Chalmers/University of Gothenburg
Department of Mathematical Sciences
Optimization
Ann-Brith Strömberg & Michael Patriksson

TMA521/MMA511
Project 1
Lagrangian duality
2018–01–17

Project 1: VLSI routing and Lagrangean duality

1 Introduction

The purpose of this project is to illustrate how a relatively difficult optimization problem can be attacked by using Lagrangian duality. During the project you will write a *Matlab* program for the solution of the Lagrangian dual problem. At your service are two functions that are called from Matlab.

We consider a routing problem in "very large-scale integrated circuit design" (VLSI), namely an application of the so-called "Manhattan channel routing problem". The mathematical classification of this problem is that of "vertex disjoint paths". The problem and the application are based on the article *Lagrangian relaxation for testing infeasibility in VLSI routing*, by T.A. Feo and D.S. Hochbaum, published in Operations Research, vol. 34, pp. 819–831 (1986), (dx.doi.org/10.1287/opre.34.6.819).

The program to be constructed is to be used to decide whether a given placement of a number of connections is possible to implement with respect to the wiring necessary between the components. The circuit board, where the connections are to be placed, is such that on one side of the board wiring can only be done horizontally, and on the other side only vertically. On the board there are predefined "vias" (or connectors), at which it is possible to connect the two sides of the board. Figure 1 illustrates such a problem having six horizontal wires on one side of the board and eight vertical wires on the other, as well as 48 vias and six contact pairs.

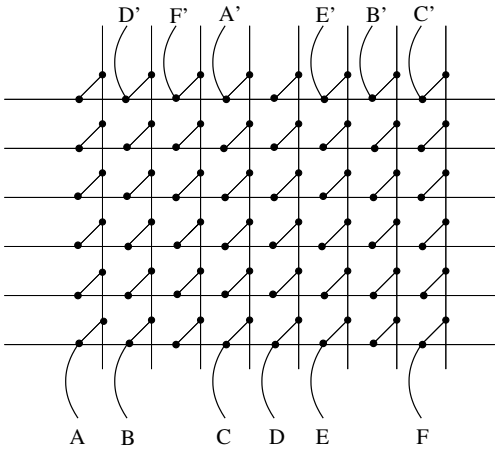

Figure 1: A Manhattan channel routing problem with the contact pairs A–A', B–B', , . . . , F–F'.

Given a number of contact pairs, we wish to determine whether it is possible to connect all of them on the two-sided board.

First, we introduce the notation needed to mathematically describe the network and how the contact pairs are interconnected. Then we present the optimization problem,

the solution of which determines whether the routing problem has a solution or not. This problem will then be studied by using Lagrangian duality, whose formulation will be attacked using subgradient optimization. *The program code for this algorithm defines your first task.*

Using the bounds on the optimal function value obtained we can then draw conclusions about the existence of feasible solutions to the wiring problem.

2 Tasks to perform

2.1 Task 1

Study the problem carefully and write down a sketch of a Matlab program that follows the scheme in Algorithm 1. Your task is to write the program code for the subgradient optimization. The program is to be tested on problems that can be downloaded from the course homepage. The files named "p6.m", "p10.m," and "p11.m" contain definitions of `dimX`, `dimY`, `k`, and `com` (described in Section 5). The problem defined in "p6.m" is illustrated in Figure 1, which hence suits as a trial problem for the algorithm coded. This problem lacks solutions to the wiring problem. On the homepage are also two additional functions, "gsp.c" and "visagrid.m", described in Section 5.

To be handed in are

- The program code and a description in words of the algorithm.
- For each problem instance, the optimal dual objective value.
- For each problem instance, a graph with the dual objective value plotted on the vertical axis against the iteration number on the horizontal axis.
- For each problem instance, a conclusion of whether it has a solution.

2.2 Task 2

In a second phase of the project you will implement a primal feasibility heuristic. In the article by Feo ands Hochbaum there is no such heuristic; in this case this is natural, considering the fact that the authors describe primal heuristic algorithms for the problem that are already in use at BELLCORE.

