CS137:
Electronic Design Automation

Day 10: February 6, 2002

Placement

(Simulated Annealing…)

Today

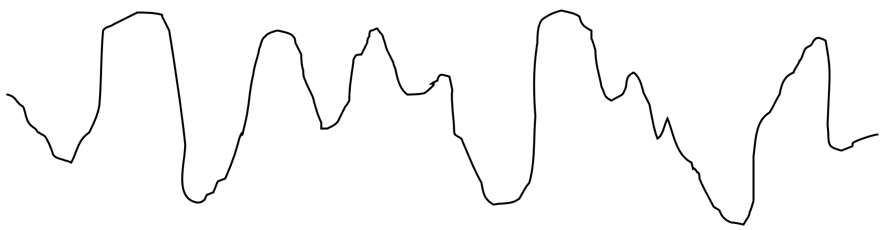
- Placement
- Improving Quality
  - Avoiding local minima
- Techniques:
  - Simulated Annealing
  - Exhaustive (Branch-and-bound)

Simulated Annealing

- Physically motivated approach
- Physical world has similar problems
  - – objects/atoms seeking minimum cost arrangement
  - – at high temperature (energy) can move around
  - – at low temperature, no free energy to move
  - – cool quickly, freeze in defects (weak structure)
  - – cool slowly, allow to find minimum cost

Key Benefit

- Avoid Local Minima
  - – Allowed to take locally not improving moves in order to avoid being stuck



CALTECH CS137 Winter2002 -- DeHon

Simulated Annealing

- At high temperature can move around
  - not trapped to only make "improving" moves
  - free energy from temperature allows exploration of non-minimum states
  - avoid being trapped in local minima
- As temperature lowers
  - less energy to take big, non-minimizing moves
  - more local / greedy moves

Design Optimization

Components:

- "Energy" (Cost) function to minimize
  - represent **entire** state, drives system forward
- Moves
  - local rearrangement/transformation of solution
- Cooling schedule
  - initial temperature
  - temperature steps (sequence)
  - time at each temperature

Basic Algorithm Sketch

- Pick an initial solution
- Set temperature (T) to initial value
- while (T> Tmin)
  - for time at T
    - pick a move at random
    - compute Δcost
    - if less than zero, accept
    - else if RND<$e^{-\Delta cost/T}$, accept
  - update T

Details

- Initial Temperature
  - $T0=\Delta avg/ln(P_{accept})$
- Cooling schedule
  - fixed ratio: $T=\lambda T$
    - (*e.g.* λ=0.85)
  - temperature dependent
- Time at each temperature
  - fixed number of moves?
  - Fixed number of rejected moves?
  - Fixed fraction of rejected moves?

Cost Function

- Can be very general
- Should drive entire solution in right direction
  - reward each good move
- Should be cheap to compute delta costs
  - *e.g.* FM

Cost Functions

- Total Wire Length
- Channel widths
  - probably wants to be more than just width
- Cut width

Bad Cost Functions

- Update cost
  - rerun maze route on every move
  - rerun timing analysis
  - recalculate critical path delay
- Drive toward solution:
  - size < threshold ?
  - Critical path delay

Initial Solution

- Spectral Placement
- Random
- Constructive Placement

Moves

- Swap two cells
- swap regions
  - …rows, columns, subtrees
- rotate cell (when feasible)
- flip (mirror) cell
- permute cell inputs (equivalent inputs)

Variant

- Allow non-legal solutions
  - capture badness in cost function
  - *E.g.* -- allow cells to overlap
- Just make sure cost function makes very expensive as cool
  - settle out to legal solutions

Variant: "Rejectionless"

- Order moves by cost
  - compare FM
- Pick random number first
- Use random to define range of move costs will currently accept
- Pick randomly within this range

- (never pick a costly move which will reject)

Theory

- If stay long enough at each cooling stage
  - will achieve tight error bound
- If cool long enough
  - will find optimum

Practice

- Good results
  - ultimately, what most commercial tools use...
- Slow convergence
- Tricky to pick schedules to accelerate convergence

Big Hammer

- Costly, but general
- Works for most all problems
  - (part, placement, route, retime, schedule…)
- Can have hybrid/mixed cost functions
  - as long as weight to single potential
- With care, can attack multiple levels
  - place and route
- Ignores structure of problem
  - resignation to finding/understanding structure

Optimal/Exhaustive

- If you run simulated annealing long enough….
  - It should converge to optimum
- If you have enough monkeys typing at keyboards keyboards long enough
  - They'll eventually produce the works of Shakespeare….

