Formalized Linear Programming

Chris Hughes

Abstract

A formally proven implementation of the simplex algorithm in the Lean theorem prover. The aim is to implement an efficiently computable and proven correct implementation of the simplex algorithm in a similar way to the implementation in the integer set library. Some additional functionality from the integer set library is also implemented in Lean.

1 Introduction

Describe motivation of project. The aim is to provide a practically computable implementation of the simplex algorithm (Coq version did not aim to do this). It differs in that we use a more complex tableau data type (dead columns and unrestricted rows). We also use a different pivot rule (Bland's Rule - lexicographic rule seems to be an ambiguous name).

Description of why Lean is a good language to do this in.

2 Linear Programming

The standard simplex algorithm. No reference to Lean code

The simplex algorithm aims to optimize the value of an objective function within the constraints of a set of affine inequalities. It can be used to solve problems of the following form. This is a slightly nonstandard way of writing a linear program. The usual way would be $Ax \leq b$. However the matrix that is stored during a simplex is $-A$ according to that representation, but A according to the representation below.

$$\begin{aligned} \text{Find } x \text{ that maximizes} \quad & c^T x \\ \text{subject to} \quad & Ax + b \geq 0 \\ \text{and} \quad & x \geq 0 \end{aligned} \tag{1}$$

Here, $x \in \mathbb{Q}^n$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, and $A \in \mathbb{Q}^{m \times n}$. This can be written in an equivalent form The inclusion of z in the objective function is slightly unusual because obviously maximizing $c^T x$ is the same as maximizing $c^T x + z$. However z is a cell in the tableau, it is the objective function value of the sample solution, and the explanation of the tableau is slightly easier with it

$$\begin{aligned} \text{Find } x \text{ and } y \text{ that maximize} \quad & c^T x + z \\ \text{subject to} \quad & y = Ax + b \\ \text{and} \quad & x \geq 0, y \geq 0 \end{aligned} \tag{2}$$

with $y \in \mathbb{Q}^m$ and $z \in \mathbb{Q}$

The subset of $\mathbb{Q}^{m+n}$, $\{(x,y) : y = Ax + b \wedge x \geq 0 \wedge y \geq 0\}$ is stored as part of the simplex algorithm. This problem can be written in tableau form.

$$\begin{array}{ccccccccccc}
y_1 & = & A_{11}x_1 & + & A_{12}x_2 & + & \dots & + & A_{1n}x_n & + & b_1 \\
y_2 & = & A_{21}x_1 & + & A_{22}x_2 & + & \dots & + & A_{2n}x_n & + & b_2 \\
\vdots & & \vdots & & \vdots & & & & \vdots & & \vdots \\
y_m & = & A_{m1}x_1 & + & A_{m2}x_2 & + & \dots & + & A_{mn}x_n & + & b_n \\
c^T x & = & c_1x_1 & + & c_2x_2 & + & \dots & + & c_nx_n & + & z
\end{array} \tag{3}$$

In this tableau representation the y_i are known as the *row* variables, and the x_j are the *column* variables. In a tableau the row variables are expressed as affine functions in terms of the column variables.

The simplex algorithm iteratively performs the *pivot* operation on a tableau. The pivot operation takes a row and a column variable as an argument, and swaps their positions in the tableau. If x_i and y_j are pivoted, then in the new tableau x_j becomes a row variable and y_i becomes a column variable. The variables $(y_1, y_2, \dots, x_i, \dots y_m)$ are expressed as affine functions of $(x_1, x_2, \dots, y_j, \dots, x_m)$. The matrix A, and the vectors c and b are updated to A', c' and b', such that the tableau represents the same subset of $\mathbb{Q}^{m+n}$, and the same optimization problem. In addition the row representing the objective function now has a potentially nonzero constant term, which we will call z. This is only possible if $A_{ij} \neq 0$.

$$\begin{array}{ccccccccccccccc}
y_1 & = & A'_{11}x_1 & + & A'_{12}x_2 & + & \dots & + & A'_{1j}y_i & + & \dots & + & A'_{1n}x_n & + & b'_1 \\
y_2 & = & A'_{21}x_1 & + & A'_{22}x_2 & + & \dots & + & A'_{2j}y_i & + & \dots & + & A'_{2n}x_n & + & b'_2 \\
\vdots & & \vdots & & \vdots & & & & \vdots & & & & \vdots & & \vdots \\
x_j & = & A'_{i1}x_1 & + & A'_{i2}x_2 & + & \dots & + & A'_{ij}y_i & + & \dots & + & A'_{in}x_n & + & b'_i \\
\vdots & & \vdots & & \vdots & & & & \vdots & & & & \vdots & & \vdots \\
y_m & = & A'_{m1}x_1 & + & A'_{m2}x_2 & + & \dots & + & A'_{mj}y_i & + & \dots & + & A'_{mn}x_n & + & b'_n \\
c^T x + z & = & c'_1x_1 & + & c'_2x_2 & + & \dots & + & c'_jy_i & + & \dots & + & c'_nx_n & + & z'
\end{array} \tag{4}$$

