

Reliable Linear Program Solver for Coho

Chao Yan (20378048)

December 14, 2005

1 Introduction

Coho[3],[4],[5],[6] is a hardware verification system. It models a circuit by a continuous, differential equations; then verifies that some properties are hold by reachability analysis. However, closed form solutions for reachable problems usually do not exist, approximation techniques are performed. Coho computes an over-approximation of the reachable space: including every point that is actually reachable. Therefore, Coho may fail to verify a correct system, but will never erroneously verify an incorrect system.

In many steps, such as computing the evolution of reachable space of a system, Coho solves a large number of linear programs. Linear programming has a classical *Simplex* algorithm. However, it may produce under-approximation result because of floating point computation, and the error might be very large and not bounded because many of the linear programming in Coho are highly ill-conditioned.

We developed a novel solver for Coho form linear programming. We use interval computation to bound the error, and apply an $O(n)$ linear system solver to reduce intermediate computation error while the solver is still efficient. We also find a over-approximation cost when the optimal basis is ill-conditioned.

The next section is the definition of Coho LP; while section 3 introduces interval computation; we presents the algorithm of CohoSolver in the 4th section, and application and conclusion followed.

2 Coho Linear Programming

In Coho, we represent a high-dimensional polyhedron by its projections onto two-dimensional sub-spaces¹. And each edge of a polygon corresponds to a $d - 1$ dimensional face of the polyhedron and is described as an inequality.

The linear program in Coho is of the form:

$$\min(c^T x)$$

¹The full-dimensional polyhedron can be restore from its projections by back-projecting each into a prism in R^d and computing the intersection of those prisms. Sometimes this is an over-approximation but it's acceptable.

$$\begin{aligned} \text{s.t.} \quad & AEx \geq b \\ & x \text{ free} \end{aligned}$$

where E is an invertible matrix and A is a *Coho Matrix*²: there are one or two non-zero variables for each row of A . The feasible region of *Coho LP* is the intersection of set of back-projection of 2D subspace polygons described as above, and each constraint corresponds to an edge of a projection polygon. The special structure make it possible to obtain a solver with greater accuracy while efficient.

Coho Dual LP is the dual form of *Coho LP*:

$$\begin{aligned} & \min(-b^T y) \\ \text{s.t.} \quad & -A^T y = -E^{-T} c \\ & y \geq 0 \end{aligned}$$

By *Strong Duality Theorem*[2], *Coho LP* and *Coho Dual LP* have the same optimal costs and optimal basis. CohoSolver uses *Primal-Dual Method* to solve *Coho LP*.

3 Interval Computation

For any real computer, errors are inevitable because of floating point computations. The error may be accumulated and lead *Simplex* to an incorrect pivot. And also no error bounds are provided for floating point computation which prevents Coho from producing a guarantee of correctness for the system to be verified.

Interval Computation produces both the result and its error bound, which represents a real number \mathbf{x} by an interval $[\underline{x}, \bar{x}]$, with a high value \bar{x} and a low value \underline{x} , the real value is contained in the interval.

Let $\mathbf{x} = [\underline{x}, \bar{x}]$ and $\mathbf{y} = [\underline{y}, \bar{y}]$, the rule for interval computation is

$$\begin{aligned} \mathbf{x} + \mathbf{y} &= [\underline{x} + \underline{y}, \bar{x} + \bar{y}] \\ \mathbf{x} - \mathbf{y} &= [\underline{x} - \bar{y}, \bar{x} - \underline{y}] \\ \mathbf{x} \times \mathbf{y} &= [\min\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\}] \\ 1/\mathbf{x} &= [1/\bar{x}, 1/\underline{x}] \text{ if } \underline{x} > 0 \text{ or } \bar{x} < 0 \end{aligned}$$

However, even the high value and low value can't be represented exactly in computer. For a double $x = 2^a \times 1.B_1B_2 \dots B_t \dots$, computer rounds off and stores it as $\tilde{x} = 2^a \times 1.B_1B_2 \dots B_{t-1}B'_t$. Where the relative error $(x - \tilde{x})/x$ is less than 2^{-t} and the absolute error $x - \tilde{x}$ is less than $2^{-(t+a)}$. Therefore, after each step of computation, we should widen the interval a little from $[\tilde{x}, \bar{x}]$ to $[(1 - 2^{-t})\tilde{x}, (1 + 2^{-t})\bar{x}]$ to make sure it contains the real number x .

