Information Sciences 538 (2020) 444–466

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Bi-objective optimization of biclustering with binary data

Saïd Hanafi [a,*], Gintaras Palubeckis [b], Fred Glover [c]

[a] LAMIH, CNRS UMR 8201, Université Polytechnique des Hauts-de-France, Valenciennes 59313, France
[b] Faculty of Informatics, Kaunas University of Technology, 51368 Kaunas, Lithuania
[c] ECEE- College of Engineering and Applied Science, University of Colorado –Boulder, Boulder, CO 80309, USA

ARTICLE INFO

Article history:
Received 14 February 2020
Received in revised form 21 April 2020
Accepted 20 May 2020
Available online 26 May 2020

Keywords:
Bi-clustering
Bi-objective optimization
Biclique
ε-Constraint method

ABSTRACT

Clustering consists of partitioning data objects into subsets called clusters according to some similarity criteria. This paper addresses a structure for generating overlapping clusters, where data objects can belong to more than one subset, which we join with bi-objective optimization and link to biclustering for problems with binary data. Biclustering simultaneously groups the objects and features so that a specific group of objects has a special group of features. In recent years, biclustering has received a lot of attention in several practical applications. First we present an integer programing formulations for the bi-objective optimization of biclustering. Next we propose a constructive heuristic based on the set intersection operation and its efficient implementation for solving a series of mono-objective problems used inside the ε-constraint method (obtained by keeping only one objective function and the other objective function is integrated into constraints). Finally, our experimental results show that our proposed heuristic provides very good results and significantly reduces the computational expense compared to using the CPLEX solver as an exact algorithm for finding an optimal solution, which drastically increases the computational cost for large instances.

1. Introduction

Clustering is a technique that involves the grouping a set of objects in such a way that objects in the same cluster are more similar (in some sense) to each other than to those in other clusters, exploiting the structure of data without requiring the assumptions common to most statistical approaches. Called unsupervised learning or unsupervised classification, clustering is used for data analysis in many fields, including data mining, machine learning, pattern recognition, image analysis and bioinformatics. Such techniques have been designed for a variety of data types — homogeneous and nonhomogeneous numerical data, categorical data, binary data. This paper addresses an overlapping form of clustering, which join with bi-objective optimization and link to biclustering.

Formally, let an input data set of m objects (samples) and n features (attributes) be given as a rectangular matrix $A = (a_{ij})_{m\times n}$, where the value a_{ij} is the expression of the ith object in the jth feature. Clustering consists of partitioning the data objects into disjoint subsets (clusters) according to some similarity criteria. Biclustering simultaneously groups the objects and features so that a specific group of objects has a special group of features. More precisely, a biclustering technique identifies a subset of objects (rows) that exhibit similar patterns on a subset of features (columns) in an input data matrix A.

* Corresponding author.
E-mail addresses: Said.Hanafi@uphf.fr (S. Hanafi), gintaras.palubeckis@ktu.lt (G. Palubeckis), glover@colorado.edu (F. Glover).

https://doi.org/10.1016/j.ins.2020.05.078
0020-0255/

A variety of names are used to designate the biclustering domain, including co-clustering, bidimensional clustering, two-mode clustering, bimodal cluster analysis, coupled two-way clustering, two-way clustering and direct clustering, among others. The term biclustering was introduced by Mirkin [24] and was notably applied by Cheng and Church [4] to analyze gene expression data, which significantly contributed to the popularization of biclustering techniques. However, the biclustering model can be traced at least as far back as the work of Malgrange [23] and was treated in the work of Hartigan [19].

A biclustering algorithm looks for a set of biclusters such that each bicluster satisfies certain characteristics of homogeneity, giving rise to versions such as: i) biclusters with constant values, ii) biclusters with constant values in columns or rows, iii) biclusters with coherent values, and iv) biclusters with coherent evolutions. (Definitions and examples of these various bicluster types can be found in [22]) Most of the literature on biclusters presents heuristic or exact algorithms for enumerating all maximal biclusters. In the general case, the biclustering problem is NP-hard [4].

Clustering is closely connected to combinatorial optimization and graph theory. In particular, biclustering is related to bipartite graph partitioning [5,37,8,9]. An interesting connection between data matrices and graph theory can be established as follows. The data matrix $A = (a_{ij})$ can be viewed as a weighted bipartite graph $G = (M, N, E)$ where each node $i \in M$ corresponds to a row and each node $j \in N$ corresponds to a column where an edge $(i, j) \in E$ has weight a_{ij}. A biclique of bipartite G is a subgraph of G that is also a complete bipartite graph. A biclique of a bipartite graph therefore corresponds to a bicluster of the associated matrix. Finding a biclique of a maximum number of vertices can be done in polynomial time [12], while finding a biclique of a maximum number of edges is NP-complete [29]. There exist several computationally challenging problems related to bicliques such as enumerating maximal biclique subgraphs and covering the edges of a bipartite graph by bicliques. Other important applications of those problems arise in the context of data compression [1], automata and language theories [11], graphs [10] and partial orders [17].

In recent years, biclustering has proved to be a powerful data analysis technique due to its wide success in various application domains, particularly in microarray and gene expression analysis, computational biology, biomedicine, text mining, pattern discovery, tokens and contexts in natural language processing, data exploration, marketing, web search, collaborative filtering and many other applications. Useful reviews on biclustering techniques and their applications can be found in [3,4,8,9,22,32]. The reader is also referred to a recent survey by Pavlopoulos et al. [28], that additionally covers a broad spectrum of problems on bipartite graphs. In the recent past, several authors have investigated a special case of the bipartite graph partitioning problem where the data matrix A is 0–1 valued and thus represented by an unweighted graph, notably in [6,25,30], and [33]. Analysis of the literature shows that different multiobjective biclustering problems have been considered where the objectives to optimize involve coherence, row variances and bicluster sizes or combinations of these factors. In the literature, coherence is often expressed by a Mean Squared Residue dissimilarity measure and the row variances concern the row fluctuations for the purpose of maximizing the mean row variances. Most models express bicluster size as a mono-objective function with two arguments consisting of the number of rows and the number of columns (see [16,20,31,38], and the references cited herein). Considering this observation, our motivation is to study a bi-objective version of the problem where the goal is to maximize the size of each component of the biclique.

1.1. Design of the paper

In this paper, we consider three new bi-objective optimization of biclustering problems with binary data, i.e. $a_{ij} \in \{0, 1\}$ for all $i \in M$ and $j \in N$. From graph view, the matrix $A = (a_{ij})$ is the biadjacency matrix of the bipartite graph G in which $a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$ if $(i, j) \notin E$. First, we begin by defining the bipartite graph $G^v = (M, N, E^v)$, for $v \in \{0, 1\}$, where $E^v = \{(i, j) : i \in M, j \in N \text{ such that } a_{ij} = v\}$. Note that the graph G^0 is the bipartite graph complement of the graph G^1 and vice-versa. A biclique $B^v(I, J)$ of G^v is a complete bipartite subgraph of G^v induced by $I \subseteq M$ and $J \subseteq N$ such that I and J are nonempty. Trivially, a biclique $B^v(I, J)$ of G^v has $|I| + |J|$ vertices and $|I| \times |J|$ edges. Hence, the two bi-objective biclustering problems, for $v \in \{0, 1\}$, consist in finding a biclique $B^v(I, J)$ of G^v that simultaneously maximizes the sizes of both sets I and J. The first two bi-objective biclustering problems, for $v \in \{0, 1\}$, can be expressed as follows

$$(G_v)\begin{cases} max & g_v = (|I|, |J|) \\ s.t. & B^v(I, J) \text{ is a biclique of } G^v \end{cases}$$

The third bi-objective biclustering problem is a junction of the two first problems G_v for $v \in \{0, 1\}$, i.e. it consists of finding a biclique $B^0\left(I^0, J^0\right)$ of G^0 and a biclique $B^1\left(I^1, J^1\right)$ of G^1 that simultaneously maximize the sizes of both sets $I^0 \cup I^1$ and $J^0 \cup J^1$. This third bi-objective biclustering problems can be expressed as follows

$$(G_{01})\begin{cases} max & g_{01} = \left(\left|I^0 \cup I^1\right|, \left|J^0 \cup J^1\right|\right) \\ s.t. & B^0\left(I^0, J^0\right) \text{ is a biclique of } G^0 \\ . & B^1\left(I^1, J^1\right) \text{ is a biclique of } G^1 \end{cases}$$

The remainder of this paper is organized as follows. In Section 2, we provide basic definitions for bi-objective optimization problems and we present an ε-constraint Method for a bi-objective optimization problem which solves a series of mono-objective problems by keeping only one objective function and the other objective function is integrated into constraints. Section 3 presents an integer programing and biclique formulations for the bi-objective optimization biclustering problem. Section 4 is dedicated to approximate algorithms for solving the mono-objective problems, proposing an effective constructive heuristic and its efficient implementation. Section 5 illustrates the running of the proposed algorithms on a small instance. Advanced considerations are covered in Section 6, including allowance for clusters to overlap. Section 7 presents the generation of the instances and the results of extensive computational experiments. Concluding remarks are given in Section 8.

2. ε-Constraint method

In this section, we recall some basic definitions for a bi-objective optimization problem (see e.g. [7]) and the adaptation of the ε-constraint method, introduced by Haimes et al. [18], for our three bi-objective biclutering problems. We assume that a a bi-objective optimization problem is described as follows

$$(G)\begin{cases} max & g(x) = (g^1(x), g^2(x)) \\ s.t. & x \in X \end{cases}$$

The objective space is defined by $Y = \{(g^1(x), g^2(x)) : x \in X\}$. Since in general, there is no feasible solution which minimizes the two objectives$g^1(x)$ and $g^2(x)$ simultaneously, we search for an acceptable trade-off between them. This compromise is defined by a dominance relation which corresponds to a partial order on the objective space Y.

Definition 1. (*Pareto dominance*). Let g and g' be two solutions in the objective space Y of a bi-objective problem. We say that g dominates g', denoted by $g \succ g'$, if and only if $g^k \geq g'^k$, for k = 1, 2, with at least one inequality being strict.

We remark that the widely-used concept of Pareto optimization, which appeared at the end of the 19th century [27], is useful for finding the compromise solutions in our present context. We apply the common terminology of referring to the set of all non-dominated solutions in the space of objectives as the optimal Pareto front [2].

Definition 2. (*Pareto efficiency*). A solution $x \in X$ is called Pareto efficient, if and only if no solution $x' \in X$ exists such that $g(x) \succ g(x')$. The efficient set is denoted by $E^* = \{x \in X : x \text{ is Pareto efficient}\}$ and the Pareto front is denoted by $F^* = \{g(x) : x \in E^*\}$.

The efficient set E^* and Pareto front F^* contain all the Pareto efficient solutions and all the non-dominated points in the objective space, respectively. In other words, $x^* \in X$ is efficient if there is no other feasible solution $x \in X$ which leads to an improvement in some criterion without simultaneous deterioration in at least one other.

The *Ideal* and *Nadir* points are upper and lower bounds on non-dominated points. These points give an indication of the range of the values which non-dominated points can attain.

Definition 3. (*Ideal and Nadir points*). The point $g^I = (g^{I,1}, g^{I,2})$ with $g^{I,k} = \max\{g^k(x) : x \in X\}$ for k = 1, 2, is called the *Ideal point*. The point $g^N = (g^{N,1}, g^{N,2})$ with $g^{N,k} = \min\{g^k(x) : x \in E^*\}$ for k = 1, 2, is called *Nadir point*.

The determination of the Ideal and Nadir points yields the following consequences:

i) The points $(g^{I,1}, g^{N,2})$ and $(g^{N,1}, g^{I,2})$ are efficient points, i.e. $\{(g^{I,1}, g^{N,2}), (g^{N,1}, g^{I,2})\} \subseteq F^*$.
ii) The evaluation vector $(g^1(x), g^2(x))$ of a solution $x \in E^*$ is bounded as follows:

$$g^{N,1} \leq g^1(x) \leq g^{I,1} \text{ and } g^{N,2} \leq g^2(x) \leq g^{I,2}.$$

The ε-constraint method chooses a single objective to be optimized while the other objective is treated as a constraint. More specifically, the bi-objective optimization problem G is replaced by one of the two parametric problems

$$\left(G^1(\varepsilon^2)\right)\begin{cases} max & g^1(x) \\ s.t. & g^2(x) \geqslant \varepsilon^2 \\ & x \in X \end{cases}$$

and

$$\left(G^2(\varepsilon^1)\right)\begin{cases} max & g^2(x) \\ s.t. & g^1(x) \geqslant \varepsilon^1 \\ & x \in X \end{cases}$$

The two parametric problems $G^1(\varepsilon^2)$ and $G^2(\varepsilon^1)$ can be encapsulated in the following form

$$(G^p(\varepsilon^{3-p}))\begin{cases} max & g^p(x) \\ s.t. & g^{3-p}(x) \geqslant \varepsilon^p \\ & x \in X \end{cases}$$

with $p \in \{1,2\}$.

The ε-constraint method is justified by the following properties:

- For $p \in \{1,2\}$, an optimal solution x^* of $G^p(\varepsilon^{3-p})$, is weakly efficient, i.e. $\nexists x \in X : g^p(x) \geq g^p(x^*)$
- If there exists an unique optimal solution x^* of $G^p(\varepsilon^{3-p})$ for some $p \in \{1,2\}$, then x^* is strictly efficient i.e. $\nexists x \in X : g^k(x) > g^k(x^*)$, for $k = 1,2$ (hence it is efficient).
- A solution $x \in X$ is efficient if and only if there exists $(\varepsilon^1, \varepsilon^2)$ such that x is an optimal solution of $G^p(\varepsilon^{3-p})$ for all $p \in \{1,2\}$.

