by values $47, 45$, and 50. To compactly represent cells of several hierarchical values that have been merged together, the shrink operator uses their average. The approximation error introduced by representing a set of slices with an average slice is computed as the *Sum Squared Error* (SSE) between the average and the original values. Two different examples of reductions induced by the shrink operator are shown in Figure 4. Specifically, the SSE associated to the average slice $\{Miami, Orlando\}$ in Figure 4.a is $(1.5^2 + 1.5^2) + (1^2 + 1^2) + (1^2 + 1^2) = 8.5$. Notice that the SSE given by merging two or more values is never negative.

The greedy implementation of the shrink operation for both size- and error-constrained problems is based on agglomerative hierarchical clustering. Specifically, the algorithm works bottom-up by merging at each iteration the

Table 1: Clusters for the example reported in Figures 1 and 2

Level 0
$C_1 = \{South\text{-}Atlantic\} \simeq \{FL, VA\}$
Level 1
$C_2 = \{Miami, Orlando, Tampa\} \simeq \{FL\}$
$C_3 = \{Washington, Richmond, Arlington\} \simeq \{VA\}$
Level 2
$C_4 = \{Miami\}$
$C_5 = \{Orlando\}$
$C_6 = \{Tampa\}$
$C_7 = \{Miami, Orlando\}$
$C_8 = \{Orlando, Tampa\}$
$C_9 = \{Miami, Tampa\}$
$C_{10} = \{Washington\}$
$C_{11} = \{Richmond\}$
$C_{12} = \{Arlington\}$
$C_{13} = \{Washington, Richmond\}$
$C_{14} = \{Richmond, Arlington\}$
$C_{15} = \{Washington, Arlington\}$

two clusters of hierarchical values (and their slices) that lead to the minimum increase in SSE. Of course the two clusters can be merged only if the result is still h-compliant. This iterative process ends when the size constraint is satisfied or, conversely, when the result is such that no more values can be merged without violating the error threshold.

Consider again the cube in Figure 1. In the following we show in detail how the greedy shrink algorithm computes a reduction that solves the error-constrained problem with a maximum total SSE of 20 (Figure 5).

1. First, six singleton clusters are created, one for each member.

2. The most promising merge is the one between the *Arlington* and the *Washington* clusters, that yields SSE equal to 2.5 (Figure 5.a, right). The SSE of the resulting reduction (Figure 5.b, left) is 2.5, which meets the SSE constraint, so there is still room for shrinking.

3. The most promising merge is now the one between the *Miami* and the *Orlando* clusters (Figure 5.b, right). The total SSE is 11, so the iterative approach can be repeated.

4. At the next iteration, the algorithm merges *Richmond* cluster with the *Washington* – *Arlington* cluster (Figure 5.c, right). Since the resulting reduction has SSE higher than 20 (Figure 5.d), the algorithm stops. The reduction returned is the one shown in Figure 5.c, left.

(a)

City \ Year	*2010*	*2011*	*2012*	***SSE***
Miami	47	45	50	*0*
Orlando	44	43	52	*0*
Tampa	39	50	41	*0*
Washington	47	45	51	*0*
Richmond	43	46	49	*0*
Arlington	—	47	52	*0*

ΔSSE	*Miami*	*Orlando*	*Tampa*	*Washington*	*Richmond*	*Arlington*
Miami						
Orlando	8.5					
Tampa	85	97.5				
Washington						
Richmond				10.5		
Arlington				**2.5**	5	

(b)

City \ Year	*2010*	*2011*	*2012*	***SSE***
Miami	47	45	50	*0*
Orlando	44	43	52	*0*
Tampa	39	50	41	*0*
Washington, Arlington	47	46	51.5	*2.5*
Richmond	43	46	49	*0*

ΔSSE	*Miami*	*Orlando*	*Tampa*	*Wash., Arlin.*	*Richmond*
Miami					
Orlando	**8.5**				
Tampa	85	97.5			
Wash., Arlin.					
Richmond				14.7	

(c)

City \ Year	*2010*	*2011*	*2012*	***SSE***
Miami, Orlando	45.5	44	51	*8.5*
Tampa	39	50	41	*0*
Washington, Arlington	47	46	51.5	*2.5*
Richmond	43	46	49	*0*

ΔSSE	*Miami, Orlando*	*Tampa*	*Wash., Arlin.*	*Richmond*
Miami, Orlando				
Tampa	127.3			
Wash., Arlin.				
Richmond			**14.7**	

(d)

City \ Year	*2010*	*2011*	*2012*	***SSE***
Miami, Orlando	45.5	44	51	*8.5*
Tampa	39	50	41	*0*
VA	45	46	50.6	*14.7*

Figure 5: Applying the greedy algorithm for shrinking. The left column shows the pivot tables, the right column reports the SSE increase for each feasible merge. Grey cells correspond to non h-compliant merges.

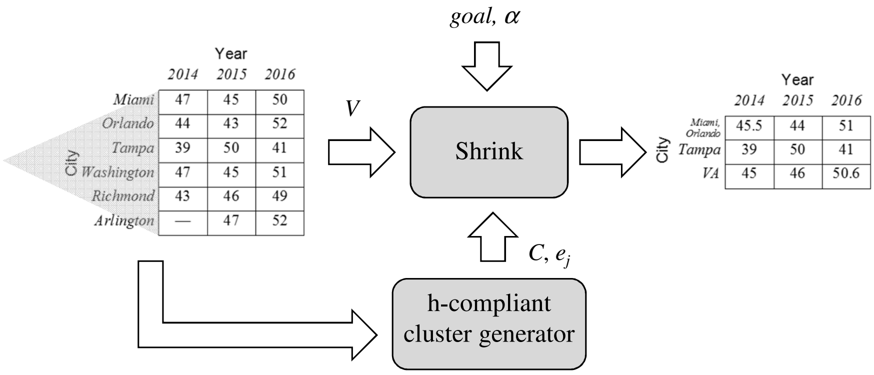

Figure 6: The shrink optimization process.

3. Mathematical Formulation

In order to achieve a better understanding of the model for optimizing the shrink operator, in Figure 6 we provide a graphical representation of the optimization process. The algorithms proposed in the next sections are implemented by the main computational module denoted with *Shrink*. The input for such a module are:

- The index set $V = \{1, \ldots, n\}$ of the n dimensional values of the hierarchy involved in the shrink operation.
- The index set $\mathbb{C}$ of all the feasible (i.e., h-compliant) clusters together with the associated loss of precision, which are computed as described in Section 2. For each cluster $j \in \mathbb{C}$ the loss of precision is denoted by e_j.
- The parameter α denoting the maximum size or the maximum loss allowed, depending on whether you are solving the size-bound ($goal = S$) or loss-bound ($goal = L$) version of the problem, respectively.

The *h-compliant cluster generator* module is in charge of generating in advance the whole set of h-compliant clusters induced by the involved hierarchy. As we will show in Section 7, this task can be accomplished in a negligible time when compared with the one required by the *Shrink* module.

We denote with $\mathbb{C}_i \subseteq \mathbb{C}$ the subset of clusters involving the value i, for each $i \in V$. C_j represents the index set of the values contained in the cluster $j \in \mathbb{C}$. Let x_j be a $0-1$ binary variable equal to one if and only if the cluster

$j \in \mathbb{C}$ is in the optimal solution. The problem can be formulated as a set partitioning problem with a side constraint as follows:

$$
\begin{aligned}
(P) \qquad z_P = \min \sum_{j \in \mathbb{C}} c_j x_j && (1) \\
s.t. \sum_{j \in \mathbb{C}_i} x_j = 1, \quad i \in V && (2) \\
\sum_{j \in \mathbb{C}} a_j x_j \leq \alpha && (3) \\
x_j \in \{0, 1\}, \quad j \in \mathbb{C}. && (4)
\end{aligned}
$$

If $goal = S$, setting $c_j = e_j$ the objective function (1) minimizes the loss of precision; conversely, if $goal = L$, setting $c_j = 1$ the objective function (1) minimizes the size of the resulting data. Constraints (2) ensure that each original dimensional value is included in a cluster. Constraint (3) guarantees that the resulting data do not exceed the maximum size allowed by setting $a_j = 1$ and $\alpha = MaxSize$, if $goal = S$, or the maximum loss of precision by setting $a_j = e_j$ and $\alpha = MaxLoss$, if $goal = L$.

Let u_i and v be the dual variables associated to constraints (2) and (3), respectively. The dual of the LP-relaxation of problem P is the following:

$$
\begin{aligned}
(D) \qquad z_D = \min \sum_{i \in V} u_i + \alpha v && (5) \\
s.t. \sum_{i \in C_j} u_i + a_j v \leq c_j, \quad j \in \mathbb{C} && (6) \\
u_i \text{ unconstrained}, \quad i \in V && (7) \\
v \leq 0. && (8)
\end{aligned}
$$

The dual D is used for defining the dual ascent procedure, described in Section 4, which is based on a Lagrangian relaxation of the problem P. The dual ascent procedure iteratively improves the dual solution which is used for defining a *core* subset of clusters by means of a pricing procedure. The dual ascent ends providing a near optimal dual solution for the problem D.

The dual solution is also used to define a core subproblem for the exact method proposed in Section 6. The exact method solves the problem P using only a limited subset of variables generated by a pricing procedure based on the dual solution found by the dual ascent procedure.

4. A Dual Ascent

The dual ascent procedure is based on a *parametric relaxation* of problem P and its Lagrangian relaxation. The resulting problem is solved by a subgradient algorithm that uses only a subset of variables defined by a pricing procedure and embeds an effective Lagrangian heuristic.

4.1. Parametric Relaxation

Parametric relaxation is a well-known approach in the literature. Some interesting applications are described by Christofides et al. [12] for vehicle routing and by Mingozzi et al. [28] and Boschetti et al. [7] for crew scheduling. Recently, dual ascent procedures based on a parametric relaxation have been proposed by Boschetti et al. [8] for the set partitioning problem and by Boschetti and Maniezzo [5] for the set covering problem with side constraints. The proposed dual ascent generalizes the approach of Boschetti et al. [8], which does not consider side constraints, and it uses an approach similar to the one used by Boschetti and Maniezzo [5] for the set covering problem. It also generalizes the dual ascent approach proposed by Christofides et al. [12], Mingozzi et al. [28], Boschetti et al. [7]. In this section we describe the parametric relaxation of problem P used by the proposed dual ascent.

We associate with each dimensional values $i \in V$ a positive real weight q_i. Let $q(C_j) = \sum_{i \in C_j} q_i$ be the total weight of column (cluster) $j \in \mathbb{C}$. Since weights $\{q_i\}$ are positive, $q(C_j) > 0$ for every column $j \in \mathbb{C}$. We replace each variable x_j by a new set of $|C_j|$ variables y_j^i, $i \in C_j$, as follows:

$$x_j = \sum_{i \in C_j} \frac{q_i}{q(C_j)} y_j^i, \quad j \in \mathbb{C} \tag{9}$$

and the resulting mathematical formulation of the parametric relaxation of problem P is the following:

$$(PR(\boldsymbol{q})) \qquad z_{PR}(\boldsymbol{q}) = \min \sum_{j \in \mathbb{C}} \sum_{i \in C_j} c_j \frac{q_i}{q(C_j)} y_j^i \tag{10}$$

$$s.t. \sum_{j \in \mathbb{C}_i} \sum_{h \in C_j} \frac{q_h}{q(C_j)} y_j^h = 1, \qquad i \in V \tag{11}$$

$$\sum_{j \in \mathbb{C}} a_j \sum_{h \in C_j} \frac{q_h}{q(C_j)} y_j^h \leq \alpha, \tag{12}$$

$$y_j^i \in \{0,1\}, \qquad j \in \mathbb{C}, i \in C_j. \quad (13)$$

Constraints (11) and (12) correspond to constraints (2) and (3) of problem P, respectively. Notice that if $y_j^i = 1$ no constraint imposes that $y_j^h = 1$ for every value $h \in C_j$ covered by column j, therefore $PR(\boldsymbol{q})$ is a relaxation of problem P, because in this case the corresponding variable x_j of P is fractional (see equation (9)).

4.2. Lagrangian Relaxation

Problem $PR(\boldsymbol{q})$ can be relaxed by dualizing constraints (11) and (12) in a Lagrangian fashion, by means of the penalty vector $\boldsymbol{\lambda} \in \mathbb{R}^{n+1}$ having the first n components λ_i, $i \in V$, unconstrained and $\lambda_{n+1} \leq 0$.

The resulting Lagrangian problem is:

$$(LR(\boldsymbol{\lambda}, \boldsymbol{q}))\ z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q}) = \min \sum_{j \in \mathbb{C}} \sum_{i \in C_j} \left(c_j - \lambda'(C_j)\right) \frac{q_i}{q(C_j)} y_j^i + \sum_{i \in V} \lambda_i + \alpha \lambda_{n+1} \quad (14)$$

$$s.t.\ y_j^i \in \{0,1\}, \qquad i \in V, j \in \mathbb{C} \quad (15)$$

where $\lambda'(C_j) = \lambda(C_j) + a_j \lambda_{n+1}$ and $\lambda(C_j) = \sum_{h \in C_j} \lambda_h$. The optimal value of problem $LR(\boldsymbol{\lambda}, \boldsymbol{q})$ is a valid lower bound for the original problem P and it can be strengthened adding the constraint $\sum_{j \in \mathbb{C}_i} y_j^i = 1$ for every $i \in V$.

Problem $LR(\boldsymbol{\lambda}, \boldsymbol{q})$ is decomposable into $|V|$ subproblems, one for each row $i \in V$:

$$(LR^i(\boldsymbol{\lambda}, \boldsymbol{q})) \qquad z_{LR}^i(\boldsymbol{\lambda}, \boldsymbol{q}) = \min \sum_{j \in \mathbb{C}_i} c_j^i(\boldsymbol{\lambda}, \boldsymbol{q}) y_j^i + \lambda_i \quad (16)$$

$$s.t. \sum_{j \in \mathbb{C}_i} y_j^i = 1 \quad (17)$$

$$y_j^i \in \{0,1\}, \qquad j \in \mathbb{C}_i \quad (18)$$

where the cost of each variable y_j^i is $c_j^i(\boldsymbol{\lambda}, \boldsymbol{q}) = c'_j \frac{q_i}{q(C_j)}$ and $c'_j = c_j - \lambda(C_j) - a_j \lambda_{n+1}$. Hence, the overall value of the Lagrangian problem is $z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q}) = \sum_{i \in V} z_{LR}^i(\boldsymbol{\lambda}, \boldsymbol{q}) + \alpha \lambda_{n+1}$.

Theorem 1 shows that any optimal solution of problem $LR(\boldsymbol{\lambda}, \boldsymbol{q})$ provides a feasible solution $(\boldsymbol{u}, v)$ of cost $z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q})$ for the dual problem D.

