Chalmers tekniska högskola
Applied mathematics
Optimization

Project 1
Lagrangian duality
2004-03-12

Project 1: VLSI routing and Lagrangian 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 you can call 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," and the problem and the application are based on the article "Lagrangian relaxation for testing infeasibility in VLSI routing" by Thomas A. Feo and Dorit S. Hochbaum, which is published in *Operations Research*, vol. 54, pp. 819–831 (1986).

The program to be constructed during the project is to be used to decide whether a given placement of a number of connections are possible to implement with respect to the wiring necessary between the components. The circuit board where the connections are to be placed are 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) where it is possible to connect one side to the other on the board.

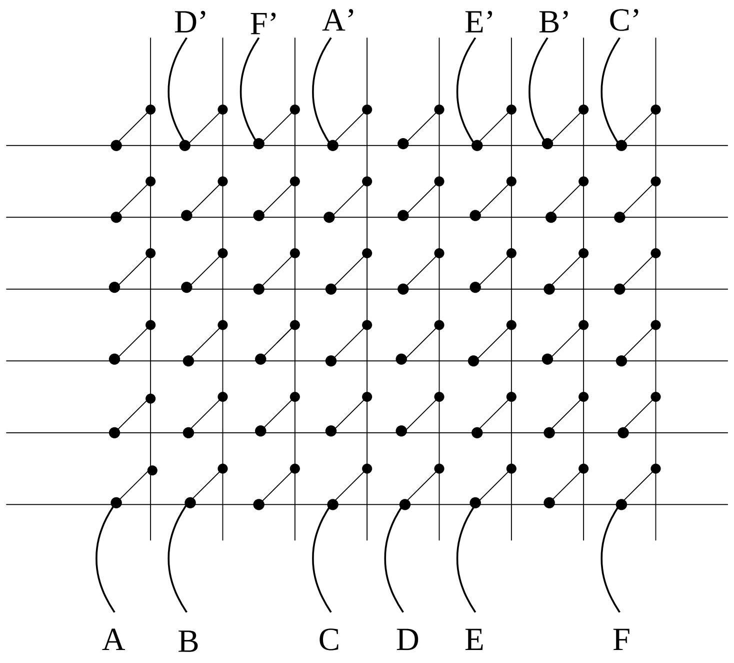

Figure 1: A Manhattan channel routing problem with contact pair A–A', . . . , F–F'.

We illustrate such a problem in Figure 1. In the figure there are six horizontal wires on one side and eight on the other. In addition, six contact pairs, A–A', B–B', up to F–F', are drawn in the figure.

First we introduce notation to mathematically describe the network and how contact points are interconnected. The we present the optimization problem which will tell if 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. *It is the program code for this algorithm that defines your first task.*

With the help of the bounds we obtain on the optimal function value we can then draw conclusions about the existence of feasible solutions to the wiring problem or not.

2 Task 1

Study the problem carefully and write down a sketch of a Matlab program that follows the scheme of Figure 2.

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 web page. The files having the names "p6.m", "p10.m," and "p11.m" contain definitions of `dimX`, `dimY`, `k`, and `com` (which are described on page 5). The problem defined in "p6.m" corresponds to the one in Figure 1, which hence suits as a trial problem for the algorithm coded. This problem lacks solutions to the wiring problem. On the home page that are also two additional functions, described on page 4, and bearing the names "gsp.c" and "visagrid.m".

To be handed in are:

- A print-out of the program code with your name on it. Also, a description in words of what the algorithm does.
- The dual "optimal" objective values for the dual problem, for each problem instance.
- Graphs for each problem instance, where the dual objective value is plotted on the y-axis against the iteration number on the x-axis.
- A conclusion, for each instance of the problem, of whether it has a solution.

3 Task 2

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

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 cannot 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.)

The usefulness of the heuristic is two-fold: (a) it provided us with a pessimistic bound on the optimal objective value, which can be used in a termination criterion for the dual algorithm; (b) it provides us with 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:

- A print-out of this part of the program code together with a description in words of the algorithm, with your name on it.

