Layout Decomposition for Quadruple Patterning Lithography and Beyond

Bei Yu
ECE Department
University of Texas at Austin, Austin, TX
bei@cerc.utexas.edu

David Z. Pan
ECE Department
University of Texas at Austin, Austin, TX
dpan@ece.utexas.edu

ABSTRACT

As the feature size of semiconductor technology nodes continues to scale down, multiple patterning lithography (MPL) has emerged as a key solution for the advanced lithography, e.g., triple patterning lithography (TPL) for 14/11nm, and quadruple patterning lithography (QPL) for sub-10nm. In this paper, we propose a generic and robust QPL layout decomposition framework, which can be further extended to handle any general K-patterning lithography (K>4). Our framework is based on the semidefinite programming (SDP) formulation with novel coloring encoding. Meanwhile, we propose a linear algorithm and achieve significant speedup. The experimental results demonstrate the effectiveness of our framework.

1. INTRODUCTION

As the minimum feature size further decreases, multiple patterning lithography (MPL) has become one of the most viable solutions to sub-14nm half-pitch patterning [1, 2]. Last few years have seen extensive researches on MPL technology such as double patterning [3], and triple patterning [4]. Quadruple patterning lithography (QPL) is a natural extension along the paradigm of double/triple patterning. In the QPL manufacturing, there are four exposure/etching processes, through which the initial layout can be produced. Compared with triple patterning lithography, QPL introduces one more mask. Although increasing the number of processing steps by 33% over triple patterning, there are several reasons/advantages for QPL. Firstly, due to the delay or uncertainty of other lithography techniques, such as EUVL, semiconductor industry needs CAD tools to be prepared and understand the complexity/implication of QPL. Even from theoretical perspective, studying the general multiple patterning is valuable. Secondly, it is observed that for triple patterning lithography, even with stitch insertion, there are several common native conflict patterns. As shown in Fig. 1 (a), contact layout within the standard cell may generate some 4-clique patterns, which are indecomposable. This conflict can be easily resolved if four masks are available (see Fig. 1 (b)). Thirdly, with one more mask, some stitches may be avoided during manufacturing. By this way it is potential to resolve the overlapping and yield issues derived from the stitches.

The process of QPL brings up several critical yet open design challenges, such as layout decomposition, where the original layout is divided into four masks (colors). Triple patterning layout decomposition with conflict and stitch minimization has been well studied for full-chip layout [4, 5, 6, 7, 8, 9] and cell based design [10, 11, 12]. The problem can be optimally solved through expensive integer linear programming (ILP) [4]. To overcome the long runtime problem of ILP solver, some speedup techniques, e.g., semidefinite programming (SDP) [4, 7], and heuristic coloring assignment [5, 6] have been proposed. However, how to effectively solve the quadruple patterning, or even general multiple patterning problems, is still an open question.

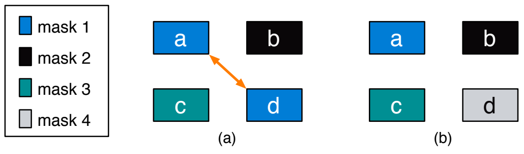

Figure 1: (a) A common native conflict from triple patterning lithography; (b) The conflict can be resolved through quadruple patterning lithography.

In this paper, we deal with the quadruple patterning layout decomposition (QPLD) problem. Our contributions are highlighted as follows. (1) To our best knowledge, this is the first layout decomposition research for QPLD problem. We believe this work will invoke more future research into this field thereby promoting the scaling of technology node. (2) Our framework consists of holistic algorithmic processes, such as semidefinite programming based algorithm, linear color assignment, and novel GH-tree based graph division. (3) We demonstrate the viability of our algorithm to suit with general K-patterning (K$\geq$4) layout decomposition, which could be advanced guidelines for future technology.

2. PRELIMINARIES

Given input layout which is specified by features in polygonal shapes, a *decomposition graphs* [4] is constructed. Now we give the problem formulation.

