

# Building large-scale conic optimization models using MOSEK Fusion

Andrea Cassioli  
Erling D. Andersen

SIAM Optimization Meeting  
San Diego, May 18-22, 2014



- 1 Intro on MOSEK
- 2 Overview of the Fusion API
- 3 A step-by-step SOCP example
- 4 Case Study: Quadratic Multi-commodity Min-cost flow
- 5 Conclusion

# MOSEK

A package for large-scale (conic) optimization.



# MOSEK

A package for large-scale (conic) optimization.



Algorithms

# MOSEK

A package for large-scale (conic) optimization.



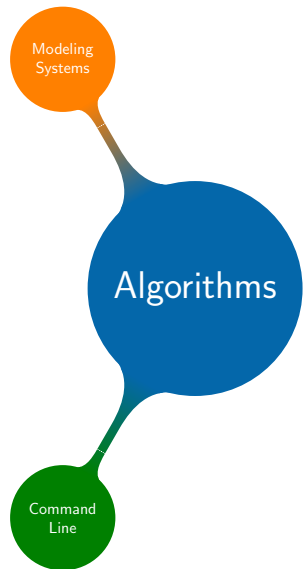
Algorithms

The diagram consists of two circles connected by a thin, curved line. The larger circle is blue and contains the word "Algorithms". The smaller circle is green and contains the words "Command Line".

Command  
Line

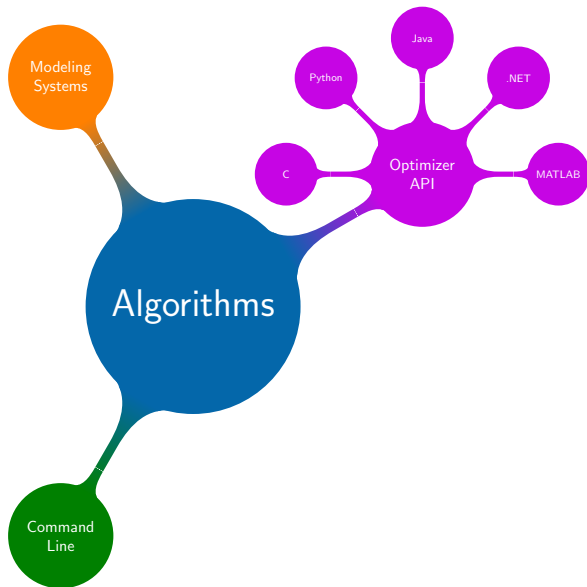
# MOSEK

A package for large-scale (conic) optimization.



# MOSEK

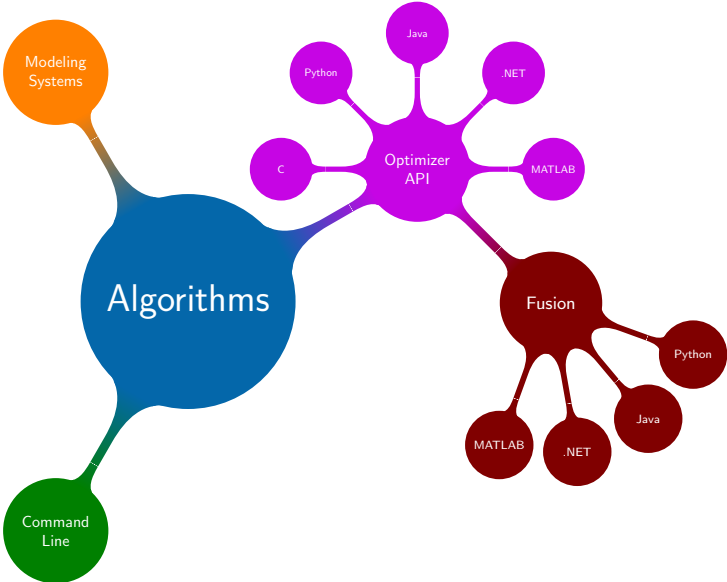
A package for large-scale (conic) optimization.



# MOSEK



A package for large-scale (conic) optimization.





# What is Fusion?

The general ideas.



A light-weight object-oriented API for linear conic optimization problems.

# What is Fusion?

The general ideas.



A light-weight object-oriented API for linear conic optimization problems.