- The best solutions found in this way for each problem instance.
- Graphs, for each problem instance, where the primal and dual objective values are given against the iteration number.

4 Mathematical model

We introduce mathematical notation to describe the routing problem.

Let the set $\mathcal{V}$ contain the nodes in the graph and let $\mathcal{A}$ contain the *directed* links between the nodes.. Let n be the number of nodes, that is, $|\mathcal{V}| = n$. The nodes correspond to places on the circuit board where we can connect the two side to each other, the so-called "vias. A via is a link from a node on one side to a node on the other. A link describes a possible connection between two nodes. In Figure 1 we represent by a line two links, one in each direction between two nodes.

The contact pairs "start" and "terminate" at a 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 $l = 1, \dots, k$.

We let the 0/1 variable x_{ijl} denote whether contact pair l uses the link from node i to node j or not. In addition we define the variables $x_{t_l s_l l}$ describing a link directly from t_l to s_l for $l = 1, \dots, k$. These variables are interpreted as "direct links" from A' to A, B' to B, and so on. They are not physical links, but logical ones that will tell us if a connection was possible or not.

The problem of finding a solution where as many connections as possible are made can with the above notation be written as the problem to

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

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

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

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

The model can be interpreted as follows: the constraint (1d) makes sure that a contact pair can pass through a link once or not at all; the constraint (1c) makes sure that each node can only be passed once, that is, be used only by one contact pair; the constraints (1b) make sure that a node that gets an incoming connection also gets an outgoing one; the objective function 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 smaller than the total number of pairs that we wished to connect then there is no way in which we can connect them, given the circuit board at hand.

5 Lagrangian dual problem

Form a Lagrangian dual problem where constraints (1c) are relaxed. The number of constraints in (1c) is the same as the number of nodes in the network (n). Let the dual variables be denoted by π_i, $i = 1, \dots, n$. Since the dual variables are associated with $\leq$ constraints in a maximization problem, we have an additional constraint that $\pi_i \geq 0$, $i = 1, \ldots, n$. The Lagrangian dual problem can be written as that to

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

where

$$h(\pi) = \text{maximum} \sum_{l=1}^{k} x_{t_l s_l l} + \sum_{i=1}^{n} \pi_i \left(1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil} \right), \tag{2b}$$

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

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

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

$$h(\pi) = \text{maximum} \sum_{i=1}^{n} \pi_i + \sum_{l=1}^{k} \left(x_{t_l s_l l} - \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_i x_{jil} \right). \tag{3}$$

We observe that since π_i is constant in the Lagrangian subproblem we can separate this problem into k independent problems, one for each contact pair.

For a given contact pair l the Lagrangian subproblem is to

$$\text{maximize} \left\{ x_{t_l s_l l} - \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_i x_{jil} \right\}, \tag{4a}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} (x_{jil} - x_{ijl}) = 0, \qquad i = 1, \ldots, n, \tag{4b}$$

$$x_{ijl} \in \{0, 1\}, \qquad (i, j) \in \mathcal{A}. \tag{4c}$$

This problem is solved in two steps. First, we find the cheapest route between the nodes s_l and t_l, where the costs for passing a node i is π_i. We let z_i describe this route with $z_i = 1$ if node i is passed, and $z_i = 0$ otherwise. In step two we calculate the route cost $\sum_{i=1}^{n} \pi_i z_i$. If the route cost is smaller than one the problem (4) has a solution at which the x variables corresponding to the links in the cheapest route are set to one. If the route cost is larger than one, then all variables for that node connection are set to zero.

The value of $h(\pi)$ 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.

Given the solutions to the k Lagrangian subproblems, that is, the values of the x variables, a subgradient d at iteration t can be computed as follows:

$$d_i^t = 1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil}, \qquad i = 1, \ldots, n. \tag{5}$$

(This expression can be interpreted in the network. The sum $\sum_{j=1}^{n} x_{jil}$ can be interpreted for a given contact pair l as the number of links used, initiated at nodes j, $j = 1, \ldots, n$, which are adjacent to node i. This interpretation can be useful when calculating the search direction.)

