Practical Mixed-Integer Optimization for Geometry Processing

David Bommes, Henrik Zimmer, and Leif Kobbelt

RWTH Aachen University,
Ahornstr. 55, 52074 Aachen, Germany
http://graphics.rwth-aachen.de

Abstract. Solving mixed-integer problems, i.e., optimization problems where some of the unknowns are continuous while others are discrete, is NP-hard. Unfortunately, real-world problems like e.g., quadrangular remeshing usually have a large number of unknowns such that exact methods become unfeasible. In this article we present a greedy strategy to rapidly approximate the solution of large quadratic mixed-integer problems within a practically sufficient accuracy. The algorithm, which is freely available as an open source library implemented in C++, determines the values of the discrete variables by successively solving relaxed problems. Additionally the specification of arbitrary linear equality constraints which typically arise as side conditions of the optimization problem is possible. The performance of the base algorithm is strongly improved by two novel extensions which are (1) simultaneously estimating sets of discrete variables which do not interfere and (2) a fill-in reducing reordering of the constraints. Exemplarily the solver is applied to the problem of quadrilateral surface remeshing, enabling a great flexibility by supporting different types of user guidance within a real-time modeling framework for input surfaces of moderate complexity.

Keywords: Mixed-Integer Optimization, Constrained Optimization

1 Introduction

The problem of optimizing objective functions $E(\mathbf{x})$ where one part of the unknowns is real valued $x_{i\in R} \in \mathbb{R}$ and the other unknowns are required to be integers $x_{j\in I} \in \mathbb{Z}$ is referred to as *Mixed-Integer Problem* (MIP). Typically such problems arise whenever a continuous optimization depends on some discrete decisions. One example from structural engineering is the optimization of a supporting structure, where two points of this structure can either be connected by a beam or not while in contrast to this discrete decision the geometric positions of the connected points can be varied continuously.

Mixed-Integer Problems are generally NP-hard to solve [1, 2], which make common approaches (e.g., Branch and Bound [3, 4, 2] or Cutting-Plane approaches [5, 2]) unfeasible for many real-world problem instances.

In this article we present an algorithm for efficiently and accurately approximating *quadratic* MIPs represented by quadratic energy functions

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^t A\mathbf{x} - \mathbf{x}^t\mathbf{b} \rightarrow min, \quad \mathbf{x} \in \mathbb{R}^n \tag{1}$$

with A symmetric and positive definite, subject to n_I integer constraints

$$x_{i\in I} \in \mathbb{Z}, \quad I \subseteq \{1, \ldots, n\}. \tag{2}$$

Additionally the feasibility of the solution $\mathbf{x}$ is restricted by n_C linear *equality* constraints of the form

$$\mathbf{C}_i \cdot \mathbf{x} = d_i \qquad \text{with} \qquad \mathbf{C}_i \in \mathbb{R}^n,\ d_i \in \mathbb{R} \tag{3}$$

which can be assembled into a single matrix $C\mathbf{x} = \mathbf{d}$ with dimension $C \in \mathbb{R}^{n_C \times n}$. Here n, n_I and n_C denote the number of variables, integer constraints and linear constraints respectively. Note also that the above formulation differs slightly from the most general setting of mixed-integer problems where additionally *in-equality* constraints are given. In the presence of in-equality constraints even finding any random feasible solution can get very hard, see e.g., [2].

Our algorithm successively determines the values of the discrete variables $x_{i\in I} \in \mathbb{Z}$ in a greedy fashion. Fixing the value of a discrete variable is equivalent to adding one explicit linear constraint $x_i = k$ with $k \in \mathbb{Z}$ to our optimization problem. Therefore our algorithm successively transforms integer constraints into explicit linear constraints until all of them are fulfilled. More precisely we start by neglecting the n_I integer constraints and compute the minimizer of this so called relaxed problem by setting the partial derivatives $\frac{\partial E}{\partial x_i} = 0$ and solving the resulting linear system

$$A\mathbf{x}^0 = \mathbf{b}. \tag{4}$$

Here we assumed $n_C = 0$ for clarity reasons. The values of the solution vector $\mathbf{x}^0$ can be seen as continuous estimates of the desired discrete integer variables. However, we found that estimating all integer constraints at once, i.e., requiring $\forall i \in I: \ x_i^1 = round(x_i^0)$, leads to poor results since the individual estimates cannot influence each other. Motivated by this observation we instead successively determine single integer constraints $x_j^{k+1} = round(x_j^k)$ which are henceforth used to solve subsequent relaxed problems until a feasible solution of our initial optimization problem is found, meaning that all $x_{i\in I}$ are integers.

