Journal of Classification 24:251-275 (2007)
DOI: 10.1007/s00357-007-0014-x

Algorithms for ℓ_1-Embeddability and Related Problems

Pierre Hansen

GERAD and HEC, Montréal

Sylvain Perron

GERAD and HEC, Montréal

Abstract: Assouad has shown that a real-valued distance $d = (d_{ij})_{1 \leq i < j \leq n}$ is isometrically embeddable in ℓ_1-space if and only if it belongs to the cut cone on n points. Determining if this condition holds is NP-complete. We use Assouad's result in a constructive column generation algorithm for ℓ_1-embeddability. The subproblem is an unconstrained 0-1 quadratic program, solved by *Tabu Search* and *Variable Neighborhood Search* heuristics as well as by an exact enumerative algorithm. Computational results are reported. Several ways to approximate a distance which is not ℓ_1-embeddable by another one which is are also studied.

Key words: ℓ_1-space; Embedding; Cut cone; Column generation.

Work of the first author was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) grant 105574-02 and FCAR (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) grant 2002-ER-73226. Work of the second author has been supported by NSERC graduate scholarship 195113 and FCAR graduate scholarship 67567. We thank Bernard Fichet for useful discussions. We thank also Christophe Meyer for the use of his implementation of the algorithm presented in Hansen et al. (2000).

Authors' Address: Pierre Hansen and Sylvain Perron, GERAD and HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal (Québec), Canada H3T 2A7; e-mails: pierre.hansen@gerad.ca; sylvain.perron@gerad.ca

1. Introduction

The isometric embedding problem is to find if a given distance space (X, d) can be embedded into a prescribed space, i.e., if there is a mapping from the first space to the second one which preserves distances. This problem has been extensively studied since the middle of the XIXth century (see Deza and Laurent 1997 for references). In this paper, we focus on algorithmic aspects of ℓ_1-embeddability, i.e., isometrically embedding a finite distance space (X, d) into the ℓ_1-space, or space equipped with the ℓ_1-norm.

Important connections have been made between ℓ_1-embeddability and basic problems of combinatorial optimization and probability theory. Assouad (1979-1980) has shown that a real-valued distance $d = (d_{ij})_{1 \leq i < j \leq n}$ is ℓ_1-embeddable if and only if it belongs to the *cut cone* CUT_n on n points. Determining if this condition is satisfied is NP-complete (Avis and Deza 1991). Characterizing ℓ_1-embeddable distance spaces (X, d) amounts to finding all facets of the cut cone CUT_n. While many families of such facets are known (see again Deza and Laurent 1997) and all of them for $n \leq 7$ (Grishukhin 1990), it is very unlikely that they can all be found for general n. Indeed, a consequence of a result of Karp and Papadimitriou (1982) is that there is no polynomially concise way of describing a list of inequalities sufficient to describe CUT_n unless NP=C_o-NP.

Avis (1977) has shown that d belongs to CUT_n if and only if there exists a measure space (Ω, A, μ) and events $A_1, A_2, \ldots, A_n \in A$ such that $d_{ij} = \mu(A_i \cap A_j)$ for all $1 \leq i < j \leq n$.

As a consequence of these connections, many equivalent results on ℓ_1-embeddability and its application have been obtained in different fields. An extensive and detailed survey and synthesis of these results, and many others, is given in the recent book on *Geometry of Cuts and Metrics* by Deza and Laurent (1997). Among applications, we mention the quadratic case (Georgakopoulos, Kavvadias, and Papadimitriou 1988; Kavvadias and Papadimitriou 1990) of Boole's probabilistic satisfiability problem (Boole 1854a;b;c; Hansen and Jaumard 2000) and principal component analysis in ℓ_1-norm (Benayade and Fichet 1994).

Assouad's condition for ℓ_1-embeddability leads to a linear program with a number of columns exponential in the input size; these columns correspond to all cuts. We consider again this problem and apply the powerful column generation technique of linear programming (Gilmore and Gomory 1961; 1963; Chvátal 1983). Then the vast majority of columns are kept implicit. The entering column is determined by solving a subproblem, which is an unconstrained quadratic program in 0–1 variables. Both heuristics, of *Tabu Search* and *Variable Neighborhood Search* type, and an exact enumerative algorithm are used for that purpose. Computational results are reported.

If a distance space (X, d) is not ℓ_1-embeddable a natural question is how to approximate it by another one which has this property. We consider several ways to do so, i.e., the *additive constant problem* in which all distances are increased by the same constant (Fichet 1987; 1994; Critchley 1980) as well as *lower, upper* and *general* ℓ_1*-approximation* (Fichet 1994). Observe that these last problems are close to multidimensional scaling in ℓ_1-norm, for which several heuristic methods have been proposed (e.g., Hubert, Arabie, and Hesson-McInnis 1992; Groenen, Heiser, and Meulman 1998; Brusco 2001).

The paper is organized as follows. Basic definitions and results on distance space, ℓ_1-embeddability and the cut cone are recalled in the next section. Problems studied are stated in Section 3. The algorithm is presented in Section 4 and detailed computational results are reported in Section 5.

2. Definitions and Previous Results

In this section, we summarize results from chapters 3 and 4 of Deza and Laurent (1997).

2.1 Distance Spaces

Consider a set X. A *distance* on X is a function $d : X \times X \to \mathbb{R}$ which is *non-negative*, i.e., $d(i, j) \geq 0$ for all $i, j \in X$, *symmetric*, i.e., $d(i, j) = d(j, i)$ for all $i, j \in X$ and such that $d(i, i) = 0$ for all $i \in X$. Then, (X, d) is called a *distance space*. A *semimetric* is a distance which satisfies the triangular inequalities

$$d(i, j) \leq d(i, k) + d(k, j) \tag{1}$$

for all $i, j, k \in X$. A *metric* is a semimetric such that $d(i, j) = 0 \Rightarrow i = j$ for all $i, j \in X$.

Let $V_n = \{1, 2, \ldots, n\}$ and $E_n = \{ij \mid i, j \in V_n, i \neq j\}$ where $ij = ji$ denotes the unordered pair of integers i and j. A distance d on V_n can be viewed as a vector $(d_{ij})_{1 \leq i < j \leq n} \in \mathbb{R}^{E_n}$; conversely every non-negative vector $d \in \mathbb{R}^{E_n}$ yields a distance d on V_n by symmetry and taking 0 for diagonal pairs; its *distance matrix* is $D = (d_{ij})$.

Recall that a *norm* on a vector space E is a function $x \in E \mapsto \| x \| \in \mathbb{R}_+$ such that

(i) $\| x \| = 0$ if and only if $x = 0$;

(ii) $\| \lambda x \| = |\lambda| \, \| x \|$ for $\lambda \in \mathbb{R}, x \in E$;

(iii) $\| x + y \| \leq \| x \| + \| y \|$.

Given a normed space $(E, \| \ . \ \|)$, a *norm metric* or *Minkowsky metric* is obtained by setting

$$d_{\|.\|}(x, y) := \| \ x - y \ \| \ .$$

For any $p \geq 1$ the $\ell_p - metric \ d_{\ell_p}$ is obtained by endowing $\mathbb{R}_m$ with the $\ell_p - norm$

$$\| \ x \ \|_p = (\sum_{1 \leq k \leq m} |x_k|^p)^{1/p}$$

for $x \in \mathbb{R}^m$.

The $\ell_1 - metric \ d_{\ell_1}$ is the particular case where $p = 1$, i.e.,

$$\| \ x - y \ \|_1 = \sum_{1 \leq k \leq m} |x_k - y_k|$$

for $x, y \in \mathbb{R}^m$. Note that the $\ell_1 - metric$ is also called *rectilinear metric*, *Manhattan metric* or *taxi-cab metric*.

2.2 Cuts and Cut-cones

Let $S \subseteq V_n$ and $\delta(S)$ denote the distance on V_n taking value 1 on the pairs (i, j) such that $| \ S \cap \{i, j\} \ | = 1$ and value 0 otherwise. It is easy to see that $\delta(S)$ is a semimetric, but not a metric if $n \geq 3$; it is called a *cut semimetric*.

