

Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm*

J.-A. Ferrez, K. Fukuda,[†] Th.M. Liebling

Institute of Mathematics

Swiss Federal Institute of Technology, Lausanne, Switzerland

October 25, 2002; Revised April 29, 2004

Abstract

We address the weighted max-cut problem, or equivalently the problem of maximizing a quadratic form in n binary variables. If the underlying (symmetric) matrix is positive semi-definite of fixed rank d , then the problem can be reduced to searching the extreme points of a zonotope, thus becoming of polynomial complexity in $O(n^{d-1})$. Reverse search is an efficient and practical means for enumerating the cells of a regular hyperplane arrangement, or equivalently, the extreme points of a zonotope. We present an enhanced version of reverse search of significantly reduced computational complexity that uses ray shooting and is well suited for parallel computation. Furthermore, a neighborhood zonotope edge following descent heuristic can be devised. We report preliminary computational experiments of a parallel implementation of our algorithms.

1 Introduction

The unconstrained quadratic maximization problem in zero-one variables (abbreviated by 01QP)

$$\begin{aligned} \max \quad & f(x) = x^T Q x \\ \text{subject to} \quad & x \in \{0, 1\}^n, \end{aligned}$$

where Q is an $n \times n$ rational symmetric matrix, is a classical NP-hard combinatorial optimization problem. It is well known that the weighted max-cut problem can be considered as a special case. In fact, there are simple polynomial reductions between the weighted max-cut problem and the 01QP. This interesting result, due to [16], will be reviewed in Section 2.

The 01QP problem remains NP-hard even when Q is positive definite or when it is indefinite of rank two [15]. If a linear term is added to the objective function, the problem remains NP-hard even when Q is negative definite. Well known polynomial cases are, for instance, when (a) the matrix Q is of rank one, in which case the solution can be found by inspection, (b) Q has nonnegative off-diagonal elements [19], and (c) the graph underlying the associated max-cut problem is series parallel [3].

Recently, a new polynomially solvable case [1] has been found. This case, that we call the *fixed-rank convex (FRC)* case, is when Q is positive semidefinite and of fixed rank d . This result, which will be reviewed in Section 3, reduces the search space of 2^n 0/1 feasible solutions to that of $O(n^{d-1})$ restricted 0/1 solutions by a geometric transformation. More precisely these restricted solutions are

*An earlier version of this paper was titled "Cuts, Zonotopes and Arrangements."

[†]Also affiliated with Institute for Operations Research and Institute of Theoretical Computer Science, Swiss Federal Institute of Technology, Zurich, Switzerland.

the extreme points of a d -dimensional affine image of the unit n -dimensional hypercube. This object is known as a *zonotope*.

One main purpose of the present article is to investigate the practical impact of this reduction and applicable algorithms. In particular, we propose a more efficient modification of the reverse search algorithm [2] for generating all extreme points of a given zonotope in \mathbb{R}^d , and we study the performance of a straightforward implementation using the C language. Some of the largest instances we could solve on standard Unix workstations are randomly generated rank 3 cases with $n = 250$ and rank 4 cases with $n = 70$. It should be observed that randomly generated cases are in some sense hardest for the enumeration of extreme points, since they are known to attain (with probability one) the maximum size of output.

While we are not aware of any specific applications of low rank FRC 01QP problems, we strongly hope that the exact solutions our algorithm can compute will be useful. As a simple application of the present work, we propose to test heuristic or approximation algorithms with these instances and evaluate their actual performances. In fact, we have tested a new heuristic algorithm and obtained some interesting results, see Section 5. A similar study can be conducted in the future to compare the guaranteed performance and the actual performance of approximation algorithms with a performance guarantee. The randomized algorithm of Goemans and Williamson [14] for the maximum cut problem is an obvious candidate for such a study.

As we will see in the following sections, any instance of 01QP can be interpreted as an instance of the weighted max-cut problem, and also as an instance of the maximum weighted induced subgraph problem.

2 01QP and MaxCut

There are various ways to relate 01QP to NP -hard combinatorial optimization problems over graphs. The most natural way is by rewriting the objective function $f(x) = x^T Q x$ over $x \in \{0, 1\}^n$ as

$$x^T Q x = \sum_{i,j} q_{ij} x_i x_j = \sum_{i \neq j: x_i = x_j = 1} 2q_{ij} + \sum_{i: x_i = 1} q_{ii}. \quad (2.1)$$

Let G be the simple complete graph of order n with node weights q_{ii} ($i \in N = \{1, \dots, n\}$) and edge weights $2q_{ij}$ ($i \neq j$). Then 01QP can be simply interpreted as the problem of finding a maximum weight node-induced subgraph $G(S)$ of G ($S \subseteq N$) where the weight of a subgraph is defined as the sum of the weights of edges and the weights of nodes in the subgraph. This problem, known as the *maximum weight induced subgraph problem with both node and edge weights*, has interesting applications, e.g. see [4, 7].

In this section, we present how 01QP can be related to other (more) well known problems in graphs, namely, the maximum-weight cut problem and the maximum weight induced subgraph problem (without node weights). The transformations are less evident.

The weighted maxcut problem

Finding a cut of maximum weight in a graph with weighted edges is a well known NP -hard combinatorial optimization problem [13]. This problem, also known as WEIGHTED MAXCUT can be stated as follows. An *instance* of the problem is a graph $G = (N, E)$ along with an edge weight function $c \in \mathbb{R}^E$. The *task* is to find a partition $\{N_1, N_2\}$ of node set N that maximizes the sum of the weights of the edges having exactly one end node in N_1 . The maximum cardinality cut (or simply MAXCUT) problem is the special case where all edge weights are equal to one. Note that the graphs considered here are *simple*, i.e. loopless and without multiple edges sharing the same end nodes. In such cases a

problem instance can be described by an $n \times n$ symmetric matrix C , where element c_{ij} denotes one half of the weight of edge $\{i, j\}$, $c_{ij} = 0$ if $\{i, j\}$ is a non edge or if $i = j$. Note also that we consider the trivial partition $\{N, \emptyset\}$ as a cut, since it simplifies our discussion below. However this is not an essential condition for the validity of the key theorem to be given here.

The following result, first presented by Hammer in 1965 [16], has been rediscovered and applied by many authors in the sequel, see e.g. [3], and it establishes the close relationship between WEIGHTED MAXCUT and 01QP. For completeness, the proof is given here.