By greedily choosing the continuous estimate which has the smallest deviation $|round(x_j^k) - x_j^k|$ from an integer in each step these subsequent relaxed problems can be solved very efficiently by a carefully designed three-level solver as presented in Section 3.2. The performance can further be improved by identifying sets of relaxed variables which do not interfere too much and hence can be estimated simultaneously.

In order to facilitate an efficient handling of arbitrary linear constraints C_i we propose to eliminate one variable for each constraint (Section 2) and support

this approach by a fill-in reducing constraint reordering (Section 2.2) which in practice significantly reduces the runtime.

In Section 4 the capabilities of the presented solver are illustrated exemplary by applying it to the surface quadrangulation problem. A large variety of possible user guidance like e.g., prescribing some singularities or preserving feature curves of the input geometry in the generated quadrangulation is achieved by simply adding additional linear constraints. Therefore the presented algorithm enables a powerful quadrangulation algorithm which is very flexible and allows for a wide range of different application scenarios ranging from a fully automatic setting up to a complete manual meshing. Finally we illustrate the immense performance benefit due to the novel extensions of the original algorithm used in [6], i.e., the rounding of sets of variables and the fill-in reducing reordering of constraints.

1.1 Previous work

To the best of our knowledge the idea of approximating MIPs by a series of real-valued problems started with [13], set in the field of Structural Engineering. Ringertz' idea of rounding variables iteratively and re-solving the problem has been cited several times and depending on the problem setting small variations appear in the proposed solutions. While some researchers argue for the use of post-processing methods as, depending on the problem and the type of variables, always rounding up (or down) might not be meaningful [16], others suggest rounding both up and down and keeping the solution with lower cost [17].

Regardless of the rounding strategy, what these approaches all have in common is that the full-sized system of linear equations needs to be re-solved in each iteration, making the iterative strategy unfeasible for many practical applications.

In the field of Geometry Processing the idea of approximating quadratic MIPs by rounding variables of a real-valued linear system has been successfully adapted by several authors (see e.g.,[9][11]). Here direct-rounding strategies were used, where the system had to be solved only twice, once initially and once after all integer constraints have been estimated (all at once). This approach is usually only applicable for MIPs with a small number of integer variables that do not interfere too much and otherwise leads to a poor approximation of the optimal solution.

The article is structured as follows: Section 2 describes the proper handling of linear constraints within the optimization of a quadratic energy, which is central also for the integer constraints discussed in Section 3. In Section 3.2 we describe an efficient update strategy which enables the iterative addition of integer constraints. Finally in Section 4 we discuss some experiments performed within the context of quadrilateral surface remeshing.

2 Linear Constraints

The ability to properly handle constraints is vital for the wide applicability of an optimization method. For a problem to be solvable usually some boundary constraints are needed to limit the solution space, or often additional user-defined design constraints might be incorporated to shape the resulting solution. In our setting we also have to deal with integer constraints which translate into simple linear conditions as soon as the specific integer is known. A common way to handle linear constraints are *Lagrangian Multipliers* as discussed next.

2.1 Lagrangian Multipliers

The method of Lagrangian Multipliers turns a constrained problem into a unconstrained one by adding one additional term per constraint to the energy. Updating energy (1) with the constraints (3) we end up with the following energy formulation:

$$E_L(\mathbf{x}) = E(\mathbf{x}) + \sum_{i=1}^{n_C} \lambda_i (C_i \cdot \mathbf{x} - d_i) \tag{5}$$

where the solution is given by the following system of linear equations

$$\frac{\partial E_L}{\partial \boldsymbol{x}} = 0 \quad \rightsquigarrow \quad \begin{bmatrix} A & C^T \\ C & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{d} \end{bmatrix} \tag{6}$$

describing the stationary point of the adapted energy. Note that the approach of Lagrangian Multipliers is not restricted to quadratic energies nor linear constraints but can be applied to non-linear problems as well, for more details see e.g., [10]. Unfortunately the approach of Lagrangian Multipliers comes with certain disadvantages making them impractical for our purpose. Instead of decreasing the number of degrees of freedom as more constraints are added the opposite is the case since for each constraint a Lagrangian Multiplier λ_i is introduced. Furthermore the symmetric positive definiteness (s.p.d.), inherent in linear systems arising from convex quadratic energies is destroyed by the diagonal block of zeros $\mathbf{0}$ effectively disabling the use of highly efficient solvers such as CHOLMOD (see [7]) and necessitating the use of slower more general solvers such as SuperLU (see [8]). As will be seen in Section 3 the s.p.d. property is crucial for the efficient local updates of the adaptive three-level solver in Section 3.2. Therefore next we describe a proper handling of linear constraints which maintains the s.p.d. property.

