Linear programming made easy with Boost Proto

Patrick Mihelich

C++Now! 2012

What is a Program?

minimize

$$f(x)$$

subject to

$$g_1(x) \leq b_1$$
$$g_2(x) \leq b_2$$
$$\dots$$
$$g_m(x) \leq b_m$$

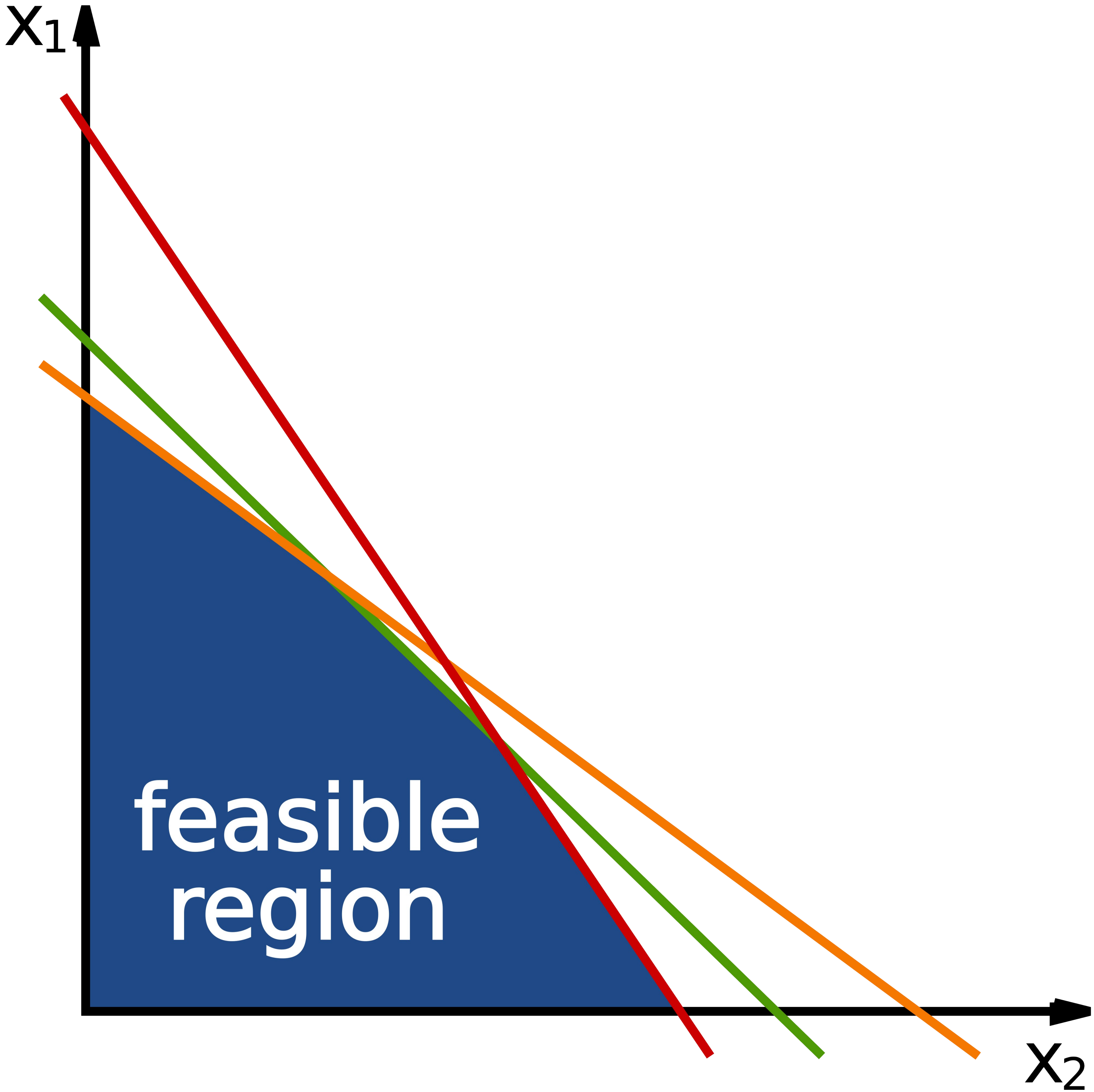
Linear Program

minimize

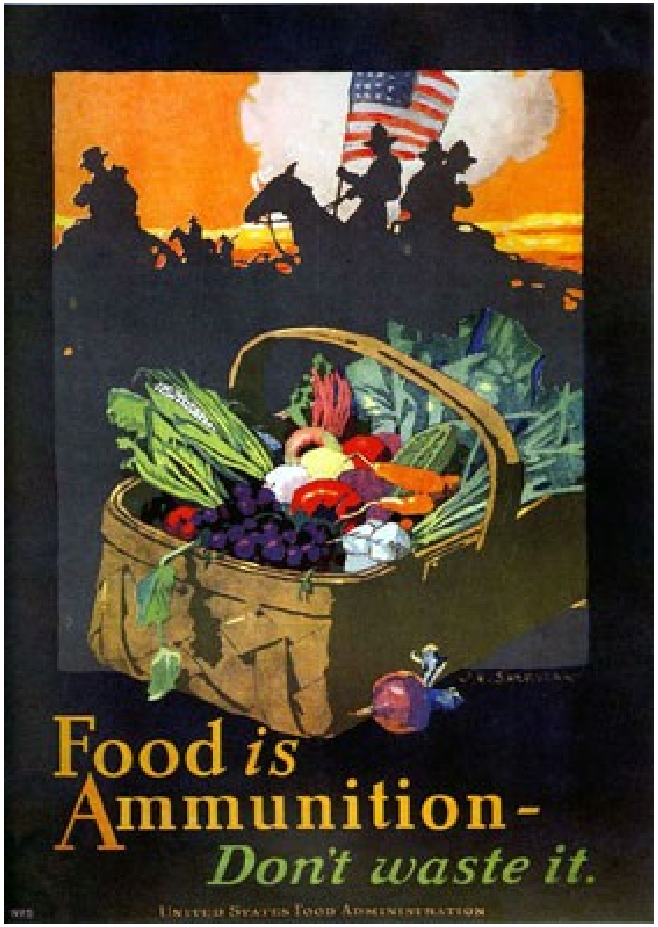
$$c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

subject to