A package for interval number and interval matrix computation is implemented. We uses Java language, and the JVM applies the IEEE 754 standard, for which t is 52.

²In fact, it can transform to a simple form $Ay \geq b$ by replace x with $y = Ex$

4 CohoSolver

The classical *simplex* algorithm doesn't work for our application. First, we don't accept under-approximation optimal value; second, the error may be quite large when the LP is ill-conditioned; at last, the *linprog* function in matlab doesn't provide an optimal basis which are required for our application.

Exploiting this special structure of *Coho LP*, we develop a *CohoSolver* for this kind of problem. It returns an interval which contains the real optimal value, and a series of *possible* optimal basis. A possible optimal basis is the basis that are possible feasible for both *Coho LP* and *Coho Dual LP*.

CohoSolver firstly converts *Coho LP* to its dual form: *Coho Dual LP*. It's of the standard form. We can apply *Simplex* algorithm to it. To find a *clearly* feasible basis, we use the *Big M* method and add some columns to the *Coho Dual LP*, which is equivalent with adding some redundant constraint to the *Coho LP*.

Then we apply our linear system solver to calculate the *Tableau* for pivoting. This solver calculates the tableau column directly from the input data, therefore error propagation and accumulation over pivot is avoided.

Interval numbers are often incomparable, when pivoting doesn't have a clearly decision, *CohoSolver* will try both. It deals with cycling problem by keep a record of visited basis. When the optimal basis is ill-condition, the error thus the interval is too large to be useful. The *CohoSolver* will try to find a suboptimal result for *Coho Dual LP*, thus a little infeasible over-approximation for *Coho LP*

4.1 Big-M method for Initial Feasible Basis

Simplex requires an initial feasible basis, while finding a feasible basis is non-trivial for general LP. Marius[1] presents a simple method to find an invertible basis. Which find an invertible basis first, then construct a helper LP: the invertible basis is feasible for helper LP, and the optimal basis of helper LP is feasible for *Coho LP*.

However, it doesn't work anymore for interval computation. That's because sometimes intervals are incomparable, thus *CohoSolver* can only return a numerically *possible* optimal basis for the helper LP thus a *possible* feasible basis for the original *Coho LP*. If the basis is actually infeasible, then this algorithm fails.

Rather, we apply *Big M* method. For a *Coho Dual LP*:

$$\begin{aligned}
 & \min(c_1 y_1 + \dots + c_n y_n) \\
 \text{s.t.} \quad & \begin{bmatrix} a_{11} & \cdot & a_{1n} \\ \cdot & \cdot & \cdot \\ a_{m1} & \cdot & a_{mn} \end{bmatrix} \begin{bmatrix} y_1 \\ \cdot \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \cdot \\ b_m \end{bmatrix} \\
 & y_i \geq 0 \quad i = 1, \dots, n
 \end{aligned}$$

we add *m extra variables* z_1, \dots, z_m , and make them expensive by assigning a very large cost M to

driven these variables out. We append an identity matrix to the end of *Coho Dual Matrix* A ³.

$$\begin{aligned} & \min(c_1y_1 + \dots + c_ny_n + Mz_1 + \dots + Mz_m) \\ \text{s.t.} \quad & \begin{bmatrix} a_{11} & \cdot & a_{1n} & 1 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{m1} & \cdot & a_{mn} & 0 & \cdot & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ \cdot \\ y_n \\ z_1 \\ \cdot \\ z_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \cdot \\ b_m \end{bmatrix} \\ & y_i \geq 0 \quad i = 1, \dots, n \quad z_j \geq 0 \quad j = 1, \dots, m \end{aligned}$$

We can see this is also a *Coho Dual LP*, it has the same optimal value and optimal basis as the original one because the added variables z_1, \dots, z_m are expensive and will not appear in the optimal basis. This is equivalent with add some redundant constraints, similar with $x_i \leq M$, to the *Coho LP*. And it's obviously that the added *extra columns* are a *clearly feasible basis* for this helper LP.