Theorem 1. *Let $\boldsymbol{\lambda}$ be a vector of $n+1$ real numbers, where λ_i, $i \in V$, are unconstrained and $\lambda_{n+1} \leq 0$. Let $\boldsymbol{q}$ be a vector of n positive real numbers, i.e., $q_i > 0$, for every $i \in V$. A feasible dual solution $(\boldsymbol{u}, v)$ of cost $z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q})$ for dual problem D can be obtained by means of the following expressions:*

$$\begin{aligned} u_i = &\quad q_i \min_{j \in \mathbb{C}_i} \left\{ \tfrac{c'_j}{Q(C_j)} \right\} + \lambda_i, \quad i \in V \\ v = &\quad \lambda_{n+1}, \end{aligned} \tag{19}$$

where $c'_j = c_j - \lambda(C_j) - a_j \lambda_{n+1}$, $\lambda(C_j) = \sum_{i \in C_j} \lambda_i$, and $Q(C_j) = \sum_{i \in C_j} q_i$.

Proof. Let us consider the dual constraint (6) corresponding to column $j \in \mathbb{C}$ of the LP-relaxation of P. For every column j, the following inequalities hold:

$$\min_{h \in \mathbb{C}_i} \left\{ \frac{c'_h}{Q(C_h)} \right\} \leq \frac{c'_j}{Q(C_j)}, \qquad \text{for every } i \in C_j. \tag{20}$$

From expression (19) we obtain

$$u_i \leq q_i \frac{c'_j}{Q(C_j)} + \lambda_i, \quad i \in C_j, j \in \mathbb{C} \tag{21}$$

and by adding inequalities (21) we derive

$$\sum_{i \in C_j} u_i \leq \sum_{i \in C_j} \left(q_i \frac{c'_j}{Q(C_j)} + \lambda_i \right), \quad j \in \mathbb{C}. \tag{22}$$

Therefore, considering the dual constraint (6) for every $j \in \mathbb{C}$, we have

$$\begin{aligned} \sum_{i \in C_j} u_i + a_j v &\leq \frac{c'_j}{Q(C_j)} \sum_{i \in C_j} q_i + \sum_{i \in C_j} \lambda_i + a_j v \\ &\leq \frac{c'_j}{Q(C_j)} Q(C_j) + \lambda(C_j) + a_j v \\ &\leq c'_j + \lambda(C_j) + a_j v \\ &\leq c_j - \lambda(C_j) - a_j v + \lambda(C_j) + a_j v \\ &\leq c_j. \end{aligned} \tag{23}$$

It is straightforward to show that the dual solution $(\boldsymbol{u}, v)$ is of cost $z_D(\boldsymbol{u}, v) = \sum_{i \in V} u_i + \alpha v = z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q})$. □

The dual solution obtained according to Theorem 1 can be further improved by applying the greedy procedure described in Balas and Carrera [2] or Caprara et al. [11].

Corollary 1 shows that the best lower bound that can be achieved using expression (19) is equal to the optimal solution cost z_D of the dual problem D and that this value can be obtained searching the maximum of the function $z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q})$ with respect to $\boldsymbol{\lambda}$.

Corollary 1. *For every $\boldsymbol{q} > \mathbf{0}$, $\boldsymbol{q} \in \mathbb{R}^n$, the following equality holds:*

$$\max\{z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q}) : \boldsymbol{\lambda} \in \mathbb{R}^{n+1}, \lambda_{n+1} \leq 0\} = z_D. \tag{24}$$

Proof. Let $(\boldsymbol{u}^*, v^*)$ be an optimal solution of problem D of cost z_D. For every $j \in \mathbb{C}$, we have

$$c_j - \sum_{h \in C_j} u_h^* - a_j v^* \geq 0 \tag{25}$$

and for every $i \in V$, there exists at least a column $j' \in \mathbb{C}_i$ such that

$$c_{j'} - \sum_{h \in C_{j'}} u_h^* - a_{j'} v^* = 0. \tag{26}$$

If for a given $i \in V$ a column j' satisfying equality (26) does not exist, we can improve the "*optimal dual solution*" by increasing the corresponding dual variable u_i, in contradiction with the hypothesis.

By setting $\boldsymbol{\lambda} = (\boldsymbol{u}^*, v^*)$, when we evaluate the dual solution by expression (19) we have $u_i = q_i \min_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\} + u_i = 0 + u_i$, for every $i \in V$, and $v = v^*$. Therefore, $z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q}) = \sum_{i \in V} z_{LR}^i(\boldsymbol{\lambda}, \boldsymbol{q}) + \alpha\lambda_{n+1} = \sum_{i \in V} u_i + \alpha v = z_D$. □

In order to find the optimal (or near optimal) dual solution of cost z_D we need to solve the Lagrangian Dual $\max\{z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q}) : \boldsymbol{\lambda} \in \mathbb{R}^{n+1}, \lambda_{n+1} \leq 0\}$.

We propose a dual ascent procedure based on a subgradient algorithm that only considers a subset of problem variables. These variables are defined by a pricing procedure following the approach proposed by Boschetti et al. [8] for the set partitioning problem (without side constraint). We also use a simple variant where the subgradient θ^k at iteration k is *smoothed* by the *direction* defined by the subgradient θ^{k-1} at previous iteration $k-1$ (see Boyd and Mutapcic [9] and Crainic et al. [13]). This variant slightly improves the convergence of the subgradient algorithm and generates a better sequence of dual variables for the Lagrangian heuristic, which helps improving the quality of its solutions. A possible interesting future research direction could be the use of a bundle method instead of the subgradient.

<u>Dual Ascent Procedure</u>

Step 1. **Initial setup**

Set $z_{LB} = -\infty$, $\beta = \beta_0$, the initial penalty vector $\boldsymbol{\lambda} = \mathbf{0}$, $\rho = 0.5$, and $\boldsymbol{s} = \mathbf{0}$.
Generate an initial *core* subset of columns $\mathbb{C}' \subseteq \mathbb{C}$.

Step 2. **Solve Lagrangian Problem**

Solve $LR(\boldsymbol{\lambda}, \boldsymbol{q})$ using only the columns in the core $\mathbb{C}'$.
Compute $(\boldsymbol{u}, v)$ according to Theorem 1 and improve it using the greedy algorithm described in Caprara et al. [11].

Step 3. **Pricing**

Generate a subset $Q \subseteq \mathbb{C}$ of columns having negative reduced costs with respect to $(\boldsymbol{u}, v)$, i.e., $Q = \{j \in \mathbb{C} : c_j - \sum_{i \in C_j} u_i - a_j v < 0\}$.
Add subset Q to the core $\mathbb{C}'$, i.e., $\mathbb{C}' = \mathbb{C}' \cup Q$.
If $Q = \emptyset$, then $(\boldsymbol{u}, v)$ is a feasible dual solution for problem D, therefore $z_{LB} = \max\{z_{LB}, LR(\boldsymbol{\lambda}, \boldsymbol{q})\}$ and all columns of reduced cost larger than $\varepsilon_0 z_{LB}$ are removed from $\mathbb{C}'$.

Step 4. **Update Lagrangian penalties**

Compute subgradient components:

- $\theta_i = 1 - \sum_{j \in \mathbb{C}_i} \sum_{h \in C_j} \frac{q_h}{q(C_j)} y_j^h$, for every $i \in V$
- $\theta_{n+1} = \alpha - \sum_{j \in \mathbb{C}} \sum_{h \in C_j} a_j \frac{q_h}{q(C_j)} y_j^h$

Compute the step size $\sigma = \beta \frac{0.01 \times z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q})}{\sum_{i=1}^{n+1} \theta_i^2}$ and update vector $\boldsymbol{\lambda}$:

- $\lambda_i = \lambda_i + \rho(\sigma\theta_i) + (1-\rho)s_i$, for every $i \in V$
- $\lambda_{n+1} = \min\{0, \lambda_{n+1} + \rho(\sigma\theta_{n+1}) + (1-\rho)s_{n+1}\}$

Save $\boldsymbol{s} = \sigma\boldsymbol{\theta}$.

Step 5. **Stop Conditions**

If the maximum number of iterations *MaxIter* is not reached and the lower bound has improved enough (i.e., the improvement is larger than $\varepsilon_1 z_{LB}$) during last $MaxIter_0$ iterations, go to Step 2.

In this paper we generate the full set $\mathbb{C}$ in advance, before starting the Dual Ascent Procedure, because the full generation is not time consuming, but the dual ascent procedure works with a small subset of columns, called *core*, adding new columns only when required. Working with a core of small size allows a large computing time saving. The initial core is generated by considering in turns the columns in $\mathbb{C}$ sorted by non-decreasing order of the values $c_j/|C_j|$. If the column covers a row already covered by another column in the core or violates the side constraint, then it is ignored, otherwise it is added to the core.

Notice that $z_{LR}(\boldsymbol{\lambda}, \boldsymbol{q})$ is a valid lower bound for problem P if and only if no columns of negative reduced costs exist (i.e., $Q = \emptyset$), with respect to the corresponding dual solution $(\boldsymbol{u}, v)$, which is feasible in this case. When the dual solution $(\boldsymbol{u}, v)$ is feasible, we remove from $\mathbb{C}'$ all columns of reduced cost larger than $\varepsilon_0 z_{LB}$ to maintain the core as small as possible. Instead, the parameter ε_1 is used in the *stop conditions* to check if the lower bound has been improved enough during the last $MaxIter_0$ iterations.

In order to improve the convergence to a near optimal dual solution, we update the step-size parameter β during the execution. If, after a given number of iterations $MaxIter_1$, the lower bound is not improved, we decrease β, i.e, $\beta = \gamma_1 \beta$, where $\gamma_1 < 1$. As soon as the lower bound is improved we increase β, i.e, $\beta = \gamma_2 \beta$, where $\gamma_2 > 1$.

The complete definition of the parameter values can be found at Section 7, where the computational results are described.

5. A Lagrangian Heuristic

The dual ascent procedure provides an effective lower bound for problem P. While following a "*matheuristic*" approach (see [4, 6, 26]) to obtain an upper bound of good quality, we develop a Lagrangian heuristic algorithm.

The proposed Lagrangian heuristic is based on a simple greedy algorithm that makes use of the solution of the Lagrangian problem $LR(\boldsymbol{\lambda}, \boldsymbol{q})$ and of the corresponding *penalized costs*. It is applied at each iteration of the dual ascent procedure when the current lower bound is *good enough*.

At the beginning, the procedure builds an initial partial solution using the columns (i.e., configurations) $\mathbb{C}'' = \left\{ j' = \text{argmin}_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\} : i \in V \right\}$. To build the initial solution, we start with an empty solution (i.e., $x'_j = 0$, for every $j \in \mathbb{C}$) and we consider each of the columns in $\mathbb{C}''$ in turns. Given a column $j \in \mathbb{C}''$, if we have $x'_j = 0$ for every $i \in C_j$, we can add the column to the current emerging solution (i.e, $x'_j = 1$). Since the order in which the columns of $\mathbb{C}''$ are considered is very important, we have considered four different sortings. Notice that $\mathbb{C}''$ is generated by selecting one column for each row $i \in V$, therefore we order its columns by sorting the rows in one of the following ways:

- for increasing $val(i) = i$ (i.e., the index $i \in V$);
- for non-decreasing $val(i) = \min_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\}$;
- for non-increasing $val(i) = \frac{q_i}{Q(C_{j'})} y^i$, where $j' = \text{argmin}_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\}$ and $y^i = \left| \left\{ i' \in V : j' = \text{argmin}_{j \in \mathbb{C}_{i'}} \left\{ \frac{c'_j}{Q(C_j)} \right\} \right\} \right|$;
- for non-increasing $val(i) = \lambda_i$.

The procedure tries to complete the emerging solution considering the remaining columns sorted in non-decreasing order of their *normalized* cost $\frac{c_j}{|C_j|}$. We use this sorting because the number of columns can be huge and we can save time computing it at the beginning of the dual ascent. We perform two iterations: the first one only considering the columns of the core $\mathbb{C}'$; the second one considering all columns $\mathbb{C}$.

Lagrangian Heuristic

Step 1. **Initial setup**

Let z_{UB}^{best} be the best upper bound found so far.
Set $z_{UB} = 0$, $x'_j = 0$, for every $j \in \mathbb{C}$, and $iter = 1$.

Step 2. **Phase 1: Build a partial solution from the *LR* solution**

For each $i \in V$, following one of the four sorting criteria, try to add to the emerging solution column $j' = \text{argmin}_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\}$.
If $\sum_{i' \in C_{j'}} \sum_{j \in \mathbb{C}_{i'}} x'_j = 0$, $\sum_{j \in \mathbb{C}} a_j x'_j \leq \alpha - a_{j'}$, and $z_{UB} + c_{j'} < z_{UB}^{best}$,

column j' is added to the emerging solution, i.e., $x'_{j'} = 1$ and $z_{UB} = z_{UB} + c_{j'}$.

Step 3. **Check if the emerging solution is complete**
If $\sum_{j \in \mathbb{C}_i} x'_j = 1$ for every $i \in V$, the solution is feasible, therefore update the current best solution $z_{UB}^{best} = z_{UB}$, $\boldsymbol{x}^{best} = \boldsymbol{x}'$, and STOP.

Step 4. **Phase 2: Complete the emerging solution**
If there exists at least a row $i \in V$ such that $\sum_{j \in \mathbb{C}_i} x'_j = 0$, we try to *complete* the emerging solution by considering the remaining columns sorted in non-decreasing order of their *normalized* cost $\frac{c_j}{|C_j|}$. We perform two iterations: the first one only considering the columns of the core $\mathbb{C}'$; and a second one considering all columns $\mathbb{C}$. If $\sum_{i' \in C_{j'}} \sum_{j \in \mathbb{C}_{i'}} x'_j = 0$, $\sum_{j \in \mathbb{C}} a_j x'_j \leq \alpha - a_{j'}$, and $z_{UB} + c_{j'} < z_{UB}^{best}$, column j' is added to the emerging solution, i.e., $x'_{j'} = 1$ and $z_{UB} = z_{UB} + c_{j'}$.

Step 5. **Check if the emerging solution is complete**
If $\sum_{j \in \mathbb{C}_i} x'_j = 1$ for every $i \in V$, the solution is feasible, therefore update the current best solution $z_{UB}^{best} = z_{UB}$, $\boldsymbol{x}^{best} = \boldsymbol{x}'$; otherwise the Lagrangian heuristic was not able to find a feasible solution of cost smaller than z_{UB}^{best}.

Notice that when the Lagrangian Heuristic adds a column j' to the emerging solution all the rows are covered by at most one column, the side constraint is satisfied, and its cost z_{UB} is less than z_{UB}^{best}. Therefore, as soon as the emerging solution covers all rows, it is certainly feasible and better than the current best solution of cost z_{UB}^{best}.

