WHERE **INNOVATION** BEGINS

DESIGN AUTOMATION CONFERENCE

JULY 9–13, 2023

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

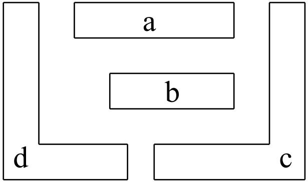
LRSDP: Low-Rank SDP for Triple Patterning Lithography Layout Decomposition

Yu Zhang[1,2], **Yifan Chen**[1], Zhonglin Xie[1], Hong Xu[2], Zaiwen Wen[1], Yibo Lin[1,3], Bei Yu[2]

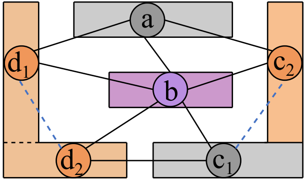
[1]Peking University
[2]Chinese University of Hong Kong
[3]Institute of Electronic Design Automation, Peking University

Background

Due to the availability issue of EUV (Extreme Ultra-Violet) lithography, triple patterning layout (TPL) decomposition is widely adopted in advanced technology nodes[1].



Input features.



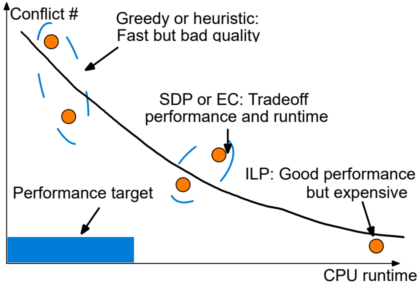
TPL decomposition.

[1] W. Li *et al.*, "Openmpl: An open-source layout decomposer", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2331–2344, 2020.

Background

Existing MPL decomposition studies follow a two-step procedure:

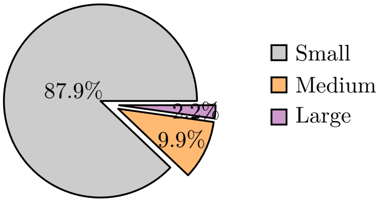
1. Graph simplification
2. Subgraph Decomposition
   - ILP, SDP, EC, etc



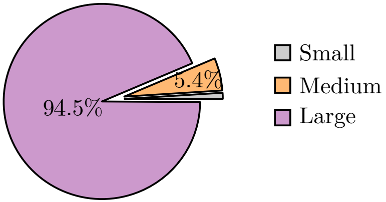
Comparison of current TPL decomposers

Observation

However, these methods usually assume the subgraphs are small, i.e., fewer than 100 vertices. When the design complexity increases, the sizes of subgraphs also boost, and large subgraphs generally take >90% of runtime.



The proportion of small, medium, and large subgraphs after graph simplification.



The time ratio spent on solving the TPL decomposition for these subgraphs.

Contribution

We propose `LRSDP`, a scalable low-rank SDP solver for the MPL decomposition problem on large graphs. Our SDP solver mainly includes three parts:

1. Low-rank factorization
2. Augmented Lagrangian method (ALM)
3. Riemannian gradient descent method with Barzilai-Borwein steps (RGBB)

Preliminary of SDP based TPL

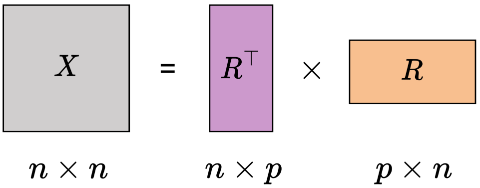
The vector program for TPL is NP-hard due to the discrete constraint. The problem can be further relaxed as the following semidefinite program[2],

$$\begin{aligned} \min_{\boldsymbol{X} \in \mathbb{R}^{n\times n}} \quad & \langle \boldsymbol{C}, \boldsymbol{X} \rangle \\ \text{s.t.} \quad & x_{ii} = 1, \quad \forall i \in V \\ & x_{ij} \geq -\frac{1}{2}, \quad \forall e_{ij} \in \text{CE} \\ & \boldsymbol{X} \succeq 0, \end{aligned} \tag{1}$$

[2]B. Yu *et al.*, "Layout decomposition for triple patterning lithography",, 2011, pp. 1–8.