Theorem 2.1 *The unconstrained binary quadratic optimization problem 01QP and the WEIGHTED MAXCUT problem are equivalent in that:*

1. *For every instance of 01QP with n binary variables one can construct a weighted graph on $n+1$ nodes $N = \{0, 1, \dots, n\}$, such that there is an order preserving bijection between the (weighted) cuts on that graph and the solutions of 01QP.*
2. *For every instance of WEIGHTED MAXCUT on a graph G with n nodes it is possible to construct an instance of 01QP with n binary variables, such that that there is an order preserving bijection between the weighted cuts and the solutions of 01QP.*

Proof.

1. Without loss of generality, let the objective function of 01QP be given as $f(x) = x^T Q x + b^T x$, where Q is an $n \times n$ symmetric matrix with zero diagonals and $b \in \mathbb{R}^n$. With the change of variables $x = (y + e)/2 : y \in \{-1, 1\}^n$, where $e = (1, \dots, 1)^T$, the objective function can be rewritten as

$$\begin{aligned} f(x) \equiv \phi(y) &= \frac{1}{4}(y + e)^T Q (y + e) + \frac{1}{2}b^T (y + e) \\ &= \frac{1}{4}y^T Q y + \frac{1}{2}(Qe + b)^T y + \frac{1}{4}e^T Q e + \frac{1}{2}b^T e \\ &= y^T \tilde{Q} y + \tilde{b}^T y + \tilde{c}. \end{aligned} \tag{2.2}$$

Herein $\tilde{Q} = \frac{1}{4}Q$, $\tilde{b} = \frac{1}{2}(Qe + b)$ and $\tilde{c} = \frac{1}{4}e^T Q e + \frac{1}{2}b^T e$.

Now define vector $s^T = (s_0, \dots, s_n) = (1, y^T)$ and the $(n + 1) \times (n + 1)$ symmetric matrix W with zero diagonals as follows

$$W = \begin{bmatrix} 0 & \frac{1}{2}\tilde{b}^T \\ \frac{1}{2}\tilde{b} & \tilde{Q} \end{bmatrix}.$$

Since $s_i \in \{-1, 1\}$, we can write

$$\begin{aligned} \phi(y) - c &\equiv s^T W s = \sum_{i,j:s_i=s_j} w_{ij} - \sum_{i,j:s_i=-s_j} w_{ij} \\ &= -2 \sum_{i,j:s_i=-s_j} w_{ij} + \sum_{i,j} w_{ij} \\ &= -2 \sum_{s_i=-s_j} w_{ij} + \text{const.} \end{aligned} \tag{2.3}$$

We see that $\phi(y)$ is equal –up to a constant– to the weight of a cut in a graph on $n + 1$ nodes with edge weights $-w_{ij}$, which concludes the proof of claim 1.

2. Now, consider a graph $G = (N, E)$ on n nodes $N = \{1, \dots, n\}$ and let edge $\{i, j\}$ have weight $2q_{ij}$ as given by the symmetric matrix $Q = (q_{ij})$ with zero diagonals. Any cut can be described by a vector $y \in \{1, -1\}^n$ and its weight is given by $\kappa(y) = \frac{1}{2}(e^T Q e - y^T Q y)$. Using the bijection $y = 2x - e$ with $x \in \{0, 1\}^n$ we get

$$\begin{aligned}
 \kappa(y) &\equiv f(x) = -\frac{1}{2}(2x - e)^T Q (2x - e) + \frac{1}{2}e^T Q e \\
 &= -2x^T Q x + 2e^T Q x - \frac{1}{2}e^T Q e + \frac{1}{2}e^T Q e \\
 &= -2x^T Q x + 2e^T Q x,
 \end{aligned} \tag{2.4}$$

which shows the claim. ■

The Maximum Weight Induced Subgraph Problem

Equations (2.4) have a nice graphical interpretation. Note first that each instance of 01QP can be seen as finding a maximum weight induced subgraph of a given graph. Transforming a WEIGHTED MAXCUT instance on graph $G = (N, E)$ with node set $N = \{1, \dots, n\}$ and weight matrix C to one of 01QP, can be thought of constructing a graph $\tilde{G} = (N \cup \{0\}, E \cup E_0)$ as follows. Make a copy of G and give its edges weights corresponding to $-2C$. Add an extra node $\{0\}$ and draw an edge joining it to each node of G , give these edges the weights to be found in the corresponding components of Ce . There is an order preserving bijection between the induced subgraphs in \tilde{G} containing node $\{0\}$ and the cuts in G . See Figure 2.1.

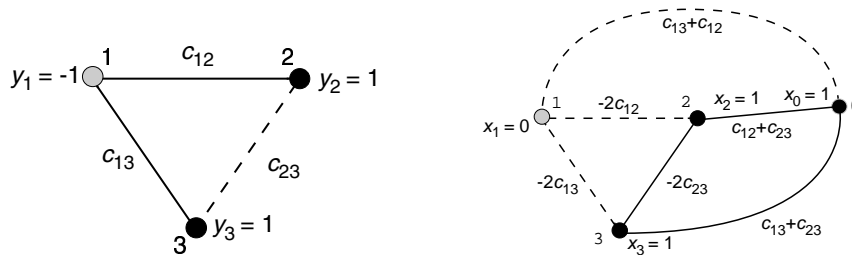


Figure 2.1: The transformation from MAXCUT to INDUCEDSUBGRAPH.

3 Fixed rank convex case, zonotopes and duality

Fixed rank convex 01QP

The recent paper [1] shows that when the matrix Q is positive semidefinite and of fixed rank, the 01QP problem can be solved in polynomial time, assuming that the eigenvectors of Q are also known. More precisely, we assume that Q is given as the product $Q = V^T V$, where V is a $d \times n$ matrix. We consider d is fixed. We shall call this problem a *fixed-rank convex* 01QP problem or simply *FRC* 01QP problem.

As discussed in the previous section, 01QP has a direct interpretation as the maximum weight induced subgraph problem with both node and edge weights. In the special case of FRC 01QP, one

can easily see how edge weights and node weights are constructed. Namely, the edge weights are $2(v^i, v^j)$ for all $i \neq j$ and the node weights are (v^i, v^i) for all i . Here v^j ($j = 1 \dots n$) denotes the j th *column vector* of V and (\cdot, \cdot) denotes the *inner product* to two vectors.