The dual variables can now be updated by taking a step in the direction of $-d$:

$$\pi_i^{t+1} = \max\{0, \pi_i^t - s^t d_i^t\}, \qquad i = 1, \ldots, n \tag{6}$$

(a step in the opposite direction of the subgradient since we are solving a minimization problem in the Lagrangian dual), where the step length $s^t > 0$ can be calculated, for example, through the formula

$$s^t = \lambda^t \frac{h(\pi) - LBD}{\sum_{i=1}^{n} \left(1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil}\right)^2}. \tag{7}$$

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 could also be the value of the best known feasible solution to the primal problem. The value of LBD could, for example, be determined through a primal feasibility heuristic.) The parameter λ^t must have a value strictly between zero and two.

We note that the max operation in (6) is crucial in order to be sure that the multipliers π maintain there sign even after an update. (We of course 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 (6), we re-solve the Lagrangian subproblem, and so on, until we terminate. The subgradient method is normally terminated after a fixed number of iterations, since there are seldom enough information on how close to the dual optimal we are; here, a maximum number of 1000 iterations might be appropriate. The parameter λ can, for example, initially be set of two, after which we multiply it by 0.95 every ten iteration. (See also the article for an alternative rule.)

The subgradient scheme is summarized in Figure 2.

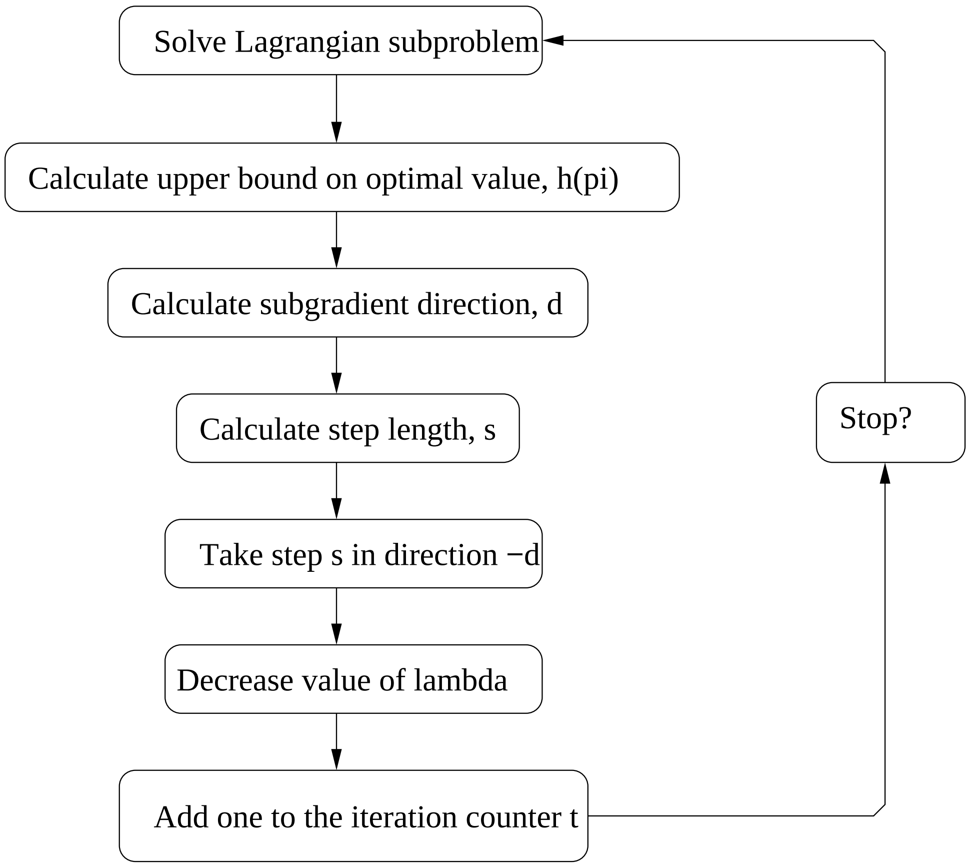
Solve Lagrangian subproblem
Calculate upper bound on optimal value, h(pi)
Calculate subgradient direction, d
Calculate step length, s
Take step s in direction −d
Decrease value of lambda
Add one to the iteration counter t
Stop?

