CS294-7: Reconfigurable Computing

Day 15: Tuesday, March 11

Interconnect

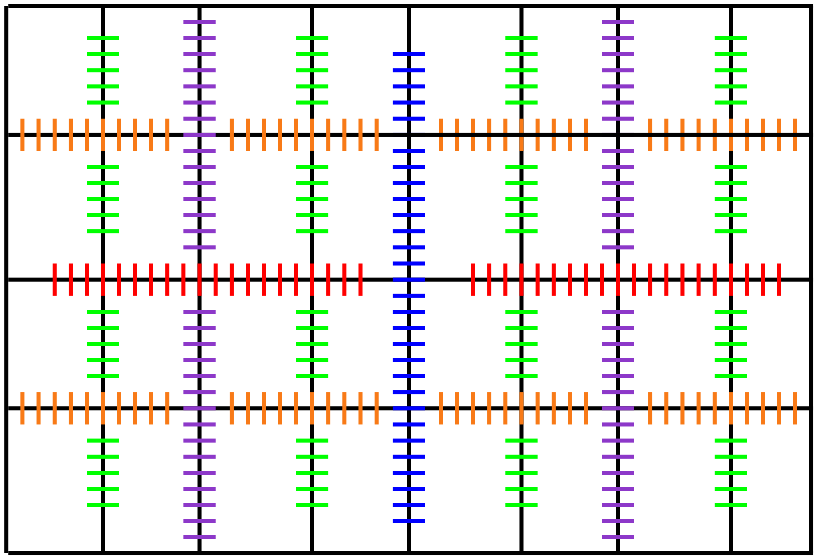
- Admin
- Partitioning (and Placement)

Admin

- Project 1
  - Due Today (3/11)
  - Netlist request
  - Talks Thursday
- Project 2
  - Assignment out today
- Office Hours this week
  - Today: try 631$\rightarrow$313
  - Wednesday/Thursday ILP (catch by BRASS poster on 6)
  - Friday out
  - by arrangement

Partitioning

- "solves" placement for strict hierarchical interconnect (*e.g.* Aggarwal and Lewis, also HP PLASMA)
- Often used to initialize placement
- Balanced partitions NP-complete, in general
- Fast heuristics exist (Fiduccia-Mattheyses)
- Doesn't address critical path delay



Partitioning Problem

Given: netlist of interconnected cells

- Partition into two (roughly) equal halves (A,B)
- minimizing the number of nets shared by halves

