INFORMS Journal on Computing
Vol. 22, No. 1, Winter 2010, pp. 168–177
ISSN 1091-9856 | EISSN 1526-5528 | 10 | 2201 | 0168

DOI 10.1287/ijoc.1090.0318

Exact Algorithms for the Quadratic Linear Ordering Problem

Christoph Buchheim
Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 53113 Bonn, Germany,
buchheim@or.uni-bonn.de

Angelika Wiegele
Institut für Mathematik, Alpen-Adria-Universität Klagenfurt, 9020 Klagenfurt, Austria,
angelika.wiegele@uni-klu.ac.at

Lanbo Zheng
NICTA, The University of New South Wales, Sydney, New South Wales 2052, Australia,
lanbo.zheng@nicta.com.au

The quadratic linear ordering problem naturally generalizes various optimization problems such as bipartite crossing minimization or the betweenness problem, which includes linear arrangement. These problems have important applications, e.g., in automatic graph drawing and computational biology. We present a new polyhedral approach to the quadratic linear ordering problem that is based on a linearization of the quadratic objective function.

Our main result is a reformulation of the 3-dicycle inequalities using quadratic terms. After linearization, the resulting constraints are shown to be face-inducing for the polytope corresponding to the unconstrained quadratic problem. We use this result both within a branch-and-cut algorithm and within a branch-and-bound algorithm based on semidefinite programming. Experimental results for bipartite crossing minimization show that this approach clearly outperforms other methods.

Key words: quadratic optimization; linear ordering; crossing minimization
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received January 2008; revised July 2008, December 2008; accepted January 2009. Published online in *Articles in Advance* May 19, 2009.

1. Introduction

The linear ordering problem is one of the classical NP-hard combinatorial optimization problems (Garey and Johnson 1979). A linear ordering of a given finite set S is a permutation of its elements. Assuming without loss of generality that $S = \{1, \dots, n\}$, as we will always do in the following, a linear ordering of S is just a permutation $\pi \in S_n$.

In the linear ordering problem, the costs of a permutation depend on the order of the elements in a pairwise fashion: for each two elements $i, j \in \{1, \dots, n\}$, one may specify costs arising in the case $\pi(i) < \pi(j)$ and costs arising in the case $\pi(i) > \pi(j)$. In the usual integer programming model for the linear ordering problem, we thus have binary variables x_{ij} specifying whether $\pi(i) < \pi(j)$ or not. Since $x_{ij} + x_{ji} = 1$ and $x_{ii} = 0$, we need only one variable for each unordered pair of elements; thus in total we have $\binom{n}{2}$ variables.

In this paper, we consider the quadratic version of this problem, motivated by applications in automatic graph drawing and computational biology. We still model the permutations in S_n, but the costs may now depend on products of the variables, i.e., on the simultaneous satisfaction of two relations $\pi(i) < \pi(j)$ and $\pi(k) < \pi(l)$. Usually, going from linear to quadratic objective functions makes an optimization problem much harder. In the unconstrained case, the trivial problem of optimizing a linear function over a hypercube becomes the unconstrained quadratic binary optimization problem, which is equivalent to the maximum cut problem (De Simone 1990), and thus becomes NP-hard. Even if the linear variant of the problem at hand is already NP-hard, as in the case of linear ordering, the practical hardness usually increases significantly. According to our experience, in most cases polyhedral knowledge about a linear problem is rather useless for solving the quadratic variant, because even facet-inducing inequalities for the linear problem might become very weak constraints for the (linearized) quadratic version of the same problem.

To develop cutting plane algorithms for the exact solution of linear ordering problems, polyhedral methods have been investigated intensively; for early references, see Grötschel et al. (1985) and Reinelt (1985). Following the above model, we have to enforce

transitivity: if $\pi(i) < \pi(j)$ and $\pi(j) < \pi(k)$, then we need $\pi(i) < \pi(k)$. Usually, transitivity is modeled by the so-called 3-dicycle inequalities

$$0 \le x_{ij} + x_{jk} - x_{ik} \le 1,$$

for $i < j < k$. The main idea presented in this paper is to replace 3-dicycle inequalities by quadratic equations, which are then linearized along with the objective function. These equations are equivalent to 3-dicycle inequalities if integrality is required, but if the integrality is dropped and the resulting LP-relaxation is considered, the new constraints are much stronger. In fact, we can show that the new constraints induce faces of the corresponding unconstrained quadratic 0–1 optimization problem.

The same quadratic reformulation of 3-dicyle inequalities has been used by Lewis et al. (2009) to derive penalty functions for the original linear ordering problem. They solve the resulting quadratic problem heuristically. However, they do not investigate polyhedral properties of the new constraints because they do not consider any linearization of the quadratic problem.

Our approach thus allows us to reduce the quadratic linear ordering problem to unconstrained quadratic 0–1 optimization in an elegant way. In particular, we can combine this reduction with an arbitrary polyhedral approach to the latter problem. In this paper, we are particularly interested in comparing integer linear programming (ILP)- and semidefinite programming (SDP)-based methods. Experimentally, our reduction turns out to be very efficient for both methods. The question of how to solve quadratic 0–1 optimization problems quickly in practice has attracted a lot of interest recently. In particular, Billionnet and Elloumi (2007) experimentally compared approaches based on linearization with those taking advantage of convexity of the objective function. They also compared their results with those obtained by an SDP approach to max-cut; see also Rendl et al. (2007).

This paper is organized as follows. In §2, we present the polyhedral results motivating our approach to quadratic linear ordering. In §3, we explain how these results can be used in ILP- and SDP-based algorithms. Finally, in §4 we present an experimental evaluation of these ideas, applied to the bipartite crossing minimization problem.