The expressions for the updated matrix A', and the updated vectors c', b' are given by the following formulas [4].

$$A'_{i'j'} = \begin{cases} A_{ij}^{-1} & \text{if } i' = i \text{ and } j' = j \\ -A_{ij'}/A_{ij} & \text{if } i' = i \text{ and } j' \neq j \\ A_{i'j}/A_{ij} & \text{if } i' \neq i \text{ and } j' = j \\ A_{i'j'} - A_{i'j}A_{ij'}/A_{ij} & \text{if } i' \neq i \text{ and } j' \neq j \end{cases} \tag{5}$$

$$b'_{i'} = \begin{cases} -b_i/A_{ij} & \text{if } i' = i \\ b_{i'} - A_{i'j}b_i/A_{ij} & \text{if } i' \neq i \end{cases} \tag{6}$$

$$c'_{j'} = \begin{cases} c_j/A_{ij} & \text{if } j' = j \\ c_{j'} - c_jA_{ij'}/A_{ij} & \text{if } j' \neq j \end{cases} \tag{7}$$

$$z' = z - c_jb_i/A_{ij} \tag{8}$$

Note that b, c and z are updated as though they are a continuation of A; b is updated in the same way as a column of A, and c is updated in the same way as a row of A.

It's nicest to talk about the tableau as $x_r = Ax_c + b$ with x_r and x_c the row and column minors of x. Do I stick to one ugly notation or change my convention in the middle?

Given any assignment of values to the column variables of a tableau, there is a unique x satisfying $x_r = Ax_c + b$ The *sample solution* of a tableau is the solution of $x_r = Ax_c + b$ found by setting all of the current column variables to zero. The row variables must then take the values given by the column b of the current tableau. Similarly, the objective function must have value z in the *sample solution*. A tableau is *feasible* if the sample solution also satisfies $x \geq 0$. This is equivalent to saying $b \geq 0$.

Given a feasible tableau, the simplex algorithm iteratively pivots whilst maintaining or increasing the objective function value of the sample solution z, as well as maintaining feasibility of the tableau. The chosen pivot row and pivot column i and j of a tableau always satisfy the following properties.

$$\begin{aligned} c_j &> 0 \\ A_{ij} &< 0 \\ \forall i', A_{i'j} < 0 \implies |b_i/A_{ij}| &\leq \left|b_{i'}/A_{i'j}\right| \end{aligned} \tag{9}$$

If the pivot row and pivot column satisfy these properties then using the expressions in equations (6) and (8), it can be shown the pivoted tableau is both feasible and that z has either increased or stayed the same.

The choice of pivot row and column satisfying (9) is non-unique, and must be chosen carefully in order to guarantee termination. Different implementations of the simplex algorithm use different pivot rules to guarantee termination. If z was strictly increasing it would straightforward to prove the algorithm terminates, since there are only finitely many partitions of the variables into row and column variables, and the simplex algorithm cannot repeat a partition if z is strictly increasing. Our implementation uses Bland's Rule [1].

The simplex algorithm terminates when there is no pivot row and column satisfying the conditions in (9).

If there is no column j satisfying $c_j > 0$, then the sample solution must maximize $c^T x_c$. This is because $c^T \leq 0$ and $x_c \geq 0$, so $c^T x_c \leq 0$, and the maximum value of this is given by setting x_c to zero.

Explanation below could be better

If there is a column j satisfying $c_j > 0$, but no row i satisfying $A_{ij} < 0$ then the objective function must be unbounded. The variable in column j is unbounded; for any $k > 0$, the solution found by setting the variable in column j to k and all other column variables to zero does not break the nonnegativity constraints of the row variables. The expression for the variable in row i is $A_{ij}k + b_i$, which is nonnegative. The objective function is $c_j k + z$, which is unbounded since k is unbounded.

Therefore, provided the simplex algorithm terminates,

3 Description of our version of the linear programming problem

Our version, dead columns, unrestricted variables, stopping early. Also very little reference to Lean code.

