Exact Algorithms for the Quadratic Linear Ordering Problem

Christoph Buchheim\(^1\), Angelika Wiegele\(^2\), and Lanbo Zheng\(^3\)

\(^1\) Institut für Informatik, Universität zu Köln, Germany.
buchheim@informatik.uni-koeln.de

\(^2\) Institut für Mathematik, Alpen-Adria-Universität Klagenfurt, Austria.
anangelika.wiegele@uni-klu.ac.at

\(^3\) School of Information Technologies, University of Sydney, Australia.
lzheng@it.usyd.edu.au

Abstract. 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 in, e.g., 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, the resulting constraints are shown to be face-inducing for the polytope corresponding to the unconstrained quadratic problem. We exploit this result both within a branch-and-cut algorithm and within an SDP-based branch-and-bound algorithm. Experimental results for bipartite crossing minimization show that this approach clearly outperforms other methods.

Key words: quadratic optimization, linear ordering, crossing minimization

\(^*\) 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.
1 Introduction

The linear ordering problem is one of the classical NP-hard combinatorial optimization problems [4]. A linear ordering of a given finite set \( S \) is a permutation of its elements. Assuming without loss of generality that \( S = \{1, \ldots, n\} \), 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 \( i, j \in \{1, \ldots, n\} \), 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 \( x_{ij} \) telling 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 [2], 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 in general increases significantly. More surprisingly, it often turns out that the techniques used successfully for the linear variant of the problem are useless for the quadratic variant.

In order to develop cutting plane algorithms for the exact solution of linear ordering problems, polyhedral methods have been investigated intensively; for early references see [5, 10]. In any integer program for linear ordering, 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

\[
0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1
\]

for \( i < j < k \). The main idea presented in this paper is to replace 3-dicycle inequalities by quadratic equations. These 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 optimization problem. This allows us to reduce the quadratic linear ordering problem to the maximum cut problem in an elegant way. Experimentally, this approach turns out to be very efficient in practice.

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

1
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 \( 1 \leq i < j \leq n \), we set

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

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

\[
LO(n) = \text{conv} \{ \chi(\pi) \mid \pi \in S_n \} \subseteq \mathbb{R}^{\binom{n}{2}}.
\]

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

\[
\begin{align*}
\max & \quad x^\top Cx \\
\text{s.t.} & \quad x \in LO(n) \cap \{0,1\}^{\binom{n}{2}},
\end{align*}
\]

(1)

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

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

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

\[
\begin{align*}
\max & \quad \sum_{(i,j,k,l) \in I} (c_{ijkl} + c_{klij})y_{ijkl} + \sum_{i<j} c_{ij}x_{ij} \\
\text{s.t.} & \quad x \in LO(n) \\
& \quad y_{ijkl} \leq x_{ij}, x_{kl} \quad \text{for all } (i,j,k,l) \in I, \\
& \quad y_{ijkl} \geq x_{ij} + x_{kl} - 1 \quad \text{for all } (i,j,k,l) \in I, \\
& \quad y_{ijkl} \in \{0,1\} \quad \text{for all } (i,j,k,l) \in I, \\
& \quad x_{ij} \in \{0,1\} \quad \text{for all } i < j .
\end{align*}
\]

(2)

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

\[
\begin{align*}
\max & \quad \sum_{(i,j,k,l) \in I} (c_{ijkl} + c_{klij})y_{ijkl} + \sum_{i<j} c_{ij}x_{ij} \\
\text{s.t.} & \quad y_{ijkl} \leq x_{ij}, x_{kl} \quad \text{for all } (i,j,k,l) \in I, \\
& \quad y_{ijkl} \geq x_{ij} + x_{kl} - 1 \quad \text{for all } (i,j,k,l) \in I, \\
& \quad y_{ijkl} \in \{0,1\} \quad \text{for all } (i,j,k,l) \in I, \\
& \quad x_{ij} \in \{0,1\} \quad \text{for all } i < j ,
\end{align*}
\]

which is the standard linearization of an unconstrained quadratic optimization problem over the binary variables \( x_{ij} \). Denote the corresponding polytope by \( P_n \). By a well-known result of De Simone [2], 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 QLO($n$) is a face of $P_n$. To this end, we will model the constraint $x \in \text{LO}(n)$ 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