2. Polyhedral Results

For a permutation $\pi \in S_n$, we define a characteristic vector $\chi(\pi) \in \{0,1\}^{\binom{n}{2}}$ as follows: for all $1 \le i < j \le n$, we set

$$\chi(\pi)_{ij} = \begin{cases} 1 & \text{if } \pi(i) < \pi(j), \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathrm{LO}(n)$ denote the linear ordering polytope on n elements, i.e., the polytope

$$\mathrm{LO}(n) = \mathrm{conv}\{\chi(\pi) \mid \pi \in S_n\} \subseteq \mathbf{R}^{\binom{n}{2}}.$$

In the following, we consider the quadratic linear ordering problem:

$$\max x^\top C x \quad \text{s.t.} \quad x \in \mathrm{LO}(n) \cap \{0,1\}^{\binom{n}{2}}, \tag{1}$$

where C is a real $\binom{n}{2} \times \binom{n}{2}$ matrix. Let I denote the set

$$I = \big\{(i,j,k,l) \mid i<j \text{ and } k<l \text{ and } (i<k \text{ or } (i=k \text{ and } j<l))\big\}.$$

In the standard linearization of problem (1), each product $x_{ij}x_{kl}$ is replaced by a new binary variable y_{ijkl}, for $(i,j,k,l) \in I$. The resulting integer program reads

$$\begin{aligned} \max \quad & \sum_{(i,j,k,l)\in I} (c_{ijkl} + c_{klij}) y_{ijkl} + \sum_{i<j} c_{ij} x_{ij} \\ \text{s.t.} \quad & x \in \mathrm{LO}(n), \\ & y_{ijkl} \le x_{ij}, x_{kl} \quad \text{for all } (i,j,k,l) \in I, \\ & y_{ijkl} \ge x_{ij} + x_{kl} - 1 \quad \text{for all } (i,j,k,l) \in I, \\ & y_{ijkl} \in \{0,1\} \quad \text{for all } (i,j,k,l) \in I, \\ & x_{ij} \in \{0,1\} \quad \text{for all } i<j. \end{aligned} \tag{2}$$

Our aim is to understand the polytope $\mathrm{QLO}(n)$ that is spanned by all feasible solutions of problem (2). We will show that the condition $x \in \mathrm{LO}(n)$ can be replaced by constraints that induce a face of the remaining unconstrained problem. More precisely, we consider

$$\begin{aligned} \max \quad & \sum_{(i,j,k,l)\in I} (c_{ijkl} + c_{klij}) y_{ijkl} + \sum_{i<j} c_{ij} x_{ij} \\ \text{s.t.} \quad & y_{ijkl} \le x_{ij}, x_{kl} \quad \text{for all } (i,j,k,l) \in I, \\ & y_{ijkl} \ge x_{ij} + x_{kl} - 1 \quad \text{for all } (i,j,k,l) \in I, \\ & y_{ijkl} \in \{0,1\} \quad \text{for all } (i,j,k,l) \in I, \\ & x_{ij} \in \{0,1\} \quad \text{for all } i<j, \end{aligned} \tag{3}$$

which is the standard linearization of an unconstrained quadratic optimization problem over the binary variables x_{ij}. Denote by P_n the convex hull of all feasible solutions of (3). By a well-known result of De Simone (1990), the polytope P_n is isomorphic to a cut polytope that corresponds to a graph on $\binom{n}{2}+1$ nodes.

We will show that $\mathrm{QLO}(n)$ is a face of P_n. To this end, we will model the constraint $x \in \mathrm{LO}(n)$, for integer x, by a set of quadratic equations, each inducing a face of P_n. Usually, the linear ordering problem is modeled by introducing the 3-dicycle inequalities

$$0 \le x_{ij} + x_{jk} - x_{ik} \le 1, \tag{4}$$

for all $i < j < k$. In the following, we present a different way to model transitivity, using the presence of product variables y_{ijkl}.

LEMMA 1. *Let (x, y) be an integer vector in P_n. Then $(x, y) \in \mathrm{QLO}(n)$ if and only if*

$$x_{ik} - y_{ijik} - y_{ikjk} + y_{ijjk} = 0,$$

for all $i < j < k$. These equations form a minimal equation system for $\mathrm{QLO}(n)$.

PROOF. First note that $(x, y) \in \mathrm{QLO}(n)$ is equivalent to $x \in \mathrm{LO}(n)$, as (x, y) is integer. So assume $x \in \mathrm{LO}(n)$. Then $x = \chi(\pi)$ for a permutation $\pi \in S_n$. If $\pi(i)$ is between $\pi(j)$ and $\pi(k)$, we have $x_{ik} - x_{jk} = 0$; otherwise, $x_{ik} - x_{ij} = 0$. Thus

$$0 = (x_{ik} - x_{jk})(x_{ik} - x_{ij}) = x_{ik} - y_{ijik} - y_{ikjk} + y_{ijjk}$$

is a valid equation for x.

On the other hand, these constraints imply that $x \in \mathrm{LO}(n)$. To check this, we have to derive transitivity. So let $x_{ij} = x_{jk} = 1$. Then $(x_{ik} - x_{jk})(x_{ik} - x_{ij}) = 0$ reduces to $x_{ik} - 1 = 0$, and hence $x_{ik} = 1$. If $x_{ij} = x_{jk} = 0$, we derive $x_{ik} = 0$.

It remains to show that the given equation system is minimal. Clearly, the matrix defined by the equation system has full row rank, as every y-variable appears in exactly one equation. Now consider any equation $a^\top(x, y) = b$ that is valid for $\mathrm{QLO}(n)$. For the decreasing permutation $n, \dots, 1$, we have $\pi(i) = n - i + 1$, and therefore $\pi(i) > \pi(j)$ for $1 \le i < j \le n$. Hence, the characteristic vector of the decreasing permutation is the zero vector and $(x, y) = 0$ is a feasible solution of Problem (2); i.e., the zero vector is contained in $\mathrm{QLO}(n)$. This implies $b = 0$.

To show that $a^\top(x, y) = 0$ is a linear combination of the given constraints, it suffices to show the following:

(a) $a_{ik} = -\sum_{j=i+1}^{k-1} a_{ijik} = -\sum_{j=i+1}^{k-1} a_{ikjk}$ for all $i < k$;

(b) $a_{ijkl} = 0$ for all $i < j$ and $k < l$ with pairwise distinct i, j, k, l; and

(c) $a_{ijjk} = -a_{ijik} = -a_{ikjk}$ for all $i < j < k$.