We shall review the basic techniques used in [1] to prove the polynomiality. First we rewrite the 01QP problem.

$$\begin{aligned} \max f(x) &= x^T Q x = x^T V^T V x = \sum_{i=1}^d (v_i, x)^2 = \|Vx\|^2 \\ \text{subject to } x &\in \{0, 1\}^n \end{aligned} \tag{3.5}$$

where v_i is the transpose of the i th row of V for $i = 1, \dots, d$ and $\|\cdot\|$ the Euclidean norm. Consider the linear map $\mathbb{R}^n \rightarrow \mathbb{R}^d : z = Vx$. The image of the hypercube $[0, 1]^n$ under this map is a convex polytope Z , known as a *zonotope*. Notice that for every extreme point \tilde{z} of Z there exists an extreme point \tilde{x} of $[0, 1]^n$, i.e. a point $\tilde{x} \in \{0, 1\}^n$, such that $\tilde{z} = V\tilde{x}$. For the optimal value f^* of (3.5) one has

$$\begin{aligned} f^* &= \max \{ \|Vx\|^2 : x \in \{0, 1\}^n \} \\ &= \max \{ \|Vx\|^2 : x \in [0, 1]^n \} \\ &= \max \{ \|z\|^2 : z \in Z \}, \end{aligned}$$

where the equality in the second line is a direct consequence of the convexity of the Euclidean norm $\|\cdot\|$. The last expression is a maximization of a convex function of z over the convex set Z and follows from the definition. Therefore the maximum is attained at some extreme point \tilde{z} of Z . Let us denote by $ext(Z)$ the set of extreme points of the zonotope Z .

$$f^* = \max \{ \|z\|^2 : z \in Z \} = \max_{z \in ext(Z)} \|z\|^2. \tag{3.6}$$

The FRC 01QP problem (3.5) is thus reduced to the enumeration of extreme points of the zonotope Z in \mathbb{R}^d . Z is the Minkowski sum of the n closed line segments $[0, v^j]$ for $j = 1, \dots, n$. The vectors v^j are called the *generators* of the zonotope. Figure 3.2 shows small examples in rank $d = 3$ and the number of generators $n = 5$ and 10. In general, each edge of a zonotope is parallel to some generator segment $[0, v^j]$, and in particular when the generators are in general position (like in our examples), each edge is a translation of some segment $[0, v^j]$. Note that the associated FRC 01QP problems are equivalent to the Euclidean norm maximization problems over these zonotopes.

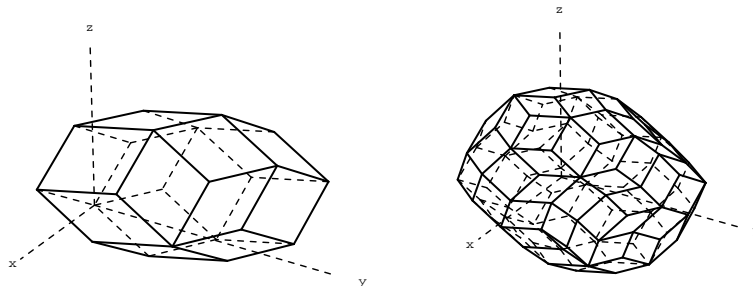


Figure 3.2: 3 dimensional zonotopes with 5 and 10 generators.

The exact upper bound of the number of extreme points of a zonotope is a classical result in discrete geometry. It is important to remark that many of the combinatorial results on zonotopes

were formulated and proved in the dual setting of hyperplane arrangements. We shall discuss this duality later in this section which enables us to “see” zonotopes better in some sense. Let us denote by $f_0(P)$ the number of extreme points of a convex polytope P .

Theorem 3.1 ([6, 20]) *Let Z be a d -dimensional zonotope with n generators ($n \geq d$). Then we have $f_0(Z) \leq 2 \sum_{i=0}^{d-1} \binom{n-1}{i}$. Furthermore, the equality is satisfied when the generators are in general position.*

The theorem immediately implies that $f_0(Z) = O(n^{d-1})$ and thus, the number of extreme points is polynomially bounded as d is fixed. The polynomial solvability of the FRC 01QP problem follows directly from the existence of an efficient algorithm (see [9, 8]) to generate all extreme points of Z . Again, the original algorithm [9] is designed for arrangements and needs to be dualized for our purpose.

Theorem 3.2 *For $d \geq 3$, there is an $O(n^{d-1})$ time algorithm to generate all extreme points of a d -dimensional zonotope given by n rational generators.*

Corollary 3.3 *The FRC 01QP problem is polynomially solvable.*

Unfortunately, the algorithm that attains the optimal time complexity may not be very practical. It relies on a so-called “incremental” strategy, that is, to solve the problem inductively: at k th stage ($k \leq n$), it maintains the list of extreme points of a subproblem with $(k - 1)$ generators and updates the list with one additional generator. The critical disadvantage of this method is its memory requirement. It has to store all the extreme points in memory. This means we use the storage of size $O(n^{d-1})$. Furthermore, the algorithm might be too complicated to implement: it stores not only the extreme points but also all faces and their incidences. For the details, see [8].

There is an alternate algorithm to solve the enumeration problem which is time and space efficient, highly parallelizable, and very easy to implement. We shall describe this method, based on the reverse search scheme by Avis and Fukuda [2], in the next section. Our preliminary computational experiments of this algorithm will be reported in Section 5. Since this algorithm is better understood in the dual setting, we first review the duality.

Duality

For the zonotope $Z = Z(V)$ generated by the columns of V , we define the associated central arrangement $\mathcal{A} = \mathcal{A}(V)$ of n hyperplanes in \mathbb{R}^d , each having a v^j as its normal vector:

$$\mathcal{A}(V) = \{h_j^0 : j = 1, 2, \dots, n\} \quad (3.7)$$

where $h_j^0 = \{y \in \mathbb{R}^d : (v^j, y) = 0\}$ for $j = 1, 2, \dots, n$. It is useful to consider the arrangement as oriented, that is, we have the positive and the negative sides of each hyperplane: $h_j^+ = \{y \in \mathbb{R}^d : (v^j, y) > 0\}$ and $h_j^- = \{y \in \mathbb{R}^d : (v^j, y) < 0\}$.

The extreme points of Z correspond one-to-one to the regions (d -dimensional faces) of the arrangement, and the facets of Z to the lines (1-dimensional faces) of the arrangement, see Figure 3.3. More generally, there is an order reversing bijection between the face posets of the two objects. To see this, first we define the location vector $\sigma(c) \in \{+, 0, -\}^n$ of $c \in \mathbb{R}^d$

$$\sigma(c)_j = \begin{cases} + & \text{if } c \in h_j^+ \\ 0 & \text{if } c \in h_j^0 \\ - & \text{if } c \in h_j^- \end{cases} \quad (3.8)$$

See Figure 3.3, where the arrangement is shown as its intersection with a sphere. A *face* of \mathcal{A} is the set $F_c = \{y \in \mathbb{R}^d : \sigma(y) = \sigma(c)\}$ for some $c \in \mathbb{R}^d$. By this definition each face F of the arrangement \mathcal{A} is a relatively open set and uniquely represented by its sign vector, namely, the location vector $\sigma^F = \sigma(c)$ of any point c in F .