Your task is, based on the Lagrangian subproblem and the constraints having been Lagrangian relaxed, to construct a feasibility heuristic that manipulates a Lagrangian subproblem solution to find a primal feasible solution. Since our problem is defined over a graph it makes good sense to consider utilizing graph operations and/or optimization. Note that the heuristic should not be very complicated, and that it must terminate; that is, it must not be constructed such that there is any risk of it never terminating with a solution; therefore, in the worst case, we must accept that the heuristic fails on occasion. In particular, consider the different Lagrangian heuristic principles presented at Lectures 4 and 6. Also, consider the methodologies presented in the MSc thesis by Aldenvik and Schierscher (2015) as well as in the article *Recovery of primal solutions from dual subgradient methods for mixed binary linear programming* by Gustavsson et al. (2015).

The usefulness of the heuristic is two-fold: (a) it provides a pessimistic bound on the optimal objective value, which can be used in a termination criterion for the dual algorithm; (b) it provides a feasible solution at the end of the process, and this solution can then be used as our candidate solution to be implemented.

To be handed in are

- This part of the program code and a description in words of the algorithm.
- The best solutions found in this way for each problem instance.
- For each problem instance, a graph where the primal and dual objective values are plotted against the iteration number.

2.3 Task 3

In the project groups consisting of more than one person, each student, individually, must hand in a written report on how the project work has been distributed within the group and how the cooperation has worked out.

3 Mathematical model

We introduce mathematical notation to describe the routing problem.

Let the set $\mathcal{V}$ contain the nodes in the graph and let the set $\mathcal{A}$ contain the directed links between the nodes. Let n be the number of nodes, that is, $n = |\mathcal{V}|$. The nodes correspond to the places—the so-called "vias"—on the circuit board where its two sides can be connected. A link represents a possible connection between two nodes. In Figure 1 a line represents two links, one in each direction between two nodes. Each contact pair "starts" at a node and "terminates" at another node. Let k be the number of contact pairs, and let s_l and t_l define the "start" and "terminal" nodes for the pairs $l = 1, \ldots, k$. Let the variable $x_{ijl} \in \{0, 1\}$ denote whether ot not the contact pair l uses the link from node i to node j. In addition let the variable $x_{t_l s_l l}$ represent a link directly from t_l to s_l for $l = 1, \ldots, k$. These variables are interpreted as "direct links" from A' to A, from B' to B, and so on. The corresponding links are not physical, but logical, telling whether or not a connection is possible.

The problem of finding a solution with as many connections as possible can with the above notation be expressed as to (the summations over the index j being simplified to the not perfectly correct expression $\sum_{j=1}^{n}$)

$$\text{maximize} \quad \sum_{l=1}^{k} x_{t_l s_l l}, \tag{1a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} (x_{jil} - x_{ijl}) = 0, \qquad i \in \mathcal{V}, \quad l = 1, \ldots, k, \tag{1b}$$

$$\sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil} \leq 1, \qquad i \in \mathcal{V}, \tag{1c}$$

$$x_{ijl} \in \{0, 1\}, \quad i, j \in \mathcal{V}, \quad l = 1, \ldots, k. \tag{1d}$$

The model (1) can be interpreted as follows. The constraints (1d) make sure that a contact pair passes through a link either once or not at all. The constraints (1c) make sure that each node is passed at most once, i.e., it is used by at most one contact pair. The constraints (1b) make sure that a node that has an incoming connection also has an outgoing one. The objective function (1a) describes the maximization of the number of connections that are made; the flow is circular according to our formulation, whence it is enough to count the "direct links."

If the optimal objective value is less than the total number of pairs that we wish to connect, then there is no way in which they all can be connected, given the circuit board at hand.