Low-rank Factorization

It has been recognized that the solution to the SDP program relaxed by combinatorial optimization is often low-rank[3], so we perform low-rank factorization to our SDP program:



[3] Y. Wang *et al.*, "A decomposition augmented lagrangian method for low-rank semidefinite programming", *arXiv preprint arXiv:2109.11707*, 2021.

Low-rank Factorization

Original SDP program[4]:

$$
\begin{aligned}
\min_{\boldsymbol{X} \in \mathbb{R}^{n \times n}} \quad & \langle \boldsymbol{C}, \boldsymbol{X} \rangle \\
\text{s.t.} \quad & x_{ii} = 1, \quad \forall i \in V \\
& x_{ij} \geq -\frac{1}{2}, \quad \forall e_{ij} \in \text{CE} \\
& \boldsymbol{X} \succeq 0,
\end{aligned} \tag{2}
$$

$\longrightarrow$

After low-rank factorization:

$$
\begin{aligned}
\min_{\boldsymbol{R} \in \mathbb{R}^{p \times n}} \quad & \langle \boldsymbol{C}, \boldsymbol{R}^\top \boldsymbol{R} \rangle \\
\text{s.t.} \quad & \|\boldsymbol{r_i}\|_2 = 1, \quad \forall i \in V \\
& \boldsymbol{r_i}^\top \boldsymbol{r_j} \geq -\frac{1}{2}, \quad \forall e_{ij} \in \text{CE},
\end{aligned} \tag{3}
$$

[4] B. Yu *et al.*, "Layout decomposition for triple patterning lithography",, 2011, pp. 1–8.

Benefits of Low-rank Factorization

1. The semidefinite constraint can be naturally omitted as the factorization implies it;
2. This low-rank factorization leads to many fewer variables in the problem of interest, as $p \ll n$;
3. Although the new formula becomes nonlinear and nonconvex, the global optimality of the solution to this factorized version can still be ensured by properly choosing p[5].

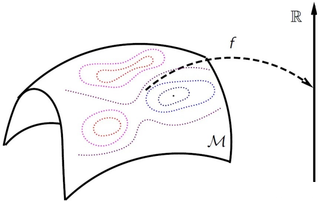
[5] N. Boumal *et al.*, "The non-convex burer-monteiro approach works on smooth semidefinite programs", *nips*, vol. 29, 2016.

Preliminary of Riemannian optimization

Riemannian optimization is used to solve optimization problems with manifold structure constraints:

$$\min_{x \in \mathcal{M}} f(x), \tag{4}$$

where $\mathcal{M}$ refers to manifold constraints and $f : \mathcal{M} \to R$ is a smooth cost function.



Some examples of smooth manifold constraints:

$$\mathcal{M} = \{X \in \mathbb{R}^{n \times n} \mid \mathrm{tr}(X) = 1,\ X \geq 0\}$$

$$\mathcal{M} = \{X \in \mathbb{R}^{n \times n} \mid X_{i,i} = 1,\ X \geq 0\}$$

$$\mathcal{M} = \{X \in \mathbb{R}^{m \times n} \mid \|x_i\|_2 = 1\}$$

Riemannian Manifold in SDP for TPL

We further restrict the optimization of variable $\boldsymbol{R}$ from $\mathbb{R}^{p\times n}$ to a smooth Riemannian manifold: $\mathcal{M} = \{\boldsymbol{R} \in \mathbb{R}^{p\times n} | \boldsymbol{R} = [\boldsymbol{r_1}, \cdots, \boldsymbol{r_n}], \|\boldsymbol{r_i}\|_2 = 1\}$, which is exactly a unit sphere.