The *cut cone*, denoted CUT_n, is the cone generated by the cut semimetrics $\delta(S)$ for $S \subseteq V_n$

$$\text{CUT}_n = \{ \sum_{S \subseteq V_n} \lambda_S \delta(S) \mid \lambda_S \geq 0 \text{ for all } S \subseteq V_n \}.$$

2.3 Isometric Embeddings

Consider two distance spaces (X, d) and (X', d'). Then (X, d) is *isometrically embeddable* into (X', d') if there exists a mapping Φ, called *isometric embedding* from X to X' such that

$$d(x, y) = d'(\Phi(x), \Phi(y))$$

for all $x, y \in X$, i.e., such that distances between all pair of points are preserved. When so, (X, d) is called an isometric subspace of (X', d').

Let d be a distance on V_n. Any decomposition

$$d = \sum_{S \subseteq V_n} \lambda_S \delta(S)$$

where $\lambda_S \geq 0$ for all S is called a $\mathbb{R}_+$*-realization* of d.

Any distance $d \in \text{CUT}_n$ satisfies the triangle inequalities (1) and is a semimetric.

The result of Assouad (1979-1980) mentioned in the introduction can be expressed as follows:

Proposition. *Let $d \in I\!R^{E_n}$ and (V_n, d) be its associated distance space. The following assertions are equivalent.*

(i) $d \in \text{CUT}_n$.

(ii) (V_n, d) *is ℓ_1-embeddable, i.e., there exist n vectors $u_1, u_2, \ldots, u_n \in \mathbb{R}^m$ (for some m) such that $d_{ij} = \| u_i - u_j \|_1$ for all $1 \leq i \leq j \leq n$.*

Let (V_n, d) be a distance space. Then, (V_n, d) is said to be ℓ_1-*rigid* if d admits a unique $\mathbb{R}_+$-*realization*.

Suppose d is a distance on V_n. If $\sum_{\emptyset \neq S \subset V_n} \lambda_S \delta(S)$ is a decomposition of d as a linear combination of nonzero cut semimetrics, then the quantity $\sum_{\emptyset \neq S \subset V_n} \lambda_S$ is called its *size*. Moreover, if d is ℓ_1-embeddable then the quantity

$$\min_{\lambda}\Big(\sum_{\emptyset \neq S \subset V_n} \lambda_S \mid d = \sum_{\emptyset \neq S \subset V_n} \lambda_S \delta(S) \text{ with } \lambda_S \geq 0 \text{ for all } S \subset V_n \Big) \quad (2)$$

is called the *minimum ℓ_1-size* of d.

3. Problems

3.1 Feasibility Problem

In this section, we recall the mathematical formulations of problems studied by Deza and Laurent (1997) and Fichet (1987; 1994). As mentioned by Fichet (1987; 1994), proving that (V_n, d) is ℓ_1-*embeddable* can be done constructively, i.e., by proving that the following linear program has at least one solution

$$\begin{aligned} \sum_{S^t \in H} \delta(S^t)_{ij} \cdot \lambda_t &= d_{ij} \qquad \forall 1 \leq i < j \leq n \\ \lambda_t &\geq 0 \qquad \forall S^t \in H \end{aligned} \quad (3)$$

where $H = \{S^1, S^2, \ldots, S^{2^{n-1}-1}\}$ is the set of all non-empty subsets of V_n containing at most $\lfloor \frac{n}{2} \rfloor$ points and $\delta(S^t)_{ij}$ is equal to 1 if $\mid S^t \cap \{i, j\} \mid = 1$ and 0 otherwise. Problem (3) may be solved by applying phase 1 of the simplex or revised simplex algorithm (see e.g. Chvátal 1983), i.e., by solving the following equivalent linear program obtained by adding artificial variables α_{ij}

$$\begin{aligned}
\min_{\alpha,\lambda} \quad & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \alpha_{ij} \\
\text{s.t.} \quad & \sum_{S^t \in H} \delta(S^t)_{ij} \cdot \lambda_t + \alpha_{ij} = d_{ij} \qquad \forall 1 \leq i < j \leq n \\
& \lambda_t \geq 0 \qquad \forall S^t \in H \\
& \alpha_{ij} \geq 0 \qquad \forall 1 \leq i < j \leq n.
\end{aligned} \tag{4}$$

The linear program (3) admits a solution if and only if the minimum value of (4) is equal to 0.

Example 1 Consider the distance d on the set $V_4 = \{1, 2, 3, 4\}$ obtained with Rao's dissimilarity index (Fichet and Calvé 1984; Joly and Le Calvé 1994) expressed by the following distance matrix

$$D = \begin{pmatrix} 0 & 0.5 & 0.5 & 1.0 \\ 0.5 & 0 & 1.0 & 1.0 \\ 0.5 & 1.0 & 0 & 1.0 \\ 1.0 & 1.0 & 1.0 & 0 \end{pmatrix}$$

Then consider all the non-empty subsets of V_4 containing at most 2 points: $S^1 = \{1\}, S^2 = \{2\}, S^3 = \{3\}, S^4 = \{4\}, S^5 = \{1, 2\}, S^6 = \{1, 3\}$ and $S^7 = \{1, 4\}$. The cuts $\delta(S^t)$ over these subsets correspond to the following vectors

$\delta(S^1)$	$\delta(S^2)$	$\delta(S^3)$	$\delta(S^4)$	$\delta(S^5)$	$\delta(S^6)$	$\delta(S^7)$
1	1	0	0	0	1	1
1	0	1	0	1	0	1
1	0	0	1	1	1	0
0	1	1	0	1	1	0
0	1	0	1	1	0	1
0	0	1	1	0	1	1

Verifying if $d \in \text{CUT}_4$ amounts to checking that the following linear program admits at least one solution:

$$\begin{array}{ccccccccccccccc}
\lambda_1 & + & \lambda_2 & & & & & & & + & \lambda_6 & + & \lambda_7 & = & 0.5 \\
\lambda_1 & & & + & \lambda_3 & & & + & \lambda_5 & & & + & \lambda_7 & = & 0.5 \\
\lambda_1 & & & & & + & \lambda_4 & + & \lambda_5 & + & \lambda_6 & & & = & 1.0 \\
& & \lambda_2 & + & \lambda_3 & & & + & \lambda_5 & + & \lambda_6 & & & = & 1.0 \\
& & \lambda_2 & & & + & \lambda_4 & + & \lambda_5 & & & + & \lambda_7 & = & 1.0 \\
& & & & \lambda_3 & + & \lambda_4 & & & + & \lambda_6 & + & \lambda_7 & = & 1.0 \\
\lambda_1 & , & \lambda_2 & , & \lambda_3 & , & \lambda_4 & , & \lambda_5 & , & \lambda_6 & , & \lambda_7 & \geq & 0\,.
\end{array}$$

A solution of this linear program is

$$\lambda_4 = 0.5, \lambda_2 = \lambda_3 = \lambda_5 = \lambda_6 = 0.25, \lambda_1 = \lambda_7 = 0.$$

By taking, for each element of V_4, a vector u_i with components $(u_i)_t = \lambda_t$ if $i \in S^t$ and $(u_i)_t = 0$ otherwise, we obtain a corresponding ℓ_1-embedding in a 7-dimensional space:

$$u_1 = (0, 0, 0, 0, 0.25, 0.25, 0),\ u_2 = (0, 0.25, 0, 0, 0.25, 0, 0),$$
$$u_3 = (0, 0, 0.25, 0, 0, 0.25, 0),\ u_4 = (0, 0, 0, 0.5, 0, 0, 0).$$