In the computational results, the Lagrangian Heuristic is run only when the percentage gap between the current lower and upper bounds is under the 10% and $LR(\boldsymbol{\lambda}, \boldsymbol{q}) \geq H_{Gap}^1 z_{LB}$ or it is under the 5% and $LR(\boldsymbol{\lambda}, \boldsymbol{q}) \geq H_{Gap}^2 z_{LB}$. The idea is to apply the Lagrangian Heuristic only when the dual solution is sufficiently good (i.e., $H_{Gap}^1 > H_{Gap}^2$). The parameter values H_{Gap}^1 and H_{Gap}^2 can be found at Section 7, where the computational results are described.

When the Lagrangian Heuristic is run, it is repeated four times, one for each criterion for sorting the dimensional values $i \in V$ in phase 1.

6. An Exact Method

Using heuristic algorithms we can obtain effective feasible solutions in a small computing time, and by means of the dual ascent procedure we can evaluate the maximum distance from the optimal solution value. But when we need to evaluate the optimal value, the only possibility is the use of an *exact* method.

In this paper we propose an exact method based on an approach similar to the ones described in [7] and [8].

The proposed approach computes a near optimal dual solution by the Dual Ascent Procedure. It uses the corresponding reduced costs $c'_j = c_j - \sum_{i \in C_j} u_i - a_j v$ and generates a reduced problem P' by replacing in P the set $\mathbb{C}$ with the subset $\mathbb{C}'$ and the original cost c_j with the reduced cost c'_j. The subset $\mathbb{C}'$ is the largest subset of the lowest reduced cost variables such that $c'_j < \min\{g^{max}, z_{UB} - z_{LB}\}$ and $|\mathbb{C}'| < \Delta^{max}$. We solve the resulting reduced problem P' by a MIP solver. Given the solution of P', we are able to check if it is optimal for the original problem P. If it is not optimal, we enlarge the subset $\mathbb{C}'$ and we solve the new reduced problem again.

The resulting exact method can be summarized as follows.

Exact Algorithm

Step 1. **Initial setup**

Set $z_{LB} = -\infty$, $z_{UB} = \infty$, $iter = 1$, and $\Delta^{max} = \Delta_0$.

Step 2. **Computing a lower bound z'_D**

Compute a solution $(\mathbf{u}', v')$ of the dual problem D of cost $z_{LB} = z'_D$ using the Dual Ascent embedding the Lagrangian Heuristic which provides an upper bound z_{UB}. Set $g^{max} = \mu_1 z_{LB}$.
If $z_{LB} = z_{UB}$, then STOP.

Step 3. **Define a reduced problem P'**

Let $c'_j = c_j - \sum_{i \in C_j} u_i - a_j v$ be the reduced cost of cluster $j \in \mathbb{C}$ with respect to the dual solution $(\mathbf{u}', v')$.
Let $\mathbb{C}'$ be the largest subset of the lowest reduced cost variables such that $c'_j < \min\{g^{max}, z_{UB} - z_{LB}\}$ and $|\mathbb{C}'| < \Delta^{max}$.
Define the reduced problem P' replacing in P the set $\mathbb{C}$ with $\mathbb{C}'$ and replacing the original cost c_j with the reduced cost c'_j.

Step 4. **Solve problem P'**

Solve problem P' using a general purpose MIP solver (e.g., IBM Ilog Cplex).

Let $z^*_{P'}$ be the cost of the optimal solution $\mathbf{x}^*$ obtained (we assume $z^*_{P'} = \infty$ if the set $\mathbb{C}'$ does not contain any feasible solution).

Update $z_{UB} = \min\{z_{UB}, z^*_{P'} + z'_D\}$.

Step 5. **Test if $\mathbf{x}^*$ is optimal for the original problem P**

Let $c_{max} = \max\{c'_j : j \in \mathbb{C}'\}$, if $\mathbb{C}' \subset \mathbb{C}$, otherwise $c_{max} = \infty$, if $\mathbb{C}' = \mathbb{C}$. We have two cases:

(a) $z^*_{P'} \leq c_{max}$, then Stop because $\mathbf{x}^*$ is guaranteed to be an optimal solution for the original problem P.

(b) $z^*_{P'} > c_{max}$, then $\mathbf{x}^*$ is not guaranteed to be an optimal solution for the original problem P, however $z'_D + c_{max}$ is a valid lower bound on the optimal solution value of problem P.

Step 6. **Update the parameters** If $iter < MaxIter$, then increase $\Delta^{max} = \mu_2\Delta^{max}$ and $g^{max} = \mu_2 g^{max}$, $\mu_2 > 1$, set $iter = iter + 1$ and go to Step 3.

At Steps 3 and 4 we use the reduced cost c'_j instead of the original cost c_j, because the solution of the LP-relaxation of P', at *node zero*, is usually faster (i.e., we incorporate the dual information in P', see [8]).

The procedure terminates when the optimal solution of P is obtained or the maximum number of iterations is reached. Notice that if we set $MaxIter = \infty$, the procedure converges to the optimal solution because in the worst case at a given iteration $\mathbb{C}' = \mathbb{C}$.

7. Computational Results

The algorithms presented in this paper were coded in C++ using Microsoft Visual Studio Community 2017, and run on a workstation equipped with an Intel Core i7-3770, 3.40 GHz, 32Gb of RAM, and operating system Windows 10 Educational (version 1803) 64bit. IBM Ilog CPLEX 12.5 is used as LP and MIP solver.

For our experiments we considered four different hierarchies: RESIDENCE, OCCUPATION, PROD_DEPARTMENT, and PROD_BRAND. The former two come from the IPUMS database [29], while the others two are extracted from the Foodmart database that can be found with the Pentaho suite [35]. After aggregating input data along these hierarchies (e.g., by state or by city), we generated, by means of sampling, several test instances with varying characteristics, such as size and average fan-out (i.e., children per parent ratio). In the remainder of this paper we refer to these instances using the name of the attribute used to aggregate the data, followed by a progressive number; for example, CITY-1 means that the data has been first aggregated by city, and then a sampling process has been performed to create the dataset. To observe how the algorithms behave not only with different problem sizes (i.e., number of clusters) but also with different data distributions, we generated instances CITY_UNI and OCCUPATION_UNI by reusing the hierarchical structure of RESIDENCE and OCCUPATION, but with uniformly random data slices. Finally, we generated some hard instances to show that the new proposed algorithm solves instances where a general-purpose solver fails. A thorough description of the dataset can be found in [15].

For every test instance we solve both versions of the problem: the problem of *Type A*, where the objective function minimizes the size of the resulting data and the side constraint guarantees that the loss of precision does not exceed a given maximum value; the problem of *Type B*, where the objective function minimizes the loss of precision and the side constraint guarantees that the size of the resulting data does not exceed a given maximum value.

For every problem type we solve the problem for different values of the maximum loss of precision or of the maximum size of the data.

In this section we summarize the computational results in Tables 2 and 3, while the complete description of the results are reported in the Appendix A in Tables A.4–A.11. When we report in the tables the value of the the maximum loss of precision, we use the notation $1.00M$ and $1.00G$ for representing the values 1.00×10^6 and 1.00×10^9, respectively. When a computing time or a percentage gap is equal to 0.00, it means that its real value is smaller that 0.01.

In Tables 2 and 3 the test instances are grouped by the value of the right-hand side α of the side constraint, which is the maximum loss of precision for problem of Type A and the maximum size of the data for problem of Type B. For each group, these tables report the average Avg, the maximum Max, and the standard deviation $s.d.$ for each column.

Notice that groups having very small values of α (i.e., $\alpha = 1.00M$ and $\alpha = 10.00M$, for problems of Type A, and $\alpha = 10$ and $\alpha = 15$, for problems of Type B) correspond to few small size instances.

7.1. Dual Ascent procedure

In our computational experiments the parameters of the dual ascent are set as follows. The parameters for defining the step size are $\beta_0 = 20$, $\gamma_1 = 0.90$, $\gamma_2 = 1.10$, for problems of Type A, and $\beta_0 = 1$, $\gamma_1 = 0.90$, $\gamma_2 = 1.005$, for problems of Type B. The parameter ε_0 used for reducing the size of the subset $\mathbb{C}'$ is 0.05, i.e., 5% of the value of the current lower bound. Instead, the parameter ε_1 used to check if the lower bound has improved enough during the last $MaxIter_0$ iterations is $\varepsilon_1 = 0.001$. The maximum number of iterations are $MaxIter = 100000$ (i.e., virtually unlimited), $MaxIter_0 = 200$, and $MaxIter_1 = 5$.

The choice of the parameter values is made empirically, because the purpose of the computational tests is to show that just with a *good* choice we can achieve effective results. Therefore, a better analysis on the choice of the parameter values is outside of the scope of this paper, but it will be an interesting research direction for the future.

In Table 2 we compare the results obtained by the CPLEX LP solvers and by the dual ascent procedure, described in Section 4.

For the CPLEX solvers we report the computing times $Time_x$ for each solver available: primal (P), dual (D), network (N), barrier (B), and sifting (S). For the dual ascent we report the percentage gap between the best lower bound z_{LB}, corresponding to the best feasible dual solution generated, and the LP-relaxation optimal value z_{LP}, $Gap_{LP} = 100 \times \frac{z_{LP} - z_{LB}}{z_{LP}}$, the number of iterations *Iter*, the computing time $Time$, and the size of subset $\mathbb{C}'$ at the end of the execution.

The more effective CPLEX LP solver is the network simplex, in particular for problems of Type B, at the contrary the barrier is very time consuming for both problem types. To improve the results provided by the barrier algorithm we also tried to apply all the barrier algorithms available by the parameter CPX_PARAM_BARALG, without any improvement. The total number of barrier iterations, returned by function “CPXgetbaritcnt”, ranges from few tens to some hundreds, for the most difficult instances, and in the worst case it reaches 648 iterations (i.e., instance “*city11*”, see Appendix A). We are not able to explain the poor performance of the barrier algorithm. It is an another interesting topic for future research.

Even the primal and the dual simplex do not work as expected. The difference in the performance is more evident for medium-large size instances and seems to depend by the number of clusters in the optimal solution: the smaller the number of clusters the greater the difference in the performance. Notice that for a problem of Type A a greater value of α allows for a smaller number of clusters in the optimal solutions. The reason for these differences in performance is unclear and further investigation will be needed to provide an explanation. Perhaps, these instances have a particular structure that gives an advantage to the network simplex.

The dual ascent provides near optimal solutions in a smaller computing time with respect to CPLEX LP solvers for problems of type A. It generates lower bounds having an average percentage gap Gap_{LP} from the optimal solution value z_{LP} equal to 0.02% and it is on average faster than the better CPLEX LP solver (see Table A.4). For problems of Type B, the dual ascent is a little worse only with respect to the network simplex and the average percentage gap Gap_{LP} is under 0.01% (see Table A.5). Even the results on the hard instances, reported in Tables A.6 and A.7, confirm these figures.

Setting a smaller $MaxIter_0$ and a more aggressive value for β_0 we can obtain a faster convergence of the dual ascent with a smaller worsening of the lower bound provided. However, the convergence is more erratic and the quality of the solutions generated by the embedded Lagrangian heuristic is a little worse. Since we are mainly interested in the heuristic or optimal solution of the problem, we prefer a slower dual ascent which enhances the quality of the solutions generated by the Lagrangian heuristic.

7.2. Greedy and Lagrangian Heuristics

In Table 3 we compare the greedy and the Lagrangian heuristics, described in Sections 2 and 5. The results of the Lagrangian heuristic include the dual ascent procedure which embeds it.

For each group of test instances we report the percentage gap between the best upper bound found z_{UB} and the value of the optimal integer solution, $Gap = 100 \times \frac{z_{UB} - z_{Opt}}{z_{Opt}}$, and the computing time *Time*. For the dual ascent with the Lagrangian heuristic we also report the number of iterations *Iter*, and the size of $\mathbb{C}'$.

Table 2: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type A and B.

			Cplex					**Dual Ascent**			
			$Time_P$	$Time_D$	$Time_N$	$Time_B$	$Time_S$	Gap_{LP}	$Iter$	$Time$	$\lvert\mathbb{C}'\rvert$
Type A	$\alpha = 1.00M$	avg	0.67	0.68	0.68	0.67	1.06	0.08	968	0.10	1393
		max	0.67	0.68	0.68	0.67	1.06	0.08	968	0.10	1393
		s.d.	0.00	0.00	0.00	0.00	0.00	0.00	0	0.00	0
	$\alpha = 10.00M$	avg	0.72	0.76	0.72	1.21	1.77	0.06	897	0.19	398
		max	0.72	0.76	0.72	1.21	1.77	0.06	897	0.19	398
		s.d.	0.00	0.00	0.00	0.00	0.00	0.00	0	0.00	0
	$\alpha = 100.00M$	avg	6.39	6.37	6.35	6.29	7.42	0.02	965	3.59	995
		max	27.37	27.42	27.21	25.18	31.23	0.06	1169	14.22	1571
		s.d.	8.10	8.09	8.02	7.38	9.19	0.01	143	4.22	349
	$\alpha = 1000.00M$	avg	0.73	0.73	0.75	0.97	0.89	0.02	759	0.48	713
		max	1.77	1.76	1.82	2.42	2.19	0.03	991	1.24	1403
		s.d.	0.65	0.64	0.66	0.84	0.78	0.01	271	0.42	567
	$\alpha = 10.00G$	avg	14.13	10.89	15.11	243.32	13.01	0.02	987	6.92	12130
		max	53.36	39.84	65.23	1014.29	44.96	0.04	1219	24.43	34057
		s.d.	16.06	12.26	18.55	300.57	13.43	0.01	184	7.18	10005
	$\alpha = 100.00G$	avg	24.61	30.01	29.39	566.75	28.29	0.03	950	11.02	83344
		max	126.64	163.83	178.73	3097.50	115.07	0.07	1179	46.13	268111
		s.d.	33.33	43.87	45.08	861.89	32.97	0.02	93	12.46	80787
	$\alpha = 1000.00G$	avg	5.06	4.78	3.69	68.96	8.42	0.03	976	4.15	62576
		max	8.01	9.47	6.43	108.05	12.83	0.03	1176	5.80	80112
		s.d.	2.38	2.99	1.99	35.62	2.86	0.01	131	1.70	16758
Type B	$\alpha = 10$	avg	0.04	0.04	0.04	0.11	0.04	0.00	635	0.04	14
		max	0.08	0.07	0.07	0.20	0.07	0.00	689	0.04	14
		s.d.	0.04	0.04	0.04	0.10	0.03	0.00	54	0.00	1
	$\alpha = 15$	avg	0.04	0.03	0.03	0.60	0.05	0.00	917	0.03	23
		max	0.07	0.06	0.06	1.19	0.07	0.00	1317	0.04	31
		s.d.	0.04	0.03	0.03	0.59	0.03	0.00	401	0.02	8
	$\alpha = 50$	avg	27.76	82.10	31.06	443.15	105.64	0.00	1223	32.21	1691
		max	146.19	516.51	242.31	2419.11	596.38	0.00	1480	152.99	4553
		s.d.	39.50	135.53	58.29	658.33	144.57	0.00	176	42.96	694
	$\alpha = 100$	avg	30.78	61.87	33.90	53.91	25.75	0.00	1145	27.93	1366
		max	148.85	425.24	205.82	308.02	92.98	0.01	1522	138.94	1975
		s.d.	43.73	102.55	55.04	76.88	30.18	0.00	172	37.42	320
	$\alpha = 150$	avg	30.95	44.15	25.64	84.88	13.25	0.00	1127	20.84	1216
		max	148.61	230.62	152.53	439.19	62.73	0.04	1313	104.17	1805
		s.d.	43.75	65.89	38.52	121.22	17.47	0.01	103	27.56	315

The computing time of the greedy heuristic is negligible but the percentage gap from the optimal solution value is on average 1.28% and 3.74% for the problems of type A and B (reported in Tables A.8 and A.9, respectively), and is on average 0.57% and 5.09% for the hard instances of type A and B (reported in Tables A.10 and A.11). The maximum gap is the 25% and is often greater than 5%.