4 Lagrangean dual problem

Form a Lagrangian dual problem to (1), where the constraints (1c) are relaxed. The number of constraints in (1c) equals the number of nodes in the network (i.e., n). Let the dual variables be denoted by π_i, $i = 1, \ldots, n$. Since the dual variables are associated with $\leq$ constraints in a maximization problem, there are also constraints $\pi_i \geq 0$, $i = 1, \ldots, n$. The Lagrangian dual problem is then written as that to

$$\underset{\boldsymbol{\pi} \geq \mathbf{0}^n}{\text{minimize}} \quad h(\boldsymbol{\pi}), \tag{2a}$$

where

$$h(\boldsymbol{\pi}) := \max_{\mathbf{x}} \left\{ \sum_{l=1}^{k} x_{t_l s_l l} + \sum_{i=1}^{n} \pi_i \Big(1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil} \Big) \right\}, \tag{2b}$$

$$\text{subject to} \quad \sum_{j=1}^{n} (x_{jil} - x_{ijl}) = 0, \qquad i = 1, \ldots, n, \quad l = 1, \ldots, k, \tag{2c}$$

$$x_{ijl} \in \{0, 1\}, \quad (i, j) \in \mathcal{A}, \quad l = 1, \ldots, k. \tag{2d}$$

The problem (2b)–(2d) is called the *Lagrangian subproblem*; the objective function in (2b) can be rewritten as

$$h(\boldsymbol{\pi}) = \sum_{i=1}^{n} \pi_i + \max_{\mathbf{x}} \left\{ \sum_{l=1}^{k} \Big(x_{t_l s_l l} - \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_i x_{jil} \Big) \right\}.$$

We observe that since π_i is a constant in the Lagrangian subproblem, it can be separated into k independent problems, one for each contact pair.

For a given contact pair l the Lagrangian subproblem is then to

$$\text{maximize} \quad \Big\{ x_{t_l s_l l} - \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_i x_{jil} \Big\}, \tag{3a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} (x_{jil} - x_{ijl}) = 0, \qquad i = 1, \ldots, n, \tag{3b}$$

$$x_{ijl} \in \{0, 1\}, \qquad (i, j) \in \mathcal{A}. \tag{3c}$$

The problem (3) is solved in two steps. First, find the cheapest route from node s_l to node t_l, where the cost for passing node i is π_i. Let z_i, $i = 1, \ldots, n$, describe this route with $z_i = 1$ if node i is passed, and $z_i = 0$ otherwise. In step two, calculate the route cost $\sum_{i=1}^{n} \pi_i z_i$. If this route cost is < 1, then the problem (3) has a solution for which the variables x_{ijl} corresponding to the links in the cheapest route are set to 1. If the route cost is ≥ 1, then all variables for that node connection are set to 0.

The value of $h(\boldsymbol{\pi})$ is, according to Lagrangian duality theory for a maximization problem, an *upper* bound (UBD) on the optimal value for the original problem, that is, an *optimistic* bound.

At iteration t, given the solutions to the k Lagrangian subproblems, i.e., the values of the variables $x_{ijl} = x_{ijl}^t$, a subgradient $\mathbf{d}^t$ can be computed as

$$d_i^t := 1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil}^t, \qquad i = 1, \ldots, n. \tag{4}$$

The following interpretation in the network of the expression (4) is useful when calculating the search direction: For a given contact pair l, the sum $\sum_{j=1}^{n} x_{jil}^t$ equals the number of links used, initiated at the nodes $j = 1, \ldots, n$ that are adjacent to node i.

The dual variables are updated by taking a step in the direction of $-\mathbf{d}^t$, as

$$\pi_i^{t+1} = \max\left\{0; \pi_i^t - s_t d_i^t\right\}, \qquad i = 1, \ldots, n \tag{5}$$

(i.e., a step in the direction opposite to the subgradient). The step length $s_t > 0$ is calculated, e.g., by the formula

$$s_t := \lambda_t \frac{h(\boldsymbol{\pi}^t) - LBD}{\sum_{i=1}^{n} \left(1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil}^t\right)^2}. \tag{6}$$

The value LBD should be a *lower* bound on the optimal value of the problem (a *pessimistic* bound). We choose to set LBD equal to zero here (the value of LBD may also be the value of the best known feasible solution to the primal problem, e.g., determined through a primal feasibility heuristic). To guarantee convergence to an optimal dual solution, the parameter λ_t must have a value strictly between zero and two, i.e., $\lambda_t \in [\varepsilon_1, 2 - \varepsilon_2]$, where $\varepsilon_1 > 0$, $\varepsilon_2 > 0$.