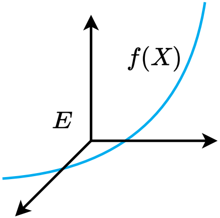
$$\begin{aligned} \min_{\boldsymbol{R}\in\mathbb{R}^{p\times n}} \quad & \langle \boldsymbol{C}, \boldsymbol{R}^\top \boldsymbol{R} \rangle \\ \text{s.t.} \quad & \|\boldsymbol{r_i}\|_2 = 1, \quad \forall i \in V \\ & \boldsymbol{r_i}^\top \boldsymbol{r_j} \geq -\frac{1}{2}, \quad \forall e_{ij} \in \text{CE}, \end{aligned} \tag{5}$$

$$\longrightarrow$$

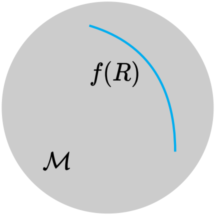
$$\begin{aligned} \min_{\boldsymbol{R}\in\mathcal{M}} \quad & \langle \boldsymbol{C}, \boldsymbol{R}^\top \boldsymbol{R} \rangle \\ \text{s.t.} \quad & \boldsymbol{r_i}^\top \boldsymbol{r_j} \geq -\frac{1}{2}, \quad \forall e_{ij} \in \text{CE}, \end{aligned} \tag{6}$$

Benefits of Riemannian Optimization

- Guarantee the satisfiability of the unit length constraint.
- Utilize the structure information of $\mathcal{M}$ to reduce searching space.



Unrestricted search in Euclidean space.



Restricted search in Riemannian space.

Eliminate Inequality Constraint

Now, the only difficult constraint is the inequality constraint. Here we introduce an auxiliary variable $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ to remove the inequality constraint, so the factorized SDP is reformulated as:

$$\begin{aligned} \min_{\boldsymbol{R} \in \mathcal{M}} \quad & \langle \boldsymbol{C}, \boldsymbol{R}^\top \boldsymbol{R} \rangle + h(\boldsymbol{W}) \\ \text{s.t.} \quad & \boldsymbol{P} \odot \boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{W}, \end{aligned} \tag{7}$$

where $\odot$ denotes element-wise product, h is a characteristic function, and $\boldsymbol{P}$ encodes the information of conflict edges:

$$h(\boldsymbol{W}) = \begin{cases} 0, & w_{ij} \geq -\frac{1}{2}, \forall e_{ij} \in \text{CE}, \\ +\infty, & \text{otherwise}, \end{cases} \qquad p_{ij} = \begin{cases} 1, & \forall e_{ij} \in \text{CE}, \\ 0, & \forall e_{ij} \notin \text{CE}. \end{cases}$$

Augmented Lagrangian Method

The major framework of `LRSDP` is based on the augmented lagrangian method, which includes an additional term to penalize infeasible points.

The ALM function is denoted by:

$$L_\sigma(\boldsymbol{R}, \boldsymbol{W}, \boldsymbol{y}, \sigma) = \langle \boldsymbol{C}, \boldsymbol{R}^\top \boldsymbol{R} \rangle + h(\boldsymbol{W}) - \langle \boldsymbol{y}, \boldsymbol{P} \odot \boldsymbol{R}^\top \boldsymbol{R} - \boldsymbol{W} \rangle + \frac{\sigma}{2} \|\boldsymbol{P} \odot \boldsymbol{R}^\top \boldsymbol{R} - \boldsymbol{W}\|_F^2, \quad (8)$$

where $\boldsymbol{y} \in \mathbb{R}^{n \times n}$, and $\sigma > 0$ are parameters for ALM. The subproblem in k-th iteration is an unconstrained Riemannian optimization problem:

$$\min_{\boldsymbol{R} \in \mathcal{M}} \Phi_k(\boldsymbol{R}) = \langle \boldsymbol{C}, \boldsymbol{R}^\top \boldsymbol{R} \rangle + h(\boldsymbol{P} \odot \boldsymbol{R}^\top \boldsymbol{R} - \boldsymbol{y}^k / \sigma_k - T(\boldsymbol{R})) + \frac{\sigma_k}{2} \|T(\boldsymbol{R})\|_F^2. \quad (9)$$

Optimal Step Length

Given the unconstrained optimization problem $\min_{x \in \mathbb{R}^n} f(x)$, the optimal step size for the k-th iteration is:

$$\alpha_k = H^{-1}, \tag{10}$$

where H is the Hessian matrix of $f(x_k)$. However, this ideal step length is usually unnecessarily expensive to compute for a general nonlinear cost function f.