2.2 Elimination approach

Assume we want to minimize a quadratic energy $E(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^n$ subject to a single linear constraint $\mathbf{D}^T\mathbf{x} - d = 0$. Geometrically this means restricting the solution space to a $n-1$ dimensional hyperplane. Consequently it is possible to convert the above problem into a new unconstrained one with $n-1$ degrees

of freedom. Assume w.l.o.g. that $D_n \neq 0$ such that we can solve the linear constraint for x_n expressing it as a function of $(x_1, \ldots, x_{n-1})$

$$x_n(\underbrace{x_1, \ldots, x_{n-1}}_{\tilde{\mathbf{x}}}) = (d/D_n) - \sum_{j=1}^{n-1} (D_j/D_n) x_j =: f - \mathbf{F}^T \mathbf{x} \tag{7}$$

and transforming the above constrained problem into the desired unconstrained form

$$\tilde{E}(\tilde{\mathbf{x}}) := E\underbrace{\begin{pmatrix} \tilde{\mathbf{x}} \\ x_n(\tilde{\mathbf{x}}) \end{pmatrix}}_{\mathbf{y}} \tag{8}$$

with equivalent minima where x_n can be computed from $\tilde{\mathbf{x}}$ by equation (7).

To compute the minimizer $\tilde{\mathbf{x}}$ we now have to solve a $(n-1)$ dimensional system of linear equations $\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ which can be derived by partitioning the matrix A of equation (1) into four blocks (with $\overline{A} \in \mathbb{R}^{(n-1)\times(n-1)}$, $\boldsymbol{v} \in \mathbb{R}^{n-1}$ and $w \in \mathbb{R}$) and re-factorizing the result:

$$\begin{aligned} \tilde{E}(\tilde{\mathbf{x}}) = \frac{1}{2}\mathbf{y}^T A \mathbf{y} - \mathbf{y}^T \mathbf{b} \;&=\; \frac{1}{2}\mathbf{y}^T \begin{pmatrix} \overline{A} & \mathbf{v} \\ \mathbf{v}^T & w \end{pmatrix} \mathbf{y} - \mathbf{y}^T \begin{pmatrix} \overline{\mathbf{b}} \\ b_n \end{pmatrix} \\ &= \frac{1}{2}\tilde{\mathbf{x}}^T \underbrace{\left(\overline{A} - \mathbf{v}\mathbf{F}^T - \mathbf{F}\mathbf{v}^T + w\mathbf{F}\mathbf{F}^T\right)}_{\tilde{A} \,\in\, \mathbb{R}^{(n-1)\times(n-1)}} \tilde{\mathbf{x}} - \tilde{\mathbf{x}}^T \underbrace{\left(\overline{\mathbf{b}} + \mathbf{F}(fw - b_n) - \mathbf{v}f\right)}_{\tilde{\mathbf{b}} \,\in\, \mathbb{R}^{n-1}} + \text{const} \end{aligned} \tag{9}$$

Note that since A is s.p.d. $\tilde{A}$ is also s.p.d. enabling highly efficient solution methods used in our three-level solver described in Section 3.2. Of course instead of eliminating the last variable each other variable can be chosen by re-indexing.

Multiple Constraints: In general we want to handle an arbitrary number of linear constraints which can be achieved by iteratively eliminating one variable for each constraint from $\{C_1, \ldots, C_{n_C}\}$. One very important aspect of multiple constraints is that in each step it is necessary to eliminate the chosen variable from all subsequent constraints since obviously once a variable is constrained and eliminated from the optimization problem it should not be reintroduced by a following constraint. More precisely, after constraining a variable x_k through a constraint C_j we have to do Gaussian elimination in the constraint matrix C in order to bring all $C_{i,k}$ with $i > j$ to zero. Clearly the constraints in the updated matrix are equivalent to the original problem.

Choosing elimination variables: For each linear constraint we have to pick a variable which is subsequently constrained by the induced linear function and eliminated from the problem. All non-zero coefficients of the linear constraint induce a valid possibility. To increase numerical stability we select the variable

whose coefficient has the largest absolute value. However, there is one important aspect to consider whenever a variable x_k with $k \in I$, i.e., which has to satisfy an integer constraint, is selected for elimination. In general it can get very problematic to guarantee that the values of the induced linear function are chosen in such a way that x_k becomes an integer. Consequently for the elimination we always prefer non-integer variables with $k \notin I$. For constraints where all non-zero coefficients belong to integer variables we currently support only those cases where all coefficients are integer and their greatest common divisor (gdc) is one of these coefficients. In such a case we can safely divide all coefficients by their gdc and eliminate a variable with coefficient ± 1 since a linear combination of integers multiplied by integers is always an integer and consequently the integer constraint is fulfilled by construction. For many practical problems (like quadrilateral surface remeshing) the above assumptions are always fulfilled and therefore we leave the more complicated general case for future work. Whenever the above assumption is violated it may happen that in the result some of the integer constraints are not fulfilled.