For the following, let π_{ij}^{st} denote the decreasing permutation $n, \dots, 1$, where first j is moved t positions to the left and then i is moved s positions to the left. If t (s) is negative, then j (i) is moved $-t$ ($-s$) positions to the right.

To show (a), consider the permutations $\pi_{ik}^{(k-i-1)0}$ and $\pi_{ik}^{(k-i)0}$. It is easy to verify that

$$a^\top \chi(\pi_{ik}^{(k-i-1)0}) = a^\top \chi(\pi_{ik}^{(k-i)0})$$

yields $a_{ik} + \sum_{j=i+1}^{k-1} a_{ijik} = 0$. Analogously,

$$a^\top \chi(\pi_{ik}^{0(-k+i+1)}) = a^\top \chi(\pi_{ik}^{0(-k+i)})$$

yields $a_{ik} + \sum_{j=i+1}^{k-1} a_{ikjk} = 0$.

For (b), we show $a_{i(i+s)k(k+t)} = 0$ for all $s, t \ge 0$ with $i + s \ne k, k + t$. First assume $i + s < k$. In this case, $a^\top \chi(\pi_{ik}^{st}) = a^\top \chi(\pi_{ik}^{(s-1)t})$ translates to

$$a_{i(i+s)} + \sum_{r=1}^{s-1} a_{i(i+r)i(i+s)} + \sum_{r=1}^{t} a_{i(i+s)k(k+r)} = 0\,. \tag{5}$$

Using (a), we derive $\sum_{r=1}^{t} a_{i(i+s)k(k+r)} = 0$. For $t = 1$, we get $a_{i(i+s)k(k+1)} = 0$. For $t > 1$, we derive $a_{i(i+s)k(k+t)} = 0$ by induction. Now assume $i + s > k$. If $i + s < k + t$, then

$$a^\top \chi(\pi_{ik}^{st}) = a^\top \chi(\pi_{ik}^{s(t-1)})$$

yields

$$a_{k(k+t)} + \sum_{r=1}^{t-1} a_{k(k+r)k(k+t)} + \sum_{r=1}^{s} a_{i(i+r)k(k+t)} - a_{ikk(k+t)} = 0.$$

Thus $\sum_{r=1}^{s} a_{i(i+r)k(k+t)} - a_{ikk(k+t)} = 0$ by (a). By induction, $a_{i(i+r)k(k+t)} = 0$ for all $r < s$ with $i + r \ne k$; thus $a_{i(i+s)k(k+t)} = 0$. If $i + s > k + t$, we get (5) again, and as above, induction over t yields $a_{i(i+s)k(k+t)} = 0$.

Finally, for (c), we consider $a^\top \chi(\pi_{ij}^{(k-i)(k-j)}) = 0$. Using (a), this equation reduces to

$$\sum_{r_1=1}^{k-i} \sum_{r_2=1}^{k-j} a_{i(i+r_1)j(j+r_2)} = 0$$

so that applying (b) we are left with $\sum_{r_2=1}^{k-j} (a_{ijj(j+r_2)} + a_{i(j+r_2)j(j+r_2)}) = 0$. By induction over $k - j$ we derive $a_{ijjk} + a_{ikjk} = 0$ for all $k > j$. Similarly, we derive $a_{ijjk} + a_{ijik} = 0$. □

THEOREM 1. *We have*

$$\dim \mathrm{QLO}(n) = \binom{n}{2} + \binom{\binom{n}{2}}{2} - \binom{n}{3}.$$

PROOF. The polytope $\mathrm{QLO}(n)$ is defined in dimension $\binom{n}{2}$ for the x-variables plus $|I| = \binom{\binom{n}{2}}{2}$ for the y-variables. By Lemma 1, a minimal equation system for $\mathrm{QLO}(n)$ in this space contains $\binom{n}{3}$ equations, hence the result. □

LEMMA 2. *For all $i < j < k$, the constraint $x_{ik} - y_{ijik} - y_{ikjk} + y_{ijjk} = 0$ induces a face of the polytope P_n.*

PROOF. It suffices to show that the inequality

$$x_{ik} - y_{ijik} - y_{ikjk} + y_{ijjk} \ge 0$$

is valid for P_n for all $i < j < k$. As before, we show that

$$x_{ik} - y_{ijik} - y_{ikjk} + y_{ijjk} = (x_{ik} - x_{jk})(x_{ik} - x_{ij}).$$

The result follows from the fact that either $x_{ik} = x_{jk}$ or $x_{ik} = x_{ij}$ or $x_{jk} = x_{ij}$; thus the right-hand side becomes either zero or $(x_{ik} - x_{jk})^2$. □

THEOREM 2. *The polytope* QLO(n) *is isomorphic to a face of* P_n *and hence to a face of a cut polytope.*

PROOF. By Lemmas 1 and 2, the polytope QLO(n) is contained in the face F of P_n that is induced by the constraints $x_{ik} - y_{ijik} - y_{ikjk} + y_{ijjk} = 0$. By definition, P_n is an integer polytope. In particular, its face F is spanned by integer vectors. Hence, it remains to show that every integer vector in F is contained in QLO(n). This is the if-part of Lemma 1. □

3. Solution Methods

Our aim is to use Theorem 2 for practical computation. In this section, we propose two different approaches: a branch-and-cut algorithm (§3.1) and a branch-and-bound algorithm based on semidefinite programming (§3.2).

3.1. Branch and Cut

Theorem 2 implies that every valid inequality for QLO(n) is induced by a valid inequality for P_n. In particular, all facets of QLO(n) are intersections of facets of P_n with the affine space determined by the constraints of Lemma 1.

We start our branch-and-cut algorithm for quadratic linear ordering by combining these constraints with any ILP modeling the cut polytope P_n. Whenever we desire to separate a vector $\bar{x}$ from QLO(n), we ignore the fact that we have additional constraints. Instead, we feed $\bar{x}$ into any separation algorithm for the cut polytope P_n. Any returned cutting plane will be valid for the polytope QLO(n) as well.