$$a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq b_1$$
$$a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leq b_2$$
$$\ldots$$
$$a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m$$


x1
x2
feasible region

Diet Problem



Minimize cost

Subject to acceptable ranges for:

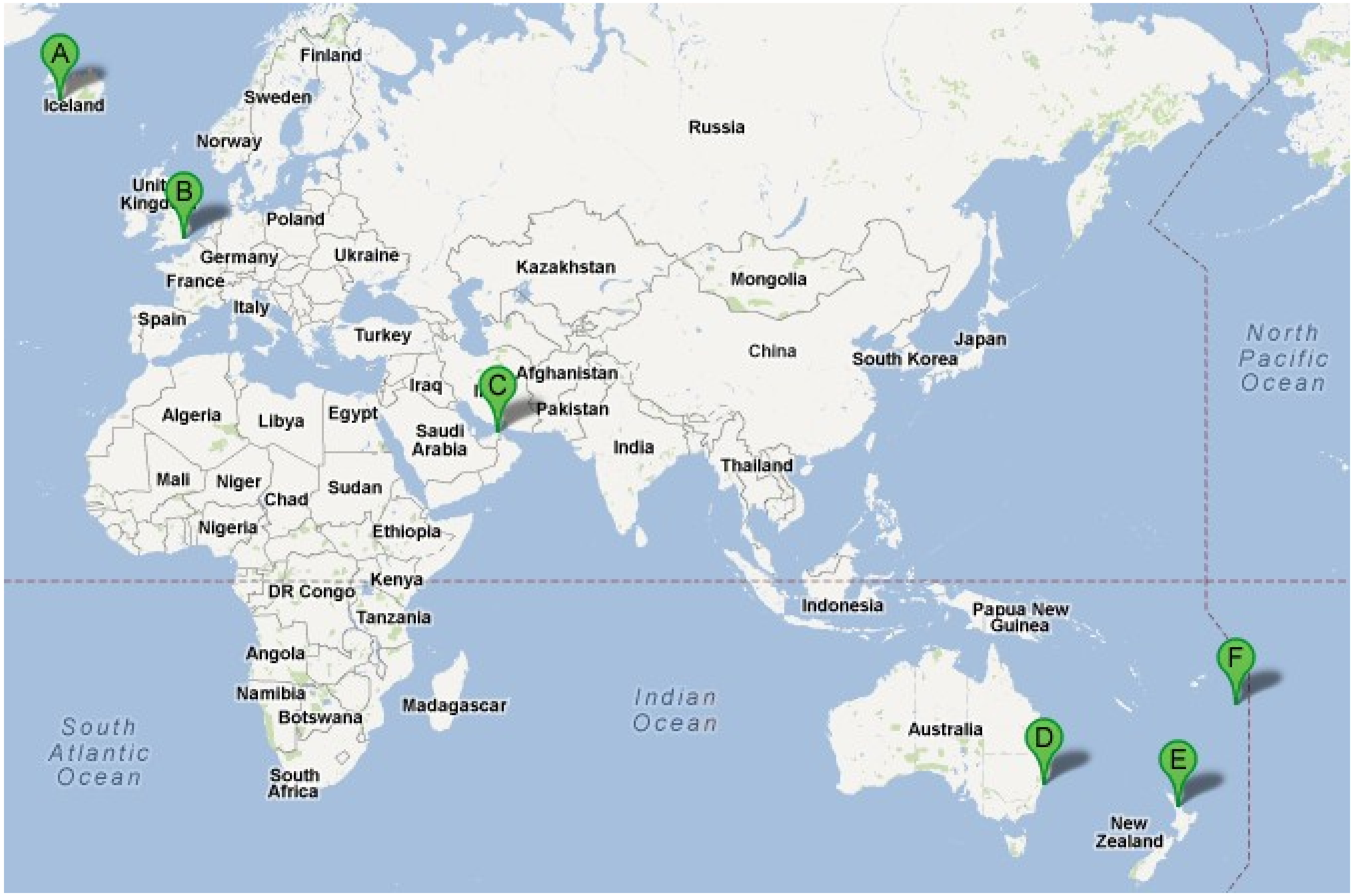
- Calories
- Vitamins
- Fats
- Sodium
- ...