Clearly, the *extra column* can't appear in the optimal basis. But sometimes, we can't driven these columns out no matter how big M is⁴. So in the *pivot* function, we should not introduce such columns; and even if the *relative cost* of column j is zero, we should try to bring it into basis, because although it can't reduce the cost, it may driven out the *extra columns* with the same cost. At last, we should remove all possible optimal basis which contains such undesirable columns.

4.2 Computation of Tableau

At each step, the *Simplex* algorithm makes pivoting decisions based on the *Tableau Matrix* for current basis \mathcal{B} :

$$T = B^{-1}[A|b] \quad \text{where } B = A_{\cdot, \mathcal{B}}$$

Generally, computing the *Tableau Matrix* from the input data for each pivoting would require $O(m^2(n-m))$ time. In practice, this expensive solution is replaced by computing each new tableau from the previous one with the cost $O(m(n-m))$. However, this algorithm accumulates error, which might cause the error unacceptable for Coho.

For Coho, we first apply the *Lazy tableau generation* policy: pivoting is to identify a non-basic column and replace it with a favorable column and reduce the cost. Once we have discover such a column, we don't need to know the higher-numbered column of the tableau.

As mentioned above, the computation of tableau columns from input data is undesirable, but for the Coho LP, there is a more efficient algorithm. This algorithm makes it possible to compute tableau directly from input data, thus avoid the problem of error accumulation. The algorithm to solve the linear system $Bx = b$, where B is a *Coho Dual Matrix* or a *Coho Matrix*⁵, is:

³Here we assume b_i are all nonnegative. If b_i is negative, we just set the i^{th} diagonal element as -1 to make the basis feasible

⁴For example, when some $b_i = 0$, then $z_i = 0$, the cost for its is zero and thus can't be driven out

⁵Although to compute tableau column, we only need to solve the case with *Coho Dual Matrix*, we need to solve the other case for other step of CohoSolver

1. If the i^{th} row of matrix B contains one non-zero element, which lies in column j , solve variable v_j and eliminate it. We removed i^{th} row and j^{th} column from B .
2. If a column j of matrix B contains one non-zero element, which lies in row i . Record and move the i^{th} row and j^{th} from B and solve v_j after the next step.
3. Now, there are at least two non-zero elements for each row and each column; and it's known that there are at most two non-zero elements for each row⁶ or each column⁷. Therefore, B has exactly two non-zero element for each row and each column. This is a *circulant matrix*. which can be solved linearly using the algorithm below.
4. Apply the variables solved to the rows from step 2, then we can solve variable v_j moved at step 2.

Now, the problem is to solve linear system where B is a circulant matrix. For a circulant matrix, it's easy⁸ to converted to the standard form:

$$\begin{bmatrix} 1 & -\alpha_1 & 0 & \dots & 0 \\ 0 & 1 & -\alpha_2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1 & -\alpha_{n-1} \\ -\alpha_n & 0 & \dots & 0 & 1 \end{bmatrix}$$

Let $P_k = \prod_{i=1}^k \alpha_k$, we can easily solve the linear system as:

$$\begin{aligned} x_1 &= \frac{\sum_{j=1}^n (b_j P_{j-1})}{1 - P_n} \\ x_{i+1} &= \frac{1 - \sum_{j=1}^i (b_j P_{j-1})}{P_i} \end{aligned} \tag{1}$$

4.3 Uncertainty and Cycling

Simplex is a greedy algorithm, during each step, it tries to replace one of current basic columns with a new columns in order to obtain a new feasible basis with lower cost. Comparisons are performed to decide which column should enter the basis and which column should leave the basis. However, when two interval number are not comparable⁹, it's not obviously which should leave or enter the basis. Deciding to take the wrong branch can make the algorithm fail:

1. If a wrong column is evicted from the basis, an infeasible basis is reached.
2. If a wrong column enters the basis, a slightly more costly basis is reached, then the algorithm may end up caught in a cycle.

