ESE533 Final Project - Internal Structure Optimization Problem

HuaLing Zheng
StonyBrook University
hualin.zheng@stonybrook.edu
05/19/2019

***Abstract*—In real life, we always need to face some area minimization problem. Sometimes it can be simple like bucket designing problem. Sometimes it can be hard as floor planning problem which is widely used in VSLI chip designing field. The objective is usually to minimize the size (e.g., area, volume, perimeter) of the bounding box, which is the smallest box that contains the boxes to be configured and placed.**

***Index Terms*—Convex Optimization; Linear Optimization; Geometric Optimization; MATLAB; CVX.**

I. Introduction

This paper has two part. The first part is to solve a paper box optimization problem; Second part is to solve floor mapping problem. They are all trying to minimize the usage of the mete rial or area.

II. Paper Box Optimization

When designing a package, the designer always wants to optimize the size of the product while maintaining the value of the product. Choosing the best size ratio can not only reduce packaging costs and transportation costs, but also beautify the appearance of the packaging to promote sales. Now I will combine the main features of the design of the packaging container mechanism, to build convex optimization model and use MATLAB to simulate the model and check if the result is best solution or not.

A. Building the optimization model

Let the $x = (x_1, x_2, x_3, ..., x_n)^T$ are the design variable for a package structure design work. Under the constraints: $f_i(x) \leq 0, i = 1, ..., m$ and $h_i(x) = 0, i = 1, ..., p$. I need get a optimized x that can get the value of the minimize $f_0(x)$.

Basically, $f_0(x)$, $f_i(x)$ and $g_i(x)$ all have linear relation with x. This is a linear optimization problem. Of course, the constraints may not be linear function. Then, this problem become a non-linear problem. However, to make it simple, I will only focused on linear optimization problem. The paper box's volume is $0.1m^3$. And sum of the height, width and length must be less than $1.6m^2$ (See Figure 1 and 2).

$minimize$ $f(x)$ = $2(x_1 + x_2)(x_2 + x_3)$.

subject to $x_1 + x_2 + x_3 \leq 1$.

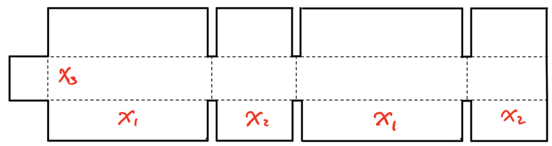

Fig. 1. Structure of the paper box

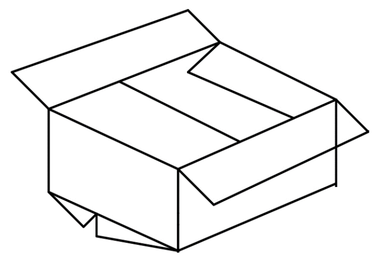

Fig. 2. Structure of the paper box

..........$x_1 * x_2 * x_3 = 0.1$

..........$x_1 \geq 0;$

..........$x_2 \geq 0;$

..........$x_3 \geq 0;$

B. MATLAB Coding

1) Build Model: Since most of the optimal design of the packaging structure is constrained nonlinear programming problem, I decided to use MATLAB fmincon( ) function to solve the multi-variable constraint optimization problem. The function format is:

fmincon function:

$$\min_x f(x) = \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A * x \leq b \\ Aeq * x = beq \\ lb \leq x \leq ub \end{cases}$$

$$[x, fval, exitflag, output, lambda, grad, hessian] = fmincon(fun, x0, A, b, Aeq, beq, lb, ub, nonlcon, options)$$

In the above function, fun is the objective function. x_0 is the initial value of the design variable. x, b, beq, lb, and ub are vectors, A and Aeq are matrices, $c(x)$ and $ceq(x)$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. $f(x)$, $c(x)$, and $ceq(x)$ can be nonlinear functions. The function nonlcon accepts a vector x and returns two vectors c and ceq. The vector c contains the nonlinear inequalities evaluated at x, and ceq contains the nonlinear equalities evaluated at x. options provides the function-specific details for the options parameters.

The return value fval is the function value of the objective function at the optimal solution x point. exitflag is the termination flag of the return algorithm. $exitflag > 0$ means the function converged to a solution x. $exitflag = 0$ means the maximum number of function evaluations or iterations was exceeded. $exitflag < 0$ means the function did not converge to a solution. output is the Structure containing information about the optimization. lambda is the Structure containing the Lagrange multipliers at the solution x (separated by constraint type). grad is the gradient of fun at the solution x. hessian is is the hessian of fun at the solution x. When fmincon function is chosen to solve the constrained nonlinear packaging structure problem, the objective function and the constraint function must be continuous and derivable. Optimization design results may sometimes be just partial solutions, and when the problem has no solution, the function will try to narrow the maximum value of the constraints.

