A Basic Toolbox for Constrained Quadratic 0/1 Optimization*

Christoph Buchheim[1], Frauke Liers[1], and Marcus Oswald[2]

[1] Universität zu Köln, Institut für Informatik, Pohligstraße 1, 50969 Köln, Germany
[2] Universität Heidelberg, Institut für Informatik, INF 368, 69120 Heidelberg, Germany

Abstract. In many practical applications, the task is to optimize a non-linear function over a well-studied polytope P as, e.g., the matching polytope or the travelling salesman polytope (TSP). In this paper, we focus on quadratic objective functions. Prominent examples are the quadratic assignment and the quadratic knapsack problem; further applications occur in various areas such as production planning or automatic graph drawing. In order to apply branch-and-cut methods for the exact solution of such problems, they have to be linearized. However, the standard linearization usually leads to very weak relaxations. On the other hand, problem-specific polyhedral studies are often time-consuming. Our goal is the design of general separation routines that can replace detailed polyhedral studies of the resulting polytope and that can be used as a black box. As unconstrained binary quadratic optimization is equivalent to the maximum cut problem, knowledge about cut polytopes can be used in our setting. Other separation routines are inspired by the local cuts that have been developed by Applegate, Bixby, Chvátal and Cook for faster solution of large-scale traveling salesman instances. By extensive experiments, we show that both methods can drastically accelerate the solution of constrained quadratic 0/1 problems.

Keywords: quadratic programming, maximum cut problem, local cuts, crossing minimization, similar subgraphs.

1 Introduction

Optimizing a linear objective function over binary variables under additional linear constraints is NP-hard in general. One of the most successful frameworks for solving such problems is branch-and-cut. In order to develop fast branch-and-cut algorithms, it is crucial to determine good outer descriptions of the polytope P consisting of the convex hull of all feasible solutions of the problem at hand. The branch-and-cut approach is well developed, and the facial description of many polytopes corresponding to classical combinatorial optimization problems is well understood. For several problems practically efficient implementations exist.

* Financial support from the German Science Foundation is acknowledged under contracts Bu 2313/1-1 and Li 1675/1-1. Partially supported by the Marie Curie RTN Adonet 504438 funded by the EU.

C.C. McGeoch (Ed.): WEA 2008, LNCS 5038, pp. 249–262, 2008.

Instead of a linear objective function, we often desire to optimize a non-linear objective function over P. We consider problems where the non-linearities are locally defined, i.e., where every non-linear term in the objective function depends on few variables. In this paper, we focus on binary quadratic functions, however some of the proposed methods can easily be adapted to general non-linear functions.

The easiest example of a binary quadratic optimization problem is the maximum cut problem, which is equivalent to optimizing a degree-two polynomial over the hyper cube [4]. Many practical applications lead to non-linear objective functions in a natural way. Several crossing minimization problems in automatic graph drawing can be modeled as quadratic optimization problems over linear ordering type polytopes. To give another example, the tool switching problem arising in production planning can be solved by minimizing a polynomial of degree three over a polytope that is closely related to the TSP.

In any integer programming based approach to such non-linear 0/1 problems, the first step is to linearize the problem by introducing artificial variables that model the non-linearities. We thus need to optimize the linearized objective function over a polytope Q defined in a higher-dimensional space instead of optimizing a non-linear objective function over the original polytope P.

It is easy to see that all facets of P yield valid inequalities for Q. A naive branch-and-cut approach for the optimization over Q would use the separation routines known for P, in combination with the constraints modeling the connection between original and new variables, and resort to branching if no violated inequality can be detected any more. According to our experience, the performance of such an approach is very weak. Often, facet-inducing inequalities for P do not induce facets of Q, and the variables modeling non-linear terms change the polyhedral structure significantly. This can even happen if only one product is introduced and linearized.

In view of this, one could decide to undertake a polyhedral investigation of Q and try to develop specialized separation routines. Doing this will – very probably – be time consuming. Instead, much (human and computer) time could be saved by having some effective black-box routines at hand that speed up the solution algorithms but need only very limited knowledge about the problem structure. For quadratic problems, we provide such black-box routines and show that they drastically improve the running time of the solution algorithms.

Assuming that P is well understood, we ask the following question: How can we exploit the knowledge of P for optimizing over Q, without detailed polyhedral studies of Q? Even if the user is willing to invest some specific knowledge of Q, he/she can still combine her own separation strategies with our general methods outlined below. Moreover, the constraints produced by our methods might give some insight into the polyhedral structure of Q and point at important classes of cutting planes, which could be separated right from the start, using tailored separation algorithms.