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

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

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

$$x_{ik} - y_{ijkl} - y_{ijkk} + y_{ijjk} = 0$$

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

**Proof.** First note that $(x, y) \in \text{QLO}(n)$ is equivalent to $x \in \text{LO}(n)$, as $(x, y)$ is integer. So assume $x \in \text{LO}(n)$. 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

$$0 = (x_{ik} - x_{jk})(x_{ik} - x_{ij}) = x_{ik} - y_{ijkl} - y_{ijkk} + y_{ijjk}$$

is a valid equation for $x$.

On the other hand, these constraints imply that $x \in \text{LO}(n)$. 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 rank, as every $y$-variable appears in exactly one equation. Now consider any equation $a^\top(x, y) = b$ that is valid for QLO($n$). As QLO($n$) contains the zero vector, we have $b = 0$. To show that $a^\top(x, y) = 0$ is a linear combination of the given constraints, it suffices to show the following:

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

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

(c) $a_{ijkl} = -a_{ijlk} = -a_{ikjl}$ for all $i < j < k$.

For the following, let $\pi_{ji}^{st}$ denote the decreasing permutation $n, \ldots, 1$ 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 $\pi_{ik}^{(k-1)0}$ and $\pi_{ik}^{(k-1)0}$. It is easy to verify that $a^\top(\pi_{ik}^{(k-1)0}) = a^\top(\pi_{ik}^{k-1})$ yields $a_{ik} + \sum_{j=1}^{k-1} a_{ijk} = 0$. Analogously, $a^\top(\pi_{ik}^{(k-1)i}) = a^\top(\pi_{ik}^{0(-k+i)})$ 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 $s, t \geq 0$ with $i + s \neq k, k + t$. For this, we proceed inductively over $s + t$. First assume $i + s < k$. In this case, $a^\top(\pi_{ik}^{(k-1)st})$ translates to

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

(4)
Using (a), we derive \( \sum_{t=1}^{r} a_{i(1+t)}k(k+r) = 0 \). For \( t = 1 \), we get \( a_{i(1+s)}k(k+1) = 0 \).

For \( t > 1 \), we derive \( a_{i(1+s)}k(k+t) = 0 \) by induction. Now assume \( i + s > k \).

If \( i + s < k + t \), then \( a \top \chi(\pi_{ik}^t) = a \top \chi(\pi_{ik}^{t+1}) \) yields

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

Thus \( \sum_{r=1}^{s} a_{i(1+r)}k(k+t) - a_{i(k(k+t))} = 0 \) by (a). By induction, \( a_{i(1+r)}k(k+t) = 0 \) for all \( r < s \) with \( i + r \neq k \), thus \( a_{i(1+s)}k(k+t) = 0 \). If \( i + s > k + t \), we get (4) again, and as above induction over \( t \) yields \( a_{i(1+s)}k(k+t) = 0 \).

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

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

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

\[ \square \]

**Theorem 1.** We have

\[
\dim \text{QLO}(n) = \binom{n}{2} + \binom{n}{3} - \binom{n}{2}.
\]

**Lemma 2.** For all \( i < j < k \), the constraint \( x_{ik} - y_{ijjk} - y_{ikjk} + y_{ijjj} = 0 \) induces a face of the polytope \( P_n \).

**Proof.** It suffices to show that the inequality

\[
x_{ik} - y_{ijjk} - y_{ikjk} + y_{ijij} \geq 0
\]

is valid for \( P_n \) for all \( i < j < k \). As before, we use that

\[
x_{ik} - y_{ijjk} - y_{ikjk} + y_{ijij} = (x_{ik} - x_{ij}) (x_{ik} - x_{ij}).
\]

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 \( (x_{ik} - x_{jk})^2 \).

\[ \square \]

**Theorem 2.** The polytope \( \text{QLO}(n) \) is isomorphic to a face of \( P_n \) and hence to a face of a cut polytope.

