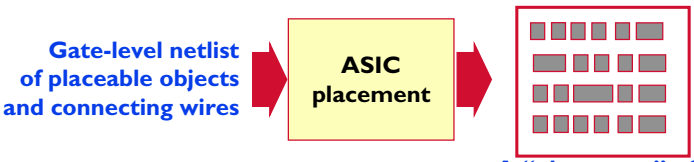
(*Lec 12*) ASIC Placement & Partitioning: (I)

- What you know about layout
  - Probably not much, at this point...
- What you don't know about *placement*...
  - Placement: which gates go where on the chip
  - Approaches: 3 big ideas here--*recursive*, *iterative*, & *direct* placement



A "placement" of
the gates in appropriate
location to "optimize" layout

Copyright Notice

Where Are We?

- Physical design--how to geometrically *place* gates in a netlist?

	M	T	W	Th	F	
Aug	27	28	29	30	31	1
Sep	3	4	5	6	7	2
	10	11	12	13	14	3
	17	18	19	20	21	4
	24	25	26	27	28	5
Oct	1	2	3	4	5	6
	8	9	10	11	12	7
Midsem break	15	16	17	18	19	8
	22	23	24	25	26	9
	29	30	31	1	2	10
Nov	5	6	7	8	9	11
	12	13	14	15	16	12
Thnxgive	19	20	21	22	23	13
	26	27	28	29	30	14
Dec	3	4	5	6	7	15
	10	11	12	13	14	16

- Introduction
- Advanced Boolean algebra
- JAVA Review
- Formal verification
- 2-Level logic synthesis
- Multi-level logic synthesis
- Technology mapping
- ***Placement***
- Routing
- Static timing analysis
- Electrical timing analysis
- Geometric data structs & apps

Handouts

- Physical
  - **Lecture 12 -- ASIC Placement & Partitioning**
- Electronic
  - **Nothing new...**

ASIC Placement: First-Order Problem

- What are we trying to do with placement?
  - Input: a netlist of connected gates and nets
  - Output: exact location on the chip of each gate
  - Optimization: *make sure we can connect all the wires*
- Is this hard?
  - Yes. A bad placement can require *dramatically* more wiring.
  - More wiring is bad: we might need more "white" space for wires
  - ...and long wires have more delay, so affects overall speed too.
  - If your placement is *very* bad, the next tool in the layout flow--the router--may not even be able to find paths for all the wires.
  - (Even if your placement is pretty good, might not be able to connect all the wires in ways that let chip function a the *speed* you intended…)

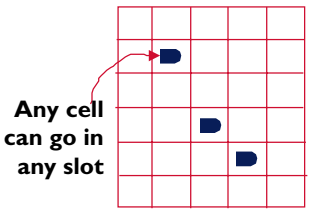
For Any Placer: 3 Big Questions

- Layout model
  - What constraints or limitations on the *shapes of individual placeables*?
  - What constraints on the shape or organization of the *chip* itself?
- Optimization
  - What exactly does the placement algorithm try to *optimize*?
  - Turns out there are several viable alternatives
- Legalization
  - Intermediate: if you stop the placer in the *middle* of running, do you get a *legal layout* (even tho it might be a *mediocre* layout)?
  - Final: at the *end* of the algorithm, is the result a real, legal placement, or does it require extra *backend* effort to finish it, *legalize* it?

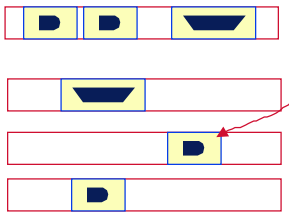
Layout Model: Issues

- Layout model
  - What do we know about geometric *shapes* of objects we are placing?
  - What constraints do we have on *where* they are allowed to go?

Simplest model:
all objects are "points"
placed in "slots"
in a simple uniform grid

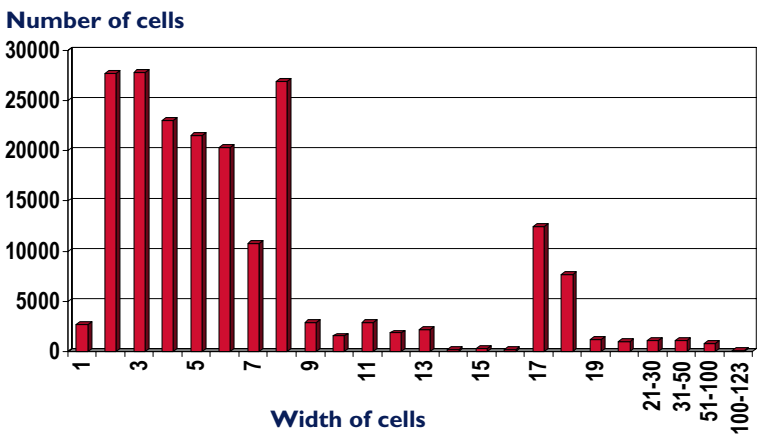


More realistic ASIC model:
all objects are rectangles of
varying width, same height,
placed in rows with variable
(but likely to be minimal) separation



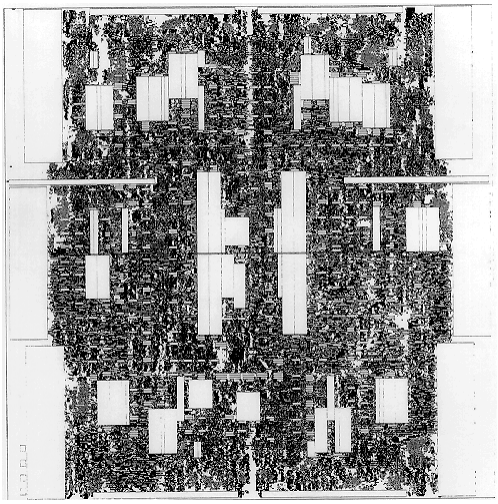
Reality Check: Row-Based Layout Model

- The row-based objects *really* do come in different widths
  - "Width" you can think of as "how many IO pins wide"
  - Example: 200K gate IBM ASIC [J. Vygen, DAC98]



Reality Check: Row-Based Layout Model

- ..and, you do still have to deal with random logic + big blocks
  - Blocks called "macros," examples are memories, registers, ALUs, etc
  - From [Vygen DATE98], 200K gates + blocks
- But, we ignore all this
  - For us, placeable gates look like "points"
  - 18-763 does more algorithms for when you have "lots of shape" in your placeables

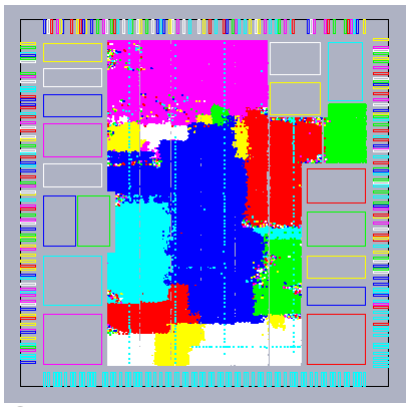


Aside: Macro-Blocks vs Atomic Gates

- In a really big design, you don't do always placement "Flat"
  - "Flat" means "place all the gates at the same time, across the entire surface of the chip"
  - Opposite of "flat" is what? *Hierarchical*
- Typically divide design into big chunks, then do 2 steps
  - **Floorplan:** just like rooms in a house, plan the arrangement of these big blocks on the surface of your chip, then try to lay out each block
  - **Detailed, block-level layout:** for each block, place it and route it
  - **Chip-level assembly:** put blocks back on the chip surface, deal with any surprises (example: "*oops! too big!*"), then route the global wires between the blocks
- Today, block-level layout common up to ~ 500k gates, flat