2) Build Function and Running Code: First, build the objective function file: fun.m

```
function f = fun(x)

f = 2 * (x(1) + x(2)) * (x(2) + x(3));
```

Second, build constraint function file: constraint.m

```
function [c, ceq] = constraint(x)

c = [ ];

ceq = x(1) * x(2) * x(3) - 0.1;
```

Third, build main testing file: test.m

```
x0 = [1 1 1];

A = [1, 1, 1];

b = 1.6;

lb = [0; 0; 0];
```

Iter	F-count	f(x)	Feasibility	First-order optimality	Norm of step
0	4	8.000000e+00	1.400e+00	1.333e+00	
1	8	6.483198e+00	1.251e+00	1.000e+00	2.598e-01
2	12	5.052119e+00	9.996e-02	5.924e+00	1.697e+00
3	16	3.536868e+00	2.205e-02	4.427e+00	6.465e-01
4	20	3.250240e+00	4.470e-04	4.014e+00	1.300e-01
5	24	3.082256e+00	1.686e-04	3.849e+00	4.297e-02
6	40	2.580167e+00	7.733e-04	3.318e+00	1.700e-01
7	56	2.242463e+00	3.949e-04	2.886e+00	1.436e-01
8	68	1.828416e+00	1.306e-03	1.242e+00	1.996e-01
9	78	1.572185e+00	1.561e-03	4.243e-01	1.828e-01
10	82	1.522384e+00	2.551e-03	2.199e-01	8.850e-02
11	86	1.541072e+00	2.717e-04	2.066e-01	3.902e-02
12	90	1.542378e+00	3.503e-05	2.028e-01	1.439e-02
13	94	1.538046e+00	2.423e-04	1.865e-01	4.043e-02
14	98	1.535658e+00	3.377e-04	1.253e-01	4.855e-02
15	102	1.538614e+00	4.808e-05	6.201e-02	1.393e-02
16	106	1.538848e+00	1.483e-05	7.281e-03	9.872e-03
17	110	1.538955e+00	2.308e-06	2.442e-03	3.331e-03
18	114	1.538978e+00	3.042e-08	2.335e-04	3.301e-04
19	118	1.538978e+00	2.236e-10	4.000e-05	3.844e-05
20	122	1.538978e+00	1.392e-10	1.775e-05	2.199e-05
21	126	1.538978e+00	1.053e-11	2.599e-07	6.012e-06

Fig. 3. The running results

```
options = optimset('LargeScale','off','Display','iter');

[x, fval, exitflag, output] =

fmincon('fun', x0, A, b, [ ], [ ], lb, [ ], 'constraint', options)
```

Fourth, start to runing the test.m file and check the resutls (see Figure 3).

Output:

```
x = 0.5848    0.2924   0.5848

fval = 1.5390

exitflag = 1

iterations: 21

funcCount: 126

constrviolation: 1.0526e-11

stepsize: 6.0117e-06

algorithm: 'interior-point'

firstorderopt: 2.5986e-07

cgiterations: 12
```

C. Conclusion of the paper box optimization

After running the optimset() function, we found that we can find the minimized f(x) when $x_1 = 0.5848$m, $x_2 = 0.2924$m and $x_1 = 0.5848$m. The fval $= 1.5390 m^2$.

Through the analysis, I turned the box design problem to the convex optimization problem. And I use MATLAB function to help me find the optimized box size based on the constraints. Optimizing design results not only beautifies the appearance

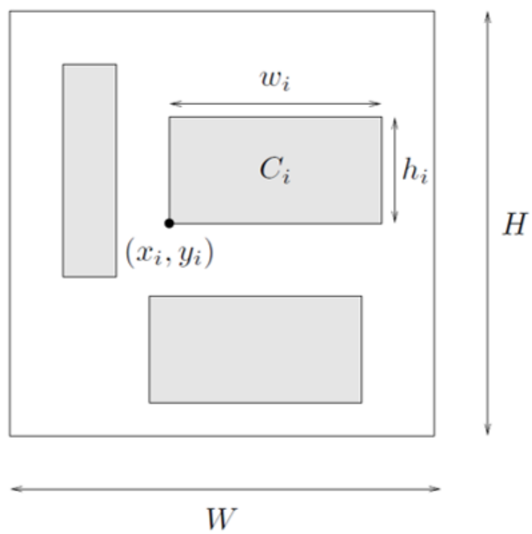

Fig. 4. Floor Planning Design [1]

of the package, but also reduces packaging costs. This reduces the workload of the designer and speeds up the design.

