SDPA : High performance package for SemiDefinite Programs

Makoto Yamashita[1], Katsuki Fujisawa[2], Mituhiro Fukuda[1], Kazuhiro Kobayashi[3], Masakazu Kojima[1], Kazuhide Nakata[1], Maho Nakata[4]

[1]Tokyo Institute of Technology, [2]Chuo University, [3]National Maritime Research Institute, [4]RIKEN

2011.04.15
@ National Cheng Kung University, Taiwan

SDPA

The fastest solver for large-scale SDPs.

- SDP (SemiDefinite Programs) has many applications.
- How large SDP can we solve?
- How fast can we solve it?

SDPA History & Computation time

Version	Year	mcp500-1	theta6	mater-4	
1.00	1995	–	–	–	Initial Version
2.01	1996	569.2	2643.5	62051.7	Mehrotra Type
3.20	1997	126.8	216.3	7605.9	Exploiting Sparsity
4.50	1998	53.6	217.6	29601.9	Full Ver. Exploiting Sparsity
5.01	1999	23.8	212.0	31258.1	Fast Step-Size
6.2.1	2002	1.6	20.7	746.7	BLAS/LAPACK
7.3.1	2009	1.5	14.2	10.4	Multi-Thread & Sparse Cholesky

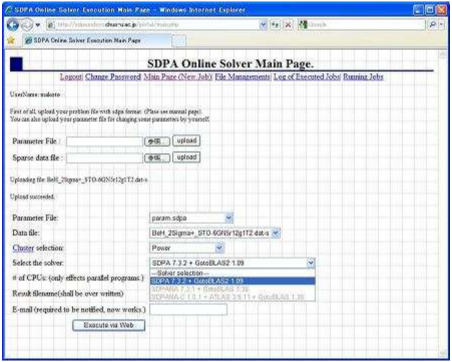
Time unit is *second*, Xeon 5550 (2.66GHz) x2, 72GB memory

The latest version is 7.3.4 (2011, March).

SDPA Online Solver

- SDPA is useful to solve SDPs.
- We prepare SDPA Online Solver.

1. Log-in the online solver
2. Upload your problem
3. Push 'Execute' button
4. Receive the result via Web/Mail



Outline

1. SDP Definition
2. Primal-Dual Interior-Point Methods
3. Inside of SDPA
4. SDPA Family (Parallel computation, Multiple precision, Online solver)

1. SemiDefinite Programs
   - Standard form
   - Applications
   - Theoretical property (Duality and Optimality)

SDP

Standard from

$$(\mathcal{P}) \quad : \quad \min : \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t.} \quad \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \ (k = 1, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

$$(\mathcal{D}) \quad : \quad \max : \sum_{k=1}^{m} b_k z_k \quad \text{s.t.} \quad \sum_{k=1}^{m} \boldsymbol{A}_k z_k + \boldsymbol{Y} = \boldsymbol{C}, \quad \boldsymbol{Y} \succeq \boldsymbol{O}$$

Notations:

$\boldsymbol{S}^n$	:	The space of $n \times n$ symmetric matrices
$\boldsymbol{S}^n_+ \subset \boldsymbol{S}^n$	:	The space of $n \times n$ positive semidefinite symmetric matrices
$\boldsymbol{X} \succeq \boldsymbol{O}$	:	$\boldsymbol{X} \in \boldsymbol{S}^n_+$, i.e., all the eigenvalues of $\boldsymbol{X}$ are non-negative
$\boldsymbol{X} \bullet \boldsymbol{Y}$	:	the inner-product between $\boldsymbol{X}$ and $\boldsymbol{Y}$, $\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} Y_{ij}$
m	:	The number of equality constraints
n	:	The size of variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$

SDP

Standard from

$$(\mathcal{P}) \quad : \quad \min : \boldsymbol{C} \bullet \boldsymbol{X} \qquad \text{s.t.} \quad \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \ (k = 1, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

$$(\mathcal{D}) \quad : \quad \max : \sum_{k=1}^{m} b_k z_k \qquad \text{s.t.} \quad \sum_{k=1}^{m} \boldsymbol{A}_k z_k + \boldsymbol{Y} = \boldsymbol{C}, \quad \boldsymbol{Y} \succeq \boldsymbol{O}$$

Notations:

$\boldsymbol{S}^n$	:	The space of $n \times n$ symmetric matrices
$\boldsymbol{S}^n_+ \subset \boldsymbol{S}^n$	:	The space of $n \times n$ positive semidefinite symmetric matrices
$\boldsymbol{X} \succeq \boldsymbol{O}$	:	$\boldsymbol{X} \in \boldsymbol{S}^n_+$, i.e., all the eigenvalues of $\boldsymbol{X}$ are non-negative
$\boldsymbol{X} \bullet \boldsymbol{Y}$	:	the inner-product between $\boldsymbol{X}$ and $\boldsymbol{Y}$, $\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} Y_{ij}$
m	:	The number of equality constraints
n	:	The size of variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$

Our target is $m \geq 30,000$.

Linear Matrix Inequality from Differential Equations (1 of 3)

Early SDP researches were strongly related to LMI from DE.
A simple example of DE w.r.t. $\boldsymbol{x}(t) = (x_1(t), x_2(t))^T \in \mathbb{R}^2$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t); \begin{pmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -3 & 1 \\ 1 & -3 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

Linear Matrix Inequality from Differential Equations (1 of 3)

Early SDP researches were strongly related to LMI from DE.
A simple example of DE w.r.t. $\boldsymbol{x}(t) = (x_1(t), x_2(t))^T \in \mathbb{R}^2$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t); \begin{pmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -3 & 1 \\ 1 & -3 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

We want to know *a stability condition*; whether $||\boldsymbol{x}(t)|| \to 0$ when $t \to \infty$.

Linear Matrix Inequality from Differential Equations (1 of 3)

Early SDP researches were strongly related to LMI from DE.
A simple example of DE w.r.t. $\boldsymbol{x}(t) = (x_1(t), x_2(t))^T \in \mathbb{R}^2$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t); \begin{pmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -3 & 1 \\ 1 & -3 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

We want to know *a stability condition*; whether $||\boldsymbol{x}(t)|| \to 0$ when $t \to \infty$.
In this case, YES.

Linear Matrix Inequality from Differential Equations (1 of 3)

Early SDP researches were strongly related to LMI from DE.
A simple example of DE w.r.t. $\boldsymbol{x}(t) = (x_1(t), x_2(t))^T \in \mathbb{R}^2$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t); \begin{pmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -3 & 1 \\ 1 & -3 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

We want to know *a stability condition*; whether $||\boldsymbol{x}(t)|| \to 0$ when $t \to \infty$.
In this case, YES.

First, apply the eigenvalue decomposition to the coefficient matrix.

$$\begin{array}{ccc} \boldsymbol{A} & = & \boldsymbol{PDP}^T \qquad (\boldsymbol{P} : \text{orthogonal} \boldsymbol{PP}^T = \boldsymbol{I}, \boldsymbol{D} : \text{diagonal}) \\ \begin{pmatrix} -3 & 1 \\ 1 & -3 \end{pmatrix} & = & \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}^T \end{array}$$

LMI from DE 2 of 3

We introduce the vector $\widetilde{\boldsymbol{x}} = \boldsymbol{P}^T \boldsymbol{x}$, then, from $\boldsymbol{P}^T \boldsymbol{P} = \boldsymbol{I}$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t) = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T \boldsymbol{x}(t) \Rightarrow \boldsymbol{P}^T \frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{D}\boldsymbol{P}^T \boldsymbol{x}(t)$$

LMI from DE 2 of 3

We introduce the vector $\widetilde{\boldsymbol{x}} = \boldsymbol{P}^T \boldsymbol{x}$, then, from $\boldsymbol{P}^T \boldsymbol{P} = \boldsymbol{I}$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t) = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T \boldsymbol{x}(t) \Rightarrow \boldsymbol{P}^T \frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{D}\boldsymbol{P}^T \boldsymbol{x}(t)$$

$$\Rightarrow \quad \frac{d\widetilde{\boldsymbol{x}}(t)}{dt} = \boldsymbol{D}\widetilde{\boldsymbol{x}}(t)$$

LMI from DE 2 of 3

We introduce the vector $\widetilde{\boldsymbol{x}} = \boldsymbol{P}^T\boldsymbol{x}$, then, from $\boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t) = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t) \Rightarrow \boldsymbol{P}^T\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t)$$

$$\Rightarrow \quad \frac{d\widetilde{\boldsymbol{x}}(t)}{dt} = \boldsymbol{D}\widetilde{\boldsymbol{x}}(t) \Rightarrow \begin{pmatrix} \frac{d\widetilde{x}_1(t)}{dt} \\ \frac{d\widetilde{x}_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \begin{pmatrix} \widetilde{x}_1(t) \\ \widetilde{x}_2(t) \end{pmatrix}$$

LMI from DE 2 of 3

We introduce the vector $\widetilde{\boldsymbol{x}} = \boldsymbol{P}^T \boldsymbol{x}$, then, from $\boldsymbol{P}^T \boldsymbol{P} = \boldsymbol{I}$,

$$
\begin{aligned}
& \frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t) = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t) \Rightarrow \boldsymbol{P}^T\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t) \\
\Rightarrow \quad & \frac{d\widetilde{\boldsymbol{x}}(t)}{dt} = \boldsymbol{D}\widetilde{\boldsymbol{x}}(t) \Rightarrow \begin{pmatrix} \frac{d\widetilde{x}_1(t)}{dt} \\ \frac{d\widetilde{x}_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \begin{pmatrix} \widetilde{x}_1(t) \\ \widetilde{x}_2(t) \end{pmatrix} \\
\Rightarrow \quad & \frac{d\widetilde{x}_1(t)}{dt} = -2\widetilde{x}_1(t), \quad \frac{d\widetilde{x}_2(t)}{dt} = -4\widetilde{x}_2(t) \\
\Rightarrow \quad & \widetilde{x}_1(t) = \widetilde{x}_1(0)e^{-2t}, \quad \widetilde{x}_2(t) = \widetilde{x}_2(0)e^{-4t} \\
\Rightarrow \quad & \lim_{t\to\infty} \widetilde{\boldsymbol{x}}(t) = 0
\end{aligned}
$$

LMI from DE 2 of 3

We introduce the vector $\widetilde{\boldsymbol{x}} = \boldsymbol{P}^T\boldsymbol{x}$, then, from $\boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}$,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t) = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t) \Rightarrow \boldsymbol{P}^T\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t)$$

$$\Rightarrow \quad \frac{d\widetilde{\boldsymbol{x}}(t)}{dt} = \boldsymbol{D}\widetilde{\boldsymbol{x}}(t) \Rightarrow \begin{pmatrix} \frac{d\widetilde{x}_1(t)}{dt} \\ \frac{d\widetilde{x}_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \begin{pmatrix} \widetilde{x}_1(t) \\ \widetilde{x}_2(t) \end{pmatrix}$$

$$\Rightarrow \quad \frac{d\widetilde{x}_1(t)}{dt} = -2\widetilde{x}_1(t), \quad \frac{d\widetilde{x}_2(t)}{dt} = -4\widetilde{x}_2(t)$$

$$\Rightarrow \quad \widetilde{x}_1(t) = \widetilde{x}_1(0)e^{-2t}, \quad \widetilde{x}_2(t) = \widetilde{x}_2(0)e^{-4t}$$

$$\Rightarrow \quad \lim_{t\to\infty} \widetilde{\boldsymbol{x}}(t) = 0$$

$$\Rightarrow \quad \boldsymbol{x} = \boldsymbol{P}\boldsymbol{P}^T\boldsymbol{x} = \boldsymbol{P}\widetilde{\boldsymbol{x}}, \quad \text{hence} \quad \lim_{t\to\infty} \boldsymbol{x}(t) = 0$$

LMI from DE 2 of 3

We introduce the vector $\widetilde{\boldsymbol{x}} = \boldsymbol{P}^T\boldsymbol{x}$, then, from $\boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}$,

$$
\begin{aligned}
& \frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}\boldsymbol{x}(t) = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t) \Rightarrow \boldsymbol{P}^T\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{D}\boldsymbol{P}^T\boldsymbol{x}(t) \\
\Rightarrow \quad & \frac{d\widetilde{\boldsymbol{x}}(t)}{dt} = \boldsymbol{D}\widetilde{\boldsymbol{x}}(t) \Rightarrow \begin{pmatrix} \frac{d\widetilde{x}_1(t)}{dt} \\ \frac{d\widetilde{x}_2(t)}{dt} \end{pmatrix} = \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \begin{pmatrix} \widetilde{x}_1(t) \\ \widetilde{x}_2(t) \end{pmatrix} \\
\Rightarrow \quad & \frac{d\widetilde{x}_1(t)}{dt} = -2\widetilde{x}_1(t), \quad \frac{d\widetilde{x}_2(t)}{dt} = -4\widetilde{x}_2(t) \\
\Rightarrow \quad & \widetilde{x}_1(t) = \widetilde{x}_1(0)e^{-2t}, \quad \widetilde{x}_2(t) = \widetilde{x}_2(0)e^{-4t} \\
\Rightarrow \quad & \lim_{t\to\infty} \widetilde{\boldsymbol{x}}(t) = 0 \\
\Rightarrow \quad & \boldsymbol{x} = \boldsymbol{P}\boldsymbol{P}^T\boldsymbol{x} = \boldsymbol{P}\widetilde{\boldsymbol{x}}, \quad \text{hence} \quad \lim_{t\to\infty} \boldsymbol{x}(t) = 0
\end{aligned}
$$

The point is $\lim_{t\to\infty} \boldsymbol{x}(t) = 0$ if all the eigenvalues of $\boldsymbol{A}$ is negative.

LMI from DE 3 of 3

The point is $\lim_{t\to\infty} \boldsymbol{x}(t) = 0$ if all the eigenvalues of $\boldsymbol{A}$ is negative.

LMI from DE 3 of 3

The point is $\lim_{t\to\infty} \boldsymbol{x}(t) = 0$ if all the eigenvalues of $\boldsymbol{A}$ is negative.
$\Leftrightarrow \quad \lambda^* < 0$, where $\lambda^* = \min \lambda : \lambda \boldsymbol{I} - \boldsymbol{A} \succeq \boldsymbol{O}$
For example,

$$\min \lambda : \begin{pmatrix} \lambda & \\ & \lambda \end{pmatrix} - \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \succeq \boldsymbol{O} \Leftrightarrow \begin{cases} \lambda + 2 \geq 0 \\ \lambda + 4 \geq 0 \end{cases} \Leftrightarrow \lambda^* = -2 < 0$$

LMI from DE 3 of 3

The point is $\lim_{t\to\infty} \boldsymbol{x}(t) = 0$ if all the eigenvalues of $\boldsymbol{A}$ is negative.
$\Leftrightarrow \quad \lambda^* < 0,$ where $\lambda^* = \min \lambda : \lambda \boldsymbol{I} - \boldsymbol{A} \succeq \boldsymbol{O}$
For example,

$$\min \lambda : \begin{pmatrix} \lambda & \\ & \lambda \end{pmatrix} - \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \succeq \boldsymbol{O} \Leftrightarrow \begin{cases} \lambda + 2 \geq 0 \\ \lambda + 4 \geq 0 \end{cases} \Leftrightarrow \lambda^* = -2 < 0$$

In some applications like control theory, $\boldsymbol{A}(\boldsymbol{z}) = \sum_{k=1}^{m} A_k z_k$; a linear combination of $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ with variables $z_1, \ldots, z_m$.
We want to know for $\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}(\boldsymbol{z})\boldsymbol{x}(t)$, whether $||\boldsymbol{x}(t)|| \to 0$ when $t \to \infty$.