The extreme point that is computed is then used to determine the bound on the objectives, and this is repeated until there are no new solutions left.

The ε-constraint method is easy to implement, but it requires potentially high computational cost (many runs may be required).

ε-Constraint algorithm for G_v problem

Choose $p \in \{1,2\}$;
Compute the Ideal point $g^I = (g^{I,1}, g^{I,2})$ and the Nadir point$g^N = (g^{N,1}, g^{N,2})$
Set $F = \{(g^{I,p}, g^{N,3-p})\}$ and $\varepsilon^{3-p} = g^{N,3-p} + \varepsilon$, (with$\varepsilon = 1$)
While $\varepsilon^{3-p} \leq g^{I,3-p}$ **do**
  Solve $G^p(\varepsilon^{3-p})$ to obtain an optimal solution x^* with the values$(g^{*,p}, g^{*,3-p})$
  Set $F = F \cup \{(g^{*,p}, g^{*,3-p})\}$ and$\varepsilon^{3-p} = g^{*,3-p} + \varepsilon$
Endwhile
Remove dominated points from F if required.
Return F.

---

3. Integer programing formulation of problems G, $G^1(^2)$ and $G^2(^1)$

For a subset $I \subseteq M$, define the *intersect sets* $N_0(I)$, $N_1(I)$ and $N_{01}(I)$ by

$$N_0(I) = \{j \in N : a_{ij} = 0 \ for \ every \ i \in I\}$$

$$N_1(I) = \{j \in N : a_{ij} = 1 \ for \ every \ i \in I\}$$

$$N_{01}(I) = N_0(I) \cup N_1(I)$$

We can easily observe that for any set $I \subseteq M$, $B^v(I, N_v(I))$ is a biclique of the graph G^v. Hence, we are interested in identifying sets $I \subseteq M$ with maximal size that maximize the size of these intersect sets, i.e., that maximize $|N_0(I)|$ or $|N_1(I)|$ or $|N_{01}(I)|$.

More precisely, for $v \in \{0,1\}$, the goal of identifying sets $I \subseteq M$ that maximize simultaneously $|I|$ the size of the set I and $|N_v(I)|$ the size of intersect set can be expressed as that of solving the following bi-objective optimization problem

$$(G_v)\begin{cases} max & g_v(I) = (|I|, |N_v(I)|) \\ s.t. & I \subseteq M \end{cases}$$

For $v = 01$, the bi-objective biclustering problem can be expressed as follows

$$(G_{01})\begin{cases} \max & g_{01}\left(I^0, I^1\right) = \left(\left|I^0 \cup I^0\right|, \| \ N_v\left(I^0\right) \cup N_v\left(I^1\right) \ \|\right) \\ s.t. & I^0 \subseteq M \\ & I^1 \subseteq M \end{cases}$$

More precisely, for $v \in \{0,1\}$, the goal of identifying sets $I \subseteq M$ that maximize simultaneously $|I|$ the size of the set I and $|N_v(I)|$ the size of intersect set can be expressed as that of solving the following bi-objective optimization problem

$$(G_v)\begin{cases} max & g_v(I) = (|I|, |N_v(I)|) \\ s.t. & I \subseteq M \end{cases}$$

For $v = 01$, the bi-objective biclustering problem can be expressed as follows

$$(G_{01})\begin{cases} max & g_{01}\left(I^0, I^1\right) = \left(\left|I^0 \cup I^1\right|, \left|N_v\left(I^0\right) \cup N_v\left(I^1\right)\right|\right) \\ s.t. & I^0 \subseteq M \\ & I^1 \subseteq M \end{cases}$$

We will call these formulations the *intersection set formulations*. For a given 0–1 vector x , the vector $\bar{x}$ denotes the complemented vector of $\times$ given by $\bar{x}_j = 1 - x_j, j \in N$. For each subset J of N, x_J denotes the subvector of x defined by $x_J = (x_j)_{j\in J}$. A vector of ones of arbitrary dimension will be denoted by e. Hence, we have $\bar{x} = e - x$. Since all data are binary, the intersect sets $N_0(I)$, $N_1(I)$ and $N_{01}(I)$, can be redefined by

$$N_0(I) = \left\{ j \in N : ea_{Ij} = \sum_{i\in I} a_{ij} = 0 \right\}$$

$$N_1(I) = \left\{ j \in N : e\bar{a}_{Ij} = \sum_{i\in I} \bar{a}_{ij} = 0 \right\} = \left\{ j \in N : ea_{Ij} = |I| \right\}$$

$$N_{01}(I) = \left\{ j \in N : ea_{Ij} \times e\bar{a}_{Ij} = 0 \right\}$$

From the definition of intersection sets, we can deduce the following remarks.

Remark 1:. A biclique $B^v(I,J)$ is a *maximal biclique* of G^v if there exists no biclique $B^v(I',J')$ such that, $I \subseteq I'$, $J \subseteq J'$ and $(I',J') \neq (I,J)$. Given a maximal biclique $B^v(I,J)$, it is clear that $J = N_v(I)$, hence $|N_v(I)| = |J|$. Each efficient solution of the problem G_v corresponds to a maximal biclique of a bipartite graph G^v, although a maximal biclique may not correspond to an efficient solution.

Remark 2:. For the bi-objective problem G_v, the coordinates of Ideal and Nadir points can be determined as follows:

$$g_v^{I,1} = \max\{|I| : I \subseteq M\} = |M| = m$$

$$g_v^{I,2} = \max\{|N_v(I)| : I \subseteq M\} = \begin{cases} \max\{|N_v(i)| : i \in M\} & if\ v \in \{0,1\} \\ n & if\ v = 01 \end{cases}$$

$$g_v^{N,1} = \min\left\{|I| : |N_v(I)| = g_v^{I,2}, I \subseteq M\right\} = 1$$

$$g_v^{N,2} = \min\{|N_v(I)| : |I| = M, I \subseteq M\} = |N_v(M)|$$

Remark 3:. We have:

$$\left\{(m, |N_v(M)|), \left(1, g_v^{I,2}\right)\right\} \subseteq F_v^*$$

$$\forall I \in E_v^*, we\ have\ 1 \leq g_v^1(I) \leq m,\ and\ |N_v(M)| \leq g_v^2(I) \leq g_v^{I,2}$$

To formulate the problems G_v for $v \in \{0,1,01\}$ as Integer Programs, we introduce the binary variables y_i for $i \in M$ and z_j^v for $j \in N$, $v \in \{0,1\}$ with the following meaning

$$y_i = \begin{cases} 1 & if\ \ row\ \ i\ \ is\ \ selected,\ \ \ i.e.\ i \in I \\ 0 & \qquad\qquad Otherwise \end{cases}$$

$$z_j^v = \begin{cases} 1 & if\ \ column\ \ j\ \ is\ \ selected,\ \ \ i.e.\ j \in N_v(I) \\ 0 & \qquad\qquad\qquad Otherwise \end{cases}$$

With these binary variables, the cardinalities of a subset $I \subseteq M$ and its intersect set $N_v(I)$ can be expressed as $|I| = \sum_{i\in M} y_i$, and $|N_v(I)| = \sum_{j\in N} z_j^v$.

Hence, the integer program IP^vformulations for problems G_v for $v \in \{0,1,01\}$ can be expressed as

$$\left(IP^0\right)\begin{cases} max & (y_0,z_0)=\left(\sum_{i\in M}y_i,\sum_{j\in N}z_j^0\right)\\ s.t. & \left(1-z_j^0\right)\leqslant\sum_{i\in M}a_{ij}y_i\leqslant m\left(1-z_j^0\right),\quad \forall j\in N\quad (CS-0)\\ & y\in\{0,1\}^m,z^0\in\{0,1\}^n\end{cases}$$

$$\left(IP^1\right)\begin{cases} max & (y_0,z_0)=\left(\sum_{i\in M}y_i,\sum_{j\in N}z_j^1\right)\\ s.t. & \left(1-z_j^1\right)\leqslant\sum_{i\in M}(1-a_{ij})y_i\leqslant m\left(1-z_j^1\right),\quad \forall j\in N\quad (CS-1)\\ & y\in\{0,1\}^m,z^1\in\{0,1\}^n\end{cases}$$

$$\left(IP^{01}\right)\begin{cases} max & (y_0,z_0)=\left(\sum_{i\in M}y_i,\sum_{j\in N}\left(z_j^0+z_j^1\right)\right)\\ s.t. & (CS-0),(CS-1)y\in\{0,1\}^m;z^0,z^1\in\{0,1\}^n\end{cases}$$

The constraint (CS-0) expresses the fact that each component $j\in N$ is a member of the set $N_0(I)$, if and only if $\sum_{i\in M}a_{ij}y_i=0$, hence $z_j^0=1$. Similarly, the constraint (CS-1) expresses the fact that each component $j\in N$ is member the set $N_1(I)$, if and only if $\sum_{i\in M}(1-a_{ij})y_i=0$, hence $z_j^1=1$.

Remark 4:. Let $f_j=\sum_{i\in M}a_{ij}$ and $\bar{f}_j=m-f_j$, hence the constraints $(CS-0)$ and $(CS-1)$ can be strengthened respectively as follows

$$\left(1-z_j^0\right)\leq\sum_{i\in M}a_{ij}y_i\leq f_j\left(1-z_j^0\right)(CS-0')$$

$$\left(1-z_j^1\right)\leq\sum_{i\in M}(1-a_{ij})y_i\leq\bar{f}_j\left(1-z_j^0\right)(CS-1')$$

Let $\overline{A}=(\bar{a}_{ij})=(1-a_{ij})$, the matrix presentation of integer program IP^vformulation $\boldsymbol{G_v}$ can be described as for $v\in\{0,1\}$

$$(IP^v)\begin{cases} max & (y_0,z_0)=(ey,ez^v)\\ s.t. & (e-z^v)\leqslant\left(A+v\left(\bar{A}-A\right)\right)y\leqslant m(e-z^v)\\ & y\in\{0,1\}^m,z^v\in\{0,1\}^n\end{cases}$$

And for $v=01$

$$\left(IP^{01}\right)\left\{max\quad (y_0,z_0)=\left(ey,e(z^0+z^1)\right).s.t.\,(e-z^k)\leq\left(A+k\left(\overline{A}-A\right)\right)y\leq m(e-z^k)k\in\{0,1\}\,.\,y\in\{0,1\}^m,z^k\in\{0,1\}^n\quad k\in\{0,1\}\right.$$

Consequently, the two parametric mono objective optimization problems $G_v^1(\varepsilon^2)$ and $G_v^2(\varepsilon^1)$ solved ε-constraint algorithm for problem G_v are the following:

for $v\in\{0,1\}$

$$\left(G_v^1(\varepsilon^2)\right)\begin{cases} max & y_0=ey\\ s.t. & (e-z^v)\leq\left(A+v\left(\overline{A}-A\right)\right)y\leq m(e-z^v)\\ & ez^v\geq\varepsilon^2\\ & y\in\{0,1\}^m,z^v\in\{0,1\}^n\end{cases}$$

$$\left(G_v^2(\varepsilon^1)\right)\begin{cases} max & z_0=ez^v\\ s.t. & (e-z^v)\leqslant\left(A+v\left(\bar{A}-A\right)\right)y\leqslant m(e-z^v)\\ & ey\geqslant\varepsilon^1\\ & y\in\{0,1\}^m,z^v\in\{0,1\}^n\end{cases}$$

And for $v=01$

$$\left(G^1_{01}(\varepsilon^2)\right)\begin{cases} max & y_0 = ey \\ & (e-z^0) \leqslant Ay \leqslant m(e-z^0) \\ s.t. & (e-z^1) \leqslant \bar{A}y \leqslant m(e-z^1) \\ & e(z^0+z^1) \geqslant \varepsilon^2 \\ & y \in \{0,1\}^m, \quad \bar{z}^0, \bar{z}^1 \in \{0,1\}^n \end{cases}$$

$$\left(G^2_{01}(\varepsilon^1)\right)\begin{cases} max & y_0 = e(z^0+z^1) \\ & (e-z^0) \leqslant Ay \leqslant m(e-z^0) \\ s.t. & (e-z^1) \leqslant \bar{A}y \leqslant m(e-z^1) \\ & ey \geqslant \varepsilon^1 \\ & y \in \{0,1\}^m, \quad z^0, z^1 \in \{0,1\}^n \end{cases}$$

4. Approximate algorithms for problems $G^1(^2)$ and $G^2(^1)$

We approach these problems by a constructive search process which has the same general form for each. It is useful to organize the search for an I that solves these problems by selecting some row index $i^1 = h \in M$ as a "first row" for I. Such an approach limits the possible remaining rows that can belong to S and aids the search for an appropriate I. The strategy of beginning with a first row index i^1 is also motivated by the fact that every I contains at least one $i \in M$, and several different rows $i \in M$ may reasonably give rise to the same I. Therefore, we may usefully discover different sets I by selecting a new i^1 that lies outside sets previously generated to explore farther. This type of strategy is additionally appealing because the sets I generated will typically have different properties and provide a boost to diversification.

Our constructive algorithm successively enlarges a set I (which begins as $I = \left\{i^1\right\}$ for a selected row index $i^1 \in M$) by adding a single row i to create a new set I. The process relies on the following relationship, which is implied by our definitions $N_v(I) = \bigcap_{i \in I} N_v(i)$. This identity motivates the terminology that calls $N_v(I)$ an intersect set.