Aside: About Floorplans

- Floorplan boundaries at chip level may not be "strict"
  - Its up to the style of the layout tools if gates are required to stay "inside" their original floorplan blocks, or can move around, later



© Larry Pileggi

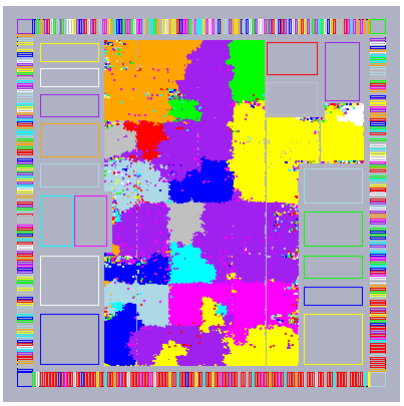
This is a chip placement in which the "hard" macro blocks are empty rectangles at left, right, and the gate-level logic blocks are individually colored.

In this example, the gates are allowed to move outside of some "soft" (think: *squishy*) floorplan regions.

Floorplan pic courtesy L. Pileggi, Monterey Design

Aside: About Floorplans

- …and, here is a much "flatter" placement of same chip



© Larry Pileggi

Same"hard" macro blocks at left, right.
But now the gate-level logic blocks are much more blended into each other, since this placement was done more flat, without the previous floorplan constraints.

(It's an active research problem, how far we can push "flat" layout.)

Floorplan pic courtesy L. Pileggi, Monterey Design

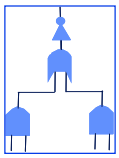
One More Aside: About Layout "Size"

- Terminology a bit vague: how big is a "5 million gate ASIC"
  - Surprisingly, its almost certainly NOT 5,000,000 logic gates
  - These "gates" numbers are sort of like "equivalent small gates"
  - …sort of like transforming everything into a 2-input or 4-input NAND

Size of this? 1 gate

NAND2

Size of this? ~4 gates

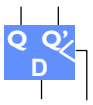


AOI22

Size of this? ~6 gates

1bit adder

Size of this? ~10 gates

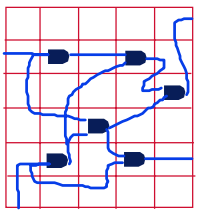


D FF

- Consequence: 2 measures people use for "size" here
  - Gates: this is "equivalent little NAND gates". Usually a big number
  - Placeables: how many things the placer *really* places. ~Rule: Gates ÷ 4

Classical Placer Optimization Goals

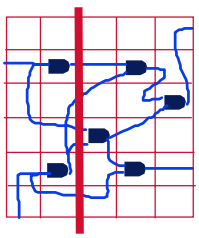
- Total estimated wirelength



Add up estimated-length for all nets in the placement.

This Σ_i length(net i) is what the placer tries to minimize

- Congestion minimization



Take any cut through the placement. Count the number of nets that cross this cut line.

For every cut line, placer tries to minimize this crossing count.

Optimization: Minimum Wirelength

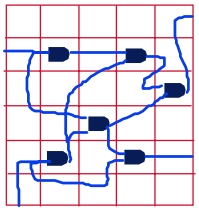
Wirelength minimization

- Every placer tries "make it possible to route all the wires"?
- We must translate this into a concrete goal for the placer.
- Classical goal: Σ_i length(net i) is to be made as small as possible

New problem

- How do we *estimate* the required total wirelength for a placement?
- This is our estimate of of the "quality" of any candidate placement

Σ_i length(net i) == ?



Placement: Wirelength Estimation

Some facts

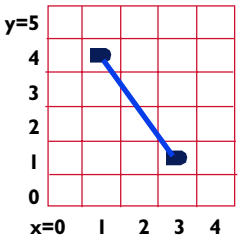
- You have to estimate the total wirelength because it's too expensive in CPU time (usually) to really call the routing tool for each wire
- So, the "estimator" is supposed to give a reasonable guess for the wirelength, but be really quick to compute

Wirelength estimators

- Many *many* different types
- Depend on what assumptions you can make about how the wires will actually get routed in the final ASIC layout
- Also depends on how much CPU time you can afford
- Let's look at a few classical strategies

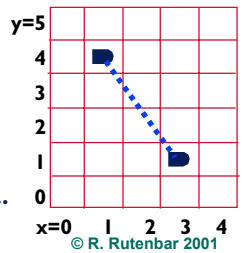
Wirelength Estimation

- Euclidean estimation
  - For a 2-point net, just the hypotenuse of the triangle.
  - Problem: nobody really allows wires at arbitrary angles in most chips



Estimate is:

- Manhattan estimation
  - For a 2-point net, just the sum of the legs of triangle
  - (Name from pt-to-pt distance measured by NY cab drivers)
  - Perfectly OK for 2 point nets...

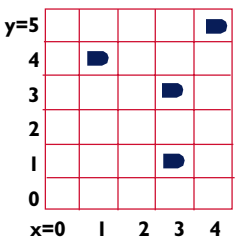


Estimate is:

Wirelength Estimation

- What happens if >2 endpoints on the nets?

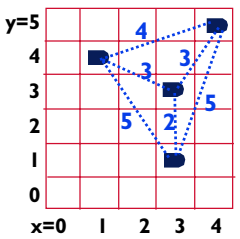
- Several options
  - Can use the simple trick of putting a 2-pt connection between *all* pairs of points...
  - ..but this dramatically overestimates the necessary wirelength



Estimate is:

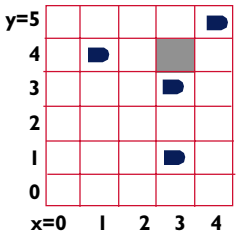
Wirelength Estimation

- Better idea
  - Take the subset of those connections that has minimum overall length, but touches every point
  - Called "minimum spanning tree" -- $O(N^2)$ algs to get it for N points



Estimate is:

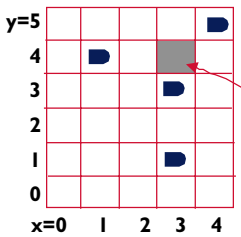
- Problems
  - It still overestimates the wire needed, since it assumes wire is made only of discrete gate-to-gate connections



Estimate is:

Wirelength Estimation

- OK, how would a real router tool wire it?
- As a "Steiner tree"
  - Difference is the Steiner tree can have connection points at *arbitrary* places, not just at the spots where there are endpoints of net
- Problem
  - Getting an optimal Steiner tree is NP Hard, ie, exponentially hard in general case.
  - There are good heuristics, though, but its still expensive to do really well.