Airline Ticket Pricing

Maximize profits

Subject to number of people willing to buy at each time and price

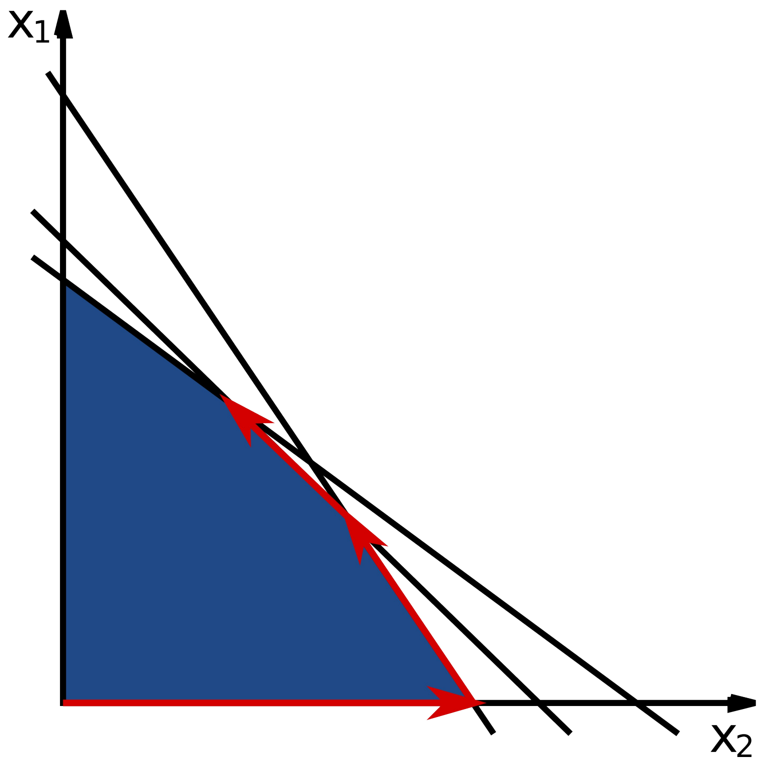
Selecting Flights

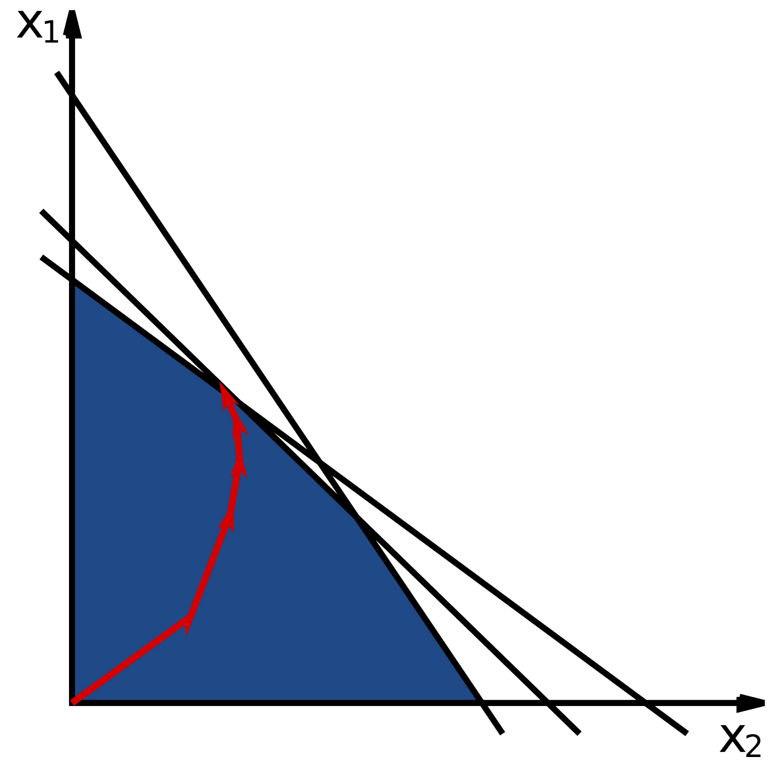


Minimize cost

Subject to network of available flights

Efficient Algorithms





Simplex method

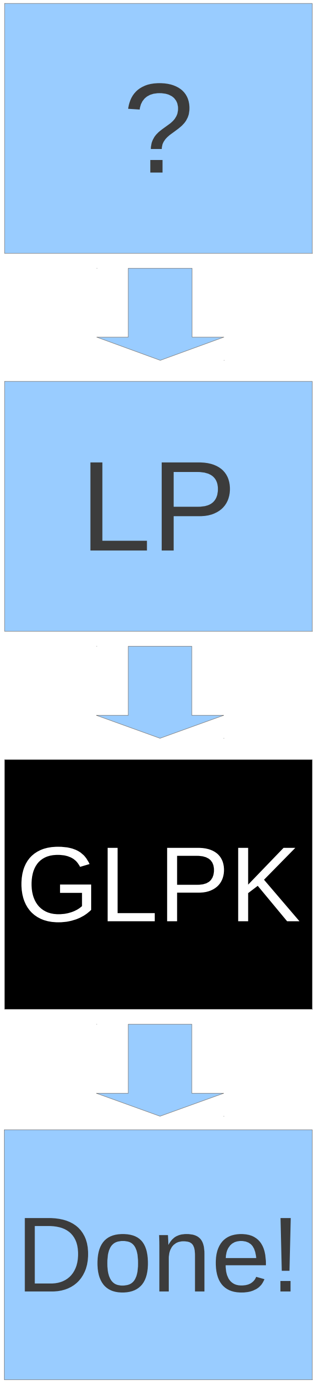
Fast in practice

Interior point methods

O(n^3) or better

Libraries

- Open source
  - GLPK
  - CLP
  - lp_solve
- Commercial
  - CPLEX
  - MOSEK
  - Excel Solver


?
LP
GLPK
Done!

Simple LP

maximize

$$10x_1 + 6x_2 + 4x_3$$

subject to

$$x_1 + x_2 + x_3 \leq 100$$
$$10x_1 + 4x_2 + 5x_3 \leq 600$$
$$2x_1 + 2x_2 + 6x_3 \leq 300$$

all non-negative:

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$$

Simple LP, GLPK C API

```
glp_prob *lp;
int ia[1+1000], ja[1+1000];
double ar[1+1000], z, x1, x2, x3;
lp = glp_create_prob();
glp_set_obj_dir(lp, GLP_MAX);