We address the general separation problem from two complementary directions. First assume that the objective function is quadratic. In case the problem

is unconstrained, one can formulate it as a maximum cut problem on an associated graph [4]. Even in the presence of constraints, valid inequalities for the cut polytope remain valid for Q after transformation, and can be separated using the same transformation. In several applications, the transformed constraints of P induce a face of the corresponding cut polytope, which gives some theoretical evidence that the inequalities derived from the cut polytope can be helpful.

On the other hand, we want to exploit the knowledge of the structure of the feasible solutions in P. Our proposed separation routine is inspired by the local cuts that have been developed by Applegate, Bixby, Chvátal and Cook (ABC^2) [1]. With the help of local cuts, they could solve big TSP instances being unsolved before. Recently, we proposed a variant of the local cut generation procedure that has some advantageous features [3]. We call our cutting planes *target cuts*. The main difference to the local cuts lies in a modified LP formulation that makes it possible to avoid the time-consuming tilting steps, as always a facet of the projected polytope is determined that can immediately be lifted to a valid inequality for Q.

For non-linear problems, the local or target cut approach is well-suited, as every non-linear term is determined by the original variables, so that the number of vertices does not change from P to Q. In particular, going from linear to non-linear objective functions does not slow down the cut generation significantly. Another advantage of this approach is that the separation can be implemented as a general framework that applies to all problems in this class. The user only needs to input some information about the structure of the feasible solutions, which is much easier than understanding the structure of the corresponding polytope. This approach can be applied to arbitrary non-linear problems in which the non-linearities are locally defined.

Our main contribution is to show that these approaches are very easy to use and lead to much better performance of general branch-and-cut approaches. By extensive computational experiments we show that not only the number of nodes in the enumeration tree but also the running time decreases dramatically, when compared to an algorithm that only uses the standard separation routines for the well-studied polytope P.

For some classical quadratic 0/1 problems, such as the quadratic knapsack problem or the quadratic assignment problem, special-purpose algorithms and implementations exist that exploit the problem structure and lead to effective algorithms. Clearly, we cannot compete with such problem-specific approaches. In this work, we aim at designing general-purpose methods that help improve the solution algorithms for quadratic problems for which not much is known about their structure. In particular, the reference point for our evaluation is the basic approach using standard linearization and separation for P.

The outline of this paper is as follows. We fix notation in Section 2. In Section 3, we discuss cutting planes derived from the cut polytope. In Section 4, we introduce target cuts and their usage in the context of quadratic problems. In Section 5, we explain the studied applications: the quadratic matching

problem in Section 5.1 and the quadratic linear ordering and the linear arrangement problem in Section 5.2 and 5.3. In Section 6 we present experimental results.

2 Definitions

Consider a combinatorial optimization problem on a finite set E with feasible solutions $\mathcal{I} \subseteq 2^E$ and with a linear objective function $c(I) = \sum_{e \in I} c_e$, where $c_e \in \mathbb{R}$ for all $e \in E$. Without loss of generality, we desire to minimize $c(I)$ over all $I \in \mathcal{I}$. Let the polytope $P \subseteq \mathbb{R}^E$ denote the convex hull of all incidence vectors of feasible solutions. The corresponding integer linear program reads

$$\begin{array}{cll} & \min & \sum_{e \in E} c_e x_e \\ \text{(P)} & \text{s.t.} & x \in P \\ & & x \in \{0;1\}^E \end{array}$$

In the following, we focus on objective functions that are quadratic in the variables x, i.e., we consider problems of the form

$$\begin{array}{cll} & \min & \sum_{e \in E} c_e x_e + \sum_{e,f \in E; e \neq f} c_{ef} x_e x_f \\ \text{(QP)} & \text{s.t.} & x \in P \\ & & x \in \{0;1\}^E, \end{array}$$

For problems defined on a graph $G = (V, E)$ with variables corresponding to edges, and for two edges $e = (i, j)$ and $f = (k, l)$, we will use the notations c_{ef}, $c_{(i,j)(k,l)}$, and c_{ijkl} interchangeably. In order to address (QP) by integer programming techniques, we apply the standard linearization: for each pair $\{e, f\}$ with $c_{ef} \neq 0$, we introduce a binary variable y_{ef} modeling $x_e x_f$, along with the constraints $y_{ef} \leq x_e$, $y_{ef} \leq x_f$, and $y_{ef} \geq x_e + x_f - 1$. The linearized problem then reads

$$\begin{array}{cll} & \min & \sum_{e \in E} c_e x_e + \sum_{e,f \in E; e \neq f} c_{ef} y_{ef} \\ \text{(LQP)} & \text{s.t.} & x \in P \\ & & y_{ef} \leq x_e, x_f \qquad \text{for all } \{e,f\} \text{ with } c_{ef} \neq 0 \\ & & y_{ef} \geq x_e + x_f - 1 \quad \text{for all } \{e,f\} \text{ with } c_{ef} \neq 0 \\ & & y_{ef} \in \{0;1\} \qquad \text{for all } \{e,f\} \text{ with } c_{ef} \neq 0 \\ & & x \in \{0;1\}^E \,. \end{array}$$

We are interested in the polytope Q spanned by all feasible solutions of (LQP).