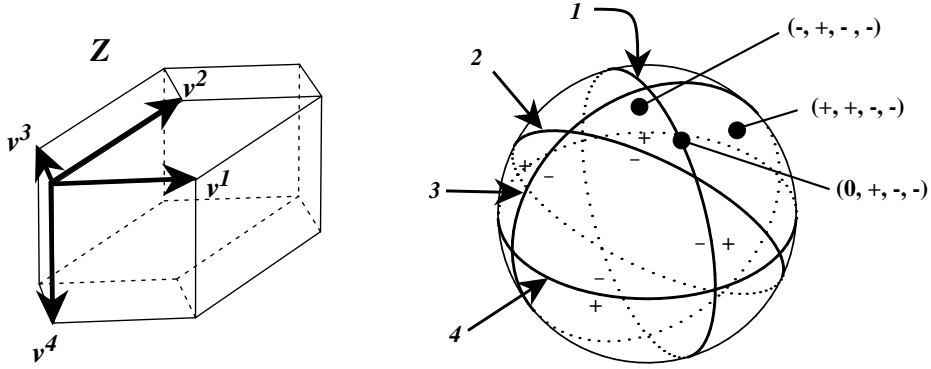


Figure 3.3: A zonotope and the associated 3-dimensional arrangement.

Now for each $c \in \mathbb{R}^d$, define a face K_c of the cube $K = [0, 1]^n$ by

$$K_c = \{x \in [0, 1]^n : \begin{array}{ll} x_j = 1 & \text{for all } j \text{ such that } (c, v^j) > 0 \text{ and} \\ x_j = 0 & \text{for all } j \text{ such that } (c, v^j) < 0 \end{array}\}.$$

Finally, we denote by Z_c the image of K_c under the linear map given by the matrix V .

$$Z_c = V \cdot K_c = \{y \in \mathbb{R}^d : y = Vx, x \in K_c\}.$$

It is easy to see that Z_c is a face of the zonotope Z and every face of Z is of this form. Also, the extreme points of Z_c are all possible sums of v^j 's such that the summands include all v^i with $(c, v^i) > 0$ and some v^k 's with $(c, v^k) = 0$. In particular, Z_c consists of a single point (i.e. Z_c is 0-dimensional) if there is no index k such that $(c, v^k) = 0$.

Now there is a close relation between F_c and Z_c . First of all both sets are invariant over all c with the same sign $\sigma(c)$. Thus, by abuse of notation, we might write $F_{\sigma(c)}$ and $Z_{\sigma(c)}$. Now let σ be any realizable sign, i.e. of form $\sigma(c)$ for some $c \in \mathbb{R}^d$. Then, one can verify that F_{σ} is the set of vectors c for which Z_{σ} is the set of maximizers of the linear function (c, y) over Z . The closure of F_c is sometimes called the *normal cone* of Z at face Z_c , and the collection of them the *normal fan* of Z .

The discussion above leads to the duality of zonotopes and arrangements, see Figure 3.4. (For a formal proof, see e.g. [21, Section 7.3].)

Theorem 3.4 (Duality) *There is an order reversing bijection between the poset of nonempty faces of $Z(V)$ and the face poset of the arrangement $\mathcal{A}(V)$. Namely, the correspondence $Z_{\sigma} \leftrightarrow F_{\sigma}$ for all realizable signs σ induces such a map.*

One important consequence of this theorem is the one-to-one correspondence between the extreme points of Z and the cells (d dimensional faces) of \mathcal{A} . Each cell F corresponds to an extreme point $z = \sum\{v^j : \sigma_j^F = +, j = 1, \dots, n\}$ of the zonotope Z . For example, in Figure 3.3, the cell

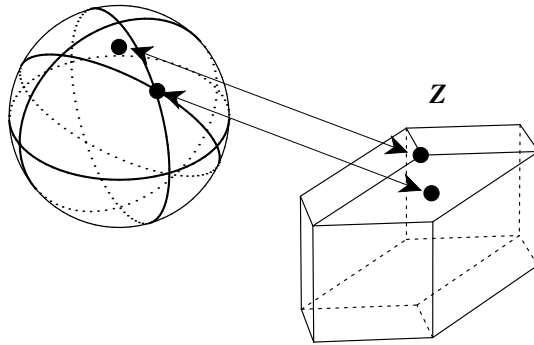


Figure 3.4: Duality of a zonotope and the associated arrangement (shown as a cut section with the unit sphere).

of sign $(+, +, -, -)$ (respectively $(-, +, -, -)$) is associated with the extreme point $z = v^1 + v^2$ (respectively v^2). Furthermore, two extreme points in Z are adjacent in Z if and only if the associated cells are adjacent (i.e. sharing a $(d - 1)$ face). Under certain regularity assumptions to be discussed in the next section, two cells are adjacent if and only if their sign vectors are different in exactly one component.

4 Cell enumeration of an arrangement

Consider the central arrangement $\mathcal{A}(V)$ of n hyperplanes in \mathbb{R}^d given by a $d \times n$ matrix V . We denote by $C = C(V)$ the set of sign vectors of cells of the arrangement. Whenever there is no ambiguity, we identify the cells and their sign vectors. The purpose of this section is to present a new practical algorithm for computing C . Because the arrangement $\mathcal{A}(V)$ is central, we only need to generate one half of the sign vectors. A natural way to ignore a half is to look at a cut section of this arrangement with a fixed hyperplane h not containing the origin. By selecting a parallel translation of the last hyperplane as h , we obtain a cut section that is a general (Euclidean) arrangement of $n - 1$ hyperplanes in \mathbb{R}^{d-1} . For example, Figure 4.5 illustrates a cut section with the 5th hyperplane of a central arrangement of 5 hyperplanes in \mathbb{R}^3 . Each cell is represented by a sign vector of length 4 rather than 5 because the 5th component is the same for all, either $+$ or $-$.

There are several different algorithms for the cell enumeration for a general arrangement. As remarked in the previous section, the incremental algorithm [9] is theoretically optimal for fixed d . However, it requires space as large as the output size and is quite hard to implement as well. There is a reverse search algorithm [2] which is memory efficient (i.e. its space complexity is polynomially bounded by the input size). Its time complexity is not optimal for the worst case output but it runs in time polynomial in both input size and output size which might be considered reasonable. More precisely, the time complexity is $O(n d LP(n, d) |C|)$, where $LP(n, d)$ is the time to solve an LP with n inequalities in d variables. The LP is polynomially solvable and one can replace $LP(n, d)$ with any realizable polynomial complexity if one wishes.