- Minimal memory/time overhead
- Minimal model reformulation/modification
- Improve expressiveness

# How Fusion is designed

## Representing LCP



$$\min_{x \in \mathbb{R}^n} c^T x$$

*s.t.*

$$A_i x + b_i \in \mathcal{K}_i^{n_i} \in \{\mathbb{R}_+^{n_i}, \mathcal{Q}^{m_i}, \mathcal{Q}_r^{m_i}\} \quad i = 1, \dots, k$$

# How Fusion is designed

## Representing LCP



$$\min_{x \in \mathbb{R}^n} c^T x$$

*s.t.*

$$A_i x + b_i \in \mathcal{K}_i^{n_i} \in \{\mathbb{R}_+^{n_i}, \mathcal{Q}^{m_i}, \mathcal{Q}_r^{m_i}\} \quad i = 1, \dots, k$$

Basic constructs:

```
obj. fun. := ( linear expression , sense)
variable := ( dimension , domain)
constraint:= ( linear expression , domain)
```

# An SOCP Example

## Optimization Model



- interface to the solver
- memory manager
- creates its own components
- its components cannot be shared

# An SOCP Example

## Optimization Model



- interface to the solver
- memory manager
- creates its own components
- its components cannot be shared

```
import mosek
from mosek.fusion import *
```

```
M= Model('SOCP example')
```

# An SOCP Example

## Variables and Constraints



$$x^k \in \mathbb{R}_+^n \quad k = 1, \dots, m \quad \iff \quad X \in \mathbb{R}_+^{n \times m}$$

# An SOCP Example

## Variables and Constraints



$$x^k \in \mathbb{R}_+^n \quad k = 1, \dots, m \quad \iff \quad X \in \mathbb{R}_+^{n \times m}$$

```
X = M.variable(NDSet(n,m), Domain.greaterThan(0.))
```



# An SOCP Example

## Variables and Constraints



$$x^k \in \mathbb{R}_+^n \quad k = 1, \dots, m \quad \iff \quad X \in \mathbb{R}_+^{n \times m}$$

```
X = M.variable(NDSet(n,m), Domain.greaterThan(0.))
```

$$A^k x^k = b^k \quad k = 1, \dots, m$$

# An SOCP Example

## Variables and Constraints



$$x^k \in \mathbb{R}_+^n \quad k = 1, \dots, m \quad \iff \quad X \in \mathbb{R}_+^{n \times m}$$

```
X = M.variable(NDSet(n,m), Domain.greaterThan(0.))
```

$$A^k x^k = b^k \quad k = 1, \dots, m$$

```
for k in range(m):  
    M.constraint(Expr.mul(A[k], X.slice([0,k],[n,k+1])),\  
                Domain.equalsTo(b[k]))
```

# An SOCP Example

Coupling constraints



$$f_i = \sum d_i x_i^k \leq c_i \quad i = 1, \dots, n \iff \begin{aligned} f - Xd &= 0 \\ f &\leq c \end{aligned}$$

# An SOCP Example

## Coupling constraints



$$f_i = \sum d_i x_i^k \leq c_i \quad i = 1, \dots, n \iff \begin{aligned} f - Xd &= 0 \\ f &\leq c \end{aligned}$$

```
f=M.variable(n,Domain.lessThan(c))
```

```
M.constraint(Expr.sub(f,Expr.mul(X,d)),Domain.equalsTo(0.))
```

# An SOCP Example

Objective function



$$\text{minimize } \frac{1}{2} \sum (w_i f_i)^2$$

# An SOCP Example

Objective function



minimize  $t$

s.t.

$$(1, t, w * f) \in Q_r^{2+n}$$

# An SOCP Example



Objective function

minimize  $t$

s.t.

$$(1, t, w * f) \in Q_r^{2+n}$$

```
t= M.variable(1, Domain.unbounded())
```

```
M.constraint(Expr.vstack(Expr.constTerm(1.),t,Expr.mulElm(w,f)),\  
Domain.inRotatedQCone())
```

```
M.objective(ObjectiveSense.Minimize, t)
```

# An SOCP Example

## The whole model



```
M= Model('SOCP example')

#  $X \in \mathbb{R}_+^{n \times m}$ 
X = M.variable(NDSet(n,m), Domain.greaterThan(0.))

#  $k = 1, \dots, m$ 
for k in range(m):
    #  $A^k x^k = b^k$ 
    M.constraint(Expr.mul(A[k], x.slice([0,k],[n,k+1])), Domain.equalsTo(b[k]) )

#  $f \in \mathbb{R}^n, f \leq c$ 
M.variable(n,Domain.lessThan(c))

#  $f - X d = 0$ 
M.constraint( Expr.sub(f, Expr.mul(X,d) ), Domain.equalsTo(0.))

#  $t \in \mathbb{R}$ 
t= M.variable(1, Domain.unbounded())

#  $(1, t, w * f) \in \mathcal{Q}_r^{2+n}$ 
M.constraint(Expr.vstack(Expr.constTerm(1.),t,Expr.mulElm(w,f)),Domain.inRotatedQCone())

# min t
M.objective(ObjectiveSense.Minimize, t)
```



## Definition

Given a directed graph  $G$  ( $n_n$  nodes,  $n_a$  arcs,  $n_o$  commodities):

- $x^k, b^k$ : the flow and the demand of each commodity
- $A^k = A, \forall k = 1, \dots, n_o$ , with  $A$  incidence matrix of  $G$ , i.e.

$$AX = B \quad , \quad B = [b^1, b^2, \dots, b^{n_o}]$$

- $f$ : the total flow vector
- $c$ : arc capacities
- quadratic separable cost, i.e.