For the Lagrangian heuristic we have set the parameters $H^1_{Gap} = 0.1\%$ and $H^2_{Gap} = 0.001\%$, for problems of Type A, and $H^1_{Gap} = 1\%$ and $H^2_{Gap} = 0.001\%$, for problems of Type B (see Section 5). The Lagrangian heuristic is more time consuming but the percentage gap from the optimal solution value is much smaller. For problems of type A it finds the optimal solution for all the instances. The quality of the upper bound is a little worse for problems of type B, where only some instances having $\alpha = 50$ and $\alpha = 100$ are not solved to optimality: the average gap is 0.17% and 1.65%, respectively. For the very large instances of Type B, the maximum gap is 8.66%, but it is still much better than the greedy heuristic.

7.3. Exact Method

In order to evaluate the effectiveness of the Lagrangian heuristic, described in Section 5, and of the exact method, described in Section 6, in Table 3 we compare them with the CPLEX MIP solver.

For the proposed exact method in our computational results we set $MaxIter = 10$, $\Delta_0 = 1000$, $\mu_1 = 0.001$, and $\mu_2 = 10$. For the exact method we setup a less aggressive setting for the Lagrangian heuristic by setting the parameters $H^1_{Gap} = H^2_{Gap} = 0.001\%$, for both problems of Type A and B (see Section 5). We made this choice because the exact method requires a small computing time for closing a possible gap between the upper bound and the optimal solution. In this case it is convenient to avoid a time consuming most aggressive setting.

For the CPLEX MIP solver we report the computing time *Time*. However, for two instances of Type A and for five instances of Type B the CPLEX MIP solver fails because of an "*out of memory*". For these instances column *Time* reports the computing time spent to generate the error (see Tables A.10 and A.11 in Appendix A for more details).

Table 3: Comparison among CPLEX MIP solver, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type A and B.

			Greedy		**Cplex**	**Dual Ascent + Lagr. Heu.**				**Exact Method**					
			Gap	*Time*	*Time*	*Gap*	*Iter*	*Time*	$\lvert\mathbb{C}'\rvert$	*Gap*	$\lvert\mathbb{C}'\rvert$	*Iter*	T_A	T_E	T_{Tot}
Type A	$\alpha = 1.00M$	avg	0.60	0.01	0.73	0.00	766	0.16	1393	0.00	0	1.00	0.14	0.00	0.14
		max	0.60	0.01	0.73	0.00	766	0.16	1393	0.00	0	1.00	0.14	0.00	0.14
		s.d.	0.00	0.00	0.00	0.00	0	0.00	0	0.00	0	0.00	0.00	0.00	0.00
	$\alpha = 10.00M$	avg	8.82	0.01	1.49	0.00	897	0.39	398	0.00	282	2.00	0.24	0.34	0.58
		max	8.82	0.01	1.49	0.00	897	0.39	398	0.00	282	2.00	0.24	0.34	0.58
		s.d.	0.00	0.00	0.00	0.00	0	0.00	0	0.00	0	0.00	0.00	0.00	0.00
	$\alpha = 100.00M$	avg	0.11	0.01	7.10	0.00	694	24.72	995	0.00	45	1.00	19.81	0.11	19.91
		max	0.32	0.01	29.37	0.00	878	76.58	1570	0.00	805	1.00	76.58	1.89	76.58
		s.d.	0.14	0.00	8.59	0.00	184	25.23	349	0.00	184	0.00	23.88	0.43	24.05
	$\alpha = 1000.00M$	avg	4.17	0.00	1.25	0.00	525	3.35	713	0.00	0	1.00	2.04	0.00	2.04
		max	25.00	0.01	2.93	0.00	736	9.52	1403	0.00	0	1.00	5.65	0.00	5.65
		s.d.	9.32	0.00	0.98	0.00	254	3.32	567	0.00	0	0.00	1.98	0.00	1.98
	$\alpha = 10.00G$	avg	0.58	0.01	118.50	0.00	721	28.96	12130	0.03	56	1.00	24.45	0.03	24.49
		max	1.54	0.02	525.40	0.00	1046	109.79	34057	0.58	1000	1.00	109.79	0.59	109.79
		s.d.	0.46	0.00	155.77	0.00	218	31.28	10005	0.13	229	0.00	30.32	0.14	30.32
	$\alpha = 100.00G$	avg	1.31	0.01	179.42	0.00	654	20.42	83344	0.00	9	1.00	19.26	0.00	19.26
		max	2.74	0.02	814.73	0.00	979	93.66	268109	0.00	198	1.00	93.66	0.06	93.66
		s.d.	0.83	0.00	248.19	0.00	112	23.77	80786	0.00	42	0.00	23.85	0.01	23.85
	$\alpha = 1000.00G$	avg	1.38	0.01	129.29	0.00	753	6.78	62576	0.37	194	1.00	5.55	0.12	5.67
		max	2.74	0.01	303.56	0.00	1001	9.95	80112	1.47	461	1.00	9.51	0.32	9.83
		s.d.	0.97	0.00	112.43	0.00	149	3.26	16758	0.64	200	0.00	2.79	0.13	2.90
Type B	$\alpha = 10$	avg	0.00	0.00	0.71	0.00	635	0.05	14	0.00	0	1.00	0.05	0.00	0.05
		max	0.00	0.00	1.28	0.00	689	0.05	14	0.00	0	1.00	0.05	0.00	0.05
		s.d.	0.00	0.00	0.57	0.00	54	0.00	1	0.00	0	0.00	0.00	0.00	0.00
	$\alpha = 15$	avg	3.64	0.00	0.71	0.00	917	0.06	23	0.00	43	2.00	0.06	0.12	0.17
		max	7.28	0.00	1.27	0.00	1317	0.08	31	0.00	86	3.00	0.08	0.23	0.30
		s.d.	3.64	0.00	0.57	0.00	401	0.02	8	0.00	43	1.00	0.02	0.12	0.13
	$\alpha = 50$	avg	5.30	0.01	189.20	1.65	1204	86.66	1691	2.11	47830	1.41	60.57	35.72	96.29
		max	8.66	0.02	974.37	8.66	1480	351.22	4553	8.66	1000000	4.00	301.36	762.65	1064.02
		s.d.	1.87	0.00	254.38	2.69	196	101.95	694	2.71	207818	0.72	90.21	158.64	224.08
	$\alpha = 100$	avg	4.25	0.01	150.58	0.17	1072	52.68	1366	0.39	248	1.05	38.69	0.60	39.29
		max	8.78	0.02	789.57	3.45	1522	228.60	1975	3.45	2949	2.00	209.75	3.30	213.04
		s.d.	2.22	0.00	209.30	0.72	178	63.97	320	0.97	592	0.21	53.66	0.85	54.44
	$\alpha = 150$	avg	2.70	0.01	126.55	0.00	1037	44.99	1216	0.06	179	1.00	30.37	0.67	31.04
		max	5.26	0.02	721.90	0.00	1258	175.00	1805	1.34	287	1.00	175.00	3.23	178.23
		s.d.	1.46	0.00	172.81	0.00	88	47.65	315	0.29	64	0.00	43.94	0.88	44.79

For the dual ascent embedding the Lagrangian heuristic, we report the gap between the best upper bounds found and the optimal solution, $Gap = 100 \times \frac{z_{UB} - z_{Opt}}{z_{Opt}}$ and the computing time T_A. For the exact method, we report the size of the subset of columns $\mathbb{C}'$ considered in the integer reduced problem P', the number of iterations *Iter*, the computing time T_E for solving the reduced problem P' (even more than one time if $Iter > 1$), and the overall computing time T_{Tot} which also includes the computing time of the dual ascent procedure and of the embedded Lagrangian heuristic. When the problem is solved by the dual ascent, we report $|\mathbb{C}'| = 0$.

Notice that we used a less aggressive setting for the Lagrangian heuristic for the exact method, therefore the *Gap* between the best upper bounds found and the optimal solution is sometimes greater than the one found by the more aggressive Lagrangian heuristic. The advantage is a smaller computing time.

The exact method performs very well for problems of Type A, where it needs to solve the integer reduced problem P' only for six instances using a very small subset of columns $\mathbb{C}'$. Only for one instance the exact method requires two iterations. The proposed exact method is on average about seven times faster than the CPLEX MIP solver and for two hard instances the CPLEX MIP solver runs out of memory.

Problems of Type B are more difficult to solve, but the exact method is on average about five times faster than the CPLEX MIP solver. As shown in Appendix A by Table A.9, for many instances the exact method needs to solve the integer reduced problem P'. However, the size of the subset $\mathbb{C}'$ is still small and the computing time for solving the reduced problem P' is very small. We need two iterations only for seven instances, three iterations for one instance, and four iterations for one hard instance. All the remaining instances are solved in one iteration. For only one hard instance of type B, the exact method requires about 18 minutes, but for the same instance the CPLEX MIP solver fails. Overall, the CPLEX MIP solver fails for five hard instances of Type B.

The most difficult medium-large instances are the ones having a small number of clusters in the optimal solution. For problems of Type A, they are imposed by the side constraint with a small right-hand-side (i.e., the maximum number of clusters). For problems of Type B, they are obtained by minimizing the number of clusters having a side constraint that allows a large error with a large right-hand-side.

8. Conclusions

In this paper we have proposed an integer linear programming model for solving the problem of implementing effectively the OLAP shrink operator.

We have modelled the problem as a set partitioning problem with one side constraint and we have considered two different approaches for finding its solution. In the first one (problem of Type A), we minimize the size of the resulting data, while the side constraint guarantees that the loss of precision does not exceed a given maximum value. In the second approach (problem of Type B), we minimize the loss of precision, while the side constraint guarantees that the size of the resulting data does not exceed a given maximum value.

The proposed mathematical formulation is able to model both problem types. For switching from one type to the other, it is sufficient to modify the coefficients of the objective function and of the side constraint, along with its right-hand side.

The first solution method considered is a dual ascent which embeds a Lagrangian heuristic. The dual ascent generates at each iteration a hopefully feasible dual solution of the LP-relaxation of the problem. The dual ascent only considers a reduced subset of columns to solve the problem and uses the generated feasible dual solutions for adding columns to the *reduced problem* using the pricing. The computing time allows an operational use of the procedure and the quality of the solution generated is of very good quality. For problems of Type A the dual ascent significantly outperforms general purpose LP solvers as CPLEX. It is able to generate a near optimal dual solution in a short computing time.

We have also proposed an exact method to use when the optimal solution is required. After running the dual ascent embedding the Lagrangian heuristic, the proposed exact procedure generates, with a very small additional computing time, an optimal solution using a very small subset of the columns of the original instance. Therefore, the exact procedure has the potential for an operational use, while a general purpose solver, like CPLEX, is time consuming and fails for some instances.

The computational results show the maximum instance sizes that can be solved to optimality and they are much smaller than the size of instances of similar problems such as the *microaggregation* (see Section 1).

For the instances used in this paper, the computing time for generating the clusters is almost negligible with respect to the time for solving the problem, even if the number of columns is huge for many instances. The use of pricing to select a small subset of columns allows a huge reduction of the overall computing time without compromising the optimal solution of the problem. However, future developments could embed a column generation procedure in the proposed algorithms in order to solve much larger instances, at least of one further order of magnitude, where the complete generation of clusters takes time and requires a huge amount of memory.

Appendix A. Complete computational results

For each instance, the number of values in the hierarchy and the number of generated clusters are shown alongside the results of the experiments in Tables A.4–A.7.

For every test instance we solve both versions of the problem. In Tables A.4, A.6, A.8, and A.10 we solve the problem of *Type A*. Whereas, in Tables A.5, A.7, A.9, and A.11 we solve the problem of *Type B*.

Appendix A.1. Dual Ascent procedure

In Tables A.4, A.5, A.6, and A.7 we compare the CPLEX LP solvers and the dual ascent procedure, described in Section 4.

For each test instance we report the number of clusters m, the number of dimensional values n, the computing time for generating the clusters T_{Gen}, and the right-hand side α of the side constraint, which is the maximum loss of precision for problem of Type A and the maximum size of the data for problem of Type B. For the CPLEX solvers we report the optimal value z_{LP} of the LP-relaxation of the problem P and the computing times $Time_x$ for each solver available: primal (P), dual (D), network (N), barrier (B), and sifting (S). For the dual ascent we report the best lower bound z_{LB} corresponding to the best feasible dual solution generated, its percentage gap from z_{LP} $Gap_{LP} = 100 \times \frac{z_{LP}-z_{LB}}{z_{LP}}$, the number of iterations $Iter$, the computing time $Time$, and the size of subset $\mathbb{C}'$ at the end of the execution.

Table A.4: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type A.