We shall present here a modification of the reverse search algorithm with an improved complexity.

Theorem 4.1 *There is a reverse search algorithm of time complexity $O(n LP(n, d) |C|)$ and space complexity $O(n d)$ that computes $C = C(V)$ for any given rational $d \times n$ matrix V .*

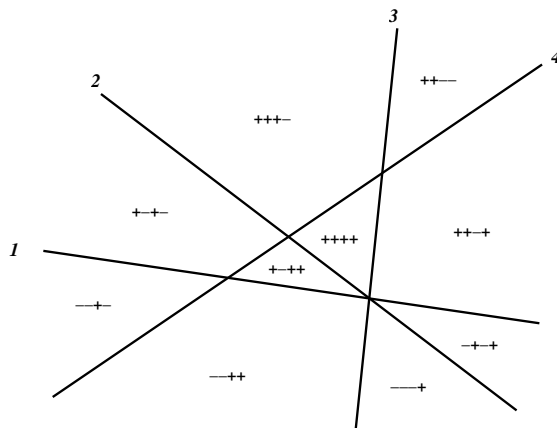


Figure 4.5: A cut section of a 3-dimensional central arrangement.

Because the notion of reverse search is well-known, it is sufficient to define two key functions that determine a reverse search algorithm. These are a *finite local search function* f and an *adjacency oracle function* Adj . An adjacency oracle defines implicitly a graph G on the set C as its vertex set by returning the set of neighbor cells of any given cell. A local search function maps any cell in C to an adjacent cell. A local search is called *finite* if there exists a special cell $c^* \in C$ such that for any $c \neq c^*$, there is a positive integer k such that $f^k(c) = c^*$. Once a finite local search f is fixed, we have a uniquely defined directed tree T_f rooted at c^* with edge set $\{(c, f(c)) | c \in C \setminus \{c^*\}\}$ spanning all elements in C . The reverse search algorithm is a procedure to visit all members of C by tracing the spanning tree T_f from the “optimal” element c^* , relying only on the two functions f and Adj .

There is a natural graph structure underlying the arrangement. This is exactly what we use for our adjacency oracle. For the local search, there are multiple choices. Before presenting these functions formally, we assume the following regularity condition for V which can be readily satisfied by a simple transformation.

Assumption (Regularity) The matrix V has no zero columns and has no two columns that are multiples of each other.

Note that if two columns are nonzero multiples of each other, they determine the same hyperplane in the arrangement, and one can be removed without changing the combinatorial structure of the arrangement. The original sign vectors can be easily obtained from the sign vectors of the simplified arrangement.

To define an adjacency oracle, let c be any cell in C . We call an index $j \in \{1, \dots, n\}$ *flippable* in c if $c_j \neq 0$ and the vector obtained from c by reversing the j th sign is again a cell. Our *adjacency oracle* $Adj(c, j)$ is defined to return this new cell if j is flippable, and NULL otherwise, for $j = 1, 2, \dots, n$. The obvious upper bound of the maximum degree of a vertex is n . Clearly one LP of size $(n - 1) \times (d - 1)$ is sufficient to decide whether j is flippable at any given cell c .

To define a local search function, let c^* be any cell in C that is known at the beginning of computation. To find one cell is easy: for example by selecting an arbitrary point in \mathbb{R}^d and checking its signature. The probability that an arbitrary point lies on any hyperplane is zero. Without loss of generality, we may assume c^* is the vector of all $+$'s. For this, we might have to replace some columns by their negatives. Such operations preserve the arrangement.

Now, for any $c \in C \setminus \{c^*\}$, we must define an easily computable next cell c' which is adjacent to c and is “closer” to the goal c^* . Any such systematic rule will define f (i.e. $f(c) = c'$). It is easy to

show under the Regularity Assumption that any cell c different from c^* has a flippable index j such that $c_j = -$. Thus, flipping such an index leads us closer to c^* .

The paper [2] uses a minimum index rule to determine c' , that is, $c' = f(c)$ is the cell obtained from c by the smallest index flip. The smallest index flip can be computed by solving at most n linear programs (LPs) of order $n \times d$.

Our modification will reduce this complexity to that of solving **only one** LP of the same size. The key idea is to use ray shooting. For this, we need two points, one interior point p^* of the goal cell c^* and one interior point p of a cell c , see Figure 4.6. Now, shoot a ray from p to p^* . It will hit all hyperplanes separating c and c^* . We select the first hyperplane hit by the ray. In case of a tie, we employ the standard symbolic perturbation to resolve it. In Figure 4.6, the hyperplane number 2 is the one we select. This gives us a way to move to a neighbor of c . It is important to note that the interior point we select for each cell must be uniquely defined. One way to satisfy this is to use the following LP to find an interior of a cell. We assume that the cell is represented by a linear inequality system: $Ay \leq b$ where A is a $(m-1) \times (d-1)$ matrix.

$$\begin{aligned} \max \quad & y_0 \\ \text{subject to} \quad & Ay + e \begin{matrix} y_0 \leq b \\ y_0 \leq K \end{matrix} \end{aligned}$$

where e is the vector of all 1's and K is any positive number to make the LP bounded. By using any deterministic algorithm, we will find a unique solution to this LP that is an interior point of the cell.

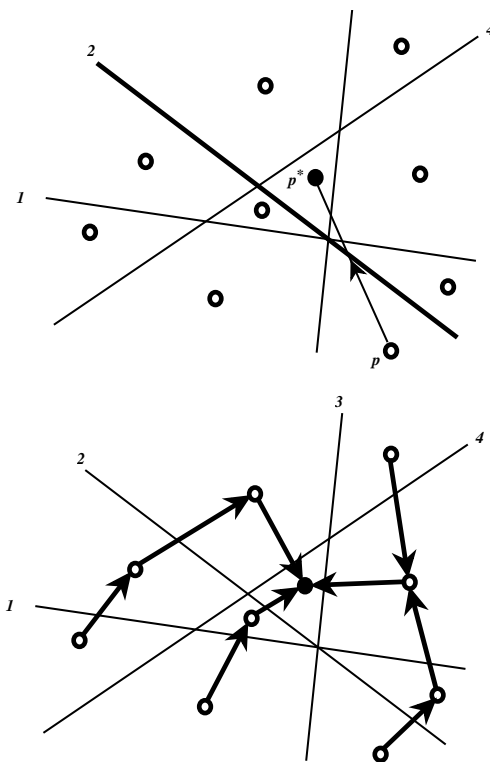


Figure 4.6: Ray shooting and the associated directed tree.