III. Flooring Mapping Problem

A. Analysis and build model

I read the text book Chapter 8.8. The whole analysis already go through most of the solutions. The non-overlap constraints make the general floor planning problem a complicated combinatorial optimization problem or rectangle packing problem. However, if the relative positioning of the boxes is specified, several types of floor planning problems can be formulated as convex optimization problems [1].

We have N cells or modules $C1, ..., CN$ that are to be configured and placed in a rectangle with width W and height H, and lower left corner at the position (0, 0). The geometry and position of the ith cell is specified by its width w_i and height h_i, and the coordinates (x_i, y_i) of its lower left corner [1]. See Figure 4.

The variables in the problem are $x_i, y_i, w_i, h_i$for $i = 1$, ...,N, and the width W and height H of the bounding rectangle. In all floor planning problems, we require that the cells lie inside the bounding rectangle, i.e., [1]

$$xi_i \geq 0,\ y_i \geq 0,\ x_i + wi_i \leq W,\ y_i + h_i \leq H,\ i = 1, ..., N.$$

The cells should not overlap, except possibly on their boundaries:

int $(C_i \cap C_j) = \emptyset$ for $i \neq j$.

C_i is left of C_j, or C_i is right of C_j, or C_i is below C_j, or C_i is above C_j.

These four geometric conditions correspond to the inequalities:

x_i + w_i $\leq$x$_j$, or x_j + w_j $\leq$x$_i$, or y_i + h_j $\leq$y$_j$, or y_j + h_i $\leq$y$_i$,

at least one of which must hold for each $i \neq j$. I need to specify constraints to give two relations on $1, ..., N$: £(meaning left of) and β (meaning below). Then impose the constraint that C_i is to the left of C_j if $(i, j) \in £$, and C_i is below C_j if $(i, j) \in \beta$. This yields the constraints

x_i + w_i $\leq$x$_j$ for $(i, j) \in £$, $y_i + h_i \leq y_j for(i, j) \in \beta$,

$$(i, j) \in £, (j, i) \in £, (i, j) \in \beta, (j, i) \in \beta.$$

Evidently, we only need to impose the inequalities that correspond to the edges of the graphs $\mathcal{H}$ and $\mathcal{V}$; the others follow from transitivity. We arrive at the set of inequalities [1]:

$$x_i + w_i \leq x_j for(i, j) \in \mathcal{H}, y_i + h_i \leq y_j for(i, j) \in \mathcal{V}$$

$$x_i \geq 0\ for\ i\ \mathcal{L}\ minimal, x_i + w_i \leq W for\ i\ \mathcal{L}\ maximal,$$

$$y_i \geq 0\ for\ i\ \mathcal{B}\ minimal, y_i + h_i \leq \mathcal{H} for\ i\ \mathcal{B}\ maximal.$$

I also need specify a minimum area $w_i h_i \geq A_i^{min}$. $w_i \geq A_i/h_i$. I can also give a upper and lower bounds to the aspect ratio of each cell: $r_i^{min} \geq h_i/w_i \geq r_i^{max}$. I should impose the constraint that two edges, or a center line, of two cells are aligned. For example, the horizontal center line of cell $\mathcal{B}_i$ aligns with the top of cell $\mathcal{B}_j$ when $y_i + h_i/2 = y_j + h_j$ [1].

We can require pairs of cells to be symmetric about a vertical or horizontal axis, that can be fixed or floating (i.e., whose position is fixed or not). For example, to specify that the pair of cells i and j are symmetric about the vertical $axis\ x = x_{axis}$, we impose the linear equality constraint [1]

$$x_{axis}(x_i + w_i/2) = x_j + w_j/2 - x_{axis}.$$

We can impose a variety of constraints that limit the distance between pairs of cells. In the simplest case, we can limit the distance between the center points of cell i and j (or any other fixed points on the cells, such as lower left corners). For example, to limit the distance between the centers of cells i and j, we use the (convex) inequality [1]

$$\|(x_i + w_i/2, y_i + h_i/2) - (x_j + w_j/2, y_j + h_j/2)\| \leq D_{ij}.$$

We can also limit the distance dist(C_i, C_j) between cell i and cell j, i.e., the minimum distance between a point in cell i and a point in cell j. In the general case this can be done as follows. To limit the distance between cells i and j in the norm $\|.\|$, we can introduce four new variables u_i, v_i, u_j, v_j. The pair (u_i, v_i) will represent a point in C_i, and the pair (u_j, v_j) will represent a point in C_j. To ensure this we impose the linear inequalities [1]