We restate Problems $G^1_v(\varepsilon^2)$ and $G^2_v(\varepsilon^1)$ by including the stipulation that I contains a given row h in M and using the alternative representation of Section 3. Then we state:

Problem. $\boldsymbol{G^1_v(\varepsilon^2, h)}$ *For a specified* $h \in M$, *solve*

$$\left(G^1_v(\varepsilon^2, h)\right)\begin{cases} max & |I| \\ s.t. & |N_v(I)| \geqslant \varepsilon^2 \\ & h \in I \\ & I \subseteq M \end{cases}$$

Problem. $\boldsymbol{G^2_v(\varepsilon^1, h)}$ *For a* specified $h \in M$, *solve*

$$\left(G^2_v(\varepsilon^1, h)\right)\begin{cases} max & |N_v(I)| \\ s.t. & |I| \geqslant \varepsilon^1 \\ & h \in I \\ & I \subseteq M \end{cases}$$

Let $ov(P)$ denotes the optimal value of an optimization problem P. It is obvious that we have

$$ov\left(G^1_v(\varepsilon^2)\right) = \max\left\{ov\left(G^1_v(\varepsilon^2, h)\right) : h \in M\right\},$$

$$ov\left(G^2_v(\varepsilon^1)\right) = \max\left\{ov\left(G^2_v(\varepsilon^1, h)\right) : h \in M\right\}.$$

4.1. Integer programming formulations for problems $\boldsymbol{G^1_v(\varepsilon^2, h)}$ and $\boldsymbol{G^2_v(\varepsilon^1, h)}$

Straightforward IP formulations of the problems $G^1_v(\varepsilon^2, h)$ and $G^2_v(\varepsilon^1, h)$ can be derived directly from IP^1_v and IP^2_v respectively by setting $y_h = 1$. However, we can also exploit the fact that we know the value of a_{hj} for $j \in N$. Hence, the IP formulations for the problems $G^1_v(\varepsilon^2, h)$ for $v \in \{0,1\}$ can be stated as follows:

$$\left(IP_v^1(\varepsilon^2,h)\right) \quad \begin{cases} max & y_0 = \sum_{i\in M-h} y_i \\ s.t. & \sum_{j\in N_v(h)} z_j^v \geqslant \varepsilon^2 \\ & \left(1-z_j^v\right) \leqslant \sum_{i\in M-h} \left(a_{ij}+v(1-2a_{ij})y_i\right) \leqslant m\left(1-z_j^v\right) \quad j\in N_v(h) \\ & y\in\{0,1\}^{m-1}, z^v\in\{0,1\}^{|N_v(h)|} \end{cases}$$

and for $v = 01$ the IP_{01}^1 formulation becomes

$$\left(IP_v^1(\varepsilon^2,h)\right)\begin{cases} max & y_0 = \sum_{i\in M-h} y_i \\ s.t. & \sum_{j\in N}\left(z_j^0+z_j^1\right) \geqslant \varepsilon^2 \\ & \left(1-z_j^0\right) \leqslant \sum_{i\in M-h} a_{ij}y_i \leqslant m\left(1-z_j^0\right) \quad j\in N_0(h) \\ & \left(1-z_j^1\right) \leqslant \sum_{i\in M-h} (1-a_{ij})y_i \leqslant m\left(1-z_j^1\right) \quad j\in N_1(h) \\ & y\in\{0,1\}^{m-1}, z^v\in\{0,1\}^{|N_v(h)|} \quad v\in\{0,1\} \end{cases}$$

The corresponding IP formulations for the problems $G_v^2(\varepsilon^1,h)$ for $v\in\{0,1\}$ can be stated as:

$$\left(IP_v^2(\varepsilon^1,h)\right)\begin{cases} max & y_0 = \sum_{j\in N_v(h)} z_j^v \\ s.t. & \sum_{i\in M-h} y_i \geqslant \varepsilon^1 - 1 \\ & \left(1-z_j^v\right) \leqslant \sum_{i\in M-h} \left(a_{ij}+v(1-2a_{ij})y_i\right) \leqslant m\left(1-z_j^v\right) \quad j\in N_v(h) \\ & y\in\{0,1\}^{m-1}, z^v\in\{0,1\}^{|N_v(h)|} \end{cases}$$

and for $v = 01$ the IP_{01}^2 formulation becomes

$$\left(IP_{01}^2(\varepsilon^1,h)\right)\begin{cases} max & y_0 = \sum_{j\in N}\left(z_j^0+z_j^1\right) \\ s.t. & \sum_{i\in M-h} y_i \geqslant \varepsilon^1 - 1 \\ & \left(1-z_j^0\right) \leqslant \sum_{i\in M-h} a_{ij}y_i \leqslant m\left(1-z_j^0\right) \quad j\in N_0(h) \\ & \left(1-z_j^1\right) \leqslant \sum_{i\in M-h} (1-a_{ij})y_i \leqslant m\left(1-z_j^1\right) \quad j\in N_1(h) \\ & y\in\{0,1\}^{m-1}, z^v\in\{0,1\}^{|N_v(h)|} \quad v\in\{0,1\} \end{cases}$$

Remark 5:. We identify several properties that can be useful for preprocessing or that can provide valid inequalities for the proposed integer programs IP^v, IP_v^1 and IP_v^2:

- For $v\in\{0,01\}$, if $f_j = 0$ then $j\in N_v(I)$ for all $I\subseteq M$. Hence the variable z_j^0 can be fixed to 1 in the IP^v, IP_v^1 and IP_v^2.
- For $v\in\{1,01\}$, if $f_j = m$, i.e. $\overline{f}_j = 0$, implies $j\in N_v(I)$ for all $I\subseteq M$. Hence the variable z_j^1 can be fixed to 1 in the IP^v, IP_v^1 and IP_v^2.
- For $v = 0$, each variable z_j^0 can be fixed to 0 for $j\in N_1(h)$ in the $IP_0^1(\varepsilon^2,h)$.
- For $v = 1$, each variable z_j^1 can be fixed to 0 for $j\in N_0(h)$ in the $IP_1^1(\varepsilon^2,h)$.
- For $v\in\{0,01\}$, let I be a feasible solution for $IP_0^1(\varepsilon^2,h)$, (i.e. $|N_v(I)|\geq\varepsilon^2$), if $j\in N_0(h)$ and $\overline{f}_j < |I|$ then the variable z_j^0 can be fixed to 0.
- For $v\in\{1,01\}$, let I be a feasible solution for $IP_1^1(\varepsilon^2,h)$, (i.e. $|N_v(I)|\geq\varepsilon^2$), if $j\in N_1(h)$ and $f_j < |I|$ then the variable z_j^0 can be fixed to 0.
- In $IP_v^1(\varepsilon^2,h)$, the binary variable y_i can be fixed to 0 if

$$\begin{cases} |N_v(h)\cap N^=(h,i)| < \varepsilon^2 & for\ \ v\in\{0,1\} \\ |N^=(h,i)| < \varepsilon^2 & for\ \ v = 01 \end{cases}$$

where $N^{=}(h,i) = \{j \in N : a_{hj} = a_{ij}\}$.

- For $v \in \{0,1\}, j \in N_v(I)$ implies $|I| \leq \overline{f}_j + v\left(m - 2\overline{f}_j\right)$.
- For $v = 01, j \in N_v(I)$ implies $|I| \leq \max\left(f_j, \overline{f}_j\right)$.

4.2. Heuristics for problems $G^1_v(\varepsilon^2, h)$ and $G^2_v(\varepsilon^1, h)$

The heuristics described in this section for problems $G^1_v(\varepsilon^2, h)$ and $G^2_v(\varepsilon^1, h)$ provide an additional option of being used in conjunction with the ε-Constraint Algorithm for problem G_v. The 'while' loop in the ε-Constraint Algorithm applies an exact algorithm for finding an optimal solution which can drastically increase the computational cost for large n. Our proposed constructive techniques can be used to replace this exact algorithm to significantly reduce the computational expense.

4.2.1. An alternative representation

Let R denote the set of rows (vectors) of the matrix A i.e. $R = \{A_i = (a_{i1}, \cdots, a_{in}) : i \in M\}$. For $S \subseteq R$, the intersect sets $N_0(S)$, $N_1(S)$ and $N_{01}(S)$ are defined as follows

$$N_0(S) = \{j \in N : x_j = 0 \ \textit{for every} \ x \in S\}$$

$$N_1(S) = \{j \in N : x_j = 1 \ \textit{for every} \ x \in S\}$$

$$N_{01}(S) = N_0(S) \cup N_1(S).$$

To visualize the constructive process subsequently described, it is convenient to define an operation $\cap_o$ on vectors $x \in S$ that gives outcomes equivalent to the set intersection operation $\cap$ on the sets $N_v(x)$. The components x_j of the x vectors that are operated on by $\cap_o$ can take a value # in addition to the values 0 and 1, where $x_j = \#$ indicates that x_j is irrelevant to defining the intersection. The vector $x = x' \cap_o x''$ generated from two vectors x' and x'' can be identified by reference to a corresponding operation, designated by the same symbol $\cap_o$, which is carried out on each component of x' and x''; that is $x_j = x'_j \cap_o x''_j, j \in N$. This operation is made precise by the following rules:

$$x_j = \begin{cases} 0 & \textit{if } x'_j = x''_j = 0 \\ 1 & \textit{if } x'_j = x''_j = 1 \\ \# & \textit{otherwise}, \text{hence if } x'_j \neq x''_j \text{ or if } x'_j = \# \text{ or } x''_j = \# \end{cases}$$

This operation be also coded simply as follows

$$x_j = \begin{cases} x'_j & \textit{if } x'_j = x''_j \\ \# & \textit{otherwise} \end{cases}$$

Note that if the value # is coded by a fractional real number $\alpha \in]0,1[$, then x_j can be expressed simply as $x_j = \frac{x'_j + x''_j}{2}$. By implication $\cap_o$ is commutative and associative, i.e., for values $t, u, w \in \{0, 1, \#\}$

$t \cap_o u = u \cap_o t$ and $t \cap_o (u \cap_o w) = (t \cap_o u) \cap_o w$.

These same relationships hold when t, u and w are vectors, using the vector form of $\cap_o$. By these definitions we can write $\cap_o(S) = \cap_o(x : x \in S)$. By convention, when S consists of a single vector x, we define $\cap_o(S) = \cap_o(x)$.

Construction Example. Consider the case where S consists of the following four vectors: Table 1.1

Then we can generate a vector $z = \cap_o(S)$ in four successive stages to identify vectors z^1, z^2, z^3 and z^4, as shown in Table 1.2, where the final vector z^4 is the vector $z = \cap_o(S)$. In sequence, these stages yield $z^1 = A_1$, $z^2 = z^1 \cap_o A_2$, $z^3 = z^2 \cap_o A_3$) and finally $z^4 = z^3 \cap_o A_4$.

We observe that once z_j receives the value # in one of these vectors z^k, it continues to receive this value in all remaining vectors for larger values of k.

Table 1.1
An example of four vectors.

j	1	2	3	4	5	6	7	8	9	10	11
A_1	0	1	0	0	1	1	1	0	1	0	0
A_2	1	1	0	0	1	1	1	1	0	0	0
A_3	0	1	0	1	1	1	0	1	1	0	0
A_4	1	1	0	1	0	1	0	1	1	1	0

Table 1.2
A vector $z = \cap_o(S)$ of four vectors.

J	1	2	3	4	5	6	7	8	9	10	11
z^1	0	1	0	0	1	1	1	0	1	0	0
z^2	#	1	0	0	1	1	1	#	#	0	0
z^3	#	1	0	#	1	1	#	#	#	0	0
z^4	#	1	0	#	#	1	#	#	#	#	0

We now complete the connection with the intersect set $N_v(S)$ defined earlier. We call $z = \cap_o(S)$ the *vector analog* of S and use it as an easy way to identify $N_v(S)$ by reference to the values z_j taken by the components of z.

$$N_0(S) = \{j \in N : z_j = 0\}, N_1(S) = \{j \in N : z_j = 1\}$$

$$N_{01}(S) = \{j \in N : z_j = 0 or z_j = 1\} = \{j \in N : z_j \neq \#\} (= N_0(S) \cup N_1(S)).$$

Thus, a glance at the 0 and 1 components of z^4 in the Construction Example above shows that $N_0(S) = \{3, 11\}$ and $N_1(S) = \{2,6\}$. (In other words, these two sets of indexes respectively identify the variables z_j such that $z_j = 0$ and such that $z_j = 1$ in every solution z in S). Then $N_{01}(S) = \{2, 3, 6, 11\}$.

To identify the associated cardinalities of these sets, define

$|z|_v = |\{j \in N : z_j = v\}|$ for $v \in \{0, 1\}$ and $|z|_v = |z|_0 + |z|_1$ for $v = 01$

and thus obtain $|N_v(S)| = |z|_v$ for $v = 0, 1$ and 01.

These relationships are used to characterize our algorithms for Problems $G^1_v(\varepsilon^2, h)$ and $G^2_v(\varepsilon^1, h)$, and thus to identify useful consistency information in the set of solutions R.

4.2.2. Constructive algorithms for problems $\boldsymbol{G^1_v(\varepsilon^2, h)}$ and $\boldsymbol{G^2_v(\varepsilon^1, h)}$

We restate Problems $G^1_v(\varepsilon^2, h)$ and $G^2_v(\varepsilon^1, h)$ by including the stipulation that S contains a given row A_h in R, and using the alternative representation of Section 3, as follows.