With the ray shooting local search f , we only need to solve one LP to evaluate it, since p^* can be computed at the beginning and stored through the course of the algorithm. By applying Theorem 2.4 in [2], we immediately obtain Theorem 4.1.

Finally, the resulting reverse search algorithm can be written as follows. Here, we must set $\delta = n - 1$. The algorithm generates all cells with positive last component.

```

procedure CellEnumeration( $Adj, \delta, c^*, f$ );
   $c := c^*$ ;  $j := 0$ ; (*  $j$ : neighbor counter *)
  repeat
    while  $j < \delta$  do
       $j := j + 1$ ;
       $next := Adj(c, j)$ ;
      if  $next \neq \text{NULL}$  then
        if  $f(next) = c$  then (* reverse traverse *)
           $c := next$ ;  $j := 0$ 
        endif
      endif
    endwhile;
    if  $c \neq c^*$  then (* forward traverse *)
       $c' := c$ ;  $c := f(c)$ ;
       $j := 0$ ; repeat  $j := j + 1$  until  $Adj(c, j) = c'$  (* restore  $j$  *)
    endif
  until  $c = c^*$  and  $j = \delta$ .

```

The while loop in the pseudocode above is to traverse the directed tree T_f against its orientation as deep as possible (i.e. depth-first manner). The subsequent if-block is executed whenever no deeper reverse traverse is possible, and it simply backtracks (i.e. traverses the tree in the forward direction) by applying the local search function f once. In order to avoid visiting the same cell twice, the last line of the if-block recovers the adjacency index j that was set at the last loop of the while loop executed for the current cell c .

5 Computational experiments

Early computational experiments for the FRC 01QP problem were performed using the ZRAM library [18] for the reverse search framework. The implementation effort reduces to providing the *adjacency oracle* and the *local search* function given in the previous section. As discussed, these components are built around basic computational geometry and linear optimization elements. More specifically, two basic functions are needed, the linear programming solving and the ray shooting operation. Our implementation uses these two functions implemented in Fukuda's cddlib library [11]. ZRAM then takes care of the reverse search mechanism, and provides efficient parallelization at no additional cost. Since all cddlib functions can be compiled with both floating-point and GMP rational (exact) arithmetics, our code called *rs_tope* can also run in these arithmetics.

We generated a range of problems in dimensions 3, 4, 5 and 6 with 10 to 250 constraints using the following approach. Given parameters s_0 and $r > 0$ (in addition to n and d), our algorithm returns the columns $v^j \in \mathbb{R}^d$ (for $j = 1, \dots, n$) of the $d \times n$ matrix V . The s_0 parameter represents a seed for the Bratley-Fox random generator [5]. Vectors v^j are first randomly generated on the d -dimensional unit sphere, and then scaled such that their norm is r . To be able to perform rational arithmetics, each component of v_i is rounded to a neighboring integer value. The last step of the algorithm is to check the non-collinearity for each pair of vectors. If that test fails, the instance is rejected and the algorithm is applied with a new s_0 seed value. The solutions to the random instances used in our experiments as well as the C source code *rs_tope.c* are available publicly in [10, 12].

Number of cells and computation time

Table 1 and figure 5.7 show the number of cells, total computation time and computation time per cell for various small and medium problems. The values confirm the theoretical result for the number of cells (see theorem 3.1 above). The computation times were obtained using both processors of a 400 MHz dual-Pentium II PC running Linux. They should be used to compare the effort required to solve different problems, but their absolute value should not be given too much importance as various elements (code optimization, processor type and frequency, parallel computing) have great influence thereon.

n	d = 3			d = 4		
	#cell	total [s]	per cell [ms]	#cell	total [s]	per cell [ms]
10	92	0.151	1.644	260	0.772	2.969
15	212	0.600	2.832	940	4.748	5.051
20	382	1.338	3.504	2320	14.640	6.310
25	602	2.861	4.754	4650	37.705	8.108
30	872	5.880	6.743	8180	84.987	10.389
35	1192	8.911	7.475	13160	170.672	12.969
40	1562	15.069	9.647	19840	310.114	15.630
45	1982	22.575	11.390	28470	519.803	18.257
50	2452	34.608	14.114	39300	861.008	21.908
55	2970	51.347	17.288	52580	1344.746	25.575
60	3540	68.050	19.223	68560	2044.175	29.815
65	4160	92.898	22.331	87490	2932.053	33.513
70	4830	130.317	26.980	109620	4163.190	37.978
80	6320	209.855	33.204			
90	8010	329.875	41.182			
100	9900	494.050	49.904			
125	15496	1149.165	74.158			
150	22344	2427.069	108.623			
175	30432	4453.112	146.330			
200	39780	7673.705	192.904			
225	50372	12004.451	238.316			
250	62210	18295.080	294.086			

n	d = 5			d = 6		
	#cell	total [s]	per cell [ms]	#cell	total [s]	per cell [ms]
10	512	2.090	4.082	764	3.762	4.924
15	2942	16.654	5.660	6946	66.424	9.562
20	10072	79.546	7.897	33328	479.358	14.383
25	25902	314.498	12.141	110910	1974.542	17.803
30	55682	840.123	15.087	293192	6511.729	22.209
35	105912	1950.940	18.420			
40	184342	4198.192	22.773			

Table 1: Number of cells, total computation time and time per cell for some small to medium problems.