Note that other methods for linearizing (QP) have been proposed in the literature. Nevertheless, we focus on the standard linearization, as it is the most natural and popular way to linearize (QP) and as it can easily be implemented.

3 Cutting Planes from Maxcut

Consider a graph $G = (V, E)$ with edge weights w_e. For $W \subseteq V$, the cut $\delta(W)$ is defined as

$$\delta(W) = \{(u, v) \in E \mid u \in W, v \notin W\} \,.$$

Its weight is $\sum_{e \in \delta(W)} w_e$. The maximum cut problem asks for a cut of maximum weight and is NP-hard for general graphs. The corresponding cut polytope, i.e., the convex hull of incidence vectors of cuts, is well studied [2,5], and practically effective branch-and-cut implementations exist for its solution [6,7].

It is a well-known result that the problem of optimizing a binary quadratic function without further constraints is equivalent to determining a maximum cut in an auxiliary graph $G_{\text{lin}} = (V_{\text{lin}}, E_{\text{lin}})$ [4]. The latter contains a node for each variable x_e. For each quadratic term $x_e x_f$ occuring in the objective function with $c_{ef} \neq 0$, the edge set E_{lin} contains an edge between the nodes corresponding to x_e and x_f. Furthermore, an additional root node and edges from this node to all nodes in V_{lin} are introduced. Now there exists a simple linear transformation between the edge variables in the maximum cut setting and the linear variables or products in the unconstrained quadratic optimization setting under which P is isomorphic to the cut polytope of G_{lin} [4].

If P is the unit hypercube, solving (LQP) thus amounts to determining a maximum cut in G_{lin}, i.e., to optimizing over a cut polytope defined in the E_{lin}-dimensional space. If P is a strict subset of the unit hypercube, i.e., if additional constraints are present, these constraints can be transformed as well and we derive that P is isomorphic to a cut polytope with further linear constraints. In particular, all inequalities valid for the cut polytope still yield inequalities valid for (LQP) and can be used in a cutting plane approach.

Clearly, intersecting the cut polytope with arbitrary hyperplanes in general yields a non-integer polytope. The structure of the resulting polytope can be very different from a cut polytope. In this case it is not clear whether the knowledge about the cut polytope can help solving the constrained optimization problem. However, several relevant applications exist in which the intersection of the cut polytope with a set of hyperplanes cuts out a face of a cut polytope, at least if certain product variables are present, e.g., for quadratic assignment and quadratic matching. The proof for the quadratic matching is a slight modification of the proof for the quadratic assignment polytope.

In any case, we obtain a correct separation algorithm for (LQP) based on cut separation. Within a branch-and-cut framework, we can always work in the original model and apply other separation algorithms as desired. When it comes to the cut separation, we build the graph $G_{\text{lin}} = (V_{\text{lin}}, E_{\text{lin}})$, transform the fractional point, and separate the inequalities known for the cut polytope. Found cutting planes are transformed back to yield cutting planes for (LQP).

4 Target Cuts for Quadratic 0/1 Problems

Usually, separation routines aim at generating faces or facets of some polytope in question that share similar structure. They are said to follow the *template paradigm*. Recently, ABC^2 proposed some general separation routine yielding so-called *local cuts* that are inequalities outside the template paradigm for which the structure is not known [1]. The size of the problem is first reduced by projecting

the incidence vectors of feasible solutions onto a small-dimensional space (ABC[2] do this by shrinking nodes into supernodes).

For $r \leq m$, let π denote a projection $\mathbb{R}^m \to \mathbb{R}^r$ and let $\overline{Q} = \pi(Q) \subseteq \mathbb{R}^r$ denote the convex hull of the projected feasible solutions. Let $x^* \in \mathbb{R}^m$ be the point to be separated and $\overline{x}^* = \pi(x^*)$ be its projection to $\mathbb{R}^r$. A face-inducing inequality that separates some projected fractional point from $\overline{Q}$ can be obtained by solving an appropriately chosen linear program. Its size is basically determined by the number of its vertices. Thus, if the dimension of $\overline{Q}$ is not too big, this is fast in practice. Furthermore, the size of the linear program can be reduced by several considerations, and by delayed column generation only necessary feasible solutions are enumerated. A found local cut is then sequentially lifted and tilted until it becomes a facet for $\overline{Q}$ and then lifted to become feasible for the original TSP polytope.