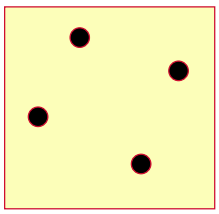


Estimate is:

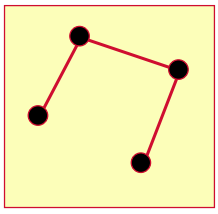
a Steiner pt

Aside: About Steiner Tree Constructions

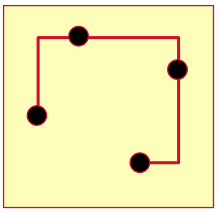
- Bigger, clearer Steiner example



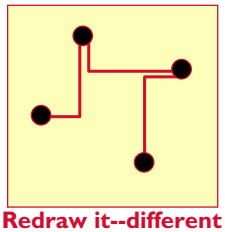
Pins to connect



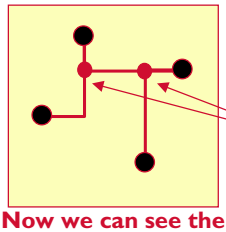
Min Spanning Tree



Draw it manhattan



Redraw it--different orientations of 2pt paths



Now we can see the better Steiner tree

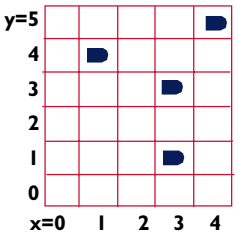
2 so-called "Steiner-points"

Aside: About Steiner Tree Constructions

- Can I always just "tweak" the minspan tree to get best Steiner?
  - Example on previous page "flips" L-shaped paths, maximizes overlap
  - Answer: No. There are optimal Steiners you cannot find this way
- OK, so how much better (shorter) is Steiner over minspan tree?
  - Big result: [F.K. Hwang 1976]
  - *Minspan tree never longer than 1.5X length of the optimal Steiner tree*
  - Said the other way: *going to Steiner tree saves at most 1/3 of length*

Wirelength Estimation

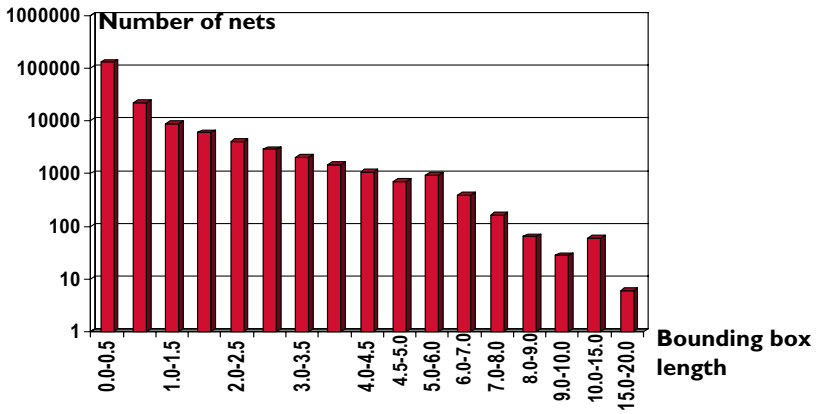
- OK, what do we really use?
- Half-perimeter metric
  - Put a box around all the pins
  - Take 1/2 of perimeter, which is just length + width of box
  - This is a guaranteed lower bound on the amount of wire you need
  - (Why?)
  - This is really easy to compute, widely used.
  - Note, for 2-point nets this IS the Manhattan estimate!



Estimate is:

Reality Check: Wirelength Estimation

- Half-perimeter metric
  - Real distribution of bounding-box sizes for big IBM ASIC [Vygen DATE98]
  - 14.6mm², 181K nets, total wirelength: 106.34 meters



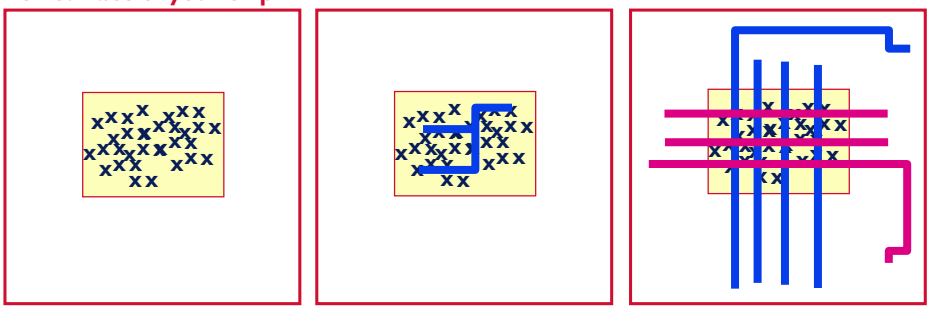
Optimization: Congestion Minimization

- Wirelength minimization is not only option
  - Small total wirelength is good: shorter wires take up less space, have less delay, etc
  - BUT--still easy to place too many gates so close you cannot wire them
  - *Estimated* wirelength does not account for *congestion*, ie, there is more demand for wires than supply of wires in a region of space
- Can target congestion instead of wirelength
  - Note they do tend to correlate, but minimizing one does not *neccessarily* optimize the other

Congestion vs Wirelength

- Common problem

Densely placed region on surface of your chip



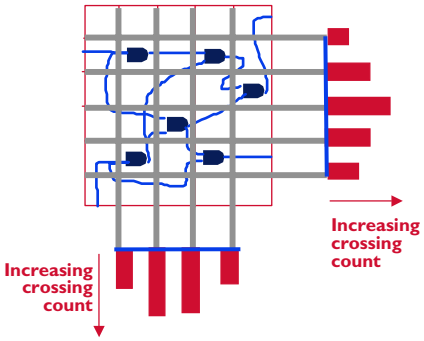
…wirelength may be very good, very small

…but can you fit all the local wires, that connect gates just inside this region?

…and is there enough space for global wires, that don't connect inside here, to pass thru?

Congestion Histograms

- A simple model shows how wirelength, congestion relate
  - Make several uniformly spaced cuts across layout, both directions
  - Count num of wires that must cross each cut; plot values as *histogram*



Note:
Area under each histogram correlates with (but is not same as) estimated wirelength

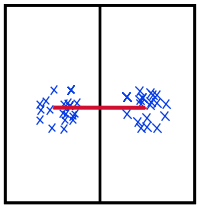
Typically, we want to "*flatten*" these histograms, so there are no regions with more wires than the max num of wires that will fit. We are especially sensitive to the *peaks* (maxima) in these plots, since they are likely hotspots for congestion.

Three Big Placer Strategies

- Recursive (bipartitioning)
  - Recursively partition the netlist onto halves of the chip
  - We cover: Kernighan-Lin and Fidduccia-Matthyses algorithms
- Iterative improvement
  - Perturb a random placement repeated until it stops getting better
  - We cover: Simulated annealing algorithm
- Direct (quadratic)
  - Write an equation (a big one) whose numerical solution = a placement(!)
  - We cover: classical quadratic placement

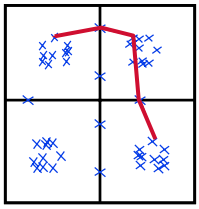
Strategy: Recursive Placement

- Usually called "min-cut" placement
  - Recursively divide chip surface into 2 parts, and partition gates across the halves to minimize the number of wires across the cut
  - Min-cut minimizes congestion directly, doesn't minimize wirelen directly



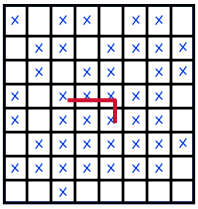
Initial Partition

- Gates swapped across partition to find min-cut
- Pin position estimated at center of partitions