Problem 1 (QPLD). *Given an input layout which is specified by features in polygonal shapes and minimum coloring distance min_s, the decomposition graph is constructed. Quadruple patterning layout decomposition (QPLD) assigns all the vertices into one of four colors (masks) to minimize conflict number and stitch number.*

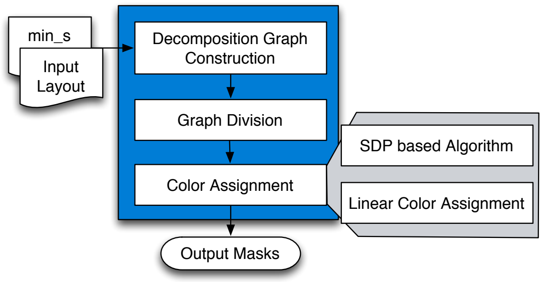

Figure 2: Proposed layout decomposition flow.

The overall flow of our layout decomposition is summarized in Fig. 2. We first construct decomposition graph to transform the original geometric patterns into a graph model. To reduce the problem size, graph division techniques are applied to partition the graph into a set of components. Then the color assignment problem can be solved independently for each component, through a set of algorithms discussed in Section 3.

3. COLOR ASSIGNMENT IN QPLD

Given decomposition graph $G = \{V, CE, SE\}$, color assignment would be carried out to assign each vertex into one of four colors (masks), to minimize both the conflict number and the stitch number. In this section, we propose two color assignment algorithms, i.e., semidefinite programming (SDP) based algorithm, and linear color assignment.

3.1 SDP Based Color Assignment

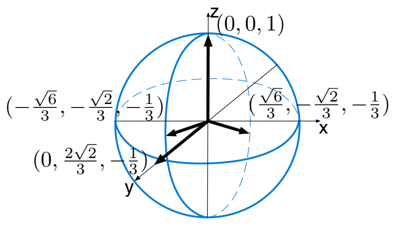
z
$(0,0,1)$
$(-\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$
$(\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$
x
$(0, \frac{2\sqrt{2}}{3}, -\frac{1}{3})$
y

Figure 3: Four vectors correspond to four colors.

Semidefinite programming (SDP) has been successfully applied to triple patterning layout decomposition [4, 7]. Here we will show that SDP formulation can be extended to solve QPLD problem. To represent four different colors (masks), as illustrated in Fig. 3, four unit vectors are introduced [13]: $(0,0,1)$, $(0, \frac{2\sqrt{2}}{3}, -\frac{1}{3})$, $(\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$ and $(-\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$. We construct the vectors in such a way that inner product for any two vectors $\vec{v_i}, \vec{v_j}$ satisfying: $\vec{v_i} \cdot \vec{v_j} = 1$ if $\vec{v_i} = \vec{v_j}$; $\vec{v_i} \cdot \vec{v_j} = -\frac{1}{3}$ if $\vec{v_i} \neq \vec{v_j}$.

Based on the vector definition, the QPLD problem can be formulated as the following vector programming:

$$\min \sum_{e_{ij} \in CE} \frac{3}{4}(\vec{v_i} \cdot \vec{v_j} + \frac{1}{3}) + \frac{3\alpha}{4} \cdot \sum_{e_{ij} \in SE} (1 - \vec{v_i} \cdot \vec{v_j}) \tag{1}$$
$$\text{s.t.} \quad \vec{v_i} \in \{(0,0,1), (0, \frac{2\sqrt{2}}{3}, -\frac{1}{3}), (\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3}), (-\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})\}$$

where the objective function is to minimize the conflict number and the stitch number. α is a user-defined parameter, which is set as 0.1 in this work. After relaxing the discrete constraints in (1) and removing the constant in objective function, we redraw the following semidefinite programming (SDP) formulation.

$$\min \sum_{e_{ij} \in CE} \vec{v_i} \cdot \vec{v_j} - \alpha \sum_{e_{ij} \in SE} \vec{v_i} \cdot \vec{v_j} \tag{2}$$
$$\text{s.t.} \quad \vec{v_i} \cdot \vec{v_i} = 1, \quad \forall i \in V$$
$$\vec{v_i} \cdot \vec{v_j} \geq -\frac{1}{3}, \quad \forall e_{ij} \in CE$$

After solving the SDP, we get a set of continuous solutions in matrix X, where each value x_{ij} in matrix X corresponds to $v_i \cdot v_j$. If x_{ij} is close to 1, vertices v_i, v_j are tend to be in the same mask (color). A greedy mapping algorithm [4] can be directly applied here to get color assignment solution. However, the performance of greedy method may not be good.