Recently, we proposed a variant of the local cuts that we call *target cuts* [3]. The local cut framework can easily be adapted to target cuts, however the time-consuming tilting steps can be omitted. The reason for this is that we propose a different cut-generating linear program that generates a facet of $\overline{Q}$ right away. Furthermore, the volume of the generated facet is expected to be big. In the following, we briefly explain the target cuts separation. Details can be found in [3]. Subsequently, we will show that their use is favorable in the context of quadratic problems.

Assuming for now that $\overline{Q}$ is full-dimensional, we choose a point $\overline{q}$ in the interior of $\overline{Q}$. In case the projected non-feasible point $\overline{x}^*$ is not contained in $\overline{Q}$, we want to return a cutting plane that separates $\overline{x}^*$ from $\overline{Q}$. We argue in [3] that a facet from $\overline{Q}$ can be obtained by solving the following linear program:

$$\begin{aligned} \max \quad & a^\top(\overline{x}^* - \overline{q}) \\ \text{s.t.} \quad & a^\top(\overline{x}_i - \overline{q}) \leq 1 \quad \text{for all } i = 1, \dots, s \\ & a \in \mathbb{R}^r \end{aligned} \tag{1}$$

Here, $x_1, \dots, x_s$ are the vertices of $\overline{Q}$. A facet for $\overline{Q}$ violated by $\overline{x}^*$ can be read off the optimum solution of (1) as follows. If the optimum value of (1) is greater than 1, the corresponding inequality $a^\top(x - \overline{q}) \leq 1$ is violated by $\overline{x}^*$. Otherwise, $\overline{x}^*$ is contained in $\overline{Q}$.

In case the dimension of $\overline{Q}$ is smaller than r, the linear program (1) can be unbounded. In this case, $a^\top(x - \overline{q}) = 0$ is a valid equation for $\overline{Q}$ violated by $\overline{x}^*$, if a is an unbounded ray in (1).

In order to reduce the size of LP (1), we adapted the delayed column generation procedure proposed for local cuts to the target cut case. The procedure requires an oracle for maximizing any linear function over $\overline{Q}$. Having this at hand, one starts with a small, possibly empty, set of vertices $\overline{x}_1, \dots, \overline{x}_h$. Then a target cut $a^\top(x - \overline{q}) \leq 1$ is produced for the polytope $\overline{Q}_h = \operatorname{conv}\{\overline{x}_1, \dots, \overline{x}_h\}$, by solving the corresponding linear program. Then, the oracle is called to maximize the left-hand side of the inequality. In case the maximum is bigger than 1, we add the maximal solution as a new $\overline{x}_{h+1}$ to (1). Otherwise we stop the procedure, having found a valid target cut. This process is iterated until the generated

inequality is found to be valid. The number of columns added in this procedure is usually much smaller than the number of vertices of $\overline{Q}$.

In order to use target cuts for quadratic problems, we need to specify which projection to choose. In general, there is no easy answer to this question, and the user might have to test the performance of different projections in order to find which one gives best results. The projection needs to allow fast recognition or determination of the points in $\mathbb{R}^r$ that can be extended to feasible solutions in the original space. For several applications this is possible with the trivial, i.e., orthogonal, projection onto some sub-graph or sub-space, or the projection through shrinking of nodes into supernodes. For a given (linear) projection, lifting of a found inequality is trivial.

For some problems, certain projections seem to be favorable to others. For example, in a problem in which the global structure is important, as is the case for the TSP, a projection through shrinking should be prefered in case it allows to characterize the points in $\mathbb{R}^r$ having a preimage in $\mathbb{R}^m$ under π. On the other hand, there are problems in which the local structure seems to be characteristic of the problem, as, e.g., for the matching problem. In the latter, trivial projections can be used.

The usage of target cuts allows the implementation of a general framework in which only the projection and the oracle need to be specified for the particular application; everything else is problem-independent. Moreover, target cuts are well-suited for quadratic 0/1 problems: the size of the cut generating program (1) remains moderate, as there is a 1-1 correspondence between the vertices of the polytope P and the polytope Q. Therefore, the projected linearized polytope $\overline{Q}$ has the same number of vertices as $\pi(P)$, so that the number of rows of (1) does not grow with the introduction of product variables. In other words, the additional product variables do not affect the performance considerably, which allows to deal with non-trivial chunk sizes.

5 Applications

Applications of constraint quadratic binary optimization problems abound. One of the more traditional examples is the quadratic assignment problem; more recently also the quadratic knapsack problem has attracted some interest. In the following, we consider two other problems: the quadratic matching and the quadratic linear ordering problem. More precisely, we consider applications that are naturally modeled as such problems. In Section 5.1, we look at the problem of finding highly similar subgraphs, which can be modeled as a quadratic (bipartite) matching problem. In Section 5.2 and 5.3, we discuss two applications of quadratic linear ordering: the bipartite crossing minimization problem and the linear arrangement problem.