LMI from DE 3 of 3

The point is $\lim_{t\to\infty} \boldsymbol{x}(t) = 0$ if all the eigenvalues of $\boldsymbol{A}$ is negative.
$\Leftrightarrow \quad \lambda^* < 0$, where $\lambda^* = \min \lambda : \lambda \boldsymbol{I} - \boldsymbol{A} \succeq \boldsymbol{O}$
For example,

$$\min \lambda : \begin{pmatrix} \lambda & \\ & \lambda \end{pmatrix} - \begin{pmatrix} -2 & \\ & -4 \end{pmatrix} \succeq \boldsymbol{O} \Leftrightarrow \begin{cases} \lambda + 2 \geq 0 \\ \lambda + 4 \geq 0 \end{cases} \Leftrightarrow \lambda^* = -2 < 0$$

In some applications like control theory, $\boldsymbol{A}(\boldsymbol{z}) = \sum_{k=1}^{m} A_k z_k$; a linear combination of $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ with variables $z_1, \ldots, z_m$.
We want to know for $\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{A}(\boldsymbol{z})\boldsymbol{x}(t)$, whether $||\boldsymbol{x}(t)|| \to 0$ when $t \to \infty$.
We solve an SDP and check λ^*.

$$\min \lambda : \lambda \boldsymbol{I} - \sum_{k=1}^{m} \boldsymbol{A}_k z_k \succeq \boldsymbol{O}$$

SDP Applications

- Control theory (LMI from DE)
- Combinatorial optimization (Max-cut problems, theta functions)
- Quadratic assignment problem
- Sensor network localization problem
- Polynomial optimization problem
- Quantum chemistry
- Statistics (Multi dimensional unfolding)
- Machine Learning

The applications generate extremely large-scale SDPs.

Theoretical Aspects

- Weak duality
- Strong duality
- KKT condition (Optimality Condition)

Theoretical Aspects

- Weak duality
- Strong duality
- KKT condition (Optimality Condition)

Lemma: positive semidefinite matrices

$$\boldsymbol{X}, \boldsymbol{Y} \succeq \boldsymbol{O} \Rightarrow \boldsymbol{X} \bullet \boldsymbol{Y} \geq O$$

Theoretical Aspects

- Weak duality
- Strong duality
- KKT condition (Optimality Condition)

Lemma: positive semidefinite matrices

$$\boldsymbol{X}, \boldsymbol{Y} \succeq \boldsymbol{O} \Rightarrow \boldsymbol{X} \bullet \boldsymbol{Y} \geq O$$

Proof: $\boldsymbol{X} \succeq \boldsymbol{O}$ has the eigenvalue decomposition with eigenvalues $\lambda_1, \ldots, \lambda_n \geq 0$.

$$\boldsymbol{X} = \sum_{i=1}^{n} \lambda_i \boldsymbol{p}_i \boldsymbol{p}_i^T$$

$$\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} = 4 \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} + 2 \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

Theoretical Aspects

- Weak duality
- Strong duality
- KKT condition (Optimality Condition)

Lemma: positive semidefinite matrices

$$\boldsymbol{X}, \boldsymbol{Y} \succeq \boldsymbol{O} \Rightarrow \boldsymbol{X} \bullet \boldsymbol{Y} \geq O$$

Proof: $\boldsymbol{X} \succeq \boldsymbol{O}$ has the eigenvalue decomposition with eigenvalues $\lambda_1, \ldots, \lambda_n \geq 0$.

$$\boldsymbol{X} = \sum_{i=1}^{n} \lambda_i \boldsymbol{p}_i \boldsymbol{p}_i^T$$

$$\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} = 4 \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} + 2 \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

$\boldsymbol{Y} \succeq \boldsymbol{O}$ also has $\boldsymbol{Y} = \sum_{j=1}^{n} \mu_j \boldsymbol{q}_j \boldsymbol{q}_j^T$ $(\mu_1, \ldots, \mu_n \geq 0)$.

Theoretical Aspects

- Weak duality
- Strong duality
- KKT condition (Optimality Condition)

Lemma: positive semidefinite matrices

$$\boldsymbol{X}, \boldsymbol{Y} \succeq \boldsymbol{O} \Rightarrow \boldsymbol{X} \bullet \boldsymbol{Y} \geq O$$

Proof: $\boldsymbol{X} \succeq \boldsymbol{O}$ has the eigenvalue decomposition with eigenvalues $\lambda_1, \ldots, \lambda_n \geq 0$.

$$\boldsymbol{X} = \sum_{i=1}^{n} \lambda_i \boldsymbol{p}_i \boldsymbol{p}_i^T$$

$$\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} = 4 \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} + 2 \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

$\boldsymbol{Y} \succeq \boldsymbol{O}$ also has $\boldsymbol{Y} = \sum_{j=1}^{n} \mu_j \boldsymbol{q}_j \boldsymbol{q}_j^T$ $(\mu_1, \ldots, \mu_n \geq 0)$. Hence,

$$\begin{aligned} \boldsymbol{X} \bullet \boldsymbol{Y} &= \sum_{i=1}^{n} \lambda_i \boldsymbol{p}_i \boldsymbol{p}_i^T \bullet \sum_{j=1}^{n} \mu_j \boldsymbol{q}_j \boldsymbol{q}_j^T = \sum_{i=1}^{n} \lambda_i \sum_{j=1}^{n} \mu_j \boldsymbol{p}_i \boldsymbol{p}_i^T \bullet \boldsymbol{q}_j \boldsymbol{q}_j^T \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \mu_j (\boldsymbol{p}_i^T \boldsymbol{q}_j)^2 \geq 0. \end{aligned}$$

Weak Duality

Standard from

$$(\mathcal{P}) \quad : \quad \min : \boldsymbol{C} \bullet \boldsymbol{X} \qquad \text{s.t.} \quad \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \ (k = 1, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

$$(\mathcal{D}) \quad : \quad \max : \sum_{k=1}^{m} b_k z_k \qquad \text{s.t.} \quad \sum_{k=1}^{m} \boldsymbol{A}_k z_k + \boldsymbol{Y} = \boldsymbol{C}, \quad \boldsymbol{Y} \succeq \boldsymbol{O}$$

Weak Duality

For any feasible point $(\boldsymbol{X}^+, \boldsymbol{Y}^+, \boldsymbol{z}^+)$ $\qquad \boldsymbol{C} \bullet \boldsymbol{X}^+ \geq \sum_{k=1}^{m} b_k z_k^+$

Weak Duality

Standard from

$$(\mathcal{P}) \quad : \quad \min : \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t.} \quad \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \ (k = 1, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

$$(\mathcal{D}) \quad : \quad \max : \sum_{k=1}^{m} b_k z_k \quad \text{s.t.} \quad \sum_{k=1}^{m} \boldsymbol{A}_k z_k + \boldsymbol{Y} = \boldsymbol{C}, \quad \boldsymbol{Y} \succeq \boldsymbol{O}$$

Weak Duality

For any feasible point $(\boldsymbol{X}^+, \boldsymbol{Y}^+, \boldsymbol{z}^+)$ $\quad \boldsymbol{C} \bullet \boldsymbol{X}^+ \geq \sum_{k=1}^{m} b_k z_k^+$

Proof:

$$\begin{aligned}
\boldsymbol{C} \bullet \boldsymbol{X}^+ - \sum_{k=1}^{m} b_k z_k^+ &= \left(\sum_{k=1}^{m} \boldsymbol{A}_k z_k^+ + \boldsymbol{Y}^+ \right) \bullet \boldsymbol{X}^+ - \sum_{k=1}^{m} (\boldsymbol{A}_k \bullet \boldsymbol{X}^+) z_k^+ \\
&= \sum_{k=1}^{m} (\boldsymbol{A}_k \bullet \boldsymbol{X}^+) z_k^+ + \boldsymbol{X}^+ \bullet \boldsymbol{Y}^+ - \sum_{k=1}^{m} (\boldsymbol{A}_k \bullet \boldsymbol{X}^+) z_k^+ \\
&= \boldsymbol{X}^+ \bullet \boldsymbol{Y}^+ \geq 0.
\end{aligned}$$

Strong Duality

Weak Duality

For any feasible point $(\boldsymbol{X}^+, \boldsymbol{Y}^+, \boldsymbol{z}^+)$ $\quad \boldsymbol{C} \bullet \boldsymbol{X}^+ \geq \sum_{k=1}^m b_k z_k^+$

Strong Duality

Assume Slater's Condition $\quad (\exists(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}}, \widehat{\boldsymbol{z}})$ such that feasible and $\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}} \succ \boldsymbol{O})$.
If feasible point $(\boldsymbol{X}^*, \boldsymbol{Y}^*, \boldsymbol{z}^*)$ satisfies $\boldsymbol{C} \bullet \boldsymbol{X}^* = \sum_{k=1}^m b_k z_k^*$,
then $(\boldsymbol{X}^*, \boldsymbol{Y}^*, \boldsymbol{z}^*)$ is an optimal solution, and *vice versa*.

Strong Duality

Weak Duality

For any feasible point $(\boldsymbol{X}^+, \boldsymbol{Y}^+, \boldsymbol{z}^+)$ $\quad \boldsymbol{C} \bullet \boldsymbol{X}^+ \geq \sum_{k=1}^m b_k z_k^+$

Strong Duality

Assume Slater's Condition $\quad (\exists(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}}, \widehat{\boldsymbol{z}})$ such that feasible and $\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}} \succ \boldsymbol{O})$.
If feasible point $(\boldsymbol{X}^*, \boldsymbol{Y}^*, \boldsymbol{z}^*)$ satisfies $\boldsymbol{C} \bullet \boldsymbol{X}^* = \sum_{k=1}^m b_k z_k^*$,
then $(\boldsymbol{X}^*, \boldsymbol{Y}^*, \boldsymbol{z}^*)$ is an optimal solution, and *vice versa*.

Proof: only $(\Rightarrow)$

$$\boldsymbol{C} \bullet \boldsymbol{X}^+ \underbrace{\geq}_{\text{weak}} \sum_{k=1}^m b_k z_k^* \underbrace{=}_{\text{strong}} \boldsymbol{C} \bullet \boldsymbol{X}^* \underbrace{\geq}_{\text{weak}} \sum_{k=1}^m b_k z_k^+$$

$$\Rightarrow \boldsymbol{C} \bullet \boldsymbol{X}^+ \geq \boldsymbol{C} \bullet \boldsymbol{X}^*, \quad \sum_{k=1}^m b_k z_k^* \geq \sum_{k=1}^m b_k z_k^+$$

Optimality Condition

$$0 = \boldsymbol{C} \bullet \boldsymbol{X}^* - \sum_{k=1}^{m} b_k z_k^*$$

Optimality Condition

$$0 = \boldsymbol{C} \bullet \boldsymbol{X}^* - \sum_{k=1}^{m} b_k z_k^* = \boldsymbol{X}^* \bullet \boldsymbol{Y}^*$$

Optimality Condition

$$0 = \boldsymbol{C} \bullet \boldsymbol{X}^* - \sum_{k=1}^{m} b_k z_k^* = \boldsymbol{X}^* \bullet \boldsymbol{Y}^*$$

Furthermore, from $\boldsymbol{X}^*, \boldsymbol{Y}^* \succeq \boldsymbol{O}$

$$\boldsymbol{X}^* \bullet \boldsymbol{Y}^* = 0 \Leftrightarrow \boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{O}$$

Optimality Condition

$$0 = \boldsymbol{C} \bullet \boldsymbol{X}^* - \sum_{k=1}^{m} b_k z_k^* = \boldsymbol{X}^* \bullet \boldsymbol{Y}^*$$

Furthermore, from $\boldsymbol{X}^*, \boldsymbol{Y}^* \succeq \boldsymbol{O}$

$$\boldsymbol{X}^* \bullet \boldsymbol{Y}^* = 0 \Leftrightarrow \boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{O}$$

Optimality Condition (Karush-Kuhn-Tucker Condition)

$$\begin{cases} \boldsymbol{A}_k \bullet \boldsymbol{X}^* = b_k \ (k = 1, \ldots, m) & \text{primal feasibility} \\ \sum_{k=1}^{n} \boldsymbol{A}_k z_k^* + \boldsymbol{Y}^* = \boldsymbol{C} & \text{dual feasibility} \\ \boldsymbol{X}^*, \boldsymbol{Y}^* \succeq \boldsymbol{O} & \text{positive semidefiniteness} \\ \boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{O} & \text{complementarity} \end{cases}$$

Optimality Condition

$$0 = \boldsymbol{C} \bullet \boldsymbol{X}^* - \sum_{k=1}^{m} b_k z_k^* = \boldsymbol{X}^* \bullet \boldsymbol{Y}^*$$

Furthermore, from $\boldsymbol{X}^*, \boldsymbol{Y}^* \succeq \boldsymbol{O}$

$$\boldsymbol{X}^* \bullet \boldsymbol{Y}^* = 0 \Leftrightarrow \boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{O}$$

Optimality Condition (Karush-Kuhn-Tucker Condition)

$$\begin{cases} \boldsymbol{A}_k \bullet \boldsymbol{X}^* = b_k \ (k = 1, \ldots, m) & \text{primal feasibility} \\ \sum_{k=1}^{n} \boldsymbol{A}_k z_k^* + \boldsymbol{Y}^* = \boldsymbol{C} & \text{dual feasibility} \\ \boldsymbol{X}^*, \boldsymbol{Y}^* \succeq \boldsymbol{O} & \text{positive semidefiniteness} \\ \boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{O} & \text{complementarity} \end{cases}$$

This $(\boldsymbol{X}^*, \boldsymbol{Y}^*, \boldsymbol{z}^*)$ is the solution we want to obtain.