"Roughly Equal"
Balance Condition: $(0.5-\delta)N \leq |A| \leq (0.5+\delta)N$

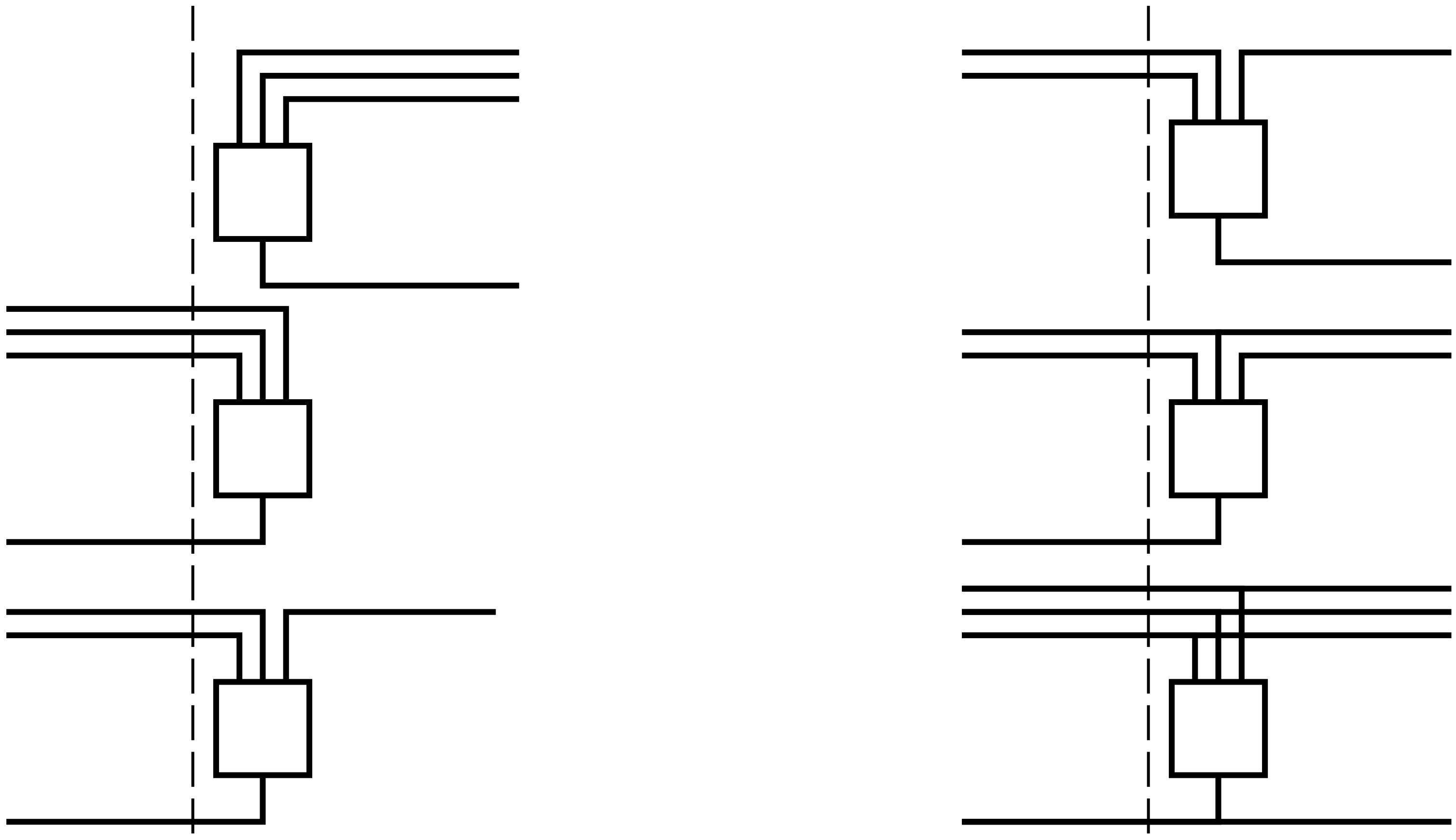
Fiduccia-Mattheyses

(refinement on Kernighan-Lin)

```
Randomly partition into two halves
Repeat until no updates
  Start with all cells free
  Repeat until no cells free
    Move cell with largest gain
    Update costs of neighbors
    Lock cell in place
    Note current cost
  Pick least cost point in previous
    sequence and use as partition
```

Repeat for several different starting points

Fiduccia-Mattheyses (cell gain)



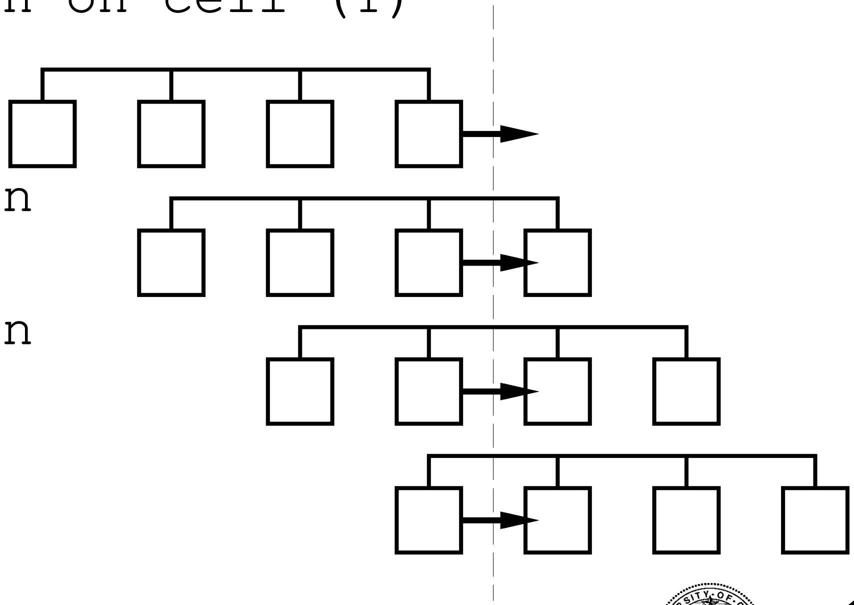
Fiduccia-Mattheyses (recompute cell gain)