	Size and Generation			Side Cons.	Cplex						Dual Ascent				
Instances	m	n	T_{Gen}	α	z_{LP}	$Time_P$	$Time_D$	$Time_N$	$Time_B$	$Time_S$	z_{LB}	Gap_{LP}	$Iter$	$Time$	$\lvert\mathbb{C}'\rvert$
state	629	46	0.00	10.00G	29.60	0.00	0.00	0.00	0.00	0.00	29.60	0.00	365	0.00	36
state	629	46	0.00	1000.00M	32.00	0.00	0.00	0.00	0.00	0.00	32.00	0.00	202	0.04	32
prod_department	32809	28	0.04	1000.00M	3.51	0.06	0.09	0.09	0.22	0.11	3.51	0.03	676	0.06	20
prod_department	32809	28	0.04	100.00M	9.31	0.04	0.04	0.04	0.04	0.06	9.31	0.01	519	0.06	34
prod_brand	37233	697	0.06	10.00M	33.01	0.72	0.76	0.72	1.21	1.77	32.98	0.06	897	0.19	398
prod_brand	37233	697	0.06	1.00M	167.70	0.67	0.68	0.68	0.67	1.06	167.57	0.08	968	0.10	1393
city-1	64069	495	0.09	100.00G	72.04	0.29	0.34	0.30	2.01	1.39	72.00	0.07	979	0.32	6903
city-1	64069	495	0.09	10.00G	140.34	0.20	0.22	0.30	1.65	0.40	140.32	0.02	1001	0.21	2725
city-1	64069	495	0.09	100.00M	306.22	0.09	0.09	0.10	0.13	0.15	306.18	0.01	1008	0.10	802
city-2	121413	523	0.07	100.00G	75.55	0.78	0.52	0.41	6.58	2.36	75.50	0.07	1049	0.79	19221
city-2	121413	523	0.07	10.00G	148.69	0.47	0.37	0.41	4.00	0.67	148.69	0.00	1069	0.40	4629
city-2	121413	523	0.07	100.00M	324.56	0.17	0.16	0.17	0.24	0.25	324.54	0.01	1087	0.16	930
city-3	227909	549	0.22	100.00G	77.91	1.56	1.17	0.94	14.92	3.76	77.87	0.05	924	1.23	21669
city-3	227909	549	0.22	10.00G	153.97	0.97	0.72	0.87	10.36	1.46	153.91	0.04	878	0.52	4367
city-3	227909	549	0.22	100.00M	341.60	0.32	0.32	0.33	0.43	0.43	341.56	0.01	918	0.23	985
city-4	432709	574	0.24	100.00G	82.46	3.06	2.58	2.39	34.08	7.77	82.45	0.02	1074	2.81	49204
city-4	432709	574	0.24	10.00G	158.09	1.93	1.37	1.84	22.93	2.45	158.08	0.01	1103	1.24	6056
city-4	432709	574	0.24	100.00M	353.65	0.60	0.61	0.63	0.84	0.77	353.44	0.06	913	0.41	1131
city-5	647749	584	0.35	100.00G	81.23	4.80	4.35	3.82	65.23	10.57	81.17	0.07	1024	3.20	40864
city-5	647749	584	0.35	10.00G	158.45	2.84	2.33	2.88	31.61	3.28	158.41	0.03	941	1.76	9390
city-5	647749	584	0.35	100.00M	355.73	0.92	0.92	0.95	1.31	1.26	355.68	0.02	905	0.59	1149
city-6	793157	596	0.56	100.00G	84.26	5.47	5.81	3.63	76.94	8.85	84.23	0.04	848	4.41	72676
city-6	793157	596	0.56	10.00G	161.79	3.38	2.61	3.89	46.55	3.86	161.75	0.03	1028	2.35	10283
city-6	793157	596	0.56	100.00M	364.93	1.13	1.13	1.17	1.55	1.51	364.90	0.01	924	0.74	1274
city-7	1184157	516	0.69	100.00G	71.71	7.94	12.38	6.18	100.72	15.68	71.69	0.03	891	5.86	91528
city-7	1184157	516	0.69	10.00G	132.34	5.09	4.00	4.71	70.69	5.10	132.29	0.03	906	3.23	15867
city-7	1184157	516	0.69	100.00M	313.63	1.75	1.74	1.77	2.14	2.20	313.59	0.01	1025	1.21	1190
city-8	2167197	531	1.29	100.00G	72.30	14.66	15.16	12.88	251.36	24.45	72.27	0.04	824	8.94	135015
city-8	2167197	531	1.29	10.00G	133.62	8.55	7.72	10.15	160.92	11.26	133.59	0.02	815	5.17	31877
city-8	2167197	531	1.29	100.00M	318.95	3.42	3.42	3.44	4.28	4.26	318.94	0.00	1096	2.40	1099
city-9	4133801	573	2.12	100.00G	78.83	30.02	34.69	38.67	648.90	38.19	78.82	0.01	923	17.14	108384
city-9	4133801	573	2.12	10.00G	146.90	22.45	15.80	22.59	398.05	17.30	146.89	0.01	1000	9.71	25124
city-9	4133801	573	2.12	100.00M	347.41	7.01	6.96	6.99	7.92	8.36	347.36	0.02	1169	4.93	1227
city-10	2693701	635	1.57	100.00G	93.37	22.37	16.01	18.36	435.50	40.22	93.36	0.02	928	15.28	200346
city-10	2693701	635	1.57	10.00G	172.91	12.02	9.76	12.60	191.97	17.07	172.86	0.03	929	6.77	21073
city-10	2693701	635	1.57	100.00M	399.50	4.29	4.29	4.32	5.23	5.13	399.45	0.01	1150	3.21	1335
city-11	4921925	652	2.93	100.00G	93.89	40.42	52.41	46.38	850.89	51.98	93.87	0.02	938	17.89	142600
city-11	4921925	652	2.93	10.00G	174.12	26.82	19.89	26.60	479.06	26.91	174.10	0.01	993	11.76	34057
city-11	4921925	652	2.93	100.00M	405.85	8.33	8.29	8.34	9.54	9.85	405.76	0.02	1083	5.54	1571
occupation-1	875995	357	0.53	100.00G	66.82	5.87	6.23	4.47	97.90	7.12	66.79	0.04	960	3.15	17557
occupation-1	875995	357	0.53	10.00G	129.32	2.29	1.85	2.08	22.25	2.54	129.28	0.03	1124	1.49	1386
occupation-1	875995	357	0.53	100.00M	259.20	1.27	1.26	1.25	1.13	1.50	259.18	0.01	946	0.69	469
occupation-2	3300827	382	2.01	100.00G	69.54	24.93	31.97	23.68	622.17	27.29	69.52	0.02	1179	11.55	24109
occupation-2	3300827	382	2.01	10.00G	135.67	9.45	9.16	9.40	116.03	8.62	135.66	0.01	1217	5.89	1523
occupation-2	3300827	382	2.01	100.00M	279.65	5.56	5.55	5.52	5.02	6.46	279.64	0.00	809	2.51	508
city_uni-1	1184157	516	0.67	1000.00G	67.72	8.01	9.47	6.43	108.05	12.83	67.70	0.03	894	5.80	77980
city_uni-1	1184157	516	0.67	100.00G	126.71	5.38	4.21	6.01	79.84	6.71	126.69	0.01	860	3.39	28631
city_uni-1	1184157	516	0.67	1000.00M	303.57	1.77	1.76	1.82	2.42	2.19	303.52	0.02	991	1.24	1205
city_uni-2	227909	549	0.13	1000.00G	72.07	1.47	1.14	0.83	14.59	4.95	72.05	0.03	831	1.48	41864
city_uni-2	227909	549	0.13	100.00G	147.19	1.04	0.71	0.89	9.80	1.54	147.18	0.01	958	0.55	4170
city_uni-2	227909	549	0.13	1000.00M	337.34	0.32	0.32	0.35	0.50	0.45	337.30	0.01	938	0.25	1160
city_uni-3	647749	584	0.37	1000.00G	74.99	4.76	4.28	3.47	61.18	8.58	74.97	0.03	1001	3.88	50347
city_uni-3	647749	584	0.37	100.00G	151.69	3.04	2.66	3.17	32.41	3.65	151.67	0.02	877	1.58	7429
city_uni-3	647749	584	0.37	1000.00M	352.75	0.95	0.94	1.00	1.56	1.20	352.68	0.02	949	0.67	1403
occupation_uni	875995	357	0.52	1000.00G	64.24	6.01	4.22	4.03	92.02	7.32	64.23	0.01	1176	5.43	80112
occupation_uni	875995	357	0.52	100.00G	124.11	2.62	1.92	2.30	26.33	2.35	124.11	0.00	1074	1.38	1323
occupation_uni	875995	357	0.52	1000.00M	253.41	1.25	1.25	1.24	1.12	1.40	253.33	0.03	797	0.59	458
Avg			0.76			5.82	5.77	5.76	91.84	7.70		0.02	939	3.38	24739
Max			2.93			40.42	52.41	46.38	850.89	51.98		0.08	1217	17.89	200346
s.d.			0.81			8.44	9.44	9.22	179.75	10.72		0.02	176	4.27	41163

Table A.5: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type B.

	Size and Generation			Side Cons.	Cplex						Dual Ascent				
Instances	m	n	T_{Gen}	α	z_{LP}	$Time_P$	$Time_D$	$Time_N$	$Time_B$	$Time_S$	z_{LB}	Gap_{LP}	$Iter$	$Time$	$\lvert\mathbb{C}'\rvert$
state	629	46	0.00	10	8295.08G	0.00	0.00	0.00	0.01	0.01	8295.08G	0.00	689	0.04	13
state	629	46	0.00	15	3274.11G	0.00	0.00	0.00	0.01	0.02	3274.06G	0.00	1317	0.01	31
prod_department	32809	28	0.04	10	65.37M	0.08	0.07	0.07	0.20	0.07	65.37M	0.00	581	0.04	14
prod_department	32809	28	0.04	15	4.70M	0.07	0.06	0.06	1.19	0.07	4.70M	0.00	516	0.04	15
prod_brand	37233	697	0.06	50	4.63M	1.55	1.26	1.14	89.67	1.02	4.63M	0.00	710	0.12	1213
prod_brand	37233	697	0.06	100	1.81M	1.44	1.25	1.21	15.79	1.00	1.81M	0.00	808	0.09	1174
city-1	64069	495	0.09	50	198.40G	0.37	0.54	0.22	4.87	2.64	198.39G	0.00	1325	0.59	1120
city-1	64069	495	0.09	100	39.57G	0.35	0.46	0.26	1.44	0.60	39.57G	0.00	1220	0.49	1072
city-1	64069	495	0.09	150	7.50G	0.31	0.39	0.27	1.25	0.30	7.50G	0.00	998	0.36	943
city-2	121413	523	0.07	50	214.22G	0.84	1.08	0.41	8.73	12.68	214.22G	0.00	1303	1.14	1295
city-2	121413	523	0.07	100	45.78G	0.77	0.98	0.48	2.59	1.47	45.78G	0.00	915	0.96	1165
city-2	121413	523	0.07	150	9.65G	0.68	0.83	0.51	2.38	0.60	9.65G	0.00	1025	0.79	1199
city-3	227909	549	0.22	50	230.64G	1.48	2.43	0.86	17.63	24.58	230.64G	0.00	1108	2.08	1372
city-3	227909	549	0.22	100	50.62G	1.47	2.00	0.97	4.11	2.29	50.62G	0.00	1473	2.09	1299
city-3	227909	549	0.22	150	11.15G	1.24	1.68	1.03	4.47	1.01	11.15G	0.00	1255	1.66	1140
city-4	432709	574	0.24	50	259.88G	3.26	5.19	2.19	33.23	21.60	259.88G	0.00	1282	3.91	1488
city-4	432709	574	0.24	100	57.25G	3.06	4.40	2.20	6.81	4.73	57.25G	0.01	1298	3.91	1393
city-4	432709	574	0.24	150	12.40G	3.19	3.65	2.21	6.65	1.91	12.40G	0.00	1112	2.95	1290
city-5	647749	584	0.35	50	256.84G	5.43	9.12	2.96	59.97	29.04	256.83G	0.00	1328	6.00	1584
city-5	647749	584	0.35	100	56.70G	5.56	7.49	3.54	8.95	12.95	56.70G	0.00	1135	5.57	1443
city-5	647749	584	0.35	150	12.56G	4.12	6.28	3.51	12.15	2.64	12.56G	0.00	1157	4.34	1270
city-6	793157	596	0.56	50	275.43G	5.93	13.65	3.42	82.16	35.93	275.43G	0.00	1325	7.51	1550
city-6	793157	596	0.56	100	61.62G	6.63	9.47	4.39	9.52	7.36	61.62G	0.00	1257	7.09	1519
city-6	793157	596	0.56	150	13.68G	5.19	8.11	4.48	11.80	3.40	13.68G	0.00	1241	5.81	1552
city-7	1184157	516	0.69	50	222.25G	9.49	19.26	5.70	166.09	52.01	222.25G	0.00	1325	11.73	1403
city-7	1184157	516	0.69	100	33.08G	10.27	16.22	7.95	21.71	9.88	33.08G	0.00	961	9.72	1354
city-7	1184157	516	0.69	150	5.83G	9.43	9.88	7.83	19.02	4.48	5.83G	0.00	894	5.89	1097
city-8	2167197	531	1.29	50	224.64G	15.64	40.47	11.32	355.23	92.58	224.64G	0.00	1286	22.21	2265
city-8	2167197	531	1.29	100	34.16G	19.42	29.30	14.92	39.33	25.23	34.16G	0.00	1032	18.35	1295
city-8	2167197	531	1.29	150	6.07G	15.78	19.42	14.99	43.56	7.32	6.07G	0.04	1106	12.12	1130
city-9	4133801	573	2.12	50	266.62G	36.23	99.13	24.27	788.03	136.24	266.62G	0.00	985	42.63	1561
city-9	4133801	573	2.12	100	47.22G	41.89	78.05	42.38	85.38	59.12	47.22G	0.00	1522	41.57	1629
city-9	4133801	573	2.12	150	9.17G	35.10	49.01	28.41	94.64	15.40	9.17G	0.00	1133	27.23	1344
city-10	2693701	635	1.57	50	357.98G	21.07	49.44	16.83	436.57	106.64	357.98G	0.00	1063	27.77	1862
city-10	2693701	635	1.57	100	82.41G	22.39	42.33	21.32	45.24	32.13	82.41G	0.00	1092	26.72	1790
city-10	2693701	635	1.57	150	17.91G	21.19	34.55	18.86	65.29	12.11	17.91G	0.00	1195	21.29	1501
city-11	4921925	652	2.93	50	361.60G	40.76	124.95	48.40	1004.73	258.64	361.60G	0.00	1480	56.28	1890
city-11	4921925	652	2.93	100	84.01G	43.52	105.65	41.15	95.53	59.17	84.01G	0.00	1101	48.77	1765
city-11	4921925	652	2.93	150	18.17G	47.47	76.60	45.07	117.62	22.93	18.17G	0.00	1184	38.45	1575
occupation-1	875995	357	0.53	50	214.30G	6.01	8.28	4.12	97.68	7.05	214.30G	0.00	1400	8.97	1148
occupation-1	875995	357	0.53	100	26.12G	5.67	5.13	3.58	11.13	2.52	26.12G	0.00	1224	6.15	730
occupation-1	875995	357	0.53	150	5.15G	5.31	3.68	2.82	12.40	2.29	5.15G	0.00	1025	3.10	574
occupation-2	3300827	382	2.01	50	231.22G	26.38	48.80	18.45	606.55	26.83	231.22G	0.00	1180	34.57	1502
occupation-2	3300827	382	2.01	100	30.49G	28.44	24.91	16.30	68.04	12.03	30.49G	0.00	1155	25.54	832
occupation-2	3300827	382	2.01	150	6.66G	24.80	17.28	14.05	86.67	10.65	6.66G	0.00	1010	12.22	656
city_uni-1	1184157	516	0.67	50	1828.53G	9.28	20.49	5.78	155.98	44.14	1828.53G	0.00	1062	11.59	1436
city_uni-1	1184157	516	0.67	100	269.73G	8.33	14.26	7.77	18.92	12.94	269.73G	0.00	963	9.75	1353
city_uni-1	1184157	516	0.67	150	47.47G	8.32	9.41	6.57	22.80	3.81	47.47G	0.00	1096	6.41	1104
city_uni-2	227909	549	0.13	50	1920.07G	1.51	2.39	0.79	18.22	24.25	1920.04G	0.00	1295	2.10	1593
city_uni-2	227909	549	0.13	100	416.04G	1.73	1.98	1.01	3.33	2.31	416.04G	0.00	1283	2.05	1284
city_uni-2	227909	549	0.13	150	92.18G	1.49	1.71	1.05	3.33	1.22	92.18G	0.00	1313	1.66	1236
city_uni-3	647749	584	0.37	50	2109.61G	5.40	8.59	2.75	62.94	20.21	2109.61G	0.00	1087	6.22	1692
city_uni-3	647749	584	0.37	100	469.84G	5.09	7.43	3.42	10.39	7.01	469.84G	0.00	993	5.69	1482
city_uni-3	647749	584	0.37	150	104.77G	3.93	5.79	3.28	12.28	2.66	104.77G	0.00	1100	4.62	1369
occupation_uni	875995	357	0.52	50	1831.35G	6.52	7.95	4.24	100.47	7.04	1831.35G	0.00	1078	8.16	1005
occupation_uni	875995	357	0.52	100	229.59G	5.64	5.06	3.58	12.88	2.60	229.59G	0.00	1085	5.86	740
occupation_uni	875995	357	0.52	150	42.41G	5.32	3.65	2.93	15.14	2.20	42.41G	0.02	1084	3.07	578
Avg			0.76			10.56	18.80	8.57	89.17	21.99		0.00	1126	10.98	1227
Max			2.93			47.47	124.95	48.40	1004.73	258.64		0.04	1522	56.28	2265
s.d.			0.81			12.67	28.05	11.89	188.36	41.51		0.01	203	13.58	461