Intermediate Partition

- Gates increasingly localized

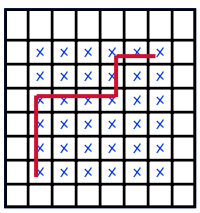


Final Placement

- Eventually all circuits placed near legal locations
- Exact pin positions known

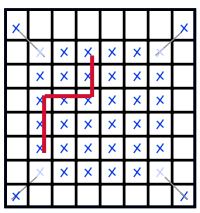
Strategy: Iterative Improvement Placement

- Placement results from many, small, random perturbations
  - Can minimize just about anything you can measure or estimate
  - But--must evaluate that estimation function many many times
  - Usually used to optimize total wirelength directly



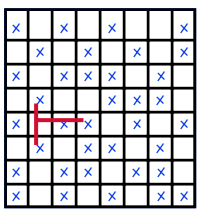
Initial Placement

- Gates randomly placed in legal locations
- Any optimization metric can be used



Intermediate Placement

- Gates move between legal locations
- Net length gradually minimized

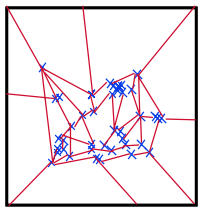


Final Placement

- Eventually all circuits settle in a location
- Exact pin positions known

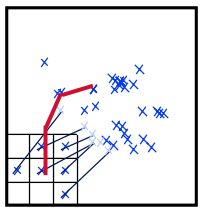
Strategy: Direct Placement

- All these use a technique called "Quadratic Placement"
  - Model all gates a points, all wires as 2-point "springs"
  - Minimize total squared Euclidean length: Σ_i EuclideanLength^2(net i)
  - Surprisingly, can do initial parts of this directly, numerically, *exactly*



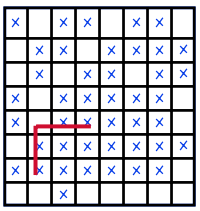
Initial Solution

- Direct soln. of quadratic total wirelength metric
- Gate and pin positions not yet legal



Legalization Phase

- Iterative snap-to-grid finds legal locations
- Any metric usable here



Final Placement

- Eventually all gates placed in legal locations
- Exact pin positions known

Placer Strategies Compared

	Layout Model		Optimization		Legalization	
Strategy	Gates	Nets	Wirelen	Congest	Middle	Final
Recursive min-cut	Have area but no shape	Multi-pt	Indirect	Direct	No, only clusters in middle	No, need final snap to grid
Iterative annealing	Both area and shape	Multi-pt	Direct	Doable, as histograms	Maybe (for our ex, yes)	Yes, legal at the end
Direct quadratic	0-dim points only	2-point nets only	Direct, quadratic	No	No, only points, not on row grid	No, need final snap to grid

Evolution of Strategies: Rough Timeline

- 1970s: Earliest techniques are iterative improvement
- 1970s: Recursive mincut emerges as viable
- 1980s: Iterative improvement gets much better with emergence of annealing; pure annealing, or mincut+annealing dominate
- 1990s: Quadratic direct emerges as viable and also much more scalable to very large designs, much faster than annealing at 100k+ gates
- 2000s: Early evidence of return to pure min-cut with emergence of some new partitioning algs, more concern about congestion, and speed at 5M+ gates

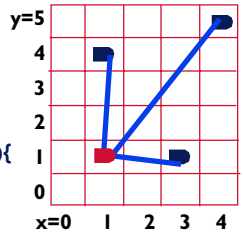
First Strategy: Iterative Improvement

- Where are we?
  - Assume you have a placement (each gate located in a cell on grid)
  - Assume use *half-perimeter* metric to compute Σ_{nets} (estimated wirelen)
  - Can now tell if this placement is good (Σ_{nets} = small) or bad (Σ_{nets} = big)
- Basic strategy
  - Basic idea: *iteratively improve via long sequence of small placement changes*
  - Start with a random placement
  - Perturb it (example: swap 2 gate's cell locations in grid)
  - Evaluate improvement = Δwirelength
- Questions
  - How do we know *what* to perturb, how *much*, *when* to quit, etc?

Earliest Iterative Improvement Approaches

- 1970s
  - "Optimal" perturbation schemes try to relocate gates to "best" new locations
  - Lots of variants

```
For(each gate g in some order){
      compute optimal spot
      move gate g
      if (spot occupied) remove
        existing gate, this is new g
}
```

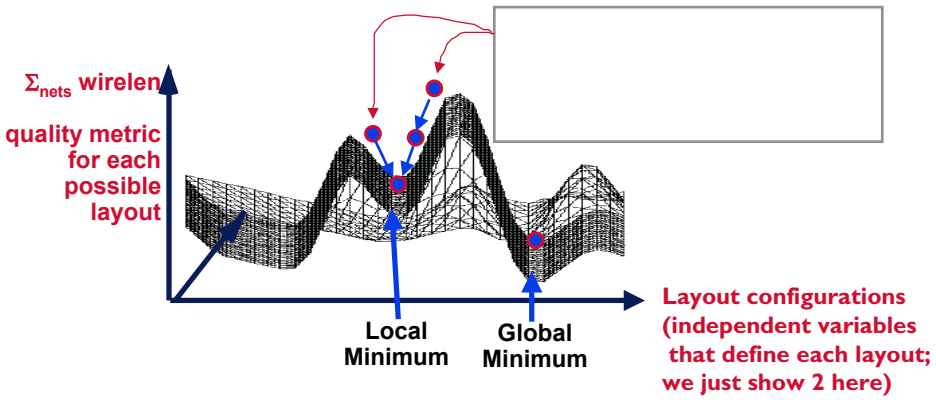


Example: treat wires as *force vectors*, decide where they "pull" center gate to "want" to settle

- How well did this work…?
  - OK (not great by modern measures)
  - Problem is these methods are inherently greedy: they quit when can't find another good perturbation

Iterative Improvement Approaches

- Problem with the 70s "optimal" strategies: *Greedy* algorithms
  - They only pick "good" perturbations that most *improve* wirelength...
  - ...and continue until they can't make any more progress
  - Problem: local minima in the cost surface for the placement task

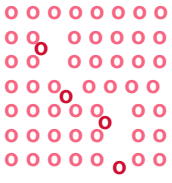


Solution Technique: Simulated Annealing

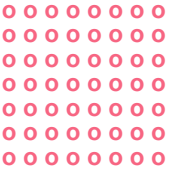
- Let's go waaay off to the side here and develop an idea
  - How far off to the side? Let's go look at some *statistical mechanics* from our friends in computational physics
  - Idea originally developed by Scott Kirkpatrick et al, physicist from IBM

- Suppose you want to make a *perfect crystal*
  - Perfect = all atoms lined up on crystal lattice sites; no defects
  - Perfect = this is the lowest energy "state" for this set of atoms

Imperfect order, has HIGHER energy

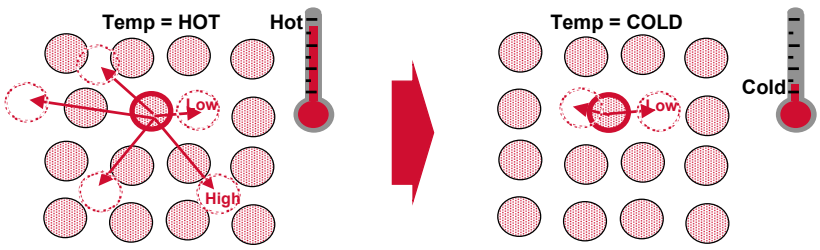


Perfect order, has MINIMUM energy



Real, Physical Annealing

- How do you do this physically?
  - You "anneal" the material
  - Get it very hot: gives atoms energy to move around
  - Cool it very slowly: gently restricts range of motion till everything freezes into (you hope) a low energy configuration



When it's hot...