$$g(f) = \frac{1}{2} \sum (w_i f_i)^2$$

```

M= Model('Quadratic multi-commodity min-cost flow')

#  $X \in \mathbb{R}_+^{n_a \times n_o}$ 
X= M.variable(NDSet(na,no), Domain.greaterThan(0.))

# Exploting A, B sparsity
A= Matrix.sparse(nn, na, A_rows, A_cols, A_nnz)
B= Matrix.sparse(nn, no, B_rows, B_cols, B_nnz)

#  $AX = B$ 
M.constraint(Expr.mul(A, X), Domain.equalsTo(B) )

#  $0 \leq f \leq c$ 
f= M.variable(na, Domain.lessThan(c) )

#  $f - X I_{n_o} = 0$ 
ones= [1.0 for i in range(no)]
M.constraint(Expr.sub(f, Expr.mul(X, ones)), Domain.equalsTo(0.))

#  $t \in \mathbb{R}$ 
t= M.variable(1, Domain.unbounded())

#  $(1, t, \sqrt{w} * f) \in \mathcal{Q}_r^{2+n_a}$ 
M.constraint(Expr.vstack(Expr.constTerm(1.),t, Expr.mulElm(w,f), Domain.inRotatedQCone())

# min t
M.objective(ObjectiveSense.Minimize, t)

```

# Quadratic Multi-commodity Min-cost Flow Optimizer API(1)



```
with mosek.Env() as env:
  with env.Task(0,0) as task:
    nx= na*no

    indxf= nx
    nf= na
    indxt=indxf+nf
    nt=1
    indxs=indxt+nt
    ns=1

    numvar= nx+nf+nt+ns
    numcon= na+nn*no

    task.appendcons(numcon)
    task.appendvars(numvar)

    task.putcj( indxt, 1.0)

    #variable bounds setup

    task.putvarboundslice(0, nx, [mosek.boundkey.lo for i in range(nx)],\
      [0. for i in range(nx)], [0. for i in range(nx)])

    task.putvarboundslice(indxf, indxf+nf, [mosek.boundkey.ra for i in range(nf)],\
      [0. for i in range(nf)], cap)

    task.putvarboundslice(indxt, indxt+nt, [mosek.boundkey.lo for i in range(nt)],\
      [0. for i in range(nt)], [0. for i in range(nt)])

    task.putvarbound(indxs, mosek.boundkey.fx, 0.5, 0.5)
```

# Quadratic Multi-commodity Min-cost Flow Optimizer API(2)



```
for k in range(no):

    task.putaijlist( A[0], A[1], A[2] )
    A[0]=[ a+nn for a in A[0] ]
    A[1]=[ a+na for a in A[1] ]

    b=[0.0 for i in range(nn) ]
    for (i,j,v) in B:
        if j==k:
            b[i]= v

    task.putconboundslice(nn*k,nn*(k+1), [mosek.boundkey.fx for i in range(nn)], b,b)

    c_i=[]
    c_j=[]
    c_v=[]
    for i in range(na):
        for j in range(no):
            c_i.append(i+nn*no)
            c_j.append(j*na+i)
            c_v.append(1.0)
            c_i.append(i+nn*no)
            c_j.append(indxf+i)
            c_v.append(-1.0/sqrt(w[i]))

    task.putaijlist( c_i,c_j,c_v)
    task.putconboundslice(nn*no,numcon, [mosek.boundkey.fx for i in range(na)],\
        [0.0 for i in range(na)],\
            [0.0 for i in range(na)])

    task.appendcone( mosek.conetype.rquad,0., [indx,indxt]+ range(indxf,indxf+nf) )
```

# Some Computational Experiments



MCMF in urban traffic management

City	$n_n$	$n_a$	$n_o$	$F$	$O$
Sioux Falls	24	76	24	0.093	0.003
				0.059	0.041
Anaheim	416	914	38	0.210	0.041
				3.720	3.189
Barcelona	1020	2838	110	1.011	0.349
				266.073	251.037
Winnipeg	1052	2836	147	1.667	0.509
				198.321	157.036
Chicago S.	933	2950	387	14.057	2.915
				228.600	217.759

Dataset adapted from <http://www.bgu.ac.il/~bargera/tnp/>

# Conclusion

and future directions.



We show how Fusion

- introduces a (reasonable) overhead;
- can scale to large scale problems;
- leads to nice and clean code!

# Conclusion

and future directions.



We show how Fusion

- introduces a (reasonable) overhead;
- can scale to large scale problems;
- leads to nice and clean code!

What's ahead?

- C++ interface
- reduce the Fusion/Opt. gap
- more operations and syntactic sugar?

# Building large-scale conic optimization models using MOSEK Fusion

Andrea Cassioli  
Erling D. Andersen

SIAM Optimization Meeting  
San Diego, May 18-22, 2014