Linear dependent or conflicting constraints: Since we iteratively process the individual constraints it is easy to identify linear dependent or conflicting constraints. This is a big advantage compared to the method of Lagrangian Multipliers which would construct an underdetermined system of linear equations not suitable for efficient standard solvers. In our implementation linear dependent or conflicting constraints are simply neglected. This behavior is very convenient since the user does not have to spend additional effort identifying the subset of linear independent constraints e.g., in the case of user provided side conditions. Due to numerical inaccuracies of floating point numbers linear dependency is checked against a tolerance which has a default value of 10^{-6}.

Fill-in reducing constraint reordering: Although mathematically equivalent the linear system belonging to the unconstrained optimization problem after processing all constraints can take many different patterns, strongly depending on the processing sequence of the constraints. In spirit of sparse Cholesky methods like [7] we are interested in finding an ordering of the constraints which minimizes the fill-in (nonzero elements) and hence increases performance. Unfortunately there is no known algorithm to achieve the best ordering apart from the naive one which explicitly checks all orderings. Obviously such an approach is far too slow such that a good compromise in form of a cheap heuristic is more desirable. Our experiments show that processing the constraints sorted by their number of non-zero coefficients leads to much higher performance than just using a random ordering (see Section 4 for timings). Please note that this ordering is dynamic since while processing the constraints the number of non-zero coefficients is altered by the Gaussian elimination steps.

After eliminating one variable per linear constraint we obtain a new equivalent optimization problem, i.e., a quadratic energy minimization subject to a set

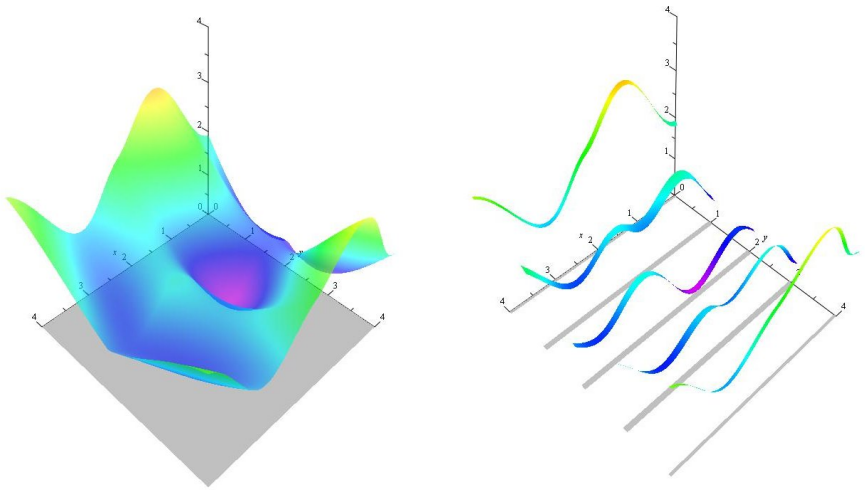

Fig. 1. (left) a continuous optimization problem where each point in the plane $\mathbb{R} \times \mathbb{R}$ is a feasible solution, i.e., a point which fulfills all constraints of the problem. (right) a mixed-integer problem where the set of feasible solutions is $\mathbb{R} \times \mathbb{Z}$. For minimizing such problems typically all discrete possibilities have to be tested explicitly.

of integer constraints. We describe next how a good approximate solution can be found efficiently.

3 Integer Constraints

The integer constraints of our initial problem dictate that for each feasible solution a subset of the variables have to be integers, i.e., $x_{i \in I} \in \mathbb{Z}$. Finding a feasible solution is simple in this formulation, since there are no dependencies between the individual variables. Therefore just setting up a set of additional linear constraints which fix the $x_{i \in I}$ variables to arbitrary integers like e.g., $x_{i \in I} = 0$ and enforcing them with the method from the previous section would indeed result in a feasible solution. However the problem of finding the best one of all these possible assignments, i.e., the one which minimizes the energy, is very hard. In contrast to continuous convex optimization it is not sufficient to simply walk into the direction of the negative gradient (see Figure 1). In general to find the optimum it would be necessary to derive lower and upper bounds for each discrete variable and then explicitly test all discrete combinations. Please notice that for problems with a large number of discrete variables even for narrow bounds like e.g., a binary problem where $x_{i \in I} \in \{0, 1\}$ such an approach is very expensive and would already require the solution of $2^{|I|}$ full-sized problems.