$x_i \le u_i \le x_i + w_i, \quad y_i \le v_i \le y_i + h_i,$

$x_j \le u_j \le x_j + w_j, \quad y_j \le v_j \le y_j + h_j,$

Finally, to limit dist(C_i, C_j), we add the convex inequality

$\|(u_i, v_i) - (u_j, v_j)\| \le D_{ij}.$

We can limit the $\ell_1 - (or\ \ell_2 -)$ distance between two cells in a similar way. Here we introduce one new variable d_v, which will serve as a bound on the vertical displacement between the cells. To limit the 1-distance, we add the constraints [1]

$y_j - (y_i + h_i) \le d_v, \;\; y_i - (y_j + h_j) \le d_v, \;\; d_v \ge 0$

$x_j - (x_i + w_i) + d_v \le D_{ij}.$

$(x_j - (x_i + w_i))^2 + d_v^2 \le D_{ij}^2.$

The minimal areas of all blocks: A_i^{min}, $i = 1, ..., N$

The relations $\mathcal{L}$ and $\mathcal{U}$ that definde relative positioning of blocks:

$$L_{ij} = \begin{cases} 1, & (i,j) \in \mathcal{L} \\ 0, & otherwise \end{cases}$$

$$U_{ij} = \begin{cases} 1, & (i,j) \in \mathcal{U} \\ 0, & otherwise \end{cases}$$

The convex problem form:

minimize 2(W+H)

subject to $-x \le 0, -y \le 0, -w \le 0, -h \le 0,$

..............$-W \le 0, -H \le 0$

..............$x + w - W * \vec{1} \le 0, \quad y + h - H * \vec{1} \le 0$

..............$A_i/h_i - w_i \le 0, \quad i = 1, ..., N$

..............$L_{ij} * (x_i + w_i - x_j) \le 0,\ i = 1, ..., N,\ j = 1, ..., N$

..............$U_{ij} * (y_i + h_i - y_j) \le 0,\ i = 1, ..., N,\ j = 1, ..., N$

B. MATLAB Coding

Now, I will combine everything that was stated above into a MATLAB coding to slove the floor planning problem. First build a function to split different area (See Figure 5).

Next, I need to build function for $\mathcal{L}$ and $\mathcal{U}$ relations ((See Figure 6):

```
function [ x1, x2 ] = diffArea( x )

    [ x1, id ] = sort(x);
    id = flip(id)';

    s1 = 0; s2 = 0;
    x1 = []; x2 = [];

    for i = id
        if s1 < s2
            x1 = [x1 i];
            s1 = s1 + x(i);
        else
            x2 = [x2 i];
            s2 = s2 + x(i);
        end
    end
end
```

Fig. 5. diffArea.m

```
function [ L, U ] = relate( x )

    n = length(x);
    [ L, U ] = buildrelate(x, 1:n, zeros(n), zeros(n), 0);

    function [ L, U ] = buildrelate( x, x_id, L, U, path)
        [ x1_id, x2_id ] = diffArea(x);
        for i = x_id(x1_id)
            for j = x_id(x2_id)
                if path == 0
                    L(i,j) = 1;
                else
                    U(i,j) = 1;
                end
            end
        end

        if length(x1_id) > 1
            [ L, U ] = buildrelate( x(x1_id), x_id(x1_id), L, U, ~path);
        end
        if length(x2_id) > 1
            [ L, U ] = buildrelate( x(x2_id), x_id(x2_id), L, U, ~path);
        end
    end
end
```

Fig. 6. relate.m

After build the relation, I also need to build a function to reduce the overlap (See Figure 7):

Now I can use CVX library function to optimize the boxes. The diag(x) is an n x n matrix where the elements of vector x are placed on the main diagonal (See Figure 8):

Also I need build a function to make sure the boxes is closure (See Figure 9):

Finally, I need a test file to check the result (See Figure 10):

C. MATLAB Results

After running the code, the results is like the blow graph (See Figure 11,12), I input a = [12; 15; 30; 20; 80], and I end up with Figure 12's placement of blocks.