From the discussion above, it is clear that if we had an exact separation algorithm for P_n, this reduction would yield an exact separation algorithm for QLO(n) as well, by Theorem 2. Clearly, as the maximum cut problem is NP-hard, we do not know any exact separation algorithm for P_n in practice, except for tiny values of n. However, the separation problem for cut polytopes has been well studied; see, e.g., Barahona and Mahjoub (1986), Laurent (1997), and Liers et al. (2004). Theorem 2 allows us to carry over all these results without further work.

3.2. Semidefinite Programming

An exact algorithm for solving the maximum cut problem based on semidefinite programming has been introduced by Rendl et al. (2007). This approach uses the basic semidefinite programming relaxation for maximum cut together with the so-called triangle inequalities in a branch-and-bound framework. More precisely, at each node of the branch-and-bound tree the following SDP is solved approximately to obtain an upper bound on the maximum cut:

$$z_{sdp-met} = \max\{\langle L, X\rangle \mid X \in \mathcal{E},\ \mathcal{A}(X) \le e\}. \qquad (6)$$

Here, L is the Laplace matrix of the underlying graph, $\mathcal{E}$ denotes the elliptope, i.e., the set of positive semidefinite matrices with an all-ones diagonal of appropriate size, and $\mathcal{A}(X) \le e$ symbolically collects the triangle inequalities. $\langle L, X\rangle$ is the trace inner product; i.e., $\langle L, X\rangle = \operatorname{tr}(LX)$.

Since the number of triangle inequalities is of order $|V|^3$, with V being the set of vertices of the graph, solving (6) directly is impractical already for medium-sized graphs. In Rendl et al. (2007), a bundle method is used to solve this SDP approximately. To apply the bundle method, the constraints that make the problem hard to solve, i.e., the triangle inequalities, are dualized and lifted into the objective function. Hence, we obtain the Lagrangian

$$\mathcal{L}(X;\gamma) := \langle L, X\rangle + \gamma^\top(e - \mathcal{A}(X)) = e^\top\gamma + \langle L - \mathcal{A}^\top(\gamma), X\rangle.$$

The problem to be solved now reads

$$z_{sdp-met} = \min_{\gamma \ge 0} f(\gamma),$$

where f is the convex, but nonsmooth function

$$f(\gamma) := \max_{X\in\mathcal{E}} \mathcal{L}(X;\gamma) = e^\top\gamma + \max_{X\in\mathcal{E}} \langle L - \mathcal{A}^\top(\gamma), X\rangle. \quad (7)$$

The triangle inequalities $\mathcal{A}(X) \le e$ are handled dynamically; i.e., in the course of the algorithm, the set of constraints is updated from time to time: newly violated constraints are added, whereas inequalities with a dual multiplier close to zero are removed.

As discussed in §2, solving the quadratic linear ordering problem amounts to solving a maximum cut problem with a few additional equations, namely, those given in Lemma 1. We can add these constraints as inequalities, as one of the directions is implied by the maximum cut formulation; see the proof of Lemma 2. Let us denote the remaining $\binom{n}{3}$ inequalities in terms of the matrix variable X by $\mathcal{B}(X) \le b$. Adding these constraints to (6) and dualizing them, introducing dual variables μ, the function to be minimized becomes

$$\begin{aligned} f_{qlo}(\gamma,\mu) &= \max_{X\in\mathcal{E}} \mathcal{L}(X;\gamma,\mu) \\ &= e^\top\gamma + b^\top\mu + \max_{X\in\mathcal{E}} \langle L - \mathcal{A}^\top(\gamma) - \mathcal{B}^\top(\mu), X\rangle. \end{aligned}$$

As in Rendl et al. (2007), the bundle method can be used to minimize f_{qlo} subject to $\gamma \ge 0$ and $\mu \ge 0$. While the triangle constraints are chosen dynamically, the problem-specific constraints $\mathcal{B}(X) \le b$ are present all the time, because the number of these constraints is small enough.

For experimenting with applications of the quadratic linear ordering problem, the parameters of the algorithm in Rendl et al. (2007) also have to be adjusted. This is discussed in §4.

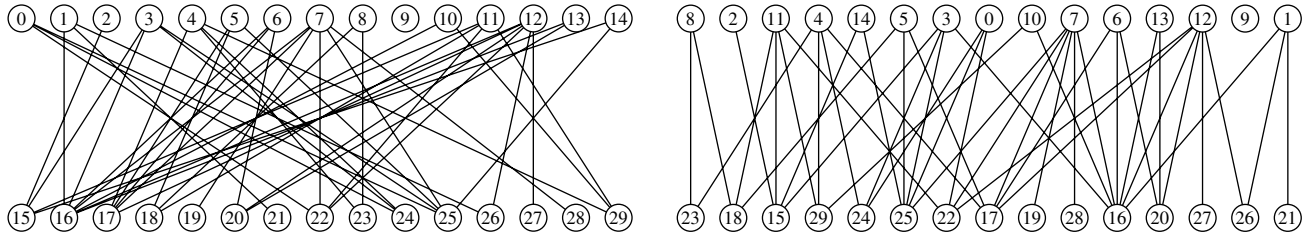

Figure 1 Two Drawings of the Same Graph; the Second Drawing Is Crossing Minimal

4. Computational Experiments

To evaluate the practical performance of our approach presented in the previous section, we performed a computational evaluation using instances arising in bipartite crossing minimization. Given a bipartite graph, we try to determine a drawing of the graph where the vertices of the two layers are placed on two parallel lines and edges are drawn as straight lines; see Figure 1. The aim is to minimize the number of edge crossings in such a drawing. This number is obviously determined by the permutations of the vertices on both layers.

An exact algorithm for this problem has been introduced by Jünger and Mutzel (1997). They first consider the problem variant where the permutation on one of the two layers is fixed. Even then, the problem remains NP-hard (Eades and Wormald 1994). However, the problem now becomes a linear ordering problem: considering two edges of the graph, the existence of a crossing between them depends on the chosen order of their end-vertices on the free layer. Using a branch-and-cut algorithm for linear ordering, Jünger and Mutzel are able to compute exact solutions for practical instances very quickly, if one layer is fixed.

