DESIGN AUTOMATION
5MD20

coping with hard problems

Ralph Otten
otten@ics.ele.tue.nl
February-March 2010
what is hard?

• depends on the intelligence
  - human age
  - training
• depends on the degree of smartness
  - fast computers
  - size of background memory
• depends on viewpoint
  - how long does it takes
  - exact answer is needed
is anything computable?

• most famous "undecidable" problem: the halting problem

```
while x is not 1 do
  if x is even do x:=x/2
  else do x:=3x+1
```

• chip library problem (Wang's dominoes)

given a set of 1x1 tiles, each side with a particular color
determine whether any integer grid can be tiled with those tiles
such that abutting sides have the same color.

• DNA self-assembly problem

given a set of 1x1 tiles, each side with a particular color
determine whether any integer grid can contain a snake of those tiles
between two given grid edges
such that abutting sides have the same color.
chip library problem
DNA self-assembly problems
unbounded-space computation

during 1920s and 30s, much of the "science" part of computer science was being developed (long before actual electronic computers existed).

many different "models of computation" were proposed, and the classes of "functions" which could be computed by each were analyzed.

one of these models was the turing machine named after Alan Turing

a "turing machine" is just a finite state machine which receives its inputs from and writes outputs onto a tape with unlimited extendability ..... solving "finiteness" problem of finite state machines.
finite state machines

a finite state machine has

• k states (one is the initial state)
  • m inputs
  • n outputs
• transition rules for each state and input
  • output rules for each state
can fsm's solve all common problems?

NO, there exist common problems that cannot be "effectively computed" by finite state machines !!!

for example: multiplying two arbitrary binary numbers

doubling a sequence of "1"s

checking for palindromes

checking for balanced parenthesis


problem:

for arbitrarily long paren sequences, arbitrarily many states are required

(a finite state machine can only keep track of a finite number of objects.)
what is difficult for a computer?

• not computable (undecidable)
  - mostly very general questions
    “can any area be tiled?”
    “can halting be determined for any program?”
  - specific design problems not in this class
• if computable, what does it take?
  - how much time?
  - how much memory space?
  - how many processors?
• what is efficient?
  - time not too fast growing with size
    theory: polynomial time complexity
    practice: even low degree polynomials
    sometimes exponential time
  - polynomial time is always polynomial space
    (but many P-space algorithms are not P-time!)

this lecture: how to cope with intractable problems!
what to do with NP-hard Problems?

- does the complexity depend on the encoding? (0-1 knapsack)
- does the problem have any special features? (channel routing without vertical constraints)
channel routing

• given:
  - an area model (rectangle)
  - pins (fixed on longitudinal sides)
  - a list of nets, i.e. subsets of pins
  - design rules
• constraints:
  - connect all pins of a net
  - design rule correct
• optimization criteria:
  - minimize channel height
  - minimize via’s
  - minimize wire length
rectilinear two reserved layer channel routing

- we assume **rectilinear** routing, i.e. only wire segments parallel to sides of the rectangle
- there are **two layers** available for wires, isolated from each other except when a **via** is made
- the layers are **reserved**, meaning that one layer has all longitudinal segments, called **trunks**, and the other layer has lattitudinal segments, called **branches**
- all segments are on regular grid, of which the longitudinal grid lines are called **tracks** (adjustable in number) and the lattitudinal grid lines are called **columns** (fixed in number)
- **pins** are in the layer of the branches and each column can have two, one or no pins.
classical channel routing problem

• special case of rectilinear two reserved layer channel routing
  • implement each net by a single longitudinal segment (trunk),
  • consequently, the trunk spans (exactly) the entire net, and
  • the pins connect to trunk by one lattitudinal branch each

- problem formulation:
  • given the pin positions on both sides, i.e. column and side
  • produce a net-to-track assignment
  • such that all pins can be connected to the trunk of their net
    these are called vertical constraints, and
  • no trunks in the same track overlap,
    the so-called horizontal constraints
• every overlap of intervals leads to horizontal constraints
• horizontal constraints can be collected in a graph:
  - a vertex for every net
  - an edge for every constraint (the black edges)
• an edge in the complement of the horizontal constraint graph indicates that two trunks might end up in the same track (the red edges)
each vertex represents an interval (of a net). assume that the 'color' of the vertex is the track number.