2. Primal-Dual Interior-Point Methods

- Central path
- Path-following algorithm
- Search direction
- Schur complement matrix

Central Path

Optimality Condition

$$\begin{cases} A_k \bullet X^* = b_k \ (k = 1, \ldots, m) & \text{primal feasibility} \\ \sum_{k=1}^n A_k z_k^* + Y^* = C & \text{dual feasibility} \\ X^*, Y^* \succeq O & \text{positive semidefiniteness} \\ X^* Y^* = O & \text{complementarity} \end{cases}$$

Perturbed Condition ($\mu > 0$)

$$\begin{cases} A_k \bullet X(\mu) = b_k \ (k = 1, \ldots, m) & \text{primal feasibility} \\ \sum_{k=1}^n A_k z_k(\mu) + Y(\mu) = C & \text{dual feasibility} \\ X(\mu), Y(\mu) \succeq O & \text{positive semidefiniteness} \\ X(\mu) Y(\mu) = \mu I & \text{complementarity} \end{cases}$$

Central Path

Optimality Condition

$$\begin{cases} \boldsymbol{A}_k \bullet \boldsymbol{X}^* = b_k \ (k = 1, \ldots, m) & \text{primal feasibility} \\ \sum_{k=1}^n \boldsymbol{A}_k z_k^* + \boldsymbol{Y}^* = \boldsymbol{C} & \text{dual feasibility} \\ \boldsymbol{X}^*, \boldsymbol{Y}^* \succeq \boldsymbol{O} & \text{positive semidefiniteness} \\ \boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{O} & \text{complementarity} \end{cases}$$

Perturbed Condition ($\mu > 0$)

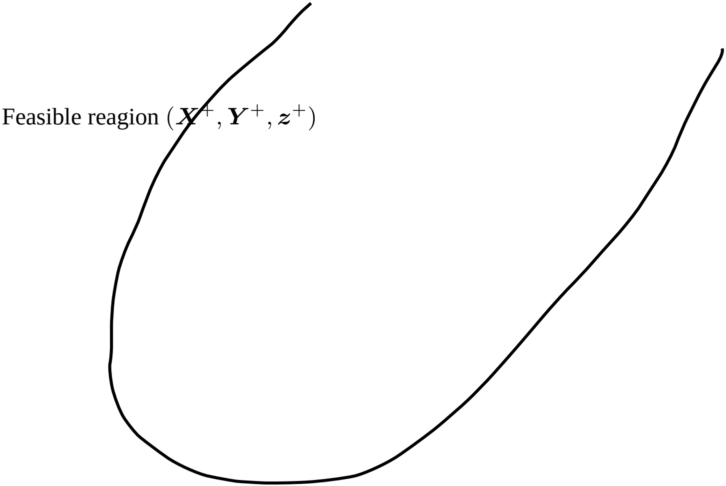
$$\begin{cases} \boldsymbol{A}_k \bullet \boldsymbol{X}(\mu) = b_k \ (k = 1, \ldots, m) & \text{primal feasibility} \\ \sum_{k=1}^n \boldsymbol{A}_k z_k(\mu) + \boldsymbol{Y}(\mu) = \boldsymbol{C} & \text{dual feasibility} \\ \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu) \succeq \boldsymbol{O} & \text{positive semidefiniteness} \\ \boldsymbol{X}(\mu) \boldsymbol{Y}(\mu) = \mu \boldsymbol{I} & \text{complementarity} \end{cases}$$

Central Path

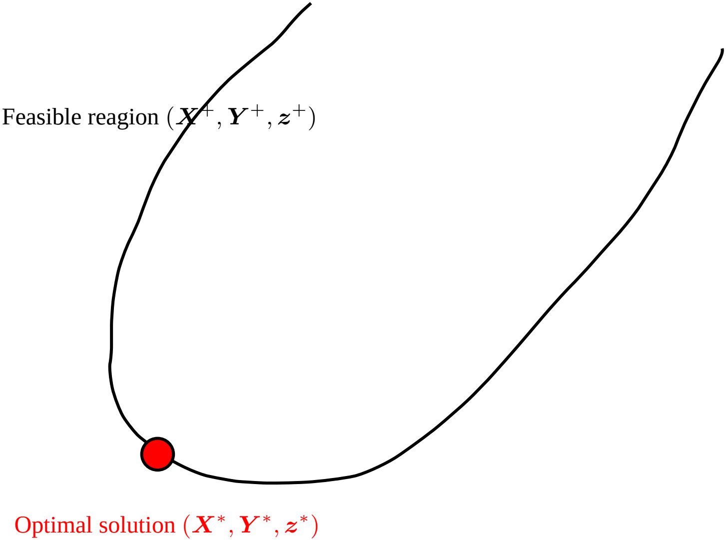
Central path is defined by $\{(\boldsymbol{X}(\mu), \boldsymbol{Y}(\mu), \boldsymbol{z}(\mu)) : \mu > 0\}$.

- For any $\mu > 0$, there exists a unique point $(\boldsymbol{X}(\mu), \boldsymbol{Y}(\mu), \boldsymbol{z}(\mu))$.
- $(\boldsymbol{X}(\mu), \boldsymbol{Y}(\mu), \boldsymbol{z}(\mu)) \to (\boldsymbol{X}^*, \boldsymbol{Y}^*, \boldsymbol{z}^*)$ when $\mu \to 0$.