When it's cold...

Annealing -> Simulated Annealing

- Now what?
  - That was a real physical system: real atoms, energy, heat, etc.
  - Think about attacking this problem *computationally*
  - How do you *compute* this low energy state, from first principles.
- Back up a bit...
  - Suppose the temperature is constant
  - How do you *simulate* what these atoms are doing as they hop around?

Annealing: Basics

- Phrase this question more exactly
  - *How do you compute the low-energy configurations of a physical system in thermal equilibrium (ie, at a constant temperature)?*
- Answer
  - *Metropolis* algorithm

Start with the system in a known configuration, at known energy E

Perturb system slightly (eg, move an atom to new location)

Compute ΔE , change in energy due to this perturbation

if ($\Delta E < 0$)

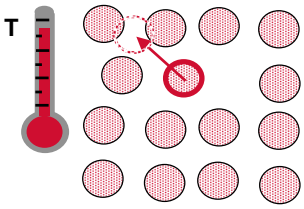
then

else

go back to start

Aside: Metropolis Criterion

- That *if-then* in algorithm is "the Metropolis criterion"
  - After you perturb an atom and compute Δenergy, it tells you if you keep this new perturbation as new configuration or throw it away
  - If the energy goes down, ΔE<0, this is a "better" state: keep it
  - If energy goes up, ΔE >0, this is a "worse state": *maybe keep it, depends on temperature*



Compute $e^{-\Delta E/KT}$
= a number in [0,1]

= Probability that you will accept this perturbation

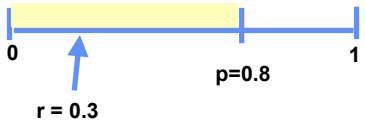
Generate r
a random number in [0, 1]

Compare r and $e^{-\Delta E/KT}$

if (r is smaller)
keep this perturbation
else reject it

Aside: Metropolis Criterion

- Example
  - Suppose ΔE > 0
  - Suppose $p = e^{-\Delta E / KT} = 0.8$
  - Suppose you generated r = uniform random number in [0,1] = 0.3
- What is really going on?



- What is the probability that $0 < r < 0.8$?

Simulated Annealing

Question

- Metropolis algorithm iteratively visits configurations with "reasonably probable" energies at the given fixed temperature
- What if I want to *find* a minimum energy state, now what do I do?

Answer

- *Simulated* annealing
- Add outer loop that starts with a high temperature, and slowly cools it
- Do enough perturbations at each temperature in the sequence of cooling steps to get to thermal equilibrium (ie, do the Metropolis procedure)
- Do enough temperatures so that the problem actually freezes into a low energy state, and further cooling does not further lower energy

Simulated Annealing

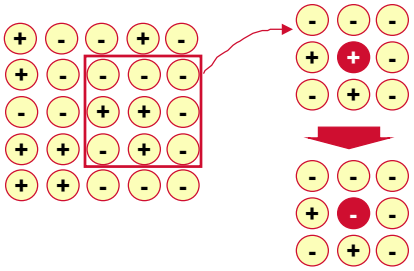
Start with the system in a known configuration, at known energy E

```
T = temperature = hot;    frozen = false;
while ( ! frozen ) {
    repeat {
        Perturb system slightly (eg, move a particle)
        Compute ΔE , change in energy due to perturbation
        if (ΔE < 0 )
        then   accept this perturbation,  this is the new system config
        else   accept maybe, with probability = e -ΔE/T
    } until (the system is in thermal equilibrium at this T)

    If (E still decreasing over the last few temperatures)
    then T = 0.9 T   // cool the temperature; do more perturbations
    else  frozen = true
}
return (final configuration as low-energy solution)
```

Toy Example

- Pretty easy to code a little example
  - Problem: 2D lattice of atoms, each in one of 2 states: + / -
  - Energy of the system:
    - Only in the bonds between neighbor atoms
    - Contribution is +1 if atom states different, else 0
  - To anneal:
    - Moves are just: pick an atom, flip the state, compute ΔE



Suppose we flip center atom

Old contribution to energy:

New contribution to energy:

ΔE is:

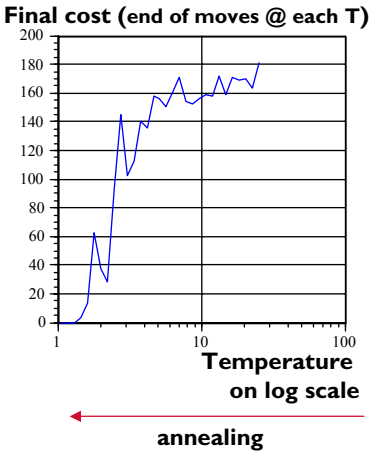
Annealing Pseudo-Code

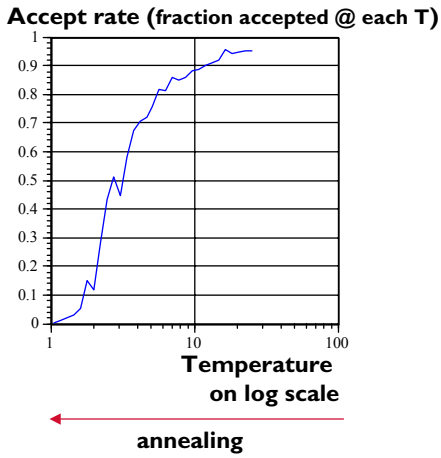
Pseudo-code

```
T = 100
Loop: for ( i = 1 to 10 * number of atoms ) {
      pick a random atom, flip it, compute ΔE
      accept = metropolis(ΔE, T)
}
if (total cost is still improving, ie, changed > 1% over last 3 temps)
      T = 0.9 * T
      goto Loop;
else quit
```

Toy Annealer: Results

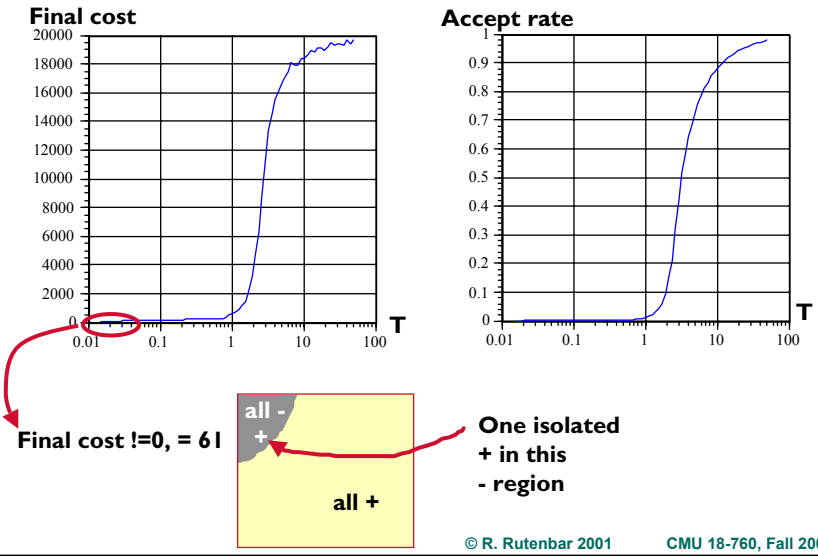
- 10x10 lattice, 1000 moves per temperature





Toy Annealer: Results

- 100x100 lattice, 250,000 moves per temperature



What Has This To Do With Placement?