Algorithm 1 SDP + Backtrack

```
Input: SDP solution $x_{ij}$, threshold value $t_{th}$;
 1: for all $x_{ij} \geq t_{th}$ do
 2:     Combine vertices $v_i, v_j$ into one larger vertex;
 3: Construct merged graph $G' = \{V', CE', SE'\}$;
 4: BACKTRACK$(0, G')$;
 5: return color assignment result in $G'$;

 6: function BACKTRACK$(t, G')$
 7:     if t $\geq$ size$[G']$ then
 8:         if Find a better color assignment then
 9:             Store current color assignment;
10:     else
11:         for all legal color $c$ do;
12:             $G'[t] \leftarrow c$;
13:             BACKTRACK$(t+1, G')$;
14:             $G'[t] \leftarrow -1$;
```

To overcome the limitation of the greedy method, in our framework a backtrack based algorithm (see Algorithm 1) is proposed to consider both SDP results and graph information. The backtrack based method accepts two arguments of the SDP solution $\{x_{ij}\}$ and a threshold value t_{th}. In our work t_{th} is set as 0.9. As discussed above, if x_{ij} is close to be 1, two vertices v_i and v_j tend to be in the same color (mask). Therefore, we scan all pairs, and combine some vertices into one larger vertex (lines $1-3$). After the combination, the vertex number can be reduced, thus the graph has be simplified (line 4). The simplified graph is called *merged graph* [7]. On the merged graph, *BACKTRACK* algorithm is presented to search an optimal color assignment (lines $7-19$).

3.2 Linear Color Assignment

Backtrack based method may still involve runtime overhead, especially for complex case where SDP solution cannot provide enough merging candidates. Therefore, an efficient color assignment is required. Here we propose an efficient color assignment algorithm. Note that our method is targeting general graph, not just planar graph. In addition, different from classical four coloring method that needs quadratic runtime [14], our color assignment is a linear runtime algorithm.

Algorithm 2 Linear Color Assignment

```
Input: Decomposition graph $G = \{V, CE, SE\}$, Stack $S$;
 1: while $\exists v_i \in V$ s.t. $d_{conf}(v_i) < 4$ & $d_{stit}(v_i) < 2$ do
 2:     $S$.push$(v_i)$;
 3:     $G$.delete$(v_i)$;
 4: Construct vector $vec$;
 5: C1 = SEQUENCE-COLORING$(vec)$;
 6: C2 = DEGREE-COLORING$(vec)$;
 7: C3 = 3ROUND-COLORING$(vec)$;
 8: C = best coloring solution among {C1, C2, C3};
 9: POST-REFINEMENT$(vec)$;
10: while !$S$.empty() do
11:     $v_i = S$.pop();
12:     $G$.add$(v_i)$;
13:     $c(v_i) \leftarrow$ a legal color;
```

The details of linear color assignment are summarized in Algorithm 2, which involves three stages. The first stage is iteratively vertex removal. For each vertex v_i, we denote its conflict degree $d_{conf}(v_i)$ as number of conflict edges incident

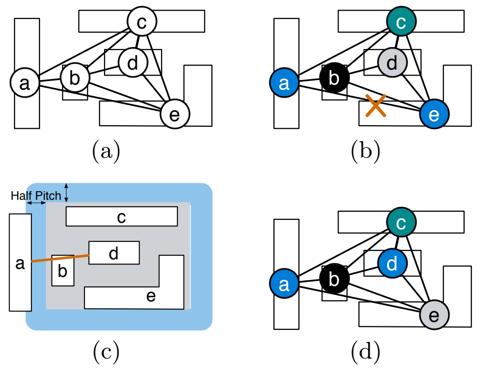

Figure 4: (a) Decomposition graph; (b) Greedy coloring with one conflict; (c) a is detected as color-friendly to d; (d) Coloring considering color-friendly rules.

to v_i, while its stitch degree $d_{stit}(v_i)$ as number of stitch edges. The main idea is that the vertices with conflict degree less than 4 and stitch degree less than 2 are identified as non-critical, thus can be temporarily removed and pushed into stack S (lines 1-4). After coloring remaining vertices, each vertex in stack S would be pop up one by one and assigned one legal color (lines 11-15). This strategy is safe in terms of conflict number. In other words, when a vertex is pop up from S, there is always one color available without introducing new conflict.