track assignment = a graph coloring of the horizontal constraint graph:

- **constraint**: no pair of adjacent nodes have the same color = no pair of overlapping intervals got the same track

any valid track assignment provides a valid coloring of the horizontal constraint graph
track assignment and graph coloring

5 colors (= 5 tracks)

4 colors (= 4 tracks)
track assignment and graph coloring

- solutions of classical channels have to satisfy the **vertical constraints**
  (on the previous sheet we were just lucky!)
- if there are **no vertical constraints** then
  any **valid coloring** of the horizontal constraint graph
  yields a **wireable track assignment**
- interval graphs can be optimally colored in polynomial time (F. Gavril, 1972)
- interval graphs with **interval representation**
  can be optimally colored in $O(n \log n)$ time (A. Hashimoto, J. Stevens, 1971)

however, same graph, same coloring .......

NOT wireable !!!!

left edge algorithm
a lower bound on the number of tracks

- a net extends from its leftmost terminal to its rightmost one
- local density at an arbitrary column $C$
  - $\text{ld}(C) = \# \text{ nets split by column } C$
- channel density
  - $d = \max \text{ ld}(c)$ over all $C$
- relationship to horizontal constraint graph?
  - local density $\Leftrightarrow$ clique
  - $d \Leftrightarrow$ size of maximum clique
- lower bound:
  - $\# \text{ tracks} \geq d$

Given a channel routing problem, its channel density is a lower bound on the number of tracks (which is equal to the size of the maximum clique in its horizontal constraint graph).

Remark: the maximum clique size of an interval graph graph with interval representation can be determined in linear time after the intervals are sorted on their left edge (which takes $O(n \log n)$).
the left edge algorithm

1. sort intervals on left edge
2. determine channel density (=number of tracks)
3. prepare a heap for right-most edge in a track
   (initially all right-most edge = 0)
4. for every non-assigned left-most interval
   1. extract track with smallest right-most edge
   2. assign the interval to that track
   3. insert track in heap with new right-most edge

the original algorithm

was differently organized:

1. sort intervals on left edge
2. while not all intervals assigned
   1. open a track
   2. fill track greedily from left side

(A.Hashimoto,J.Stevens, 1971)

the left edge algorithm applies to

- classical channel routing problems without vertical constraints
  - single side channels
  - other problems with pins of at most one net in a column
- linear transistor arrays
- register allocation for straight line codes, some scheduling, etc
linear transistor arrays

metal strips cannot be placed in the same track when their contacts do interleave
what to do with NP-hard Problems?

- does the complexity depend on the encoding?
  (0-1 knapsack)

- does the problem have any special features?
  (channel routing without vertical constraints)

read carefully the conditions!
coloring a graph with the minimum number of colors is NP-hard, but for an interval graph there is an efficient algorithm, and with an interval representation it can be done fastly
• every column with pins of different nets leads to vertical constraint
• vertical constraints can be collected in a graph:
  - a vertex for every net
  - a (black) arc for every constraint
  - direction consistently assigned corresponding to side
  - a vertical constraint indicates
    that the trunk of the ‘tail’ net must be in a track closer to its pin’s side
    than the trunk of the ‘head’ net
• when the vertical constraint graph has cycles, then the channel routing problem cannot be solved in the classical channel routing model

• when the vertical constraint graph is acyclic, then the channel routing problem can be solved in the classical channel routing model

• when the vertical constraint graph is acyclic, the number of vertices on the longest path is a lower bound on the number of tracks needed
channel routing without cyclic constraints

- always solvable in the classical channel routing model
  - for example, by performing a topological sort on the vertical constraint graph
- reduce the number of tracks by merging nets that can be placed in the same track without violating any vertical constraints
  - for example, by a tetris-like technique after the topological sort
- a kind of left edge algorithm, avoiding vertical constraint violations
  - for example, assign and remove sources of the vertical constraint graph and fill tracks from left to right (constrained left edge algorithm)
- optimum solution in the classical channel routing model can be found by branch and bound techniques