For the problem with two free layers, Jünger and Mutzel (1997) use this algorithm as a black box inside a branch-and-bound scheme. More precisely, the permutations on the smaller layer are enumerated, while the permutations on the larger layer are then determined by solving a linear ordering problem as above. Using intelligent bounding techniques, the resulting algorithm was shown to be fast on sparse instances with up to 15 vertices on the smaller layer.

Many heuristics are known and well used for the bipartite crossing minimization problem. In particular, heuristics for the case of one fixed layer can be applied repeatedly while alternating the layer being considered as fixed. As Jünger and Mutzel (1997) show, the results of these heuristic approaches in general are far away from the optimum in terms of the number of crossings. For instances on 10 vertices per layer and 10 edges, the solutions produced by the heuristics had between 424% and 3,214% more crossings than the optimal solutions; for 20 edges, the numbers were between 61% and 235%. This motivates the use of exact solution methods.

In this section, we experimentally compare the exact algorithm of Jünger and Mutzel (1997) for two free layers with the two approaches presented in §3. Clearly, the bipartite crossing minimization problem with two free layers can be modeled directly as a quadratic linear ordering problem, as the existence of a crossing is determined by the order of the end-vertices on both layers.

Notice, however, that bipartite crossing minimization is a special case of quadratic linear ordering: because we only have to care about the order of vertices within one layer of the bipartite graph, we do not need an ordering variable x_{ij} if i and j belong to different layers. Moreover, all products are taken between ordering variables belonging to different layers. The reason for us to focus on this special case is that it is a natural application of quadratic linear ordering and that there exist other algorithms for comparison. However, the general picture does not change if we apply our approach to general instances.

In this section, we additionally give results for an implementation using the CPLEX MIP-solver, applied to the standard linearization of the objective function in combination with the standard integer programming formulation of the linear ordering problem. In summary, we compared four algorithms:

- The exact algorithm introduced by Jünger and Mutzel (1997), which is denoted by `JM` in the remainder of this section. We use the original implementation that was also used for experiments in Jünger and Mutzel, except for an update to CPLEX 11.1.
- The standard linearization approach `LIN`, in which we apply the MIP-solver of CPLEX 11.1 to the linearized model (2), using default parameter settings. The constraint $x \in \mathrm{LO}(n)$ is realized by the linear 3-dicycle inequalities (4).
- The branch-and-cut algorithm `ILP` outlined in §3.1. We apply CPLEX to the linearized model (2), using the quadratic constraints of Lemma 1. We do not change default parameters except that cutting-plane generation is switched off. Instead, we call a separation routine for the corresponding cut polytope. In our implementation, the latter routine generates

violated cycle inequalities, using the exact method of Barahona and Mahjoub (1986).

- The SDP-based branch-and-bound approach outlined in §3.2, denoted by SDP in the remainder of this section. Our implementation uses the software BiqMac for unconstrained quadratic 0–1 optimization, described in Rendl et al. (2007).

An intermediate approach between LIN and ILP has been presented by Zheng and Buchheim (2007): separation for the cut polytope was used to strengthen the standard linearization, but 3-dicycle inequalities were not reformulated. In particular, the polyhedral results of §2 were not used. We will compare this approach to ILP.

To use BiqMac efficiently, we have to adjust the parameters according to our problem. An important parameter is the number of outer iterations of the bundle routine. Because of our problem sizes, we allow 30 outer iterations. The rounding heuristic inside the algorithm in Rendl et al. (2007) works astonishingly well for max-cut instances, and thus, in the original algorithm, it is called only twice at each node of the branch-and-bound tree. In the presence of additional constraints, the solution obtained by the heuristic has to be repaired; i.e., transitivity has to be restored if necessary. This is done in a straightforward left-to-right fashion. Because more attempts are needed to finally get to the optimal solution (and because the heuristic is computationally cheap anyway), we call the heuristic in every outer iteration. With this setting we hope to find the optimal solution and close the gap already in the root node, so that branching is not necessary.

For our experiments, we create graphs in the same way as in Jünger and Mutzel (1997): we generate random bipartite graphs using the function random_bigraph of the Stanford GraphBase (Knuth 1993). Results are reported for graphs having $n = 10, 12, \ldots, 24$ vertices on each layer. For each n, we consider graphs with density $d = 10, 20, \ldots, 90$, if $n \leq 16$, and with density $d = 10, 20$ otherwise. Density d means that the number of edges is $\lfloor dn^2/100 \rfloor$. For each pair (n, d), we report the averages over 10 random instances.

All experiments were carried out on an Intel Xeon processor with 2.33 GHz; the results are shown in Table 1. For all (n, d) and for all four approaches tested, we state the number of instances being solved within one CPU-hour, and the average running times for solved instances in CPU-seconds. For the three last methods, we also state the average number of nodes in the enumeration tree for solved instances. Note that all algorithms are started from scratch so that they both have to find optimal solutions and prove their optimality.

Our results confirm that the JM algorithm is very fast on sparse and small graphs: it is the fastest of the compared methods for $n = 10$ and $20 \leq d \leq 80$ as well as for $n = 12$ and $20 \leq d \leq 40$. However, the running time increases sharply with density. For $n = 14$, the runtime limit of one hour is reached for the first time at $d = 40$ while it is reached already for $d = 10$ if $n \geq 16$.

For the LIN approach, the effect of density is even more evident. Although it is the fastest method for all n if $d = 10$, it is much slower than all other methods on denser instances. Even for $n = 10$, two instances cannot be solved within one CPU-hour. For $n = 14$, the limit is reached at $d = 30$, for $16 \leq n \leq 20$ at $d = 20$, and for $n \geq 22$ at $d = 10$.

The ILP approach is considerably faster than LIN on denser instances and is less affected by an increase in density, but it is still unable to solve large and dense instances. However, it is obvious from the number of nodes needed in the enumeration tree that the polyhedral results given in §2 are strong: for the solved instances, branching was almost never necessary; most instances could be solved in the root node. In terms of running time, ILP clearly beats both JM and LIN on larger instances.

However, the clear winner of our evaluation is the SDP approach. It could solve all instances with $n \leq 14$ and most instances with $n \geq 16$ in time. Particularly for denser instances, the difference between SDP and the other approaches is impressive. For example, for $n = 14$ and each $40 \leq d \leq 90$, the solution was computed after less than three minutes on average, whereas no other method could solve all 10 instances in one hour.