In the second stage (lines 5-9), all remaining vertices would be assigned colors one by one. However, color assignment through one specific order may be stuck at *local optimum* which stems from the greedy nature. For example, given a decomposition graph in Fig. 4 (a), if the coloring order is a-b-c-d-e, when vertex d is greedily selected grey color, the following vertex e cannot find any color without conflict (see Fig. 4 (b)). In other words, vertex ordering significantly impacts the coloring result.

To alleviate the impact of vertex ordering, two strategies are proposed. The first strategy is called **color-friendly rules**, as in Definition 1. In Fig. 4 (c), all conflict neighbors of pattern d are labeled inside a grey box. Since the distance between a and d is within the range of $(min_s, min_s + hp)$, a is color-friendly to d. Interestingly, we discover a rule that for a complex/dense layout, color-friendly patterns tend to be with the same color. Based on these rules, during linear color assignment, to determine one vertex color, instead of just comparing its conflict/stitch neighbors, the colors of its color-friendly vertices would also be considered. Detecting color-friendly vertices is similar to the conflict neighbor detection, thus it can be finished during decomposition graph construction without much additional efforts.

Definition 1 (Color-Friendly). *A pattern a is color-friendly to pattern b, iff their distance is larger than min_s, but smaller than $min_s + hp$. Here hp is the half pitch.*

Our second strategy is called **peer selection**, where three different vertex orders would be processed simultaneously, and the best one would be selected as the final coloring solution (lines 6-8). Although color assignment is solved thrice, since for each order the coloring is in linear time, the total computational time is still linear.

In the third stage (line 10), post-refinement greedily checks each vertex to see whether the solution can be further improved.

For a decomposition graph with color-friendly information and n vertices, in the first stage vertex removal/pop up can be finished in $O(n)$. In the second stage, as mentioned above the coloring needs $O(n)$. In post-refinement stage, all vertices are traveled once, which requires $O(n)$ time. Therefore, the total complexity is $O(n)$.

4. GENERAL K-PATTERNING LAYOUT DECOMPOSITION

In this section, we demonstrate that our layout decomposition framework is generalizable to K-patterning layout decomposition, where $K > 4$.

Theorem 1: *SDP formulation in (3) can provide $v_i \cdot v_j$ pairs for K-patterning color assignment problem.*

$$\min \sum_{e_{ij} \in CE} (\vec{v_i} \cdot \vec{v_j} + \frac{1}{k-1}) + \alpha \sum_{e_{ij} \in SE} (1 - \vec{v_i} \cdot \vec{v_j}) \tag{3}$$
$$\text{s.t. } \vec{v_i} \cdot \vec{v_i} = 1, \quad \forall i \in V$$
$$\vec{v_i} \cdot \vec{v_j} \geq -\frac{1}{k-1}, \quad \forall e_{ij} \in CE$$

We can see that if $K = 4$, formulation (3) equivalents to (2). Rephrasing both the SDP formulation in (3) and backtrack method in Algorithm 1, the color assignment problem for K-patterning can be resolved. In addition, the linear color assignment algorithm in Section 3.2 can be extended to general K-patterning problem as well.

5. EXPERIMENTAL RESULTS

We implemented the proposed layout decomposition algorithms in C++, and tested on a Linux machine with 2.9GHz CPU. We choose GUROBI [15] as the integer linear programming (ILP) solver, and CSDP [16] as the SDP solver. The benchmarks in [4, 5] are used as our test cases. We scale down the Metal1 layer to 20nm half pitch. Both the minimum feature width w_m and the minimum spacing between features s_m are 20nm. In our experiments, for quadruple patterning min_s is set as $2 \cdot s_m + 2 \cdot w_m = 80$nm, while for pentuple patterning min_s is set as $3 \cdot s_m + 2.5 \cdot w_m = 110$nm. When larger min_s is applied, there are too many native conflicts in layouts, as the benchmarks are not multiple patterning friendly.