- **For each net, keep track of number instances in each partition**
- **critical net** has one element on one side

Move Update:

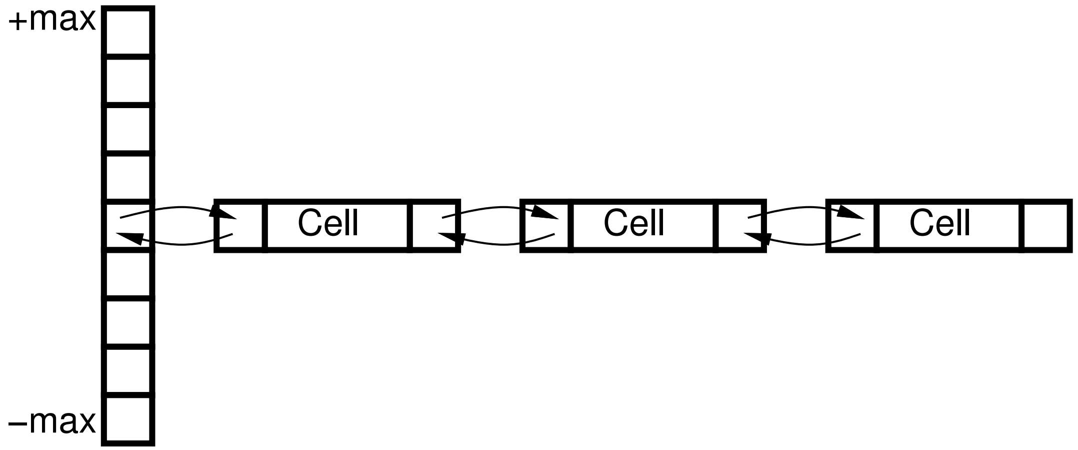
```
For each net ∈ inputs+outputs
      of move cell
   if T(net)=0 increment gain all cells on net (F)
   if T(net)=1 decrement gain on cell (T)
   decrement F(net)
   increment T(net)
   if F(net)=1 increment gain
       on cell (F)
   if F(net)=0 decrement gain
       all cells on net (T)
```

;; F==From Partition
;; T==To Partition



Fiduccia-Mattheyses (data structures)

- N cells
  - partition counts A, B
  - consumers
  - inputs
  - locked status
- two gain arrays



Fiducccia-Mattheyses (Optimization Sequence)

+3	+2	-1
+3	+1	-1
+2	0	-1
+2	-1	0
+1	0	+1
0	-1	-1
+1	-1	-1
0	-2	-2
0	-2	-2
-1	-1	-2
+1	0	-1
-1	-1	-2
-2	-2	-1
-2	-3	-3
-3	-3	-3
-3	-3	-3
+12	**+3**	**0**

Fiducccia-Mattheyses (Running Time)

```
Randomly partition into two halves
Repeat until no updates
  | Start with all cells free
  | Repeat until no cells free
  |   | Move cell with largest gain
  |   | Update costs of neighbors
  |   | Lock cell in place
  |   | Note current cost
  | Pick least cost point in previous
  |   sequence and use as partition
```

Repeat for several different starting points

Fiduccia-Mattheyses (Running Time)

- **Claim: small number of passes (constant?) to converge**
- **Small, constant number of random starts**
- N **cell updates**
- **Updates** K **+ fanout** (average fanout K)
- **Maintain ordered list** $O(1)$ **per move**

Running time: $O(KN)$

Tweaks on Fiduccia-Mattheyses

- clustering
- technology mapping
- initial partition
- number runs
- maximum partition size variation (δ)
- replication

(Comparisons from Hauck and Boriello '96)

Clustering

1. group together several leaf cells into one larger for FM partitioning
2. run partition on clustered cells
3. uncluster, keep partitions (uncluster iteratively rather than all at once)
4. run partition again (with previous step as initial partition)

Benefits:

- catch local connectivity global algorithm may miss
- runs faster (smaller N)
- FM work better with 6+ input nodes (?)