5.1 Finding Highly Similar Subgraphs – Quadratic Matching

Assume we are given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, and we want to get insight into how similar the two graphs are. This problem occurs in several

practical applications, e.g., in automatic graph drawing and computational biology. The task is to determine a matching of a subset or all nodes of G_1 to those of G_2 such that as many edges as possible in the two graphs are mapped onto each other. Obviously, this problem is a generalization of the graph isomorphism problem in which we decide whether there exists a matching of the nodes in V_1 to those in V_2 such that all edges in E_1 are mapped onto edges in E_2, and vice versa.

In the generalization we are concerned with, we also allow but penalize the case in which $u_1 \in V_1$ is matched on $u_2 \in V_2$ and $v_1 \in V_1$ on $v_2 \in V_2$, but exactly one of the edges (u_1, v_1) or (u_2, v_2) exists. A straight-forward model for this problem is the following quadratic matching formulation

$$\begin{array}{lll} & \max & \sum_{i\in V_1, j\in V_2} x_{ij} + \sum_{i,k\in V_1, j,l\in V_2} c_{ijkl} x_{ij} x_{kl} \\ \text{(QMP)} & \text{s.t.} & \sum_{i\in V_1} x_{ij} \leq 1 \qquad \forall j \in V_2 \\ & & \sum_{j\in V_2} x_{ij} \leq 1 \qquad \forall i \in V_1 \\ & & x_{ij} \in \{0;1\} \; \forall i \in V_1, j \in V_2 \end{array}$$

with costs $c_{ijkl} < 0$ if either $(i,k) \in E_1$ or $(j,l) \in E_2$, but not both. Otherwise $c_{ijkl} \geq 0$. In this model, $x_{ij} = 1$ means that node $i \in V_1$ is matched with node $j \in V_2$.

5.2 Bipartite Crossing Minimization – Quadratic Linear Ordering I

Consider a bipartite graph $G = (V_1 \cup V_2, E)$. We want to draw G in the plane so that the nodes of V_1 and V_2 are placed on two parallel horizontal lines. The task is to minimize the number of crossings between edges, assuming that all edges are drawn as straight lines. Several applications exist in the area of automatic graph drawing. Clearly, the number of crossings only depends on the orders of vertices on the two lines.

First, we assume that the nodes V_1 on the upper level are layouted in some fixed order, whereas the nodes on the lower level are allowed to permute within the layer. The permutation of the nodes in V_2 has to be chosen such that the number of edge crossings is minimal. Let $i, j \in V_1$, $k, l \in V_2$ and edges $(i,k), (j,l)$ be present. Assume i is before j in the fixed order. No crossing exists in case k is before l on the second level, otherwise there is a crossing.

Hence the bipartite crossing minimization problem with one fixed layer can easily be formulated as a linear ordering problem. Now let us formulate the problem with two free layers as a quadratic optimization problem over the linear ordering polytope. For i, j, k, l chosen as above, there is no crossing in case i is before j and k is before l, or j is before i and l is before k. Let us introduce variables x_{uv} that take value 1 if u is drawn before v, and 0 otherwise. Then we have to solve the problem

$$\begin{array}{lll} & \max & \sum_{(i,k),(j,l)\in E} x_{ij} x_{kl} \\ \text{(QLO}_1\text{)} & \text{s.t.} & x \in P_{LO} \\ & & x_{ij} \in \{0;1\} \; \forall i, j \in V_1 \text{ or } i, j \in V_2 \,, \end{array}$$

where P_{LO} is the linear ordering polytope.

5.3 Linear Arrangement – Quadratic Linear Ordering II

The linear arrangement problems is given as follows. We are looking for a permutation of n objects in such a way that a linear function c on the differences of positions of the objects is minimized. More precisely, we desire to determine a permutation π of $\{1, \ldots, n\}$ minimizing

$$\sum_{1 \le i,j \le n} c_{ij} |\pi(i) - \pi(j)| \; .$$

To this end we use the fact that the distances of the positions of two elements i and j with respect to a permutation π can be expressed in terms of betweenness variables. This distance equals 1 plus the number of elements lying between i and j, i.e., $|\pi(i) - \pi(j)| = 1 + \sum_k x_{ik} x_{kj}$ where x_{ij} is the usual linear odering variable modeling whether $\pi(i) < \pi(j)$ or not. Therefore, up to a constant, the linear arrangement problem can be rewritten as

$$\begin{array}{lll} & \max & \sum_{i \neq j \neq k \neq i} c_{ij} x_{ik} x_{kj} \\ (\mathrm{QLO}_2) & \text{s.t.} & x \in P_{LO} \\ & & x_{ij} \in \{0;1\} \; \forall i,j \in \{1 \ldots n\}, i \neq j. \end{array}$$

Note that for this application only products of linear ordering variables are required that are of the type $x_{ik} x_{kj}$, which are only $O(n^3)$ many.