Eliminating the dimension t for which $\lambda_t = 0$, we get an embedding in a 5-dimensional space:

$$u_1 = (0, 0, 0, 0.25, 0.25),\ u_2 = (0.25, 0, 0, 0.25, 0),$$
$$u_3 = (0, 0.25, 0, 0, 0.25),\ u_4 = (0, 0, 0.5, 0, 0)$$

and, applying lemma 11.1.3 of Deza and Laurent (1997), we find an embedding in a 2-dimensional space:

$$u_1 = (0.25, 0.25), u_2 = (0, 0), u_3 = (0.5, 0.5), u_4 = (1, 0).$$

Then for instance, d_{12} is given by:

$$d_{12} = \| u_1 - u_2 \|_1 = |0.25 - 0| + |0.25 - 0| = 0.5.$$

■

3.2 Optimization Problems

When the linear program (3) has a feasible solution, i.e., when distance d is ℓ_1-embeddable, one can be interested in finding an optimal solution with respect to a specific criterion or, in other words, choosing a best ℓ_1-embedding in some sense. An objective function is then added to (3). The linear program becomes:

$$\begin{aligned} \min_{\lambda} \quad & \sum_{S^t \in H} c_t \cdot \lambda_t \\ \text{s.t.} \quad & \\ & \sum_{S^t \in H} \delta(S^t)_{ij} \cdot \lambda_t = d_{ij} \qquad \forall 1 \le i < j \le n \\ & \lambda_t \ge 0 \qquad \forall S^t \in H \end{aligned} \tag{5}$$

where the c_t express the desired property. When $c_t = 1$ for all t problem (5) reduces to problem (2), hence its optimal value gives the *minimum* ℓ_1*-size* of

d. Benayade and Fichet (1994) propose a criterion which is useful in *Principal Component Analysis in ℓ_1-norm*: choose

$$c_t = \min\{|S^t|, |V_n \setminus S^t|\} \text{ for all } t$$

in order to find the minimum of the ℓ_1-median criterion over all ℓ_1-embeddings of (V_n, d).

When the linear program (3) has a unique solution, (V_n, d) is ℓ_1-*rigid* as mentioned above. This can be checked as follows: solve (3), let (λ_t^*) denote the solution found and

$$T^+ = \{t \mid \lambda_t^* > 0\}\,.$$

If this solution is not unique, there exists a solution with at least one variable λ_t with $t \notin T^+$ strictly positive, and as all coefficients in the constraints of (3) are positive, at least one variable $\lambda_t < \lambda_t^*$ with $t \in T^+$. So one may consider in turn $|T^+|$ linear programs with

$$\min_{\lambda} \lambda_t$$

for each $t \in T^+$ as objective function and stop as soon as a value different from λ_t^* is found. If this does not happen (V_n, d) is ℓ_1-*rigid*.

3.3 Approximation Problems

When the linear program (3) has no solution, one can be interested in finding how to modify d into d' in such a way that (V_n, d') is ℓ_1-*embeddable*. This amounts to approximating a given dissimilarity by a semi-distance of ℓ_1-type.

A first type of approximation corresponds to the *general additive constant problem*, expressed by the following linear program:

$$\begin{aligned} \min_{c,\lambda} \quad & c & & \\ \text{s.t.} \quad & & & \\ & \sum_{S^t \in H} \delta(S^t)_{ij} \cdot \lambda_t - c = d_{ij} & \forall 1 \le i < j \le n & \qquad (6) \\ & \lambda_t \ge 0 & \forall S^t \in H & \\ & c \ge 0. & & \end{aligned}$$

As observed by Critchley (1980), the linear program (6) always admits a feasible solution. The next three approximations are least absolute deviation approximations. Solving problem (4) above, we get a *lower ℓ_1-norm approximation*.

Similarly, an *upper ℓ_1-norm approximation* is obtained by solving the following linear program:

$$\begin{array}{ll} \min\limits_{\beta,\lambda} & \displaystyle\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\beta_{ij} \\ \text{s.t.} & \\ & \displaystyle\sum_{S^t\in H}\delta(S^t)_{ij}\cdot\lambda_t-\beta_{ij}=d_{ij} \qquad \forall 1\le i<j\le n \\ & \lambda_t\ge 0 \qquad \forall S^t\in H \\ & \beta_{ij}\ge 0 \qquad \forall 1\le i<j\le n. \end{array} \tag{7}$$

Finally, a *lower-upper ℓ_1-norm approximation* is obtained by solving a last linear program:

$$\begin{array}{ll} \min\limits_{\beta,\alpha,\lambda} & \displaystyle\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\beta_{ij}+\alpha_{ij}) \\ \text{s.t.} & \\ & \displaystyle\sum_{S^t\in H}\delta(S^t)_{ij}\cdot\lambda_t-\beta_{ij}+\alpha_{ij}=d_{ij} \qquad \forall 1\le i<j\le n \\ & \lambda_t\ge 0 \qquad \forall S^t\in H \\ & \beta_{ij},\alpha_{ij}\ge 0 \qquad \forall 1\le i<j\le n. \end{array} \tag{8}$$

4. Algorithm

4.1 General Method

Solving any of the linear programs (3) to (8) by a simplex-based algorithm presents two major difficulties: (i) the enormous number of columns; (ii) the fact that at each iteration, deciding whether the algorithm should stop or not implies solving an unconstrained quadratic 0–1 program which is difficult in practice (see Section 4.3 below). These programs contain $2^{n-1}-1$ columns and, unless n is small, this number is much too large just to write them down explicitly. However, each of these programs can be solved exactly by the *column generation* technique of linear programming (Gilmore and Gomory 1961; 1963; Chvátal 1983). Then two programs are associated with the original linear program: on the one hand, the *master problem* which is identical to the original program itself but with only a small number of explicit columns, and on the other hand the *subproblem*, whose role is to determine the entering column, as in the simplex or revised simplex algorithm (Chvátal 1983). A specific optimization problem must be solved for that purpose. Once the entering column

is determined, its expression in the current master problem is calculated and a simplex iteration takes place.

4.2 Formulation of the Subproblem

We first consider the subproblem of problem (5) with $c_t = \min(|S^t|, |V_n \setminus S^t|)$. This subproblem is to compute the smallest reduced cost, i.e., solve

$$\min_{t \in N} c_t - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta(S^t)_{ij} \cdot v_{ij} \tag{9}$$

where N is the set of non-basic variables and v_{ij} is the dual variable of the constraint associated with d_{ij}. By solving the problem (9) exactly, we can find the non-basic variable which has the smallest reduced cost without considering explicitly the whole set of variables. If the optimal value of problem (9) is nonnegative, it means that the current basic solution of the master problem is optimal for the complete problem (5). Otherwise, the column associated with the non-basic variable λ_t of the optimal solution of problem (5) is added to the master problem, which is re-optimized.

Let $x_i = 1$ if $i \in S^t$ and 0 otherwise. Then problem (9) can be expressed as:

$$\begin{aligned} \min_x \sum_{i=1}^{n} x_i - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} v_{ij} \cdot [x_i(1-x_j) + x_j(1-x_i)] \\ x_i \in \{0,1\} \qquad i = 1, 2, \ldots, n. \end{aligned} \tag{10}$$

or, after easy simplifications:

$$\begin{aligned} \min_x f(X) = \sum_{i=1}^{n} \alpha_i \cdot x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 \cdot v_{ij} \cdot x_i \cdot x_j \\ x_i \in \{0,1\} \qquad i = 1, 2, \ldots, n \end{aligned} \tag{11}$$

where $\alpha_i = 1 - \sum_{j=1}^{i-1} v_{ji} - \sum_{j=i+1}^{n} v_{ij}$. This is an unconstrained quadratic 0–1 programs.

It expresses (9) as: (a) $\delta(S^t)_{ij} = 1$ if and only if x_i and x_j take complementary values, i.e., $x_i(1-x_j) + x_j(1-x_i) = 1$ and this is symmetric in i and j; (b) as the first term in the objective is $\sum_{i=1}^{n} x_i$, the solution corresponding to a given cut will always be that one with the smallest number of $x_i = 1$, and not the complementary one.

Observe further that problem (9) with $c_t = 0$ consist in solving a MAX CUT problem in $G(V_n, E_n)$ with real-valued weights on each edge ij of E_n equal to v_{ij}.