⁶If B is Coho form

⁷If B is Coho dual form

⁸by permutation and normalization

⁹Their ranges are intersected

The solution to the problems from uncertainty is to try both possible branches. Clearly this could potentially lead to an explosion in the running time; in practice, however, the number of uncertain comparisons is expected to be quite low.

Under this policy, the optimal basis is never omitted. Thus, the optimal basis is always contained in the set of possible optimal bases. The optimal cost returned by the *CohoSolver* is the union of all cost intervals corresponding to possible optimal basis. The real optimal cost must lie in the result.

$$\min(c_i.lo()) \leq c_{opt} \leq \max(c_i.hi())$$

c_i is the cost corresponds to the possible optimal basis \mathcal{B}_i

But it may also lead to undesirable computation path. For the first kind of branch, if some basis are clearly infeasible, *CohoSolver* will drop the branch. If it's numerically possible feasible but actually little infeasible, the cost is close to the optimal value. This just widen the optimal cost interval a little and introduce a numerically possible optimal basis. For the cycling branches, we solve it by recording all bases visited. Once a cycling detected, we terminate this branch and find the possible optimal result from the history.

4.4 Use of Non-optimal Basis

When the optimal basis is highly ill-conditioned, the interval of optimal cost computed by the method so far might be unacceptably large. Under that case, a slightly primal-infeasible, dual-suboptimal, well-conditioned basis might be a better over-approximation of the optimal cost. This is achieved by dropping a column from the ill-conditioned basis matrix, which is equivalent with dropping a constraint of the *Coho LP*.

A basis is ill-conditioned if the condition number of that basis matrix is quite large. Marius[1] gives an estimation of condition number K_A for *circulant matrix* A as:

$$\begin{aligned} \beta &= \sqrt[n]{P_n} \\ s_k &= \frac{s_{k-1}\beta}{\alpha_{k-1}} \\ K_B &\leq \frac{1 + |\beta|}{|1 - |\beta||} \\ K_S &= \frac{\max(|s_i|)}{\min(|s_i|)} \\ K_A &\leq \begin{cases} \text{well conditioned} & \text{if } P_n \leq 0 \\ K_B K_S^2 & \text{if } P_n > 0 \end{cases} \end{aligned}$$

A square matrix is ill conditioned if at least one of its row/column is nearly a linear combination of the other rows/columns. The number of rows/columns that are nearly equal to linear combination of other rows/columns represents the degree of ill-conditioning. For the case of our application, basic columns in the *Coho Dual Matrix* represents inward halfspace normals in the primal. And at any feasible basis, the cost vector lies inside the cone generated by the basic inward halfspace normals.

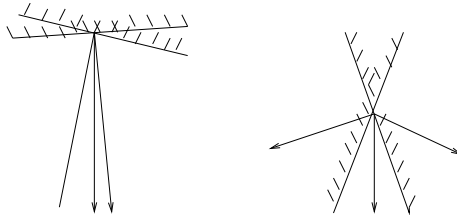


Figure 1: Highly obtuse optimal vertex and highly acute optimal vertex

Let's now consider a simple 2-dimensional example. From Figure 1 we see there are two cases: the optimal vertex is either highly obtuse or highly acute. For the first case, the vertices that lie on the lines that determine the optimal vertex are good approximation for the optimal value. And the more ill-conditioned the system is, the better the approximation is. For the second case, it's eliminated by other steps of Coho¹⁰.

Once an ill-conditioned basis found, we should decide which column to drop. In order to get a better approximation, we want to drop a constraint for *Coho LP* such that:

1. The number of ill-conditioning decreases. Otherwise, this basis is still ill-conditioned although a new constraint is introduced.
2. The approximation should not be far away from the optimal result.

The first step of our solution is to find all of the columns of basis matrix that are nearly linear combination of columns. Removing such columns will decrease the degree of ill-conditioning. The second step is to find the one with smallest error from those columns. There are several method for this problem. 1) The simplest way is try all the possibilities and use the one with smallest optimal

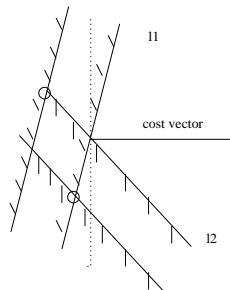


Figure 2: Suppose there are parallel line for l_1 and l_2 with same distance. We see removing l_2 produce smaller error.