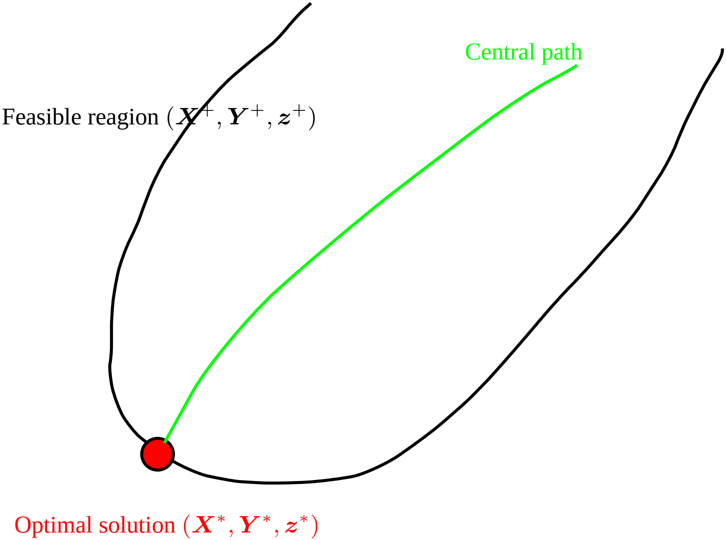
Path Following Algorithm



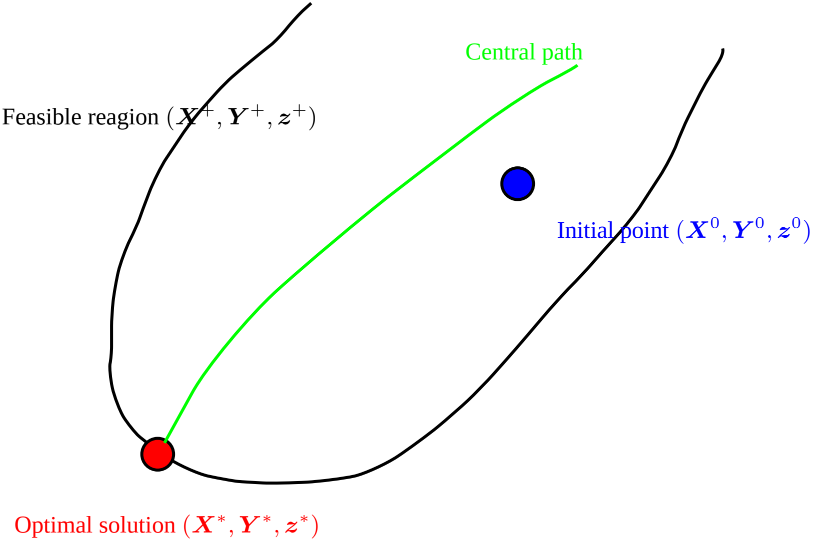
Path Following Algorithm



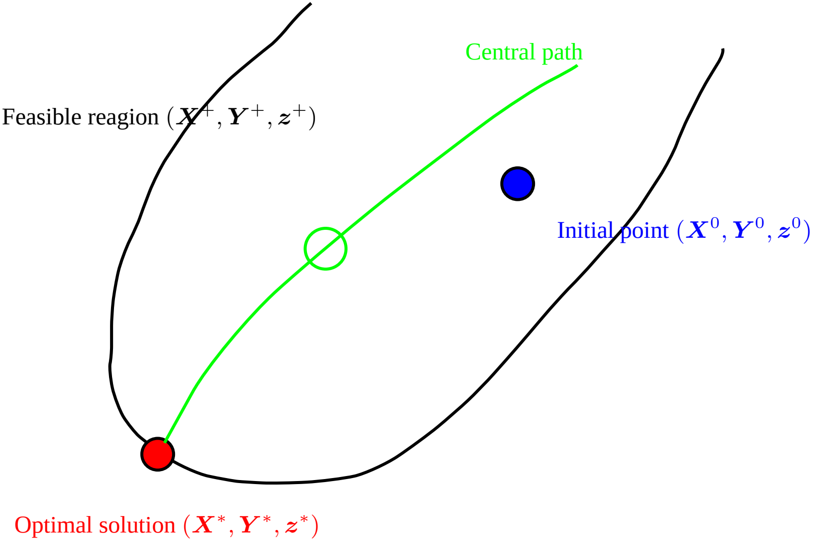
Path Following Algorithm



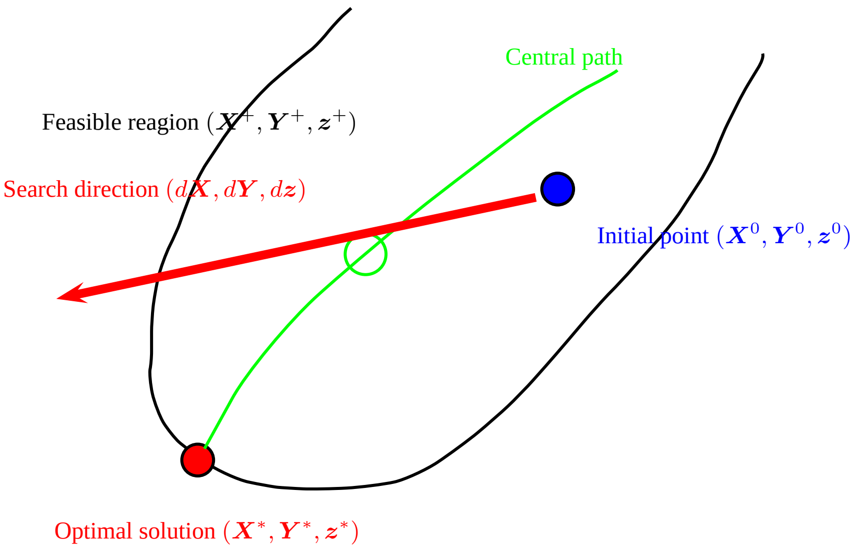
Path Following Algorithm



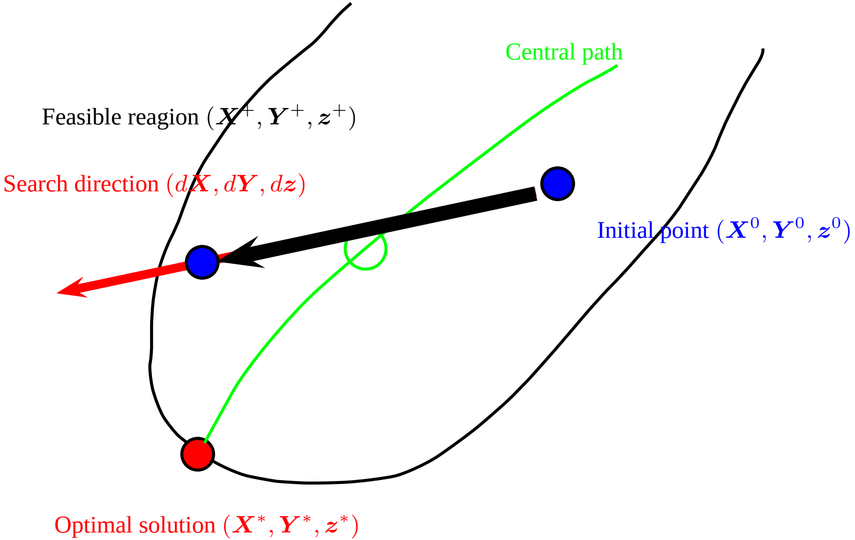
Path Following Algorithm



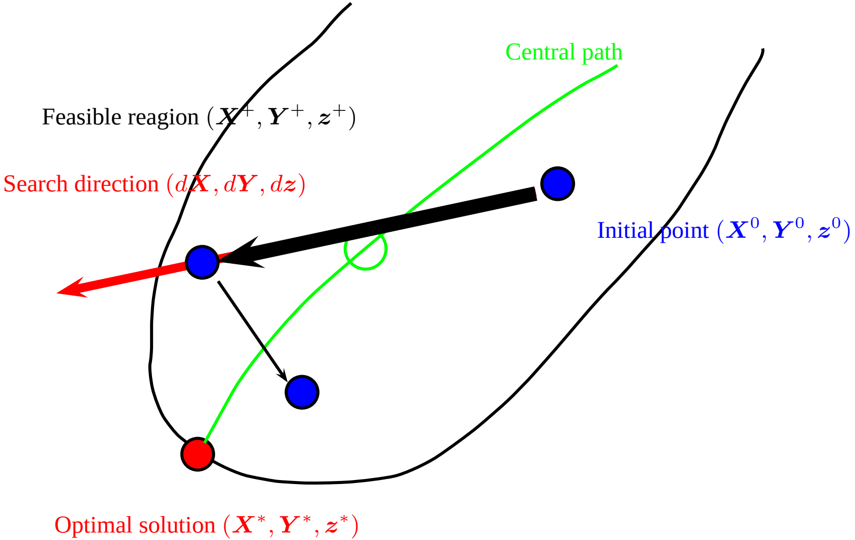
Path Following Algorithm



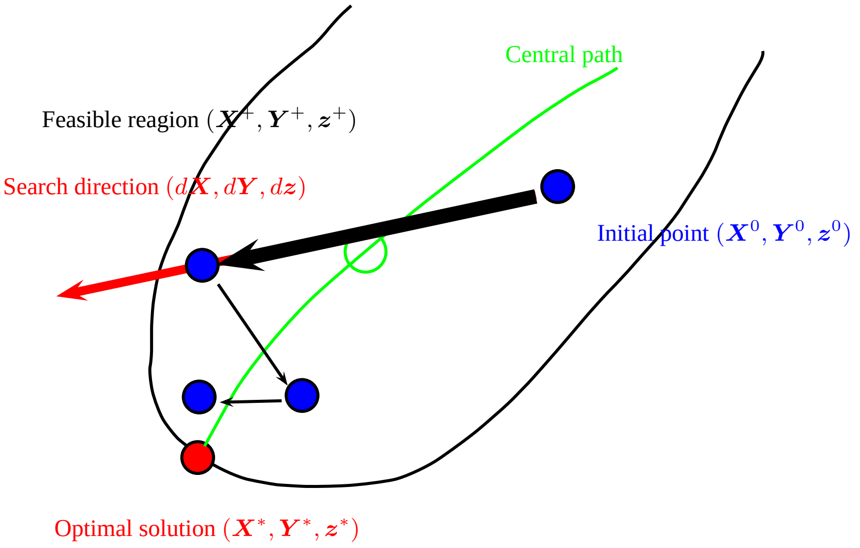
Path Following Algorithm



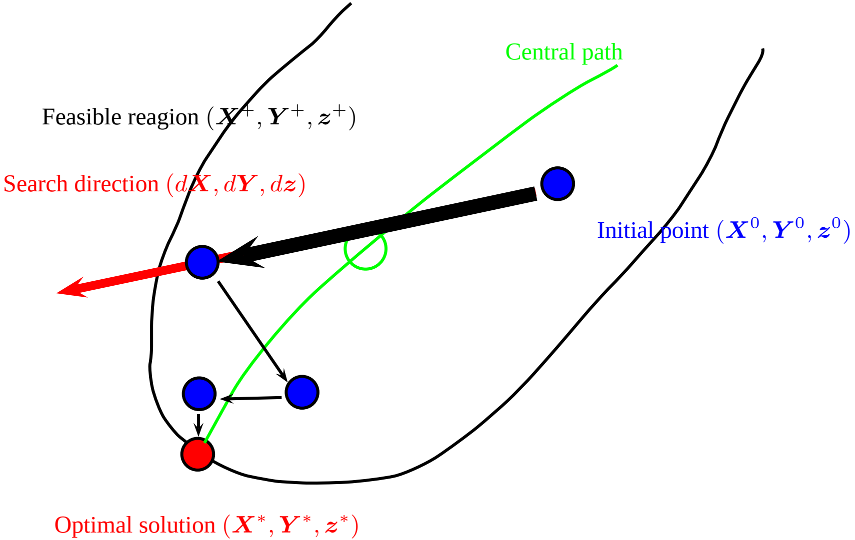
Path Following Algorithm



Path Following Algorithm



Path Following Algorithm



Path-Following Interior-Point Algorithm

1. Choose an initial point $(\boldsymbol{X}^0, \boldsymbol{Y}^0, \boldsymbol{z}^0)$ with $\boldsymbol{X}^0, \boldsymbol{Y}^0 \succ \boldsymbol{O}$. Set the iteration number $h = 0$. Choose parameters $0 < \beta < 1, 0 < \gamma < 1$.
2. Compute a search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$.
3. To keep positive semidefiniteness, compute the maximum step length $\alpha_{\max}$ by
$$\alpha_{\max} = \max\{\alpha : \boldsymbol{X}^h + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}, \boldsymbol{Y}^h + \alpha d\boldsymbol{Y} \succeq \boldsymbol{O}\}$$
4. Set $(\boldsymbol{X}^{h+1}, \boldsymbol{Y}^{h+1}, \boldsymbol{z}^{h+1}) = (\boldsymbol{X}^h, \boldsymbol{Y}^h, \boldsymbol{z}^h) + \gamma\alpha_{\max}(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$, $h \to h + 1$.
5. If $(\boldsymbol{X}^h, \boldsymbol{Y}^h, \boldsymbol{z}^h)$ satisfies a stopping criteria, then stop. Otherwise, return to 2.

Path-Following Interior-Point Algorithm

1. Choose an initial point $(\boldsymbol{X}^0, \boldsymbol{Y}^0, \boldsymbol{z}^0)$ with $\boldsymbol{X}^0, \boldsymbol{Y}^0 \succ \boldsymbol{O}$. Set the iteration number $h = 0$. Choose parameters $0 < \beta < 1, 0 < \gamma < 1$.
2. Compute a search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$.
3. To keep positive semidefiniteness, compute the maximum step length $\alpha_{\max}$ by

$$\alpha_{\max} = \max\{\alpha : \boldsymbol{X}^h + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}, \boldsymbol{Y}^h + \alpha d\boldsymbol{Y} \succeq \boldsymbol{O}\}$$

4. Set $(\boldsymbol{X}^{h+1}, \boldsymbol{Y}^{h+1}, \boldsymbol{z}^{h+1}) = (\boldsymbol{X}^h, \boldsymbol{Y}^h, \boldsymbol{z}^h) + \gamma\alpha_{\max}(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$, $h \to h+1$.
5. If $(\boldsymbol{X}^h, \boldsymbol{Y}^h, \boldsymbol{z}^h)$ satisfies a stopping criteria, then stop. Otherwise, return to 2.

The most important computation is the search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$.

Search Direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$

The point $(\boldsymbol{X}(\mu), \boldsymbol{Y}(\mu), \boldsymbol{z}(\mu))$ on the central path satisfies

$$\boldsymbol{X}(\mu) \bullet \boldsymbol{Y}(\mu) = n\mu.$$

Search Direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$

The point $(\boldsymbol{X}(\mu), \boldsymbol{Y}(\mu), \boldsymbol{z}(\mu))$ on the central path satisfies

$$\boldsymbol{X}(\mu) \bullet \boldsymbol{Y}(\mu) = n\mu.$$

The search direction from $(\boldsymbol{X}^h, \boldsymbol{Y}^h, \boldsymbol{z}^h)$ should head to $(\boldsymbol{X}(\beta\mu), \boldsymbol{Y}(\beta\mu), \boldsymbol{z}(\beta\mu))$ with $\mu = \frac{\boldsymbol{X}^h \bullet \boldsymbol{Y}^h}{n}, 0 < \beta < 1.$

Search Direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$

The point $(\boldsymbol{X}(\mu), \boldsymbol{Y}(\mu), \boldsymbol{z}(\mu))$ on the central path satisfies

$$\boldsymbol{X}(\mu) \bullet \boldsymbol{Y}(\mu) = n\mu.$$

The search direction from $(\boldsymbol{X}^h, \boldsymbol{Y}^h, \boldsymbol{z}^h)$ should head to $(\boldsymbol{X}(\beta\mu), \boldsymbol{Y}(\beta\mu), \boldsymbol{z}(\beta\mu))$ with $\mu = \frac{\boldsymbol{X}^h \bullet \boldsymbol{Y}^h}{n}, 0 < \beta < 1$.

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\begin{cases} \boldsymbol{A}_k \bullet (\boldsymbol{X}^h + d\boldsymbol{X}) = b_k \quad (k = 1, \ldots, m) \\ \sum_{k=1}^m \boldsymbol{A}_k (z_k^h + dz_k) + (\boldsymbol{Y} + d\boldsymbol{Y}) = \boldsymbol{C} \\ (\boldsymbol{X}^h + d\boldsymbol{X})(\boldsymbol{Y}^h + d\boldsymbol{Y}) = \beta\mu\boldsymbol{I} \end{cases}$$

- In the moment we ignore $\boldsymbol{X}, \boldsymbol{Y} \succeq \boldsymbol{O}$,
  since we control the point by the step length $\alpha_{\max}$.

Search Direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$

The point $(\boldsymbol{X}(\mu), \boldsymbol{Y}(\mu), \boldsymbol{z}(\mu))$ on the central path satisfies

$$\boldsymbol{X}(\mu) \bullet \boldsymbol{Y}(\mu) = n\mu.$$

The search direction from $(\boldsymbol{X}^h, \boldsymbol{Y}^h, \boldsymbol{z}^h)$ should head to $(\boldsymbol{X}(\beta\mu), \boldsymbol{Y}(\beta\mu), \boldsymbol{z}(\beta\mu))$ with $\mu = \frac{\boldsymbol{X}^h \bullet \boldsymbol{Y}^h}{n}, 0 < \beta < 1$.

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\begin{cases} \boldsymbol{A}_k \bullet (\boldsymbol{X}^h + d\boldsymbol{X}) = b_k \quad (k = 1, \ldots, m) \\ \sum_{k=1}^m \boldsymbol{A}_k (z_k^h + dz_k) + (\boldsymbol{Y} + d\boldsymbol{Y}) = \boldsymbol{C} \\ (\boldsymbol{X}^h + d\boldsymbol{X})(\boldsymbol{Y}^h + d\boldsymbol{Y}) = \beta\mu\boldsymbol{I} \end{cases}$$

- In the moment we ignore $\boldsymbol{X}, \boldsymbol{Y} \succeq \boldsymbol{O}$, since we control the point by the step length $\alpha_{\max}$.
- This system is nonlinear due to the cross term $d\boldsymbol{X}d\boldsymbol{Y}$ in $(\boldsymbol{X}^h + d\boldsymbol{X})(\boldsymbol{Y}^h + d\boldsymbol{Y}) = \beta\mu\boldsymbol{I}$. We ignore it, and replace by $\boldsymbol{X}^h\boldsymbol{Y}^h + d\boldsymbol{X}\boldsymbol{Y}^h + \boldsymbol{X}^h d\boldsymbol{Y} = \beta\mu\boldsymbol{I}$.

Schur Complement Equation

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\left\{ \begin{array}{l} \boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i := b_i - \boldsymbol{A}_k \bullet \boldsymbol{X}^h \quad (i = 1, \ldots, m) \\ \sum_{j=1}^m \boldsymbol{A}_j dz_j + d\boldsymbol{Y} = \boldsymbol{D} := \boldsymbol{C} - \sum_{j=1}^m \boldsymbol{A}_j z_j^h - \boldsymbol{Y}^h \\ d\boldsymbol{X}\boldsymbol{Y}^h + \boldsymbol{X}^h d\boldsymbol{Y} = \boldsymbol{R} := \beta\mu\boldsymbol{I} - \boldsymbol{X}^h\boldsymbol{Y}^h \end{array} \right.$$

Schur Complement Equation

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\begin{cases} \boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i := b_i - \boldsymbol{A}_k \bullet \boldsymbol{X}^h \quad (i = 1, \ldots, m) \\ \sum_{j=1}^m \boldsymbol{A}_j dz_j + d\boldsymbol{Y} = \boldsymbol{D} := \boldsymbol{C} - \sum_{j=1}^m \boldsymbol{A}_j z_j^h - \boldsymbol{Y}^h \\ d\boldsymbol{X}\boldsymbol{Y}^h + \boldsymbol{X}^h d\boldsymbol{Y} = \boldsymbol{R} := \beta\mu\boldsymbol{I} - \boldsymbol{X}^h\boldsymbol{Y}^h \end{cases}$$

$$\begin{cases} d\boldsymbol{Y} = \boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j \\ \\ \\ \end{cases}$$

Schur Complement Equation

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\left\{ \begin{array}{l} \boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i := b_i - \boldsymbol{A}_k \bullet \boldsymbol{X}^h \quad (i = 1, \dots, m) \\ \sum_{j=1}^m \boldsymbol{A}_j dz_j + d\boldsymbol{Y} = \boldsymbol{D} := \boldsymbol{C} - \sum_{j=1}^m \boldsymbol{A}_j z_j^h - \boldsymbol{Y}^h \\ d\boldsymbol{X}\boldsymbol{Y}^h + \boldsymbol{X}^h d\boldsymbol{Y} = \boldsymbol{R} := \beta\mu\boldsymbol{I} - \boldsymbol{X}^h\boldsymbol{Y}^h \end{array} \right.$$

$$\left\{ \begin{array}{l} d\boldsymbol{Y} = \boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j \\ d\boldsymbol{X} = (\boldsymbol{R} - \boldsymbol{X}^h d\boldsymbol{Y})(\boldsymbol{Y}^h)^{-1} = (\boldsymbol{R} - \boldsymbol{X}^h(\boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j))(\boldsymbol{Y}^h)^{-1} \\ \\ \end{array} \right.$$

Schur Complement Equation

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\begin{cases} \boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i := b_i - \boldsymbol{A}_k \bullet \boldsymbol{X}^h \quad (i = 1, \ldots, m) \\ \sum_{j=1}^m \boldsymbol{A}_j dz_j + d\boldsymbol{Y} = \boldsymbol{D} := \boldsymbol{C} - \sum_{j=1}^m \boldsymbol{A}_j z_j^h - \boldsymbol{Y}^h \\ d\boldsymbol{X}\boldsymbol{Y}^h + \boldsymbol{X}^h d\boldsymbol{Y} = \boldsymbol{R} := \beta\mu\boldsymbol{I} - \boldsymbol{X}^h\boldsymbol{Y}^h \end{cases}$$

$$\begin{cases} d\boldsymbol{Y} = \boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j \\ d\boldsymbol{X} = (\boldsymbol{R} - \boldsymbol{X}^h d\boldsymbol{Y})(\boldsymbol{Y}^h)^{-1} = (\boldsymbol{R} - \boldsymbol{X}^h(\boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j))(\boldsymbol{Y}^h)^{-1} \\ \boldsymbol{A}_i \bullet (\boldsymbol{R} - \boldsymbol{X}^h(\boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j))(\boldsymbol{Y}^h)^{-1} = p_i \quad (i = 1, \ldots, m) \end{cases}$$

Schur Complement Equation

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\left\{ \begin{array}{l} \boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i := b_i - \boldsymbol{A}_k \bullet \boldsymbol{X}^h \quad (i = 1, \ldots, m) \\ \sum_{j=1}^m \boldsymbol{A}_j dz_j + d\boldsymbol{Y} = \boldsymbol{D} := \boldsymbol{C} - \sum_{j=1}^m \boldsymbol{A}_j z_j^h - \boldsymbol{Y}^h \\ d\boldsymbol{X}\boldsymbol{Y}^h + \boldsymbol{X}^h d\boldsymbol{Y} = \boldsymbol{R} := \beta\mu \boldsymbol{I} - \boldsymbol{X}^h \boldsymbol{Y}^h \end{array} \right.$$

$$\left\{ \begin{array}{l} d\boldsymbol{Y} = \boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j \\ d\boldsymbol{X} = (\boldsymbol{R} - \boldsymbol{X}^h d\boldsymbol{Y})(\boldsymbol{Y}^h)^{-1} = (\boldsymbol{R} - \boldsymbol{X}^h(\boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j))(\boldsymbol{Y}^h)^{-1} \\ \boldsymbol{A}_i \bullet (\boldsymbol{R} - \boldsymbol{X}^h(\boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j))(\boldsymbol{Y}^h)^{-1} = p_i \quad (i = 1, \ldots, m) \\ \sum_{j=1}^m \boldsymbol{A}_i \bullet (\boldsymbol{X}^h \boldsymbol{A}_j (\boldsymbol{Y}^h)^{-1}) dz_j = p_i - \boldsymbol{A}_i \bullet (\boldsymbol{R} - \boldsymbol{X}^h \boldsymbol{D})(\boldsymbol{Y}^h)^{-1} \quad (i = 1, \ldots, m) \end{array} \right.$$

Schur Complement Equation

The system for next point $(\boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y}, \boldsymbol{z}^h + d\boldsymbol{z})$

$$\left\{ \begin{array}{l} \boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i := b_i - \boldsymbol{A}_k \bullet \boldsymbol{X}^h \quad (i = 1, \ldots, m) \\ \sum_{j=1}^m \boldsymbol{A}_j dz_j + d\boldsymbol{Y} = \boldsymbol{D} := \boldsymbol{C} - \sum_{j=1}^m \boldsymbol{A}_j z_j^h - \boldsymbol{Y}^h \\ d\boldsymbol{X}\boldsymbol{Y}^h + \boldsymbol{X}^h d\boldsymbol{Y} = \boldsymbol{R} := \beta\mu\boldsymbol{I} - \boldsymbol{X}^h\boldsymbol{Y}^h \end{array} \right.$$

$$\left\{ \begin{array}{l} d\boldsymbol{Y} = \boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j \\ d\boldsymbol{X} = (\boldsymbol{R} - \boldsymbol{X}^h d\boldsymbol{Y})(\boldsymbol{Y}^h)^{-1} = (\boldsymbol{R} - \boldsymbol{X}^h(\boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j))(\boldsymbol{Y}^h)^{-1} \\ \boldsymbol{A}_i \bullet (\boldsymbol{R} - \boldsymbol{X}^h(\boldsymbol{D} - \sum_{j=1}^m \boldsymbol{A}_j dz_j))(\boldsymbol{Y}^h)^{-1} = p_i \quad (i = 1, \ldots, m) \\ \sum_{j=1}^m \boldsymbol{A}_i \bullet (\boldsymbol{X}^h \boldsymbol{A}_j (\boldsymbol{Y}^h)^{-1}) dz_j = p_i - \boldsymbol{A}_i \bullet (\boldsymbol{R} - \boldsymbol{X}^h \boldsymbol{D})(\boldsymbol{Y}^h)^{-1} \quad (i = 1, \ldots, m) \end{array} \right.$$