Brute Force?

- If you are really going to give up on structure and explore the entire space
  - …there are more efficient ways to do it
- …and maybe they're not terrible
  - For small/modest problems
  - As our computers get faster

Optimum Placement

- Simplest case:
  - Gate/array (FPGA) w/ fixed cell locations
- N locations
- M cells
- Try all permutations of N choose M

  N!/M! cases

Improving

- Prune off symmetry cases
  - Rotate 90, 180, 270
  - Mirror X, Y, XY
- Reject provably bad starts
  - (return to in a minute)

Exhaustive Placement

- More general:
  - Modules have variable size
  - Modules can be rotated/flipped…
- To explore all cases:
  - For each module
    - For each orientation

Branch-and-Bound

- Flavor now
  - More next week
- Consider dense 1D placement
  - Search space of all placements
  - Tree branch is choice of logical cell for physical cell position
  - Keep track of best solution so far as reach leaves
  - If partial solution worse than best solution, prune branch

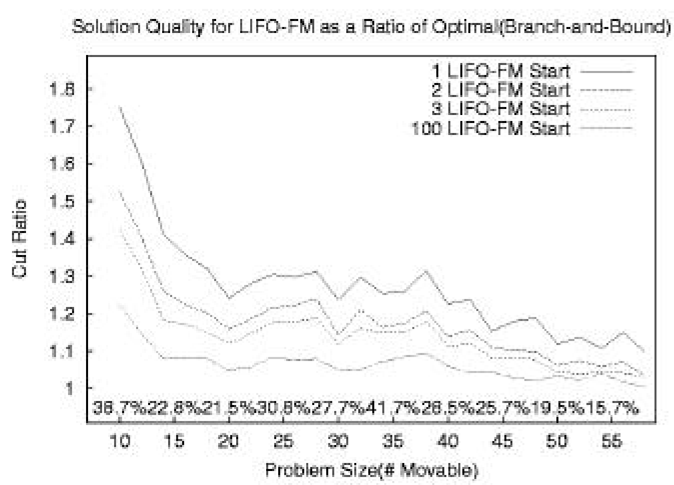
Pruning: 1D example

- Reducing channel width
  - Have solution with width=10
  - When find a partial solution with width>=10
    - Can abort that branch
- Reducing Delay
  - Have solution with delay=20
  - When find a partial solution with delay>=20
    - Can abort branch

Viable

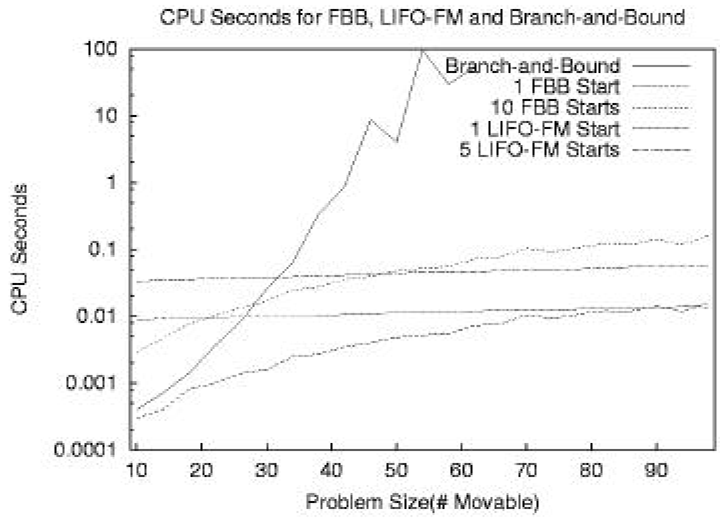
- Only on small problems
  - But "small" growing with machine speed
- Use for end-case in constructive
  - Flatten bottom of hierarchy
  - Maybe even in iteration/overlap relaxation

Caldwell *et. al.* Results



[TRCAD v19n11p1304-13]

Runtime



[TRCAD v19n11p1304-13]

Faster?

- Accounting and gain complexity
  - Make it linear time
  - But do make each update somewhat complex
  - Exhaustive case less bookkeeping
    - Not an atypical result…

Summary

- Simulated Annealing
  - use randomness to explore space
  - accept “bad” moves to avoid local minima
  - decrease tolerance over time
- General purpose solution
  - costly in runtime
- Small (sub)problems
  - May solve exhaustively
  - Can prune to accelerate…

Big Ideas:

- Use randomness to explore large (non-convex) space
  - Simulated Annealing
- Use dominance to quickly skip over obviously bad solutions
  - Branch and Bound