Barzilai-Borwein (BB) method

The basic idea of the Barzilai-Borwein (BB) method[6] is to approximate the computationally expensive Hessian matrix. When $s_k^\top y_k > 0$, the BB step-length is

$$\alpha_k^{BB} = \frac{s_k^\top s_k}{s_k^\top y_k}. \tag{11}$$

with $s_k := x_k - x_{k-1}$ and $y_k := \nabla f(x_k) - \nabla f(x_{k-1})$. Then, the BB method performs the iteration: $x_{k+1} = x_k + \alpha_k^{BB} g_k$.

[6] B. Iannazzo and M. Porcelli, "The riemannian barzilai–borwein method with nonmonotone line search and the matrix geometric mean computation", *IMA Journal of Numerical Analysis*, vol. 38, no. 1, pp. 495–517, 2018.

Riemannian Gradient Descent with Barzilai-Borwein Steps

We can compare Euclidean gradient descent with Riemannian gradient descent:

Euclidean optimization:

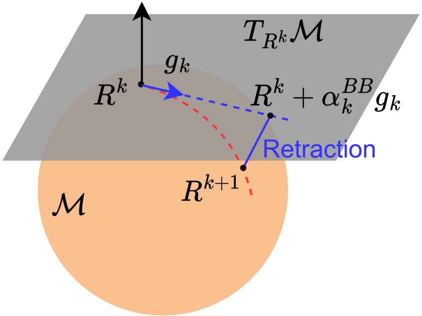
1. Find a descent direction $g_k = -\nabla f(x_k)$;
2. Update $x_{k+1} = x_k + \alpha_k^{BB} g_k$.

Riemannian optimization:

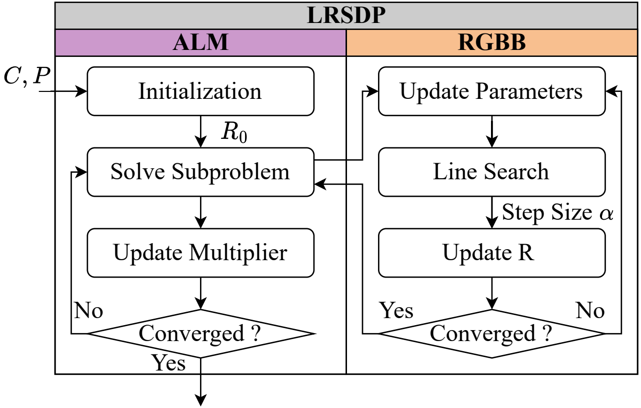
1. Find a descent direction $g_k = -\text{grad} f(x_k) \in T_{x_k}\mathcal{M}$;
2. Update $x_k' = x_k + \alpha_k^{BB} g_k \in T_{x_k}\mathcal{M}$;
3. Retract $x_{k+1} = R_{x_k}(\alpha_k^{BB} g_k) \in \mathcal{M}$.

Riemannian Gradient Descent with Barzilai-Borwein Steps

RGBB is performed to find the optimal solution for the subproblem. The RGBB at k-th step is illustrated as follows:



LRSDP Flow


LRSDP
ALM
RGBB
C, P
Initialization
R0
Solve Subproblem
Update Multiplier
No
Converged ?
Yes
Update Parameters
Line Search
Step Size α
Update R
Yes
No
Converged ?

Experiment result

Table 1. ISPD'19 benchmarks that can be solved by all three decomposers.