In Table 1, one can observe a decrease of running time for very dense instances both for LIN and ILP. The reason is that the percentage of product variables needed is actually not proportional to d. On the contrary, the expected percentage of products with nonzero coefficient is $d^2 - d^4$. This is because the expected number of $K_{2,2}$ subgraphs increases with d. Every such subgraph has exactly one crossing in every ordering. Algebraically, the corresponding coefficients cancel out each other. The maximum of $d^2 - d^4$ is attained at $d = 1/\sqrt{2} \approx 70.7\%$, beyond which it decreases to zero for $d = 100\%$.

It is noticeable in Table 1 that the number of nodes in the enumeration tree that are needed to solve an instance to optimality is very small in general, both for the ILP and the SDP approach. In fact, most instances can be solved without any branching—if they can be solved at all within the time limit of one hour. This demonstrates the strength of our polyhedral results derived in §2.

On the other hand, this means that most of the running time is spent in the root node in general.

Table 1 Results for Bipartite Graphs with Increasing Size and Density (Limit One Hour)

		JM		LIN			ILP			SDP		
n	*d*	No.	Time	No.	Time	Nodes	No.	Time	Nodes	No.	Time	Nodes
10	10	10	0.02	10	0.01	1.0	10	0.30	1.0	10	1.16	1.0
10	20	10	0.05	10	0.74	9.4	10	1.01	1.0	10	2.25	1.0
10	30	10	0.15	10	14.95	329.5	10	4.55	1.0	10	4.77	1.0
10	40	10	0.33	10	51.20	823.8	10	12.17	1.0	10	5.07	1.0
10	50	10	0.61	10	180.86	2,922.4	10	18.31	1.0	10	4.71	1.0
10	60	10	1.14	10	738.58	12,095.0	10	27.34	1.0	10	5.35	1.0
10	70	10	2.35	8	*1,225.62*	*21,176.5*	10	33.46	1.0	10	6.81	1.0
10	80	10	4.05	10	538.68	6,959.9	10	15.64	1.0	10	5.15	1.0
10	90	10	8.86	10	86.51	560.2	10	8.59	2.0	10	6.79	1.0
12	10	10	0.20	10	0.02	1.0	10	8.07	1.0	10	9.54	1.0
12	20	10	1.52	10	5.93	79.8	10	19.00	1.0	10	18.36	1.0
12	30	10	4.53	10	140.60	1,408.6	10	35.95	1.0	10	21.61	1.0
12	40	10	16.36	7	*1,808.35*	*17,897.7*	10	106.01	1.0	10	25.29	1.0
12	50	10	57.05	0	—	—	10	440.96	1.0	10	44.84	1.0
12	60	10	102.15	0	—	—	10	622.10	1.0	10	48.26	1.0
12	70	10	211.37	0	—	—	10	607.73	1.0	10	40.31	1.0
12	80	10	527.75	0	—	—	10	273.39	1.0	10	28.71	1.0
12	90	10	1,036.30	6	*1,693.75*	*4,083.2*	10	73.60	1.8	10	22.21	1.0
14	10	10	15.68	10	0.33	3.4	10	19.02	1.0	10	41.03	1.0
14	20	10	110.83	10	102.07	1,062.2	10	155.14	1.0	10	89.61	1.0
14	30	10	747.49	4	*1,267.86*	*9,748.8*	10	688.01	1.0	10	132.72	1.0
14	40	9	*1,432.45*	0	—	—	8	*1,667.63*	*1.0*	10	144.03	1.0
14	50	2	*2,718.05*	0	—	—	1	*1,453.35*	*1.0*	10	180.49	1.0
14	60	0	—	0	—	—	1	*2,594.94*	*1.0*	10	141.93	1.0
14	70	0	—	0	—	—	5	*2,177.86*	*1.0*	10	149.68	1.0
14	80	0	—	0	—	—	7	*1,829.18*	*1.0*	10	145.97	1.0
14	90	0	—	0	—	—	10	398.75	1.0	10	81.27	1.0
16	10	8	*328.92*	10	2.77	32.9	10	190.83	1.0	10	124.57	1.0
16	20	5	*2,220.12*	7	*809.30*	*4,125.1*	9	*882.19*	*1.0*	10	309.31	1.0
16	30	0	—	0	—	—	4	*2,112.61*	*1.0*	10	630.77	1.2
16	40	0	—	0	—	—	0	—	—	9	*800.87*	*1.2*
16	50	0	—	0	—	—	0	—	—	7	*451.09*	*1.0*
16	60	0	—	0	—	—	0	—	—	9	*403.82*	*1.7*
16	70	0	—	0	—	—	0	—	—	8	*789.62*	*1.2*
16	80	0	—	0	—	—	0	—	—	10	568.55	1.0
16	90	0	—	0	—	—	7	*2,373.15*	*1.0*	10	362.29	1.0
18	10	7	*1,010.78*	10	7.06	39.3	10	571.66	1.0	10	408.04	1.0
18	20	0	—	1	*2,166.05*	*10,671.0*	4	*2,563.59*	*1.0*	10	778.86	1.0
20	10	1	*1,112.12*	10	117.72	836.2	2	*1,756.11*	*1.0*	10	1,543.39	1.0
20	20	0	—	0	—	—	0	—	—	8	*1,813.87*	*1.0*
22	10	0	—	9	*546.71*	*2,710.3*	0	—	—	6	*2,658.64*	*1.0*
22	20	0	—	0	—	—	0	—	—	1	*3,443.81*	*1.0*
24	10	0	—	2	*2,225.82*	*7,360.0*	0	—	—	0	—	—
24	20	0	—	0	—	—	0	—	—	0	—	—

Notes. In Tables 1–3, running times for the fastest method are underlined. Numbers in italics are averages over less than 10 instances.

For harder instances, one hour does not even suffice to solve the root relaxation. For this reason, we also experimented with various methods to enforce earlier branching, using different tailing-off strategies for ILP or decreasing the number of outer iterations for SDP. For ILP, we also tried to separate cycle inequalities heuristically. However, all these variants led to longer running times in general and to a smaller number of solved instances.