IV. Conclusion

In this report, I tried to use convex optimization to analysis the two problems. Both paper box and floor planning problem are possible to get optimized solution. The linear

```
function [ A ] = reduction( R )

    if size(R,1) ~= size(R,2)
        error('Must be square matrix');
    end

    n = size(R,1);
    A = zeros(n);

    for u = 1:n
        for k = 1:n
            if R(u,k) == 1
                A(u,k) = 1;

                for i = 1:n
                    if R(u,i) == 1 && R(i,k) == 1
                        A(u,k) = 0;
                    end
                end
            end
        end
    end
end
```

Fig. 7. reduction.m

```
function [ x, y, w, h, S_W, S_H ] = optimize( T, L, c )

    if size(T,1) ~= size(T,2) || size(T,1) ~= size(L,1) || size(L,1) ~= size(L,2)
        error('invalid matrix');
    end
    if size(T,1) ~= length(c)
        error('invalid length');
    end

    m = length(c);

    cvx_begin quiet
        variables x(m) y(m);
        variable w(m) nonnegative;
        variable h(m) nonnegative;
        variable Rect_W nonnegative;
        variable Rect_H nonnegative;
        minimize 2*(Rect_W+Rect_H);
        subject to
            0 <= x <= S_W - w;
            0 <= y <= S_H - h;
            diag(x)*T + diag(w)*T - T*diag(x) <= 0;
            diag(y)*L + diag(h)*L - L*diag(y) <= 0;
            c .* inv_pos(h) - w <= 0;
    cvx_end
end
```

Fig. 8. optimize.m

programming and geometric optimization can be carefully turn to convex optimization. And with the help of MATLAB programming, we can easily get optimized results.

References

[1] S. Boyd, L. Vandenberghe, "Convex Optimization", Cambridge University Press New York, NY, USA 2004. ISBN:0521833787.


```
function [ R ] = close( R )

    if size(R,1) ~= size(R,2)
        error('Must be square matrix');
    end

    u = size(R,1);

    for k = 1:ceil(log2(u))
        R = double(logical(R + logical(R * R)));
    end
end
```

Fig. 9. close.m

```
a = [12; 15; 30; 20; 80];

[ L, U ] = relate(a);

H = reduction(L);
V = reduction(U);

gH = graph(H); view(gH);
gV = graph(V); view(gV);

[ x, y, w, h, S_W, S_H ] = optimize(H,V,a);

u = length(a);

for m=1:u
    fill([x(m); x(m)+w(m); x(m)+w(m); x(m)],[y(m);y(m);y(m)+h(m);y(m)+h(m)],0.90*[1 1 1]);
    hold on;
end

for n=1:u
    k = text(x(n)+w(n)/2, y(n)+h(n)/2,['U_{',int2str(n),'}']);
    set(k, 'HorizontalAlignment', 'center');
    set(k, 'VerticalAlignment', 'middle');
    set(k, 'FontWeight', 'bold');
    set(k, 'FontName', 'Cambria');
end

axis([0 S_W 0 S_H]);
axis equal; axis off;
```

Fig. 10. test.m

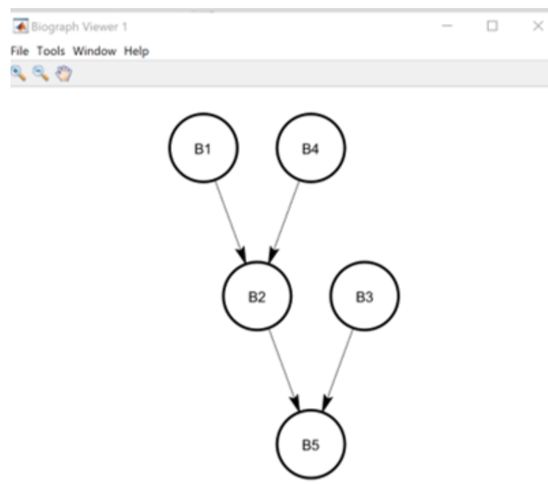

Fig. 11. reduction.m

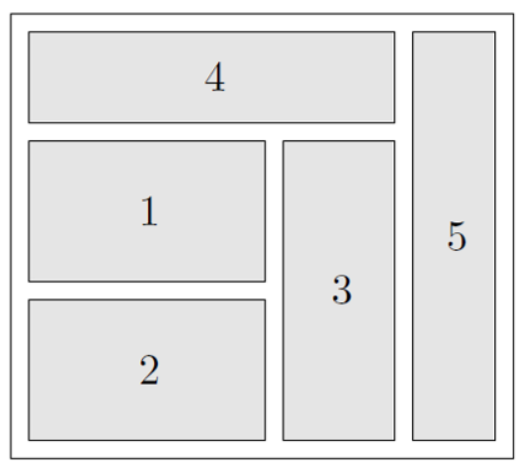

Fig. 12. reduction.m