Connectivity Clustering

- examine nodes in random order
- cluster node with neighbor with highest "connectivity"

$$con_{ij} = \frac{bw_{ij}}{size_i \cdot size_j \cdot \left(fanout_i - bw_{ij}\right) \cdot \left(fanout_j - bw_{i}j\right)}$$

$$bw = \frac{1}{\left(\textbf{cells on net}\right) - 1}$$

- best of several techniques
- 30% better than random clustering
- 16% faster than random clustering

Technology Mapping

Better to partition at “gate” level, than after mapped to LUTs.

Initial Partition

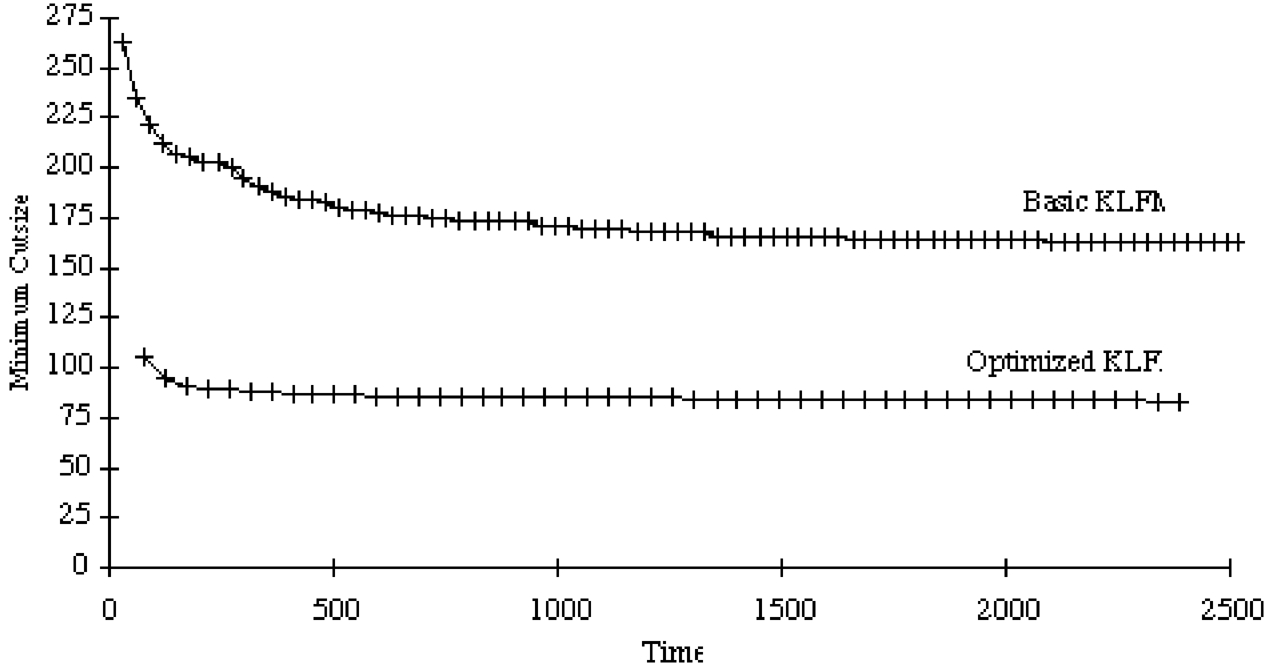
Random best:

- Random (82.4)
- Seeded (95.3)
- Breadth-First (86.7)
- Depth-First (86.5)