4.3 Solution of the Subproblem

Problem (11) must be solved at each iteration of the column generation method and may be time-consuming. Indeed, the unconstrained quadratic 0–1 programming problem is NP-hard in the general case since the NP-hard (Garey and Johnson 1979) MAX CUT problem can be easily expressed in that form. Whether it remains NP-hard for problem (11) is an open question. However, for guaranteeing convergence it is not mandatory to solve (11) exactly at all iterations. A heuristic method may be applied as long as it gives a negative reduced cost. Indeed, an iteration of the simplex algorithm can take place by entering any column with a negative reduced cost (not necessarily the column with the minimum reduced cost). The use of such a heuristic is important in the solution of large instances since, when the number of 0–1 variables is large, using an exact method can be very time-consuming. For problem (5) and for infeasible instances of problem (3) an exact algorithm must be applied to the subproblem at least once in order to prove that there is no more column with a reduced cost of the desired sign when the heuristic fails to find such a column, i.e., to prove that the optimality conditions are satisfied.

4.3.1 Heuristic Method

Various heuristics can be used to solve the subproblem, i.e., to minimize a quadratic function in 0–1 variables. Recent heuristics include Alkhamis, Hasan, and Ahmed (1998); Glover, Kochenberger, and Alidaee (1998); Hasan, Alkhamis, and Ali (2000); Lodi, Allemand, and Liebling (1999); Palubeckis (1995); Zhou, Wang, Tian, and Guo (1997). Two metaheuristics, or frameworks to build heuristics, have been implemented in our program to solve (11): *Tabu Search* (Glover and Laguna 1997) and *Variable Neighborhood Search* (VNS) (Hansen and Mladenović 2003; Mladenović and Hansen 1997).

Tabu Search fully exploits gradient information while still providing a way to get out of local minima. When minimizing a function, from an initial solution, a steepest descent method is applied until a local minimum is reached. In order to get out of this local minimum, the method allows increasing the value of the function by applying a move of mildest ascent. In order to prevent cycling, when a move of ascent is applied, the backtracking move is declared tabu, i.e., forbidden for some iterations. Figure 1 presents the steps of our implementation of Tabu Search. It differs from usual ones in that one seeks several solutions with a good value and not only a single one with the best or near-best value. The set of neighbors $N(X_0)$ of a solution X_0 is the set of all elementary moves not currently forbidden. An elementary move is a complementation of one variable in the 0-1 vector defining the current solution of problem (11). The step 2.2.1 is done by finding the non-forbidden elementary move with partial derivative

1 Initialization:
- 1.1 Let $L = \emptyset$ be the set of solutions with negative value;
- 1.2 Set $t_i = 0$ for $i = 1, \ldots, n$;
- 1.3 Choose the length γ of the Tabu list;
- 1.4 Select an initial solution X_0:
  - 1.4.1 $f_{opt} = f(X_0)$; $X_{opt} = X_0$;
  - 1.4.2 If $f(X_0) < 0$ then add X_0 to L;
  - 1.4.3 Initialize the partial derivatives;

2 Repeat the following until $f'_{opt} = f_{opt}$:
- 2.1 $f'_{opt} = f_{opt}$;
- 2.2 Repeat the following steps until the stopping condition is met:
  - 2.2.1 Select $X_k \in N(X_0)$ such that $\Delta_k = f(X_k) - f(X_0) = \min_{i|t_i=0} \Delta_i$;
  - 2.2.2 $X_0 = X_k$; Update the partial derivatives;
  - 2.2.3 If $f(X_k) < f_{opt}$ then $f_{opt} = f(X_0)$; $x_{opt} = X_0$; endif;
  - 2.2.4 If $f(X_0) < 0$ then add X_0 to L;
  - 2.2.5 If $\Delta_k > 0$ then $t_k = \gamma$;
  - 2.2.6 Set $t_i = t_i - 1$ for $t_i > 0$, $i = 1, 2, \ldots, n$.

Figure 1. Steps of the Tabu Search heuristic for the subproblem

$$\Delta_i = \alpha_i + \sum_{j=1, j\neq i}^{n} 2 \cdot v_{ij} \cdot x_j$$

of minimum value. The variables t_i give the remaining number of iterations for which variable x_i will remain tabu. X_k denotes the vector obtained from X_0 by complementing x_k.

The second heuristic we implemented to solve (11) is a version of the basic VNS presented in Hansen and Mladenović (2003); Mladenović and Hansen (1997). VNS is a metaheuristic based on the idea of systematic change of neighborhood during the search. VNS explores close and then increasingly far neighborhoods of the incumbent (or best known) solution in a probabilistic way. Therefore, often favorable characteristics of the incumbent solution will be kept and used to obtain promising neighboring solutions. VNS applies a local search routine repeatedly to get from these neighboring solutions to local optima. Figure 2 presents the steps of our implementation of VNS. The set of solutions in the k^{th} neighborhood of a solution is obtained by applying k complementation

1 Initialization:
  1.1 Let $L = \emptyset$ be the set of solutions with negative value;
  1.2 Select an initial solution X: If $f(X) < 0$ then add X to L;

2 Repeat the following until the stopping condition is met:
  2.1 Set $k \leftarrow 1$;
  2.2 Until $k = k_{max}$, repeat the following steps:
    2.2.1 *shaking*. Generate a vector X' at random by complementing k variables of X;
    2.2.2 *local search*. Apply a steepest descent method with X' as initial solution; denote with X'' the so obtained local optimum; if $f(X'') < 0$ and $X'' \notin L$ then add X'' to L;
    2.2.3 *move or not*. If $f(X'') < f(X)$, then move there ($X \leftarrow X''$), and continue the search with $k = 1$; otherwise, set $k \leftarrow k + 1$.

Figure 2. Steps of the VNS heuristic for the subproblem

on the 0-1 vector defining a solution of problem (11). The move used during the local search phase (step 2.2.2) is complementation of the variable with the most negative partial derivative Δ_i.

Our implementations of Tabu Search and VNS exploit the fact that in our column generation algorithm, we wish to generate many columns with negative reduced cost at each iteration. Then, many vectors X with $f(X) < 0$ have to be memorized. During our experiments with VNS, we realized that memorizing only local optima of (11) with negative value gave better results than memorizing all the solutions with negative value. This may be due to the fact that the columns obtained in the former way differ more than those obtained in the latter. The list L obtained at the end of the algorithm gives the columns to add to the master program.

Two stopping conditions are used simultaneously, in both heuristics: limits are imposed on the maximum number of iterations and on the maximum cardinality of L. The heuristic stops when any one of these limits is reached.

An adaptation of the formulas presented in Hansen, Jaumard, and Silva (1991) has been implemented in order to update efficiently the partial derivatives Δ_i during the step 2.2.2 of both heuristics. Updating of each partial derivative can be done in constant time. Then each iteration of the local search takes $O(n)$ time.

Results for problem (3) presented in Tables 1 and 3 show that VNS is more efficient than Tabu Search in our column generation algorithm. Therefore we limited ourselves to VNS in further variants of our program.

Table 1. Results on feasibility problems with Tabu Search as heuristic for the subproblem's solution

Prob. size	Complete algorithm			Heuristic		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ
10	0.021	0.007	5.8	0.011	1.9	0.010	3.0
15	0.128	0.029	31.9	0.058	2.7	0.063	3.7
20	0.542	0.122	69.5	0.156	2.6	0.380	3.6
25	2.263	0.433	159.8	0.332	2.5	1.917	3.5
30	7.341	1.463	245.9	0.452	2.6	6.859	3.6
35	18.296	3.283	307.2	0.316	3.1	17.939	4.1
40	38.120	6.510	335.0	0.228	3.4	37.832	4.3
45	78.027	11.814	395.0	0.231	4.0	77.706	5.0
50	152.930	23.261	420.0	0.281	4.2	152.519	5.2
55	252.193	39.151	455.0	0.308	4.5	251.723	5.5
60	499.365	84.642	500.0	0.379	5.0	498.753	6.0
65	849.762	123.311	560.0	0.483	5.6	848.985	6.6

4.3.2 Exact Method

In order to solve the subproblem exactly, we use a recent and simple algorithm (Hansen, Jaumard, and Meyer 2000) for unconstrained quadratic 0–1 programming which exploits systematically first order derivatives within a branch-and-bound scheme (for related methods, see e.g. Hammer and Hansen (1981); Hammer and Rubin (1970); Pardalos and Rodgers (1990)). To enhance its performance, the objective function is first expressed as a *quadratic posiform*, i.e., a function of the x_j and their complements $\overline{x}_j$ with positive coefficients and largest possible fixed term (Hammer, Hansen, and Simeone 1984).