Figure 2: The subgradient scheme.

6 Help functions in Matlab

At your service there are two ready-made functions in Matlab. One is a routine for solving 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-, respectively, y-coordinates. `pi` is a vector of dual variable values, `pi` has `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 other one starts at node 2, and ends at node 42, and the third pair starts at node 3 and terminates at node 43. The result `nl` is a vector with node numbers describing the cheapest routes.

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 3.

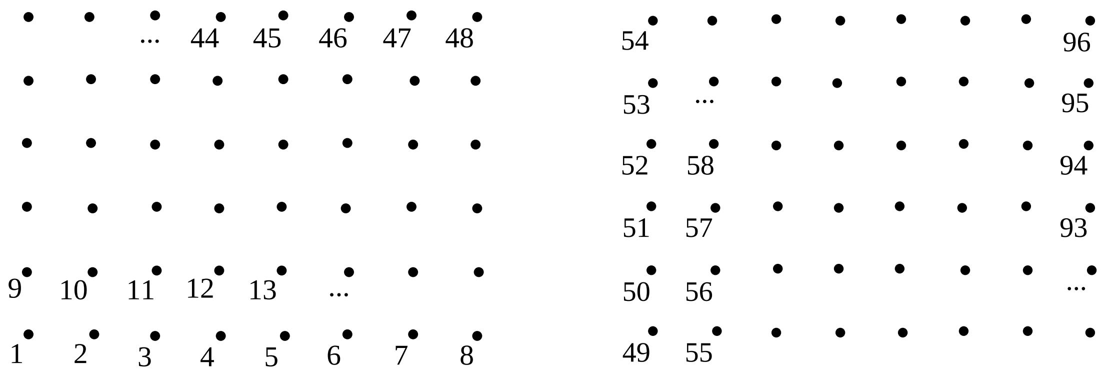

Figure 3: Node numbering for the upper (left) and lower (right) layer.

Example: Let `k = 3` with contact pair `com = [1 48; 3 41; 5 42]`. Suppose all elements in the π vector 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 *lower than one* we can use the following code:

```
% Calculate cost per route; remove route with
% cost > 1 (Requires routes stored in nl and pairs in com vectors.)
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, stored after each other 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. We 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-, respectively, y-coordinates. `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 with dual variable values. `pi` has `dimX*dimY` elements. `shift` is a number describing the displacement of the vias in the screen. A good value is 25.

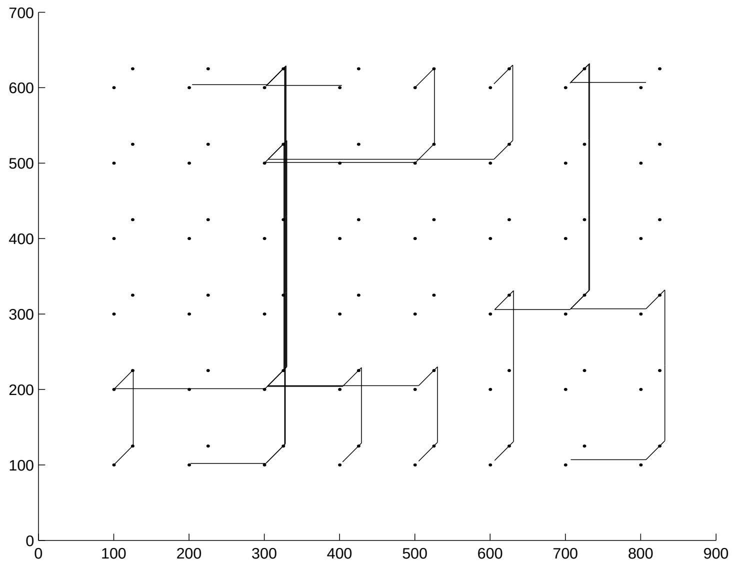

Figure 4: Example result from `visagrid`.

A Appendix: A short command list for Matlab

Matlab programs are preferably written using the text editor Emacs. The Matlab program is stored with a name with 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 < *kommando* >" 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.