### 3 Solution Methods

Our aim is to exploit Theorem 2 for practical computation. In this section, we propose two different approaches: a branch-and-cut algorithm (Section 3.1) and a branch-and-bound algorithm based on semidefinite programming (Section 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 integer linear program (ILP) modeling the cut polytope $P_n$. Whenever we desire to separate a vector $\pi$ from QLO($n$), we ignore the fact that we have additional constraints. Instead, we feed $\pi$ 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. [1, 8, 9]. 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 in [11]. 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 semidefinite program (SDP) is solved approximately in order to obtain an upper bound on the maximum cut.

$$z_{\text{sdp-met}} = \max \{ \langle L, X \rangle \mid X \in \mathcal{E}, A(X) \leq e \}. \quad (5)$$

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 $A(X) \leq e$ symbolically collects the triangle inequalities. $\langle L, X \rangle$ is the trace inner product, i.e., $\langle L, X \rangle = \text{tr}(LX)$.

Since the number of triangle inequalities is of order $|V|^3$, with $V$ being the set of vertices of the graph, solving (5) directly is impractical already for medium-sized graphs. In [11] 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

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

The problem to be solved now reads

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

$$f(\gamma) = \gamma^\top (e - A(X)) = e^\top \gamma + \langle L - A^\top(\gamma), X \rangle.$$
where \( f \) is the convex, but non-smooth function

\[
f(\gamma) := \max_{X \in E} \mathcal{L}(X; \gamma) = e^T \gamma + \max_{X \in E} \langle L - A^T(\gamma), X \rangle.
\]

(6)

The triangle inequalities \( A(X) \leq e \) 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 Section 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 \( \mathcal{B}(X) \leq b \). Adding these constraints to (5) and dualizing them, introducing dual variables \( \mu \), the function to be minimized becomes

\[
f_{\text{qlo}}(\gamma, \mu) = \max_{X \in E} \mathcal{L}(X; \gamma, \mu)
\]

\[
= e^T \gamma + b^T \mu + \max_{X \in E} \langle L - A^T(\gamma) - \mathcal{B}^T(\mu), X \rangle.
\]

Like in [11], the bundle method can be used to minimize \( f_{\text{qlo}} \) subject to \( \gamma \geq 0 \) and \( \mu \geq 0 \). While the triangle constraints are chosen dynamically, the problem-specific constraints \( \mathcal{B}(X) \leq b \) are present all the time, since the number of these constraints is small enough.

For experimenting with applications of the quadratic linear ordering problem, also the parameters of the algorithm in [11] have to be adjusted. This is discussed in Section 4.

4 Computational Experiments

In order 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 shores are placed on two parallel lines and edges are drawn as straight lines; see Fig. 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 [6]. They first consider the problem variant where the permutation on one of the two layers is fixed. Even then, the problem remains NP-hard [3]. 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.
Fig. 1. Two drawings of the same graph, the second drawing is crossing-minimal.

For the problem with two free layers, they 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 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 3214% 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 for two free layers with the two approaches presented in Section 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. Additionally, we 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:

- **JM**: the exact algorithm introduced by Jünger and Mutzel [6]
- **CPLEX**: the CPLEX 10 MIP-solver with default parameters
- **ILP**: the branch-and-cut algorithm described in Section 3.1
- **SDP**: the branch-and-bound approach using SDP described in Section 3.2
For the experiments, we used the original implementation of JM used also in [6], except that we updated it to CPLEX 10. For ILP, we used CPLEX with an additional separation routine for odd cycle inequalities [1]. Except for that, we did not change the standard parameter setting of CPLEX. A similar approach has been presented by two of the authors in [12], however, without exploiting the polyhedral results of Section 2.