B. W. Kernighan, D. G. Schweikert, G. Persky (1973)

minimizing the number of tracks in a classical channel routing model without cyclic vertical constraints is NP-hard

A. Lapaugh (1980)
heuristics for classical channel routing

\[
\text{TOP} = [1, 1, 4, 2, 3, 4, 3, 6, 5, 8, 5, 9] \\
\text{BOT} = [2, 3, 2, 0, 5, 6, 4, 7, 6, 9, 8, 7]
\]

maximum clique size = 4 

longest path = 6 (nodes)
constrained left-edge algorithm

- assign nodes with no incoming edge first
- use tracks top-to-bottom, left-to-right

A.Perskey, D.N.Deutsch, D.Schweikert (1976)
zone representation

- the two constraint graphs contain all the information for track assignment in the classical channel routing problem
- however, all operations on the horizontal constraint graph are performed on its interval representation column-by-column
- for track assignment the interval representation can be replaced by the zone representation, that is ordered maximal cliques

<table>
<thead>
<tr>
<th>net</th>
<th>zone 1 c1 c2</th>
<th>zone 2 c3 c4</th>
<th>zone 3 c5 c6 c7</th>
<th>zone 4 c8 c9</th>
<th>zone 5 c10 c11 c12</th>
</tr>
</thead>
</table>
the column-versus-net matrix for Deutsch' example

the maximal cliques versus nets matrix for Deutsch' example
classical channel routing

- classical channel routing problem uses **two layers**:  
  - one containing the pins and all lattitudinal parts  
  - one containing all longitudinal parts  
  - each net has only one longitudinal part
- **horizontal constraints**: wires in the same layer with overlapping intervals need different tracks
- **vertical constraints**: wires that have pins at the same longitudinal height must change layer before they overlap
- when the vertical constraints form **cycles** then the routing **cannot** be completed in the classical model
- if there are **no cycles** in the vertical constraints then a solution to the classical channel routing problem **exists**, but finding the **minimum number of tracks** is **NP-hard**
what to do with NP-hard Problems?

- does the complexity depend on the encoding?
  (0-1 knapsack)
- does the problem have any special features?
  (channel routing without vertical constraints)
- solve it anyway (e.g. by general purpose optimization)
  (using branch-and-bound, integer programming)
- does it have an equally acceptable formulation?
  (allowing doglegs)
cycles in the vertical constraint graph

-狗腿可以打破垂直约束图中的循环
-狗腿可以减少轨道的数量
-避绕可能使布线可行

狗狗腿是D.N.德鲁希特提出的术语，但只适用于引脚位置。

在'狗狗腿'路由器中最小化轨道数是NP-hard的。

T.G.齐曼斯基(1985)
burstein's channel routing problem

- Dogleg routing cannot solve all channel routing problems (even the simple 2-net 2-column with cycle does not have a dogleg solution).
- Even dogleg routing cannot always be solved in density.
- There is no obvious heuristic for breaking intervals before track assignment.
burstein's channel routing problem

- When doglegs and detours are allowed, any channel routing problem can be solved if there is one longitudinal position without pins.
- Most channel routing problems in practice can be solved without an extra longitudinal position and in or otherwise close to density.
completion guarantee
two layer reserved routing

- doglegs may break cycles in the vertical constraint graph
- doglegs may reduce the number of tracks needed
- detours may render routability
- minimizing the number of tracks in two layer reserved routing is NP-hard
what to do with NP-hard Problems?