Problem. $\boldsymbol{G^1_v(\varepsilon^2, h)}$: *For a specified $A_h \in R$, find a set S with analog vector $z = \cap_o(S)$ to*

$$\left(G^1_v(\varepsilon^2, h)\right) \begin{cases} max & |S| \\ s.t. & |z|_v \geqslant \varepsilon^2 \\ & A_h \in S \\ & S \subseteq R \end{cases}$$

Problem. $\boldsymbol{G^2_v(\varepsilon^1, h)}$: *For a specified $A_h \in R$, find a set S with analog vector $z = \cap_o(S)$ to*

$$\left(G^2_v(\varepsilon^1, h)\right) \begin{cases} max & |z|_v \\ s.t. & |S| \geqslant \varepsilon^1 \\ & A_h \in S \\ & S \subseteq R \end{cases}$$

We find an approximate vector to $G^1_v(\varepsilon^2, h)$ by identifying a maximal S in R satisfying the stated conditions and find an approximate vector to $G^2_v(\varepsilon^1, h)$ in a similar manner. Recall $|z|_v$ = the number of components of S with $z_j = v$, interpreting v = 01 to mean v = 0 or 1.

These algorithms rely on the principle that increasing the size of S can only decrease the value $|z|_v$ or leave it unchanged.

Algorithm 1: For problem $\boldsymbol{G^1_v(\varepsilon^2, h)}$ (Maximize $|\boldsymbol{S}|$: For $\boldsymbol{A_h \in S}$ and $\boldsymbol{\varepsilon^2 \leq |z|_v}$)

Let $z^k = \cap_o(S)$ denotes the analog vector of the current set S at iteration k. (Note that $k = |S|$ since a single solution is added to S at each iteration.)
Choose $A_h \in R$; $S = \{A_h\}$; $k = 1$; $z^1 = A_h$; Done = False;
While Done = False and $S \neq R$ **do**
  Select a row $r \in R_{\S}$ that maximizes $|z|_v$ for $z = z^k \cap_o r$, and denote the chosen row by r^*. Hence $r^* = argmax\left\{\left|z^k \cap_o r\right|_v : r \in R_{\S}\right\}$;
  If $\left|z^k \cap_o r^*\right|_v \geq \varepsilon^2$ **then** set $S = S \cup \{r^*\}$, $z^{k+1} = z^k \cap_o r^*$, $k = k + 1$;
  Else the method stops with the current S, Done = True;
Endwhile

Algorithm 2: For problem $\boldsymbol{G_v^2(\varepsilon^1, h)}$ (Maximize $|\boldsymbol{z}|_{\boldsymbol{v}}$: For $\boldsymbol{A_h} \in \boldsymbol{S}$ and $\boldsymbol{\varepsilon^1} \leq |\boldsymbol{S}|$)

At iteration $k = 1, 2, \cdots$, let $z^k = \cap_o(S)$ denote the analog vector of the current set S, where initially choose $A_h \in R$;
$S = \{A_h\}$; $k = 1$; $z^1 = A_h$; Done = False;
While Done = False and $S \neq R$ **do**
Select a row $r \in R_{\S}$ that maximizes $|z|_v$ for $z = z^k \cap_o r$, and denote the chosen row
by r^*. Hence $r^* = argmax\left\{|z^k \cap_o r|_v : r \in R_{\S}^{\S}\right\}$;
Set $S = S \cup \{r^*\}$, $z^{k+1} = z^k \cap_o r^*$, $k = k + 1$;
If $\varepsilon^1 = |S| (= k + 1)$ **stop** with the current S, Done = True;
Endwhile

Note this Algorithm outline is the same as for Problem $G_v^1(\varepsilon^2, h)$ except for the last few instructions before "Continue next iteration." Specifically, the instructions

If $|z^k \cap_o r^*|_v \geq \varepsilon^2$ **then** set $S = S \cup \{r^*\}$, $z^{k+1} = z^k \cap_o r^*$, $k = k + 1$;
Else (1) is violated and the method stops with the current S, Done = True
are replaced by the instructions
Set $S = S \cup \{r^*\}$, $z^{k+1} = z^k \cap_o r^*$, $k = k + 1$;
If $\varepsilon^1 = |S| (= k + 1)$ **stop** with the current S, Done = True;

4.2.3. Efficient implementation of constructive algorithms

In this section, we describe an efficient implementation of constructive Algorithms 1 and 2 for problems G_v^1 and G_v^2 using set notation and the vector notation. The efficiency of these algorithms is based on the following observations.

Observation 1: Once z_j receives the value # at iteration k, it continues to receive this value in all iterations with larger values of k. To exploit this, it is not necessary to compute the intersection vectors $z \cap_0 r$ and $z \cap_0 r^*$ for all components $j \in N$ but just over the set $N^* = \{j \in N : z_j \neq \#\} = \{j \in N : z_j \in \{0, 1\}\}$.

The consequences of this observation are exploited in the determination of $r^* = argmax\left\{|z^k \cap_o r|_v : r \in R_{\S}^{\S}\right\}$ and also in the update step.

Observation 2: Let S' and S'' be two subsets in R. Then

$$|N_v\left(S' \cup S''\right)| \leq \min\left(\left|N_v\left(S'\right)\right|, |N_v\left(S''\right)|\right)$$

and

$$\text{if } S' \subseteq S'' \text{ or } S'' \subseteq S' \text{ then} \left|N_v\left(S' \cup S''\right)\right| = \min\left(\left|N_v\left(S'\right)\right|, |N_v\left(S''\right)|\right)$$

In our constructive algorithm we will observe this relationship for the case where we want to enlarge a set S by adding a row r to it to create the set $S \cup \{r\}$. Then

$$|N_v(S \cup \{r\})| \leq \min(|N_v(S)|, |N_v(\{r\})|)$$

The consequences of this second observation are exploited in the initialization phase by excluding rows $r \in R$ such that $|r|_v < \varepsilon^2$ for $v \in \{0, 1\}$ and also in the While loop, if $r = A_h$ is not feasible for Problem $G_v^1(\varepsilon^2, h)$ and will never be feasible on later. In the pseudo-code, the set F is used to save the excluded rows.

In summary, the method goes through all $r \in Rleft(S \cup F)$ and picks the one r^* that gives the largest n_v^* value for $n_v = N_v(S \cup r)$. This r^* is added to S if $n_v^* \geq \varepsilon^2$, and the method terminates otherwise.

Accelerated Algorithm 1: For problem $\boldsymbol{G_v^1(\varepsilon^2, h)}$ (Maximize $|\boldsymbol{S}|$: For $\boldsymbol{A_h} \in \boldsymbol{S}$ and $\boldsymbol{\varepsilon^2} \leq |\boldsymbol{z}|_{\boldsymbol{v}}$)

Choose $A_h \in R$; $S = \{A_h\}$; $F = \varnothing$; $z = A_h$; $N^* = N_v(z)$, Done = False;
For each $r \in Rleft(S \cup F)$ **do %** This need only be executed for $v \in \{0, 1\}$
If $|r|_v < \varepsilon^2$ **then** $F = F \cup \{r\}$
Endfor
While Done = False and $S \cup F \neq R$ and $N* \neq$ do
$n^*_v = -1$;
For each $r \in Rleft(S \cup F)$ **do**

$n_v = |\{j \in N^* : z_j \cap_0 r_j = v\}|$
If $n_v > n_v^*$ **then** $n_v^* = n_v$, $r^* = r$
If $n_v < \varepsilon^2$ **then** $F = F \cup \{r\}$
Endfor
% Execute Update/Terminate Routine for Problem $G_v^1(\varepsilon^2, h)$
If $n_v^* \geq \varepsilon^2$ **then**
$S = S \cup \{r^*\}$
For $j \in N^*$ **do** $z_j = z_j \cap_0 r_j^*$
$N^* = N^* left\{j \in N^* : z_j = \#\}$
Else
Done = True
Endif
Endwhile

The detailed algorithm for Problem $G_v^2(\varepsilon^1, h)$ employs a corresponding design, which can be inferred from the above and the description of the $G_v^2(\varepsilon^1, h)$ algorithm in Section 4.2.2.

The supplementary document [14] shows how the foregoing algorithms can be equivalently expressed by reference to notation involving the index sets $N_v(x)$ in place of the vectors z_v and gives a more fully detailed version of Algorithm 1 designed for greater efficiency.

Next, we describe advanced considerations for these algorithms and then in Section 6 give a numerical example of applying these algorithms. The following illustration discloses additional advanced considerations that become apparent by numerical example.

5. Illustration

5.1. Algorithm 1 for Problem G_v^1

We illustrate Algorithm 1 for Problem G_v^1 by reference to a set R containing 12 rows, labeled A_1 to A_{12}. These rows are organized in the Algorithm Example below so that the sets S derived from each of the three objectives associated with $v = 0, 1$ and 01 can be illustrated simultaneously in the table for this example. Later comments show how the example can be applied to Algorithm 2, and to an adaptive method that may be viewed as an extension of both Algorithms 1 and 2.

It should be observed that our following illustration does not show the z vectors corresponding to the successive sets S generated for a given v value. (Consequently, no "# components" appear in the vectors.) However, the z vectors can readily be inferred by inspection based on our associated commentary and the fact that the size of S for each case is small. This has an advantage of allowing the outcomes for different choices of v to be considered in a single table. Specifically, the $|z|_v$ values listed immediately to the right of the rows include all three cases for $v = 0, 1$ and 01. For emphasis, the entries for $|z|_v$ values that correspond to the goal of maximizing $|S|$ in each case are shown in bold face.

The Algorithm Example shown here is extended in in the supplementary document [14] to identify the z vectors for the cases $v = 0$ and $v = 1$, starting from the point where Table 2.1 in the Algorithm Example leaves off. The seed rows for $v = 0, 1$ and 01 for this example are given by $A_h = A_1$, A_4 and A_6 respectively. The choices for A_h correspond to those given in the Master Algorithm of Section 6. Thus $A_h = A_1$ for $v = 0$ because A_1 is the row containing the most components equal to 0, i.e. $|N_0(A_1)| = \max\{|N_0(A_i)| : i \in M\}$, and similarly $A_h = A_4$ for $v = 1$ because A_4 is the row containing the most components equal to 1, i.e. $|N_1(A_4)| = \max\{|N_1(A_i)| : i \in M\}$. Finally, $A_h = A_6$ for $v = 01$ because that the number of components equal to 0 and 1 are more nearly balanced in A_6, i.e. $|N_{01}(A_6)| = \max\{|N_{01}(A_i)| : i \in M\}$.

We first consider the case for $v = 0$ which appears first in the Algorithm Example. Here, S is built in three iterations starting with $S = \{A_1\}$ (since $A_h = A_1$). This yields a value for $|z|_0$ of 8 as shown in bold to the right of A_1 (in the column under $v = 0$). The value $|z|_0 = 8$ for $S = \{A_1\}$ is confirmed by counting the number of 0 s in A_1. The next iteration adds A_2 to S (as shown under "$S =$"), because A_2 is the row that maximizes $|z|_0$ when added to $S = \{A_1\}$. This yields a value for $|z|_0$ of 6, as confirmed by counting the number of components equal to 0 in both A_1 and A_2 (and as listed in the column under $v = 0$ for A_2). We note that it is easy to identify which row yields the largest $|z|_0$ value when added to S.

Finally, the third S produced for $v = 0$ in the Algorithm Example shows that A_3 is the best row to join the set $S = \{A_1, A_2\}$ when $v = 0$, by yielding $|z|_0 = 5$. This $|z|_0$ value may be similarly confirmed by counting the components equal to 0 in all three solutions A_1, A_2 and A_3. Since the lower bound $\varepsilon_0^2 = 5$, the only way to expand S further would be to find a row to add to $S = \{A_1, A_2, A_3\}$ that again yields $|z|_0 = 5$. However, the best solution to add to the current S yields $|z|_0 = 3$, and consequently the construction of S for $v = 0$ ends here.

Algorithm Example: Lower bounds $\varepsilon_0^2 = 5$, $\varepsilon_1^2 = 4$, $\varepsilon_{01}^2 = 9$.

Table 2.1
The algorithm example.

$j=$	Rows in R												$\|z\|_v = \|N_v(S)\|$ for $v=$			$S=$
	1	2	3	4	5	6	7	8	9	10	11	12	0	1	01	
A_1	1	1	0	1	0	0	0	0	1	0	0	0	**8**	4	12	$\{A_1\}$
A_2	0	0	1	1	1	0	0	0	1	0	0	0	**6**	2	8	$\{A_1, A_2\}$
A_3	1	1	1	0	1	0	1	0	1	0	0	0	**5**	1	6	$\{A_1, A_2, A_3\}$
Above generates S for $v=0$ (with $\|S\|=3$)																
A_4	1	1	1	1	1	1	1	1	0	0	0	0	4	**8**	12	$\{A_4\}$
A_5	1	1	1	0	1	1	1	0	0	0	1	0	3	**6**	9	$\{A_4, A_5\}$
$A_3{}^*$	1	1	1	0	1	0	1	0	1	0	0	0	2	**5**	7	$\{A_4, A_5, A_3\}$
Above generates S for $v=1$ (with $\|S\|=3$)																
A_6	0	1	0	1	0	1	0	1	0	1	0	1	6	6	**12**	$\{A_6\}$
A_7	0	1	0	1	0	1	0	1	0	0	1	1	5	5	**10**	$\{A_6, A_7\}$
A_8	0	1	0	0	0	1	0	1	0	1	1	1	5	4	**9**	$\{A_6, A_7, A_8\}$
Above generates S for $v=01$ (with $\|S\|=3$)																
A_9	0	0	0	0	1	0	1	1	1	1	1	1	5	**7**	12	$\{A_9\}$
A_{10}	0	1	0	0	0	0	1	1	1	1	1	1	4	**6**	10	$\{A_9, A_{10}\}$
A_{11}	1	0	0	0	0	0	1	1	1	1	0	1	3	**5**	8	$\{A_9, A_{10}, A_{11}\}$
A_{12}	0	0	0	0	1	0	1	1	1	1	0	1	3	**5**	8	$\{A_9, A_{10}, A_{11}, A_{12}\}$
Above generates *different* S for $v=1$ (with $\|S\|=4$)																

We next consider the case where $v=1$ and $A_h=A_4$. A similar progression ensues by adding first A_5 and then A_3, to yield a succession of sets $S=\{A_4\}$, $S=\{A_4,A_5\}$ and $S=\{A_4,A_5,A_3\}$ with corresponding $|z|_1$ values given by $|z|_1=8,6$ and 5. (By chance, this is the same succession of values produced for $|z|_0$ in the $v=0$ case previously illustrated.) The row A_3 is duplicated in the Algorithm Example (and marked the second time with "*") to allow the basis for adding A_3 as the third solution for the $v=1$ case to be seen more easily. Here we stop after adding A_3 with $|z|_1=5$. Although the lower bound for $v=1$ is $\varepsilon_1^2=4$, we observe that the next row to be added to S yields $|z|_1=2$.