In order to use the SDP solver efficiently, we have to adjust the parameters according to our problem. An important parameter is the number of outer iterations of the bundle routine. Due to our problem sizes, we allow 30 outer iterations. The heuristic inside the algorithm in [11] works astonishingly well for maximum 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”, in order to be feasible for our problem. Since more attempts are needed to finally get to the optimal solution (and since 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 created graphs in the same way as in [6]: we generate random bipartite graphs using the function random_bigraph of the Stanford GraphBase [7]. Results are reported for graphs having \( n = 10, 12, \ldots, 18 \) vertices on each layer. For each \( n \), we consider graphs with density \( d = 10, 20, \ldots, 90 \), i.e., with \( \lfloor dn^2/100 \rfloor \) edges. For each pair \((n, d)\), we report the averages over 10 random instances.

All experiments were carried out on an Intel Xeon 5130 processor with 2 GHz; the results are shown in Table 1. Besides \( n \) and \( d \), we state the running times for all four approaches tested, given in cpu-seconds. We limited the average running time per pair \((n, d)\) to one cpu-hour. For the three latter methods, we also state the number of nodes in the enumeration tree.

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 \( 20 \leq d \leq 70 \) as well as for \( n = 12 \) and \( 20 \leq d \leq 30 \). However, the running time increases sharply with the density. For \( n = 14 \), the runtime limit of one hour average is reached at \( d = 50 \), while it is reached already for \( d = 10 \) if \( n \geq 16 \).

For the pure CPLEX approach, the effect of density is even more evident. While it is the fastest method for all \( n \) if \( d = 10 \), it is much slower than all other methods on denser instances. For \( n = 14 \) and \( n = 16 \), the limit is reached at \( d = 30 \), while for \( n = 18 \) it is reached at \( d = 20 \).

The ILP approach is considerably faster than CPLEX on denser instances and is less affected by an increase in density, but it is still not able 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 Section 2 are strong: within the instances being solved, there were only two on which branching was necessary, all other instances could be solved in the root node. In terms of running time, ILP beats both JM and CPLEX on the largest instances.
However, the clear winner of our evaluation is the SDP approach. It could solve all instances with \( n \leq 16 \) and most instances with \( n = 18 \) in time. Particularly for denser instances, the difference between SDP and the other approaches is impressive. E.g., for \( n = 14 \) and all \( 50 \leq d \leq 90 \), the solution was computed after less than two minutes on average, while no other method was able to do this in one hour.

In Table 1, one can observe a decrease of running time for very dense instances both for CPLEX and ILP. The reason is that the percentage of product variables needed is actually not proportional to \( d \). On contrary, the expected percentage of products with non-zero 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 \( d = 100\% \).

One might ask whether our polyhedral results presented in Section 2 have a significant effect on the performance of the SDP approach. In other words, would the results be similar if the SDP algorithm used the 3-dicycle inequalities (3) instead of the quadratic equations of Lemma 1? 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, starting from \( n = 14 \) the situation changes dramatically. Now branching becomes necessary for many instances if 3-dicycle inequalities are used. In fact, for an instance with \( n = 14 \) and \( d = 40 \) we had to stop the computation after one day of cpu-time, in which 95 branch-and-bound nodes had been created.

In summary, we can conclude that the reformulation of 3-dicycle inequalities by quadratic face-inducing equations leads to significant runtime improvements both in the ILP and in the SDP setting. We are convinced that this remains true for general instances of the quadratic linear ordering problem.
Table 1. Results for bipartite graphs with increasing size and density (limit 1h)