- Combinatorial optimization problems are like these physical systems being coerced into low-E states

Physical System	Engineering Problem
System with atoms in various states	Optimization problem with many variables (x1, x2, x3, …, xn)
Energy	Cost metric (eg, wirelength)
ΔE perturbation	Iterative improvement step, Δcost perturb
Lowest energy "groundstate"	Optimum solution
Temperature	*Hill climbing control parameter*
Annealing	*Simulated Annealing*

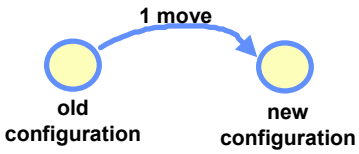
Annealing Algorithm: Essential Pieces

- What are the components of any annealing solution to a combinatorial problem?
  - There are 4 key pieces
  - We go over them here…

1. State representation
   - Exactly what are the configurations of solutions to your problem that you will visit as you iteratively perturb things?
2. Cost function
   - How will you measure how good each visited configuration is during iterative perturbations?
   - This plays the role of "energy" in simulated annealing

Annealing Algorithm: Essential Pieces

3. Move set
   - In annealing-speak, perturbations are always called "moves"
   - The *move set* is the set of "types" of perturbations that you with do to evolve from one solution configuration to the next



   - Examples:
     - Move that atom from (x,y,z) to (x',y',z')
     - Rotate that block in the floorplan for the chip
     - *Swap the position of those 2 gates in the placement*

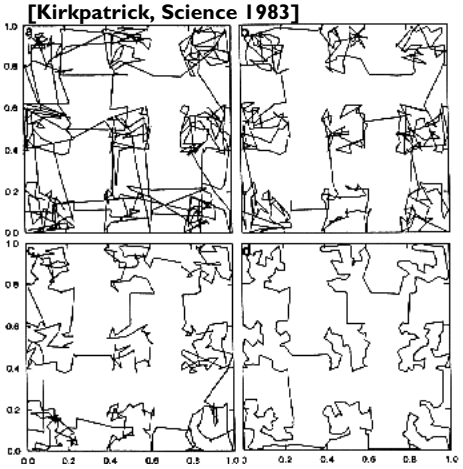
Annealing: Essential Pieces

4. Cooling Schedule
   - Starting temperature
     - How hot is hot enough at the start of annealing?
     - *Usually want it hot enough that any move you try is accepted*
     - *When it's hot, you basically randomize the solution*
   - Equilibrium criterion
     - How do you know you have done enough moves at the current temperature to stop, and exit to see if you should cool T?
     - *For now, just do a lot of moves at each temperature (~100*objects)*
   - Cooling rate
     - How fast to cool? Tnew = 0.9•Told ? Tnew = 0.8 • Told ?
     - *Slower cooling (0.9) gives better answers, but takes longer*
   - Frozen criterion
     - When is overall solution as good as it will get, so it's time to quit?
     - *Usually wait a few temps and see if cost stops changing much*

Simple Combinatorial Optimization Example

- Travelling Salesman Problem
  - Visit a set of cities in order, one visit per city, first city = last city
  - Minimize total length of travel
- To anneal
  - *State* = list of cities in order, called a *tour*
    - Ex: (Detroit, Paris, Lisbon, London, Detroit)
  - *Move* = swap 2 cities in tour
    - Ex: (Detroit, London, Lisbon, Paris, Detroit)
  - *Cost* = sum of lengths of travel, city to city, on tour
  - *Cooling* -- you know

[Kirkpatrick, Science 1983]

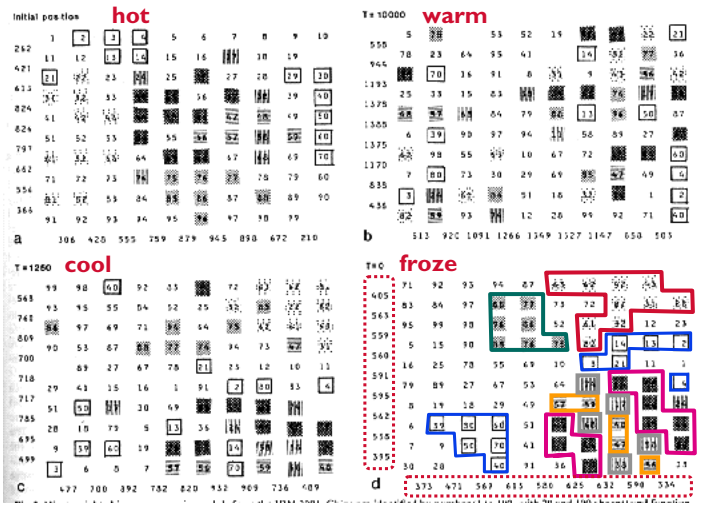


OK, How to Do ASIC Placement?

- Surprisingly easy to do a "toy" placer
  - State
    - Just the (x,y) location of each placeable object in our grid
  - Cost
    - Just total estimated half-perimeter wirelength over all nets
  - Moves
    - Easiest is pick 2 random gates and swap their locations on the grid
  - Cooling
    - T init = hot; Tnew = 0.9*Told; do a lot of moves at each temperature to ensure equilibrium (eg, 100*#gates moves/temp)
    - Quit when the cost curve versus temperature is *flat* enough
  - (Real placers are a *lot* more complicated, but this is surprisingly *OK*…)

Example: [Kirkpatrick, Science 1983]

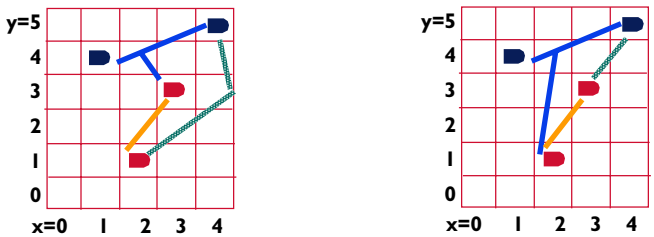
- Actually placing chips on a package, but same idea



Optimizations

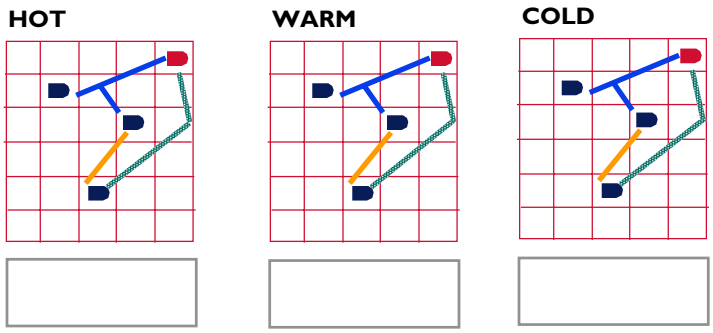
- Incremental cost calculation
  - You cannot afford to go recompute the cost of each net in the entire placement after you do one measly little swap
  - For one thing, it's *stupid*: most lengths didn't change!
  - You *have* do this incrementally--just look at the wires that *could* change