5.2. ε-constraint method

We show in Table 2.2 the execution of ε-Constraint algorithm for G_v problem for $p=1$ on the example described in Section 5.1 with $m=n=12$.

6. Advanced considerations

The determination of the set S by Algorithms 1 and 2 for problems G_v^1 and G_v^2 can be modified in several ways. First, we consider how these algorithms can be extended by applying them to different choices of the starting row A_h to generate more than one set S.

6.1. The choice of the starting row

One possibility for generating multiple sets S from different starting rows A_h is to select a new A_h to be the last solution added to create a previous S. Although ideally it should be possible to generate the same S by starting from any of its component rows, this may not happen because Algorithms 1 and 2 are approximation methods. Starting from the last row added to S gives an increased chance to discover variations caused by the inexact nature of these algorithms. (Such a process could be continued, for example, until a final S duplicates a previous S, not necessarily the immediate predecessor.) However, to avoid unnecessary generation of sets that may not differ substantially from each other, it seems worthwhile to select new rows A_h that lie outside sets previously generated. We formalize this in the following algorithm that employs the algorithms for problems G_v^1 and G_v^2 as subroutines.

We make use of an upper limit U on the number n_S of sets S generated and a lower limit L on the size of S so that when either n_S reaches U or a set S generated fails to satisfy $|S|\geq L$ the generation of new sets S terminates. (L has a function similar to that of ε_v^1 in Algorithm 2 for G_v^2, in this case functioning as an ultimate limit.)

Table 2.2
The execution of ε-Constraint algorithm.

Iter	ε_v^1	$\lvert S^*\rvert$	$\lvert N_v(S^*)\rvert$	S^*	$N_v(S^*)$
$v=0, \lvert F\rvert=7$					
1	0	12	0	R	$\varnothing$
2	1	8	1	$\{A_1, A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{12}\}$	{3}
3	2	6	2	$\{A_1, A_6, A_7, A_8, A_{10}, A_{11}\}$	{3, 5}
4	3	4	3	$\{A_1, A_2, A_3, A_4\}$	{10, 11, 12}
5	4	3	5	$\{A_1, A_2, A_3\}$	{6, 8, 10, 11, 12}
6	6	2	6	$\{A_1, A_2\}$	{6, 7, 8, 10, 11, 12}
7	7	1	8	$\{A_1\}$	{3, 5, 6, 7, 8, 10, 11, 12}
$v=1, \lvert F\rvert=8$					
1	0	12	0	R	$\varnothing$
2	1	8	1	$\{A_1, A_3, A_4, A_5, A_6, A_7, A_8, A_{10}\}$	{2}
3	2	7	2	$\{A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{12}\}$	{8, 12}
4	3	6	3	$\{A_6, A_8, A_9, A_{10}, A_{11}, A_{12}\}$	{8, 10, 12}
5	4	4	5	$\{A_9, A_{10}, A_{11}, A_{12}\}$	{7, 8, 9, 10, 12}
6	6	2	6	$\{A_9, A_{12}\}$	{5, 7, 8, 9, 10, 12}
7	7	1	7	$\{A_9\}$	{5, 7, 8, 9, 10, 11, 12}
8	8	1	8	$\{A_4\}$	{1, 2, 3, 4, 5, 6, 7, 8}
$v=01, \lvert F\rvert=11$					
1	0	12	0	R	$\varnothing$
2	1	8	1	$\{A_1, A_3, A_4, A_5, A_6, A_7, A_8, A_{10}\}$	{2}
3	2	7	3	$\{A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{12}\}$	{3, 8, 12}
4	4	6	4	$\{A_6, A_8, A_9, A_{10}, A_{11}, A_{12}\}$	{3, 8, 10, 12}
5	5	5	5	$\{A_6, A_9, A_{10}, A_{11}, A_{12}\}$	{3, 4, 8, 10, 12}
6	6	4	6	$\{A_6, A_7, A_8, A_{10}\}$	{1, 2, 3, 5, 8, 12}
7	7	4	8	$\{A_9, A_{10}, A_{11}, A_{12}\}$	{3, 4, 6, 7, 8, 9, 10, 12}
8	9	3	9	$\{A_6, A_7, A_8\}$	{1, 2, 3, 5, 6, 7, 8, 9, 12}
9	10	2	10	$\{A_6, A_8\}$	{1, 2, 3, 5, 6, 7, 8, 9, 10, 12}
10	11	2	11	$\{A_9, A_{12}\}$	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12}
11	12	1	12	$\{A_{11}\}$	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Master Algorithm:
Initialization: $n_S = 0 = 0$, $C = \varnothing$.

Do{
Select $A_h \in RÇ$ that maximizes $|r|_v$ (the number of components r_j that equal v) for
$v = 0$ or 1, or that maximizes $||r|_1 - |r|_0||$ for $v = 01$;
Execute Algorithm 1 or Algorithm 2 starting from A_h to generate a set S;
$n_S = n_S + 1$
$C = C \cup S$;
} While $|S| \geq L$ and $n_S < U$;

Remark 6:. *If the convention* $v = 01$ *is coded by* $v = 1/2$, *the instruction*
Select $A_h \in RÇ$ that maximizes $|r|_v$ (the number of components r_j that equal v) for
$v = 0$ or 1, or that maximizes $||r|_1 - |r|_0||$ for $v = 01$;
can be replaced by

$$A_h = argmax\{|(1-v)|r|_0 - v|r|_1| : r \in RÇ\}$$

or equivalently

$$A_h = argmax\{|(1-v)e\bar{r} - ver| : r \in RÇ\}$$

We now elaborate this description of the Master Algorithm with several important considerations.

6.2. Overlapping clusters

We note the interesting fact that the sets S for $v = 0$ and $v = 1$ can have overlaps, here consisting of the row A_3. The existence of overlapping clusters in the present setting, where the sets are generated from different starting rows and con-

structed according to different goals, invites exploration to see whether such overlaps are common for problems commonly encountered, and to identify the significance of such overlaps when they occur. (If the values chosen for ε_v^2 are small enough, however, it is clear that a large number of overlaps may be expected.)

The case for $v = 01$ in the preceding example generates a set S that does not overlap with those for other v values and produces sets S with larger values of $|z|_v$. It should not be surprising that $v = 01$ enables a larger number of components to be fixed than in the cases for $v = 0$ and $v = 1$, though the relevance of this undoubtedly depends on the problem setting. In the present example, we have set a lower bound $\varepsilon_{01}^2 = 9$ for $v = 01$, which is somewhat larger than the lower bounds chosen for $v = 0$ and $v = 1$.

Finally, the Algorithm Example shows the result of picking a different starting row A_h for $v = 1$. This second choice for A_h follows the proposal of the Master Algorithm in Section 4, which selects a new starting row to be one that contains a largest number of components equal to v, subject to the condition that this row does not lie in a set S previously generated for v. Thus, A_9, which does not belong to the previous set S for $v = 1$, and which contains 7 components equal to 1, is the new choice for A_h.

Although $S = \{A_9\}$ does not yield $|z|_1$ as large as produced by the earlier choice that gives $S = \{A_4\}$, the ultimate set $S = \{A_9, A_{10}, A_{11}, A_{12}\}$ has a larger size ($|S| = 4$) than in the previous instance. The two sets S generated for $v = 1$ in this example are disjoint, suggesting that the two cases have a meaningful difference from the standpoint of forming clusters.

This illustrates another interesting phenomenon. Suppose we use a simple frequency count of the number of times that the assignments equal to v appear in the rows A_1 to A_{12} as a basis for assigning probabilities for creating new rows, and consider the case for $v = 1$. The resulting frequencies are shown in Table 2.3.

Grouped by frequency, r_2 and r_8 are the most attractive for setting $r_j = 1$, followed by r_7, r_9 and r_{12}, while r_1, r_3, r_4, r_5 and r_{11} are the least attractive. However, $r_2 = 1$ does not show up in any rows where r_8 receives the value 1, but instead is associated with the less attractive variables r_1, r_3 and r_5 (in the set $S = \{A_3, A_4, A_5\}$ generated above). This shows that a measure of "attractiveness" by frequency counts can be misleading and fail to capture associations between components. Interestingly, however, the example suggests that we may use the appearance of a component in more than one set S based on the same $v = 0$ or 1 as an indication of being strongly determined without the computational expense otherwise required to identify such components. Here, $r_7 = 1$ appears in both of the sets S illustrated in the Algorithm Example for $v = 1$, and hence may be viewed as strongly determined (in spite of the fact that r_2 and r_8 have higher frequency counts). In short, a "higher order" frequency count, which identifies the number of occurrences of $r_j = v$ in the sets S generated by Algorithm 1 or 2 for a given v, gives a useful basis for identifying variables that may be considered strongly determined.

As alluded to in earlier comments, a further elaboration of the underlying algorithmic processes for this example is possible by examining "next S sets" starting from the final sets in this example. Pursuing this strategy shows that the best remaining evaluation for $v = 0$ yields $|z|_0 = 3$ as remarked above. If ε_0^2 was selected to be 3 instead of 4 (as currently stipulated), the next solution added to S for $v = 0$ after A_3 in the Algorithm Example would be A_4 or A_5. However, this drop from $|z|_0 = 5$ to $|z|_0 = 3$ represents a decline in $|z|_0$ from $0.42 \times n$ to $0.25 \times n$. An even greater decline occurs for the case $v = 1$, where $|z|_1 = 5$ drops to $|z|_1 = 2$. Consequently, this suggests that the algorithm might be modified to adaptively monitor $|z|_v$ to allow the method to stop before allowing a significant decline in $|z|_v$ by adding one more solution to S. The structure of the algorithm is congenial to making such a modification.

7. Computational results

The proposed algorithms were implemented in C++ and run on a PC computer with a 2.9 Ghz processor. The MIPs were solved using CPLEX 12.6.1. To perform the computational experiments, we note that an instance of the problem is characterized by an input binary matrix $A(m, n)$ where m is the number of rows and n is the number of columns (see Section 1.1). Several sources exist in the literature where these matrices can be obtained from real-world data. In this paper, the binary matrices are generated randomly using a simple uniform distribution provided by the Microsoft Visual Studio. Specifically, for a given dimension (i.e. m and n are fixed), a component a_{ij} of the matrix A, is generated randomly in {0, 1}. For each fixed dimension (m, n), 10 instances are generated randomly.

We compare the performance of the proposed integer program $IP_v^1(\varepsilon^2, h)$ with the constructive heuristic and its accelerated version. The maximum size of $m \times n$ is fixed to 500×500 or 200×2000 according the memory capacity of the computer used. The computation time for each integer program $IP_v^1(\varepsilon^2, h)$ is limited to 3600 s. For some hard instances, we found that CPLEX crashed with an "Out of memory" message before reaching the time limit. In such cases we reduced the time limit to lie between 100 and 360 s in order to obtain a feasible solution for comparison.

For all computational tests we choose the target row A_h as follows:

Table 2.3
The frequency.

j	1	2	3	4	5	6	7	8	9	10	11	12
Frequency $=\sum_{i=1}^{m=12} a_{ij}$	5	8	4	5	6	5	7	8	7	6	5	7

$$A_h = \begin{cases} argmax\{|N_v(A_i)| : i \in M\} & if\ v \in \{0,1\} \\ argmax\{||N_0(A_i)| - |N_1(A_i)|| : i \in M\} & if\ v = 01 \end{cases}$$

For each instance with fixed dimension n and m, the lower bound is chosen by setting:

$$\varepsilon_v^2 = |N_v(R)| + \alpha\big(|N_v(A_h)| - |N_v(R)|\big)$$

$$\alpha \in \{0.2, 0.4, 0.6, 0.8\}$$

The legend of the columns of the tables uses the following conventions:

- CPLEX refers to the MIP solver of CPLEX used to solve $IP_v^1(\varepsilon^2, h)$.
- H corresponds to the constructive heuristic described in Algorithm 1
- AH corresponds to the accelerated version of the constructive heuristic described in Algorithm 2.
- $|S^*|$ is the size of the best set of rows generated by the corresponding approach.
- $|N_v(S^*)|$ is the size of the intersect set corresponding to S^*.
- CPU is the computation time in seconds needed by the procedure.
- #Opt is the number of instances solved optimally by CPLEX.
- #E1 is the number of best solutions generated by the constructive heuristics with the same first objective $|S^*|$ as used by CPLEX (i.e. MIP($|S^*|$) = H($|S^*|$)).
- #E2 is the number of best solutions generated by the constructive heuristics with the same first objective value $|S^*|$ as generated by CPLEX and the second objective value $|N_v(S^*)|$ is strictly greater than the one generated by CPLEX (i.e. MIP($|S^*|$) = H($|S^*|$) and MIP($|N_v(S^*)|$) < H($|N_v(S^*)|$)). Hence #E2 identifies the number of solutions generated by the constructive heuristics that dominate those obtained by CPLEX.

Each row is an average of the values computed over various instances and the last row corresponds to the average of all instances.