Table A.6: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type A.

	Size and Generation			**Side Cons.**	**Cplex**						**Dual Ascent**				
Instances	m	n	T_{Gen}	α	z_{LP}	$Time_P$	$Time_D$	$Time_N$	$Time_B$	$Time_S$	z_{LB}	Gap_{LP}	$Iter$	$Time$	$\lvert\mathbb{C}'\rvert$
City480	12883061	536	2.01	100.00G	103.76	86.15	125.58	109.44	2642.47	108.25	103.74	0.02	801	34.99	256212
City480	12883061	536	2.01	10.00G	180.12	45.15	37.67	54.39	854.86	36.78	180.11	0.01	998	19.24	15548
City480	12883061	536	2.01	100.00M	364.51	23.35	23.29	23.18	21.53	26.54	364.46	0.02	1001	12.37	921
City538	7804077	598	1.24	100.00G	120.87	57.73	68.28	70.52	1346.73	52.19	120.85	0.01	877	23.48	160320
City538	7804077	598	1.24	10.00G	210.70	28.29	19.74	25.53	437.33	23.57	210.67	0.01	1219	14.70	8839
City538	7804077	598	1.24	100.00M	418.14	13.49	13.46	13.42	12.48	15.70	418.09	0.01	975	7.27	1035
City552	15144109	612	2.46	100.00G	123.54	126.64	163.83	178.73	3097.50	115.07	123.50	0.03	921	46.13	268111
City552	15144109	612	2.46	10.00G	213.99	53.36	39.84	65.23	1014.29	44.96	213.93	0.03	1052	24.43	15764
City552	15144109	612	2.46	100.00M	426.80	27.37	27.42	27.21	25.18	31.23	426.70	0.02	964	14.22	1087
City604	9116229	668	1.58	100.00G	138.34	71.94	79.42	84.06	1459.44	64.80	138.32	0.01	1035	27.36	93950
City604	9116229	668	1.58	10.00G	242.09	31.12	22.92	28.59	517.27	27.99	242.08	0.01	1124	15.62	9797
City604	9116229	668	1.58	100.00M	474.01	15.87	15.72	15.50	14.31	17.90	473.83	0.04	878	7.94	1159
Avg			1.82			48.37	53.10	57.98	953.62	47.08		0.02	987	20.65	69395
Max			2.46			126.64	163.83	178.73	3097.50	115.07		0.04	1219	46.13	268111
s.d.			0.46			31.98	46.00	46.54	992.81	31.94		0.01	109	10.94	97868

Table A.7: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type B.

	Size and Generation			Side Cons.	Cplex						Dual Ascent				
Instances	m	n	T_{Gen}	α	z_{LP}	$Time_P$	$Time_D$	$Time_N$	$Time_B$	$Time_S$	z_{LB}	Gap_{LP}	$Iter$	$Time$	$\lvert\mathbb{C}'\rvert$
City480	12883061	536	2.01	150	22.87G	129.26	157.15	94.52	338.95	50.08	22.87G	0.00	1049	60.19	1243
City480	12883061	536	2.01	100	115.08G	133.73	205.84	138.84	308.02	89.75	115.08G	0.00	1134	105.14	1505
City480	12883061	536	2.01	50	774.04G	117.62	401.87	157.27	1118.54	346.30	774.03G	0.00	1407	132.76	4553
City538	7804077	598	1.24	150	44.78G	75.90	109.19	60.76	204.63	32.50	44.78G	0.00	1243	53.91	1464
City538	7804077	598	1.24	100	205.76G	76.53	161.41	97.82	191.66	52.54	205.76G	0.00	991	66.55	1565
City538	7804077	598	1.24	50	1097.79G	66.49	193.51	76.42	2083.12	231.52	1097.78G	0.00	1428	79.08	1885
City552	15144109	612	2.46	150	46.98G	148.61	230.62	152.53	439.19	62.73	46.98G	0.00	1266	104.17	1461
City552	15144109	612	2.46	100	217.84G	148.85	425.24	205.82	39.84	92.98	217.84G	0.00	1300	138.94	1695
City552	15144109	612	2.46	50	1112.72G	146.19	516.51	242.31	39.73	596.38	1112.72G	0.00	1323	152.99	1810
City604	9116229	668	1.58	150	74.43G	103.32	178.27	72.72	268.25	37.94	74.43G	0.00	1176	67.34	1805
City604	9116229	668	1.58	100	323.05G	106.40	212.20	126.87	185.49	75.96	323.04G	0.00	1249	83.53	1975
City604	9116229	668	1.58	50	1437.34G	83.26	231.18	53.46	2419.11	246.75	1437.34G	0.00	1123	90.10	1971
Avg			1.82			111.35	251.92	123.28	636.38	159.62		0.00	1224.08	94.56	1911
Max			2.46			148.85	516.51	242.31	2419.11	596.38		0.00	1428.00	152.99	4553
s.d.			0.46			29.20	120.24	56.46	773.18	162.92		0.00	128.76	31.34	826

In Tables A.4, A.5, A.6, and A.7 column $|\mathbb{C}'|$ shows that the size of the subset of columns $\mathbb{C}'$ evaluated in the dual ascent procedure is always very small with respect to the total number of columns n of the original instances.

Notice that the computing time for generating the full set of column $\mathbb{C}$ is usually very small and it never dominates the time for solving the instances.

Appendix A.2. Greedy and Lagrangian Heuristics

In Tables A.8, A.9, A.10, and A.11 we compare the greedy and the Lagrangian heuristics, described in Sections 2 and 5. The results of the Lagrangian heuristic include the dual ascent procedure which embeds it.

For each test instance we report the right-hand side α of the side constraint, which is the maximum loss of precision for problem of Type A and the maximum size of the data for problem of Type B, and for both heuristics we report the best upper bound provided z_{UB}, the percentage gap from the value of the optimal integer solution $Gap = 100 \times \frac{z_{UB} - z_{Opt}}{z_{Opt}}$, and the computing time *Time*. For the dual ascent with the Lagrangian heuristic we also report the best lower bound z_{LB} corresponding to the best feasible dual solution generated, the number of iterations *Iter*, and the size of $\mathbb{C}'$.

Appendix A.3. Exact Method

In order to evaluate the effectiveness of the Lagrangian heuristic, described in Section 5, and of the exact method, described in Section 6, in Tables A.8, A.9, A.10, and A.11 we compare them with the CPLEX MIP solver. For the CPLEX MIP solver we report the integer optimal solution value z_{Opt} and the computing time *Time*. We report the symbol "–" in column z_{Opt} when the CPLEX MIP solver fails because of an "*out of memory*". For these instances column *Time* reports the computing time spent to generate the error.

For the exact method we report the integer optimal solution value z_{Opt}, the gap *Gap* between the best upper bound found by the Lagrangian heuristic and the optimal solution, the size of the subset of columns $\mathbb{C}'$ considered in the integer reduced problem P', the number of iterations *Iter*, the computing time T_A for the Lagrangian heuristic, the computing time T_E for solving the reduced problem P' (even more than one time if $Iter > 1$), and the overall computing time T_{Tot} which also includes the computing time of the dual ascent procedure and of the embedded Lagrangian heuristic. When the problem is solved by the dual ascent we report $|\mathbb{C}'| = 0$.

Table A.8: Comparison among CPLEX MIP solver, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type A.