3.1 Direct Rounding

Instead of achieving optimality for practical problems we aim at finding an approximate solution which is close enough to the optimum but can be computed in a fraction of time. The most efficient way to determine adequate assignments for the integer variables is to estimate them from a relaxed solution, i.e., computing the minimizer $\boldsymbol{x}^c$ where all variables are allowed to be continuous leading to the estimates $x_{i\in I} = round(x_i^c)$. Following (9) the elimination approach results in a very simple update for such explicit constraints:

$$\tilde{A} = \overline{A} \quad \text{and} \quad \tilde{\boldsymbol{b}} = \overline{\boldsymbol{b}} - \boldsymbol{v} \cdot round(x_i^c) \tag{10}$$

Estimating all integer assignments at once which is called *direct rounding* is very efficient since it requires the solution of only two linear systems. However the drawback is that the interrelation between the discrete variables is completely ignored which often leads to poor results (see e.g., the comparison in [6]). This suggests to successively add one integer constraint at a time and immediately compute the altered relaxed problem to update the estimates of the yet unconstrained discrete variables. This strategy is denoted *iterative rounding* and is discussed in more detail next.

3.2 Iterative Greedy Rounding

The key to an efficient implementation of the iterative rounding is the observation that, for problems with sparse variable dependencies (few non-zeros per row), changing the value of one variable usually has little influence on "far-away" variables. This is a property inherent in many Geometry Processing problems formulated over, e.g., simplicial complexes or spline bases with local support.

The problem inherent to iterative rounding is that it requires the solution of $|I|+1$ many linear systems which can get very slow when implemented in a naïve way. Fortunately in many steps of this iterative process the solution changes only slightly which can be exploited by carefully designed iterative solvers.

Suppose that we have computed the solution of the relaxed problem $A\boldsymbol{x} = \boldsymbol{b}$ and that we want to add a single integer constraint. Following (10) the residual $\boldsymbol{e} = \tilde{A}\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{b}}$ after adding the new constraint has the same nonzero pattern as $\boldsymbol{v}$. And consequently for a sparse $\boldsymbol{v}$ the relaxed solution from the previous step $\tilde{\boldsymbol{x}}$ violates only a few equations of the linear system. Due to this observation we first try to iteratively update the solution only where it is necessary, i.e., for all variables $\tilde{x}_i$ with $|e_i| > \epsilon$. This so called *Local Gauss-Seidel* method executes single Gauss-Seidel updates for variables with a local residual above the allowed tolerance. All these candidates are stored in a queue and convergence is reached when the queue gets empty meaning that all residuals are below the prescribed tolerance. Notice that due to the elimination approach the system matrix remains s.p.d. guaranteeing convergence of the Gauss-Seidel method. The complete algorithm is depicted below:

Algorithm: *Local Gauss-Seidel*
Input: Linear system $A\boldsymbol{x} = \boldsymbol{b}$ (which is not fulfilled)
Index set of variables with non-zero residual N,
End conditions ϵ and $maxiters_{GS}$
Output: Updated $\mathbf{x}$ with residuals $|e_k| < \epsilon$ or NOT converged.
01: push N onto *queue*
02: $iter = 0$
03: **while** *queue* not empty **and** $iter < maxiters_{GS}$
04: iter = iter +1
05: $x_k = \text{pop}(\ queue\ )$
06: $e_k = b_k - \sum_{j=1}^{n} A_{kj}x_j$
07: **if** $|e_k| > \epsilon$ **then**
08: $x_k \leftarrow x_k + e_k/A_{kk}$
09: push $nonzero(A_{k*})$ onto *queue*
07: **end if**
10: **end while**

The parameters ϵ and $maxiters_{GS}$ can be chosen by the user. In cases where the above method does not converge within the prescribed number of iterations, a more global conjugate gradient method is used and in rare cases where this is still not sufficient after a few iterations a sparse Cholesky method is executed. This adaptive solution strategy is very fast if the previous solution is close to the new one and only spends more time if a novel integer constraint has global impact. In our implementation the conjugate gradient solver is taken from the GMM++ library [12] and the Sparse Cholesky solver is the CHOLMOD solver [7].