Tables 3.1–3.3 compare the results of the MIP solver (CPLEX), the constructive heuristic (H) and its accelerated version (AH) for value v = 0, 1 and 01 respectively, on small-sized instances with n = 125, $m \in \{12, 25, 62\}$ and $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$. Each line corresponds to the average of 50 instances (10 randomly generated instances with 5 different densities). The computational time for the heuristics H and AH are not reported since they are negligible (i.e. less than 0.0005 s). Also, the solution objective values $|S^*|$ and $|N_v(S^*)|$ for H and AH are the same in all cases and hence are reported under the single heading "H – AH".

For $v = 0$ (see Table 3.1), most of the 600 instances are solved optimally except for 7 instances for m = 62 and $\alpha \in \{0.2, 0.4\}$ (i.e. six instances are not solved optimally due to the memory limitation "Out of Memory" error and one instance is not solved optimally within the time limit of one hour). Table 3.1 shows also that the constructive heuristic reaches an optimal value for 84% of the instances and, significantly, provides solutions that dominate those provided by CPLEX in 49% of the instances. For $v = 1$ (see Table 3.2), all 600 instances are solved optimally by CPLEX. The constructive heuristic reaches an optimal value for 97% of the instances and provides solutions that dominate those provided by CPLEX in 24% of the instances. For $v = 01$ (see Table 3.3), CPLEX solves optimally 98% of the instances (11 instances are not solved optimally due to the memory limitation "Out of Memory" error) while the constructive heuristic reaches optimal value for 87% of the instances and provides solutions that dominate those provided by CPLEX in 43% of the instances. Note also that regarding the CPU time, the hardest instances correspond to instances with the parameters ($v \in \{0, 1, 01\}$, m = 62 and $\alpha \in \{0.2, 0.4\}$).

Table 3.1
Comparison of MIP, Heuristic and Accelerated Heuristic for n = 125 and v = 0.

m	α	MIP			H – AH		#Opt	#E1	#E2
		$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$			
12	0.2	6.36	26.60	0.069	6.30	29.00	50	47	31
12	0.4	4.24	45.70	0.064	4.20	49.00	50	47	37
12	0.6	2.66	68.20	0.030	2.60	71.00	50	47	24
12	0.8	1.60	87.80	0.005	1.60	88.00	50	50	6
25	0.2	9.52	20.60	1.200	9.10	23.00	50	30	19
25	0.4	5.66	41.30	0.340	5.50	46.00	50	43	36
25	0.6	3.52	63.10	0.120	3.50	65.00	50	48	32
25	0.8	1.78	87.70	0.027	1.80	89.00	50	49	16
62	0.2	13.80	19.30	450.000	13.00	20.00	45	18	12
62	0.4	7.48	39.50	250.000	7.00	44.00	48	30	27
62	0.6	4.32	60.10	8.300	4.20	63.00	50	47	34
62	0.8	2.06	88.00	0.220	2.10	89.00	50	50	17
Average		**5.25**	**53.99**	**59.20**	**5.08**	**56.33**	**49.42**	**42.17**	**24.25**

Table 3.2
Comparison of MIP, Heuristic and Accelerated Heuristic for n = 125 and v = 1.

m	α	MIP			H – AH		#Opt	#E1	#E2
		$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$			
12	0.2	2.64	10.20	0.056	2.60	12.00	50	49	27
12	0.4	1.52	28.50	0.033	1.50	30.00	50	50	16
12	0.6	1.04	44.10	0.003	1.00	44.00	50	50	1
12	0.8	1.00	45.30	0.002	1.00	45.00	50	50	0
25	0.2	2.96	9.62	0.120	2.80	12.00	50	44	34
25	0.4	1.74	26.00	0.072	1.70	28.00	50	49	16
25	0.6	1.12	43.20	0.004	1.10	43.00	50	50	1
25	0.8	1.00	46.60	0.003	1.00	47.00	50	50	0
62	0.2	3.50	9.64	1.500	3.30	11.00	50	42	22
62	0.4	1.86	25.40	0.460	1.80	28.00	50	47	21
62	0.6	1.18	43.30	0.008	1.20	43.00	50	50	5
62	0.8	1.00	48.70	0.006	1.00	49.00	50	50	0
Average		**1.71**	**31.71**	**0.19**	**1.67**	**32.67**	**50.00**	**48.42**	**11.92**

Table 3.3
Comparison of MIP, Heuristic and Accelerated Heuristic for n = 125 and v = 01.

m	α	MIP			H – AH		#Opt	#E1	#E2
		$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$			
12	0.2	5.78	33.60	0.088	5.70	36.00	50	46	32
12	0.4	3.66	61.90	0.100	3.60	67.00	50	47	40
12	0.6	2.24	95.70	0.039	2.20	97.00	50	49	19
12	0.8	1.20	122.00	0.005	1.20	120.00	50	50	3
25	0.2	8.64	26.60	1.700	8.30	30.00	50	35	26
25	0.4	5.06	54.50	0.370	5.00	59.00	50	45	33
25	0.6	2.90	86.50	0.120	2.90	89.00	50	50	29
25	0.8	1.46	119.00	0.025	1.50	120.00	50	50	3
62	0.2	12.40	25.20	830.000	11.00	27.00	41	23	15
62	0.4	6.60	52.30	97.000	6.30	56.00	48	37	26
62	0.6	3.72	78.60	7.100	3.60	83.00	50	43	25
62	0.8	1.76	116.00	0.150	1.70	120.00	50	48	8
Average		**4.62**	**72.66**	**78.06**	**4.42**	**75.33**	**49.08**	**43.58**	**21.58**

Tables 4.1–4.3 compare the results of the MIP solver (CPLEX), the constructive heuristic (H) and its accelerated version (AH) for value $v = 0, 1$ and 01 respectively, on medium-sized instances with $n \in \{125, 250, 500\}$, $m = \lfloor \beta n \rfloor$ with $\beta \in \{0.1, 0.2, 0.5, 1\}$ and $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$. Each line corresponds to the average of 10 randomly generated instances for a fixed value of n, m and α. Once again, H and AH obtain the same objective values but now their solution times differ enough in some cases to be reported separately.

For $v = 0$ (see Table 4.1), 86% of the instances are solved optimally except for 68 instances over 480 (i.e. only one instance is not solved optimally in the time limit of one hour and the other 67 instances are not solved optimally due to the memory limitation "Out of Memory" error). Table 4.1 shows also that the constructive heuristic reaches optimal value for 90% of the instances and provides solutions that dominate those provided by CPLEX in 36% of the instances. For $v = 1$ (see Table 4.2), 87% of the instances are solved optimally except for 63 instances over 480 (i.e. the other 63 instances are not solved optimally due to the memory limitation "Out of Memory" error). Table 4.2 shows that the constructive heuristic reaches an optimal value for 93% of the instances and provides solutions that dominate those provided by CPLEX in 40% of the instances. For $v = 01$ (see Table 4.3), 84% of the instances are solved optimally except for 79 instances over 480 (i.e. 10 instances are not solved optimally in the time limit of one hour and the other 69 instances are not solved optimally due to the memory limitation "Out of Memory" error). The constructive heuristic reaches an optimal value for 95% of the instances and provides solutions that dominate those provided by CPLEX in 43% of the instances. Note also that regarding the CPU time consumed by CPLEX, the hardest instances correspond to instances with the parameter $\alpha \in \{0.2, 0.4\}$.

Tables 5.1–5.3 compare the results of the MIP solver (CPLEX), the constructive heuristic (H) and its accelerated version (AH) for value $v = 0, 1$ and 01 respectively, on large-sized instances with $n = 2000$ and $m = 50$. Each line corresponds to the average of 4 randomly generated instances for a fixed value of n, m. For $v = 0$ (see Table 5.1), 94% of the instances are solved optimally except for 4 instances over 72 (i.e. only one instance is not solved optimally due to the memory limitation "Out of Memory" error and the other 3 instances are not solved optimally in the time limit of one hour). Table 5.1 shows also that the constructive heuristic reaches an optimal value for 71% of the instances and provides solutions that dominate those provided by CPLEX in 64% of the cases. For $v = 1$ (see Table 5.2), all 72 instances are solved optimally. Table 3.2 shows that the constructive heuristic reaches optimal value for 96% of the instances and provides solutions that dominate those provided by CPLEX in 33% of the instances. For $v = 01$ (see Table 5.3), 79% of the instances are solved optimally except

Table 4.1
Comparison of MIP, Heuristic and Accelerated Heuristic for $v = 0$.

n	m	α	MIP			H			AH			#E2
			$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$	CPU	CPU			
125	12	0.2	4.0	14.4	0.125	3.9	15.8	0.000	0.000	10	9	3
125	12	0.4	2.0	37.6	0.135	2.0	41.8	0.000	0.000	10	10	8
125	12	0.6	1.3	63.1	0.004	1.3	63.1	0.000	0.000	10	10	0
125	12	0.8	1.0	71.9	0.006	1.0	71.9	0.000	0.000	10	10	0
125	25	0.2	4.4	14.5	0.401	4.1	17.1	0.000	0.000	10	7	6
125	25	0.4	2.5	33.8	0.180	2.5	36.7	0.000	0.000	10	10	6
125	25	0.6	1.7	53.7	0.006	1.7	54.4	0.000	0.000	10	10	3
125	25	0.8	1.0	73.9	0.003	1.0	73.9	0.000	0.000	10	10	0
125	62	0.2	5.0	14.8	5.610	4.6	17.0	0.000	0.000	10	6	4
125	62	0.4	2.7	33.8	1.707	2.6	37.2	0.000	0.000	10	9	2
125	62	0.6	2.0	45.3	0.015	2.0	46.2	0.000	0.000	10	10	6
125	62	0.8	1.0	75.1	0.009	1.0	75.1	0.001	0.000	10	10	0
125	125	0.2	5.5	15.0	66.747	5.3	16.3	0.001	0.000	10	8	5
125	125	0.4	3.0	30.7	7.153	3.0	32.9	0.000	0.000	10	10	9
125	125	0.6	2.0	47.6	0.072	2.0	49.6	0.000	0.000	10	10	7
125	125	0.8	1.0	76.6	0.014	1.0	76.6	0.000	0.000	10	10	0
250	25	0.2	3.9	28.8	0.707	3.9	33.1	0.000	0.000	10	10	10
250	25	0.4	2.0	73.6	0.279	2.0	82.3	0.000	0.000	10	10	9
250	25	0.6	1.3	124.1	0.006	1.3	124.1	0.000	0.000	10	10	0
250	25	0.8	1.0	141.1	0.006	1.0	141.1	0.000	0.000	10	10	0
250	50	0.2	4.0	27.9	9.831	4.0	33.9	0.000	0.000	10	10	10
250	50	0.4	2.0	72.9	2.329	2.0	84.3	0.002	0.000	10	10	10
250	50	0.6	1.7	102.2	0.013	1.7	102.2	0.000	0.000	10	10	0
250	50	0.8	1.0	141.6	0.008	1.0	141.6	0.000	0.000	10	10	0
250	125	0.2	4.4	29.0	810.670	4.0	37.1	0.000	0.000	9	6	6
250	125	0.4	2.5	68.4	32.983	2.4	76.3	0.000	0.000	10	9	4
250	125	0.6	1.6	113.1	0.021	1.6	113.6	0.000	0.000	10	10	1
250	125	0.8	1.0	146.8	0.019	1.0	146.8	0.000	0.000	10	10	0
250	250	0.2	4.0	29.3	252.044	4.5	33.9	0.002	0.001	0	5	5
250	250	0.4	2.6	64.7	268.853	2.5	74.7	0.001	0.001	3	9	5
250	250	0.6	1.9	95.3	0.054	1.9	96.2	0.002	0.000	10	10	3
250	250	0.8	1.0	146.8	0.033	1.0	146.8	0.002	0.000	10	10	0
500	50	0.2	4.0	54.9	41.723	3.5	73.3	0.001	0.000	10	5	2
500	50	0.4	2.0	133.8	5.237	2.0	156.4	0.002	0.000	10	10	10
500	50	0.6	1.1	263.7	0.020	1.1	263.7	0.000	0.000	10	10	0
500	50	0.8	1.0	275.0	0.019	1.0	275.0	0.001	0.000	10	10	0
500	100	0.2	3.6	59.3	166.053	4.0	57.5	0.004	0.001	0	6	6
500	100	0.4	2.0	137.0	142.226	2.0	160.4	0.001	0.000	10	10	10
500	100	0.6	1.0	278.1	0.027	1.0	278.1	0.000	0.000	10	10	0
500	100	0.8	1.0	278.1	0.026	1.0	278.1	0.000	0.000	10	10	0
500	250	0.2	3.2	65.0	100.035	4.0	63.5	0.003	0.003	0	2	2
500	250	0.4	2.0	137.0	100.032	2.0	166.6	0.002	0.001	0	10	10
500	250	0.6	1.5	225.9	0.054	1.5	225.9	0.001	0.000	10	10	0
500	250	0.8	1.0	282.1	0.052	1.0	282.1	0.003	0.001	10	10	0
500	500	0.2	3.1	72.5	120.069	4.0	64.8	0.007	0.003	0	1	1
500	500	0.4	2.0	144.1	120.185	2.0	166.0	0.008	0.003	0	10	10
500	500	0.6	1.1	271.8	4.323	1.1	271.8	0.004	0.002	10	10	0
500	500	0.8	1.0	283.0	3.417	1.0	283.0	0.002	0.001	10	10	0
Average			**2.2**	**105.6**	**47.157**	**2.2**	**109.6**	**0.001**	**0.000**	**8.6**	**9.0**	**3.6**

for 15 instances over 72 (i.e. 3 instances are not solved optimally in the time limit of one hour and the other 12 instances are not solved optimally due to the memory limitation "Out of Memory" error). The constructive heuristic reaches an optimal value for 76% of the instances and provides solutions that dominate those provided by CPLEX in 56% of the instances.