We compare different color assignment algorithms for quadruple patterning, and the results are listed in Table 1. "**ILP**", "**SDP+Backtrack**", "**SDP+Greedy**" and "**Linear**" denote ILP formulation, SDP followed by backtrack mapping (Section 3.1), SDP followed by greedy mapping, and linear color assignment (Section 3.2), respectively. Here we implement an ILP formulation extended from the triple patterning work [4]. The columns "cn#" and "st#" denote the conflict number and the stitch number, respectively. Column "CPU(s)" is color assignment time in seconds. From Table 1 we can see that for small cases the ILP formulation can achieve best performance in terms of conflict number and stitch number. However, for large cases (S38417, S35932, S38584, S15850) ILP suffers from long runtime problem that none of them can be finished in one hour. Compared with ILP, SDP+Backtrack can achieve near-optimal solutions, i.e., in every case the conflict number is optimal, while only in one case 2 more stitches are introduced. SDP+Greedy method can achieve 2× speedup against SDP+Backtrack. But the performance of SDP+Greedy is not good that for complex designs hundreds of additional conflicts are reported. The linear color assignment can achieve around 200× speedup against

Table 1: Comparison for Quadruple Patterning

Circuit	ILP			SDP+Backtrack			SDP+Greedy			Linear		
	cn#	st#	CPU(s)	cn#	st#	CPU(s)	cn#	st#	CPU(s)	cn#	st#	CPU(s)
C432	2	0	0.6	2	0	0.24	2	0	0.02	2	1	0.001
C499	1	4	0.7	1	4	0.16	1	4	0.05	1	4	0.001
C880	1	0	0.3	1	0	0.02	1	0	0.02	1	2	0.001
C1355	0	4	0.6	0	4	0.1	0	4	0.04	0	4	0.001
C1908	2	3	1.0	2	3	0.28	2	3	0.09	2	4	0.001
C2670	0	6	1.1	0	6	0.16	0	6	0.1	0	7	0.001
C3540	1	3	1.1	1	3	0.09	2	2	0.05	1	3	0.001
C5315	1	13	2.8	1	13	0.6	2	12	0.24	1	15	0.002
C6288	9	0	2.3	9	0	0.36	9	0	0.17	9	1	0.001
C7552	2	13	3.4	2	13	0.6	3	12	0.22	2	18	0.003
S1488	0	6	0.7	0	6	0.05	4	2	0.01	0	6	0.001
S38417	20	549	1226.7	20	551	6.6	142	429	2.7	21	576	0.03
S35932	N/A	N/A	>3600	50	1745	28.7	460	1338	16.4	64	1927	0.15
S38584	N/A	N/A	>3600	41	1653	21.1	470	1224	10.4	47	1744	0.12
S15850	N/A	N/A	>3600	42	1462	18	420	1084	7.8	48	1571	0.11
avg.	-	-	>802.7	11.5	364.0	5.14	101.2	274.7	2.56	13.3	392.2	0.03
ratio	-	-	**>156.3**	**1.0**	**1.0**	**1.0**	**8.83**	**0.75**	**0.49**	**1.15**	**1.08**	**0.005**

Table 2: Comparison for Pentuple Patterning

Circuit	SDP+Backtrack			SDP+Greedy			Linear		
	cn#	st#	CPU(s)	cn#	st#	CPU(s)	cn#	st#	CPU(s)
C6288	19	2	2.4	19	2	0.49	19	5	0.005
C7552	1	1	0.3	1	1	0.05	1	4	0.001
S38417	0	4	1.45	0	4	0.21	0	4	0.001
S35932	5	20	8.11	5	20	0.62	5	25	0.009
S38584	3	4	1.66	7	3	0.3	3	6	0.008
S15850	6	5	2.7	7	5	0.4	5	15	0.007
avg.	5.7	6.0	2.77	6.5	5.83	0.35	5.5	9.8	0.005
ratio	**1.0**	**1.0**	**1.0**	**1.15**	**0.97**	**0.12**	**0.97**	**1.64**	**0.002**

SDP+Backtrack, while only 15% more conflicts and 8% more stitches are reported.