6 Experiments

We implemented the two separation approaches discussed in Section 3 and 4 within the branch-and-cut framework ABACUS, using CPLEX 11. All test runs were performed on Xeon machines with 2.66 GHz.

For each application we addressed, we start a branch-and-cut algorithm with the linear programming relaxation of the linearized problem (LQP). Separation routines for the polytopes P are assumed to be readily available. We compare the performance of this basic approach with the same approach extended by appropriately used maximum cut separation as described in Section 3 and the target cut separation as introduced in Section 4. For the tested applications, we used trivial projections onto subsets of variables, called *chunks*.

The chunks were chosen randomly in the sense that we first generate a subgraph randomly and then project onto all those linear and product variables that are completely determined by the subgraph. For the maximum cut separation, we separate the cycle inequalities [2]. We aimed at developing one relatively abstract implementation that can easily be used for all quadratic problems of type (QP) without having to incorporate many changes. Only the target-cut oracle and the test whether some vector represents a feasible solution are specific to the problem and have to be implemented separately for each application. We tested our approaches on randomly generated instances.

6.1 The Quadratic Matching Problem

For the quadratic matching problem, we studied instances defined on complete graphs. Note that a product $x_{ij}x_{kl}$ is necessarily zero if i, j, k, l are not pairwise distinct. We create random instances where for given pairwise distinct i, j, k, l the weight c_{ijkl} is non-zero with a given probability p. In this case, the weight is randomly chosen from $\{-1000, \dots, 1000\}$. All linear weights c_{ij} are also chosen randomly from $\{-1000, \dots, 1000\}$. An instance is thus defined by the number of nodes n, the percentage p of products with non-zero coefficient, and a random seed r for the weights.

Our implementations either determine a maximum quadratic matching or a minimum perfect quadratic matching. In the basic branch-and-bound approach, we separate the blossom inequalities that are known to be the only non-trivial facets of the matching polytope. We compare this basic approach with a branch-and-cut algorithm that uses separation of cutting planes derived from the cut polytope and of target cuts on varying chunk sizes, as explained in Section 3 and 4. We also test an implementation with both separation routines.

It turns out that better performance can be achieved if the maximum-cut separation procedure is only called in the root node of the branch-and-bound tree, and not after branching has been done. For the target cuts, the extendable solutions under a trivial projection are the incidence vectors of (not necessarily perfect) quadratic matchings. For their generation, two oracles are implemented: First, a heuristic greedy oracle tries to identify fast necessary incidence vectors of quadratic matchings. In case it is successful, the delayed column generation procedure continues. In case it is not successful, we test whether a violating vector exists by calling an exact oracle. In the latter, the integer programming formulation for the quadratic matching problem on the small chunk is solved exactly. The column generation procedure is iterated until no more violating vector is found by the exact oracle.

We show some running times in Table 1. We report the cpu time in seconds and the number of subproblems needed to solve the instance to optimality. IP refers to the basic algorithm, MxTy means that we apply cut generation if $x = 1$, and target cut separation with chunk size y.

As can be expected, in practice the number of found blossom inequalities is very small, often none of them is violated, and so the basic implementation solves the problem basically via branching. Only very small instances can be solved to optimality. It is obvious that the separation of inequalities from the cut polytope considerably improves the running times. Also the target cut separation strongly reduces the number of subproblems, the number of linear problems and the running time. The best improvement is achieved when both separation routines are included.

The optimal size of the chunks depends on the size of the instances. Clearly, using too large chunks can increase the total runtime, since the effect of having to solve less subproblems is foiled by the long running time needed to compute the target cuts; the latter increases exponentially in the size of the chunk. For the larger instances we considered, the best results where obtained with chunks of 5 to 7 nodes.

Table 1. Results for the quadratic matching problem, perfect matchings (top) and general matchings (bottom)