Table 6 compares the CPU time consumed by the constructive heuristic (H) and its accelerated version (AH) for value $v = 0, 1$ and 01, on large-sized instances with $n = 2000$ and $m \in \{100, 200\}$. Each line corresponds to the average of 4 randomly generated instances for a fixed value of m. For $v = 0$ and $v = 1$, the CPU time of the accelerated heuristic is half of the CPU time consumed by the constructive heuristic on average.

8. Conclusions

In addition to applications of our algorithms to clustering, the exploitation of consistency by the approaches described above open a number of opportunities for applying our algorithms to generate reference sets for evolutionary metaheuris-

Table 4.2
Comparison of MIP, Heuristic and Accelerated Heuristic for $v = 1$.

n	m	α	MIP			H			AH			#E2
			$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$	CPU	CPU			
125	12	0.2	4.0	14.3	0.165	3.7	17.0	0.000	0.000	10	7	2
125	12	0.4	2.0	37.2	0.134	2.0	41.3	0.000	0.001	10	10	6
125	12	0.6	1.3	63.3	0.002	1.3	63.4	0.000	0.000	10	10	1
125	12	0.8	1.0	71.6	0.002	1.0	71.6	0.000	0.000	10	10	0
125	25	0.2	4.2	14.6	0.346	4.0	16.8	0.000	0.000	10	8	6
125	25	0.4	2.1	36.9	0.213	2.1	41.1	0.000	0.000	10	10	9
125	25	0.6	1.5	57.5	0.008	1.5	57.5	0.000	0.000	10	10	0
125	25	0.8	1.0	72.7	0.005	1.0	72.7	0.000	0.000	10	10	0
125	62	0.2	5.0	15.1	5.798	4.8	16.7	0.000	0.000	10	8	5
125	62	0.4	3.0	30.7	1.875	2.8	34.4	0.000	0.000	10	8	4
125	62	0.6	2.0	47.0	0.019	2.0	47.7	0.000	0.000	10	10	5
125	62	0.8	1.0	76.2	0.007	1.0	76.2	0.000	0.000	10	10	0
125	125	0.2	5.6	15.2	69.342	5.0	17.5	0.001	0.000	10	4	3
125	125	0.4	3.0	30.9	7.642	3.0	33.7	0.000	0.000	10	10	9
125	125	0.6	2.0	48.0	0.120	2.0	50.0	0.000	0.000	10	10	8
125	125	0.8	1.0	76.5	0.015	1.0	76.5	0.001	0.000	10	10	0
250	25	0.2	4.0	28.2	0.705	3.9	30.8	0.000	0.000	10	9	3
250	25	0.4	2.0	70.6	0.281	2.0	81.2	0.000	0.000	10	10	10
250	25	0.6	1.2	129.8	0.002	1.2	129.8	0.000	0.000	10	10	0
250	25	0.8	1.0	141.4	0.006	1.0	141.4	0.000	0.000	10	10	0
250	50	0.2	4.0	28.5	10.367	4.0	33.9	0.002	0.000	10	10	10
250	50	0.4	2.1	70.7	2.519	2.1	81.8	0.000	0.000	10	10	9
250	50	0.6	1.5	115.7	0.008	1.5	115.7	0.000	0.000	10	10	0
250	50	0.8	1.0	143.8	0.006	1.0	143.8	0.000	0.000	10	10	0
250	125	0.2	4.2	29.7	727.591	4.1	36.0	0.002	0.001	9	9	8
250	125	0.4	2.4	68.6	33.135	2.3	79.0	0.000	0.000	10	9	7
250	125	0.6	1.8	100.3	0.035	1.8	100.6	0.001	0.000	10	10	2
250	125	0.8	1.0	145.7	0.016	1.0	145.7	0.000	0.000	10	10	0
250	250	0.2	4.1	29.9	262.017	4.3	36.2	0.001	0.001	0	8	8
250	250	0.4	2.7	64.4	310.897	2.7	68.7	0.002	0.001	8	10	7
250	250	0.6	1.8	101.5	0.149	1.8	103.4	0.001	0.001	10	10	6
250	250	0.8	1.0	147.7	0.128	1.0	147.7	0.001	0.000	10	10	0
500	50	0.2	3.8	58.0	46.154	3.4	76.5	0.001	0.000	10	6	4
500	50	0.4	2.0	140.7	3.880	2.0	158.3	0.000	0.000	10	10	10
500	50	0.6	1.0	276.3	0.019	1.0	276.3	0.000	0.000	10	10	0
500	50	0.8	1.0	276.3	0.019	1.0	276.3	0.000	0.000	10	10	0
500	100	0.2	3.6	59.1	318.040	3.8	64.7	0.002	0.001	0	8	8
500	100	0.4	2.0	136.8	149.210	2.0	160.1	0.001	0.000	10	10	10
500	100	0.6	1.1	267.6	0.030	1.1	267.6	0.000	0.000	10	10	0
500	100	0.8	1.0	278.4	0.028	1.0	278.4	0.000	0.000	10	10	0
500	250	0.2	3.6	61.9	100.025	4.0	61.8	0.003	0.002	0	6	6
500	250	0.4	2.0	142.5	100.023	2.0	166.1	0.003	0.001	0	10	10
500	250	0.6	1.3	248.1	0.055	1.3	248.1	0.001	0.000	10	10	0
500	250	0.8	1.0	282.0	0.053	1.0	282.0	0.002	0.000	10	10	0
500	500	0.2	3.4	65.2	120.054	4.0	66.6	0.007	0.003	0	4	4
500	500	0.4	2.0	144.8	120.050	2.0	168.3	0.003	0.002	0	10	10
500	500	0.6	1.4	238.7	4.306	1.4	238.7	0.004	0.001	10	10	0
500	500	0.8	1.0	284.2	3.589	1.0	284.2	0.001	0.000	10	10	0
Average			**2.2**	**105.9**	**49.981**	**2.2**	**110.1**	**0.001**	**0.000**	**8.7**	**9.3**	**4.0**

tics. Questions related to such opportunities include the following, whose answers will depend in part on the evolutionary metaheuristics employed.

1. Which values of $v \in \{0, 1, 01\}$ prove most useful for particular classes of optimization problems?
2. What values for the bounds ε_v^1 and ε_v^2 for Problems $G_v^1(\varepsilon_v^2)$ and $G_v^2(\varepsilon_v^1)$ are best (e.g., as a function of n)
3. If the approximation algorithm is run only for a small number of iterations to generate new instances of R after the first, what portion of a previous R should be saved to merge with a new R?
4. What type of diversification approach should be used as a foundation for generating instances of R after the first instance?
5. What range of different sets S may be generated to give useful variation in the consistent variables identified and to identify strongly determined variables by higher order frequencies?
6. Do advantages result by using an adaptive version of Algorithms 1 and 2 as indicated in Section 6?

Table 4.3
Comparison of MIP, Heuristic and Accelerated Heuristic for v = 01.

n	m	α	MIP			H			AH	#Opt	#E1	#E2
			$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$	CPU	CPU			
125	12	0.2	3.4	29.8	0.171	3.4	33.6	0.000	0.000	10	10	5
125	12	0.4	2.0	65.1	0.195	2.0	70.0	0.000	0.000	10	10	8
125	12	0.6	1.1	120.0	0.004	1.1	120.0	0.000	0.000	10	10	0
125	12	0.8	1.0	125.0	0.007	1.0	125.0	0.000	0.000	10	10	0
125	25	0.2	4.1	25.6	0.699	3.8	31.1	0.000	0.000	10	7	4
125	25	0.4	2.0	64.2	0.295	2.0	73.8	0.000	0.000	10	10	9
125	25	0.6	1.5	100.7	0.010	1.5	100.7	0.000	0.000	10	10	0
125	25	0.8	1.0	125.0	0.005	1.0	125.0	0.000	0.000	10	10	0
125	62	0.2	4.1	25.3	20.880	4.0	30.3	0.000	0.000	10	9	9
125	62	0.4	2.1	61.7	4.091	2.1	71.4	0.000	0.000	10	10	9
125	62	0.6	1.4	105.4	0.011	1.4	105.5	0.000	0.000	10	10	1
125	62	0.8	1.0	125.0	0.010	1.0	125.0	0.000	0.000	10	10	0
125	125	0.2	4.5	25.2	453.025	4.0	32.4	0.001	0.000	10	5	5
125	125	0.4	2.1	62.3	18.843	2.1	74.2	0.001	0.000	10	10	9
125	125	0.6	1.8	86.3	0.020	1.8	86.8	0.000	0.000	10	10	3
125	125	0.8	1.0	125.0	0.015	1.0	125.0	0.001	0.000	10	10	0
250	25	0.2	3.3	56.6	1.730	3.3	70.5	0.000	0.000	10	10	9
250	25	0.4	2.0	127.0	0.600	2.0	141.0	0.000	0.000	10	10	10
250	25	0.6	1.1	240.2	0.009	1.1	240.2	0.000	0.000	10	10	0
250	25	0.8	1.0	250.0	0.011	1.0	250.0	0.000	0.000	10	10	0
250	50	0.2	3.9	51.4	30.994	3.7	61.3	0.000	0.000	10	8	8
250	50	0.4	2.0	126.3	3.518	2.0	143.4	0.000	0.000	10	10	10
250	50	0.6	1.0	250.0	0.016	1.0	250.0	0.000	0.000	10	10	0
250	50	0.8	1.0	250.0	0.016	1.0	250.0	0.000	0.000	10	10	0
250	125	0.2	4.0	50.9	2279.830	4.0	56.1	0.003	0.002	1	10	10
250	125	0.4	2.0	123.5	182.474	2.0	145.6	0.000	0.000	10	10	10
250	125	0.6	1.0	250.0	0.033	1.0	250.0	0.000	0.000	10	10	0
250	125	0.8	1.0	250.0	0.031	1.0	250.0	0.000	0.000	10	10	0
250	250	0.2	3.7	53.0	248.032	4.0	57.8	0.002	0.001	0	7	7
250	250	0.4	2.0	124.5	2166.380	2.0	145.6	0.001	0.000	0	10	10
250	250	0.6	1.0	250.0	0.064	1.0	250.0	0.000	0.000	10	10	0
250	250	0.8	1.0	250.0	0.054	1.0	250.0	0.001	0.000	10	10	0
500	50	0.2	3.0	119.0	262.746	3.0	153.6	0.000	0.000	10	10	10
500	50	0.4	2.0	249.2	46.086	2.0	274.6	0.001	0.000	10	10	10
500	50	0.6	1.0	500.0	0.029	1.0	500.0	0.001	0.001	10	10	0
500	50	0.8	1.0	500.0	0.029	1.0	500.0	0.000	0.000	10	10	0
500	100	0.2	3.0	118.3	298.100	3.0	155.9	0.003	0.002	0	10	10
500	100	0.4	2.0	249.6	1203.750	2.0	276.1	0.001	0.000	10	10	10
500	100	0.6	1.0	500.0	0.040	1.0	500.0	0.000	0.000	10	10	0
500	100	0.8	1.0	500.0	0.040	1.0	500.0	0.000	0.000	10	10	0
500	250	0.2	3.0	121.7	100.034	3.2	151.1	0.003	0.001	0	8	8
500	250	0.4	2.0	248.4	100.051	2.0	278.9	0.000	0.000	0	10	10
500	250	0.6	1.0	500.0	0.206	1.0	500.0	0.004	0.001	10	10	0
500	250	0.8	1.0	500.0	0.233	1.0	500.0	0.004	0.000	10	10	0
500	500	0.2	2.8	162.9	116.086	3.5	132.7	0.008	0.005	0	4	4
500	500	0.4	1.8	294.1	120.250	2.0	280.4	0.008	0.004	0	8	8
500	500	0.6	1.0	500.0	0.284	1.0	500.0	0.006	0.000	10	10	0
500	500	0.8	1.0	500.0	0.296	1.0	500.0	0.007	0.000	10	10	0
Average			**1.9**	**198.7**	**159.590**	**1.9**	**205.1**	**0.001**	**0.000**	**8.4**	**9.5**	**4.3**

Metaheuristic strategies found in [13,15,21,26,34,35], and [36] are relevant to exploring the issues raised by the preceding questions. However, the new considerations introduced in this paper invite empirical studies of forms not previously conducted.

Table 5.1
Comparison of MIP, Heuristic and Accelerated Heuristic for n = 2000, m = 50 and v = 0.