test case	Vertices			ILP				CSDP				Ours (RGBB)			
	Total	Mean	Max	conflict	stitch	cost	time/s	conflict	stitch	cost	time/s	conflict	stitch	cost	time/s
test1 100	8073	25	171	241	299	270.9	88.9	269	287	297.7	4.5	262	285	290.5	2.8
test1 101	4398	61	834	78	138	91.8	3739.1	94	134	107.4	34.1	98	141	112.1	4.8
test1 102	109	16	46	1	1	1.1	2.2	1	1	1.1	0.1	1	1	1.1	0.1
test2 100	253454	34	1068	5046	8934	5939.4	22120.4	6439	8179	7256.9	330.1	6456	8202	7276.2	101.2
test2 102	13021	42	2375	213	502	263.2	12243.6	479	475	526.5	579.8	297	486	345.6	28.6
test3 100	21064	92	7060	680	757	755.7	24566.2	1058	1109	1168.9	13577.3	911	733	984.3	168.3
test3 101	8682	71	2858	130	270	157.0	10422.4	196	276	223.6	854.4	194	266	220.6	30.7
test3 102	76	13	26	2	1	2.1	0.1	2	1	2.1	0.0	2	1	2.1	0.0
test5 100	9187	19	781	354	330	387.0	5523.0	396	329	428.9	43.9	402	321	434.1	6.8
test5 101	12515	20	246	467	232	490.2	113.1	527	228	549.8	9.9	496	229	518.9	3.8
test5 102	8265	51	3295	197	174	214.4	7225.2	262	151	277.1	1526.5	238	144	252.4	40.8
test6 102	26540	28	978	115	482	163.2	451.1	144	477	191.7	65.0	150	479	197.9	10.8
test7 100	287412	18	2678	8424	9740	9398.0	36696.6	9020	9509	9970.9	2936.4	9089	9490	10038.0	698.6
test8 100	95194	8	78	5683	4606	6143.6	158.2	5750	4547	6204.7	47.2	5752	4549	6206.9	38.0
test8 101	553934	25	4897	6199	13139	7512.9	52660.6	7275	12741	8549.1	7466.4	7235	12840	8519.0	820.7
test9 100	144539	8	71	8739	6969	9435.9	249.3	8842	6880	9530.0	73.1	8841	6879	9528.9	60.3
test10 100	211030	10	362	9775	9580	10733.0	409.3	9963	9457	10908.7	115.9	9964	9457	10909.7	94.7
average ratio	–	–	–	0.87	1.03	0.89	186.62	1.05	1.03	1.05	12.48	1.00	1.00	1.00	1.00

Experiment result

Table 2. ISPD'19 benchmarks that can't be solved by all three decomposers. Some algorithms crash ('Failed') or exceed the time limit ('TLE').

	Vertices			ILP				CSDP				Ours (RGBB)			
test case	Total	Mean	Max	conflict	stitch	cost	time/s	conflict	stitch	cost	time/s	conflict	stitch	cost	time/s
test2 101	165137	90	3505	TLE	TLE	TLE	TLE	4553	5026	5055.6	12327.0	3837	5124	4349.4	489.5
test4 100	203283	63	20521	TLE	TLE	TLE	TLE	Failed	Failed	Failed	Failed	16377	10559	17432.9	18357.1
test4 101	231944	76	57176	18012	6250	18637.0	30439.1	TLE	TLE	TLE	TLE	12041	7238	12764.8	41421.6
test6 100	632812	28	309	14954	23427	17296.7	11318.6	17657	22134	19870.4	407.9	17596	22215	19817.5	213.8
test6 101	399298	96	25155	TLE	TLE	TLE	TLE	Failed	Failed	Failed	Failed	8851	12238	10074.8	7469.2
test7 101	762019	57	31521	TLE	TLE	TLE	TLE	Failed	Failed	Failed	Failed	13831	18247	15655.7	23700.9
test7 102	314479	92	9473	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE	6480	6192	7099.2	1880.9
test8 102	568566	66	94828	TLE	TLE	TLE	TLE	Failed	Failed	Failed	Failed	97885	8937	98778.7	16016.8
test9 101	911524	25	12887	TLE	TLE	TLE	TLE	24475	21418	26616.8	11375.8	12031	21909	14221.9	2471.8
test9 102	903364	56	49695	TLE	TLE	TLE	TLE	Failed	Failed	Failed	Failed	10015	17668	11781.8	33270.6
test10 101	1304220	38	23389	TLE	TLE	TLE	TLE	Failed	Failed	Failed	Failed	18480	28608	21340.8	9748.4

Conclusion

- Among two SDP-based approaches, our method is 12.48× faster than CSDP on average with 5% lower cost.
- Our method is 186.62× faster than ILP and only increases about 11% cost, which makes a better trade-off between performance and efficiency.
- Our method is able to deal with fairly large cases within the time limit, whereas CSDP is prone to fail on these large layouts.

THANK YOU!