Schur Complement Equation

Define $\boldsymbol{B} \in \mathbb{S}^m$ by $B_{ij} = \boldsymbol{A}_i \bullet (\boldsymbol{X}^h \boldsymbol{A}_j (\boldsymbol{Y}^h)^{-1})$
and $\boldsymbol{r} \in \mathbb{R}^m$ by $r_i = p_i - \boldsymbol{A}_i \bullet (\boldsymbol{R} - \boldsymbol{X}^h \boldsymbol{D})(\boldsymbol{Y}^h)^{-1}$.
Then Schur complement equation is

$$\boldsymbol{B} d\boldsymbol{z} = \boldsymbol{r}$$

Computational Bottlenecks

Schur Complement Matrix

$$\boldsymbol{B} \in \mathbb{S}^m \text{ with } B_{ij} = \boldsymbol{A}_i \bullet (\boldsymbol{X}^h \boldsymbol{A}_j (\boldsymbol{Y}^h)^{-1})$$

Computational Bottlenecks

Schur Complement Matrix

$$\boldsymbol{B} \in \mathbb{S}^m \text{ with } B_{ij} = \boldsymbol{A}_i \bullet (\boldsymbol{X}^h \boldsymbol{A}_j (\boldsymbol{Y}^h)^{-1})$$

To solve $\boldsymbol{B} d\boldsymbol{z} = \boldsymbol{r}$, we apply

the Cholesky factorization $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^T$ ($\boldsymbol{L}$ is the lower triangular matrix)

and then solve $\boldsymbol{L}^T \widetilde{d\boldsymbol{z}} = \boldsymbol{r}$ and $\boldsymbol{L} d\boldsymbol{z} = \widetilde{d\boldsymbol{z}}$.

Computational Bottlenecks

Schur Complement Matrix

$$\boldsymbol{B} \in \mathbb{S}^m \text{ with } B_{ij} = \boldsymbol{A}_i \bullet (\boldsymbol{X}^h \boldsymbol{A}_j (\boldsymbol{Y}^h)^{-1})$$

To solve $\boldsymbol{B}d\boldsymbol{z} = \boldsymbol{r}$, we apply

the Cholesky factorization $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^T$ ($\boldsymbol{L}$ is the lower triangular matrix)

and then solve $\boldsymbol{L}^T \widetilde{d\boldsymbol{z}} = \boldsymbol{r}$ and $\boldsymbol{L}d\boldsymbol{z} = \widetilde{d\boldsymbol{z}}$.

Computational Bottlenecks

1. (ELEMENTS) Evaluation of $\boldsymbol{B}$ by $B_{ij} = \boldsymbol{A}_i \bullet (\boldsymbol{X}^h \boldsymbol{A}_j (Y^h)^{-1})$
2. (CHOLESKY) the Cholesky factorization of $\boldsymbol{B}$

	ELEMENTS	CHOLESKY	Total
Stability Condition	22228	1593	23986
Polynomial Optimization	668	1992	2713

Time unit is *second*, SDPA 7, Xeon 5460 (3.16GHz)

3. Inside of SDPA

Most improvements are for ELEMENTS and CHOLESKY for *the fastest solver for large-scale SDPs*.

1. Exploiting Sparsity by three formulas $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$
2. Multiple Threads
3. Sparse Cholesky factorization

Sparsity in Input Data Matrices

Standard from

$$(\mathcal{P}) : \min : \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t.} \quad \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \ (k = 1, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

$\boldsymbol{A}_k \in \mathbb{S}^n$ are very sparse in most applications. (*i.e.*, most elements in $\boldsymbol{A}_k$ are zero)
For example, in Max-cut problems, $\boldsymbol{A}_k$ has only one non-zero elements in $n \times n$ positions.

$$\boldsymbol{A}_k = \begin{array}{c} \\ k \end{array} \begin{array}{c} k \\ \left(\begin{array}{ccc} & & \\ & 1 & \\ & & \end{array} \right) \end{array}$$

Sparsity in Input Data Matrices

Standard from

$$(\mathcal{P}) : \min : \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t.} \quad \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \ (k = 1, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

$\boldsymbol{A}_k \in \mathbb{S}^n$ are very sparse in most applications. (*i.e.*, most elements in $\boldsymbol{A}_k$ are zero) For example, in Max-cut problems, $\boldsymbol{A}_k$ has only one non-zero elements in $n \times n$ positions.

$$\begin{array}{cc} & k \\ \boldsymbol{A}_k = k & \left(\begin{array}{ccc} & & \\ & 1 & \\ & & \end{array} \right) \end{array}$$

The SCM is evaluated by

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j$$

In Max-Cut Problem, computing $(\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1})$ is efficient?

Sparsity in Input Data Matrices

Standard from

$$(\mathcal{P}) : \min : \boldsymbol{C} \bullet \boldsymbol{X} \qquad \text{s.t.} \quad \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \ (k = 1, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

$\boldsymbol{A}_k \in \mathbb{S}^n$ are very sparse in most applications. (*i.e.*, most elements in $\boldsymbol{A}_k$ are zero) For example, in Max-cut problems, $\boldsymbol{A}_k$ has only one non-zero elements in $n \times n$ positions.

$$\boldsymbol{A}_k = \begin{array}{cc} & k \\ k & \left(\begin{array}{ccc} \quad & & \quad \\ & 1 & \\ & & \end{array} \right) \end{array}$$

The SCM is evaluated by

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j$$

In Max-Cut Problem, computing $(\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1})$ is efficient?
No! Only (j, j) element of $(\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1})$ is necessary.

$$\boldsymbol{U} \bullet \boldsymbol{V} = \sum_{p=1}^{n}\sum_{q=1}^{n} U_{pq}V_{pq} \Rightarrow \boldsymbol{U} \bullet \boldsymbol{A}_j = \sum_{p=1}^{n}\sum_{q=1}^{n} U_{pq}[\boldsymbol{A}_j]_{pq} = U_{jj}[\boldsymbol{A}_j]_{jj} = U_{jj}$$

Sparsity in Evaluation Formula

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j$$

If $\boldsymbol{A}_i$ is dense, we first compute $\boldsymbol{U}_i := \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}$,
then take $\boldsymbol{U}_i \bullet \boldsymbol{A}_j$ for $j = 1, \ldots, m$.

Sparsity in Evaluation Formula

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j$$

If $\boldsymbol{A}_i$ is dense, we first compute $\boldsymbol{U}_i := \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}$,
then take $\boldsymbol{U}_i \bullet \boldsymbol{A}_j$ for $j = 1, \ldots, m$.
If $\boldsymbol{A}_i$ is sparse,

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}]_{\alpha,\beta}[\boldsymbol{A}_j]_{\alpha,\beta}$$

Sparsity in Evaluation Formula

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j$$

If $\boldsymbol{A}_i$ is dense, we first compute $\boldsymbol{U}_i := \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}$,
then take $\boldsymbol{U}_i \bullet \boldsymbol{A}_j$ for $j = 1, \ldots, m$.
If $\boldsymbol{A}_i$ is sparse,

$$\begin{aligned} B_{ij} &= (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}]_{\alpha,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \\ &= \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{X}]_{\alpha,*}\boldsymbol{A}_i[\boldsymbol{Y}^{-1}]_{*,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \end{aligned}$$

Sparsity in Evaluation Formula

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j$$

If $\boldsymbol{A}_i$ is dense, we first compute $\boldsymbol{U}_i := \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}$,
then take $\boldsymbol{U}_i \bullet \boldsymbol{A}_j$ for $j = 1, \ldots, m$.
If $\boldsymbol{A}_i$ is sparse,

$$\begin{aligned} B_{ij} &= (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}]_{\alpha,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \\ &= \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{X}]_{\alpha,*}\boldsymbol{A}_i[\boldsymbol{Y}^{-1}]_{*,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \\ &= \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}\sum_{\gamma=1}^{n}\sum_{\delta=1}^{n}[\boldsymbol{X}]_{\alpha,\gamma}[\boldsymbol{A}_i]_{\gamma,\delta}[\boldsymbol{Y}^{-1}]_{\delta,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \end{aligned}$$

Sparsity in Evaluation Formula

$$B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j$$

If $\boldsymbol{A}_i$ is dense, we first compute $\boldsymbol{U}_i := \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}$,
then take $\boldsymbol{U}_i \bullet \boldsymbol{A}_j$ for $j = 1, \ldots, m$.
If $\boldsymbol{A}_i$ is sparse,

$$\begin{aligned} B_{ij} &= (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}]_{\alpha,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \\ &= \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{X}]_{\alpha,*}\boldsymbol{A}_i[\boldsymbol{Y}^{-1}]_{*,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \\ &= \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}\sum_{\gamma=1}^{n}\sum_{\delta=1}^{n}[\boldsymbol{X}]_{\alpha,\gamma}[\boldsymbol{A}_i]_{\gamma,\delta}[\boldsymbol{Y}^{-1}]_{\delta,\beta}[\boldsymbol{A}_j]_{\alpha,\beta} \end{aligned}$$

By counting $\#\boldsymbol{A}_i$ (the number of nonzeros in $\boldsymbol{A}_i$), we select the better formula.

Computation Cost of Each Formula

By counting $\#\boldsymbol{A}_i$ (the number of nonzeros in $\boldsymbol{A}_i$), we select the better formula.

Computation Cost of Each Formula

By counting $\#\boldsymbol{A}_i$ (the number of nonzeros in $\boldsymbol{A}_i$), we select the better formula.
We count the number of multiplications.

- $\boldsymbol{U}_i = \underbrace{\boldsymbol{X}\,\underbrace{\boldsymbol{A}_i\boldsymbol{Y}^{-1}}_{n\times\#\boldsymbol{A}_i}}_{n^3},$

Computation Cost of Each Formula

By counting $\#\boldsymbol{A}_i$ (the number of nonzeros in $\boldsymbol{A}_i$), we select the better formula.
We count the number of multiplications.

- $\boldsymbol{U}_i = \underbrace{\boldsymbol{X}\underbrace{\boldsymbol{A}_i\boldsymbol{Y}^{-1}}_{n\times\#\boldsymbol{A}_i}}_{n^3}, \qquad B_{ij} = \underbrace{\boldsymbol{U}_i \bullet \boldsymbol{A}_j}_{\#\boldsymbol{A}_j}.$

  Total $n \times \#\boldsymbol{A}_i + n^3 + \#\boldsymbol{A}_j$.

Computation Cost of Each Formula

By counting $\#\boldsymbol{A}_i$ (the number of nonzeros in $\boldsymbol{A}_i$), we select the better formula.
We count the number of multiplications.

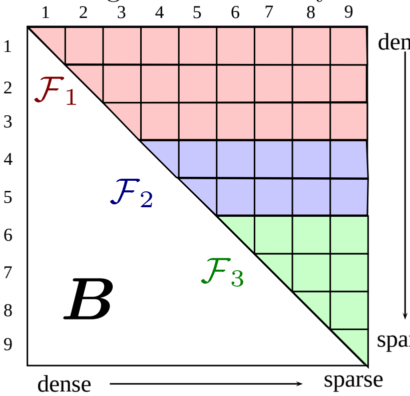
- $\boldsymbol{U}_i = \underbrace{\boldsymbol{X} \underbrace{\boldsymbol{A}_i \boldsymbol{Y}^{-1}}_{n \times \#\boldsymbol{A}_i}}_{n^3}, \qquad B_{ij} = \underbrace{\boldsymbol{U}_i \bullet \boldsymbol{A}_j}_{\#\boldsymbol{A}_j}.$

  Total $n \times \#\boldsymbol{A}_i + n^3 + \#\boldsymbol{A}_j$.
- $B_{ij} = \sum_{\alpha=1}^{n} \sum_{\beta=1}^{n} \sum_{\gamma=1}^{n} \sum_{\delta=1}^{n} \underbrace{\underbrace{[\boldsymbol{X}]_{\alpha,\gamma} [\boldsymbol{A}_i]_{\gamma,\delta} [\boldsymbol{Y}^{-1}]_{\delta,\beta}}_{2 \times \#\boldsymbol{A}_i} [\boldsymbol{A}_j]_{\alpha,\beta}}_{\#\boldsymbol{A}_j}$

  Total $2 \times \#\boldsymbol{A}_i \times \#\boldsymbol{A}_j$.

Three formulas $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$

We change the formula by row-wise.



$\mathcal{F}_1$ $\boldsymbol{A}_i$: dense $\boldsymbol{U}_i = \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}$
$\boldsymbol{A}_j$: dense $B_{ij} = \boldsymbol{U}_i \bullet \boldsymbol{A}_j$

$\mathcal{F}_2$ $\boldsymbol{A}_i$: dense $\boldsymbol{V}_i = \boldsymbol{A}_i\boldsymbol{Y}^{-1}$
$\boldsymbol{A}_j$: sparse $B_{ij} = \sum_{\alpha,\beta}[\boldsymbol{X}\boldsymbol{V}]_{\alpha,\beta}[\boldsymbol{A}_j]_{\alpha,\beta}$

$\mathcal{F}_3$ $\boldsymbol{A}_i$: sparse $B_{ij} = \sum_{\alpha,\beta}\sum_{\gamma,\delta}$
$\boldsymbol{A}_j$: sparse $[\boldsymbol{X}]_{\alpha,\gamma}[\boldsymbol{A}_i]_{\gamma,\delta}[\boldsymbol{Y}^{-1}]_{\delta,\beta}[\boldsymbol{A}_j]_{\alpha,\beta}$

Effect of Three Formula

	control10		mcp500-1	
density	3.76%		4.0×10^{-4}%	
	ELEMENTS	Total	ELEMENTS	Total
Only $\mathcal{F}_1$	1278.90	1293.66	1321.50	1346.90
$\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$	233.18	236.78	0.16	2.01

Time unit is *second*, SDPA7, Xeon 5460 3.16 GHz, 48GB meory
control10 and mcp500-1 are from SDPLIB.

Multiple Threading for SCM

- Modern CPU (Xeon, Atom) has multiple cores.
- Easy for parallel computing.

Multiple Threading for SCM

- Modern CPU (Xeon, Atom) has multiple cores.
- Easy for parallel computing.

We can expect speed-up from optimized BLAS (Basic Linear Algebra Sets).
BLAS contains many matrix manipulations;
SDPA uses matrix-matrix multiplications, Cholesky factorization, etc.