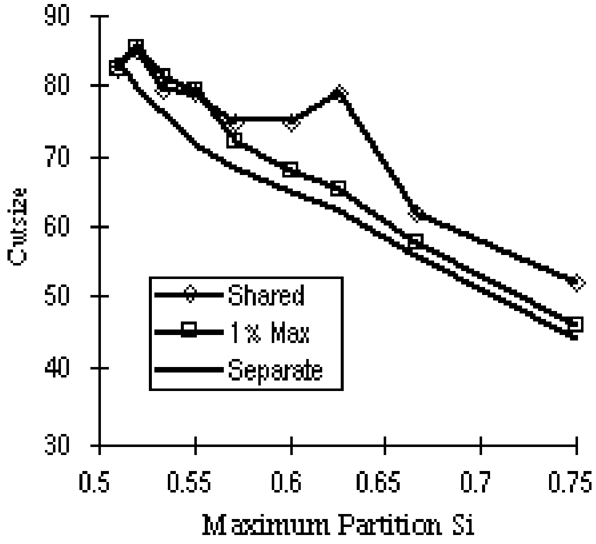
v/s spectral initial:

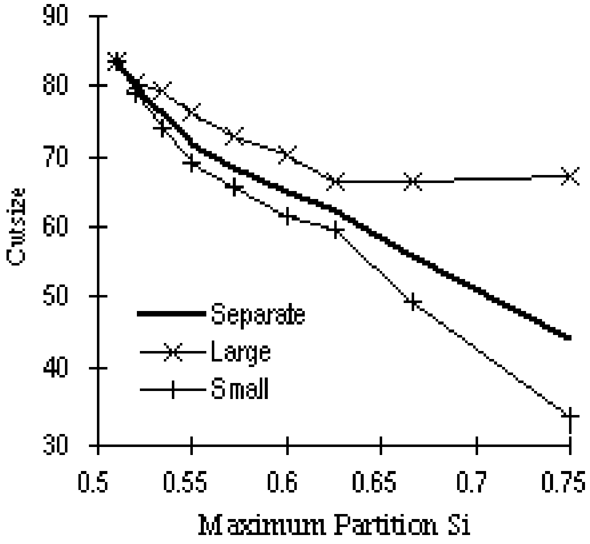
- Random (68.6)
- spectral (73.5)