¹⁰This ensures the Coho LP after throw a constraint is still bounded.

cost. Usually the number of such columns are small, so this is acceptable. 2) Another approximation algorithm is much faster, but the best choice is not guaranteed. It is based on the inspection (see Figure2) that the dropping a hyperplane that is nearly parallel with cost vector usually find a better approximation than dropping the one that are vertical with the cost vector. Therefore, we calculate the angle of cost vector and norm of the hyperplane, then drop the hyperplane with smallest angle. 3) Marius[1] also presents an algorithm using bounding box, which may find a better approximation for the optimal value, however, the information of optimal basis is lost. Therefore, we haven't implemented it.

5 Experience

One application of *CohoSolver* is to project a high-dimensional polyhedron to a 2D subspace. The basis idea is that for each direction c , we can find the right most point along the direction by solving a Coho LP $\min(-c^T x), st. Ax \geq b$. Then we turn around the direction anti clock-wise, we can find all the vertices of the projection polygon.

However, it's impossible to try all possible direction. Rather, using the property of basis feasibility, we find the norm for each edge of the polygon and thus find all vertices.

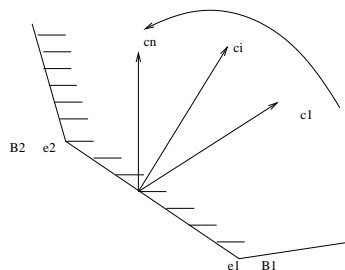


Figure 3: Turning round the direction over the norm, which will driven the optimal point to the other vertex of the edge

For example, in Figure3, $\{c_1, \dots, c_i, \dots, c_n\}$ is a set of directions, that across the norm c_i of an edge $[e_1, e_2]$. Their corresponding Coho LP are denoted as $\{CLP(c_1), \dots, CLP(c_i), \dots, CLP(c_n)\}$. It's easy to see that the optimal point for $\{CLP(c_1), \dots, CLP(c_{i-1})\}$ is e_1 and the optimal basis is \mathcal{B}_1 ; while optimal point for $\{CLP(c_{i+1}), \dots, CLP(c_n)\}$ is e_2 and optimal basis is \mathcal{B}_2 . Both e_1 and e_2 are optimal point for $CLP(c_i)$. So $CLP(c_i)$ is the turning point that optimal basis/point changes. While turning round the direction, the old basis \mathcal{B}_1 is driven out, it's no longer a optimal basis for the *Coho LP*, thus infeasible for *Coho Dual LP*. Therefore the norm of the edge is the minimum value that make at least one variable of *Coho Dual LP* negative.

6 Conclusion

This project implements a novel linear program solver for *Coho LP*. The solver provides optimal cost with error bound by applying interval computation. An efficient linear system solver is developed

for tableau computation, which eliminates the accumulation error. The solver also uses slightly infeasible result to replace the optimal value when the optimal basis is ill-conditioned.

References

- [1] Marius Laza, *A Robust Linear Program Solver for Projectahedra*, Master Thesis, University of British Columbia, 2001.
- [2] Robert J. Vanderbei, *Linear Programming: Foundations and Extensions*, Second Edition, 2001.
- [3] Mark R. Greenstreet, *Verifying Safety Properties of Differential Equations*, Proceedings of CAV'96, pages 277-287, 1996.
- [4] Mark R. Greenstreet, Xuemei Huang, *A Smooth Dynamical System that Counts in Binary*, Proceedings of ISCAS'97, pages 977-980, 1997.
- [5] Mark R. Greenstreet, Ian Mitchell, *Integration Projections*, Proceedings of HSCC'98, pages 159-174, 1998.
- [6] Mark R. Greenstreet, Ian Mitchell, *Reachability Analysis Using Polygonal Projections*, Proceedings of HSCC'99, pages 103-116, 1999.