4 Description of Lean code and proofs

Only decribe Lean code where I had to do something in a way that was not the obvious way. The termination proof is interesting since the majority of the length of the proof is about stuff that was not mentioned on paper. Also 'pequiv' made a few things a little easier and is not obvious.

4.1 Our version of the linear programming problem

We have dead columns and restricted rows. Describe the tableau data type, and the partition data type. Also describe and explain the Lean definition of the problem i.e. sol_set etc. Aim to make it accessible to somebody who does not know about linear programming. Maybe include a section on our version of the linear programming problem with no reference to Lean code whatsoever, and then introduce Lean code later.

Some formalization papers (Rob Lewis - p-adic numbers) have a section on mathlib without reference to the p-adic numbers. I don't really think this is necessary, but maybe it is.

4.1.1 Tableau

The tableau data type represents a polyhedron as a system of affine equalities, a set of "restricted" variables that are constrained to be nonnegative, and a set of "dead" column variables constrained to be equal to zero. The representation is similar to the representation used in the Simplify theorem prover [3].

`tableau m n` represents a relation between $m + n$ rational variables, by partitioning the $m + n$ variables into a vector of m "row" variables and n "column" variables. This partition is represented as two vectors in Lean.

```
structure partition (m n : ℕ) : Type :=
(row_indices : vector (fin (m + n)) m)
(col_indices : vector (fin (m + n)) n)
(mem_row_indices_or_col_indices :
  ∀ v : fin (m + n), v ∈ row_indices.to_list ∨ v ∈ col_indices.to_list)
```

The type `fin (m + n)` is the type of nonnegative integers less than `m + n`. The `row_indices` and `column_indices` fields of the structure are vectors of length `m` and `n` respectively, containing elements of `fin (m + n)`. The final field of the structure stipulates that all element of `fin (m + n)` are an element of either `row_indices` or `col_indices`. This also implies, via a pigeonhole argument, that the two vectors are disjoint and have no duplicates. Given vectors r and c of row and column indices, it is possible to define the following minors of any vector $x \in \mathbb{Q}^{m+n}$.

$$\begin{aligned} x_r &:= (x_{r_0}, ..., x_{r_{m-1}}) \\ x_c &:= (x_{c_0}, ..., x_{c_{n-1}}) \end{aligned} \tag{10}$$

Given a `partition m n`, it is useful to define the functions taking a row or column index and returning the variable in that column or row.

```
def rowg : fin m → fin (m + n) := P.row_indices.nth
def colg : fin n → fin (m + n) := P.col_indices.nth
```

Listing 1: `rowg` and `colg` return the variable in a given column or row of a tableau

We also define the matrices corresponding to the linear maps taking the minors defined in equation 10. To do this for the row minor, we first define a `pequiv` between `fin (m + n)` and `fin m`. This is a representation of a bijection between a subset of `fin m` and a subset of `fin (m + n)`.

Should I go into detail on the `pequiv` data type and how I use it? It is referred to in the definition of `flat` and `of_col`, and consequently in the statements of the correctness proofs, but actually not used very much in the proofs, and turned out to be not as useful as I thought at first.

These "row" variables are then expressed as affine combinations of the "column" variables by the tableau. Given an $m \times n$ matrix A, and a constant vector $k \in \mathbb{Q}^m$, we can express the following relation between the current row variables x_r and column variables x_c.

$$x_r = Ax_c + k \tag{11}$$

The tableau data type in Lean, consists of an $m \times n$ matrix, the constant vector `const`, a set of restricted variables, constrained to be nonnegative, and the set of dead columns, constrained to be equal to zero. It also contains a `partition m n`, defining the current row and column variables.

```
structure tableau (m n : ℕ) extends partition m n :=
(to_matrix  : matrix (fin m) (fin n) ℚ)
(const      : cvec m)
(restricted : finset (fin (m + n)))
(dead       : finset (fin n))
```

The `matrix` datatype in Lean is implemented as a binary function from any pair of finite indexing types into a ring, in this case $\mathbb{Q}$. For this application the indexing types are `fin m` and `fin n`.