In this iterative rounding strategy we can choose $|I|!$ many different orders in which the integers are estimated. A natural greedy choice is the yet unconstrained integer variable whose estimate has the smallest deviation $|x_i - round(x_i)|$ from an integer since it is most likely to be correct. A nice side effect of this strategy is that it increases the efficiency of the above hierarchical solution strategy. The reason is that for small deviations from an integer also the non-zero residuals usually get small. The complete iterative greedy rounding algorithm is shown below:

Algorithm: *Iterative Greedy Rounding*
Input: Linear system of relaxed problem $A\boldsymbol{x}^c = \boldsymbol{b}$ with $\boldsymbol{x}^c, \boldsymbol{b} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$
index set of integer variables $I \subset \{1, \ldots, n\}$
Output: Approximation of mixed-integer solution $\boldsymbol{x} \in \mathbb{R}^n$ satisfying $x_{i\in I} \in \mathbb{Z}$
01: $\boldsymbol{x} = \boldsymbol{x}^c$
02: **while** $I \neq \emptyset$
03: *// greedy selection*
04: $j = \arg\min_{i\in I}(|x_i - round(x_i)|)$

```
05:    I ← I \ j
06:    // add new constraint and get nonzero residuals N
07:    N = eliminateConstraint( x_j = round(x_j) , A , x , b )
08:    // update solution
09:    converged = localGaussSeidel( A, x, b, N) // level 1
10:    if not converged then
11:       converged = conjugateGradient( A, x, b) // level 2
14:       if not converged then
15:          sparseCholesky( A, x, b) // level 3
16:       end if
17:    end if
18: end while
```

To avoid the necessary re-indexing of the variables in the above algorithm the update rule (10) was slightly modified by keeping an identity row and column for each eliminated variable x_k, i.e., $\tilde{A}_{kj} = \tilde{A}_{jk} = \delta_{kj} \quad \forall j$.

In our implementation the user is able to control the behavior of the adaptive three level solver with several parameters. First of all the tolerance ϵ for checking convergence of the iterative methods (level 1 and 2) and a maximum number of iterations $maxiters_{GS}$ and $maxiters_{CG}$ can be adjusted. Furthermore it is possible to disable complete levels. The reason is that for mixed-integer problems where it is known that the rounding of a discrete variable always has global impact it is e.g., not reasonable to execute the Local Gauss-Seidel step since it would almost never converge. Therefore it is very important to experiment a little bit with these parameters in order to optimize the performance for a specific class of problems. In Section 4 we will provide two different useful settings for the quadrangulation problem.

Simultaneous Rounding: The motivation for the iterative rounding strategy was mainly the observation that the estimates of individual integer variables should influence each other to achieve satisfactory accuracy. It would be possible to achieve the same accuracy in fewer computation steps if some prior knowledge about the rate of influence between variables is available. Clearly variables which do not influence each other could be rounded simultaneously in one step without introducing an error. Unfortunately computing the influence between variables corresponds to the solution of a full-sized linear system which would be too expensive. What we need instead is a cheap apriori estimate which never underestimates the interdependency. A very simple apriori estimate which holds for many problems is the following one: If one variable is changed by a value of Δx due to a constraint all other variables are changed by a value smaller or equal than Δx. Consequently in each step several variables can be rounded as long as their estimated maximal deviation $\sum_i \Delta x_i$ does not influence any of the rounding decisions. Obviously this apriori estimate does not hold for all problems. However we included the possibility to use it into our implementation

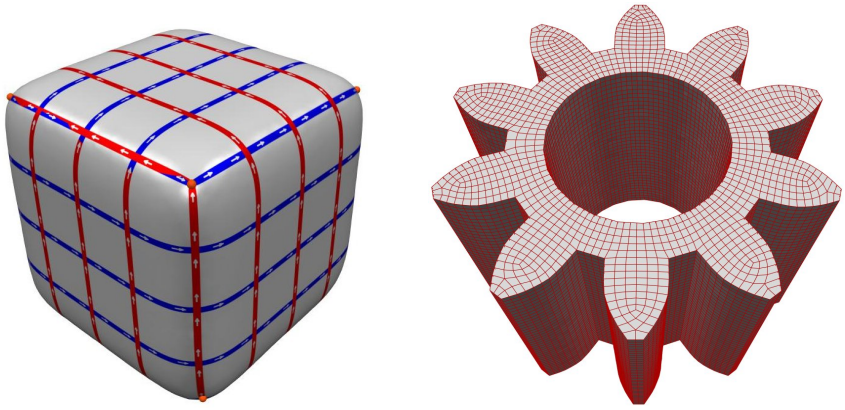

Fig. 2. (left) the SMOOTHED CUBE model with low geometric complexity. (right) the PINION model with many sharp features.

since it is useful for many practical applications and can speed up the computation significantly. Finding a cheap way for estimating sharper bounds for the interdependency between discrete variables is an interesting question for future work.