Note that the max operation in (5) is crucial in order to ensure that the multipliers $\boldsymbol{\pi}$ maintain their sign after an update. (Start at a non-negative vector, such as $\pi_i^0 := 0$ or $\pi_i^0 := 1/n$ for all i.)

Given the new values of the multipliers, π_i^{t+1}, $i = 1, \ldots, n$, according to (5), resolve the Lagrangian subproblem (3) until termination. The subgradient method is normally terminated after a fixed number of iterations, since there are seldom enough information on how close to the dual optimum the value is; here, a maximum number of 1000 iterations might be appropriate. The parameter λ_t can, e.g., be initiated to 2 and multiplied by 0.95 every ten iterations. See the article by Feo and Hochbaum and the MSc thesis by Junberg (2001) for alternative rules.

The subgradient scheme is summarized in Algorithm 1.

5 Help functions in Matlab

At your service are two ready-made functions in Matlab. One is a routine for finding k cheapest routes in a graph. The other is a routine for graphically representing a graph and a solution (a wiring).

`nl=gsp(dimX,dimY,pi,k,com)`

`dimX` and `dimY` are the number of "vias" in the x- and y-coordinates, respectively.

Algorithm 1 The subgradient scheme

1: Let $t := 0$ and initialize $\boldsymbol{\pi}^0$ and λ_0
2: Solve the Lagrangean subproblem for $\boldsymbol{\pi} = \boldsymbol{\pi}^t$ and calculate an upper bound $h(\boldsymbol{\pi}^t)$ on the optimal value
3: Calculate a subgradient direction $\mathbf{d}^t$ and the step length s_t
4: Calculate $\boldsymbol{\pi}^{t+1}$ according to (5)
5: Update the value of λ_t
6: Until a termination criterion is fulfilled, let $t := t + 1$ and repeat from 2

`pi` is a vector of dual variable values, with `dimX*dimY*2` elements. `k` is the number of contact pairs and `com` is a matrix describing the initial and ending nodes of each contact pair. For example, `com = [1 48; 2 42; 3 43]`, with `k = 3`, states that the first pair starts at node 1 and ends at node 48, the second starts at node 2, and ends at node 42, and the third pair starts at node 3 and ends at node 43. The result coming from the function `gsp` is a list (`nl`) with node numbers describing a cheapest route for each for each contact pair. For a graph with `dimX` = 8 and `dimY` = 6 the nodes are numbered according to Figure 2.

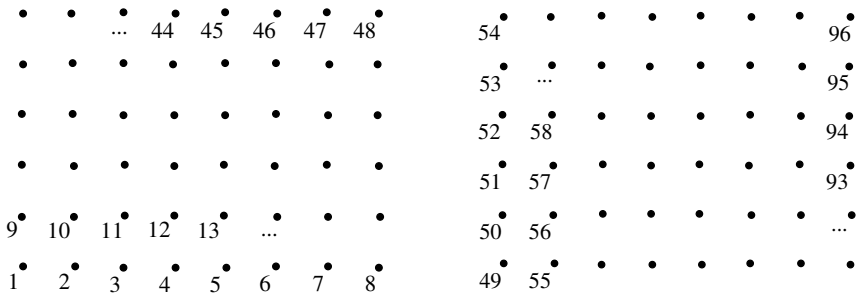

Figure 2: Node numbering for the upper (left) and lower (right) layer.

Example: Let `k = 3` with contact pair `com = [1 48; 3 41; 5 42]`. Suppose all elements in the vector `pi` has the value 0.1. By calling `nl = gsp(8, 6, pi, k, com)` we obtain `nl = 48 47 46 45 44 43 42 41 54 53 52 51 50 49 1 42 60 59 58 57 56 55 2 43 66 65 64 63 62 61 3`. □

Given the vectors `pi`, `nl`, `com`, and `k`, in order to find out which contact pairs have a route cost < 1, use the following code:

```
% Calculate cost per route; remove route with
% cost > 1 (required routes stored in nl and pairs in com)
last = 0;
for i = 1 : k;
    first = last+1;
    slask = find(nl(last+1:length(nl)) == com(i,1));
    last = slask(1)+first-1;
    if (sum(pi(nl(first:last))) < 1)
        okcom = [okcom i]; newnl = [newnl; nl(first:last)];
    end
end
```

Having run this command there is a vector `okcom` containing the indices (row

numbers in the `com` vector) of the contact pairs with costs lower than one, and a vector `newnl` containing all the node numbers for all the routes in these contact pairs, sequentially stored in the vector.