Given a tableau it is possible to define the polyhedron corresponding to the tableau. First the `flat` is defined, the affine subset of $\mathbb{Q}^{m+n}$ that satisfies the affine equalities.

```
def flat : set (cvec (m + n)) :=
{ x | T.to_partition.rowp.to_matrix ⬝ x =
  T.to_matrix ⬝ T.to_partition.colp.to_matrix ⬝ x + T.const }
```

`T.to_partition.rowp.to_matrix` is an $m \times (m + n)$ matrix, which corresponds to the linear map taking the minor x_r of x defined in (10). `T.to_partition.colp.to_matrix` does the same for the column variables. This equation is the same as the equation in (11). The notation $\cdot$ is used for matrix multiplication.

The other relevant sets are the `res_set`, the intersection of the flat and the set such that all restricted variables are nonnegative, and the `dead_set`, the intersection of the flat and the set such that the variables assigned to all dead columns are equal to zero. Finally the main object of study is the `sol_set`; the intersection of the `res_set` and the `dead_set`. In the definition of `sol_set`, `T.to_partition.colg j` returns the variable assigned to the column `j` in `T`.

```
def res_set : set (cvec (m + n)) := flat T ∩ { x | ∀ i, i ∈ T.restricted → 0 ≤ x i 0 }

def dead_set : set (cvec (m + n)) :=
flat T ∩ { x | ∀ j, j ∈ T.dead → x (T.to_partition.colg j) 0 = 0 }

def sol_set : set (cvec (m + n)) :=
res_set T ∩ { x | ∀ j, j ∈ T.dead → x (T.to_partition.colg j) 0 = 0 }
```

Given any assignment of values to the column variables of the tableau, there is a unique point in the `flat` such that the column variables have these values. The function `of_col` returns this point.

```
def of_col (T : tableau m n) (x : cvec n) : cvec (m + n) :=
T.to_partition.colp.to_matrixᵀ ⬝ x + T.to_partition.rowp.to_matrixᵀ ⬝ (T.to_matrix ⬝
    x + T.const)
```

A tableau `T` is said to be `feasible` if the sample point of the tableau, `of_col T 0` is in the `sol_set` of `T`, or equivalently, the constant column of the tableau is nonnegative in every row owned by a restricted variable.

```
def feasible (T : tableau m n) : Prop :=
∀ i, T.to_partition.rowg i ∈ T.restricted → 0 ≤ T.const i 0
```

4.2 Simplex

pivot_col and pivot_row function description. The type and behaviour of the simplex function. Statements of correctness proofs.

Probably the description of what the simplex algorithm should come before the description of the data types

The simplex implemented in Lean differs from a classical simplex in that there are both dead columns and unrestricted variables. These slightly complicate the proofs of correctness of the simple algorithm.

The simplex algorithm finds a point within a polyhedron that maximises a given objective variable. This objective variable must own a row in the starting tableau. If this variable is unbounded within the polyhedron, then it will return a proof of unboundedness. The simplex algorithm works by iteratively pivoting the tableau, whilst increasing the value of the objective variable in the sample point of the tableau. This objective value is the value in the constant column of the objective row. The starting tableau must be feasible, and every tableau visited by the algorithm will also be feasible.

The pivot operation takes a row index i and a column index j and moves the variable assigned to row i to column j, and the variables assigned to column j to row i, whilst updating the tableau to preserve the same polyhedron. It is possible to preserve the same polyhedron if the entry `T.to_matrix i j` of the tableau matrix is nonzero.

The simplex algorithm always chooses a pivot that maintains or increases the objective value. In order to do this the pivot column and row must satisfy certain properties.

The pivot column `c` is selected first. The pivot column must have the following property.

```
T.to_matrix obj c ≠ 0 ∧ T.to_partition.colg c ∈ T.restricted →
  0 < T.to_matrix obj c ≠ 0
```

If there is no column with this property, then the sample solution of the current tableau is optimal, and the simplex algorithm terminates.

The pivot row `r` must be chosen such that `T.to_matrix obj c / T.to_matrix r c < 0`. In order to maintain feasibility of the pivotted tableau, out of the rows with that property the algorithm chooses the row that minimises the value `T.to_matrix abs (T.const r 0 / T.to_matrix r c)` If no row with these properties can be found, the objective variable must be unbounded, and the simplex algorithm terminates.

As long as the objective value is strictly increasing, it is straightforward to prove that this algorithm terminates. However, in certain circumstances, the objective value does not increase after pivoting, and it is necessary to choose carefully the pivot row and column within the constraints above in order to ensure termination. There are many methods of doing this; the method implemented in Lean is Bland's rule [1]. This is described in section 4.3

For the maximisation problem, the simplex algorithm terminates only when it fails to find either a pivot row or column with the required properties. Sometimes it is only necessary to verify if the maximum value of a variable is positive. For this purpose a boolean "while" condition for early termination is added to the simplex function.