3.3 Open Source CoMISo Library

On the web page `http://www.graphics.rwth-aachen.de/comiso` the source-code of the solver explained above as well as example programs can be found. Note that even though this solver was created for sparse problems as they usually occur e.g., in areas of Finite Elements or Geometry Processing, it can also be applied to dense problems without any modifications. However, in some cases it might be advantageous to replace sparse-specific parts such as the sparse Cholesky solver by dense-optimized counterparts.

4 Experiments

We evaluate our algorithm by applying it to the surface quadrangulation problem as formulated in [6]. In this method two mixed-integer problems have to be solved where the first one is the computation of a smooth orientation field while the second one is a seamless parametrization mapping singularities and feature edges to integer grid points and lines respectively. For more details about the quadrangulation method we refer the reader to [6]. With the help of several experiments we derived two different parameters for the two diverse problems. For the computation of the orientation field we used $\epsilon = 10^{-3}$, $maxiters_{GS} = 100000$ and $maxiters_{CG} = 50$ while for the parametrization we chose $maxiters_{GS} = 0$, $maxiters_{CG} = 20$ and the use of sparse cholesky was disabled completely. The reason for those different parameter settings is that both problems have very

MODEL	10k	50k	200k	800k
ARMADILLO	0.3	1.2	6.3	33.9
CUBE	0.11	0.5	2.8	18.5

Table 1. Orientation Field Timings in s

MODEL	10k	50k	200k	800k
ARMADILLO	1.3	5.3	21.4	100.3
CUBE	0.15	0.9	6.7	55.1

Table 2. Parametrization Timings in s

diverse characteristics. While the orientation field exhibits a large number of integers with local influence, the parametrization problem requires only few integers but with rather global influence. With the above settings we were able to compute visually equivalent results compared to the original algorithm of [6] within a fraction of time. The performance benefit is a result of the tuned parameters as well as the novel extension which are the fill-in reducing reordering, the simultaneously rounding and some changes within the internal data structures. All examples were computed on a single CPU of an intel i7 quadcore 2.80GHz with 8GB of RAM.

Performance: To give one representative example the orientation field computation on the LEVER model of [6] took $3.3s$ compared to $0.22s$ while the parametrization timing decreases from $19.9s$ to $2.8s$. However, further experiments showed that the runtime strongly depends on the geometric complexity of the object. In Table 1 we compare the timing of the orientation field computation of the ARMADILLO model (Figure 3) and a simple SMOOTHED CUBE (Figure 2). For the same number of triangles the geometric more complex ARMADILLO (121 singularities) model needs more computation time than the smoothed cube (4 singularities). In the case of constant geometric complexity the runtime depends almost linearly on the number of triangles, enabling very large inputs. A similar behavior can be observed for the parametrization problem in Table 2. The algorithm behaves sensible to the geometric input complexity and nicely adapts to situations of different difficulty which is due to the simultaneous rounding approach.

To underline the importance of the fill-in reducing reordering we did a separate experiment where the PINION model (Figure 2) with many sharp features was parametrized, leading to a huge set of dependent integer constraints. By applying the reordering the computation took $1.3s$ and the system matrix had $418k$ nonzero entries compared to a much slower runtime of $7.4s$ and $581k$ nonzero entries without the reordering.

Besides the performance our algorithm offers a nice flexibility due to the convenient and robust handling of linear constraints as underlined by the following experiment.

Flexibility: Often designers are not satisfied with the result of fully automatic quadrangulation algorithms because they want additional symmetries or structures alleviating animation. Therefore we extended the method of [6] by an

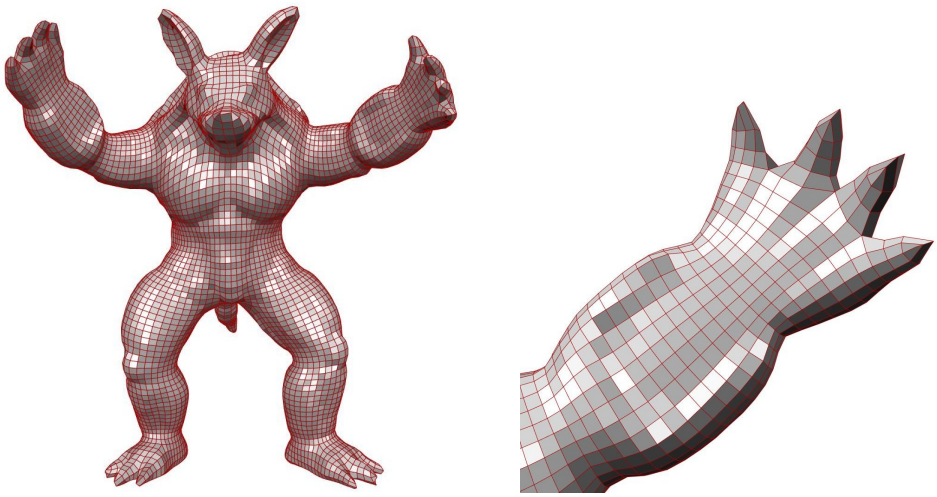

Fig. 3. (left) a quadrangulation of the ARMADILLO designed in an interactive session. (right) close up of the right hand where valence two singularities at the finger tips were created manually.