$\|V\|$	p	IP		MC0 T5		MC0 T6		MC0 T7		MC1 T0		MC1 T5		MC1 T6		MC1 T7	
14	0.4	1:06	505	0:48	267	0:47	49	7:42	13	1:23	297	0:55	91	2:01	33	9:23	11
16	0.4	29:25	5145	13:42	2021	6:29	377	23:30	69	21:21	2187	9:06	703	7:12	221	21:57	65
18	0.1	0:27	331	0:27	281	0:27	195	0:17	73	0:22	159	0:26	123	0:22	83	0:20	59
20	0.1	2:47	1021	2:30	807	1:49	525	1:04	203	1:42	425	1:42	211	1:04	153	1:01	81
16	0.2	0:22	255	0:16	119	0:23	69	0:20	1	0:11	33	0:46	5	0:22	39	0:51	1
16	0.2	1:33	877	1:15	591	0:48	257	0:47	99	1:50	65	1:16	363	0:57	203	2:36	47
16	0.2	0:41	473	0:37	379	0:19	71	0:28	43	1:25	23	0:44	391	0:35	83	1:38	21
16	0.2	1:08	751	1:06	667	0:37	231	0:42	87	1:33	45	0:54	335	0:45	197	2:18	39
18	0.1	0:21	267	0:27	281	0:27	195	0:22	83	0:26	79	0:22	159	0:25	123	0:45	49
18	0.1	0:20	261	0:19	225	0:15	125	0:18	103	0:22	91	0:16	149	0:20	147	0:33	65
18	0.1	0:17	211	0:18	185	0:13	107	0:14	45	0:17	51	0:12	69	0:16	63	0:25	35
18	0.1	0:20	225	0:19	197	0:18	155	0:14	45	0:31	111	0:20	169	0:23	131	0:33	73
18	0.2	16:16	3501	12:42	2463	4:06	527	3:53	327	5:11	259	10:29	1583	12:42	2623	7:08	205
18	0.2	12:39	3043	10:28	2387	2:09	245	2:27	175	3:35	205	7:58	1337	4:24	373	4:24	123
18	0.2	10:10	2359	8:29	1749	3:33	513	3:03	147	5:13	277	8:41	1251	5:01	415	5:19	123
18	0.2	20:13	4497	17:23	3343	6:56	1193	5:15	485	8:14	583	16:28	2663	8:57	1257	8:18	339
20	0.1	2:38	1063	2:29	807	1:48	525	1:04	153	1:19	199	1:42	425	1:41	211	1:16	71
20	0.1	3:49	1129	4:05	1117	2:21	615	1:32	141	2:16	507	2:44	543	2:45	521	1:44	171
20	0.1	3:54	1259	4:04	1089	2:47	767	1:40	235	2:11	453	2:55	679	2:37	541	1:40	215
20	0.1	2:40	915	2:39	807	1:54	419	1:28	121	1:28	255	2:09	477	2:02	269	1:35	81

Table 2. Results for the quadratic linear ordering problem, random instances (top) and linear arrangement instances (bottom). Instances marked with −− could not be solved within 2 GB of RAM.

$\|V\|$	$\|E\|$	IP		MC0 T5		MC0 T6		MC1 T0		MC1 T5		MC1 T6	
10	45	0:13:45	10995	0:00:43	1	0:13:55	1	0:23:18	341	0:00:34	1	0:10:40	1
10	45	0:15:50	12349	0:00:58	1	0:21:15	1	0:16:06	301	0:00:39	21	0:12:55	1
10	45	0:14:39	13319	0:00:41	1	0:14:01	1	0:23:52	527	0:00:29	1	0:09:14	1
10	45	0:11:18	8871	0:00:55	1	0:19:27	1	0:31:47	491	0:00:40	1	0:13:34	1
11	55	>3:00:00	−−	0:02:31	1	0:37:15	1	2:50:34	965	0:01:51	1	0:21:59	1
11	55	>0:30:00	−−	0:02:55	1	0:44:25	1	6:06:02	1955	0:02:05	1	0:24:57	1
11	55	>0:32:00	−−	0:02:00	1	0:25:27	1	3:15:18	1003	0:01:34	1	0:18:25	1
11	55	>0:29:00	−−	0:02:15	1	0:30:15	1	2:23:34	867	0:01:37	1	0:20:51	1
14	29	0:00:16	1049	0:00:07	189	0:00:23	25	0:00:02	15	0:00:02	5	0:00:10	1
14	38	>0:06:23	−−	0:02:03	2347	0:02:27	3	0:00:55	15	0:00:30	3	0:01:38	1
14	47	>0:09:08	−−	0:00:13	3	0:01:54	1	0:00:34	17	0:00:10	1	0:00:58	1
14	56	>0:11:40	−−	0:00:22	3	0:02:30	1	0:01:49	29	0:00:18	1	0:02:12	1
14	65	>0:14:30	−−	0:01:54	27	0:15:04	11	0:03:19	21	0:02:06	3	0:13:48	3
14	74	>0:17:27	−−	0:00:09	3	0:02:08	1	0:01:38	41	0:00:15	1	0:02:10	1
14	83	>0:20:19	−−	0:00:19	3	0:06:56	3	0:14:48	329	0:00:29	3	0:05:43	1
16	37	>0:09:33	−−	0:03:11	2447	0:07:44	381	0:01:04	33	0:00:23	1	0:01:07	1
16	46	>0:13:11	−−	0:00:38	55	0:01:58	3	0:00:39	1	0:00:14	1	0:01:03	1
16	55	>0:16:48	−−	0:01:10	3	0:04:29	1	0:03:49	27	0:00:53	3	0:02:39	1
16	64	>0:20:36	−−	0:01:04	3	0:05:07	3	0:04:51	9	0:00:56	1	0:03:44	1
16	73	>0:24:19	−−	0:02:24	3	0:14:58	1	0:07:06	3	0:02:48	1	0:12:04	1
16	82	>0:28:52	−−	0:02:09	9	0:15:21	1	0:11:27	11	0:02:17	3	0:14:43	3
16	91	>0:33:43	−−	0:02:23	3	0:28:02	5	0:43:05	71	0:03:03	3	0:14:59	1
16	100	>0:40:05	−−	0:00:33	1	0:07:40	1	0:18:40	17	0:00:57	1	0:09:05	1
16	103	>0:42:10	−−	0:01:43	1	0:23:00	3	0:36:19	61	0:02:23	3	0:18:32	1
20	38	0:03:29	3049	0:02:12	1597	0:02:58	1019	0:00:15	1	0:00:13	1	0:00:26	1
20	41	>0:22:24	−−	>0:26:10	−−	0:30:13	6917	0:02:04	15	0:00:57	3	0:01:16	3
20	44	>0:24:49	−−	>0:31:19	−−	1:00:08	8961	0:04:36	7	0:03:10	1	0:03:09	1