Number of Runs

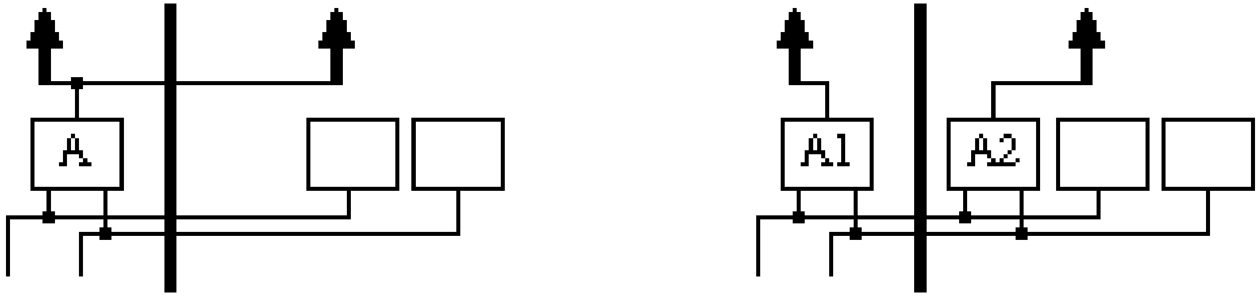


Partition Size Variation





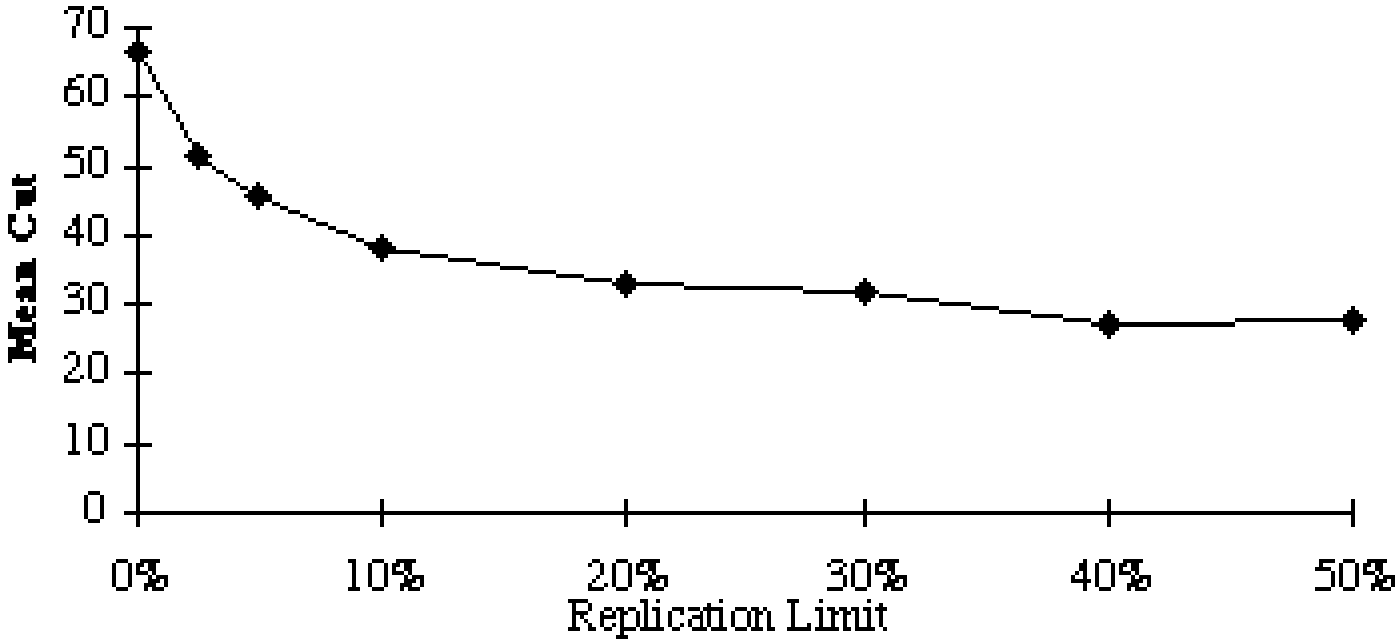
Replication



Trade some additional area logic for smaller cut size.

(replication data/figures from Enos, Hauck, Sarrafzadeh '97)

Replication



- 5% additional logic → 38% smaller cut
- 50% → 50+% smaller

Partition Wrapup

Hack and Boriello "Optimized"
– half size of vanilla FM.

30-40% better if allow 5% replication.

All tweaks address partition quality and execution time not critical path.

?are there ways to fold time into this heuristic?

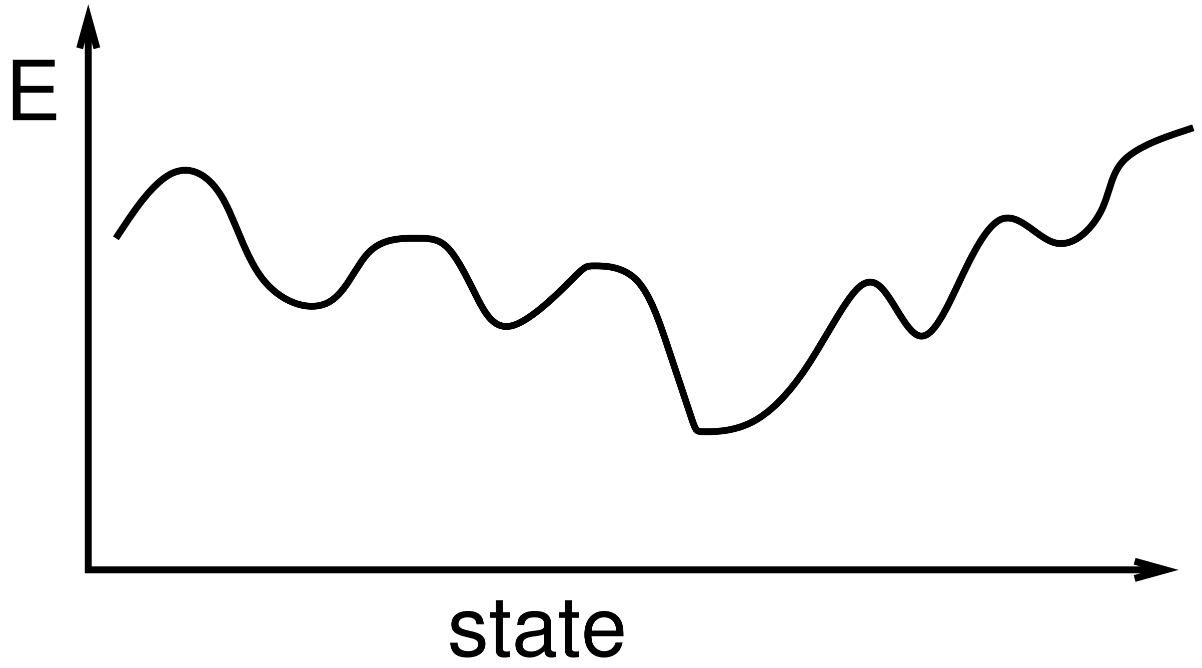
Look for a bigger hammer: Simulated Annealing