Multiple Threading for SCM

- Modern CPU (Xeon, Atom) has multiple cores.
- Easy for parallel computing.

We can expect speed-up from optimized BLAS (Basic Linear Algebra Sets).
BLAS contains many matrix manipulations;
SDPA uses matrix-matrix multiplications, Cholesky factorization, etc.

The SCM is evaluated by $B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1})\boldsymbol{A}_j$,

Multiple Threading for SCM

- Modern CPU (Xeon, Atom) has multiple cores.
- Easy for parallel computing.

We can expect speed-up from optimized BLAS (Basic Linear Algebra Sets).
BLAS contains many matrix manipulations;
SDPA uses matrix-matrix multiplications, Cholesky factorization, etc.

The SCM is evaluated by $B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1})\boldsymbol{A}_j$,
The computation of i_1th row is completely independent from i_2th row ($i_1 \neq i_2$).

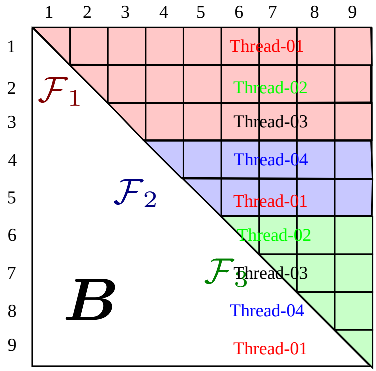
Multiple Threading for SCM

- Modern CPU (Xeon, Atom) has multiple cores.
- Easy for parallel computing.

We can expect speed-up from optimized BLAS (Basic Linear Algebra Sets).
BLAS contains many matrix manipulations;
SDPA uses matrix-matrix multiplications, Cholesky factorization, etc.

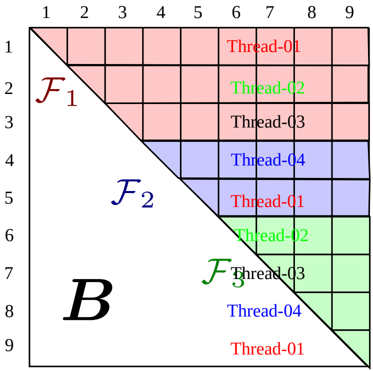
The SCM is evaluated by $B_{ij} = (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1})\boldsymbol{A}_j$,
The computation of i_1th row is completely independent from i_2th row ($i_1 \neq i_2$).
$\Rightarrow$ Very good for multiple threading.

Row-wise distribution



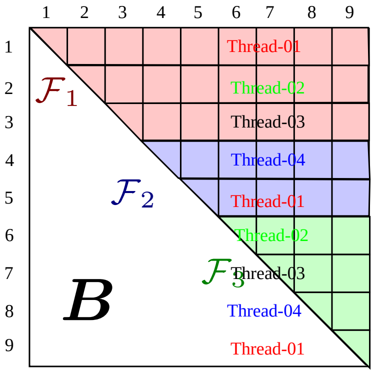
- 4 cores available

Row-wise distribution



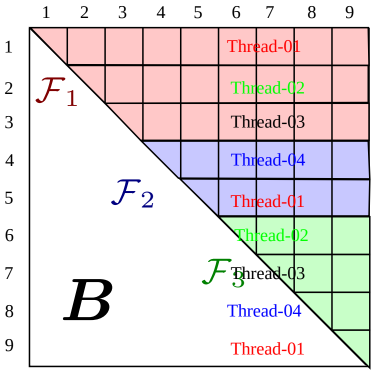
- 4 cores available
- Assign threads in cyclic manner

Row-wise distribution



- 4 cores available
- Assign threads in cyclic manner
- No communication between threads
- Each thread concentrates its tasks

Row-wise distribution



- 4 cores available
- Assign threads in cyclic manner
- No communication between threads
- Each thread concentrates its tasks
- Excellent speed-up

Computation Time and Scalability

problem	threads	ELEMENTS	CHOLESKY	Total
thetaG51	1	134.7 (1.0)	308.6 (1.0)	494.9 (1.0)
	8	26.2 (5.1)	45.8 (6.8)	86.8 (5.7)
rambo	1	56.3 (1.0)	88.1 (1.0)	146.6 (1.0)
	8	11.2 (5.0)	13.5 (6.5)	28.8 (5.1)
control11	1	77.3 (1.0)	6.8 (1.0)	86.1 (1.0)
	8	29.6 (2.6)	4.1 (1.7)	35.9 (2.4)

second(speed-up), SDPA7, Xeon X5550 2.67GHz x2

Sparsity of SCM

In some applications (like Polynomial Optimization),
the Schur complement matrix becomes sparse.

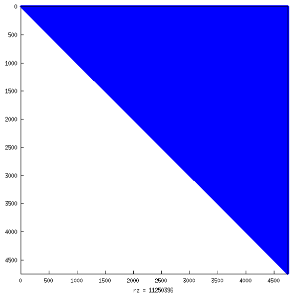


Figure: Fully dense (Quantum Chem, 100%)

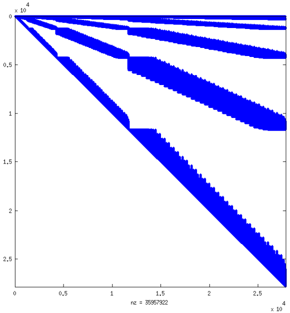


Figure: Sparse (POP, 9.26%)

Sparsity of SCM comes from *the diagonal block structure*.

Diagonal Block Structure

Input data matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ can be decomposed into same size sub-matrices.

$$\boldsymbol{A}_k = diag(\boldsymbol{A}_k^1, \boldsymbol{A}_k^2, \ldots, \boldsymbol{A}_k^s) = \begin{pmatrix} \boldsymbol{A}_k^1 & \boldsymbol{O} & \ldots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{A}_k^2 & \ldots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \ldots & \boldsymbol{A}_k^s \end{pmatrix}$$

The number of sub-matrices s sometimes $s > 1000$.

Diagonal Block Structure

Input data matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ can be decomposed into same size sub-matrices.

$$\boldsymbol{A}_k = diag(\boldsymbol{A}_k^1, \boldsymbol{A}_k^2, \ldots, \boldsymbol{A}_k^s) = \begin{pmatrix} \boldsymbol{A}_k^1 & \boldsymbol{O} & \ldots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{A}_k^2 & \ldots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \ldots & \boldsymbol{A}_k^s \end{pmatrix}$$

The number of sub-matrices s sometimes $s > 1000$. Then, the variable matrices $\boldsymbol{X}, \boldsymbol{Y}$ can also be decomposed.

$$\boldsymbol{X} = diag(\boldsymbol{X}^1, \boldsymbol{X}^2, \ldots, \boldsymbol{X}^s), \boldsymbol{Y} = diag(\boldsymbol{Y}^1, \boldsymbol{Y}^2, \ldots, \boldsymbol{Y}^s)$$

Diagonal Block Structure

Input data matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ can be decomposed into same size sub-matrices.

$$\boldsymbol{A}_k = diag(\boldsymbol{A}_k^1, \boldsymbol{A}_k^2, \ldots, \boldsymbol{A}_k^s) = \begin{pmatrix} \boldsymbol{A}_k^1 & \boldsymbol{O} & \ldots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{A}_k^2 & \ldots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \ldots & \boldsymbol{A}_k^s \end{pmatrix}$$

The number of sub-matrices s sometimes $s > 1000$. Then, the variable matrices $\boldsymbol{X}, \boldsymbol{Y}$ can also be decomposed.

$$\boldsymbol{X} = diag(\boldsymbol{X}^1, \boldsymbol{X}^2, \ldots, \boldsymbol{X}^s), \boldsymbol{Y} = diag(\boldsymbol{Y}^1, \boldsymbol{Y}^2, \ldots, \boldsymbol{Y}^s)$$

$$\begin{aligned} B_{ij} &= (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j \\ &= \sum_{l=1}^{s} (\boldsymbol{X}^l \boldsymbol{A}_i^l (\boldsymbol{Y}^l)^{-1}) \bullet \boldsymbol{A}_j^l \end{aligned}$$

Diagonal Block Structure

Input data matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ can be decomposed into same size sub-matrices.

$$\boldsymbol{A}_k = diag(\boldsymbol{A}_k^1, \boldsymbol{A}_k^2, \ldots, \boldsymbol{A}_k^s) = \begin{pmatrix} \boldsymbol{A}_k^1 & \boldsymbol{O} & \ldots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{A}_k^2 & \ldots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \ldots & \boldsymbol{A}_k^s \end{pmatrix}$$

The number of sub-matrices s sometimes $s > 1000$. Then, the variable matrices $\boldsymbol{X}, \boldsymbol{Y}$ can also be decomposed.

$$\boldsymbol{X} = diag(\boldsymbol{X}^1, \boldsymbol{X}^2, \ldots, \boldsymbol{X}^s), \boldsymbol{Y} = diag(\boldsymbol{Y}^1, \boldsymbol{Y}^2, \ldots, \boldsymbol{Y}^s)$$

$$\begin{aligned} B_{ij} &= (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j \\ &= \sum_{l=1}^{s} (\boldsymbol{X}^l \boldsymbol{A}_i^l (\boldsymbol{Y}^l)^{-1}) \bullet \boldsymbol{A}_j^l \end{aligned}$$

If $\#\boldsymbol{A}_i^l = 0$ or $\#\boldsymbol{A}_j^l = 0$ through $l = 1, \ldots, s$, then $B_{ij} = 0$.

Sparse Cholesky

If $\#\boldsymbol{A}_i^l = 0$ or $\#\boldsymbol{A}_j^l = 0$ through $l = 1, \ldots, s$, then $B_{ij} = 0$.

Sparse Cholesky

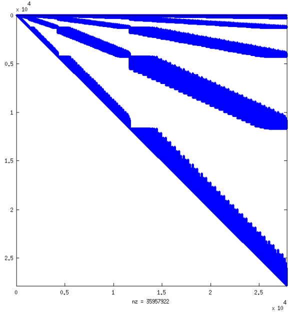
If $\#\boldsymbol{A}_i^l = 0$ or $\#\boldsymbol{A}_j^l = 0$ through $l = 1, \dots, s$, then $B_{ij} = 0$.
Nonzero pattern of $\boldsymbol{B} \in \mathbb{S}_+^m$ is

$$\mathcal{S}(\boldsymbol{B}) = \cup_{l=1}^{s} \{(i,j) : \#\boldsymbol{A}_i^l \neq 0 \text{ and } \#\boldsymbol{A}_j^l \neq 0\}$$

Sparse Cholesky

If $\#\boldsymbol{A}_i^l = 0$ or $\#\boldsymbol{A}_j^l = 0$ through $l = 1, \ldots, s$, then $B_{ij} = 0$.
Nonzero pattern of $\boldsymbol{B} \in \mathbb{S}_+^m$ is

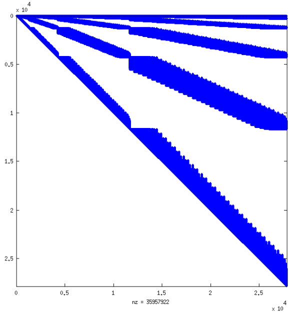
$$\mathcal{S}(\boldsymbol{B}) = \cup_{l=1}^{s} \{(i,j) : \#\boldsymbol{A}_i^l \neq 0 \text{ and } \#\boldsymbol{A}_j^l \neq 0\}$$



Sparse Cholesky

If $\#\boldsymbol{A}_i^l = 0$ or $\#\boldsymbol{A}_j^l = 0$ through $l = 1, \ldots, s$, then $B_{ij} = 0$.
Nonzero pattern of $\boldsymbol{B} \in \mathbb{S}_+^m$ is

$$\mathcal{S}(\boldsymbol{B}) = \cup_{l=1}^{s}\{(i,j) : \#\boldsymbol{A}_i^l \neq 0 \text{ and } \#\boldsymbol{A}_j^l \neq 0\}$$

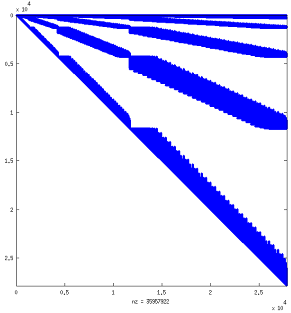


To solve the Schur complement equation $\boldsymbol{B}d\boldsymbol{z} = \boldsymbol{r}$,
we apply sparse Cholesky factorization $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^T$.

Sparse Cholesky

If $\#\boldsymbol{A}_i^l = 0$ or $\#\boldsymbol{A}_j^l = 0$ through $l = 1, \ldots, s$, then $B_{ij} = 0$.
Nonzero pattern of $\boldsymbol{B} \in \mathbb{S}_+^m$ is

$$\mathcal{S}(\boldsymbol{B}) = \cup_{l=1}^{s}\{(i,j) : \#\boldsymbol{A}_i^l \neq 0 \text{ and } \#\boldsymbol{A}_j^l \neq 0\}$$



To solve the Schur complement equation $\boldsymbol{B}d\boldsymbol{z} = \boldsymbol{r}$,
we apply sparse Cholesky factorization $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^T$.
We employ MUMPS developed by Amestoy *et al.*
MUMPS implements multifrontal method.

Selection of Dense or Sparse

- Through all the iteration, $\mathcal{S}(\boldsymbol{B})$ is invariant.
- Before the iteration, we build $\mathcal{S}(\boldsymbol{B})$.

Selection of Dense or Sparse

- Through all the iteration, $\mathcal{S}(\boldsymbol{B})$ is invariant.
- Before the iteration, we build $\mathcal{S}(\boldsymbol{B})$.

If $\frac{\#\mathcal{S}(\boldsymbol{B})}{m^2} > 0.70$, then we use dense; otherwise sparse.

Selection of Dense or Sparse

- Through all the iteration, $\mathcal{S}(\boldsymbol{B})$ is invariant.
- Before the iteration, we build $\mathcal{S}(\boldsymbol{B})$.