4.4 Solution of the Master Program

The linear programming part of our algorithm relies upon CPLEX 7.0 and uses its capacity to add columns.

4.5 Improvements

A first improvement to the general column generation algorithm is to allow introducing many columns at each iteration. This technique is called *multiple pricing* (*MP*). Experiments show that when only one column is added at each iteration, the objective function decreases very slowly. Applying *MP* reduces this number of iterations and speeds up the algorithm as there are less calls to CPLEX. As shown in the next section, the time used by CPLEX is the main component of the overall time taken by the algorithm. Table 2 shows that

Table 2. Results on optimization problems with $c_t = \min(|S^t|, |V_n \setminus S^t|)$

Solution method	Prob. size	Complete algorithm			Heuristic		CPLEX 7.0	
		CPU time		Nb cols	CPU time	Calls	CPU time	Calls
	n	μ	σ	μ	μ	μ	μ	μ
General algo.	15	21.285	1.019	457.4	11.786	458.4	9.431	459.4
Gen. + MP	15	4.741	0.249	560.2	1.393	55.5	3.321	56.5
Gen. + HS	15	1.178	0.523	36.4	0.869	37.4	0.295	38.4
Improved algo.	15	0.220	0.058	89.5	0.059	6.8	0.148	7.8

applying *MP* on 20 problems with 15 points reduces the mean computing time by a factor of 4.5.

A second improvement consists in inserting well-chosen columns in the initial master program. This technique is called *hot start* (*HS*). By analyzing the columns in the optimal basis of many problems (5) with $c_t = \min(|S^t|, |V_n \setminus S^t|)$, we found that most of these columns have the same structure: they are associated with 1-dichotomies and 2-dichotomies. Introducing all such columns in the initial master program reduces the number of iterations substantially; computing times decrease accordingly. Comparing the first and third line of Table 2 clearly illustrates the impact of this method: applying *HS* on 20 problems with 15 points reduced the mean computing time by a factor of 18. For the *minimum ℓ_1-size problem*, results show that many columns are associated with $\frac{n}{2}$-dichotomies in the optimal basis. As such columns are numerous, only some of them, drawn at random, are used in *HS*.

The last line of Table 2 present results obtained by applying both improvements. For 20 problems with 15 points the mean computing time is reduced by a factor of 96.

5. Numerical Results

There are various techniques to generate a distance d on a set of points. The numerical results presented here have been obtained on problems generated by the 3 following methods:

- Method 1: this method finds d such that (V_n, d) *is* always ℓ_1*-embeddable*. The distance is obtained by generating a set of subsets S^t of V_n, or which is equivalent, of boolean vectors $X^t = \{x_1^t, x_2^t, \ldots, x_n^t\}$ where $x_i^t = 1$ if $i \in S^t$ and 0 otherwise. A random real positive value is associated with each vector and is added to d_{ij}, (which is initially equal to 0) if $\mid S^t \cap \{i, j\} \mid = 1$.
- Method 2: this method finds d by generating a set of n vectors $u_1, u_2, \ldots, u_n \in \mathbb{R}^m$ (for some m) giving the coordinates of n points in a space of

m dimensions. The ℓ_1-norm distance between each pair of points is then computed, i.e., $d_{ij} = \| u_i - u_j \|_1$ for all $1 \leq i \leq j \leq n$.

- Method 3: this method finds a distance d where each d_{ij} is a randomly generated positive real number.

All results presented here have been obtained by the improved column generation algorithm described in the previous section. These results are summarized in Tables 3 to 15. Nbcols denotes the number of columns generated, not including those of *Hot start*. The maximum number of columns added at each iteration is equal to 100. VNS is used for solving the subproblems. The heuristic solution of each subproblem is stopped when 100 columns with a strictly negative reduced cost have been found. Except for *minimum ℓ_1-size problem*, the number of columns of the *hot start* is equal to $n+n(n-1)/2$, i.e., the number of 1-dichotomies and 2-dichotomies. For *minimum ℓ_1-size problem*, only $n(n-1)/2$ columns associated with $\frac{n}{2}$-dichotomies are considered. The algorithm is implemented in C and uses CPLEX 7.0 as linear programming solver. The results have been obtained on a PIII computer with 750 Mhz and 768 MByte of RAM. In each case 20 instances are solved and average values presented, when the total computation time per problem does not exceed 5000 seconds.

5.1 Feasibility Problems

Tables 3, 4 and 5 present results for problems 3 where d is generated by methods 1, 2 and 3 respectively. From Table 3, it appears that:

(i) VNS performs well, as no call to the exact algorithm for the subproblem is needed;
(ii) the proportion of the computing time taken by CPLEX 7.0 augments with n; it attains over 99 % CPU time for the largest problems solved, i.e., for $n = 85$;
(iii) fairly large instances can be solved in reasonable time.

From Table 4, it appears that: conclusions (i) and (ii) obtained for Table 3 are also true here; conclusion (iii) does not carry over: problems solved and generated by method 2 are at most half the size of those solved and generated by method 1; (iv) for a given number n of points, CPU time and number of iterations decrease with the dimension m of the space. Note that problems obtained with method 1 are likely to be embedded in larger spaces than those obtained by method 2.

Problems generated by method 3 were not feasible. From Table 5, it appears that:

Table 3. Results on feasibility problems where d is generated by method 1

Prob. size	Complete algorithm			Heuristic		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ
10	0.017	0.005	5.6	0.006	1.9	0.008	3.0
15	0.089	0.021	28.5	0.022	2.7	0.064	3.7
20	0.415	0.085	58.3	0.048	2.7	0.361	3.7
25	1.849	0.339	124.4	0.085	2.9	1.750	3.9
30	5.868	0.752	199.0	0.137	3.0	5.700	4.0
35	15.973	2.855	275.4	0.189	3.1	15.737	4.2
40	36.385	6.566	323.5	0.221	3.3	36.090	4.3
45	75.181	10.578	374.9	0.239	3.8	74.828	4.8
50	143.217	19.150	417.8	0.268	4.2	142.793	5.2
55	262.894	44.552	460.0	0.282	4.6	262.403	5.6
60	430.769	43.496	500.0	0.309	5.0	430.175	6.0
65	715.600	84.283	530.0	0.331	5.3	714.915	6.3
70	1214.527	127.901	610.0	0.376	6.1	1213.682	7.1
75	1826.263	234.541	630.0	0.424	6.3	1825.263	7.3
80	2678.345	286.468	660.0	0.482	6.6	2677.169	7.6
85	3995.200	502.558	715.0	0.564	7.2	3993.786	8.2

Table 4. Results on feasibility problems where d is generated by method 2

Prob. size		Complete algorithm			Heuristic		CPLEX 7.0	
		CPU time		Nb cols	CPU time	Calls	CPU time	Calls
n	m	μ	σ	μ	μ	μ	μ	μ
10	5	0.077	0.014	34.9	0.052	9.4	0.019	10.4
10	10	0.046	0.012	19.4	0.028	5.0	0.016	6.0
10	20	0.028	0.007	9.2	0.015	3.0	0.011	4.0
15	5	0.959	0.175	257.4	0.287	14.9	0.664	15.9
15	10	0.578	0.086	126.0	0.212	11.2	0.355	12.2
15	20	0.279	0.053	55.4	0.124	6.3	0.151	7.3
20	5	14.213	2.523	1040.3	0.851	18.6	13.327	19.6
20	10	8.131	1.474	518.9	0.634	12.8	7.479	13.8
20	20	2.872	0.401	186.3	0.445	8.8	2.414	9.8
25	5	218.893	55.524	2692.8	1.358	30.7	217.413	31.7
25	10	79.565	8.597	1268.0	0.991	16.4	78.515	17.4
25	20	25.164	4.167	562.9	0.966	9.7	24.165	10.7
30	5	2768.003	924.399	5164.6	1.758	53.0	2765.926	54.0
30	10	541.226	64.234	2483.1	1.282	26.3	539.773	27.3
30	20	157.675	31.135	1055.8	1.323	12.2	156.272	13.2
35	10	3125.744	292.425	4596.6	1.815	46.7	3123.486	47.7
35	20	865.202	106.542	1816.8	1.455	18.8	863.547	19.8
40	20	4065.629	453.256	2984.0	1.626	30.0	4063.623	31.0