glp_add_rows(lp, 3);
glp_set_row_bnds(lp, 1, GLP_UP, 0.0, 100.0);
glp_set_row_bnds(lp, 2, GLP_UP, 0.0, 600.0);
glp_set_row_bnds(lp, 3, GLP_UP, 0.0, 300.0);
glp_add_cols(lp, 3);
glp_set_col_bnds(lp, 1, GLP_LO, 0.0, 0.0);
glp_set_obj_coef(lp, 1, 10.0);
glp_set_col_bnds(lp, 2, GLP_LO, 0.0, 0.0);
glp_set_obj_coef(lp, 2, 6.0);
glp_set_col_bnds(lp, 3, GLP_LO, 0.0, 0.0);
glp_set_obj_coef(lp, 3, 4.0);
```

```
ia[1] = 1, ja[1] = 1, ar[1] =  1.0;
ia[2] = 1, ja[2] = 2, ar[2] =  1.0;
ia[3] = 1, ja[3] = 3, ar[3] =  1.0;
ia[4] = 2, ja[4] = 1, ar[4] = 10.0;
ia[5] = 2, ja[5] = 2, ar[5] =  4.0;
ia[6] = 2, ja[6] = 3, ar[6] =  5.0;
ia[7] = 3, ja[7] = 1, ar[7] =  2.0;
ia[8] = 3, ja[8] = 2, ar[8] =  2.0;
ia[9] = 3, ja[9] = 3, ar[9] =  6.0;
glp_load_matrix(lp, 9, ia, ja, ar);
glp_simplex(lp, NULL);
z = glp_get_obj_val(lp);
x1 = glp_get_col_prim(lp, 1);
x2 = glp_get_col_prim(lp, 2);
x3 = glp_get_col_prim(lp, 3);
glp_delete_prob(lp);
```