Instance	MIP			H			AH	#Opt	#E1	#E2
	$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$	CPU	CPU			
1	14.50	1274.50	197.730	14.50	1283.75	0.007	0.004	3	4	4
2	16.00	1317.50	151.897	14.50	1326.50	0.008	0.005	4	1	1
3	14.75	1229.50	1123.480	13.75	1235.75	0.005	0.003	3	1	1

(*continued on next page*)

Table 5.1 (*continued*)

Instance	MIP			H			AH	#Opt	#E1	#E2
	$\|S^*\|$	$\|N_v(S^*)\|$	CPU	$\|S^*\|$	$\|N_v(S^*)\|$	CPU	CPU			
4	16.50	1169.25	300.759	16.00	1177.50	0.010	0.003	4	2	2
5	16.00	1228.00	317.072	16.00	1238.75	0.005	0.005	4	4	4
6	16.50	1241.00	489.668	16.50	1256.00	0.008	0.000	4	4	4
7	18.50	1313.00	169.470	18.50	1320.75	0.008	0.005	4	4	3
8	16.50	1200.00	384.925	16.50	1207.75	0.007	0.003	4	4	4
9	15.00	1244.50	1055.460	15.00	1260.25	0.010	0.003	3	4	3
10	15.00	1256.25	1099.630	14.25	1264.75	0.005	0.000	3	1	1
11	16.50	1223.50	212.398	16.50	1235.50	0.008	0.008	4	4	4
12	16.00	1220.00	435.857	15.00	1233.75	0.008	0.003	4	1	1
13	16.25	1181.00	586.565	16.25	1185.75	0.008	0.003	4	4	2
14	15.25	1330.00	243.923	14.75	1335.25	0.005	0.000	4	2	1
15	15.25	1255.25	604.018	15.00	1267.75	0.008	0.003	4	3	3
16	17.00	1270.75	278.472	15.50	1290.50	0.008	0.000	4	2	2
17	15.75	1296.00	253.367	16.00	1312.75	0.006	0.003	4	3	3
18	17.50	1265.75	157.707	17.25	1273.75	0.008	0.003	4	3	3
Avg	**16.04**	**1250.88**	**447.911**	**15.65**	**1261.49**	**0.007**	**0.003**	**3.78**	**2.83**	**2.56**

Table 5.2
Comparison of MIP, Heuristic and Accelerated Heuristic for n = 2000, m = 50 and v = 1.

Instance	MIP			H			AH	#Opt	#E1	#E2
	$\|\boldsymbol{S}^*\|$	$\|\boldsymbol{N_v}(\boldsymbol{S}^*)\|$	**CPU**	$\|\boldsymbol{S}^*\|$	$\|\boldsymbol{N_v}(\boldsymbol{S}^*)\|$	**CPU**	**CPU**			
1	2.00	130.50	0.974	2.00	130.50	0.001	0.001	4	4	0
2	2.25	132.25	1.576	2.00	137.25	0.000	0.000	4	3	1
3	1.50	160.00	0.155	1.50	160.00	0.003	0.000	4	4	0
4	2.00	134.75	1.181	2.00	137.00	0.000	0.000	4	4	2
5	2.00	132.75	1.184	2.00	135.00	0.000	0.000	4	4	2
6	2.25	133.75	1.628	2.25	134.00	0.000	0.000	4	4	1
7	2.50	132.75	6.286	2.50	138.00	0.003	0.000	4	4	2
8	2.00	132.50	1.310	2.00	138.00	0.001	0.000	4	4	2
9	2.00	133.75	0.675	2.00	134.50	0.000	0.000	4	4	1
10	1.75	132.00	0.510	1.75	137.00	0.003	0.000	4	4	1
11	2.25	130.75	0.786	2.00	137.00	0.000	0.000	4	3	1
12	2.00	131.75	0.582	2.00	135.75	0.005	0.000	4	4	2
13	1.75	135.00	0.190	1.75	136.75	0.003	0.000	4	4	1
14	2.00	133.75	1.135	2.00	135.75	0.000	0.000	4	4	1
15	2.00	131.50	1.723	2.00	136.50	0.000	0.000	4	4	2
16	2.25	134.50	0.403	2.25	138.50	0.003	0.000	4	4	2
17	2.75	130.25	2.208	2.50	138.00	0.002	0.000	4	3	1
18	2.50	132.00	2.997	2.50	138.25	0.000	0.000	4	4	2
Avg	**2.10**	**134.14**	**1.417**	**2.06**	**137.65**	**0.001**	**0.000**	**4.00**	**3.83**	**1.33**

Table 5.3
Comparison of MIP, Heuristic and Accelerated Heuristic for n = 2000, m = 50 and v = 01.

	MIP			H			AH			
Instance	$\|\boldsymbol{S}^*\|$	$\|\boldsymbol{N_v}(\boldsymbol{S}^*)\|$	**CPU**	$\|\boldsymbol{S}^*\|$	$\|\boldsymbol{N_v}(\boldsymbol{S}^*)\|$	**CPU**	**CPU**	**#Opt**	**#E1**	**#E2**
1	12.00	1371.75	380.550	12.00	1384.50	0.007	0.005	3	4	4
2	13.00	1427.25	514.354	12.00	1429.75	0.005	0.003	4	2	1
3	11.75	1396.50	511.151	11.25	1400.00	0.003	0.005	3	2	1
4	14.00	1283.00	652.712	13.50	1294.00	0.008	0.000	3	3	2
5	13.25	1330.25	309.424	13.25	1348.50	0.005	0.003	3	4	3
6	13.75	1345.25	222.256	13.75	1366.50	0.005	0.003	3	2	2
7	15.00	1417.00	936.245	15.00	1431.25	0.006	0.003	4	4	2
8	13.75	1320.25	257.064	13.75	1323.25	0.005	0.005	3	4	2
9	12.75	1351.25	1037.430	12.75	1363.50	0.003	0.003	2	4	4
10	12.00	1361.75	555.008	11.75	1368.00	0.008	0.000	3	3	2
11	14.00	1320.75	1186.710	14.00	1336.75	0.008	0.003	3	4	3
12	13.25	1326.25	307.274	12.50	1352.00	0.007	0.000	3	1	1
13	13.75	1284.50	714.217	13.50	1314.00	0.003	0.005	4	3	2
14	12.75	1434.25	137.308	12.25	1437.25	0.005	0.003	3	2	1

Table 5.3 (*continued*)

	MIP			H			AH			
Instance	$\|S^*\|$	$\|N_v(S^*)\|$	**CPU**	$\|S^*\|$	$\|N_v(S^*)\|$	**CPU**	**CPU**	**#Opt**	**#E1**	**#E2**
15	12.50	1365.75	782.128	12.25	1380.25	0.005	0.005	3	3	3
16	14.00	1385.50	222.542	13.25	1389.25	0.007	0.006	3	2	1
17	13.50	1407.75	760.556	13.50	1417.25	0.005	0.004	4	4	3
18	14.50	1368.25	9675.590	14.50	1389.25	0.005	0.008	3	4	3
Avg	**13.31**	**1360.96**	**1064.584**	**13.04**	**1373.63**	**0.005**	**0.003**	**3.17**	**3.06**	**2.22**

Table 6
CPU time comparison of Heuristic and Accelerated Heuristic for n = 2000.

m	α	v = 0		v = 1		v = 01	
		H	AH	H	AH	H	AH
100	0.2	0.004	0.003	0.004	0.002	0.004	0.003
100	0.4	0.003	0.002	0.003	0.002	0.003	0.003
100	0.6	0.002	0.001	0.002	0.001	0.002	0.002
100	0.8	0.002	0.001	0.002	0.001	0.002	0.002
200	0.2	0.007	0.005	0.007	0.005	0.008	0.006
200	0.4	0.006	0.004	0.006	0.004	0.006	0.005
200	0.6	0.004	0.002	0.003	0.002	0.004	0.003
200	0.8	0.004	0.002	0.004	0.002	0.003	0.003
Average		**0.004**	**0.002**	**0.004**	**0.002**	**0.004**	**0.003**

CRediT authorship contribution statement

Saïd Hanafi: Conceptualization, Methodology. **Gintaras Palubeckis:** Conceptualization, Methodology, Writing - review & editing. **Fred Glover:** Conceptualization, Methodology, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] P.K. Agarwal, N. Alon, B. Aronov, S. Suri, Can visibility graphs be represented compactly?, Discr. Comput. Geometry 12 (3) (1994) 347–365.
[2] M. Allais, Pareto, Vilfredo: contributions to economics, In: International Encyclopedia of the Social Sciences, vol. 11, New York, 1968, pp. 399–411.
[3] S. Busygin, O. Prokopyev, P.M. Pardalos, Biclustering in data mining, Comput. Oper. Res. 35 (9) (2008) 2964–2987.
[4] Y. Cheng G.M. Church Biclustering of expression data In: Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology 2000 93 103
[5] I.S. Dhillon, Co-clustering documents and words using bipartite spectral graph partitioning, In: Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 269–274.
[6] S. Dolnicar, S. Kaiser, K. Lazarevski, F. Leisch, Biclustering: overcoming data dimensionality problems in market segmentation, J. Travel Res. 51 (1) (2012) 41–49.
[7] M. Ehrgott, Multicriteria Optimization, 2nd Edition., Springer, Berlin, 2005.
[8] N. Fan, N. Boyko, P.M. Pardalos, Recent advances of data biclustering with application in computational neuroscience, In: Computational Neuroscience, Springer, New York, NY, 2010, pp. 85–112.
[9] N. Fan A. Chinchuluun P.M. Pardalos Integer programming of biclustering based on graph models, In: Optimization and Optimal Control 2010 Springer New York, NY 479 498
[10] P.C. Fishburn, P.L. Hammer, Bipartite dimensions and bipartite degrees of graphs, Discrete Mathematics 160 (1–3) (1996) 127–148.
[11] V. Froidure, Rangs des relations binaires et semigroupes de relations non ambigus, Doctoral dissertation, Paris 6 (1995).
[12] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, NY, 1979.
[13] F. Glover, Multi-wave algorithms for metaheuristic optimization, J. Heuristics 22 (3) (2016) 331–358.
[14] F. Glover, S. Hanafi, G. Palubeckis, Supplementary material: bi-objective clustering with binary data, arXiv: 2002.04711, http://arxiv.org/abs/2002.04711, (2020).
[15] F. Glover, J.-K. Hao, Diversification-based learning in computing and optimization, J. Heurist. 25 (4–5) (2019) 521–537.
[16] M. Golchin, A.W.C. Liew, Parallel biclustering detection using strength Pareto front evolutionary algorithm, Inf. Sci. 415 (2017) 283–297.
[17] M. Habib, L. Nourine, O. Raynaud, A new lattice-based heuristic for taxonomy encoding, In: International KRUSE Symposium: Knowledge, Retrieval, Use and Storage for Efficiency, Vancouver, 1997, pp. 60–71.
[18] Y.Y. Haimes, L.S. Lasdon, D.A. Wismer, On a bicriterion formulation of the problems of integrated system identification and system optimization, IEEE Transactions on Systems, Man, and Cybernetics 1(3) (1971) 296–297.
[19] J.A. Hartigan, Direct clustering of a data matrix, J. Am. Stat. Assoc. 67 (337) (1972) 23–129.
[20] Q. Huang, X. Huang, Z. Kong, X. Li, D. Tao, Bi-phase evolutionary searching for biclusters in gene expression data, IEEE Trans. Evol. Comput. 23 (2019) 803–814.
[21] X. Lai, J.-K. Hao, Z. Lü, F. Glover, A learning-based path relinking algorithm for the bandwidth coloring problem, Eng. Appl. Artif. Intell. 52 (2016) 81–91.

[22] S.C. Madeira, A.L. Oliveira, Biclustering algorithms for biological data analysis: a survey, IEEE/ACM Trans. Computat. Biol. Bioinformat. (TCBB) 1 (1) (2004) 24–45.
[23] Y. Malgrange, Recherche des sous-matrices premières d'une matrice à coefficients binaires. Applications à certains problèmes de graphe 1962 Gauthier-Villars, Paris 231 242
[24] B. Mirkin, Mathematical Classification and Clustering, Springer, 1996.
[25] P. Orzechowski, K. Boryczko, Propagation-based biclustering algorithm for extracting inclusion-maximal motifs, Comput. Inform. 35 (2) (2016) 391–410.
[26] G. Palubeckis, Multistart tabu search strategies for the unconstrained binary quadratic optimization problem, Ann. Oper. Res. 131 (2004) 259–282.
[27] [27] V. Pareto, Cours d'Économie Politique, 2 volumes, F. Rouge, Éditeur, Lausanne, 1896-1897.
[28] G.A. Pavlopoulos, P.I. Kontou, A. Pavlopoulou, C. Bouyioukos, E. Markou, P. Bagos, Bipartite graphs in systems biology and medicine: a survey of methods and applications, GigaScience 7 (4) (2018) giy014.
[29] R. Peeters, The maximum edge biclique problem is NP-complete, Discrete Appl. Math. 131 (3) (2003) 651–654.
[30] A. Prelić, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, E. Zitzler, A systematic comparison and evaluation of biclustering methods for gene expression data, Bioinformatics 22 (9) (2006) 1122–1129.
[31] K. Seridi, L. Jourdan, E.G. Talbi, Using multiobjective optimization for biclustering microarray data, Appl. Soft Comput. 33 (2015) 239–249.
[32] A. Tanay, R. Sharan, R. Shamir, Biclustering algorithms: A survey, Handbook of Computational, Mol. Biol. 9 (1–20) (2005) 122–124.
[33] B. Wang, Y. Miao, H. Zhao, J. Jin, Y. Chen, A biclustering-based method for market segmentation using customer pain points, Eng. Appl. Artif. Intell. 47 (2016) 101–109.
[34] Y. Wang, Z. Lu, F. Glover, J.-K. Hao, Path relinking for unconstrained binary quadratic programming, Eur. J. Oper. Res. 223 (3) (2012) 595–604.
[35] Y. Wang, Q. Wu, F. Glover, Effective metaheuristic algorithms for the minimum differential dispersion problem, Eur. J. Oper. Res. 258 (3) (2017) 829–843.
[36] Y. Wang, Q. Wu, A. Punnen, F. Glover, Adaptive tabu search with strategic oscillation for the bipartite boolean quadratic programming problem with partitioned variables, Inf. Sci. 450 (2018) 284–300.
[37] H. Zha, X. He, C. Ding, H. Simon, M. Gu, Bipartite graph partitioning and data clustering, In: Proceedings of the tenth International Conference on Information and Knowledge Management, ACM, 2001, pp. 25–32.
[38] X. Zhu, J. Qiu, M. Xie, J. Wang, A multi-objective biclustering algorithm based on fuzzy mathematics, Neurocomputing 253 (2017) 177–182.