6.2 The Quadratic Linear Ordering Problem

According to our experience, separating inequalities known to be valid for the polytope P does not speed up the optimization over Q considerably, and so our basic branch-and-cut algorithm for the solution of (QLO_1) and (QLO_2) only separates the 3-dicycle inequalities. The latter are known to be facets for the linear ordering polytope. In contrast to the quadratic matching case, max-cut separation turns out to be very effective for the quadratic linear ordering problem, and so it is called in every node of the branch-and-bound tree. The target cut separation is again performed on randomly chosen chunks that are generated via trivial projection. The vectors that are extendable under the trivial projection are again linear orderings on the chunks.

We studied instances defined on complete graphs. Again, weights of linear and product variables are chosen randomly in $\{-1000, \ldots, 1000\}$. All products are generated. An instance is defined by the number of nodes n of the complete graph and a random seed r for the randomly chosen weights. Moreover, we created linear arrangement instances defined by random graphs, see Section 5.3.

In Table 2 we show some running times for both types of instances. As above, we report the cpu time in seconds and the number of subproblems needed to solve the instance to optimality.

The results are similar to the quadratic matching case: the basic implementation solves the problem essentially via branching, only very small instances can be solved. Again it is obvious that the separation of inequalities from the cut polytope considerably improves the running time. Also the target cut separation strongly reduces the number of subproblems, the number of linear problems and the running time. The best chunk sizes are 5 to 6.

In summary, our results show that both presented separation methods improve the performance of the basic branch-and-cut approach significantly.

7 Conclusion

We present and evaluate two methods for improving the performance of branch-and-cut approaches to general quadratic 0/1 optimization problems, addressing the problem from two different directions. The first method addresses the quadratic structure, exploiting separation routines for cut polytopes, while the second implicitly takes into account the specific structure of the underlying polytope, applying a technique similar to local cut generation. Our results show that the total running time can be decreased significantly by both techniques.

References

1. Applegate, A., Bixby, R., Chvátal, V., Cook, W.: TSP cuts which do not conform to the template paradigm. In: Jünger, M., Naddef, D. (eds.) Computational Combinatorial Optimization. LNCS, vol. 2241, pp. 261–304. Springer, Heidelberg (2001)
2. Barahona, F., Mahjoub, A.R.: On the cut polytope. Mathematical Programming 36, 157–173 (1986)

3. Buchheim, C., Liers, F., Oswald, M.: Local cuts revisited. Operations Research Letters (2008), doi:10.1016/j.orl.2008.01.004
4. De Simone, C.: The cut polytope and the Boolean quadric polytope. Discrete Mathematics 79, 71–75 (1990)
5. Deza, M., Laurent, M.: Geometry of Cuts and Metrics. Algorithms and Combinatorics, vol. 15. Springer, Heidelberg (1997)
6. Liers, F., Jünger, M., Reinelt, G., Rinaldi, G.: Computing Exact Ground States of Hard Ising Spin Glass Problems by Branch-and-Cut. New Optimization Algorithms in Physics, pp. 47–68. Wiley-VCH, Chichester (2004)
7. Rendl, F., Rinaldi, G., Wiegele, A.: A branch and bound algorithm for Max-Cut based on combining semidefinite and polyhedral relaxations. In: Fischetti, M., Williamson, D.P. (eds.) IPCO 2007. LNCS, vol. 4513, pp. 295–309. Springer, Heidelberg (2007)