Domain Specific Languages

- DSLs
  - Have limited expressiveness
  - Focus on particular domain
- DSL for LP should
  - Closely resemble mathematical notation
  - Be easy to understand, modify

External DSLs

- Standalone language with own parser
- Pros
  - Designer has full control
- Cons
  - Limited capabilities
  - Multilingualism has costs

Embedded DSLs

- Implemented within host language
- Pros
  - No extra parsing
  - Tight integration with GP language
- Cons
  - Limited by host language syntax

DSLs for Linear Programming

- External
  - AMPL
  - GAMS
  - CPLEX
- Embedded
  - CVX (Matlab)
  - CVXPY (Python)

Simple LP, CPLEX

```
Maximize
 obj: + 10 x1 + 6 x2 + 4 x3

Subject To
 p: + x3 + x2 + x1 <= 100
 q: + 5 x3 + 4 x2 + 10 x1 <= 600
 r: + 6 x3 + 2 x2 + 2 x1 <= 300

End
```

Simple LP, CVX

```
cvx_begin
  variables x1 x2 x3;
  maximize( 10*x1 + 6*x2 + 4*x3 );
  subject to
    x1 + x2 + x3 <= 100;
    10*x1 + 4*x2 + 5*x3 <= 600;
    2*x1 + 2*x2 + 6*x3 <= 300;
    x1 >= 0;
    x2 >= 0;
    x3 >= 0;
cvx_end
```

Can we do this in C++?

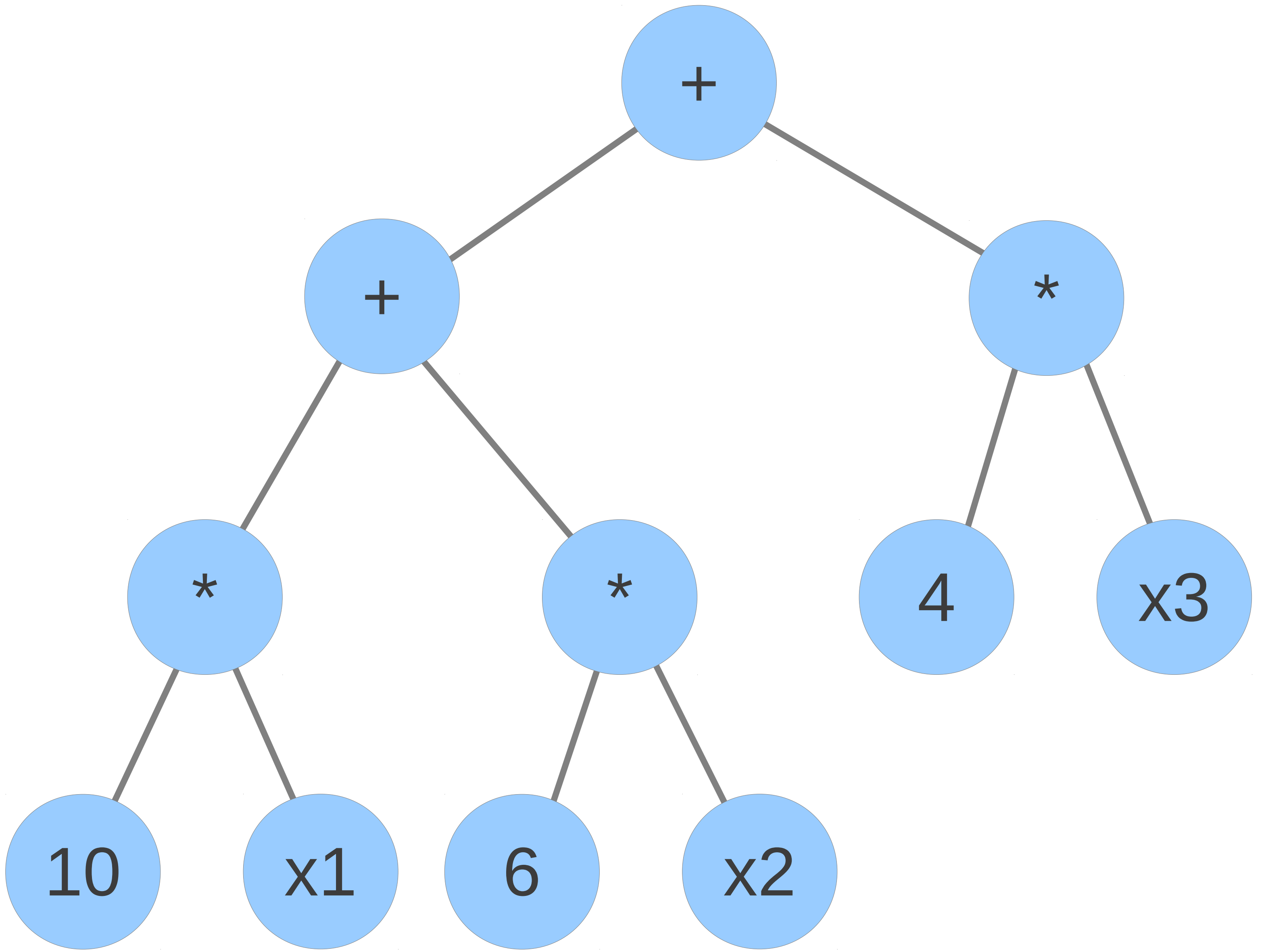
Simple LP, CVX++

```
Problem p;
CVX_VARIABLES( (x1)(x2)(x3) );

p.maximize( 10*x1 + 6*x2 + 4*x3 );

p.constrain( x1 + x2 + x3 <= 100 );
p.constrain( 10*x1 + 4*x2 + 5*x3 <= 600 );
p.constrain( 2*x1 + 2*x2 + 6*x3 <= 300 );
p.constrain( x1 >= 0 );
p.constrain( x2 >= 0 );
p.constrain( x3 >= 0 );

double z = p.solve();
```