If $\frac{\#\mathcal{S}(\boldsymbol{B})}{m^2} > 0.70$, then we use dense; otherwise sparse.
We prepare different memory storage for $\boldsymbol{B} \in \mathbb{S}^n_+$

- Dense data format
  We use one-dimensional storage.
  $B_{11}, B_{12}, \ldots, B_{1n}, B_{21}, \ldots, B_{2n}, \cdots, B_{n1}, \ldots, B_{nn}$
- Sparse data format
  We use the set of tuples.
  $(1, 1, B_{1,1}), (1, 10, B_{1,10}), (1, 11, B_{1,11}), (2, 2, B_{2,2}), \ldots, (n, n, B_{n,n})$

Effect of Sparse Cholesky

problem	sfsdp500 (Sensor Network Localization)			
SCM density	0.94 %			
	ELEMENTS	CHOLESKY	Total	Iter
Dense	9.83	74.94	100.09	34
Sparse	2.48	4.33	13.54	34
problem	BroydenTri500 (Polynomial Optimization)			
SCM density	0.48 %			
	ELEMENTS	CHOLESKY	Total	Iter
Dense	10.24	2744.45	2744.20	23
Sparse	2.72	3.33	12.13	22
problem	theta6 (Combinatorial Optimization)			
SCM density	100 %			
	ELEMENTS	CHOLESKY	Total	Iter
Dense	10.45	7.79	20.70	18
Sparse	10.45	7.80	20.71	18

Time unit is *second*, SDPA7, Xeon 5460 3.16 GHz, 48GB meory

Comparison with Other Software Packages

Other software packages based on PDIPM

- CSDP (Borcher,https://projects.coin-or.org/Csdp/)
- SDPT3 (Toh et al, http://www.math.nus.edu.sg/~mattohkc/sdpt3.html)
- SeDuMi (Sturm, http://sedumi.ie.lehigh.edu/)

name	m	nBlock	BlockSize
Quantum Chemistry			
NH3	2964	22	$(744 \times 2, 224 \times 4, \ldots, 1 \times 154)$
Be	4743	22	$(1062 \times 2, 324 \times 4, \ldots, 1 \times 190)$
Sensor Network Localization			
d2s4Kn0r01a4	31630	3885	$(43 \times 2, 36 \times 1, \ldots, 1 \times 31392)$
s5000n0r05g2FD_R	33061	4631	$(73 \times 1, 65 \times 1, 64 \times 2)$

Results with Other Software Packages

name		SDPA	CSDP	SDPT3	SeDuMi
NH3	time	495.3	5675.6	4882.9	1375.1
	memory	1004	568	3676	4065
Be	time	2238.3	15592.3	15513.8	5550.4
	memory	1253	744	3723	4723
d2s4Kn0r01a4	time	45.7	5162.4	4900.3	92.6
	memory	1093	8006	63181	3254
s5000n0r05g2FD_R	time	284.7	6510.9	4601.6	1005.0
	memory	2127	8730	100762	4914

Time unit is *second*, Memory is *Mega Bytes*,
Xeon 5550 (2.66GHz) x2, 72GB memory.

- For Quantum Chem, $\mathcal{F}_3$ and Multiple Threading are effective.
- For SNL, Sparse Cholesky is very important.

Results with Other Software Packages

name		SDPA	CSDP	SDPT3	SeDuMi
NH3	time	495.3	5675.6	4882.9	1375.1
	memory	1004	568	3676	4065
Be	time	2238.3	15592.3	15513.8	5550.4
	memory	1253	744	3723	4723
d2s4Kn0r01a4	time	45.7	5162.4	4900.3	92.6
	memory	1093	8006	63181	3254
s5000n0r05g2FD_R	time	284.7	6510.9	4601.6	1005.0
	memory	2127	8730	100762	4914

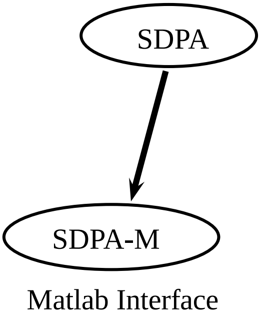
Time unit is *second*, Memory is *Mega Bytes*,
Xeon 5550 (2.66GHz) x2, 72GB memory.

- For Quantum Chem, $\mathcal{F}_3$ and Multiple Threading are effective.
- For SNL, Sparse Cholesky is very important.

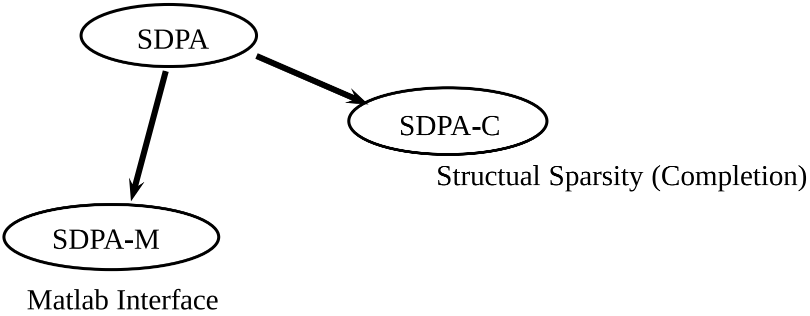
SDPA is *the fastest solver for large-scale SDPs.*

4. SDPA Family [SDPA, SDPA-M, SDPA-C, SDPARA, SDPARA-C, SDPA-GMP]

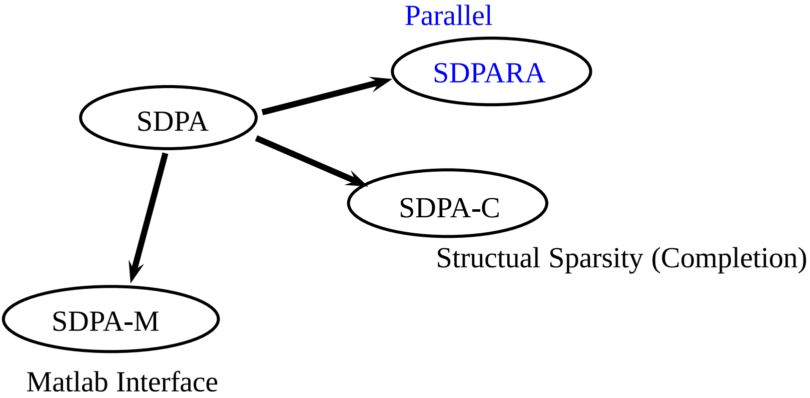
4. SDPA Family [SDPA, SDPA-M, SDPA-C, SDPARA, SDPARA-C, SDPA-GMP]



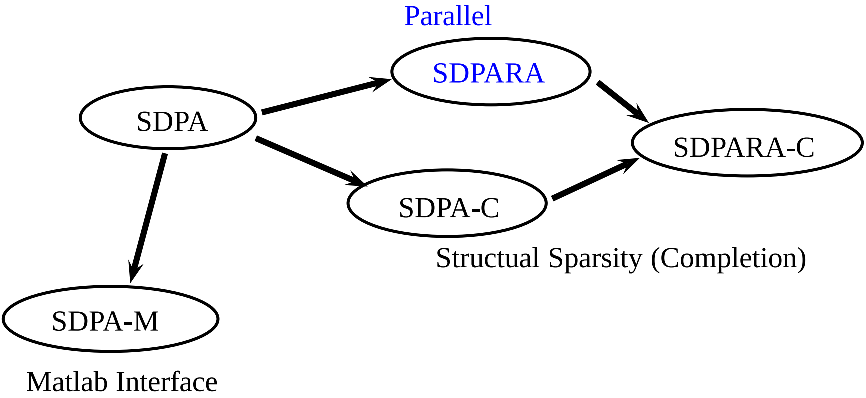
4. SDPA Family [SDPA, SDPA-M, SDPA-C, SDPARA, SDPARA-C, SDPA-GMP]



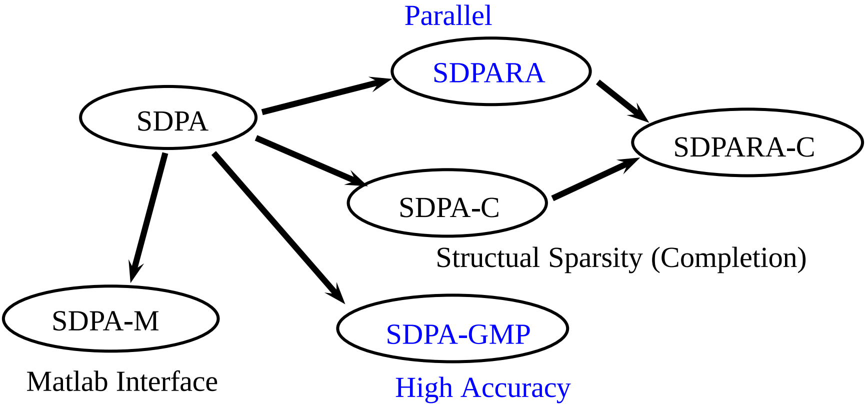
4. SDPA Family [SDPA, SDPA-M, SDPA-C, SDPARA, SDPARA-C, SDPA-GMP]



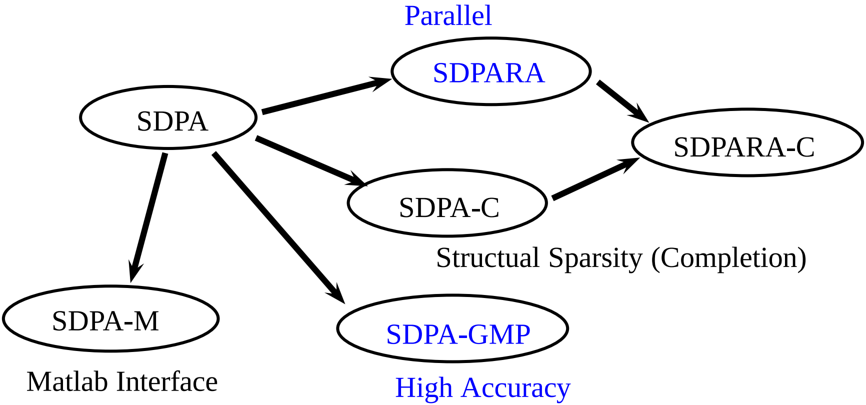
4. SDPA Family [SDPA, SDPA-M, SDPA-C, SDPARA, SDPARA-C, SDPA-GMP]



4. SDPA Family [SDPA, SDPA-M, SDPA-C, SDPARA, SDPARA-C, SDPA-GMP]



4. SDPA Family [SDPA, SDPA-M, SDPA-C, SDPARA, SDPARA-C, SDPA-GMP]



In this talk, SDPARA and SDPA-GMP are discussed.

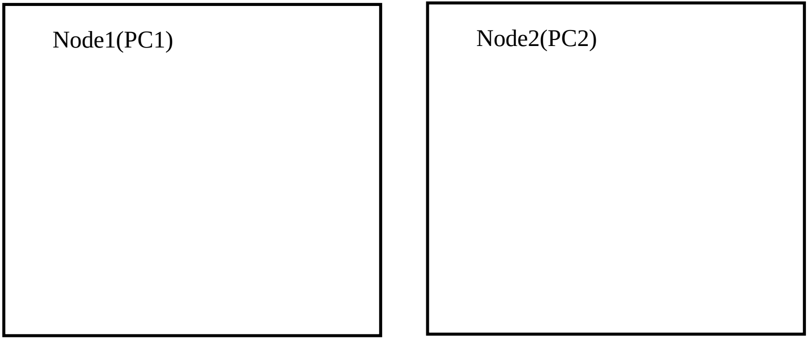
SDPARA

A parallel version of SDPA designed for extremely large-scale SDPs.

- Dense Schur complement matrix
  - (ELEMENTS) Hybrid Parallel [MPI & Thread]
  - (CHOLESKY) Two-Dimensional Block-Cyclic Distribution
- Sparse Schur complement matrix
  - (ELEMENTS) Formula-Cost-Based Distribution
  - (CHOLESKY) Multiple-frontal method by MUMPS

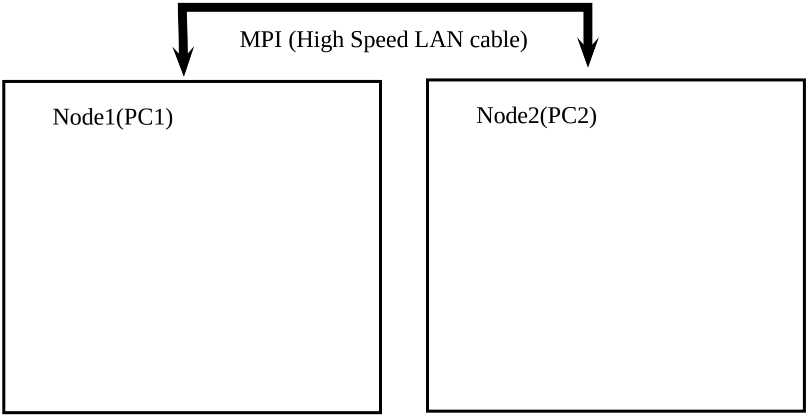
MPI & Thread

- We connect all PCs by MPI (Message Passing Interface).
- Each PC has multiple CPU.
- Each CPU has multiple threads.



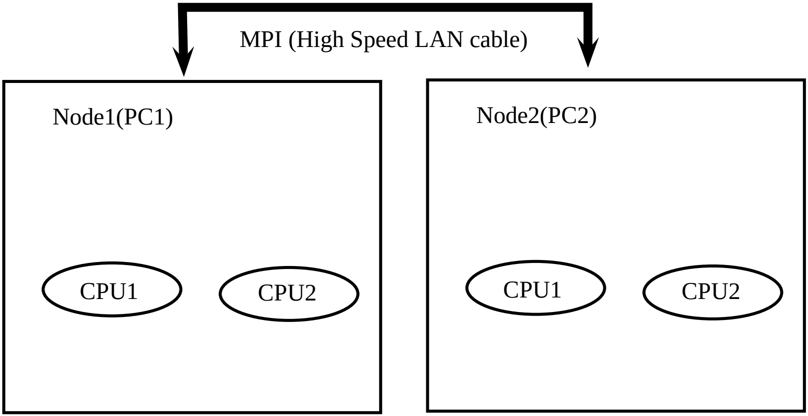
MPI & Thread

- We connect all PCs by MPI (Message Passing Interface).
- Each PC has multiple CPU.
- Each CPU has multiple threads.



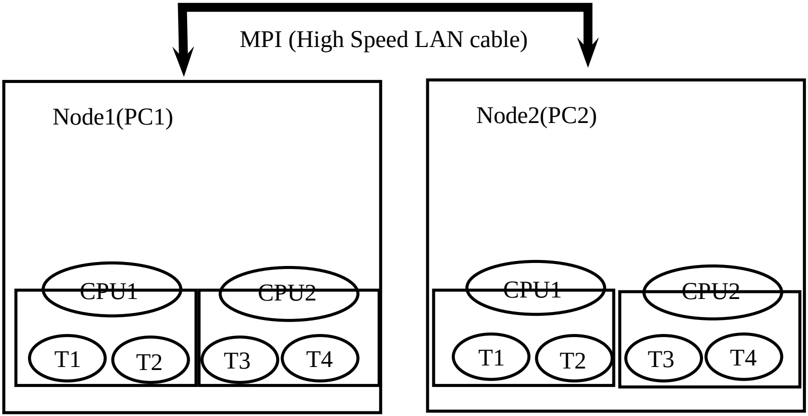
MPI & Thread

- We connect all PCs by MPI (Message Passing Interface).
- Each PC has multiple CPU.
- Each CPU has multiple threads.



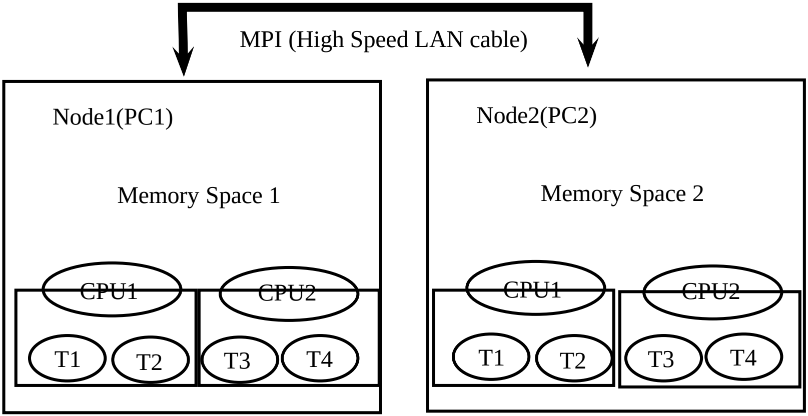
MPI & Thread

- We connect all PCs by MPI (Message Passing Interface).
- Each PC has multiple CPU.
- Each CPU has multiple threads.