<table>
<thead>
<tr>
<th>n</th>
<th>d</th>
<th>JM time</th>
<th>CPLEX time</th>
<th>JM nodes</th>
<th>CPLEX nodes</th>
<th>ILP time</th>
<th>ILP nodes</th>
<th>SDP time</th>
<th>SDP nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>0.02</td>
<td>0.01</td>
<td>1.0</td>
<td>0.28</td>
<td>1.0</td>
<td>1.57</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>0.07</td>
<td>0.42</td>
<td>7.8</td>
<td>1.64</td>
<td>1.0</td>
<td>1.72</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>0.20</td>
<td>4.16</td>
<td>182.4</td>
<td>7.81</td>
<td>1.0</td>
<td>3.46</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>40</td>
<td>0.44</td>
<td>49.69</td>
<td>2067.5</td>
<td>19.03</td>
<td>1.0</td>
<td>3.65</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>0.80</td>
<td>258.01</td>
<td>6797.4</td>
<td>28.41</td>
<td>1.0</td>
<td>3.38</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>1.50</td>
<td>1162.67</td>
<td>21710.9</td>
<td>44.01</td>
<td>1.0</td>
<td>4.72</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>2.05</td>
<td>1162.67</td>
<td>21710.9</td>
<td>44.01</td>
<td>1.0</td>
<td>4.72</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>5.29</td>
<td>962.50</td>
<td>8643.2</td>
<td>24.26</td>
<td>1.0</td>
<td>3.61</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>11.59</td>
<td>228.82</td>
<td>1065.4</td>
<td>15.34</td>
<td>1.0</td>
<td>3.38</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>0.26</td>
<td>0.03</td>
<td>0.1</td>
<td>8.65</td>
<td>1.0</td>
<td>12.81</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>1.94</td>
<td>3.07</td>
<td>74.5</td>
<td>28.53</td>
<td>1.0</td>
<td>13.80</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>5.92</td>
<td>70.31</td>
<td>2333.5</td>
<td>60.36</td>
<td>1.0</td>
<td>15.16</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>40</td>
<td>21.27</td>
<td>3282.85</td>
<td>54971.3</td>
<td>172.25</td>
<td>1.0</td>
<td>17.49</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>50</td>
<td>74.36</td>
<td>—</td>
<td>—</td>
<td>723.55</td>
<td>1.0</td>
<td>31.20</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>60</td>
<td>134.40</td>
<td>—</td>
<td>—</td>
<td>962.23</td>
<td>1.0</td>
<td>32.17</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>70</td>
<td>281.69</td>
<td>—</td>
<td>—</td>
<td>1067.25</td>
<td>1.0</td>
<td>26.89</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>80</td>
<td>682.36</td>
<td>—</td>
<td>—</td>
<td>499.45</td>
<td>1.0</td>
<td>19.95</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>90</td>
<td>1373.39</td>
<td>—</td>
<td>—</td>
<td>145.50</td>
<td>1.0</td>
<td>31.20</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>19.87</td>
<td>0.26</td>
<td>3.3</td>
<td>8.65</td>
<td>1.0</td>
<td>51.82</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>142.11</td>
<td>40.09</td>
<td>1152.4</td>
<td>272.99</td>
<td>1.0</td>
<td>61.85</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>30</td>
<td>963.12</td>
<td>—</td>
<td>—</td>
<td>1198.01</td>
<td>1.0</td>
<td>93.19</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>40</td>
<td>2299.98</td>
<td>—</td>
<td>—</td>
<td>3121.92</td>
<td>1.0</td>
<td>97.93</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>50</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>117.55</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>60</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>95.85</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>70</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>101.83</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>80</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>101.85</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>90</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>56.30</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>19.87</td>
<td>2.11</td>
<td>26.2</td>
<td>316.98</td>
<td>1.0</td>
<td>119.14</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>2701.59</td>
<td>40293.8</td>
<td>1905.32</td>
<td>1905.32</td>
<td>1.0</td>
<td>200.87</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>30</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>432.85</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>40</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>1432.02</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>1181.20</td>
<td>2.4</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>60</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>1186.81</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>70</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>916.86</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>80</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>444.92</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>90</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>224.13</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td>—</td>
<td>5.54</td>
<td>51.8</td>
<td>978.23</td>
<td>1.0</td>
<td>334.23</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>491.87</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>30</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>1243.73</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>40</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>57.23</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>50</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>491.87</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>60</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>1243.73</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>70</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>2624.98</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>80</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>2523.04</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>90</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1.0</td>
<td>601.30</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>
References