Table 5. Results on feasibility problems where d is generated by method 3

Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.067	0.020	15.4	0.051	8.4	0.000	1.0	0.009	8.4
15	0.344	0.084	50.1	0.300	12.5	0.003	1.0	0.032	12.5
20	1.200	0.212	129.2	0.981	15.2	0.003	1.0	0.201	15.2
25	3.051	0.555	265.6	2.267	16.2	0.005	1.0	0.755	16.2
30	6.619	1.074	509.4	4.220	16.1	0.019	1.0	2.328	16.1
35	12.694	1.719	797.7	6.397	17.1	0.067	1.0	6.143	17.1
40	25.551	4.736	1145.2	8.853	18.9	0.211	1.0	16.338	18.9
45	56.133	12.299	1589.2	12.940	23.1	0.991	1.0	41.933	23.1
50	93.913	21.656	1903.0	19.431	26.6	2.881	1.0	71.233	26.6
55	192.461	70.907	2375.9	23.944	31.1	9.043	1.1	158.906	31.1
60	393.606	176.618	2870.1	29.126	35.7	44.006	1.1	319.687	35.7
65	737.001	254.704	3524.2	36.294	42.0	107.839	1.2	591.771	42.0
70	1488.785	725.919	4085.6	53.888	49.6	462.872	1.4	970.532	49.6

(i) VNS still performs well as a single call to the exact algorithm for the subproblem is needed except in a few cases for the larger instances;

(ii) The proportion of the computing time taken by CPLEX 7.0 still augments with n but it is not as dominant: for larger n (i.e., $n \geq 40$), CPLEX 7.0 takes the largest proportion of CPU time but time taken by the exact algorithm increases rapidly: it is about half that of CPLEX 7.0 for $n = 70$ and its rapid increase prohibits solution of larger instances;

(iii) for those instances which can be solved, CPU time is similar to that necessary to solve problems generated by method 1.

An anonymous referee observed that problems generated by method 3 are likely to be infeasible simply because they violate a triangle inequality. We checked that this is indeed the case. This referee further proposed to test the algorithm on instances that satisfy the triangle inequalities but violate another facet of the cut cone CUT_n. We have therefore conducted two more series of experiments with instances which violate a pentagonal or a heptagonal facet (Deza and Laurent 1997, page 447). Results are reported in Table 6 and Table 7 respectively. It appears that these instances are easier to solve than those generated by method 3.

To the best of our knowledge, the only previous algorithm for ℓ_1-embedding proposed in the open literature was to solve (4) as a linear program (Fichet 1987; 1994). Note that in chapter 9 of the unpublished thesis Avis (1977) a column generation algorithm is proposed for that purpose, without an implementation. For completeness, we applied this approach to the same problems

Table 6. Results on feasibility problems where d violates a pentagonal facet

Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.044	0.013	3.9	0.039	4.6	0.001	1.0	0.002	4.6
15	0.240	0.209	12.9	0.220	6.4	0.000	1.0	0.016	6.4
20	0.503	0.219	5.9	0.477	5.0	0.001	1.0	0.019	5.0
25	1.173	0.387	5.5	1.125	5.5	0.000	1.5	0.038	5.5
30	2.260	0.594	4.9	2.173	5.4	0.003	1.9	0.067	5.4
35	4.216	1.124	5.5	4.074	5.8	0.008	2.6	0.110	5.8
40	6.689	2.777	5.5	6.485	5.7	0.017	3.1	0.154	5.7
45	10.276	2.405	5.1	9.995	5.5	0.011	2.4	0.220	5.5
50	15.554	4.017	4.9	15.169	5.6	0.014	2.9	0.310	5.6
55	20.685	4.768	5.2	20.181	5.9	0.027	3.0	0.384	5.9
60	24.251	6.222	5.3	23.602	6.1	0.033	4.0	0.491	6.1
65	24.399	5.857	4.4	23.643	5.0	0.038	3.1	0.568	5.0
70	31.398	9.088	4.7	30.422	5.5	0.051	3.4	0.728	5.5
75	35.285	7.468	4.8	34.093	5.5	0.054	3.5	0.895	5.5
80	44.812	15.494	5.3	43.246	5.9	0.074	3.5	1.187	5.9
85	50.522	14.294	5.2	48.700	5.7	0.074	3.2	1.377	5.7

Table 7. Results on feasibility problems where d violates a heptagonal facet

Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.080	0.038	12.9	0.067	8.1	0.000	1.0	0.011	8.1
15	0.340	0.119	14.8	0.316	9.1	0.001	1.0	0.023	9.1
20	1.200	0.544	50.5	1.083	10.8	0.004	1.0	0.107	10.8
25	1.958	0.872	13.6	1.882	9.0	0.002	1.1	0.063	9.0
30	4.199	1.831	14.2	4.052	10.1	0.003	1.4	0.127	10.1
35	9.173	4.010	18.9	8.884	12.7	0.011	3.1	0.241	12.7
40	17.278	8.000	17.8	16.799	15.2	0.017	9.2	0.390	15.2
45	27.038	11.106	20.4	26.357	14.9	0.043	6.1	0.553	14.9
50	41.731	18.226	21.7	40.775	15.5	0.062	7.2	0.790	15.5
55	52.104	23.724	18.4	50.968	15.4	0.079	8.0	0.928	15.4
60	64.691	30.691	19.0	63.044	16.5	0.140	10.8	1.336	16.5
65	62.023	31.403	17.5	60.255	13.1	0.119	7.3	1.446	13.1
70	100.354	51.450	21.5	97.506	18.2	0.260	12.6	2.301	18.2
75	39.113	21.084	20.4	38.575	15.4	0.067	9.1	0.400	15.4
80	42.590	19.324	18.1	441.951	15.2	0.090	10.9	0.463	15.2
85	146.775	59.443	22.0	4141.622	18.1	0.447	13.2	4.205	18.1

as solved in Table 3. The results are presented in Table 8. As expected computation times rise quickly and are larger than those for the column generation algorithm when $n \geq 15$.

5.2 Optimization Problems

It appears that optimization problems, which imply using both phases of the simplex algorithm, are harder to solve than feasibility problems of similar size.

Results for optimization problems with the minimum of the ℓ_1-median criterion are presented in Tables 9 and 10. It appears that again VNS performs well and that for the largest instances, most of the computing time is taken up by CPLEX 7.0.

Results for optimization problems with the minimum ℓ_1-size criterion are presented in Tables 11 and 12. These problems appear to be more difficult than the previous ones. This may be due to there being many optimal solutions, i.e., considerable primal degeneracy which is detrimental to column generation. Indeed numbers of calls to the heuristics and of columns generated, for largest instances solved, are about 4 times larger for method 1 problems and 2 times for method 2 problems.

5.3 Approximation Problems

As mentioned in Section 3, lower-approximation problems are equivalent to feasibility problems. Then results of Table 5 can be interpreted in the former way too. Results for upper approximation problem (7) and lower-upper approximation problem (8) are presented in Tables 13 and 14. Finally results for the general additive constant problem (6) are given in Table 15.

Lower approximation problems appear to be the easiest; lower-upper approximation problems are slightly easier than upper approximation problems. This appears to be due to the less constrained solution space. Once more VNS performs well as the average number of calls to the exact algorithm is moderate. It appears also that the time of the exact algorithm augments much more rapidly than for other problems. Again it appears to be the limitation factor.