Boost Proto

- “EDSL for defining EDSLs.”
- Build expression trees
- Check conformance to grammar
- Apply transformations
- Execute expressions

Expression Trees



Expression Trees

```
expr<
    tag::multiplies
  , list<
        expr<
            tag::terminal
          , term< int const & >
        >
      , expr<
            tag::terminal
          , term< Variable const & >
        >
    >
>
```

Grammar

```
AffineExpr → Constant
           → Variable
           → AffineExpr + AffineExpr
           → AffineExpr – AffineExpr
           → AffineExpr * Constant
           → Constant * AffineExpr
           → AffineExpr / Constant
           → -AffineExpr
```

Grammar

```
Constraint → AffineExpr == AffineExpr
           → AffineExpr <= AffineExpr
           → AffineExpr >= AffineExpr
```

Grammar in Proto

```
struct AffineExpr
  : or_<
        terminal< convertible_to< double > >
      , Variable
      , plus< AffineExpr, AffineExpr >
      , minus< AffineExpr, AffineExpr >
      , multiplies< AffineExpr, Constant >
      , multiplies< Constant, AffineExpr >
      , divides< AffineExpr, Constant >
      , negate< AffineExpr >
    >
{};
```

Grammar in Proto

```
struct Constraint
  : or_<
          equal_to< AffineExpr, AffineExpr >
        , less_equal< AffineExpr, AffineExpr >
        , greater_equal< AffineExpr, AffineExpr >
    >
{};
```

Validating the Grammar

```
template <typename Expr>
void minimize(const Expr& objective)
{
  BOOST_MPL_ASSERT(( boost::proto::matches< Expr, AffineExpr > ))
  ...
}

template <typename Expr>
void constrain(const Expr& constraint)
{
  BOOST_MPL_ASSERT(( boost::proto::matches< Expr, Constraint > ))
  ...
}
```

Linear Program

minimize

$$c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

subject to

$$a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq b_1$$
$$a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leq b_2$$
$$\ldots$$
$$a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m$$

Linear Program

minimize

$$c^T x$$

subject to

$$Ax \leq b$$

Simple LP

minimize

$$10\,x_1 + 6\,x_2 + 4\,x_3$$

subject to

$$1\,x_1 + 1\,x_2 + 1\,x_3 \leq 100$$
$$10\,x_1 + 4\,x_2 + 5\,x_3 \leq 600$$
$$2\,x_1 + 2\,x_2 + 6\,x_3 \leq 300$$

all non-negative:

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$$

Simple LP, GLPK C API

```
glp_prob *lp;
int ia[1+1000], ja[1+1000];
double ar[1+1000], z, x1, x2, x3;
lp = glp_create_prob();
glp_set_obj_dir(lp, GLP_MAX);
glp_add_rows(lp, 3);
glp_set_row_bnds(lp, 1, GLP_UP, 0.0, 100.0);
glp_set_row_bnds(lp, 2, GLP_UP, 0.0, 600.0);
glp_set_row_bnds(lp, 3, GLP_UP, 0.0, 300.0);
glp_add_cols(lp, 3);
glp_set_col_bnds(lp, 1, GLP_LO, 0.0, 0.0);
glp_set_obj_coef(lp, 1, 10.0);
glp_set_col_bnds(lp, 2, GLP_LO, 0.0, 0.0);
glp_set_obj_coef(lp, 2, 6.0);
glp_set_col_bnds(lp, 3, GLP_LO, 0.0, 0.0);
glp_set_obj_coef(lp, 3, 4.0);
```

```
ia[1] = 1, ja[1] = 1, ar[1] =  1.0;
ia[2] = 1, ja[2] = 2, ar[2] =  1.0;
ia[3] = 1, ja[3] = 3, ar[3] =  1.0;
ia[4] = 2, ja[4] = 1, ar[4] = 10.0;
ia[5] = 2, ja[5] = 2, ar[5] =  4.0;
ia[6] = 2, ja[6] = 3, ar[6] =  5.0;
ia[7] = 3, ja[7] = 1, ar[7] =  2.0;
ia[8] = 3, ja[8] = 2, ar[8] =  2.0;
ia[9] = 3, ja[9] = 3, ar[9] =  6.0;
glp_load_matrix(lp, 9, ia, ja, ar);
glp_simplex(lp, NULL);
z = glp_get_obj_val(lp);
x1 = glp_get_col_prim(lp, 1);
x2 = glp_get_col_prim(lp, 2);
x3 = glp_get_col_prim(lp, 3);
glp_delete_prob(lp);
```

Transforming to GLPK Format

- For each constraint:
  - Coefficient vector is one row of *A*
  - Scalar value is corresponding element of *b*

$$
\begin{aligned}
x_1 + x_2 + x_3 &\le 100 \\
10x_1 + 4x_2 + 5x_3 &\le 600 \\
2x_1 + 2x_2 + 6x_3 &\le 300
\end{aligned}
\qquad
A = \begin{pmatrix} 1 & 1 & 1 \\ 10 & 4 & 5 \\ 2 & 2 & 6 \end{pmatrix}, b = \begin{pmatrix} 100 \\ 600 \\ 300 \end{pmatrix}
$$

Computing Coefficient Vector

- Assign each variable an index
- Replace with a unit vector

$$10x_1 + 4x_2 + 5x_3$$

$$10\begin{pmatrix}1\\0\\0\end{pmatrix} + 4\begin{pmatrix}0\\1\\0\end{pmatrix} + 5\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}10\\4\\5\end{pmatrix}$$

Variables → Unit Vectors

```
typedef ublas::unit_vector<double> unit;
typedef terminal<unit>::type unit_terminal;

struct Coefficients
  : or_<
        when<
             Variable
           , unit_terminal(unit(N, VarId(_value)))
        >
      , when<
             less_equal< Constant, Coefficients >
           , minus< FreeConstant(_left),
                    Coefficients(_right)>(FreeConstant(_left),
                                          Coefficients(_right))
        >
    >
{}
```

Adding up Scalar Term

Multiply scalars by unit vector with index 0

```
struct FreeConstant
  : when<
         Constant
       , multiplies<_expr, unit_terminal>(
             _expr,
             unit_terminal(unit(N, mpl::int_<0>()))
         )
    >
{};
```