In the ILP approach, the main reason for the long running time in the root node is the huge number of cycle inequalities being separated. In part, this is due to the size of the instances. Moreover, it is possible that cycle inequalities are separated that do not

Table 2 Results for ILP with Linear and Quadratic Constraints (Limit One Hour)

		ILP (linear constraints)				ILP (quadratic constraints)			
n	*d*	No.	Time	Nodes	Root	No.	Time	Nodes	Root
10	10	10	0.01	1.0	0.00	10	0.30	1.0	0.00
10	20	10	0.07	1.2	0.00	10	1.01	1.0	0.00
10	30	10	0.97	1.1	0.00	10	4.55	1.0	0.00
10	40	10	6.83	1.2	0.00	10	12.17	1.0	0.00
10	50	10	30.35	1.0	0.00	10	18.31	1.0	0.00
10	60	10	114.19	2.2	0.27	10	27.34	1.0	0.00
10	70	10	450.18	5.2	0.71	10	33.46	1.0	0.00
10	80	10	359.64	1.0	0.00	10	15.64	1.0	0.00
10	90	10	518.22	2.4	0.27	10	8.59	2.0	0.27
12	10	10	0.04	1.0	0.00	10	8.07	1.0	0.00
12	20	10	0.43	1.0	0.00	10	19.00	1.0	0.00
12	30	10	7.15	1.0	0.00	10	35.95	1.0	0.00
12	40	10	102.33	1.0	0.00	10	106.01	1.0	0.00
12	50	9	*1,275.08*	*13.8*	*1.58*	10	440.96	1.0	0.00
12	60	6	*2,081.05*	*1.0*	*0.00*	10	622.10	1.0	0.00
12	70	1	*332.87*	*1.0*	*0.00*	10	607.73	1.0	0.00
12	80	1	*2,448.13*	*1.0*	*0.00*	10	273.39	1.0	0.00
12	90	4	*1,774.03*	*1.5*	*0.00*	10	73.60	1.8	0.15
14	10	10	0.11	1.0	0.00	10	19.02	1.0	0.00
14	20	10	8.45	1.3	0.00	10	155.14	1.0	0.00
14	30	10	220.41	2.4	0.05	10	688.01	1.0	0.05
14	40	8	*1,602.86*	*1.0*	*0.00*	8	*1,667.63*	*1.0*	*0.00*
14	50	0	—	—	—	1	*1,453.35*	*1.0*	*0.00*
14	60	0	—	—	—	1	*2,594.94*	*1.0*	*0.00*
14	70	0	—	—	—	5	*2,177.86*	*1.0*	*0.00*
14	80	0	—	—	—	7	*1,829.18*	*1.0*	*0.00*
14	90	0	—	—	—	10	398.75	1.0	0.00
16	10	10	0.32	1.0	0.00	10	190.83	1.0	0.00
16	20	10	64.13	1.1	0.00	9	*882.19*	*1.0*	*0.00*
16	30	9	*1,736.68*	*1.0*	*0.00*	4	*2,112.61*	*1.0*	*0.00*
16	40	0	—	—	—	0	—	—	—
16	50	0	—	—	—	0	—	—	—
16	60	0	—	—	—	0	—	—	—
16	70	0	—	—	—	0	—	—	—
16	80	0	—	—	—	0	—	—	—
16	90	0	—	—	—	7	*2,373.15*	*1.0*	*0.00*
18	10	10	0.90	1.0	0.00	10	571.66	1.0	0.00
18	20	10	351.92	1.1	0.00	4	*2,563.59*	*1.0*	*0.00*
20	10	10	7.02	2.9	0.00	2	*1,756.11*	*1.0*	*0.00*
20	20	7	*1,735.38*	*1.0*	*0.00*	0	—	—	—
22	10	10	33.32	1.2	0.00	0	—	—	—
22	20	0	—	—	—	0	—	—	—
24	10	10	161.44	1.3	0.00	0	—	—	—
24	20	0	—	—	—	0	—	—	—

induce facets of the face of the cut polytope that corresponds to the quadratic linear ordering problem. It would be desirable to characterize the subset of facet-inducing cycle inequalities and to find a fast separation algorithm for this subset. This is future work.

For the SDP approach, the small number of nodes in the enumeration tree is less surprising. In fact, similar observations can be made when addressing the (pure) max-cut problem by an SDP-based branch-and-bound approach: the root relaxation is very strong in general, but on the other hand, branching is less effective compared with ILP approaches. This is mainly due to the lack of warm-start methods: after each branching, the new SDP relaxation has to be solved from scratch. Observe that for $n = 16$, the dimension of our SDP problem is 240. Solving pure max-cut

Table 3 Results for SDP with Linear and Quadratic Constraints (Limit One Hour)