	Side Con.	Greedy			Cplex		Dual Ascent + Lagr. Heu.						Exact Method						
Instances	α	z_{UB}	*Gap*	*Time*	z_{opt}	*Time*	z_{LB}	z_{UB}	*Gap*	*Iter*	*Time*	$\lvert\mathbb{C}'\rvert$	z_{opt}	*Gap*	$\lvert\mathbb{C}'\rvert$	*Iter*	T_A	T_E	T_{Tot}
state	10.00G	30	0.00	0.00	30	0.01	29.50	30	0.00	3	0.03	34	30	0.00	0	1	0.03	0.00	0.03
state	1000.00M	32	0.00	0.00	32	0.03	32.00	32	0.00	0	0.03	32	32	0.00	0	1	0.03	0.00	0.03
prod_department	1000.00M	5	25.00	0.00	4	0.41	3.50	4	0.00	438	0.06	20	4	0.00	0	1	0.06	0.00	0.06
prod_department	100.00M	10	0.00	0.00	10	0.20	9.05	10	0.00	51	0.03	33	10	0.00	0	1	0.03	0.00	0.03
prod_brand	10.00M	37	8.82	0.01	34	1.49	32.98	34	0.00	897	0.39	398	34	0.00	282	2	0.24	0.34	0.58
prod_brand	1.00M	169	0.60	0.01	168	0.73	167.43	168	0.00	766	0.16	1393	168	0.00	0	1	0.14	0.00	0.14
city-1	100.00G	75	2.74	0.01	73	1.06	72.00	73	0.00	979	0.50	6903	73	0.00	198	1	0.37	0.06	0.43
city-1	10.00G	141	0.00	0.01	141	0.87	140.10	141	0.00	625	0.30	2725	141	0.00	0	1	0.25	0.00	0.25
city-1	100.00M	307	0.00	0.00	307	0.21	306.03	307	0.00	806	0.38	802	307	0.00	0	1	0.19	0.00	0.19
city-2	100.00G	78	2.63	0.01	76	1.99	75.29	76	0.00	713	0.69	19221	76	0.00	0	1	0.75	0.00	0.75
city-2	10.00G	150	0.67	0.01	149	1.48	148.14	149	0.00	662	0.69	4629	149	0.00	0	1	0.71	0.00	0.71
city-2	100.00M	325	0.00	0.01	325	0.31	324.14	325	0.00	663	0.58	930	325	0.00	0	1	0.36	0.00	0.36
city-3	100.00G	80	2.56	0.01	78	4.55	77.53	78	0.00	712	1.51	21669	78	0.00	0	1	1.40	0.00	1.40
city-3	10.00G	155	0.65	0.01	154	3.21	153.88	154	0.00	818	2.90	4367	154	0.00	0	1	1.15	0.00	1.15
city-3	100.00M	343	0.29	0.00	342	0.48	341.29	342	0.00	636	1.19	985	342	0.00	0	1	0.69	0.00	0.69
city-4	100.00G	84	1.20	0.01	83	127.55	82.15	83	0.00	662	2.78	49204	83	0.00	0	1	2.69	0.00	2.69
city-4	10.00G	159	0.00	0.01	159	5.83	158.00	159	0.00	868	7.06	6056	159	0.00	0	1	2.94	0.00	2.94
city-4	100.00M	354	0.00	0.01	354	0.85	353.07	354	0.00	694	2.74	1131	354	0.00	0	1	1.67	0.00	1.67
city-5	100.00G	83	1.22	0.01	82	18.00	81.03	82	0.00	691	4.32	40864	82	0.00	0	1	4.06	0.00	4.06
city-5	10.00G	160	0.63	0.01	159	10.90	158.33	159	0.00	767	6.30	9390	159	0.00	0	1	3.88	0.00	3.88
city-5	100.00M	356	0.00	0.01	356	1.28	355.00	356	0.00	625	3.69	1149	356	0.00	0	1	2.66	0.00	2.66
city-6	100.00G	86	1.18	0.01	85	17.73	84.18	85	0.00	646	5.26	72676	85	0.00	0	1	5.19	0.00	5.19
city-6	10.00G	163	0.62	0.01	162	12.27	161.40	162	0.00	740	8.13	10283	162	0.00	0	1	5.11	0.00	5.11
city-6	100.00M	366	0.27	0.01	365	1.53	364.80	365	0.00	731	7.81	1274	365	0.00	0	1	3.80	0.00	3.80
city-7	100.00G	73	1.39	0.01	72	34.65	71.25	72	0.00	582	8.14	91528	72	0.00	0	1	7.67	0.00	7.67
city-7	10.00G	133	0.00	0.01	133	15.78	132.02	133	0.00	611	10.26	15866	133	0.00	0	1	6.57	0.00	6.57
city-7	100.00M	315	0.32	0.01	314	2.21	313.09	314	0.00	800	10.52	1190	314	0.00	0	1	6.62	0.00	6.62
city-8	100.00G	73	0.00	0.01	73	220.07	72.05	73	0.00	570	14.89	135015	73	0.00	0	1	14.07	0.00	14.07
city-8	10.00G	135	0.75	0.01	134	58.83	133.06	134	0.00	504	15.01	31878	134	0.00	0	1	12.15	0.00	12.15
city-8	100.00M	320	0.31	0.01	319	4.09	318.55	319	0.00	736	22.01	1099	319	0.00	0	1	12.81	0.00	12.81
city-9	100.00G	80	1.27	0.01	79	198.73	78.63	79	0.00	669	30.59	108384	79	0.00	0	1	27.21	0.00	27.21
city-9	10.00G	148	0.68	0.01	147	152.29	146.74	147	0.00	708	44.84	25123	147	0.00	0	1	27.18	0.00	27.18
city-9	100.00M	349	0.29	0.01	348	8.14	347.10	348	0.00	839	49.66	1227	348	0.00	0	1	27.23	0.00	27.23
city-10	100.00G	96	2.13	0.01	94	140.49	93.07	94	0.00	640	21.30	200346	94	0.00	0	1	19.19	0.00	19.19
city-10	10.00G	174	0.58	0.01	173	71.53	172.66	173	0.00	682	31.12	21072	173	0.58	1000	1	24.06	0.59	24.65
city-10	100.00M	401	0.25	0.01	400	5.26	399.03	400	0.00	824	30.29	1335	400	0.00	0	1	17.35	0.00	17.35
city-11	100.00G	96	2.13	0.02	94	295.74	93.60	94	0.00	660	34.27	142600	94	0.00	0	1	30.13	0.00	30.13
city-11	10.00G	175	0.00	0.02	175	239.65	174.00	175	0.00	780	66.57	34057	175	0.00	0	1	33.63	0.00	33.63
city-11	100.00M	407	0.25	0.01	406	10.18	405.03	406	0.00	834	59.26	1570	406	0.00	0	1	36.99	0.00	36.99
occupation-1	100.00G	68	1.49	0.00	67	53.88	66.71	67	0.00	705	7.62	17557	67	0.00	0	1	5.92	0.00	5.92
occupation-1	10.00G	132	1.54	0.00	130	11.40	129.08	130	0.00	800	6.44	1386	130	0.00	0	1	4.82	0.00	4.82
occupation-1	100.00M	260	0.00	0.00	260	1.46	259.07	260	0.00	639	5.30	469	260	0.00	0	1	2.61	0.00	2.61
occupation-2	100.00G	71	1.43	0.01	70	203.92	69.21	70	0.00	672	25.76	24109	70	0.00	0	1	21.74	0.00	21.74
occupation-2	10.00G	138	1.47	0.01	136	51.94	135.17	136	0.00	717	22.59	1523	136	0.00	0	1	18.69	0.00	18.69
occupation-2	100.00M	280	0.00	0.01	280	6.00	279.19	280	0.00	478	15.15	508	280	0.00	0	1	7.25	0.00	7.25
city_uni-1	1000.00G	69	1.47	0.01	68	147.78	67.66	68	0.00	733	9.95	77980	68	1.47	461	1	9.51	0.32	9.83
city_uni-1	100.00G	127	0.00	0.01	127	28.20	126.12	127	0.00	468	7.17	28634	127	0.00	0	1	6.54	0.00	6.54
city_uni-1	1000.00M	304	0.00	0.01	304	2.93	303.04	304	0.00	720	9.52	1205	304	0.00	0	1	5.65	0.00	5.65
city_uni-2	1000.00G	75	2.74	0.01	73	7.71	72.02	73	0.00	629	1.59	41864	73	0.00	0	1	1.62	0.00	1.62
city_uni-2	100.00G	149	0.68	0.01	148	6.93	147.01	148	0.00	658	1.38	4170	148	0.00	0	1	1.13	0.00	1.13
city_uni-2	1000.00M	338	0.00	0.00	338	0.77	337.05	338	0.00	736	1.52	1160	338	0.00	0	1	0.88	0.00	0.88
city_uni-3	1000.00G	77	1.32	0.01	76	303.56	74.97	76	0.00	1001	9.09	50347	76	0.00	313	1	5.53	0.14	5.67
city_uni-3	100.00G	154	1.32	0.01	152	18.65	151.36	152	0.00	601	4.92	7429	152	0.00	0	1	3.46	0.00	3.46
city_uni-3	1000.00M	353	0.00	0.01	353	1.90	352.07	353	0.00	659	4.30	1403	353	0.00	0	1	2.88	0.00	2.88
occupation_uni	1000.00G	65	0.00	0.00	65	58.11	64.04	65	0.00	648	6.47	80112	65	0.00	0	1	5.54	0.00	5.54
occupation_uni	100.00G	125	0.00	0.01	125	11.64	124.04	125	0.00	842	9.37	1323	125	0.00	0	1	4.60	0.00	4.60
occupation_uni	1000.00M	254	0.00	0.00	254	1.47	253.11	254	0.00	595	4.68	458	254	0.00	0	1	2.76	0.00	2.76
Avg			1.28	0.01		45.42			0.00	665.49	11.56	24739		0.04	40		7.94	0.03	7.96
Max			25.00	0.02		303.56			0.00	1001.00	66.57	200346		1.47	1000		36.99	0.59	36.99
s.d.			3.44	0.00		78.56			0.00	188.24	15.00	41163		0.21	153		9.63	0.10	9.65

Table A.9: Comparison among CPLEX, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type B.

	Side Con.	Greedy			Cplex		Dual Ascent + Lagr. Heu.						Exact Method						
Instances	α	z_{UB}	*Gap*	*Time*	z_{opt}	*Time*	z_{LB}	z_{UB}	*Gap*	*Iter*	*Time*	$\lvert\mathbb{C}'\rvert$	z_{opt}	*Gap*	$\lvert\mathbb{C}'\rvert$	*Iter*	T_A	T_E	T_{Tot}
state	10	8295.08G	0.00	0.00	8295.08G	0.14	8295.08G	8295.08G	0.00	689	0.05	13	8295.08G	0.00	0	1	0.05	0.00	0.05
state	15	3593.47G	7.28	0.00	3349.66G	0.14	3274.06G	3349.66G	0.00	1317	0.08	31	3349.66G	0.00	86	3	0.08	0.23	0.30
prod_department	10	65.37M	0.00	0.00	65.37M	1.28	65.37M	65.37M	0.00	581	0.05	14	65.37M	0.00	0	1	0.05	0.00	0.05
prod_department	15	4.70M	0.00	0.00	4.70M	1.27	4.70M	4.70M	0.00	516	0.04	15	4.70M	0.00	0	1	0.04	0.00	0.04
prod_brand	50	4.82M	4.16	0.01	4.63M	7.75	4.63M	4.63M	0.00	710	0.19	1213	4.63M	0.00	0	1	0.13	0.00	0.13
prod_brand	100	1.82M	0.61	0.01	1.81M	7.80	1.81M	1.81M	0.00	720	0.18	1174	1.81M	0.00	204	1	0.10	0.16	0.26
city-1	50	208.50G	4.93	0.01	198.70G	5.29	198.39G	202.73G	2.03	1325	1.37	1120	198.70G	2.03	3303	2	0.71	0.38	1.10
city-1	100	40.91G	3.26	0.01	39.61G	6.48	39.57G	39.67G	0.13	1220	1.08	1072	39.61G	0.13	2949	2	0.54	0.33	0.87
city-1	150	7.53G	0.38	0.01	7.50G	2.84	7.50G	7.50G	0.00	937	0.56	943	7.50G	0.00	178	1	0.40	0.14	0.54
city-2	50	224.77G	4.74	0.01	214.60G	16.83	214.22G	214.60G	0.00	1303	3.58	1295	214.60G	4.43	7975	2	1.55	1.28	2.83
city-2	100	47.63G	4.04	0.01	45.78G	5.30	45.78G	45.78G	0.00	915	1.48	1165	45.78G	0.00	0	1	1.07	0.00	1.07
city-2	150	9.81G	1.67	0.01	9.65G	5.04	9.65G	9.65G	0.00	981	1.31	1199	9.65G	0.00	186	1	0.88	0.14	1.02
city-3	50	237.71G	3.06	0.01	230.64G	9.73	230.64G	230.64G	0.00	1037	7.08	1372	230.64G	0.00	89	1	2.51	0.16	2.67
city-3	100	54.64G	7.85	0.01	50.66G	27.41	50.62G	50.75G	0.17	1473	7.53	1299	50.66G	0.17	124	1	3.36	0.16	3.52
city-3	150	11.46G	2.83	0.01	11.15G	9.13	11.15G	11.15G	0.00	1108	3.84	1140	11.15G	0.00	202	1	2.00	0.17	2.17
city-4	50	264.33G	1.53	0.01	260.35G	34.50	259.88G	264.33G	1.53	1282	26.36	1488	260.35G	1.53	10000	2	5.60	1.81	7.41
city-4	100	58.98G	2.99	0.01	57.27G	50.59	57.25G	57.27G	0.00	1298	19.67	1393	57.27G	2.62	121	1	6.18	0.20	6.38
city-4	150	12.83G	3.50	0.01	12.40G	16.42	12.39G	12.40G	0.00	1018	7.55	1290	12.40G	0.00	185	1	3.44	0.11	3.55
city-5	50	264.10G	2.80	0.01	256.92G	63.50	256.83G	261.96G	1.96	1328	45.31	1584	256.92G	1.96	84	1	9.82	0.14	9.96
city-5	100	59.26G	4.51	0.01	56.70G	23.89	56.70G	56.70G	0.00	1013	14.92	1443	56.70G	0.00	129	1	6.73	0.14	6.87
city-5	150	12.91G	2.77	0.01	12.56G	22.32	12.56G	12.56G	0.00	982	11.37	1270	12.56G	0.00	204	1	5.15	0.14	5.29
city-6	50	287.05G	4.16	0.01	275.59G	92.85	275.43G	280.95G	1.95	1325	55.93	1550	275.59G	2.49	82	1	13.91	0.15	14.06
city-6	100	63.82G	3.57	0.01	61.62G	28.90	61.61G	61.62G	0.00	1068	26.44	1519	61.62G	0.00	129	1	8.80	0.16	8.96
city-6	150	14.14G	3.33	0.01	13.68G	27.68	13.68G	13.68G	0.00	1103	18.71	1552	13.68G	0.00	201	1	7.59	0.25	7.84
city-7	50	233.48G	5.01	0.01	222.35G	141.96	222.25G	222.35G	0.00	1325	73.52	1403	222.35G	1.14	90	1	16.18	0.21	16.40
city-7	100	33.32G	0.73	0.01	33.08G	42.01	33.08G	33.08G	0.00	944	24.34	1354	33.08G	0.00	121	1	11.80	0.22	12.03
city-7	150	5.89G	0.92	0.01	5.83G	36.33	5.83G	5.83G	0.00	894	15.45	1097	5.83G	0.00	0	1	7.01	0.00	7.01
city-8	50	235.51G	4.81	0.01	224.69G	265.35	224.64G	227.22G	1.12	1286	138.98	2265	224.69G	1.12	101	1	27.71	0.38	28.09
city-8	100	34.66G	1.47	0.01	34.16G	81.61	34.16G	34.16G	0.00	930	44.52	1295	34.16G	0.00	121	1	21.98	0.38	22.36
city-8	150	6.15G	1.34	0.01	6.07G	67.88	6.07G	6.07G	0.00	1106	55.72	1130	6.07G	1.34	181	1	19.41	0.39	19.79
city-9	50	289.19G	8.47	0.01	266.62G	224.63	266.60G	266.62G	0.00	878	62.03	1560	266.62G	0.00	85	1	48.71	0.78	49.49
city-9	100	49.74G	5.27	0.01	47.25G	402.68	47.22G	48.88G	3.45	1522	228.60	1629	47.25G	3.45	133	1	91.67	0.74	92.42
city-9	150	9.35G	2.02	0.01	9.17G	222.40	9.17G	9.17G	0.00	931	80.51	1344	9.17G	0.00	192	1	34.19	0.81	35.00
city-10	50	378.33G	5.68	0.01	357.98G	132.51	357.96G	357.98G	0.00	1008	65.18	1862	357.98G	0.00	113	1	35.91	0.51	36.42
city-10	100	87.04G	5.62	0.01	82.41G	132.15	82.41G	82.41G	0.00	1059	71.73	1790	82.41G	0.00	143	1	33.72	0.57	34.29
city-10	150	18.18G	1.50	0.01	17.91G	116.50	17.91G	17.91G	0.00	1055	75.11	1501	17.91G	0.00	197	1	26.54	0.49	27.03
city-11	50	381.75G	5.49	0.02	361.87G	974.37	361.60G	368.64G	1.87	1480	351.22	1890	361.87G	1.87	110	1	162.20	0.97	163.17
city-11	100	88.85G	5.77	0.02	84.01G	313.75	84.00G	84.01G	0.00	1003	83.31	1765	84.01G	0.00	172	1	63.76	1.00	64.77
city-11	150	18.40G	1.25	0.02	18.17G	305.67	18.17G	18.17G	0.00	982	128.10	1575	18.17G	0.00	190	1	48.81	0.90	49.71
occupation-1	50	224.97G	4.95	0.00	214.37G	82.65	214.30G	214.37G	0.00	1400	50.61	1148	214.37G	4.10	69	1	18.59	0.27	18.86
occupation-1	100	26.72G	2.29	0.01	26.12G	28.80	26.12G	26.12G	0.00	1073	19.50	730	26.12G	2.29	136	1	8.05	0.18	8.23
occupation-1	150	5.40G	5.02	0.00	5.15G	27.17	5.15G	5.15G	0.00	957	10.21	574	5.15G	0.00	0	1	4.03	0.00	4.03
occupation-2	50	245.80G	6.30	0.01	231.22G	179.24	231.22G	231.22G	0.00	1180	50.01	1502	231.22G	0.00	0	1	50.07	0.00	50.07
occupation-2	100	32.11G	5.33	0.01	30.49G	137.40	30.49G	30.49G	0.00	1132	34.63	832	30.49G	0.00	132	1	34.53	0.59	35.12
occupation-2	150	6.96G	4.41	0.01	6.66G	149.05	6.66G	6.66G	0.00	952	47.55	656	6.66G	0.00	172	1	15.81	0.61	16.43
city_uni-1	50	1943.69G	6.30	0.01	1828.53G	45.05	1828.38G	1828.53G	0.00	973	26.17	1436	1828.53G	0.00	97	1	13.74	0.23	13.96
city_uni-1	100	275.67G	2.20	0.01	269.73G	42.92	269.73G	269.73G	0.00	963	23.51	1353	269.73G	0.00	0	1	11.43	0.00	11.43
city_uni-1	150	48.36G	1.89	0.01	47.47G	36.40	47.46G	47.47G	0.00	987	19.11	1104	47.47G	0.00	183	1	7.97	0.21	8.18
city_uni-2	50	1993.39G	3.70	0.01	1922.24G	24.69	1920.04G	1952.35G	1.57	1295	10.97	1593	1922.24G	1.57	10000	2	3.17	1.67	4.84
city_uni-2	100	452.58G	8.78	0.01	416.04G	8.99	416.00G	416.04G	0.00	1108	3.00	1284	416.04G	0.00	114	1	2.20	0.08	2.27
city_uni-2	150	96.69G	4.89	0.01	92.18G	8.79	92.17G	92.18G	0.00	1111	2.97	1236	92.18G	0.00	179	1	1.90	0.08	1.98
city_uni-3	50	2196.94G	4.14	0.01	2109.61G	26.22	2109.44G	2109.61G	0.00	1068	24.54	1692	2109.61G	0.00	0	1	8.62	0.00	8.62
city_uni-3	100	501.82G	6.81	0.01	469.84G	24.25	469.81G	469.84G	0.00	921	8.23	1482	469.84G	0.00	130	1	6.31	0.22	6.53
city_uni-3	150	110.29G	5.26	0.01	104.77G	23.40	104.76G	104.77G	0.00	1018	11.62	1369	104.77G	0.00	187	1	5.28	0.15	5.43
occupation_uni	50	1973.03G	7.74	0.00	1831.35G	31.47	1831.33G	1831.35G	0.00	1006	9.67	1005	1831.35G	0.00	72	1	9.65	0.17	9.82
occupation_uni	100	235.46G	2.56	0.00	229.59G	27.77	229.56G	229.59G	0.00	955	13.96	740	229.59G	0.00	120	1	6.58	0.26	6.83
occupation_uni	150	42.88G	1.10	0.00	42.41G	26.74	42.41G	42.41G	0.00	1084	11.91	578	42.41G	0.00	171	1	4.21	0.26	4.48
Avg			3.74	0.01		85.23			0.28	1067	37.39	1227		0.57	704		16.53	0.34	16.88
Max			8.78	0.02		974.37			3.45	1522	351.22	2265		4.43	10000		162.20	1.81	163.17
s.d.			2.23	0.00		148.41			0.71	210	58.51	461		1.10	2119		26.49	0.39	26.65