We further compare the algorithms for pentuple patterning, that is, $K = 5$. To our best knowledge there is no exact ILP formulation for pentuple patterning in literature. Therefore we consider three baselines, i.e., SDP+Backtrack, SDP+Greedy, and Linear. All the graph division techniques are applied. Table 2 evaluates six most dense cases. We can see that compared with SDP+Backtrack, SDP+Greedy can achieve around 8× speedup, but 15% more conflicts are reported. In terms of runtime, linear color assignment can achieve 500× and 60× speedup, against SDP+Backtrack and SDP+Greedy, respectively. In terms of performance, linear color assignment reports the best conflict number minimization, but more stitches may be introduced.

6. CONCLUSIONS

In this paper we have proposed the first layout decomposition framework for quadruple patterning and beyond. Experimental evaluations have demonstrated that our algorithm is effective and efficient to obtain high quality solution. As continuing scaling of technology node to sub-10nm, MPL may be a promising manufacturing solution. We believe this paper will stimulate more future research into this field, thereby facilitating the advancement of MPL technology.

Acknowledgment

The authors would like to thank Tsung-Wei Huang for helpful discussions. This work is supported in part by NSF, NSFC, SRC, Oracle, and Toshiba.

7. REFERENCES

[1] D. Z. Pan, B. Yu, and J.-R. Gao, "Design for manufacturing with emerging nanolithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 32, no. 10, pp. 1453–1472, 2013.

[2] B. Yu, J.-R. Gao, D. Ding, Y. Ban, J.-S. Yang, K. Yuan, M. Cho, and D. Z. Pan, "Dealing with IC manufacturability in extreme scaling," in *ICCAD*, 2012, pp. 240–242.

[3] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *ICCAD*, 2008, pp. 465–472.

[4] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," in *ICCAD*, 2011, pp. 1–8.

[5] S.-Y. Fang, W.-Y. Chen, and Y.-W. Chang, "A novel layout decomposition algorithm for triple patterning lithography," in *DAC*, 2012, pp. 1185–1190.

[6] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *DAC*, 2013, pp. 69:1–69:6.

[7] B. Yu, Y.-H. Lin, G. Luk-Pat, D. Ding, K. Lucas, and D. Z. Pan, "A high-performance triple patterning layout decomposer with balanced density," in *ICCAD*, 2013, pp. 163–169.

[8] Y. Zhang, W.-S. Luk, H. Zhou, C. Yan, and X. Zeng, "Layout decomposition with pairwise coloring for multiple patterning lithography," in *ICCAD*, 2013, pp. 170–177.

[9] B. Yu, J.-R. Gao, and D. Z. Pan, "Triple patterning lithography (TPL) layout decomposition using end-cutting," in *Proc. of SPIE*, vol. 8684, 2013.

[10] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and M. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in *ICCAD*, 2012, pp. 57–64.

[11] B. Yu, X. Xu, J.-R. Gao, and D. Z. Pan, "Methodology for standard cell compliance and detailed placement for triple patterning lithography," in *ICCAD*, 2013, pp. 349–356.

[12] H. Tian, Y. Du, H. Zhang, Z. Xiao, and M. Wong, "Constrained pattern assignment for standard cell based triple patterning lithography," in *ICCAD*, 2013, pp. 57–64.

[13] D. Karger, R. Motwani, and M. Sudan, "Approximate graph coloring by semidefinite programming," *J. ACM*, vol. 45, pp. 246–265, March 1998.

[14] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas, "Efficiently four-coloring planar graphs," in *ACM Symposium on Theory of computing*. ACM, 1996, pp. 571–575.

[15] "GUROBI," http://www.gurobi.com/html/academic.html.

[16] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 613 – 623, 1999.