Δwirelen =



Optimizations

- Range limiting
  - You don't get any rewards for proposing moves that have a very high probability of being rejected -- rejected moves don't advance solution
  - Sometimes you can tell in advance which are more likely to succeed
  - Range = amount by which the cost is *likely* to change if you do this move
  - T = HOT, moves with large range are OK; T=COLD, not



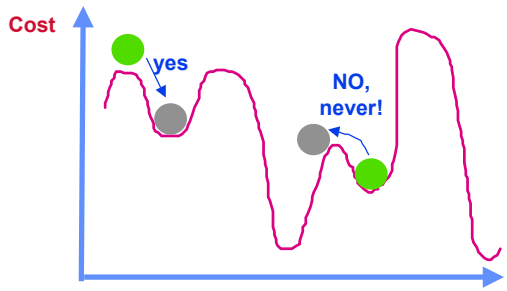
Why Does Annealing Work?

- Helpful mental model #1: *Balls & Hills*
  - Look at a simple representation of a combinatorial optimization task
  - Can model as a cost surface (also called a "landscape" or "space")
  - The configuration we are visiting now is the "ball" on the "hill"

All possible configurations of system that is being optimized.
Note we only draw for 1 variable

Balls & Hills

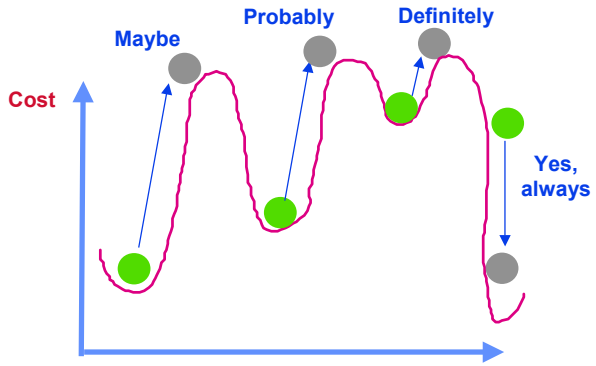
- Consider classical "greedy" iterative improvement
  - Only take moves that *improve* the cost
  - Physical analogy: like a *quench*, cool too fast and you get lousy crystal
  - Can get easily trapped in local minima



All possible configurations of system
that is being optimized.
Note we only draw for 1 variable

Balls & Hills

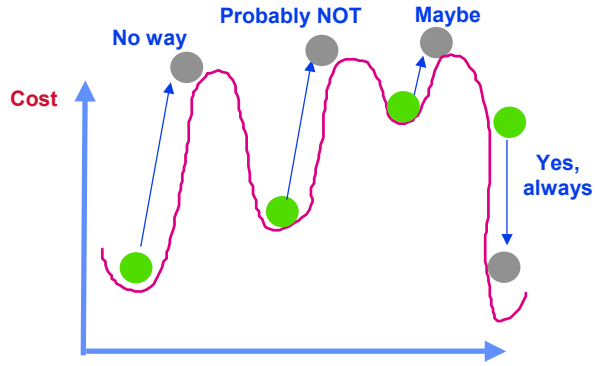
- Simulated annealing allows probabilistic hill climbing
  - Suppose temperature T = HOT, remember Pr[accept] = $e^{-\Delta C/T}$



All possible configurations of system
that is being optimized.
Note we only draw for 1 variable

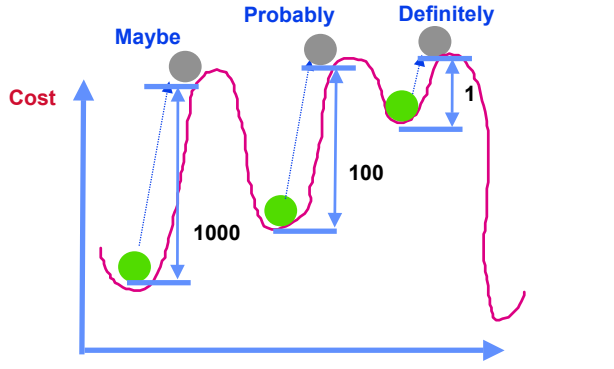
Balls & Hills

- Simulated annealing allows probabilistic hill climbing
  - Suppose temperature T = COLD, remember Pr[accept] = $e^{-\Delta C/T}$
  - As temperature cools, fewer uphill moves acceptable



All possible configurations of system
that is being optimized.
Note we only draw for 1 variable

Balls & Hills: Some Numbers

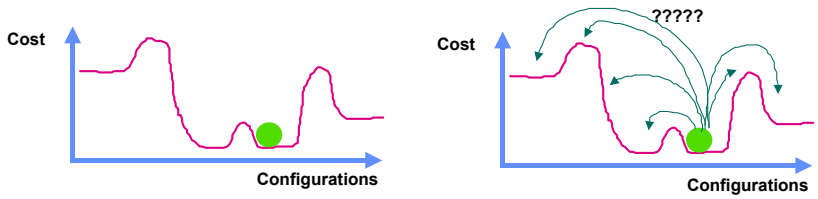


All possible configurations of system

Uphill ΔC	*Probability we will accept this move* Hot T=1000	Warm T=100	Cold T=1
1	0.999	0.99	0.37
100	0.900	0.37	~0
1000	0.37	0.00004	~0

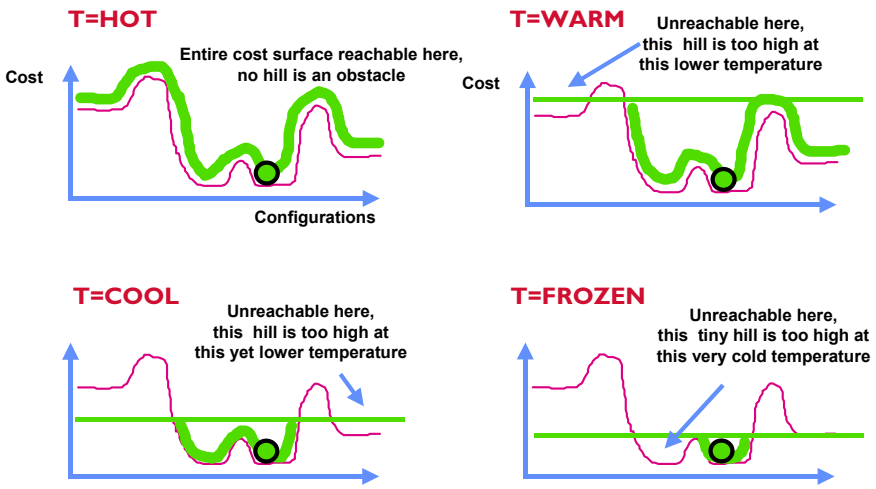
Helpful Model #2: Landscape Flattening

- Consider this bumpy cost surface (ball & hills)



- Question
  - *As a function of temperature, how much of this cost surface is reachable if we start from where the ball is in this figure?*
  - We think temp T "hides" the obstacles when hot; adaptively "smooths" or "flattens" these obstacles so we ignore them at the start
  - Cooling restricts us to ever smaller "good" areas; obstacles reappear
  - Idea sometimes referred to as "adaptive smoothing" of cost surface