Finally, the general additive constant problem is more difficult than other approximation problems and of similar difficulty as the minimum ℓ_1-size optimization problem. From Table 15, it appears that:

(i) CPU time augments very rapidly with n (e.g. for n going from 15 to 30 it is multiplied roughly by 900);
(ii) VNS performs well as the number of calls to the exact algorithm for the subproblem is almost minimum;
(iii) For the largest instances solved CPLEX 7.0 takes over 96 % of the computing time.

Table 8. Results on feasibility problems without using column generation

Prob. size	CPU time	
n	μ	σ
10	0.010	0.005
15	2.359	0.609
20	270.284	81.689

Table 9. Results on optimization problems with $c_t = \min(|S^t|, |V_n \setminus S^t|)$ where d is generated by method 1

Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.033	0.016	10.2	0.015	4.3	0.000	1.0	0.012	5.3
15	0.220	0.058	89.5	0.059	6.8	0.000	1.0	0.148	7.8
20	1.522	0.525	271.6	0.138	7.2	0.002	1.0	1.364	8.2
25	9.352	2.322	575.5	0.259	8.6	0.003	1.0	9.055	9.6
30	38.393	10.292	819.0	0.360	10.2	0.010	1.0	37.955	11.2
35	122.610	23.674	1075.4	0.486	12.2	0.039	1.0	121.947	13.2
40	347.069	66.292	1413.5	0.664	15.2	0.212	1.0	345.968	16.2
45	829.228	111.177	1714.8	0.878	18.1	0.965	1.0	827.041	19.1
50	1970.159	288.347	2117.8	1.169	22.2	4.544	1.0	1963.917	23.2
55	4678.657	823.377	2710.0	1.506	28.1	15.283	1.0	4661.076	29.1

Table 10. Results on optimization problems with $c_t = \min(|S^t|, |V_n \setminus S^t|)$ where d is generated by method 2

Prob. size		Complete algorithm			Heuristic		Exact		CPLEX 7.0	
		CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	m	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	5	0.138	0.024	52.2	0.097	15.1	0.000	1.0	0.036	16.6
10	10	0.091	0.021	36.4	0.052	9.4	0.000	1.0	0.034	10.4
10	20	0.059	0.020	17.9	0.036	6.1	0.000	1.0	0.019	7.1
15	5	1.991	0.252	391.9	0.560	23.5	0.001	1.0	1.411	25.5
15	10	1.640	0.199	326.2	0.458	18.4	0.001	1.0	1.164	19.4
15	20	0.793	0.178	192.2	0.294	11.9	0.000	1.0	0.486	12.9
20	5	45.651	12.794	1456.8	1.816	30.1	0.007	1.0	43.785	32.1
20	10	25.710	3.713	1201.2	1.265	21.1	0.002	1.0	24.402	22.1
20	20	13.861	1.272	737.8	1.124	17.1	0.000	1.0	12.709	18.1
25	5	721.345	233.499	3322.3	3.177	43.4	0.092	1.1	717.888	45.2
25	10	290.113	36.203	3008.6	1.778	35.2	0.018	1.0	288.171	36.2
25	20	167.695	24.195	1882.9	1.750	23.9	0.010	1.0	165.835	24.9
30	10	1953.958	264.493	5879.4	2.996	62.5	0.199	1.1	1950.329	63.6
30	20	1300.140	161.446	3645.8	2.487	39.1	0.077	1.0	1297.311	40.1

Table 11. Results on optimization problems with $c_t = 1$ where d is generated by method 1

Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	1.544	0.228	89.4	1.480	13.7	0.002	1.0	0.057	14.7
15	6.709	0.827	421.8	5.135	14.1	0.013	1.0	1.549	15.1
20	38.124	11.757	1014.0	12.174	17.4	0.374	1.0	25.534	18.4
25	289.590	256.024	2168.4	39.084	34.1	16.371	1.2	234.011	35.1

Table 12. Results on optimization problems with $c_t = 1$ where d is generated by method 2

Prob. size		Complete algorithm			Heuristic		Exact		CPLEX 7.0	
		CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	m	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	5	0.163	0.029	71.5	0.096	16.4	0.000	1.0	0.061	18.0
10	10	0.153	0.026	73.8	0.089	14.4	0.000	1.0	0.058	15.4
10	20	0.153	0.020	82.2	0.094	14.4	0.001	1.0	0.051	15.4
15	5	2.347	0.366	424.9	0.470	20.6	0.002	1.0	1.854	22.6
15	10	2.238	0.469	397.4	0.393	16.9	0.002	1.0	1.827	17.9
15	20	2.183	0.508	412.4	0.364	16.2	0.007	1.0	1.793	17.2
20	5	53.515	18.185	1580.2	1.477	26.9	0.013	1.0	51.969	28.9
20	10	32.550	6.350	1186.8	0.998	20.2	0.038	1.0	31.468	21.2
20	20	42.213	11.400	1459.0	0.942	23.4	0.155	1.0	41.052	24.4
25	5	792.231	201.917	3730.6	3.109	46.0	0.179	1.0	788.726	47.7
25	10	409.865	122.989	3466.5	1.927	39.6	3.086	1.0	404.648	40.6
25	20	492.669	163.904	4018.7	2.476	46.5	3.712	1.0	486.269	47.5
30	10	4692.654	4936.89	7515.4	10.061	116.2	1324.048	11.7	3357.885	117.3

Table 13. Results on upper ℓ_1-norm approximation problems where d is generated by method 3

Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.113	0.018	39.3	0.089	14.9	0.000	1.0	0.022	16.9
15	0.652	0.139	218.8	0.386	16.9	0.006	1.0	0.250	18.9
20	4.552	0.303	762.4	0.994	17.4	0.059	1.0	3.471	19.4
25	28.174	3.682	1436.5	1.724	21.9	0.935	1.1	25.444	23.9
30	136.572	20.996	2282.7	3.562	32.1	16.520	1.4	116.341	34.1
35	788.164	192.934	3412.6	7.713	45.7	359.708	2.1	420.423	47.7

To conclude, ℓ_1-embedding of a real valued distance as well as related optimization problems can be performed for instances of small to moderate size (i.e., up to $n = 25$ to $n = 85$) using column generation. Our results show that our algorithm considers few columns explicitly compared to the total number

Table 14. Results on lower-upper ℓ_1-norm approximation problems where d is generated by method 3

- Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.087	0.023	23.1	0.065	11.4	0.001	1.0	0.017	12.4
15	0.417	0.067	113.3	0.274	11.9	0.004	1.0	0.135	12.9
20	1.830	0.247	333.1	0.687	10.8	0.051	1.0	1.077	11.8
25	8.423	1.336	635.8	1.170	11.8	0.777	1.0	6.436	12.8
30	36.812	5.761	949.6	2.186	14.8	10.145	1.0	24.408	15.8
35	236.762	44.145	1323.2	3.721	18.4	155.496	1.1	77.415	19.4
40	2751.747	984.009	1749.3	6.660	23.1	2550.537	1.2	194.331	24.1

Table 15. Results on general additive constant problems where d is generated by method 3

Prob. size	Complete algorithm			Heuristic		Exact		CPLEX 7.0	
	CPU time		Nb cols	CPU time	Calls	CPU time	Calls	CPU time	Calls
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.550	0.115	37.5	0.498	6.6	0.000	1.0	0.047	8.6
15	3.654	0.631	348.2	2.181	9.1	0.003	1.0	1.462	11.1
20	36.388	8.598	1133.2	4.713	15.1	0.081	1.0	31.549	17.1
25	372.783	109.836	3029.7	8.310	32.2	3.147	1.0	361.178	34.2
30	3301.986	865.874	8019.2	33.405	85.0	81.375	1.1	3186.662	87.0

of columns (which equals $2^{n-1} - 1$). So a difficulty pointed out by Fichet (1987; 1994) and Benayade and Fichet (1994), i.e., the high dimensionality of the linear programming models involved is, at least to some extent, overcome.

References

ALKHAMIS, T.M., HASAN, M., and AHMED, M.A. (1998), "Simulated Annealing for the Unconstrained Quadratic Pseudo-Boolean Function", *European Journal of Operational Research, 108*, 641–652.