Table A.10: Comparison among CPLEX MIP solver, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type A.

	Side Con.	Greedy			Cplex		Dual Ascent + Lagr. Heu.						Exact Method						
Instances	α	z_{UB}	Gap	$Time$	z_{opt}	$Time$	z_{LB}	z_{UB}	Gap	$Iter$	$Time$	$\lvert\mathbb{C}'\rvert$	z_{opt}	Gap	$\lvert\mathbb{C}'\rvert$	$Iter$	T_A	T_E	T_{Tot}
City480	100.00G	105	0.96	0.01	–	814.73	103.23	104	0.00	437	54.63	256211	104	0.00	0	1	52.17	0.00	52.17
City480	10.00G	182	0.55	0.01	181	279.56	180.00	181	0.00	796	162.18	15548	181	0.00	0	1	66.73	0.00	66.73
City480	100.00M	365	0.00	0.00	365	24.75	364.01	365	0.00	711	102.61	921	365	0.00	0	1	62.75	0.00	62.75
City538	100.00G	123	1.65	0.01	121	655.10	120.50	121	0.00	544	53.53	160317	121	0.00	0	1	45.09	0.00	45.09
City538	10.00G	213	0.95	0.01	211	305.52	210.14	211	0.00	800	79.94	8839	211	0.00	0	1	50.32	0.00	50.32
City538	100.00M	419	0.00	0.01	419	14.60	418.01	419	0.00	838	142.78	1035	419	0.00	0	1	40.61	0.00	40.61
City552	100.00G	124	0.00	0.01	–	141.08	123.06	124	0.00	606	103.73	268109	124	0.00	0	1	93.66	0.00	93.66
City552	10.00G	216	0.93	0.01	214	386.47	213.65	214	0.00	812	165.95	15764	214	0.00	0	1	109.79	0.00	109.79
City552	100.00M	427	0.00	0.01	427	29.37	426.19	427	0.00	702	130.49	1087	427	0.00	0	1	76.58	0.00	76.58
City604	100.00G	141	1.44	0.01	139	773.06	138.01	139	0.00	679	75.91	93949	139	0.00	0	1	57.45	0.00	57.45
City604	10.00G	244	0.41	0.01	243	525.40	242.02	243	0.00	999	187.87	9797	243	0.00	0	1	72.16	0.00	72.16
City604	100.00M	475	0.00	0.01	475	16.95	473.83	475	0.00	878	218.80	1159	475	0.00	805	1	56.34	1.89	58.23
Avg			0.57	0.01		330.55			0.00	757	65.30	69395		0.00	67		65.30	0.16	65.46
Max			1.65	0.01		814.73			0.00	1046	109.79	268109		0.00	805		109.79	1.89	109.79
s.d.			0.58	0.00		288.41			0.00	176	19.41	97867		0.00	222		19.41	0.52	19.35

Table A.11: Comparison among CPLEX, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type B.

	Side Con.	Greedy			Cplex		Dual Ascent + Lagr. Heu.						Exact Method						
Instances	α	z_{UB}	*Gap*	*Time*	z_{opt}	*Time*	z_{LB}	z_{UB}	*Gap*	*Iter*	*Time*	$\lvert\mathbb{C}'\rvert$	z_{opt}	*Gap*	$\lvert\mathbb{C}'\rvert$	*Iter*	T_A	T_E	T_{Tot}
City480	150	23.17G	1.34	0.01	22.87G	362.18	22.87G	22.87G	0.00	1032	328.82	1243	22.87G	0.00	195	1	88.80	2.61	91.41
City480	100	117.34G	1.96	0.01	115.08G	603.52	115.08G	115.08G	0.00	1000	441.02	1505	115.08G	0.00	130	1	139.75	2.62	142.36
City480	50	836.71G	7.01	0.01	–	383.00	774.03G	836.71G	7.01	1407	301.21	4553	781.88G	7.01	1000000	4	301.36	762.65	1064.02
City538	150	46.36G	3.54	0.01	44.78G	309.50	44.77G	44.78G	0.01	1133	292.71	1464	44.78G	0.00	232	1	81.13	1.54	82.68
City538	100	217.43G	5.67	0.01	205.76G	367.85	205.76G	205.76G	0.00	991	137.85	1565	205.76G	0.00	0	1	82.54	0.00	82.54
City538	50	1194.95G	8.61	0.01	–	434.98	1097.78G	1194.95G	8.61	1428	165.46	1885	1100.19G	8.61	10000	2	165.54	4.91	170.45
City552	150	48.87G	4.03	0.01	–	160.29	46.98G	46.98G	0.00	1214	680.66	1461	46.98G	0.00	239	1	175.00	3.23	178.23
City552	100	229.83G	5.50	0.01	–	159.12	217.82G	217.85G	0.00	1145	285.96	1695	217.84G	0.00	184	1	209.75	3.30	213.04
City552	50	1211.59G	8.66	0.01	–	158.55	1112.72G	1211.59G	8.66	1323	294.25	1810	1115.07G	8.66	10000	2	294.41	9.08	303.49
City604	150	77.20G	3.73	0.01	74.43G	721.90	74.42G	74.43G	0.00	1086	313.80	1805	74.43G	0.00	287	1	98.27	1.84	100.11
City604	100	344.85G	6.75	0.01	323.05G	789.57	323.04G	323.05G	0.00	1047	284.90	1975	323.05G	0.00	160	1	100.35	1.80	102.15
City604	50	1498.44G	4.25	0.01	1437.34G	827.37	1437.34G	1437.34G	0.00	1123	473.94	1971	1437.34G	0.00	0	1	142.52	0.00	142.52
Avg			5.09	0.01		439.82			2.02	1178	156.62	1911		2.02	85119		156.62	66.13	222.75
Max			8.66	0.01		827.37			8.66	1428	301.36	4553		8.66	1000000		301.36	762.65	1064.02
s.d.			2.27	0.00		232.09			3.53	138	74.17	826		3.53	275871		74.17	210.02	261.10

Acknowledgement

The authors would like to thank the anonymous referees for their useful comments and suggestions which contributed to improve this paper.

References

[1] Abbiw-Jackson, R., Golden, B., Raghavan, S., Wasil, E., 2006. A divide-and-conquer local search heuristic for data visualization. Computers & Operations Research 33 (11), 3070–3087.

[2] Balas, E., Carrera, M., 1996. A dynamic subgradient-based branch-and-bound procedure for the set covering. Operations Research 44, 875–890.

[3] Börner, K., 2015. Atlas of knowlegde: anyone can map. MIT Press.

[4] Boschetti, M., Maniezzo, V., 2009. Benders decomposition, Lagrangean relaxation and metaheuristic design. Journal of Heuristics 15, 283–312.

[5] Boschetti, M., Maniezzo, V., 2015. A set covering based matheuristic for a real world city logistics problem. International Transactions in Operational Research 22 (1), 169–195.

[6] Boschetti, M., Maniezzo, V., Roffilli, M., Bolufe Rohler, A., 2009. Matheuristics: Optimization, simulation and control. In: Blesa, M., Blum, C., Di Gaspero, L., Roli, A., Samples, M., A., S. (Eds.), Hybrid Metaheuristics. Vol. 5818 of LNCS. Springer, pp. 171–177.

[7] Boschetti, M., Mingozzi, A., Ricciardelli, S., 2004. An exact algorithm for the simplified multiple depot crew scheduling problem. Annals of Operations Research 127, 177–201.

[8] Boschetti, M., Mingozzi, A., Ricciardelli, S., 2008. A dual ascent procedure for the set partitioning problem. Discrete Optimization 5 (4), 735–747.

[9] Boyd, S., Mutapcic, A., 2008. Subgradient method, Lecture notes for EE364b, Winter 2006-07.

[10] Bradley, P. S., Fayyad, U. M., Mangasarian, O. L., 1999. Mathematical programming for data mining: Formulations and challenges. INFORMS Journal on Computing 11 (3), 217–238.

[11] Caprara, A., Fischetti, M., Toth, P., 2000. Algorithms for the set covering problem. Annals of Operations Research 98, 353–371.

[12] Christofides, N., Mingozzi, A., Toth, P., 1981. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. Mathematical Programming 20, 255–282.

[13] Crainic, T. G., Frangioni, A., Gendron, B., 2001. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. Discrete Applied Mathematics 112 (1), 73–99, combinatorial Optimization Symposium, Selected Papers.
URL http://www.sciencedirect.com/science/article/pii/S0166218X00003103

[14] Domingo-Ferrer, J., Mateo-Sanz, J. M., 2002. Practical data-oriented microaggregation for statistical disclosure control. IEEE Transactions on Knowledge and data Engineering 14 (1), 189–201.

[15] Golfarelli, M., Graziani, S., Rizzi, S., 2014. Shrink: An OLAP operation for balancing precision and size of pivot tables. Data & Knowledge Engineering 93, 19–41.

[16] Han, J., 1997. OLAP mining: Integration of OLAP with data mining. In: Proc. Working Conf. on Database Semantics. Leysin, Switzerland, pp. 3–20.

[17] Hansen, S. L., Mukherjee, S., 2003. A polynomial algorithm for optimal univariate microaggregation. IEEE Transactions on Knowledge and Data Engineering 15 (4), 1043–1044.

[18] Huang, C.-C., Tseng, T.-L. B., Li, M.-Z., Gung, R. R., 2006. Models of multi-dimensional analysis for qualitative data and its application. European Journal of Operational Research 174 (2), 983–1008.

[19] Janssens, D., Brijs, T., Vanhoof, K., Wets, G., 2006. Evaluating the performance of cost-based discretization versus entropy- and error-based discretization. Computers & Operations Research 33 (11), 3107 – 3123.

[20] Keim, D., 2014. Exploring big data using visual analytics. In: Proc. EDBT/ICDT Workshops.

[21] Kros, J. F., Brown, M. L., Lin, M., 2006. Effects of the neural network s-Sigmoid function on KDD in the presence of imprecise data. Computer & Operations Research 33 (11), 3136—-3149.

[22] Łatuszko, M., Pytlak, R., 2015. Methods for solving the mean query execution time minimization problem. European Journal of Operational Research 246, 582—-596.

[23] Lenz, H.-J., Shoshani, A., 1997. Summarizability in olap and statistical data bases. In: Scientific and Statistical Database Management, 1997. Proceedings., Ninth International Conference on. IEEE, pp. 132–143.

[24] Lu, X., Lowenthala, F., 2004. Arranging fact table records in a data warehouse to improve query performance. Computer & Operations Research 31, 2165—-2182.

[25] Mami, I., Bellahsene, Z., Apr. 2012. A survey of view selection methods. SIGMOD Rec. 41 (1), 20–29.

[26] Maniezzo, V., Stützle, T., Voss, S., 2009. Matheuristics: Hybridizing Metaheuristics and Mathematical Programming. Vol. 10 of Annals of Information Systems. Springer.

[27] Marcel, P., Missaoui, R., Rizzi, S., 2012. Towards intensional answers to OLAP queries for analytical sessions. In: Proceedings of the 15th international workshop on DW and OLAP. ACM, pp. 49–56.

[28] Mingozzi, A., Boschetti, M., Bianco, L., Ricciardelli, S., 1999. A set partitioning approach to the crew scheduling problem. Operations Research 47, 883–888.

[29] Minnesota Population Center, 2016. Integrated public use microdata series. http://www.ipums.org.

[30] Oganian, A., Domingo-Ferrer, J., 2001. On the complexity of optimal microaggregation for statistical disclosure control. Statistical Journal of the United Nations Economic Commission for Europe 18 (4), 345–353.

[31] Ólafsson, S., 2006. Introduction to operations research and data mining. Computer & Operations Research 33 (11), 3067–3069.

[32] Ólafsson, S., Li, X., Wu, S., 2008. Operations research and data mining. European Journal of Operational Research 187 (3), 1429—-1448.

[33] Padmanabhan, B., Tuzhilin, A., 2003. On the use of optimization for data mining: Theoretical interactions and eCRM opportunities. Management Science 49 (10), 1327–1343.

[34] Pendharkar, P. C., 2006. A data mining constraint satisfaction optimization problem for cost effective classification. Computer & Operations Research 33 (11), 3124—3135.

[35] Pentaho Corporation, 2016. Pentaho Web Site. www.pentaho.com.

[36] Yu, J. X., Yao, X., Choi, C.-H., Gou, G., 2003. Materialized view selection as constrained evolutionary optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 33 (4), 458–467.