Landscape Flattening

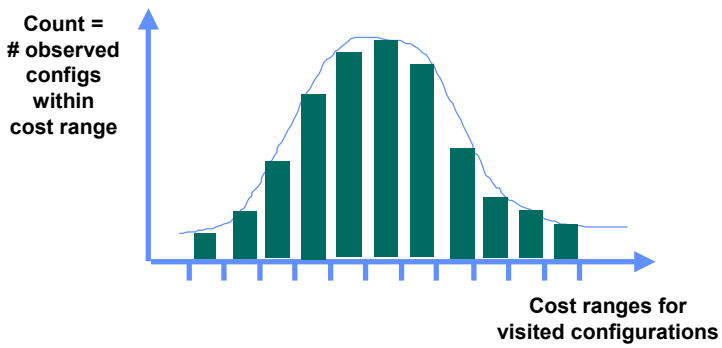


Annealing Dynamics

- Question
  - When my annealer is running, what do I actually see happening at each temperature, and across sequences of decreasing temperatures?
- Answer
  - At each temperature, you visit solution configurations in your "neighborhood" of the cost surface
  - Those solution configurations will *all have different costs*
  - You will see a "distribution" of costs at any fixed T
  - What does that distribution look like?

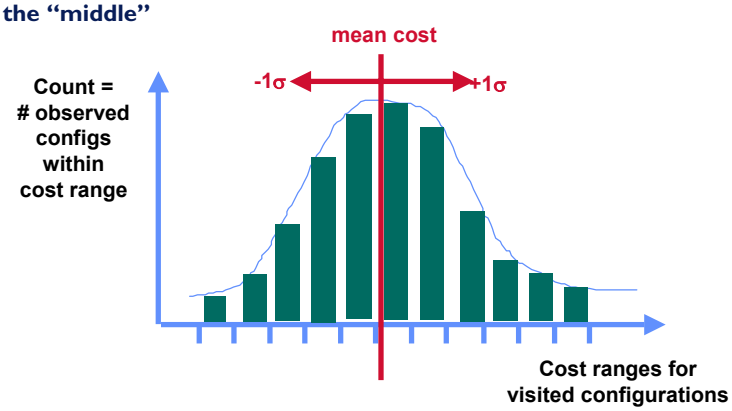
Annealing Dynamics

- Distribution of configurations at temperature
  - Can make a histogram, with ranges for cost of solutions seen
  - Vertical axis counts how many configurations visited that fall into each cost "bucket"
  - Get a bell-shaped distribution



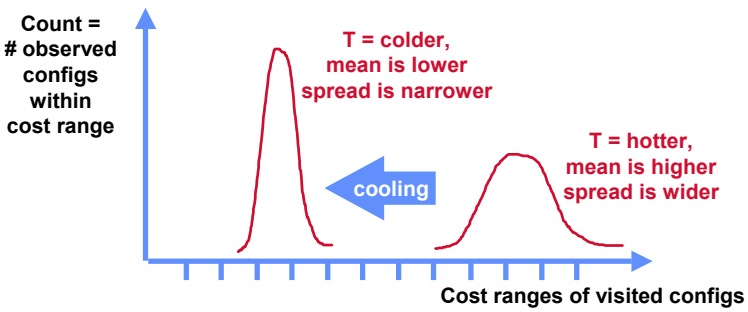
Annealing Dynamics

- Typically...
  - You visit some really good (low cost solutions), but temperature is high enough you keep jumping out
  - You visit some really lousy configurations (uphill) but keep falling back to the "middle"



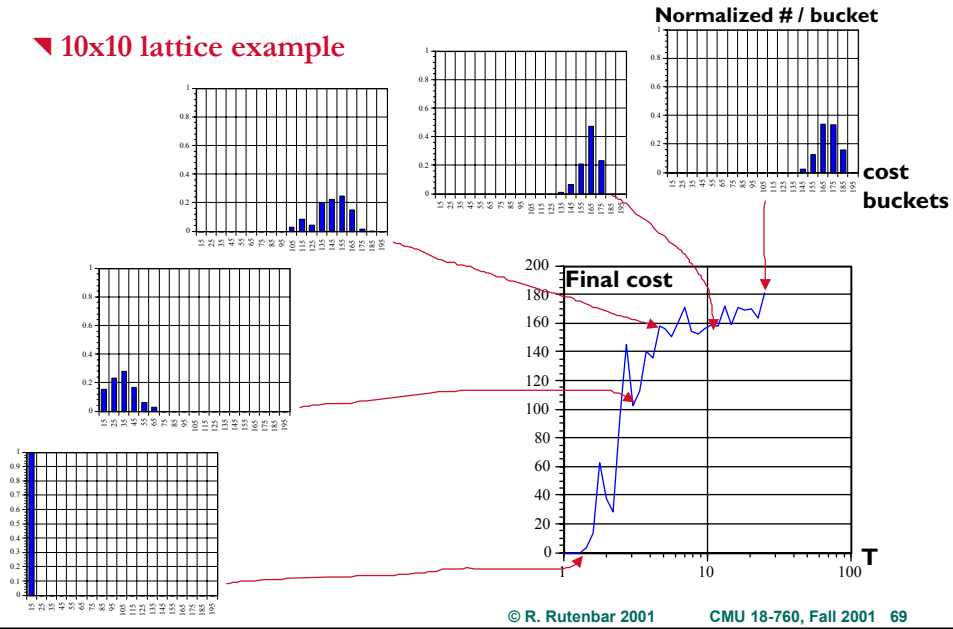
Annealing Dynamics

- What happens to distribution as cooling proceeds?
  - Histograms get narrower: unwilling to visit so many bad configs in the neighborhood, and there are fewer "better" configs around
  - Histograms get taller: more of the solutions you find are near the mean, temp is too low to jump uphill to really worse ones, and again there are fewer better ones around to fall down into



Toy Example: How Cost Distribution Evolves

- 10x10 lattice example



Some Annealing FAQs

Question	Answer
Does annealing always get global, optimum solution?	No. It just avoids a whole lot of suboptimal local solutions
How fast is annealing?	Usually regarded as "slow", tho depends a lot on implementation; must visit many solution configurations.
Are results deterministic, and repeatable?	No. If you run same random initial config 10 times (different random num sequences) you get 10 different answers
Can I affect this..?	Yes. Well-tuned annealers have tighter "spreads" on their solutions
Do I really have to guess all those cooling nums myself?	No. There are more complex adaptive algs that auto-tune cooling to problem
Does annealing work on other combinatorial problems?	Yes. Very well on lots of other probs.

Summary

- Annealing is
  - A way of constructing algorithms for combinatorial optimiz. problems
  - Iterative improvement with hill climbing
  - Composed of a few essential pieces
    - State representation, cost function, move set, cooling schedule
  - Good at not getting stuck in some local minima

- ASIC placement
  - 3 big strategies: recursive, direct, iterative improvement
  - 2 big optimization goals: estimated total wirelength, congestion
  - Annealing has been very successful in itetrative improvement placement with total wirelength minimization as the goal
  - Annealing runs out of gas around 100k-gates
  - Part II covers recursive & direct techniques (surprise: they are *related*)