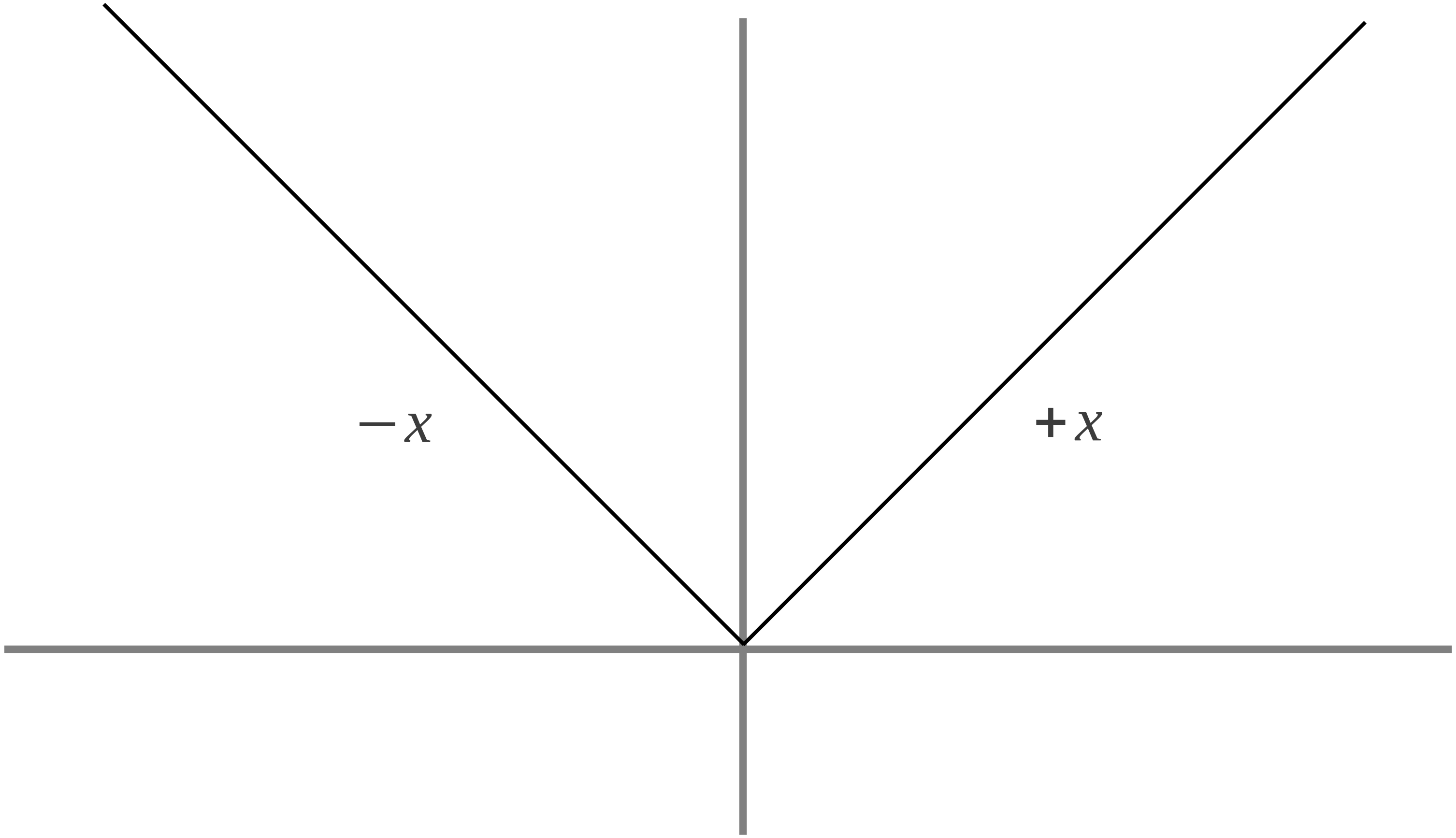
Pass to GLPK

```
template< typename Expr >
void constrain(const Expr& constraint) {
  // Convert constraint to sparse coefficient vector.
  CoefficientVector coeff = coefficients(constraint);
  // Extract the scalar component.
  double bound = -coeff[0];
  // Append new row to the GLPK problem object.
  int row = glp_add_rows(m_lp, 1);
  glp_set_mat_row(m_lp, row, coeff.nnz() - 1,
                  indices(coeff), values(coeff));
  // Set upper bound.
  glp_set_row_bnds(m_lp, row, GLP_UP, bound, bound);
}
```

Now Add Functions

- Absolute value $|x|$
- Minimum $min(x_{1,} x_2)$
- Maximum $max(x_{1,} x_2)$
- Manhattan norm $|x_1|+|x_2|$
- Chebyshev norm $max(|x_1|,|x_2|)$

Absolute value



Replace $|x|$ with new variable y and constrain

$$y \geq x$$
$$y \geq -x$$

Minimum

$$min(x_{1,} x_2) + x_3 \geq 42$$

$$y + x_3 \geq 42$$

$$y \leq x_1$$

$$y \leq x_2$$

Transforming Functions

- Proto transform replaces each function call with new variable
- Tag dispatches to transform which adds necessary constraints

Transforming abs(.)

```
struct AbsHelper
{
  typedef Variable result_type;

  explicit AbsHelper(Problem& p)
    : m_p(p) {}

  template< typename Expr >
  result_type operator()(Expr const& expr) const {
    Variable& aux = m_p.aux_variable();
    m_p.constrain(aux >= expr);
    m_p.constrain(aux >= -expr);
    return aux;
  }

  Problem& m_p;
};
```

Future Work

- Matrix variables
- Other solver backends
- Convex optimization

Questions?

bitbucket.org/mihelich/cvxpp