		SDP (linear constraints)				SDP (quadratic constraints)			
n	*d*	No.	Time	Nodes	Root	No.	Time	Nodes	Root
10	10	10	2.41	1.0	0.00	10	1.16	1.0	0.00
10	20	10	2.21	1.0	0.00	10	2.25	1.0	0.00
10	30	10	4.66	1.0	0.00	10	4.77	1.0	0.00
10	40	10	5.23	1.0	0.00	10	5.07	1.0	0.00
10	50	10	4.97	1.0	0.00	10	4.71	1.0	0.00
10	60	10	5.31	1.0	0.00	10	5.35	1.0	0.00
10	70	10	6.80	1.0	0.00	10	6.81	1.0	0.00
10	80	10	5.49	1.0	0.00	10	5.15	1.0	0.00
10	90	10	6.35	1.0	0.00	10	6.79	1.0	0.00
12	10	10	9.84	1.0	0.00	10	9.54	1.0	0.00
12	20	10	15.81	1.0	0.00	10	18.36	1.0	0.00
12	30	10	21.24	1.0	0.00	10	21.61	1.0	0.00
12	40	10	26.23	1.0	0.00	10	25.29	1.0	0.00
12	50	10	51.29	1.0	0.00	10	44.84	1.0	0.00
12	60	10	50.03	1.0	0.00	10	48.26	1.0	0.00
12	70	10	45.58	1.0	0.00	10	40.31	1.0	0.00
12	80	10	31.17	1.0	0.00	10	28.71	1.0	0.00
12	90	10	22.26	1.0	0.00	10	22.21	1.0	0.00
14	10	10	60.09	1.0	0.00	10	41.03	1.0	0.00
14	20	10	86.00	1.0	0.00	10	89.61	1.0	0.00
14	30	8	*115.85*	*1.0*	*0.00*	10	132.72	1.0	0.00
14	40	10	293.74	1.2	0.01	10	144.03	1.0	0.00
14	50	10	189.24	1.0	0.00	10	180.49	1.0	0.00
14	60	10	144.36	1.0	0.00	10	141.93	1.0	0.00
14	70	10	170.00	1.0	0.00	10	149.68	1.0	0.00
14	80	10	155.25	1.0	0.00	10	145.97	1.0	0.00
14	90	10	80.50	1.0	0.00	10	81.27	1.0	0.00
16	10	10	118.55	1.0	0.00	10	124.57	1.0	0.00
16	20	10	287.01	1.0	0.00	10	309.31	1.0	0.00
16	30	9	*393.79*	*1.0*	*0.00*	10	630.77	1.2	0.02
16	40	7	*551.23*	*1.0*	*0.00*	9	*800.87*	*1.2*	*0.02*
16	50	7	*534.98*	*1.0*	*0.00*	7	*451.09*	*1.0*	*0.00*
16	60	5	*484.64*	*1.0*	*0.00*	9	*403.82*	*1.7*	*0.03*
16	70	7	*515.93*	*1.0*	*0.00*	8	*789.62*	*1.2*	*0.01*
16	80	9	*506.98*	*1.0*	*0.00*	10	568.55	1.0	0.00
16	90	10	319.81	1.0	0.00	10	362.29	1.0	0.00
18	10	8	*553.05*	*1.0*	*0.00*	10	408.04	1.0	0.00
18	20	10	756.97	1.0	0.00	10	778.86	1.0	0.00
20	10	10	1,184.73	1.0	0.00	10	1,543.39	1.0	0.00
20	20	7	*1,787.55*	*1.0*	*0.00*	8	*1,813.87*	*1.0*	*0.00*
22	10	6	*2,704.50*	*1.0*	*0.00*	6	*2,658.64*	*1.0*	*0.00*
22	20	0	—	—	—	1	*3,443.81*	*1.0*	*0.00*
24	10	0	—	—	—	0	—	—	—
24	20	0	—	—	—	0	—	—	—

instances of this size is already very challenging (Rendl et al. 2007).

One might ask whether our polyhedral results presented in §2 have a significant effect on the performance of the ILP and the SDP approach. In other words, would the results be similar if the algorithm used the 3-dicycle inequalities (4) instead of the linearized quadratic equations of Lemma 1? For ILP, the quadratic reformulation significantly improves running times for harder instances, as shown in Table 2. For sparse instances, however, the introduction of additional variables needed for the quadratic reformulation does not pay off.

For SDP, the effect of the quadratic reformulation is less obvious; for small instances, we found that the difference is negligible; the bounds are still strong

enough to prevent branching and to guarantee a small number of outer iterations in the root node. However, for larger instances the quadratic reformulation clearly has a positive effect on the results: as shown in Table 3, significantly more instances can be solved in one hour when using quadratic constraints.

In summary, we can conclude that the reformulation of 3-dicycle inequalities by quadratic face-inducing equations often leads to significant runtime improvements both in the ILP and in the SDP setting, particularly for harder instances.

Acknowledgments

This work was partially supported by the Marie Curie Research Training Network 504438 (ADONET) funded by the European Commission. The first author is supported by the German Science Foundation under contract BU 2313/1-1.

References

Barahona, F., A. R. Mahjoub. 1986. On the cut polytope. *Math. Programming* **36**(2) 157–173.

Billionnet, A., S. Elloumi. 2007. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0–1 problem. *Math. Programming* **109**(1) 55–68.

De Simone, C. 1990. The cut polytope and the Boolean quadric polytope. *Discrete Math.* **79**(1) 71–75.

Eades, P., N. C. Wormald. 1994. Edge crossings in drawing bipartite graphs. *Algorithmica* **11** 379–403.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

Grötschel, M., M. Jünger, G. Reinelt. 1985. Facets of the linear ordering polytope. *Math. Programming* **33**(1) 43–60.

Jünger, M., P. Mutzel. 1997. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.* **1**(1) 1–25.

Knuth, D. 1993. *The Stanford GraphBase: A Platform for Combinatorial Computing*. ACM Press, Addison-Wesley Publishing Company, New York.

Laurent, M. 1997. The max-cut problem. M. Dell'Amico, F. Maffioli, S. Martello, eds. *Annotated Bibliography in Combinatorial Optimization*. John Wiley & Sons, New York, 247–249.

Lewis, M., B. Alidaee, F. Glover, G. Kochenberger. 2009. A note on xQx as a modeling and solution framework for the linear ordering problem. *Internat. J. Oper. Res.* **5**(2) 152–162.

Liers, F., M. Jünger, G. Reinelt, G. Rinaldi. 2004. Computing exact ground states of hard Ising spin glass problems by branch-and-cut. A. K. Hartmann, H. Rieger, eds. *New Optimization Algorithms in Physics*. Wiley-VCH, Weinheim, Germany, 47–69.

Reinelt, G. 1985. *The Linear Ordering Problem: Algorithms and Applications*. Heldermann Verlag, Lemgo, Germany.

Rendl, F., G. Rinaldi, A. Wiegele. 2007. A branch and bound algorithm for max-cut based on combining semidefinite and polyhedral relaxations. *Integer Programming and Combinatorial Optimization—IPCO 2007, Lecture Notes in Computer Science*, Vol. 4513. Springer, Berlin, 295–309.

Zheng, L., C. Buchheim. 2007. A new exact algorithm for the two-sided crossing minimization problem. A. Dress, Y. Xu, B. Zhu, eds. *Combinatorial Optimization and Applications—COCOA 2007, Lecture Notes in Computer Science*, Vol. 4616. Springer, Berlin, 301–310.