MPI & Thread

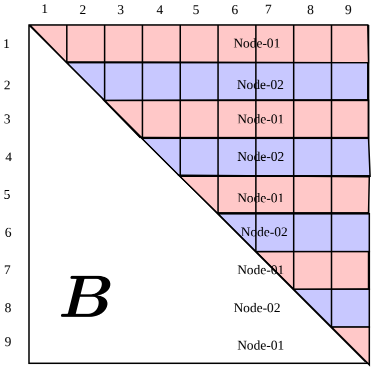
- We connect all PCs by MPI (Message Passing Interface).
- Each PC has multiple CPU.
- Each CPU has multiple threads.



Hybrid Parallel

The case of 2 Nodes × 1 CPU × 2 Threads.

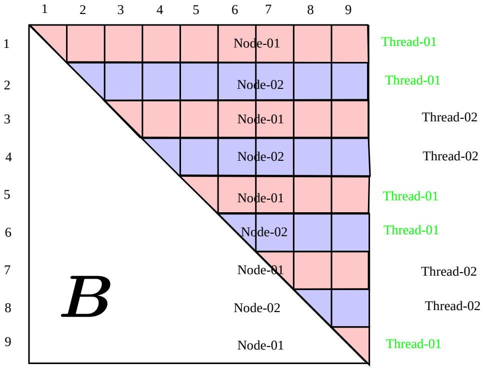
1. Assign the rows to the nodes.



Hybrid Parallel

The case of 2 Nodes $\times$ 1 CPU $\times$ 2 Threads.

1. Assign the rows to the nodes.
2. Assign the rows to the threads in each node.



Parallel Cholesky factorization

- *Parallel* Cholesky factorization of ScaLAPACK.
- To enhance the performance, we redistribute the SCM.

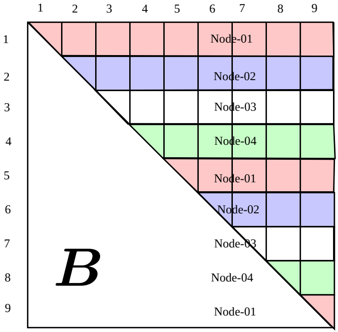


Figure: Row-wise distribution

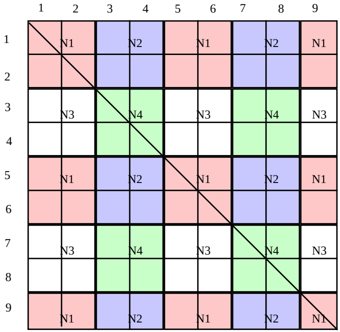
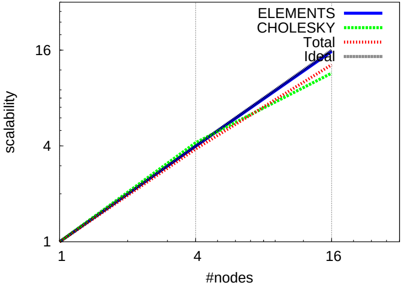


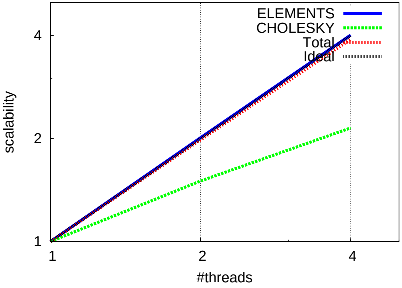
Figure: Two-Dimensional Block-Cyclic Distribution

Scalability of SDPARA

	Nodes			Threads		
	1	4	16	1	2	4
ELEMENTS	28678	7192	1826	28678	14252	7143
CHOLESKY	548	131	47	548	365	255
Total	29700	7764	2294	29700	14981	7613

Time unit is *second*





SDP: B.2P ($m = 7230, n = 5990, n_{\max} = 1450, \mathrm{SCM} = 100\%$)
SDPARA 7.3.1, Xeon X5460, 3.16GHz x2, 48GB memory

MPI & Multi Threading

#threads \ #node	1	2	4	8	16
1	36206	18134	9190	4729	2479
8	5983	2002	1680	901	565

Time unit is *second*
SDP: B.2P ($m = 7230, n = 6010, n_{\max} = 1460, \mathrm{SCM} = 100\%$)
SDPARA 7.3.1, Xeon X5460, 3.16GHz x2, 48GB memory

Computation time is reduced from
36206 seconds (1nodes $\times$ 1threads) to 565 seconds (16nodes $\times$ 8threads).
$\Rightarrow$ 64$\times$ speed-up

Formula-cost-based Distribution for Sparse SCM

$\#\boldsymbol{A}_i$ enables us to estimate computation cost of $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$.
We distribute the computation to the nodes based on this estimation.

	1	2	3	4	5	6
1	150		40	30		20
2		135			20	
3			70		10	
4				50		5
5		$\boldsymbol{B}$			30	
6						3

Load on each CPU
Node1 : 190
Node2 : 185
Node3 : 188

Parallel Cholesky factorization for Sparse SCM

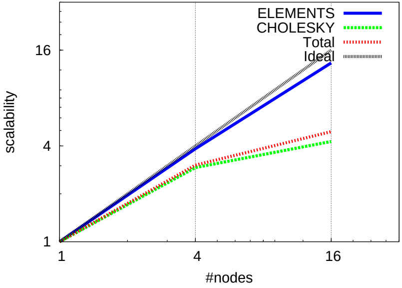
- Parallel Sparse Cholesky factorization from MUMPS.
- Memory storage on each processor should be consecutive in row-wise.
- The distribution for ELEMENTS matches this method.

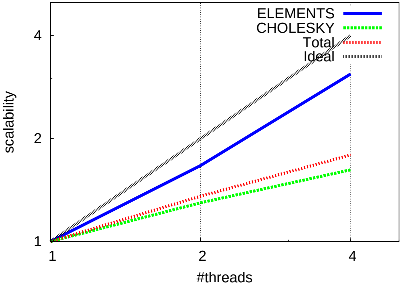
	1	2	3	4	5	6
1	150		40	30		20
2		135			20	
3			70		10	
4				50		5
5		B			30	
6						3

Numerical Results on Sparse SCM

	Nodes			Threads		
	1	4	16	1	2	4
ELEMENTS	1137	296	85	1137	682	368
CHOLESKY	4053	1386	950	4053	3122	2500
Total	5284	1744	1074	5284	3895	2949

Time unit is *second*

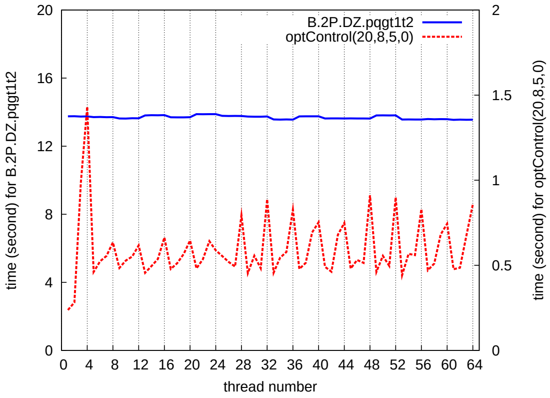




SDP: optControl(20,8,5,0) ($m = 109,246, n = 33847, n_{\max} = 136$, SCM = 4.39%)

SDPARA 7.3.1, Xeon X5460, 3.16GHz x2, 48GB memory

Load-balance on ELEMENTS



- For B.2P, $\frac{\max}{\min} = \frac{13.88}{13.54} = 1.02 \Rightarrow 62\times$ speed-up on 64 threads.
- For optControl, $\frac{\max}{\min} = \frac{1.43}{0.23} = 6.21 \Rightarrow 35\times$ speed-up on 64 threads.

Numerical Experiments on Large-Scale SDPs

- Comparison with PCSDP
  developed by Ivanov and deKlerk based on CSDP by Borcher
- Dense SCM
  Quantum Chemistry
- Sparse SCM
  Sensor Network Localization Problem generated by SFSDP

SDPARA vs PCSDP on Dense SCM

#threads \ #node	1	2	4	8	16
PCSDP(8)	53763	27854	14273	7995	4050
SDPARA(1)	36206	18134	9190	4729	2479
SDPARA(8)	5983	2002	1680	901	565

Time unit is *second*

SDP: B.2P ($m = 7230, n = 6010, n_{\max} = 1460$, SCM $= 100\%$)

SDPARA 7.3.1, Xeon X5460, 3.16GHz x2, 48GB memory

- SDPARA is much faster than PCSDP.
- SDPARA *combines* MPI and Multi-threading.
- Scalability of SDPARA is better than PCSDP.

SDPARA vs PCSDP on Sparse SCM

#sensors	1,000 (m=16450; density=1.23%)				
#Nodes	1	2	4	8	16
PCSDP	O.M.	1527	887	591	368
SDPARA	28.2	22.1	16.7	13.8	27.3

#sensors	35,000 (m=527096; density=6.53×10^{-3}%)				
#Nodes	1	2	4	8	16
PCSDP	Out of memory if #sensors $\geq$ 4000				
SDPARA	1080	845	614	540	506

- Sparse Cholesky has a great impact.
- SDPARA can solve extremely large-scale SDPs in a short time.

SDPA-GMP

- PD-IPM usually attains numerical accuracy $10^{-6} \sim 10^{-8}$.
- Some applications require 10^{-30}.
- *double* precision in C language (approx. 10^{-16}) is NOT enough.
- GMP (Gnu Multiple Precision) library can handle arbitrary accuracy.

SDPA replaces BLAS (Matrix manipulation libraries) by MLAPACK library with the help of GMP.

http://mplapack.sourceforge.net/

Here, we report the numerical results of SDPA-GMP on Graph Partition Problem.

A perturbation on GPP (Graph Partition Problem)

$$[GPP] \quad \min \ \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t. } \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} = 0, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succeq \boldsymbol{O}$$

A perturbation on GPP (Graph Partition Problem)

$$[GPP] \quad \min \ \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t. } \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} = 0, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succeq \boldsymbol{O}$$

GPP does not satisfy the Slater's condition.

$$\{\boldsymbol{X} \in \mathbb{S}^n : \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} = 0, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succ \boldsymbol{O}\} = \emptyset$$

PDIPM gets into *numerical instability* near optimal solution.

A perturbation on GPP (Graph Partition Problem)

$$[GPP] \quad \min\ \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t. } \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} = 0, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succeq \boldsymbol{O}$$

GPP does not satisfy the Slater's condition.

$$\{\boldsymbol{X} \in \mathbb{S}^n : \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} = 0, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succ \boldsymbol{O}\} = \emptyset$$

PDIPM gets into *numerical instability* near optimal solution.
The $\bar{\epsilon}$-perturbed problem

$$[GPP(\bar{\epsilon})] \quad \min\ \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t. } \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} \le \bar{\epsilon}, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succeq \boldsymbol{O}$$

has an interior, for example, $\boldsymbol{X} = (1-a)\boldsymbol{I} + a\boldsymbol{e}\boldsymbol{e}^T$ with $a = \frac{\frac{\bar{\epsilon}}{2} - n}{n(n-1)}$.

A perturbation on GPP (Graph Partition Problem)

$$[GPP] \quad \min\ \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t. } \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} = 0, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succeq \boldsymbol{O}$$

GPP does not satisfy the Slater's condition.

$$\{\boldsymbol{X} \in \mathbb{S}^n : \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} = 0, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succ \boldsymbol{O}\} = \emptyset$$

PDIPM gets into *numerical instability* near optimal solution.
The $\bar{\epsilon}$-perturbed problem

$$[GPP(\bar{\epsilon})] \quad \min\ \boldsymbol{C} \bullet \boldsymbol{X} \quad \text{s.t. } \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} \leq \bar{\epsilon}, \boldsymbol{e}_i\boldsymbol{e}_i^T \bullet \boldsymbol{X} = 1 (i = 1, \ldots, n), \boldsymbol{X} \succeq \boldsymbol{O}$$

has an interior, for example, $\boldsymbol{X} = (1-a)\boldsymbol{I} + a\boldsymbol{e}\boldsymbol{e}^T$ with $a = \frac{\frac{\bar{\epsilon}}{2} - n}{n(n-1)}$.
We can control the numerical hardness by the parameter $\bar{\epsilon}$.
The small $\rho_p(d)$ in Toh's paper indicates the closeness to the infeasible region.

$\bar{\epsilon}$	1e-1	1e-4	1e-7	1e-10	1e-15	0
$\rho_p(d)$	2.0e-4	2.0e-7	4.9e-9	4.8e-9	4.8e-9	0

Numerical Results on GPP

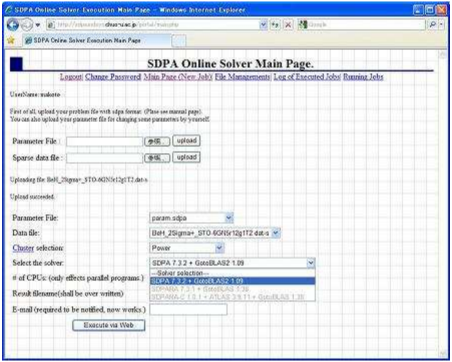
$\bar{\epsilon}$		CSDP	SDPT3	SeDuMi	SDPA	GMP
1.0e-1	Accuracy	3.95e-9	1.21e-11	2.16e-10	1.08e-8	4.80e-48
	Time	5.67	6.42	245.86	2.03	77760.19
	Iter	23	21	27	19	206
1.0e-7	Accuracy	4.37e-9	1.11e-6	1.48e-4	4.27e-7	4.21e-48
	Time	6.57	6.64	303.81	2.37	78385.60
	Iter	26	20	30	23	209
1.0e-15	Accuracy	2.76e-8	3.83e-9	1.08e-4	1.63e-7	2.97e-48
	Time	5.50	6.26	332.12	2.26	82115.52
	Iter	27	19	33	21	219

Accuracy is the maximum of DIMACS errors.
Time unit is second.
GMP uses 300 digits.

- SDPA-GMP needs long-long-long computation time.
- SDPA-GMP attains extremely high accuracy.

SDPA Online Solver

- SDPA and SDPA Family are useful to solve SDPs.
- However, its install is not easy (In particular, optimized BLAS).
- SDPARA requires a parallel computing environment.



- SDPA
- SDPARA (on PC-cluster)
- SDPARA-C (on PC-cluster)

Conclusion

- SDP (SemiDefinite Programs) has many applications.
- SDPA is the fastest solver (on single) for large-scale SDPs.
- SDPARA can solve the largest SDPs.
- SDPA-GMP obtains high accuracy.

http://sdpa.sourceforge.net/

Thank you very much for your attention.