interactive manipulation mode where additional (linear) constraints can be iteratively provided by the user. After the fully automatic computation the user is able to (1) change (move, add, remove) singularities, (2) connect singularities by a parametric line to improve the high-level structure of the quadrangulation like used in [15] or (3) add element orientation or alignment constraints. It turned out that such an interaction could be provided easily due to the available robust handling of linear constraints. In the extreme case of specifying all singularities within the orientation field our method is equivalent to [11] and [14]. However in contrast to them our approach is flexible enough to compute solutions for an arbitrary number of known singularities perfectly supporting an interactive design approach. Figure 3 shows the result of an interactive session where the user provided a few orientations. Furthermore some singularities (of the automatic solution) at the hand of the ARMADILLO were merged into valence 2 singularities to capture the spiky shape of the fingers without requiring a very small edge length.

5 Conclusion

In this article we presented the technical details of our mixed-integer approximation algorithm for linearly constrained quadratic mixed-integer problems. By identifying suitable algorithm settings for a given class of optimization problems high efficiency combined with sufficient accuracy is achieved as illustrated by the quadrangulation example. In the future we would like to apply our algorithm

to more mixed-integer problems from Geometry Processing and explore further strategies which extend and generalize the idea of simultaneous rounding.

Acknowledgments

This work has been supported by the UMIC Research Centre, RWTH Aachen University.

References

1. Floudas, C.A.: Nonlinear and Mixed Integer Optimization: Fundamentals and Applications, Oxford University Press, New York (1995)
2. Nowak, I.: Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming, Birkhäuser, Basel (2005)
3. Gupta, O.K., Ravindran, A.: Branch and Bound Experiments in Convex Nonlinear Integer Programming, Manage Sci., 31 (12), 1533-1546, (1985)
4. Quesada, I., Grossmann, I.E.:An LP/NLP Based Branch and Bound Algorithm for Convex MINLP Optimization Problems, Computer Chem. Eng., 16, 937-947, (1992)
5. Westerlund, T., Petersson, F.: A Cutting Plane Method for Solving Convex MINLP Problems, Computers Chem. Eng., 19, 131-136, (1995)
6. Bommes, D., Zimmer, H., Kobbelt, L.: Mixed-integer quadrangulation. ACM Trans. Graph. 28(3), 1-10 (2009)
7. Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S.: Algorithm 8xx: Cholmod, supernodal sparse cholesky factorization and update/downdate. Technical Report TR-2006-005, University of Florida (2006)
8. Demmel, J.W., Eisenstat, S.C., Gilbert, J.R., Li, X.S., Liu, J.W.H.: A supernodal approach to sparse partial pivoting. SIAM J. Matrix Analysis and Applications 20(3), 720-755 (1999)
9. Kaelberer, F., Nieser, M., Polthier, K.: Quadcover - surface parameterization using branched coverings. Computer Graphics Forum 26(3), 375-384 (Sep 2007)
10. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer (August 1999)
11. Ray, N., Vallet, B., Li, W.C., Lévy, B.: N-symmetry direction field design. ACM Transactions on Graphics (2008), presented at SIGGRAPH
12. Renard, Y.: gmm++, Generic Matrix Methods. `http://home.gna.org/getfem/gmm_intro.html` (2003)
13. Ringertz, U.T.: On methods for discrete structural optimization. Engineering Optimization 13(1), 47-64 (1988)
14. Crane C., Desbrun M., Schröder P.: Trivial Connections on Discrete Surfaces. Computer Graphics Forum (SGP) 29(5), 1525-1533 (2010)
15. Myles A., Pietroni N., Kovacs D., Zorin D.: Feature-aligned T-meshes. ACM Trans. Graph. 29(4), 1-11 (2010)
16. Yu X., Zhang S., Johnson E.: A discrete post-processing method for structural optimization. Engineering with Computers 19(2), 213-220 (2003)
17. Groenwold A. A., Stander N., Snyman J. A.: A pseudo-discrete rounding method for structural optimization. Structural and Multidisciplinary Optimization 11(3), 218-227 (1996)