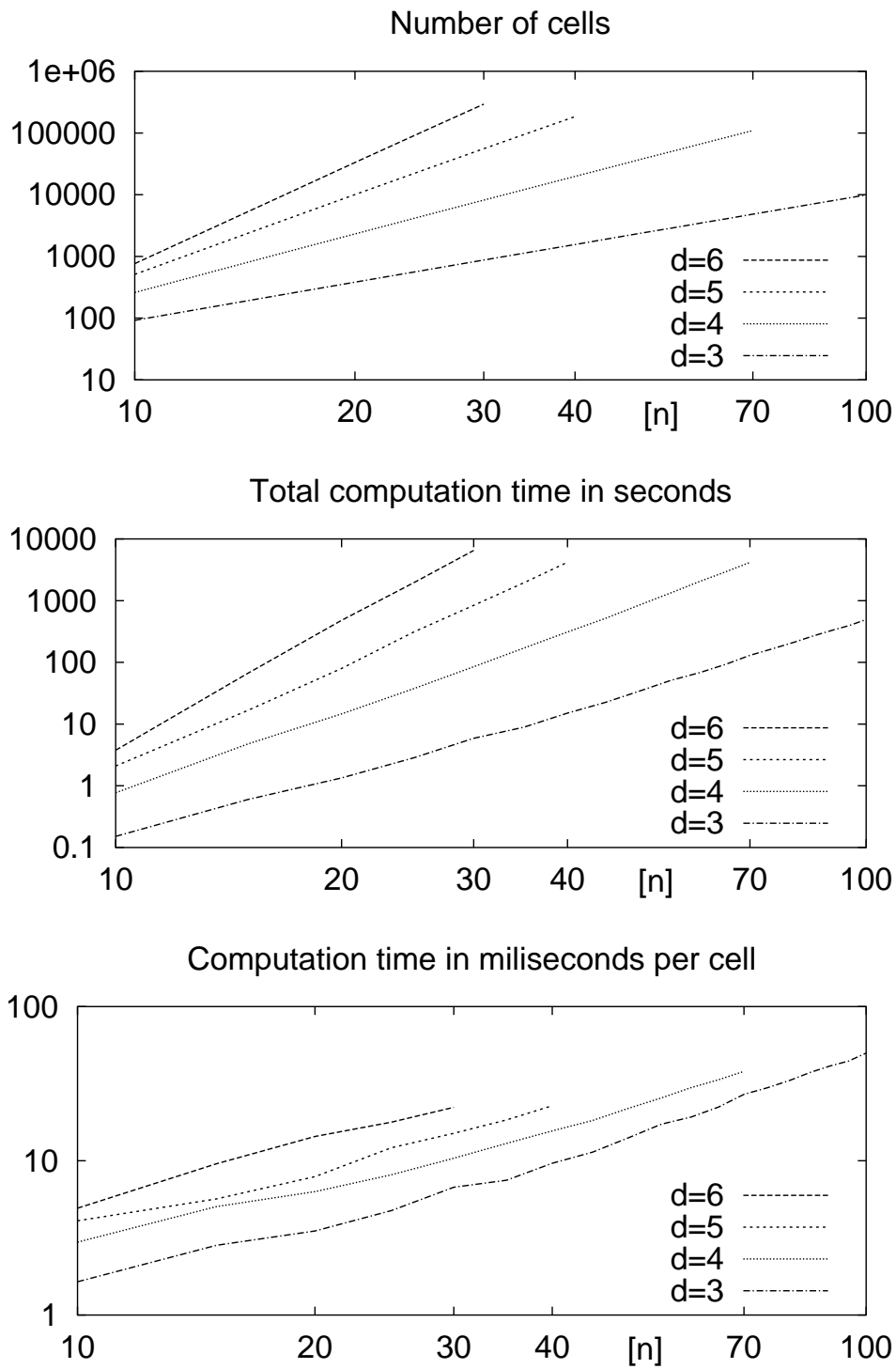


Figure 5.7: Number of cells, total computation time and time per cell for some small to medium problems.

Speedup of the parallel computation

Table 2 shows computation times for various numbers of processors and the speedup, defined as the ratio between the time on 1 processor and the time on p processors. Up to 8 processors were used, provided by two quadri-processors Intel Pentium-based machines.

#proc	n200d3			n30d6		
	time [s]	speedup	eff. [%]	time [s]	speedup	eff. [%]
1	11616.53	—	—	10242.47	—	—
2	5928.98	1.95	97.96	5143.86	1.99	99.56
3	4131.63	2.81	93.72	3460.82	2.95	98.65
4	3279.41	3.54	88.55	2611.68	3.92	98.04
5	2579.75	4.50	90.05	2122.89	4.82	96.49
6	2194.18	5.29	88.23	1784.35	5.74	95.66
7	1925.32	6.03	86.19	1540.64	6.64	94.97
8	1703.90	6.81	85.22	1348.56	7.59	94.93

Table 2: Computation time and speedup on 1 to 8 processors.

As can be seen, one instance scales much better than the other. This is due to the total number of cells to be generated, 39780 for n200d3 and 293192 for n30d6. With more cells, it is possible to keep a better load balance on all processors, as can be seen in table 3 where the number of cells found by each of the 8 processors is reported. The obvious conclusion is that parallel computing works better for very large problems, which is precisely what we want.

proc no	n200d3		n30d6	
	#cell	%	#cell	%
1	4870	12.242	38184	13.023
2	6138	15.429	36144	12.327
3	4568	11.483	38054	12.979
4	<i>4494</i>	<i>11.297</i>	<i>34214</i>	<i>11.669</i>
5	5206	13.087	35802	12.211
6	5134	12.906	36728	12.526
7	4686	11.779	37246	12.703
8	4684	11.774	36820	12.558

Table 3: Workload balance on 8 processors. For each problem, the largest part is shown in **bold** and the smallest in *italics*.

Floating point vs. rational arithmetics

On very large cases, we ran into numerical problems due to the intrinsic imprecision of floating point computation, a very common problem in computational geometry. Switching to rational arithmetics (using the GMP library in our case) provided robust computations at the cost of heavily degraded performances, as can be seen in table 4:

input size		#cell	comp. time [s]		ratio
d	n		floating point	rational	
3	20	382	1.338	18.561	13.86
4	20	2320	14.640	254.177	17.36
5	20	10072	79.546	1581.368	19.88
6	20	33328	479.358	13146.854	27.43

Table 4: Comparison of floating point and rational arithmetics for small problems. Floating point computations usually fail on very large problems due to rounding errors.

For all the cases we ran, the floating point computation either succeeds, returning the same results as the rational computation, or fails with an explicit error message such as the detection of an infeasible LP when looking for an interior point for a cell.

As expected, the slowdown ratio grows with the dimension of the problem, as the size of the rational numbers involved also increases. Furthermore, this is confirmed by the fact that the size n of the largest problem solvable in floating point decreases as the dimension d increases.

A heuristic method derived from the RS

By taking the *adjacency oracle* and *minimization direction* out of the reverse search framework, we obtained a randomized heuristic that works as follows: choose an initial cell at random (as we did to define the minimization direction) and repeatedly use the adjacency oracle to move to an adjacent cell as long as one improving the current solution is found. If none is found, we are in a local optimum and stop, without any indication on the quality of the solution.

Table 5 reports the results we have obtained with this method for some of the test problems. Out of 10 runs, between 5 and 9 found the optimal solution, with average paths length between the random initial cell and the optimal cell ranging between 5 and 20 hops, growing with problem size. It should be noted, though, that the value of local optima can be quite different from the optimal value, but this is balanced by the high proportion of successful searches.