ASSOUAD, P. (1979-1980), "Plongements isométriques dans ℓ_1: aspect analytique", in *Séminaire d'Initiation à l'analyse, 14*, ed. G. Choquet, Université de Paris VI.

AVIS, D. (1977), "Some Polyhedral Cones Related to Metric Spaces", Ph.D. thesis, Stanford University.

AVIS, D., and DEZA, M. (1991), "The Cut Cone, ℓ_1-embeddability, Complexity, and Multicommodity Flows", *Networks, 21*, 595–617.

BENAYADE, M., and FICHET, B. (1994), "Algorithms for a Geometrical P.C.A. with the ℓ_1-norm", in *New Approaches in Classification and Data Analysis*, ed. E. Diday, Berlin: Springer-Verlag, pp. 75–84.

BOOLE, G. (1854a), "An Investigation of the Laws of Thought, on which are Founded the Mathematical Theories of Logic and Probabilities", London: Walton and Maberley. (Reprint New York: Dover 1958).

BOOLE, G. (1854b), "On the Conditions by which Solutions of Questions in the Theory of Probabilities are Limited", *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science, 4*, 91–98.

BOOLE, G. (1854c), "On the General Method in the Theory of Probabilitites", *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science, 4*, 431–444.

BRUSCO, M.J. (2001), "A Simulated Annealing Heuristic for Unidimensional and Multidimensional (City-Block) Scaling of Symmetric Proximity Matrices", *Journal of Classification, 18*, 3–33.

CHVÁTAL, V. (1983), "Linear Programming", New York: Freeman.

CRITCHLEY, F. (1980), "Optimal Norm Characterizations of Multidimensional Scaling Methods and Some Related Data Analysis Problems", in *Data Analysis and Informatics*, ed. .E. Diday, Amsterdam: North Holland.

DEZA, M.M., and LAURENT, M. (1997), "Geometry of Cuts and Metrics", Berlin: Springer-Verlag.

FICHET, B. (1987), "The Role Played by ℓ_1 in Data Analysis", in *Statistical Data Analysis Based on the ℓ_1-norm and Related Methods (Neuchâtel, 1987)*, ed. Y. Dodge, Amsterdam: North-Holland, pp. 185–193.

FICHET, B. (1994), "Dimensionality Problems in ℓ_1-norm Representations", in *Classification and Dissimilarity Analysis*, ed. B. Van Custem, Lecture Notes in Statistics, New York: Springer, pp. 201–224.

FICHET, B., and LE CALVÉ, G. (1984), "Structure géométrique des principaux indices de dissimilarité sur signe de présence-absence", *Statistiques et Analyse de données, 9*, 11–44.

GAREY, M.R., and JOHNSON, D.S. (1979), "Computers and Intractability: A Guide to the Theory of NP-completeness", San Francisco: W. H. Freeman and Co.

GEORGAKOPOULOS, G., KAVVADIAS, D., and PAPADIMITRIOU, C.H. (1988), "Probabilistic Satisfiability", *Journal of Complexity, 4*, 1–11.

GILMORE, P.C., and GOMORY, R.E. (1961), "A Linear Programming Approach to the Cutting-Stock Problem", *Operations Research, 9*, 849–859.

GILMORE, P.C., and GOMORY, R.E. (1963), "A Linear Programming Approach to the Cutting Stock Problem: Part II", *Operations Research, 11*, 863–888.

GLOVER, F., KOCHENBERGER, G.A., and ALIDAEE, B. (1998), "Adaptive Memory Tabu Search for Binary Quadratic Programs", *Management Science, 44*, 336–345.

GLOVER, F., and LAGUNA, M. (1997), "Tabu Search", Dordrecht: Kluwer.

GRISHUKHIN, V.P. (1990), "All Facets of the Cut Cone $\mathbf{C}_n$ for $n = 7$ are Known", *European Journal of Combinatorics, 11*, 115–117.

GROENEN, P.J.F., HEISER, W.J., and MEULMAN, J.J. (1998), "City-block Scaling: Smoothing Strategies for Avoiding Local Minima", in *Classification, Data Analysis, and Data Highways*, eds. I. Balderjahn, P. Mathar, and M. Schader, Berlin: Springer, pp. 46–53.

HAMMER, P.L., HANSEN, P., and SIMEONE, B. (1984), "Roof Duality, Complementation and Persistency in Quadratic 0-1 Optimization", *Mathematical Programming, 28*, 121–155.

HAMMER, P.L., and HANSEN, P. (1981), "Logical Relations in Quadratic 0-1 Programming", *Revue Roumaine de Mathématique Pures et Appliquées, 26*, 421–429.

HAMMER, P.L., and RUBIN, A.A. (1970), "Some Remarks on Quadratic Programming with 0-1 Variables", *Revue française d'informatatique et de recherche opérationnelle, 4*, 67–79.

HANSEN, P., and JAUMARD, B. (2000), "Probabilistic Satisfiability", in *Algorithms for Uncertainty and Defeasible Reasoning*, *Handbook of Defeasible Reasoning and Uncertainty, Management Systems*, Vol. 5., Dordrecht: Kluwer Academic Publishers, pp. 321–367.

HANSEN, P., JAUMARD, B., and MEYER, C. (2000), "A Simple Enumerative Algorithm for Unconstrained 0–1 Quadratic Programming", *Les Cahiers du GERAD*, G–2000–59, HEC Montréal.

HANSEN, P., JAUMARD, B., and DA SILVA, E. (1991), "Average-linkage Divisive Hierarchical Clustering", *Les Cahiers du GERAD*, G–91–55, HEC Montréal.

HANSEN, P., and MLADENOVIĆ, N. (2003), "Variable Neighborhood Search", in *Handbook of Metaheuristics*, Vol. 57 of *International Series in Operations Research and Management Science*, eds. F. Glover and G. Kochenberger, Boston MA: Kluwer Academic Publishers, pp. 145–184.

HASAN, M., ALKHAMIS, T., and ALI, J. (2000), "A Comparison Between Simulated Annealing, Genetic Algorithm and Tabu Search Methods for the Unconstrained Quadratic Pseudo-Boolean Function", *Computers and Industrial Engineering, 38*, 323–340.

HUBERT, L., ARABIE, P., and HESSON-MCINNIS, M. (1992), "Multidimensional Scaling in the City-Block Metric: A Combinatorial Approach", *Journal of Classification, 9*, 211–236.

JOLY, S., and LE CALVÉ, G. (1994), "Similarity Functions", in *Classification and Dissimilarity Analysis*, ed. B. Van Custem, Lecture Notes in Statistics, New York: Springer, pp. 67–86.

KARP, R.M., and PAPADIMITRIOU, C.H. (1982), "On Linear Characterizations of Combinatorial Optimization Problems", *SIAM Journal on Computing, 11*, 620–632.

KAVVADIAS, D., and PAPADIMITRIOU, C.H. (1990), "A Linear Programming Approach to Reasoning about Probabilities", *Annals of Mathematics and Artificial Intelligence, 1*, 189–205.

LODI, A., ALLEMAND, K., and LIEBLING, T.M. (1999), "An Evolutionary Heuristic for Quadratic 0-1 Programming", *European Journal of Operational Research, 119*, 662–670.

MLADENOVIĆ, N., and HANSEN, P. (1997), "Variable Neighbourhood Search", *Computers and Operations Research, 24*, 1097–1100.

PALUBECKIS, G. (1995), "A Heuristic-Based Branch and Bound Algorithm for Unconstrained Quadratic Zero-One Programming", *Computing, 54*, 283–301.

PARDALOS, P.M., and RODGERS, G.P. (1990), "Computational Aspects of a Branch and Bound Algorithm for Quadratic Zero-One Programming", *Computing, 45*, 131–144.

ZHOU, X., WANG, Y., TIAN, X., and GUO, R. (1997), "A Tabu Search Algorithm for Quadratic 0-1 Programming Problem", *Chinese Quarterly Journal of Mathematics, 12*, 98–102.