Example (cont'd): With `nl` calculated in the previous example we get `okcom = 2 3` and `newnl = 42 60 59 58 57 56 55 2 43 66 65 64 63 62 61 3` as the result from the above program code. Observe that the connection between 1 and 48 passes 15 nodes and hence has a route cost of $15 \cdot 0.1 > 1$. Therefore, the pair is not included in `okcom` or `newnl`. □

`visagrid(dimX,dimY,nl,com,pi,shift)`

`dimX` and `dimY` are the number of "vias" in the x- and y-coordinates, respectively. `nl` is a list of node numbers describing all the connections between contact pairs. `com` is a matrix describing the start and end nodes for each contact pair. `pi` is a vector of dual variable values with `dimX*dimY` elements. `shift` is a number describing the displacement of the vias on the screen; a good value is 25. See Figure 3.

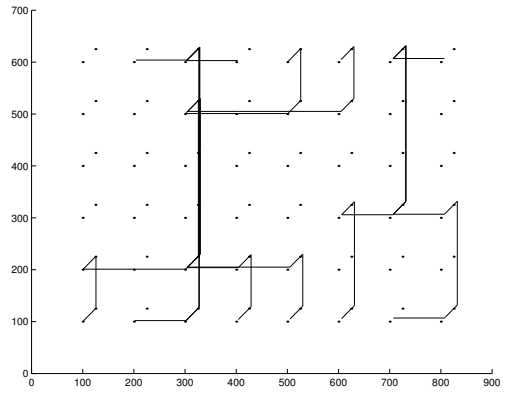

Figure 3: Example result from `visagrid`.

A Appendix: A short command list for Matlab

The Matlab program is stored with a name and the extension ".m". The program is started from Matlab by typing the name of the file you stored (without the .m) at the prompt.

In general, if a row in Matlab is ended with a ;-sign, there will be no screen output. Without the ;-sign Matlab will type out the result. Put a %-sign at the beginning of a row if you wish Matlab to skip it.

In order to get help about a command, type "help < `command` >" at the Matlab prompt. The command "who" shows which variables are defined and their respective sizes.

```
Command          Example              Description
-------------------------------------------------------------------------
                Variables and Matrices
-------------------------------------------------------------------------
```

```
            A=[];               Sets the variable A to zero.

            A=[1 2 7;4 5 6];    Assigns a matrix with two rows and three
                                columns to the variable A.

            A(2,1)=4;           Assigns the element on row 2 and column
                                1 to the value 4.

zeros       A=zeros(3,1);       Assigns to A a matrix with three rows
                                and one column with only zeros.

ones        A=ones(3,1);        Assigns to A a matrix with three rows
                                and one column with only ones.

-------------------------------------------------------------------
                Program control
-------------------------------------------------------------------
            a=13;
while       while (a<=100)      Executes the while-loop where a increases
               a=a+1;           by one until a is 100.
            end

for         for i=1:100         Executes for-loop where a increases by one
              a=a+1;            100 times.
            end

if          if a==100           Checks if 1 equals 100; if so
              a=a+1;            a is increased by 1, otherwise
            else                decreased by 1.
              a=a-1;
            end

-------------------------------------------------------------------
                Operators
-------------------------------------------------------------------
max         [a,b]=max(x)        Calculates the largest value and its
                                index in the vector x. Here, a is the
                                value while b is the index.

mod         a=mod(x,10)         a becomes zero if x is divisible by
                                ten, otherwise a becomes one.

sum         a=sum(x)            a becomes the sum of all the elements
                                of the vector x.

find        a=find(x>1.5)       a becomes the value of all indices of
                                elements of x where the value is larger
                                than 1.5. If more than one such index
                                exists, a becomes the vector of them.

-------------------------------------------------------------------
                Graphs
-------------------------------------------------------------------
plot        plot(y)             Plots a graph of the values in y.
```