- does the complexity depend on the encoding?
  (0-1 knapsack / subset sum)
- does the problem have any special features?
  (channel routing without vertical constraints)
- solve it anyway (e.g. by general purpose optimization)
  (using branch-and-bound, integer programming)
- does it have an equally acceptable formulation?
  (allowing doglegs)
- are there any good heuristics?
rerouting

1. perform a left edge algorithm to obtain tracks
2. order the tracks according to a subset of vertical constraints
3. for every conflict in the routing
   1. remove and reroute
rerouting
rerouting

1. perform a left edge algorithm to obtain tracks
2. order the tracks according to a subset of vertical constraints
3. for every conflict in the routing
   1. remove one of the wires in that conflict
   2. collect maximal available segments in both layers
3. assign a vertex to each segment
4. an edge when segments cross (obviously a bipartite graph)
5. find a minimal connected component containing all pins of the removed net
   1. use the corresponding segments
   2. minimize the segments
6. if such a component does not exist, add a track (and a vertex with edges!) and, if not there, two empty end columns
7. goto 5
rerouting

1. remove a conflicting net, e.g. net 9
rerouting

3. collect all maximal available segments in both layers
3. assign a vertex to each segment
4. an edge when two segments cross
5. find a minimal connected component containing all pins of the removed net
rerouting
rerouting

minimize the segments
rerouting
rerouting

1. perform a left edge algorithm to obtain tracks
2. order the tracks
   according to a subset of vertical constraints
3. for every conflict in the routing
   1. remove one of the wires in that conflict
   2. collect maximal available segments
      in both layers
3. assign a vertex to each segment
4. an edge when segments cross
   (obviously a bipartite graph)
5. find a minimal connected component
   containing all pins of the removed net
   1. use the corresponding segments
   2. minimize the segments
6. if such a component does not exist,
   add a track (and a vertex with edges!)
   and, if not there, two empty end columns
7. goto 5

track addition

dogleg for net 10
this issue with greedy routing

- the concept is to take (small) greedy steps:
  - each time implement the pattern in full detail.
  - no back-tracking! once a wire its laid down, it stays there!

ALARM!
putting down net 1 like this blocks the connection from 5!
new bottom track needed!!!
the greedy routability invariant

• at each step:
  - lay down a partial wire pattern in full detail
    (tracks are not changed anymore)

• such that the following invariant is maintained:
  - 1: the existing layout pattern is drc correct
  - 2: still unrouted pins (nets) remain routable.

• this should guarantee routing completion!

• this works for channel routing
• it does not work for most other routing problems!
column-greedy routing

- idea: sweep a column from left to right through the channel (the part left of the column is fully routed, the right part is not)
- for every column perform the following steps in sequence
  - make feasible top and bottom connections in minimal manner (fig A-F)
  - free up as many tracks as possible by collapsing split nets (fig G-J)
  - add jogs to free to reduce the range of split nets (fig K)
  - add jogs to raise rising nets and lower falling nets (fig L)
  - widen channel to make infeasible top or bottom connections (fig M)
  - extend to next column (fig N)
- if necessary maintain the routability by widening the channel (i.e. add a track to have nets entering the channel from side)

Rivest and Fiduccia (1982)
the greedy channel router

Free up as many tracks as possible by collapsing split nets.

Make feasible top and bottom connections in minimal manner.

Add jogs to free to reduce the range of split nets.

Add jogs to raise rising nets and lower falling nets.

Widen channel if necessary.

Extend to next column.
net greedy routing

- idea: route nets one at a time

- routability invariant maintained by allocating a 'territory' above each unrouted pin (this is a 'keep-out' column)

- the channel is partitioned into 2 halves, each of flexible height

P.R. Groeneveld (1987)
net-greedy gridless routing

using contours (one per layer)

gridless
variable width
variable spacing
completion guarantee

- net greedy: all instances of the classical routing are routable
- ... all except one unlikely situation:
completion guarantee (2)

• adding just one empty column makes the entire channel routable
completion guarantee (3)
what to do with NP-hard Problems?

- does the complexity depend on the encoding?  
  (0-1 knapsack / subset sum)
- does the problem have any special features?  
  (channel routing without vertical constraints)
- solve it anyway (e.g. by general purpose optimization)  
  (using branch-and-bound, integer programming)
- does it have an equally acceptable formulation?  
  (allowing doglegs)
- are there any good specific heuristics?
- use a general-purpose heuristic  
  (like annealing, genetic and evolution analogons)
- does it have an approximation algorithm?  
  (chapter 11: vertex cover, tsp)

read always carefully the conditions!