Simulated Annealing

- Analogy to cooling of materials: search for minimum cost $\sim$ search for minimum energy state in physical system.
- Atoms trying to find minimum energy
- Thermal energy (kT) allows atoms to move – changing configurations within energy state.

  i.e. energy barrier less than kT
  $\rightarrow$ thermal energy allows intermediate fluctuations necessary to move between energy states.
- Random walk – influenced by energy function.
- T high – lots of energy, atoms free to move around
- T low – little free energy, atoms localized

Simulated Annealing



Simulated Annealing

Physical system, we know:

if cool too fast, will freeze in defects (high-energy states); won't find structured, low-energy states —only local minima

- must carefully anneal
- lower temperature slowly
- spend considerable time around freezing point
  so atoms can relocate themselves
  into minimum energy states
  before they loose sufficient energy to move

Metropolis Algorithm

for simulating collection of atoms

1. give each atom a random displacement (E)
2. if $E \leq 0$, keep
3. if $E > 0$:
   keep with probability $P(E) = e^{-E/kT}$

Repeat multiple times to simulate thermal movement of atoms at temperature T.

Simulated Annealing

- Melt material (start with high T)
- lower temperature slowly
- stay at each temperature long enough to reach steady state
- stop when temperature low enough to "freeze"
  – no further changes

Annealing schedule – sequence of temperatures and lengths of time at each temperature

If cool slowly enough, find state close to minimum energy state.

Optimization Analog

Energy $\rightarrow$ cost function
Temperature $\rightarrow$ freedom to make non-greedy moves.

- Start at high temperature – most any move accepted
- Lower temperature slowly – makes more "greedy"
- At $T = 0$ reduces to greedy moves only

Using SA

To use:

- Identify moves
- Define cost function

Power of technique is $\sim$ arbitrary cost function.

Formulation Partitioning

Partitioning:

- **Move:** swap partitions
- **Cost:** cut-set and balance

$$E = C_1\,(\text{cut}) + C_2\,(|A| - |B|)^2$$

Partitioning and Timing

Easy to add constraints to cost function.

$$E = C_1\,(\text{cut}) + C_2\,(|A| - |B|)^2 + C_3\,(\text{critical path partition crossings})$$

May be tricky to calculate E efficiently.
(often the limit to complexity/accuracy of E will be evaluating a very large number of potential moves)

Formulation Placement

- **Move:** swap location of two cells
- **Cost:**
  - total wire length
  - channel congestion
  - wire delay
- may limit distance swaps considered by temperature

Simulated Annealing Wrapup

- Big-hammer for hard optimization problems
- General cost model – accommodates most any constraints
- If cool slowly enough, will get good results
- Finesse in working out parameters
- Cost should be cheap to update
- Annealing schedule can be tricky to optimize (balance speed versus quality)
- ...generally takes a long time...
- (...why PPR is slow)