The simplex algorithm in Lean outputs the tableau it terminated on, as well as the reason for termination; either the objective variable is unbounded, the tableau sample solution is optimal, or the "while" condition was false. In the cases that the objective variable is unbounded, it also returns the pivot column that it found before it failed to find a pivot row. The simplex algorithm also requires a proof that the input tableau is feasible. This condition is necessary to prove termination.

```
def simplex (w : tableau m n → bool) (obj : fin m) : Π (T : tableau m n)
  (hT : feasible T), tableau m n × termination n
```

The simplex algorithm returns a `tableau m n` and an element of `termination n`, an inductive type specifying the conditions for termination, and also returning the final pivot column index in the case that the objective value is unbounded.

```
inductive termination (n : ℕ) : Type
| while {}            : termination
| unbounded (c : fin n) : termination
| optimal {}          : termination
```

4.3 Bland's Rule

How much detail to go into in this proof? This was one of the more challenging parts of the formalization, however, the proof is actually not very nicely written at the moment, for example the definition of fickle is not quite the same as in Chvatal. I used a weaker notion that seemed good enough,but then one variable that should be fickle ended up not being fickle, and I had to get round this in a slightly messy way.

The simplex implemented in Lean uses Bland's rule to ensure termination. If the pivot column is restricted then we choose the column owned by a variable with the smallest index out of the columns that satisfy the condition specified in [refer to earlier section]. We do the same for the rows that satisfy the specified condition.

In order to prove that this rule will terminate, it suffices to prove that the simplex does not repeat a tableau. This is because there are only finitely many tableaux that can be visited by the simplex algorithm, at most one for each of the finitely many partitions of the variables. Supposing that the simplex algorithm does repeat then there is a finite set of “fickle” variables that are pivoted during a cycle. By always choosing the variable with the smallest index, we know that the largest of the fickle variables was the unique fickle column variable satisfying [refer to earlier section] in some tableau in the cycle, and the unique fickle row variable in some other tableau in this cycle. It is possible to derive a contradiction from this, though the proof is omitted here [2].

In order to provide a proof of termination of the simplex algorithm in Lean, it is necessary to give a relation, a proof of well foundedness of this relation, and a proof that the sequence of tableaux accessed by the simplex algorithm is decreasing according to this relation. For this proof of termination, there is no natural choice of relation, so the relation is just defined using the pivot rule. Given tableaux T’ and T, rel obj T’ T is a relation saying that if the simplex algorithm visits T, then at some point after it will visit T’. It is defined inductively.

```
inductive rel : tableau m n → tableau m n → Prop
| pivot : ∀ {T}, feasible T → ∀ {r c}, c ∈ pivot_col T obj →
  r ∈ pivot_row T obj c → rel (T.pivot r c) T
| trans_pivot : ∀ {T₁ T₂ r c}, rel T₁ T₂ → c ∈ pivot_col T₁ obj →
  r ∈ pivot_row T₁ obj c → rel (T₁.pivot r c) T₂
```

Include description of pivot_col and pivot_row functions somewhere

Two tableau T and T’ are related if either T’ = T.pivot r c where r and c are the pivot row and column selected according to Bland’s rule, or if there exists another tableau T_1 such that rel obj T_1 T and T’ = T_1.pivot r c.

4.4 sign_of_max and whatever else we do

I have written add_row, sign_of_max, and assert_ge although there are no proofs about assert_ge. This part is lacking a little bit, these functions don’t really solve problems yet, they’re just part of functions I haven’t implemented that do solve problems.

5 Refined algorithm and performance

How fast is it after refinement? Compare with ISL etc. Talk about how it was done in a rubbish way, but could be done better with a refinement framework.

6 Conclusion

Follow up work. Difficulties I came across. It would have been nice to have a generic theory of linear programming. Some of what I wrote could be applied to other implementations of simplex, but not much.

References

[1] New finite pivoting rules for the simplex method. *Math. Oper. Res.*, 2(2):103–107, May 1977.

[2] Vašek Chvátal. *Linear programming.* A series of books in the mathematical science. Freeman, New York, 1983.

[3] David Detlefs, Greg Nelson, and James Saxe. Simplify: A theorem prover for program checking. *Journal of the ACM*, 52, 09 2003.

[4] Charles Gregory Nelson. *Techniques for Program Verification.* PhD thesis, Stanford, CA, USA, 1980. AAI8011683.