The random nature of this heuristic makes it tedious to report computation times. They are however always several orders of magnitude smaller than the (deterministic) complete reverse search times reported in table 1.

input size		#successes (over 10 runs)	average path length	maximum gap to optimum of the worst solution
d	n			
3	10	9	4.3	70.3%
3	30	8	9.2	68.7%
3	50	6	15.8	39.1%
3	70	8	19.2	62.8%
4	30	5	8.1	5.5%
5	30	9	10.3	37.2%
6	30	7	12.7	66.8%

Table 5: Measured behavior of the heuristic method (seven instances, ten runs each).

6 Concluding Remarks

As we observed in Section 3, every rank d FRC 01QP problem is equivalent to the Euclidean norm maximization over a d -dimensional zonotope. One might suspect that a simplex-method-type algorithm works for the latter problem which traces the graph of a zonotope by moving from an extreme point to an adjacent one with larger Euclidean norm until it cannot find any better neighbor. Actually this algorithm is essentially the same as the heuristic algorithm presented in Section 5.

Figure 6.8 shows the **norm-oriented** graph of a 3 dimensional zonotope with 10 generators, where the orientation shows the augmenting direction of the Euclidean norm. This is a randomly generated instance and it has three sinks. We have observed that having multiple sinks is highly likely in randomly generated instances. Using the geometry of zonotopes, for any fixed $k > 0$, one can construct a zonotope for which the ratio of the largest norm and the smallest norm of sinks is at least k . This shows that the dumb local search algorithm finding one sink cannot guarantee its performance. It will be of great interest if we can analyze the expected behavior of any randomized algorithm.

Another interesting problem is to study the length of a monotone path from any given extreme point to a local optimum (i.e. sink). As we observed by our heuristics in Figure 5, randomized paths tend to be very short relative to the total number of extreme points. On the other hand, the paper [17] shows the existence of a function defined on the extreme points of the 0/1 d -cube for which the length of the greedy (i.e. maximum improvement) path from the minimum valued vertex to the maximum valued vertex is of exponential length. Understanding the norm-oriented graph of a zonotope is a challenging subject of our future research.

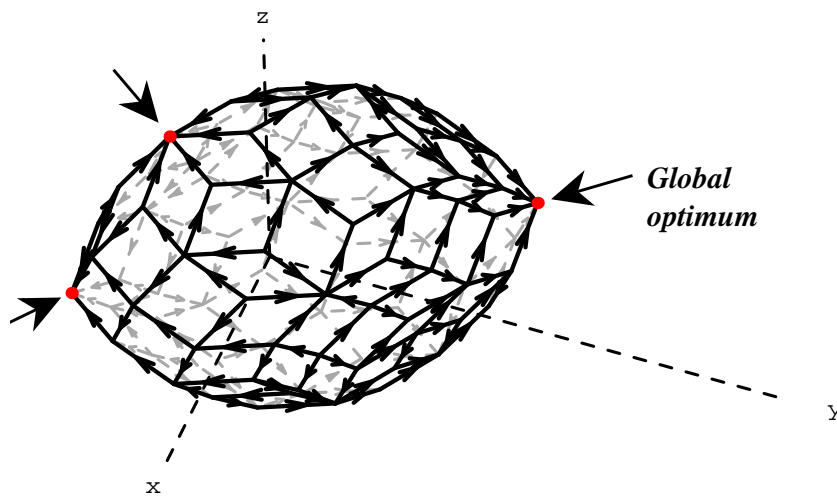


Figure 6.8: The three sinks of the norm-oriented graph of a 3-zonotope.

References

- [1] K. Allemand, K. Fukuda, Th. M. Lieblich, and E. Steiner. A polynomial case of unconstrained zero-one quadratic optimization. *Mathematical Programming, Ser. A*, 91:49–52, 2001.
- [2] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996.

- [3] F. Barahona. A solvable case of quadratic 0-1 programming. *Discrete Applied Mathematics*, 13:23–26, 1986.
- [4] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical, nuclear and general*, 15:3241–3253, 1982.
- [5] P. Bratley and B. Fox and L. Schrage. *A guide to simulation*. Springer-Verlag, 1987.
- [6] R. C. Buck. Partition of space. *Amer. Math. Monthly*, 50:541–544, 1943.
- [7] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt and G. Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. *Journal of Statistical Physics*, 80:487–496, 1995.
- [8] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, 1987.
- [9] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15:341–363, 1986.
- [10] J.A. Ferrez, K. Fukuda and T.M. Liebling,. Solutions to random instances of the 01QP obtained by the parallel zonotope construction code `rs_tope.c`. http://www.cs.mcgill.ca/~fukuda/download/cutzono_solutions.tar.gz.
- [11] K. Fukuda. *cddlib reference manual, cddlib Version 092b*. Swiss Federal Institute of Technology, Lausanne and Zurich, Switzerland, 2002.
- [12] K. Fukuda and J.A. Ferrez,. Implementations of LP-based reverse search algorithms for the zonotope construction and the fixed-rank convex quadratic maximization in binary variables using the ZRAM and the cddlib libraries. http://www.cs.mcgill.ca/~fukuda/download/mink/RS_TOPE020713.tar.gz.
- [13] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.
- [14] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.
- [15] P. L. Hammer, P. Hansen, P. M. Pardalos, and D. J. Rader. Maximizing the product of two linear functions in 0-1 variables. Research report, RUTCOR, Rutgers University, 1997. available from <http://rutcor.rutgers.edu/~rrr/1997.html>.
- [16] P. L. Ivănescu (Hammer). Some network flow problems solved with pseudo-boolean programming. *Operations Research*, 13:388–399, 1965.
- [17] P. L. Hammer, B. Simeone, Th. M. Liebling and D. de Werra. From linear separability to unimodality: A hierarchy of pseudo-boolean functions. *SIAM J. Disc. Math.*, 1:174–184, 1988.
- [18] A. Marzetta. ZRAM homepage. <http://www.cs.unb.ca/profs/bremner/zram/>.
- [19] J. C. Picard and H. D. Ratliff. Minimum cuts and related problems. *Networks*, 5:357–370, 1974.
- [20] T. Zaslavsky. *Facing up to arrangements: face-count formulas for partitions of space by hyperplanes*, volume 1(1): No. 154 MR 50 of *Mem. Amer. Math. Soc.* American Mathematical Society, 1975.
- [21] G. M. Ziegler. *Lectures on polytopes*. Graduate Texts in Mathematics 152. Springer-Verlag, 1994.